Oracle® Database

Application Developer's Guide - Large Objects

10*g* Release 1 (10.1)

Part No. B10796-01

December 2003

ORACLE®

Oracle Database Application Developer's Guide - Large Objects, 10*g* Release 1 (10.1)

Part No. B10796-01

Primary Author: Eric Paapanen

Contributing Authors: K. Akiyama, Geeta Arora, S. Banerjee, Yujie Cao, Thomas 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

Contents

Send Us Your Comments xxiii

Preface xxv

Audience xxvi
Organization xxvi
Related Documents xxviii
Conventions xxx
Documentation Accessibility xxxiii

What's New in Large Objects? xxxv

LOB Features Introduced in Oracle Database 10*g* Release 1 xxxvi
Restrictions Removed in Oracle Database 10*g* Release 1 xxxviii
LOB Features Introduced in Oracle9*i* Release 2 xxxviii
Restrictions Removed in Oracle9*i* Release 2 xxxix
LOB Features Introduced in Oracle9*i* Release 1 xl
Restrictions Removed in Oracle9*i* Release 1 xlii

Part I Getting Started

1 Introduction to Large Objects

What Are Large Objects? 1-2
Why Use Large Objects? 1-2
Using LOBs for Semi-structured Data 1-3
Using LOBs for Unstructured Data 1-3

Why Not Use LONGs? 1-3
Different Kinds of LOBs 1-4
Internal LOBs 1-4
External LOBs and the BFILE Datatype 1-5
Introducing LOB Locators 1-6
Database Semantics for Internal and External LOBs 1-6
Large Object Datatypes 1-7
Abstract Datatypes and LOBs 1-7
Storing and Creating Other Datatypes with LOBs 1-8
VARRAYs Stored as LOBs 1-8
XMLType Columns Stored as CLOBs 1-8
LOBs Used in Oracle *inter*Media 1-8

2 Working with LOBs

LOB Column States 2-2
Locking a Row Containing a LOB 2-2
Opening and Closing LOBs 2-2
LOB Locator and LOB Value 2-3
Using the Data Interface for LOBs 2-3
Using the LOB Locator to Access and Modify LOB Values 2-3
LOB Locators and BFILE Locators 2-4
Initializing a LOB Column to Contain a Locator 2-4
Accessing LOBs 2-6
Accessing a LOB Using SQL 2-7
Accessing a LOB Using the Data Interface 2-7
Accessing a LOB Using the Locator Interface 2-7
LOB Restrictions 2-8
Restrictions on LOB Columns 2-8
Restrictions for LOB Operations 2-11

3 Managing LOBs: Database Administration

Database Utilities for Loading Data into LOBs 3-2
Using SQL*Loader to Load LOBs 3-2
Using SQL*Loader to Populate a BFILE Column 3-3
Using Oracle DataPump to Transfer LOB Data 3-6

Managing Temporary LOBs ... 3-6
Managing Temporary Tablespace for Temporary LOBs ... 3-6
Managing BFILEs ... 3-6
Rules for Using Directory Objects and BFILEs ... 3-6
Setting Maximum Number of Open BFILEs ... 3-7
Changing Tablespace Storage for a LOB ... 3-7

Part II Application Design

4 LOBs in Tables

Creating Tables That Contain LOBs ... 4-2
Initializing Persistent LOBs to NULL or Empty ... 4-2
Initializing LOBs ... 4-3
Initializing Persistent LOB Columns to a Value ... 4-3
Initializing BFILEs to NULL or a File Name ... 4-4
Restriction on First Extent of a LOB Segment ... 4-4
Choosing a LOB Column Datatype ... 4-4
LOBs Compared to LONG and LONG RAW Types ... 4-4
Storing Varying-Width Character Data in LOBs ... 4-5
Implicit Character Set Conversions with LOBs ... 4-6
Selecting a Table Architecture ... 4-6
LOB Storage ... 4-7
In-line and Out-of-Line LOB Storage ... 4-7
Defining Tablespace and Storage Characteristics for Persistent LOBs ... 4-8
LOB Storage Characteristics for LOB Column or Attribute ... 4-9
TABLESPACE and LOB Index ... 4-10
PCTVERSION ... 4-10
RETENTION ... 4-11
CACHE / NOCACHE / CACHE READS ... 4-12
LOGGING / NOLOGGING ... 4-13
CHUNK ... 4-14
ENABLE or DISABLE STORAGE IN ROW Clause ... 4-14
Guidelines for ENABLE or DISABLE STORAGE IN ROW ... 4-15
Indexing LOB Columns ... 4-15
Using Domain Indexing on LOB Columns ... 4-15

Indexing LOB Columns Using a Text Index 4-16
Function-Based Indexes on LOBs 4-16
Extensible Indexing on LOB Columns 4-16
Oracle Text Indexing Support for XML 4-18
Manipulating LOBs in Partitioned Tables 4-18
Partitioning a Table Containing LOB Columns 4-18
Creating an Index on a Table Containing Partitioned LOB Columns 4-19
Moving Partitions Containing LOBs 4-19
Splitting Partitions Containing LOBs 4-20
Merging Partitions Containing LOBs 4-20
LOBs in Index Organized Tables 4-20
Restrictions for LOBs in Partitioned Index-Organized Tables 4-22
Updating LOBs in Nested Tables 4-22

5 Advanced Design Considerations

LOB Buffering Subsystem 5-2
Advantages of LOB Buffering 5-2
Guidelines for Using LOB Buffering 5-2
LOB Buffering Subsystem Usage 5-4
Flushing the LOB Buffer 5-6
Flushing the Updated LOB 5-7
Using Buffer-Enabled Locators 5-8
Saving Locator State to Avoid a Reselect 5-8
OCI Example of LOB Buffering 5-9
Opening Persistent LOBs with the OPEN and CLOSE Interfaces 5-12
Index Performance Benefits of Explicitly Opening a LOB 5-12
Working with Explicitly Open LOB Instances 5-13
Read Consistent Locators 5-13
A Selected Locator Becomes a Read Consistent Locator 5-14
Updating LOBs and Read-Consistency 5-14
Updating LOBs Through Updated Locators 5-16
Example of Updating a LOB Using SQL DML and DBMS_LOB 5-17
Example of Using One Locator to Update the Same LOB Value 5-19
Example of Updating a LOB with a PL/SQL (DBMS_LOB) Bind Variable 5-22
LOB Locators and Transaction Boundaries 5-24

Reading and Writing to a LOB Using Locators 5-25
Selecting the Locator Outside of the Transaction Boundary 5-25
Selecting the Locator Within a Transaction Boundary 5-26
LOB Locators Cannot Span Transactions 5-27
Example of Locator Not Spanning a Transaction 5-28
LOBs in the Object Cache 5-29
Terabyte-Size LOB Support 5-30
Maximum Storage Limit for Terabyte-Size LOBs 5-31
Using Terabyte-Size LOBs with JDBC 5-31
Using Terabyte-Size LOBs with the DBMS_LOB Package 5-31
Using Terabyte-Size LOBs with OCI 5-31
Interfaces Not Supporting LOBs Greater Than 4 Gigabytes 5-32
Guidelines for Creating Gigabyte LOBs 5-32
Creating a Tablespace and Table to Store Gigabyte LOBs 5-33

6 Overview of Supplied LOB APIs

Programmatic Environments That Support LOBs 6-2
Comparing the LOB Interfaces 6-3
Using PL/SQL (DBMS_LOB Package) to Work with LOBs 6-7
Provide a LOB Locator Before Running the DBMS_LOB Routine 6-7
Guidelines for Offset and Amount Parameters in DBMS_LOB Operations 6-8
PL/SQL Functions and Procedures for LOBs 6-9
PL/SQL Functions/Procedures to Modify LOB Values 6-9
PL/SQL Functions and Procedures for Introspection of LOBs 6-10
PL/SQL Operations on Temporary LOBs 6-10
PL/SQL Read-Only Functions/Procedures for BFILEs 6-11
PL/SQL Functions/Procedures to Open and Close Internal and External LOBs 6-11
Using OCI to Work with LOBs 6-11
Setting the CSID Parameter for OCI LOB APIs 6-11
Fixed-Width and Varying-Width Character Set Rules for OCI 6-12
OCILobLoadFromFile2() Amount Parameter 6-13
OCILobRead2() Amount Parameter 6-13
OCILobLocator Pointer Assignment 6-13
LOB Locators in Defines and Out-Bind Variables in OCI 6-14
OCI LOB Examples 6-14

Further Information About OCI 6-14
OCI Functions That Operate on BLOBs, CLOBs, NCLOBs, and BFILEs 6-14
OCI Functions to Modify Persistent LOB (BLOB, CLOB, and NCLOB) Values 6-15
OCI Functions to Read or Examine Persistent LOB and External LOB (BFILE) Values 6-15
OCI Functions for Temporary LOBs 6-16
OCI Read-Only Functions for BFILEs 6-16
OCI LOB Locator Functions 6-16
OCI LOB-Buffering Functions 6-17
OCI Functions to Open and Close Internal and External LOBs 6-17
Using C++ (OCCI) to Work with LOBs 6-17
OCCI Classes for LOBs 6-18
Fixed Width Character Set Rules 6-19
Varying-Width Character Set Rules 6-20
Offset and Amount Parameters for Other OCCI Operations 6-21
Amount Parameter for OCCI LOB copy() Methods 6-21
Amount Parameter for OCCI read() Operations 6-21
Further Information About OCCI 6-22
OCCI Methods That Operate on BLOBs, BLOBs, NCLOBs, and BFILEs 6-22
OCCI Methods to Modify Persistent LOB (BLOB, CLOB, and NCLOB) Values 6-22
OCCI Methods to Read or Examine Persistent LOB and BFILE Values 6-23
OCCI Read-Only Methods for BFILEs 6-23
Other OCCI LOB Methods 6-23
OCCI Methods to Open and Close Internal and External LOBs 6-24
Using C/C++ (Pro*C) to Work with LOBs 6-24
First Provide an Allocated Input Locator Pointer That Represents LOB 6-24
Pro*C/C++ Statements That Operate on BLOBs, CLOBs, NCLOBs, and BFILEs 6-25
Pro*C/C++ Embedded SQL Statements to Modify Persistent LOB Values 6-25
Pro*C/C++ Embedded SQL Statements for Introspection of LOBs 6-26
Pro*C/C++ Embedded SQL Statements for Temporary LOBs 6-26
Pro*C/C++ Embedded SQL Statements for BFILEs 6-26
Pro*C/C++ Embedded SQL Statements for LOB Locators 6-26
Pro*C/C++ Embedded SQL Statements for LOB Buffering 6-27
Pro*C/C++ Embedded SQL Statements to Open and Close LOBs 6-27
Using COBOL (Pro*COBOL) to Work with LOBs 6-27
First Provide an Allocated Input Locator Pointer That Represents LOB 6-27

Pro*COBOL Statements That Operate on BLOBs, CLOBs, NCLOBs, and BFILEs 6-28
Pro*COBOL Embedded SQL Statements to Modify Persistent LOB Values 6-29
Pro*COBOL Embedded SQL Statements for Introspection of LOBs 6-29
Pro*COBOL Embedded SQL Statements for Temporary LOBs 6-29
Pro*COBOL Embedded SQL Statements for BFILEs 6-30
Pro*COBOL Embedded SQL Statements for LOB Locators... 6-30
Pro*COBOL Embedded SQL Statements for LOB Buffering 6-30
Pro*COBOL Embedded SQL Statements for Opening and Closing LOBs and BFILEs... 6-30
Using Visual Basic (Oracle Objects for OLE (OO4O)) to Work with LOBs 6-31
OO4O Syntax Reference 6-31
OraBlob, OraClob, and OraBfile Object Interfaces Encapsulate Locators......................... 6-32
Example of OraBlob and OraBfile 6-32
OO4O Methods and Properties to Access Data Stored in LOBs 6-33
OO4O Methods to Modify BLOB, CLOB, and NCLOB Values 6-35
OO4O Methods to Read or Examine Internal and External LOB Values.......................... 6-35
OO4O Methods to Open and Close External LOBs (BFILEs) 6-36
OO4O Methods for Persistent LOB-Buffering 6-36
OO4O Properties for Operating on LOBs 6-36
OO4O Read-Only Methods for External Lobs (BFILEs) 6-37
OO4O Properties for Operating on External LOBs (BFILEs) 6-37
Using Java (JDBC) to Work with LOBs 6-37
Changing Internal Persistent LOBs Using Java.. 6-37
Reading Internal Persistent LOBs and External LOBs (BFILEs) with Java 6-38
Calling DBMS_LOB Package from Java (JDBC).. 6-38
Referencing LOBs Using Java (JDBC)... 6-38
JDBC Syntax References and Further Information 6-39
JDBC Methods for Operating on LOBs 6-39
JDBC oracle.sql.BLOB Methods to Modify BLOB Values 6-40
JDBC oracle.sql.BLOB Methods to Read or Examine BLOB Values 6-40
JDBC oracle.sql.BLOB Methods and Properties for BLOB-Buffering 6-41
JDBC oracle.sql.CLOB Methods to Modify CLOB Values... 6-41
JDBC oracle.sql.CLOB Methods to Read or Examine CLOB Value 6-41
JDBC oracle.sql.CLOB Methods and Properties for CLOB-Buffering 6-42
JDBC oracle.sql.BFILE Methods to Read or Examine External LOB (BFILE) Values....... 6-42
JDBC oracle.sql.BFILE Methods and Properties for BFILE-Buffering 6-43

JDBC Temporary LOB APIs 6-43
JDBC: Opening and Closing LOBs 6-44
JDBC: Opening and Closing BLOBs.... 6-44
JDBC: Opening and Closing CLOBs 6-46
JDBC: Opening and Closing BFILEs 6-47
Trimming LOBs Using JDBC 6-51
JDBC BLOB Streaming APIs 6-52
JDBC CLOB Streaming APIs.... 6-53
New BFILE Streaming APIs 6-55
JDBC and Empty LOBs 6-60
Oracle Provider for OLE DB (OraOLEDB) 6-60
Overview of Oracle Data Provider for .NET (ODP.NET).... 6-61

7 Performance Guidelines

LOB Performance Guidelines 7-2
Performance Guidelines for Small Size LOBs.... 7-2
General Performance Guidelines.... 7-2
Temporary LOB Performance Guidelines.... 7-3
Performance Considerations for SQL Semantics and LOBs.... 7-6
Moving Data to LOBs in a Threaded Environment 7-6

Part III SQL Access to LOBs

8 DDL and DML Statements with LOBs

Creating a Table Containing One or More LOB Columns.... 8-2
Creating a Nested Table Containing a LOB.... 8-5
Inserting a Row by Selecting a LOB From Another Table.... 8-6
Inserting a LOB Value Into a Table 8-7
Inserting a Row by Initializing a LOB Locator Bind Variable.... 8-8
PL/SQL: Inserting a Row by Initializing a LOB Locator Bind Variable.... 8-9
C (OCI): Inserting a Row by Initializing a LOB Locator Bind Variable.... 8-10
COBOL (Pro*COBOL): Inserting a Row by Initializing a LOB Locator Bind Variable 8-11
C/C++ (Pro*C/C++): Inserting a Row by Initializing a LOB Locator Bind Variable 8-12
Visual Basic (OO4O): Inserting a Row by Initializing a LOB Locator Bind Variable 8-13

Java (JDBC): Inserting a Row by Initializing a LOB Locator Bind Variable 8-14
Updating a LOB with EMPTY_CLOB() or EMPTY_BLOB() 8-15
Updating a Row by Selecting a LOB From Another Table 8-17

9 SQL Semantics and LOBs

Using LOBs in SQL 9-2
SQL Functions and Operators Supported for Use with LOBs 9-2
UNICODE Support 9-7
Codepoint Semantics 9-7
Return Values for SQL Semantics on LOBs 9-8
LENGTH Return Value for LOBs 9-8
Implicit Conversion of LOB Datatypes in SQL 9-8
Implicit Conversion Between CLOB and NCLOB Datatypes in SQL 9-8
Unsupported Use of LOBs in SQL 9-11
VARCHAR2 and RAW Semantics for LOBs 9-11
LOBs Returned from SQL Functions 9-12
IS NULL and IS [NOT] NULL Usage with VARCHAR2s and CLOBs 9-13
WHERE Clause Usage with LOBs 9-13

10 PL/SQL Semantics for LOBs

PL/SQL Statements and Variables 10-2
Implicit Conversions Between CLOB and VARCHAR2 10-2
Explicit Conversion Functions 10-3
VARCHAR2 and CLOB in PL/SQL Built-In Functions 10-3
PL/SQL CLOB Comparison Rules 10-6
CLOBs Follow the VARCHAR2 Collating Sequence 10-6

11 Migrating Table Columns from LONGs to LOBs

Benefits of Migrating LONG Columns to LOB Columns 11-2
Preconditions for Migrating LONG Columns to LOB Columns 11-2
Dropping a Domain Index on a LONG Column Before Converting to a LOB 11-3
Preventing Generation of Redo Space on Tables Converted to LOB Datatypes 11-3
Using utldtree.sql to Determine Where Your Application Needs Change 11-3
Converting Tables from LONG to LOB Datatypes 11-4

Using ALTER TABLE to Convert LONG Columns to LOB Columns ... 11-4
Copying a LONG to a LOB Column Using the TO_LOB Operator ... 11-5
Online Redefinition of Tables with LONG Columns ... 11-6
Migrating Applications from LONGs to LOBs ... 11-10
LOB Columns Not Allowed in Clustered Tables ... 11-10
LOB Columns Not Allowed in UPDATE OF Triggers ... 11-11
Indexes on Columns Converted from LONG to LOB Datatypes ... 11-11
Empty LOBs Compared to NULL and Zero Length LONGs ... 11-11
Overloading with Anchored Types ... 11-12
Some Implicit Conversions Are Not Supported for LOB Datatypes ... 11-13

Part IV Using LOB APIs

12 Operations Specific to Persistent and Temporary LOBs

Persistent LOB Operations ... 12-2
Inserting a LOB into a Table ... 12-2
Selecting a LOB from a Table ... 12-2
Temporary LOB Operations ... 12-2
Creating and Freeing a Temporary LOB ... 12-3
Creating Persistent and Temporary LOBs in PL/SQL ... 12-4

13 Data Interface for Persistent LOBs

Overview of the Data Interface for Persistent LOBs ... 13-2
Benefits of Using the Data Interface for Persistent LOBs ... 13-3
Using the Data Interface for Persistent LOBs in PL/SQL ... 13-3
Guidelines for Accessing LOB Columns Using the Data Interface in SQL and PL/SQL 13-4
Implicit Assignment and Parameter Passing ... 13-5
Passing CLOBs to SQL and PL/SQL Built-In Functions ... 13-6
Explicit Conversion Functions ... 13-6
Calling PL/SQL and C Procedures from SQL ... 13-7
Calling PL/SQL and C Procedures from PL/SQL ... 13-7
Binds of All Sizes in INSERT and UPDATE Operations ... 13-8
4,000 Byte Limit on Results of SQL Operator ... 13-8
Restrictions on Binds of More Than 4,000 Bytes ... 13-8

Example: PL/SQL - Using Binds of More Than 4,000 Bytes in INSERT and UPDATE... 13-9
4,000 Byte Result Limit for SQL Operators ... 13-10
Using the Data Interface for LOBs with INSERT, UPDATE, and SELECT Operations. 13-10
Using the Data Interface for LOBs in Assignments and Parameter Passing ... 13-11
Using the Data Interface for LOBs with PL/SQL Built-In Functions ... 13-12
Using the Data Interface for Persistent LOBs in OCI ... 13-13
Binding LOB Datatypes in OCI ... 13-13
Defining LOB Datatypes in OCI ... 13-13
Using Multibyte Charactersets in OCI with the Data Interface for LOBs ... 13-14
Using OCI Functions to Perform INSERT or UPDATE on LOB Columns ... 13-14
Using the Data Interface to Fetch LOB Data in OCI ... 13-15
PL/SQL and C Binds from OCI ... 13-17
Example: C (OCI) - Binds of More than 4,000 Bytes for INSERT and UPDATE ... 13-18
Using the Data Interface for LOBs in PL/SQL Binds from OCI on LOBs ... 13-21
Binding LONG Data for LOB Columns in Binds Greater Than 4,000 Bytes ... 13-22
Binding LONG Data to LOB Columns Using Piecewise INSERT with Polling ... 13-22
Binding LONG Data to LOB Columns Using Piecewise INSERT with Callback ... 13-24
Binding LONG Data to LOB Columns Using an Array INSERT ... 13-26
Selecting a LOB Column into a LONG Buffer Using a Simple Fetch ... 13-27
Selecting a LOB Column into a LONG Buffer Using Piecewise Fetch with Polling ... 13-28
Selecting a LOB Column into a LONG Buffer Using Piecewise Fetch with Callback ... 13-29
Selecting a LOB Column into a LONG Buffer Using an Array Fetch ... 13-31

14 LOB APIs for Basic Operations

Supported Environments ... 14-2
Appending One LOB to Another ... 14-4
PL/SQL DBMS_LOB Package: Appending One LOB to Another ... 14-5
C (OCI): Appending One LOB to Another ... 14-6
COBOL (Pro*COBOL): Appending One LOB to Another ... 14-7
C/C++ (Pro*C/C++): Appending One LOB to Another ... 14-8
Visual Basic (OO4O): Appending One LOB to Another ... 14-9
Java (JDBC): Appending One LOB to Another ... 14-10
Determining Character Set Form ... 14-13
C (OCI): Determining Character Set Form ... 14-13
Determining Character Set ID ... 14-15

C (OCI): Determining Character Set ID 14-15
Loading a LOB with Data from a BFILE 14-17
PL/SQL (DBMS_LOB): Loading a LOB with Data from a BFILE 14-19
C (OCI): Loading a LOB with Data from a BFILE 14-20
COBOL (Pro*COBOL): Loading a LOB with Data from a BFILE 14-21
Visual Basic (OO4O): Loading a LOB with Data from a BFILE 14-22
Java (JDBC): Loading a LOB with Data from a BFILE 14-23
Loading a BLOB with Data from a BFILE 14-26
PL/SQL: Loading a BLOB with BFILE Data 14-27
Loading a CLOB or NCLOB with Data from a BFILE 14-29
PL/SQL: Loading Character Data from a BFILE into a LOB 14-30
PL/SQL: Loading Segments of Character Data into Different LOBs 14-31
Determining Whether a LOB is Open 14-34
PL/SQL (DBMS_LOB Package): Checking If a LOB Is Open 14-35
C (OCI): Checking If a LOB Is Open 14-35
COBOL (Pro*COBOL): Checking If a LOB Is Open 14-36
C/C++ (Pro*C/C++): Checking If a LOB Is Open 14-38
Java (JDBC): Checking If a LOB Is Open 14-39
Displaying LOB Data 14-42
PL/SQL (DBMS_LOB Package): Displaying LOB Data 14-43
C (OCI): Displaying LOB Data 14-44
COBOL (Pro*COBOL): Displaying LOB Data 14-46
C/C++ (Pro*C/C++): Displaying LOB Data 14-47
Visual Basic (OO4O): Displaying LOB Data 14-49
Java (JDBC): Displaying LOB Data 14-49
Reading Data from a LOB 14-52
PL/SQL (DBMS_LOB Package): Reading Data from a LOB 14-54
C (OCI): Reading Data from a LOB 14-54
COBOL (Pro*COBOL): Reading Data from a LOB 14-58
C/C++ (Pro*C/C++): Reading Data from a LOB 14-59
Visual Basic (OO4O): Reading Data from a LOB 14-60
Java (JDBC): Reading Data from a LOB 14-61
Reading a Portion of a LOB (SUBSTR) 14-63
PL/SQL (DBMS_LOB Package): Reading a Portion of the LOB (substr) 14-64
COBOL (Pro*COBOL): Reading a Portion of the LOB (substr) 14-64

C/C++ (Pro*C/C++): Reading a Portion of the LOB (substr) 14-66
Visual Basic (OO4O): Reading a Portion of the LOB (substr) 14-67
Java (JDBC): Reading a Portion of the LOB (substr) 14-68
Comparing All or Part of Two LOBs 14-71
PL/SQL (DBMS_LOB Package): Comparing All or Part of Two LOBs 14-72
COBOL (Pro*COBOL): Comparing All or Part of Two LOBs 14-72
C/C++ (Pro*C/C++): Comparing All or Part of Two LOBs 14-74
Visual Basic (OO4O): Comparing All or Part of Two LOBs 14-76
Java (JDBC): Comparing All or Part of Two LOBs 14-76
Patterns: Checking for Patterns in a LOB Using INSTR 14-79
PL/SQL (DBMS_LOB Package): Checking for Patterns in a LOB (instr) 14-80
COBOL (Pro*COBOL): Checking for Patterns in a LOB (instr) 14-80
C/C++ (Pro*C/C++): Checking for Patterns in a LOB (instr) 14-82
Java (JDBC): Checking for Patterns in a LOB (instr) 14-83
Length: Determining the Length of a LOB 14-86
PL/SQL (DBMS_LOB Package): Determining the Length of a LOB 14-87
C (OCI): Determining the Length of a LOB 14-87
COBOL (Pro*COBOL): Determining the Length of a LOB 14-88
C/C++ (Pro*C/C++): Determining the Length of a LOB 14-89
Visual Basic (OO4O): Determining the Length of a LOB 14-90
Java (JDBC): Determining the Length of a LOB 14-91
Copying All or Part of One LOB to Another LOB 14-93
PL/SQL (DBMS_LOB Package): Copying All or Part of One LOB to Another LOB 14-94
C (OCI): Copying All or Part of One LOB to Another LOB 14-95
COBOL (Pro*COBOL): Copying All or Part of One LOB to Another LOB 14-96
C/C++ (Pro*C/C++): Copying All or Part of a LOB to Another LOB 14-98
Visual Basic (OO4O): Copying All or Part of One LOB to Another LOB 14-100
Java (JDBC): Copying All or Part of One LOB to Another LOB 14-100
Copying a LOB Locator 14-103
PL/SQL (DBMS_LOB Package): Copying a LOB Locator 14-104
C (OCI): Copying a LOB Locator 14-104
COBOL (Pro*COBOL): Copying a LOB Locator 14-105
C/C++ (Pro*C/C++): Copying a LOB Locator 14-107
Visual Basic (OO4O): Copying a LOB Locator 14-108
Java (JDBC): Copying a LOB Locator 14-109

Equality: Checking If One LOB Locator Is Equal to Another 14-111
C (OCI): Checking If One LOB Locator Is Equal to Another... 14-112
C/C++ (Pro*C/C++): Checking If One LOB Locator Is Equal to Another 14-113
Java (JDBC): Checking If One LOB Locator Is Equal to Another 14-115
Determining Whether LOB Locator Is Initialized.. 14-117
C (OCI): Determining Whether a LOB Locator Is Initialized 14-118
C/C++ (Pro*C/C++): Determining Whether a LOB Locator Is Initialized.................. 14-119
Appending to a LOB... 14-120
PL/SQL (DBMS_LOB Package): Writing to the End of (Appending to) a LOB........... 14-122
C (OCI): Writing to the End of (Appending to) a LOB 14-123
COBOL (Pro*COBOL): Writing to the End of (Appending to) a LOB.......................... 14-124
C/C++ (Pro*C/C++): Writing to the End of (Appending to) a LOB 14-125
Java (JDBC): Writing to the End of (Append-Write to) a LOB....................................... 14-126
Writing Data to a LOB 14-128
PL/SQL (DBMS_LOB Package): Writing Data to a LOB 14-131
C (OCI): Writing Data to a LOB... 14-132
COBOL (Pro*COBOL): Writing Data to a LOB 14-135
C/C++ (Pro*C/C++): Writing Data to a LOB 14-137
Visual Basic (OO4O):Writing Data to a LOB 14-140
Java (JDBC): Writing Data to a LOB... 14-141
Trimming LOB Data 14-143
PL/SQL (DBMS_LOB Package): Trimming LOB Data 14-144
C (OCI): Trimming LOB Data 14-145
COBOL (Pro*COBOL): Trimming LOB Data 14-146
C/C++ (Pro*C/C++): Trimming LOB Data... 14-147
Visual Basic (OO4O): Trimming LOB Data 14-149
Java (JDBC): Trimming LOB Data 14-149
Erasing Part of a LOB 14-154
PL/SQL (DBMS_LOB Package): Erasing Part of a LOB 14-155
C (OCI): Erasing Part of a LOB 14-156
COBOL (Pro*COBOL): Erasing Part of a LOB.. 14-156
C/C++ (Pro*C/C++): Erasing Part of a LOB.. 14-158
Visual Basic (OO4O): Erasing Part of a LOB 14-159
Java (JDBC): Erasing Part of a LOB 14-159
Enabling LOB Buffering... 14-162

COBOL (Pro*COBOL): Enabling LOB Buffering 14-163
C/C++ (Pro*C/C++): Enabling LOB Buffering 14-165
Visual Basic (OO4O): Enabling LOB Buffering 14-166
Flushing the Buffer 14-167
COBOL (Pro*COBOL): Flushing the Buffer 14-168
C/C++ (Pro*C/C++): Flushing the Buffer 14-170
Disabling LOB Buffering 14-171
C (OCI): Disabling LOB Buffering 14-173
COBOL (Pro*COBOL): Disabling LOB Buffering 14-174
C/C++ (Pro*C/C++): Disabling LOB Buffering 14-176
Visual Basic (OO4O): Disabling LOB Buffering 14-177
Determining Whether a LOB instance Is Temporary 14-178
PL/SQL (DBMS_LOB Package): Determining Whether a LOB Is Temporary 14-179
C (OCI): Determining Whether a LOB Is Temporary 14-179
COBOL (Pro*COBOL): Determining Whether a LOB Is Temporary 14-180
C/C++ (Pro*C/C++): Determining Whether a LOB Is Temporary 14-182
Java (JDBC): Determining Whether a BLOB Is Temporary 14-183
Java (JDBC): Determining Whether a CLOB Is Temporary 14-184
Converting a BLOB to a CLOB 14-185
Converting a CLOB to a BLOB 14-185

15 LOB APIs for BFILE Operations

Supported Environments for BFILE APIs 15-2
Accessing BFILEs 15-4
Directory Object 15-4
Initializing a BFILE Locator 15-4
How to Associate Operating System Files with Database Records 15-5
BFILENAME() and Initialization 15-6
Characteristics of the BFILE Datatype 15-6
DIRECTORY Name Specification 15-7
BFILE Security 15-7
Ownership and Privileges 15-8
Read Permission on a DIRECTORY Object 15-8
SQL DDL for BFILE Security 15-9
SQL DML for BFILE Security 15-9

Catalog Views on Directories 15-9
Guidelines for DIRECTORY Usage 15-10
BFILEs in Shared Server (Multithreaded Server) Mode 15-11
External LOB (BFILE) Locators 15-11
Loading a LOB with BFILE Data 15-13
PL/SQL (DBMS_LOB): Loading a LOB with BFILE Data 15-15
C (OCI): Loading a LOB with BFILE Data 15-16
COBOL (Pro*COBOL): Loading a LOB with BFILE Data 15-16
C/C++ (Pro*C/C++): Loading a LOB with BFILE Data 15-18
Visual Basic (OO4O): Loading a LOB with BFILE Data 15-20
Opening a BFILE with OPEN 15-21
PL/SQL (DBMS_LOB): Opening a BFILE with OPEN 15-22
C (OCI): Opening a BFILE with OPEN 15-22
COBOL (Pro*COBOL): Opening a BFILE with OPEN 15-23
C/C++ (Pro*C/C++): Opening a BFILE with OPEN 15-24
Visual Basic (OO4O) Opening a BFILE with OPEN 15-25
Java (JDBC): Opening a BFILE with OPEN 15-26
Opening a BFILE with FILEOPEN 15-28
PL/SQL (DBMS_LOB): Opening a BFILE with FILEOPEN 15-29
C (OCI): Opening a BFILE with FILEOPEN 15-29
Java (JDBC): Opening a BFILE with FILEOPEN 15-30
Determining Whether a BFILE Is Open Using ISOPEN 15-32
PL/SQL (DBMS_LOB): Determining Whether a BFILE Is Open with ISOPEN 15-33
C (OCI): Determining Whether a BFILE Is Open with ISOPEN 15-34
COBOL (Pro*COBOL): Determining Whether a BFILE Is Open with ISOPEN 15-34
C/C++ (Pro*C/C++): Determining Whether a BFILE Is Open with ISOPEN 15-36
Visual Basic (OO4O): Determining Whether a BFILE Is Open with ISOPEN 15-37
Java (JDBC): Determining Whether a BFILE Is Open with ISOPEN 15-38
Determining Whether a BFILE Is Open with FILEISOPEN 15-41
PL/SQL (DBMS_LOB): Determining Whether a BFILE Is Open with FILEISOPEN 15-42
C (OCI): Determining Whether a BFILE Is Open with FILEISOPEN 15-43
Java (JDBC): Determining Whether a BFILE Is Open with FILEISOPEN 15-43
Displaying BFILE Data 15-46
PL/SQL (DBMS_LOB): Displaying BFILE Data 15-47
C (OCI): Displaying BFILE Data 15-47

COBOL (Pro*COBOL): Displaying BFILE Data 15-49
C/C++ (Pro*C/C++): Displaying BFILE Data 15-51
Visual Basic (OO4O): Displaying BFILE Data 15-53
Java (JDBC): Displaying BFILE Data 15-53
Reading Data from a BFILE 15-56
PL/SQL (DBMS_LOB): Reading Data from a BFILE 15-57
C (OCI): Reading Data from a BFILE 15-58
COBOL (Pro*COBOL): Reading Data from a BFILE 15-60
C/C++ (Pro*C/C++): Reading Data from a BFILE 15-61
Visual Basic (OO4O): Reading Data from a BFILE 15-62
Java (JDBC): Reading Data from a BFILE 15-63
Reading a Portion of BFILE Data Using SUBSTR 15-66
PL/SQL (DBMS_LOB): Reading a Portion of BFILE Data Using SUBSTR 15-67
COBOL (Pro*COBOL): Reading a Portion of BFILE Data Using SUBSTR 15-67
C/C++ (Pro*C/C++): Reading a Portion of BFILE Data Using SUBSTR 15-69
Visual Basic (OO4O): Reading a Portion of BFILE Data Using SUBSTR 15-70
Java (JDBC): Reading a Portion of BFILE Data Using SUBSTR 15-71
Comparing All or Parts of Two BFILES 15-73
PL/SQL (DBMS_LOB): Comparing All or Parts of Two BFILES 15-74
COBOL (Pro*COBOL): Comparing All or Parts of Two BFILES 15-74
C/C++ (Pro*C/C++): Comparing All or Parts of Two BFILES 15-76
Visual Basic (OO4O): Comparing All or Parts of Two BFILES 15-78
Java (JDBC): Comparing All or Parts of Two BFILES 15-79
Checking If a Pattern Exists in a BFILE Using INSTR 15-82
PL/SQL (DBMS_LOB): Checking If a Pattern Exists in a BFILE Using INSTR 15-83
COBOL (Pro*COBOL): Checking If a Pattern Exists in a BFILE Using INSTR 15-83
C/C++ (Pro*C/C++): Checking If a Pattern Exists in a BFILE Using INSTR 15-85
Java (JDBC): Checking If a Pattern Exists in a BFILE Using INSTR 15-87
Determining Whether a BFILE Exists 15-89
PL/SQL (DBMS_LOB): Determining Whether a BFILE Exists 15-90
C (OCI): Determining Whether a BFILE Exists 15-90
COBOL (Pro*COBOL): Determining Whether a BFILE Exists 15-91
C/C++ (Pro*C/C++): Determining Whether a BFILE Exists 15-93
Visual Basic (OO4O): Determining Whether a BFILE Exists 15-94
Java (JDBC): Determining Whether a BFILE Exists 15-95

Getting the Length of a BFILE 15-97
PL/SQL (DBMS_LOB): Getting the Length of a BFILE 15-98
C (OCI): Getting the Length of a BFILE........ 15-98
COBOL (Pro*COBOL): Getting the Length of a BFILE 15-99
C/C++ (Pro*C/C++): Getting the Length of a BFILE 15-100
Visual Basic (OO4O): Getting the Length of a BFILE 15-102
Java (JDBC): Getting the Length of a BFILE........ 15-103
Assigning a BFILE Locator 15-105
PL/SQL: Assigning a BFILE Locator 15-106
C (OCI): Assigning a BFILE Locator 15-106
COBOL (Pro*COBOL): Assigning a BFILE Locator........ 15-107
C/C++ (Pro*C/C++): Assigning a BFILE Locator........ 15-108
Java (JDBC): Assigning a BFILE Locator 15-109
Getting Directory Object Name and Filename of a BFILE 15-111
PL/SQL (DBMS_LOB): Getting Directory Object Name and Filename 15-112
C (OCI): Getting Directory Object Name and Filename 15-112
COBOL (Pro*COBOL): Getting Directory Object Name and Filename........ 15-113
C/C++ (Pro*C/C++): Getting Directory Object Name and Filename........ 15-114
Visual Basic (OO4O): Getting Directory Object Name and Filename........ 15-115
Java (JDBC): Getting Directory Object Name and Filename 15-116
Updating a BFILE by Initializing a BFILE Locator........ 15-119
PL/SQL: Updating a BFILE by Initializing a BFILE Locator 15-120
C (OCI): Updating a BFILE by Initializing a BFILE Locator 15-120
COBOL (Pro*COBOL): Updating a BFILE by Initializing a BFILE Locator........ 15-121
C/C++ (Pro*C/C++): Updating a BFILE by Initializing a BFILE Locator........ 15-123
Visual Basic (OO4O): Updating a BFILE by Initializing a BFILE Locator........ 15-124
Java (JDBC): Updating a BFILE by Initializing a BFILE Locator 15-125
Closing a BFILE with FILECLOSE 15-127
PL/SQL (DBMS_LOB): Closing a BFILE with FILECLOSE 15-128
C (OCI): Closing a BFILE with FILECLOSE 15-128
Java (JDBC): Closing a BFile with FILECLOSE 15-129
Closing a BFILE with CLOSE........ 15-131
PL/SQL (DBMS_LOB): Closing a BFILE with CLOSE........ 15-132
C (OCI): Closing a BFile with CLOSE........ 15-133
COBOL (Pro*COBOL): Closing a BFILE with CLOSE 15-133

C/C++ (Pro*C/C++): Closing a BFile with CLOSE 15-135
Visual Basic (OO4O): Closing a BFile with CLOSE 15-136
Java (JDBC): Closing a BFile with CLOSE 15-136
Closing All Open BFILEs with FILECLOSEALL 15-139
PL/SQL (DBMS_LOB): Closing All Open BFiles 15-140
C (OCI): Closing All Open BFiles 15-140
COBOL (Pro*COBOL): Closing All Open BFiles 15-141
C/C++ (Pro*C/C++): Closing All Open BFiles 15-143
Visual Basic (OO4O): Closing All Open BFiles 15-144
Java (JDBC): Closing All Open BFiles 15-145
Inserting a Row Containing a BFILE 15-148
PL/SQL (DBMS_LOB): Inserting a Row Containing a BFILE 15-149
C (OCI): Inserting a Row Containing a BFILE 15-149
COBOL (Pro*COBOL): Inserting a Row Containing a BFILE 15-150
C/C++ (Pro*C/C++): Inserting a Row Containing a BFILE 15-152
Visual Basic (OO4O): Inserting a Row Containing a BFILE 15-153
Java (JDBC): Inserting a Row Containing a BFILE 15-154

A LOB Demonstration Files

PL/SQL LOB Demonstration Files A-2
OCI LOB Demonstration Files A-4
Pro*COBOL LOB Demonstration Files A-6
Pro*C LOB Demonstration Files A-9
Visual Basic OO4O LOB Demonstration Files A-11
Java LOB Demonstration Files A-13

Glossary

Index

Send Us Your Comments

Oracle Database Application Developer's Guide - Large Objects, 10*g* Release 1 (10.1)

Part No. B10796-01

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
- 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 database features that support applications using Large Object (LOB) datatypes. The information in this guide 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

Oracle Database Application Developer's Guide - Large Objects contains information that describes the features and functionality of Oracle Database 10*g*.

What You Need To Use LOBs

The database includes all of the resources you need to use LOBs in your application; however, there are some restrictions on how you can use LOBs as described in the following:

See Also:

- "LOB Restrictions" on page 2-8
- "Restrictions Removed in Oracle9i Release 2" on page xxxix
- "Restrictions for LOBs in Partitioned Index-Organized Tables" on page 4-22

Audience

Oracle Database Application Developer's Guide - Large Objects is intended for programmers developing new applications that use LOBs, as well as those who have already implemented this technology and now want 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

This guide, *Oracle Database Application Developer's Guide - Large Objects*, is organized as follows:

Part I, "Getting Started"

This part gives introductory information and explains concepts that you must be familiar with to use LOBs in your application.

Chapter 1, "Introduction to Large Objects"

This chapter gives an introduction to LOB datatypes and describes the kinds of applications in which LOBs are useful.

Chapter 2, "Working with LOBs"

This chapter gives guidelines on working with LOB instances in your application and the database.

Chapter 3, "Managing LOBs: Database Administration"

This chapter describes database administration tasks required to setup and use databases with LOBs. Issues that application developers and database administrators must coordinate on are described.

Part II, "Application Design"

This part covers application design issues for applications that use LOBs.

Chapter 4, "LOBs in Tables"

This chapter describes how to create basic tables that contain LOB columns and gives general guidelines on selecting the best table architecture for your application.

Chapter 5, "Advanced Design Considerations"

This chapter describes more advanced application and database design issues that you may encounter when using LOBs in your application such as: buffering, caching, locators, transactions, and supersized LOBs.

Chapter 6, "Overview of Supplied LOB APIs"

This chapter gives an overview of APIs supplied with the database for using LOBs in applications.

Chapter 7, "Performance Guidelines"

This chapter discusses performance issues you should consider when designing applications that use LOBs.

Part III, "SQL Access to LOBs"

This part describes SQL usage with LOB datatypes in the SQL and PL/SQL environments.

Chapter 8, "DDL and DML Statements with LOBs"

This chapter discusses DDL and DML statements for common tasks performed on tables with LOB columns.

Chapter 9, "SQL Semantics and LOBs"

This chapter describes SQL semantics support for LOB datatypes.

Chapter 10, "PL/SQL Semantics for LOBs"

This chapter describes PL/SQL semantics support for LOB datatypes.

Chapter 11, "Migrating Table Columns from LONGs to LOBs"

This chapter describes techniques for migrating data in tables that use the LONG datatype to LOB columns.

Part IV, "Using LOB APIs"

This part gives details on LOB APIs supplied with the database.

Chapter 12, "Operations Specific to Persistent and Temporary LOBs"

This chapter describes APIs and procedures that vary depending on whether they are performed on persistent or temporary LOB instances.

Chapter 13, "Data Interface for Persistent LOBs"

This chapter describes the data interface for persistent LOBs.

Chapter 14, "LOB APIs for Basic Operations"

This chapter describes LOB APIs for operations that can be used to operate on either persistent LOB or temporary LOB instances.

Chapter 15, "LOB APIs for BFILE Operations"

This chapter describes LOB APIs that are used exclusively with BFILEs.

Appendix A, "LOB Demonstration Files"

This appendix lists and describes sample code included in this book for demonstration purposes.

Glossary

Defines terms used in discussions about LOBs.

Related Documents

For more information, see the following manuals:

- *PL/SQL Packages and Types Reference:* Use this book to learn PL/SQL and to get a complete description of this high-level programming language, which is a 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++ 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 Programmer's Guide*: The Pro*COBOL precompiler enables you to embed SQL and PL/SQL in your COBOL programs for access to Oracle Server.
- P*rogrammer'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.
- **Java:** Oracle Database offers the opportunity of working with Java in the database. The Oracle Java documentation set includes the following:
  - *Oracle Database JDBC Developer's Guide and Reference*
  - *Oracle Database Java Developer's Guide*
  - *Oracle Database JPublisher User's Guide*

Oracle Database *error message documentation* is only available in HTML. If you only have access to the Oracle Documentation CD, you can browse the error messages by range. Once you find the specific range, use your browser "find in page" feature to locate the specific message. When connected to the Internet, you can search for a specific error message using the error message search feature of the Oracle online documentation.

Multimedia

You can access the Oracle 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 the Oracle extensibility framework in *Oracle Data Cartridge Developer's Guide*
- To use the Oracle *inter*Media applications, refer to the following:
  - *Oracle interMedia Reference*.
  - *Oracle interMedia Java Classes Reference*
  - *Oracle Text Reference*
  - *Oracle Text Application Developer's Guide*
  - *Oracle interMedia Reference*

Basic References

- For SQL information, see the *Oracle Database SQL Reference* and *Oracle Database Administrator's Guide*
- For information about using LOBs with Oracle XML DB, refer to *Oracle XML DB Developer's Guide*
- For information about Oracle XML SQL with LOB data, refer to *Oracle Database Advanced Replication*
- For basic Oracle concepts, see *Oracle Database Concepts*.
- For information on using Oracle Data Pump, SQL*Loader, and other database utilities, see *Oracle Database Utilities*

Printed documentation is available for sale in the Oracle Store at

```
http://oraclestore.oracle.com/
```

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
```

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
Bold	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 **index-organized table**.
Italics	Italic typeface indicates book titles or emphasis.	*Oracle Database Concepts* Ensure that the recovery catalog and target database do *not* reside on the same disk.
`UPPERCASE monospace (fixed-width 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. You can back up the database by using the `BACKUP` command. Query the `TABLE_NAME` column in the `USER_TABLES` data dictionary view. Use the `DBMS_STATS.GENERATE_STATS` procedure.
`lowercase monospace (fixed-width 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. **Note:** Some programmatic elements use a mixture of UPPERCASE and lowercase. Enter these elements as shown.	Enter `sqlplus` to open SQL*Plus. The password is specified in the `orapwd` file. Back up the datafiles and control files in the `/disk1/oracle/dbs` directory. The `department_id`, `department_name`, and `location_id` columns are in the `hr.departments` table. Set the `QUERY_REWRITE_ENABLED` initialization parameter to `true`. Connect as `oe` user. The `JRepUtil` class implements these methods.
`lowercase monospace (fixed-width font) italic`	Lowercase monospace italic font represents placeholders or variables.	You can specify the *`parallel_clause`*. Run `U`*`old_release`*`.SQL` where *`old_release`* 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 (`*`digits`*` [ , `*`precision`*` ])`
{ }	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}` `[COMPRESS | NOCOMPRESS]`
`...`	Horizontal ellipsis points indicate either:	
	■ That we have omitted parts of the code that are not directly related to the example	`CREATE TABLE ... AS `*`subquery`*`;`
	■ That you can repeat a portion of the code	`SELECT `*`col1`*`, `*`col2`*`, ... , `*`coln`*` 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;`
		`NAME`
		`------------------------------------`
		`/fsl/dbs/tbs_01.dbf`
		`/fs1/dbs/tbs_02.dbf`
		`.`
		`.`
		`.`
		`/fsl/dbs/tbs_09.dbf`
		`9 rows selected.`

Other notation	You must enter symbols other than brackets, braces, vertical bars, and ellipsis points as shown.	`acctbal NUMBER(11,2);` `acct    CONSTANT NUMBER(4) := 3;`
`Italics`	Italicized text indicates placeholders or variables for which you must supply particular values.	`CONNECT SYSTEM/`*`system_password`* `DB_NAME = `*`database_name`*

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.	`SELECT last_name, employee_id FROM employees;`
		`SELECT * FROM USER_TABLES;`
		`DROP TABLE hr.employees;`

`lowercase`	Lowercase typeface indicates programmatic elements that you supply. For example, lowercase indicates names of tables, columns, or files.
	Note: Some programmatic elements use a mixture of UPPERCASE and lowercase. Enter these elements as shown.
		`SELECT last_name, employee_id FROM employees;`
		`sqlplus hr/hr`
		`CREATE USER mjones IDENTIFIED BY ty3MU9;`

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 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 does not own or control. Oracle neither evaluates nor makes any representations regarding the accessibility of these Web sites.

What's New in Large Objects?

This section describes the new features in the following releases:

- LOB Features Introduced in Oracle Database 10g Release 1
- LOB Features Introduced in Oracle9i Release 2
- LOB Features Introduced in Oracle9i Release 1

LOB Features Introduced in Oracle Database 10*g* Release 1

The following features are introduced in Oracle Database 10*g* Release 1 (10.1):

- **Increased LOB size limit**

  The maximum size limit for LOBs is 8 to 128 terabytes, depending on your database block size. The following APIs support this new size limit:

  - DBMS_LOB PL/SQL package
  - OCI
  - JDBC

  Previous releases supported LOBs up to a maximum size of 4 GB. For details see "Terabyte-Size LOB Support" on page 5-30.

- **Performance Enhancements**

  A number of performance enhancements have been added for this release including:

  - LOB performance in `INSERT`, `UPDATE`, and `SELECT` operations is greatly enhanced in this release. For more information on maximizing LOB performance, see "Temporary LOB Performance Guidelines" on page 7-3.
  - Direct support for LOBs in the JDBC Thin driver.
    The JDBC Thin driver now provides direct support for BFILEs, BLOBs, and CLOBs. Prior to this release, it supported them through calls to PL/SQL routines.

- **Heterogeneous Cross-Platform Transportable Tablespace Support for LOBs**

  Support for LOBs in heterogeneous cross-platform transportable tablespaces is introduced in this release.

  See Also: *Oracle Database Administrator's Guide* for details on transportable tablespaces

- **Regular Expression Support**

  A set of SQL functions introduced in this release allow you to perform queries and manipulate string data stored in LOB types and other character datatypes using regular expressions.

See Also:

- *Oracle Database Application Developer's Guide - Fundamentals* for information on supported regular expression syntax.
- *Oracle Database SQL Reference* for additional details on Oracle SQL functions for regular expressions.
- *Oracle Database Globalization Support Guide* for details on using SQL regular expression functions in a multilingual environment.
- *Mastering Regular Expressions* published by O'Reilly & Associates, Inc.

- **Implicit Conversion Between NCLOB and CLOB Datatypes**

  This release introduces support for implicit conversions between NCLOB and CLOB datatypes. You can perform operations such as cross-type assignments and parameter passing between these types without losing data or character set formatting properties.

  See Also: "Implicit Conversion Between CLOB and NCLOB Datatypes in SQL" on page 9-8

- **Partitioned Index-Organized Tables - LOB Support**

  LOB columns are now supported in all types of partitioned index-organized tables.

  See Also: "LOBs in Index Organized Tables" on page 4-20

- **LogMiner Support for More Types: LONG, Multibyte CLOB and NCLOB**

  LogMiner and SQL Apply now support multibyte CLOB and NCLOB data. SQL Apply now also supports LONG data. Support of additional datatypes means that you can now mine a greater variety of data.

  See Also: *Oracle Data Guard Concepts and Administration*

- **New Column in V$TEMPORARY_LOBS**

  A new column named `'ABSTRACT_LOBS'` has been added to the `V$TEMPORARY_LOBS` table. This column displays the number of abstract LOBs accumulated in the current session. Abstract LOBs are temporary lobs returned from queries involving XMLType columns. See the *Oracle Database Reference* for details on the V$TEMPORARY_LOBS table.

Restrictions Removed in Oracle Database 10*g* Release 1

The following restrictions on LOB features were removed in Oracle Database 10*g* Release 1:

- NCLOB as an attribute of an object type at table creation

  In previous releases you could not specify an NCLOB as an attribute of an object type when creating a table. This restriction no longer applies.

- Restrictions on LOBs in index organized tables were removed. See "Restrictions for LOBs in Partitioned Index-Organized Tables" on page 4-22 for more information.

LOB Features Introduced in Oracle9*i* Release 2

This section describes features introduced in Oracle9*i* 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 enables you to load binary large objects from operating system files into internal persistent LOBs and temporary LOBs.

  See Also: "Loading a BLOB with Data from a BFILE" on page 14-26

- **DBMS_LOB.LOADCLOBFROMFILE**

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

  See Also: "Loading a CLOB or NCLOB with Data from a BFILE" on page 14-29

- **Parallel Execution Support for DML Operations on LOBs**

Support for parallel execution of the following DML operations on tables with LOB columns is introduced in this release. These operations run in parallel execution mode only when performed on a partitioned table. If the table is not partitioned, then these operations run in serial execution mode.

- `INSERT AS SELECT`
- `CREATE TABLE AS SELECT`
- `DELETE`
- `UPDATE`
- `MERGE` (conditional `UPDATE` and `INSERT`)
- Multitable `INSERT`

Restrictions Removed in Oracle9*i* Release 2

The following restrictions are removed in Oracle9*i* Release 2 (9.2):

- **Trigger restrictions removed**

  This release supports DML `BEFORE ROW` Trigger `:new` for LOBs. This means that triggers on LOBs follow the same rules as triggers on any other type of column.

  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.

  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 */
  ```

- **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.

- **Selecting a bind variable into a LOB column**

  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 Features Introduced in Oracle9*i* Release 1

The following LOB features were introduced in Oracle9*i* Release 1 (9.0.1):

- Data Interface for LOBs

  Using the data interface for LOBs, you can bind and define character data for CLOB columns and binary data for BLOB columns. Doing so, enables you to insert data directly into the LOB column and select data from the LOB column without using a LOB locator.

  When using a version of the Oracle Database client that differs from the version of the Oracle Database server, queries produce different results when a client application selects a LOB column defining it as a character type or a LOB type. The following table outlines the characteristics of various Oracle Database client and server combinations in this release and prior to this release.

Client Release	LOB Column Defined on the Client Side As	Result Using Server from Oracle Database Release 1 (9.0.1) and higher	Result Using Server prior to Oracle Database Release 1 (9.0.1)
9.0.1 and higher	Character type	Server sends data	Client raises error.
9.0.1 and higher	LOB type	Server sends locator	Server sends locator.
Prior to Rel.9.0.1	Character type	Client raises an error	Client raises error.
Prior to Rel.9.0.1	LOB type	Server sends locator	Server sends locator.

See Also: Chapter 13, "Data Interface for Persistent 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 9, "SQL Semantics and LOBs"

- Using Oracle C++ Call Interface (OCCI) with LOBs

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

  See Also: Chapter 6, "Overview of Supplied LOB APIs"

- 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.

- Support for LOBs in Partitioned Index-Organized Tables

  Oracle9*i* introduces support for LOB, VARRAY columns stored as LOBs, and BFILEs in partitioned index-organized tables. Results of queries on 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, "Advanced Design Considerations"

- 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: "Oracle Provider for OLE DB (OraOLEDB)" on page 6-60

Restrictions Removed in Oracle9*i* Release 1

This section describes restrictions removed in Oracle9*i* Release 1 (9.0.1).

In earlier releases, you could not call functions and procedures in DBMS_LOB packages from client-side PL/SQL. This restriction is removed in release version 9.0.1. In this release, you can call DBMS_LOB functions and procedures from client-side or server-side PL/SQL.

Part I

Getting Started

This part gives an introduction to Large Objects and introduces general concepts you need to be familiar with to use LOBs in your application.

This part contains the following chapters:

- Chapter 1, "Introduction to Large Objects"
- Chapter 2, "Working with LOBs"
- Chapter 3, "Managing LOBs: Database Administration"

1

Introduction to Large Objects

This chapter introduces Large Objects (LOBs) and discusses how LOB datatypes are used in application development. This chapter includes the following sections:

- What Are Large Objects?
- Why Use Large Objects?
- Why Not Use LONGs?
- Different Kinds of LOBs
- Introducing LOB Locators
- Database Semantics for Internal and External LOBs
- Large Object Datatypes
- Abstract Datatypes and LOBs
- Storing and Creating Other Datatypes with LOBs

What Are Large Objects?

Large Objects (LOBs) are a set of datatypes that are designed to hold large amounts of data. A LOB can hold up to a maximum size ranging from 8 terabytes to 128 terabytes depending on how your database is configured. Storing data in LOBs enables you to access and manipulate the data efficiently in your application.

Why Use Large Objects?

This section introduces different types of data that you encounter when developing applications and discusses which kinds of data are suitable for large objects.

In the world today, applications must deal with the following kinds of data:

- Simple structured data.

  This data can be organized into simple tables that are structured based on business rules.

- Complex structured data

  This kind of data is complex in nature and is suited for the object-relational features of the Oracle database such as collections, references, and user-defined types.

- Semi-structured data

  This kind of data has a logical structure that is not typically interpreted by the database. For example, an XML document that is processed by your application or an external service, can be thought of as semi-structured data. The database provides technologies such as Oracle XML DB, Advanced Queueing, and Messages to help your application work with semi-structured data.

- Unstructured data

  This kind of data is not broken down into smaller logical structures and is not typically interpreted by the database or your application. A photographic image stored as a binary file is an example of unstructured data.

Large objects are suitable for these last two kinds of data: semi-structured data and unstructured data. Large objects features allow you to store these kinds of data in the database as well as in operating system files that are accessed from the database.

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

- Can store unstructured and semi-structured data in an efficient manner.

- Is optimized for large amounts of data.
- Provides a uniform way of accessing data stored within the database or outside the database.

Using LOBs for Semi-structured Data

Examples of semi-structured data include document files such as XML documents or word processor files. These kinds of documents contain data in a logical structure that is processed or interpreted by an application, and is not broken down into smaller logical units when stored in the database.

Applications involving semi-structured data typically use large amounts of character data. The Character Large Object (`CLOB`) and National Character Large Object (`NCLOB`) datatypes are ideal for storing and manipulating this kind of data.

Binary File objects (`BFILE` datatypes) can also store character data. You can use `BFILE`s to load read-only data from operating system files into `CLOB` or `NCLOB` instances that you then manipulate in your application.

Using LOBs for Unstructured Data

Unstructured data cannot be decomposed into standard components. For example, data about an employee can be structured into a name, which is stored as a string; an identifier, such as an ID number, a salary and so on. A photograph, on the other hand, consists of a long stream of 1s and 0s. These bits are used to switch pixels on or off so that you can see the picture on a display, but are not broken down into any finer structure for database storage.

Also, unstructured data such as text, graphic images, still video clips, full motion video, and sound waveforms tends to be large in size. A typical employee record may be a few hundred bytes, while even small amounts of multimedia data can be thousands of times larger.

SQL datatypes that are ideal for large amounts of unstructured binary data include the `BLOB` datatype (Binary Large Object) and the `BFILE` datatype (Binary File object).

Why Not Use LONGs?

The database 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 enables you to easily migrate your existing applications that access LONG columns, to use LOB columns.

> **See Also:** Chapter 11, "Migrating Table Columns from LONGs to LOBs"

Applications developed for use with Oracle Database version 7 and earlier, used the LONG or LONG RAW data type to store large amounts of unstructured data.

With the Oracle8*i* and later versions of the database, using LOB datatypes is recommended for storing large amounts of structured and semi-structured data. LOB datatypes have several advantages over LONG and LONG RAW types including:

- LOB Capacity: LOBs can store much larger amounts of data. LOBs can store 4GB of data or more depending on you system configuration. LONG and LONG RAW types are limited to 2GB of data.
- Number of LOB columns in a table: A table can have multiple LOB columns. LOB columns in a table can be of any LOB type. In Oracle Database 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.
- LOBs can also be object attributes.

Different Kinds of LOBs

Different kinds of LOBs can be stored in the database or in external files.

> **Note:** LOBs in the database are sometimes also referred to as *internal LOBs* or *internal persistent LOBs*.

Internal LOBs

LOBs in the database are stored inside database tablespaces in a way that optimizes space and provides efficient access. The following SQL datatypes are supported for declaring internal LOBs: `BLOB`, `CLOB`, and `NCLOB`. Details on these datatypes are given in "Large Object Datatypes" on page 1-7.

Persistent and Temporary LOBs

Internal LOBs (LOBs in the database) can be either persistent or temporary. A persistent LOB is a LOB instance that exists in a table row in the database. A temporary LOB instance is created when you instantiate a LOB only within the scope of your local application.

A temporary instance becomes a persistent instance when you insert the instance into a table row.

Persistent LOBs use copy semantics and participate in database transactions. You can recover persistent LOBs in the event of transaction or media failure, and any changes to a persistent LOB value can be committed or rolled back. In other words, all the Atomicity Consistency Isolation Durability (ACID) properties that pertain to using database objects pertain to using persistent LOBs.

External LOBs and the BFILE Datatype

External LOBs are data objects stored in operating system files, outside the database tablespaces. The database accesses external LOBs using the SQL datatype `BFILE`. The `BFILE` datatype is the only external LOB datatype.

BFILEs are read-only datatypes. The database allows read-only byte stream access to data stored in BFILEs. You cannot write to a BFILE from within your application.

The database uses reference semantics with `BFILE` columns. Data stored in a table column of type `BFILE`, is physically located in an operating system file, not in the database tablespace.

You typically use BFILEs to hold:

- Binary data that does not change while your application is running, such as graphics.
- Data that is loaded into other large object types, such as a BLOB or CLOB where the data can then be manipulated.
- Data that is appropriate for byte-stream access, such as multimedia.
- Read-only data that is relatively large in size, to avoid taking up large amounts database tablespace.

Any storage device accessed by your operating system can hold BFILE data, including hard disk drives, CD-ROMs, PhotoCDs and DVDs. The database can access `BFILE`s provided the operating system supports stream-mode access to the operating system 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.

Introducing LOB Locators

A LOB instance has a locator and a value. The LOB locator is a reference to where the LOB value is physically stored. The LOB value is the data stored in the LOB.

When you use a LOB in an operation such as passing a LOB as a parameter, you are actually passing a LOB locator. For the most part, you can work with a LOB instance in your application without being concerned with the semantics of LOB locators. There is no need to dereference LOB locators, as is required with pointers in some programming languages.

There are some issues regarding the semantics of LOB locators and how LOB values are stored that you should be aware of. These details are covered in the context of the discussion where they apply throughout this guide.

See Also:

- "LOB Locator and LOB Value" on page 2-3
- "LOB Locators and BFILE Locators" on page 2-4
- "LOB Storage" on page 4-7

Database Semantics for Internal and External LOBs

In all programmatic environments, database semantics differ between internal LOBs and external LOBs as follows:

- Internal LOBs use *copy semantics*.

  With copy semantics, both the LOB locator and LOB value are logically copied during insert, update, or assignment operations. This ensures that each table cell or each variable containing a LOB, holds a unique LOB instance.

- External LOBs use *reference semantics*.

  With reference semantics, only the LOB locator is copied during insert operations. (Note that update operations do not apply to external LOBs as external LOBs are read-only. This is explained in more detail later in this section.)

Large Object Datatypes

Table 1–1 describes each large object datatype supported by the database and describes the kind of data each datatype is typically used for. The names of datatypes given here are the SQL datatypes provided by the database. In general, the descriptions given for the datatypes in this table and the rest of this book also apply to the corresponding datatypes provided for other programmatic environments. Also, note that the term "LOB" is generally used to refer to the set of all large object datatypes.

Table 1–1 Large Object Datatypes

SQL Datatype	Description
`BLOB`	Binary Large Object Stores any kind of data in binary format. Typically used for multimedia data such as images, audio, and video.
`CLOB`	Character Large Object Stores string data in the database character set format. Used for large strings or documents that use the database character set exclusively. Characters in the database character set are in a non-varying width format.
`NCLOB`	National Character Set Large Object Stores string data in National character set format. Used for large strings or documents in the National character set. Supports characters of varying width format.
`BFILE`	External Binary File A binary file stored outside of the database in the host operating system file system, but accessible from database tables. BFILEs can be accessed from your application on a **read-only** basis. Use BFILEs to store static data, such as image data, that does not need to be manipulated in applications. Any kind of data, that is, any operating system file, can be stored in a BFILE. For example, you can store character data in a BFILE and then load the BFILE data into a CLOB specifying the character set upon loading.

Abstract Datatypes and LOBs

You can declare LOB datatypes as fields, or members, of abstract datatypes. For example, you can have an attribute of type `CLOB` on an `object` type. In general, there is no difference in the usage of a LOB instance in a LOB column and the usage of a LOB instance that is a member or of an abstract datatype. Any difference in usage is called out when it applies. When used in this guide, the term **LOB attribute** refers to a LOB instance that is a member of an abstract datatype. Unless otherwise specified, discussions that apply to LOB columns also apply to LOB attributes.

Storing and Creating Other Datatypes with LOBs

You can use LOBs to create other user-defined datatypes or store other datatypes as LOBs. This section discusses some of the datatypes provided with the database as examples of datatypes that are stored or created with LOB types.

VARRAYs Stored as LOBs

An instance of type `VARRAY` in the database is stored as an array of LOBs when you create a table in the following scenarios:

- If the VARRAY storage clause— `VARRAY varray_item STORE AS` —is not specified, and the declared size of varray data is more than 4000 bytes.
- If the varray column properties are specified using the STORE AS LOB clause— `VARRAY varray_item STORE AS LOB ...`

XMLType Columns Stored as CLOBs

A good example of how LOB datatypes can be used to store other datatypes is the `XMLType` datatype. The `XMLType` datatype is stored as a CLOB type. Setting up your table or column to store XMLType datatypes as CLOBs enables you to store schema-less XML documents in the database.

See Also:

- *Oracle XML DB Developer's Guide* for information on creating XMLType tables and columns.
- *Oracle XML Developer's Kit Programmer's Guide,* for information about the XMLType datatype, and how XML is stored in CLOBs.

LOBs Used in Oracle *inter*Media

Oracle *inter*Media uses LOB datatypes to create datatypes specialized for use in multimedia application such as interMedia ORDAudio, ORDDoc, ORDImage, and ORDVideo. Oracle *inter*Media uses the database infrastructure to define object types, methods, and LOBs necessary to represent these specialized types of data in the database.

See Also:

- *Oracle interMedia User's Guide* for more information on using *inter*Media.
- *Oracle interMedia Reference* for more information on using *inter*Media datatypes.

2

Working with LOBs

This chapter describes the usage and semantics of LOBs that you need to be familiar with to use LOBs in your application. Various techniques for working with LOBs are covered.

Most of the discussions in this chapter regarding persistent LOBs assume that you are dealing with LOBs in tables that already exist. The task of creating tables with LOB columns is typically performed by your database administrator. See Chapter 4, "LOBs in Tables" of this guide for details on creating tables with LOB columns.

This chapter includes the following sections:

- LOB Column States
- Locking a Row Containing a LOB
- Opening and Closing LOBs
- LOB Locator and LOB Value
- LOB Locators and BFILE Locators
- Accessing LOBs
- LOB Restrictions

LOB Column States

The techniques you use when accessing a cell in a LOB column differ depending on the state of the given cell. A cell in a LOB Column can be in one of the following states:

- **NULL**

  The table cell is created, but the cell holds no locator or value.

- **Empty**

  A LOB instance with a locator exists in the cell, but it has no value. The length of the LOB is zero.

- **Populated**

  A LOB instance with a locator and a value exists in the cell.

Locking a Row Containing a LOB

You can lock a row containing a LOB to prevent other database users from writing to the LOB during a transaction. To lock a row containing a LOB, specify the `FOR UPDATE` clause when you select the row. While the row is locked, other users cannot lock or update a the LOB, until you end your transaction.

Opening and Closing LOBs

The LOB APIs include operations that enable you to explicitly open and close a LOB instance. You can open and close a persistent LOB instance of any type: `BLOB`, `CLOB`, `NCLOB`, or `BFILE`. You open a LOB to achieve one or both of the following results:

- Open the LOB in read only mode.

  This ensures that the LOB (both the LOB locator and LOB value) cannot be changed in your session until you explicitly close the LOB. For example, you can open the LOB to ensure that the LOB is not changed by some other part of your program while you are using the LOB in a critical operation. After you perform the operation, you can then close the LOB.

- Open the LOB in read write mode—persistent BLOB, CLOB, or NCLOB instances only.

  Opening a LOB in read write mode defers any index maintenance on the LOB column until you close the LOB. Opening a LOB in read write mode is only useful if there is an extensible index on the LOB column and you do not want the database to perform index maintenance every time you write to the LOB. This technique can increase the performance of your application if you are doing several write operations on the LOB while it is open.

If you open a LOB, then you must close the LOB at some point later in your session. This is the only requirement for an open LOB. While a LOB instance is open, you can perform as many operations as you want on the LOB—provided the operations are allowed in the given mode.

See Also: "Opening Persistent LOBs with the OPEN and CLOSE Interfaces" on page 5-12 for details on usage of these APIs.

LOB Locator and LOB Value

There are two techniques that you can use to access and modify LOB values:

- Using the Data Interface for LOBs
- Using the LOB Locator to Access and Modify LOB Values

Using the Data Interface for LOBs

You can perform bind and define operations on CLOB and BLOB columns in C applications using the data interface for LOBs in OCI. Doing so, enables you to insert or select out data in a LOB column without using a LOB locator as follows:

- Using a bind variable associated with a LOB column to insert character data into a CLOB, or RAW data into a BLOB.
- Using a define operation to define an output buffer in your application that will hold character data selected from a CLOB, or RAW data selected from a BLOB.

See Also: Chapter 13, "Data Interface for Persistent LOBs" for more information on implicit assignment of LOBs to other datatypes.

Using the LOB Locator to Access and Modify LOB Values

The value of a LOB instance stored in the database can be accessed through a LOB locator, a reference to the location of the LOB value. Database tables store only locators in `CLOB`, `BLOB`, `NCLOB` and `BFILE` columns. Note the following with respect to LOB locators and values:

- To access or manipulate a LOB value, you pass the LOB locator to the various LOB APIs.
- A LOB locator can be assigned to any LOB instance of the same type.
- The characteristics of a LOB as being temporary or persistent have nothing to do with the locator. The characteristics of temporary or persistent apply only to the LOB instance.

LOB Locators and BFILE Locators

There are some differences between the semantics of locators for LOB types `BLOB`, `CLOB`, and `NCLOB`; and the semantics of locators for the `BFILE` type that you need to be aware of:

- For LOB types `BLOB`, `CLOB`, and `NCLOB`, the LOB column stores a locator to the LOB value. Each LOB instance has its own distinct LOB locator and also a distinct copy of the LOB value.
- For initialized BFILE columns, the row stores a locator to the external operating system file that holds the value of the BFILE. Each BFILE instance in a given row has its own distinct locator; however, two different rows can contain a BFILE locator that points to the same operating system file.

Regardless of where the value of a LOB is stored, a locator is stored in the table row of any initialized LOB column. Note that when the term locator is used without an identifying prefix term, it refers to both LOB locators and BFILE locators. Also, when your select a LOB from a table, the LOB returned is always a temporary LOB. For more information on working with locators for temporary LOBs, see "LOBs Returned from SQL Functions" on page 9-12.

Initializing a LOB Column to Contain a Locator

Any LOB instance that is NULL does not have a locator. Before you can pass a LOB instance to any LOB API routine, the instance must contain a locator. For example, you can select a NULL LOB from a row, but you cannot pass the instance to the PL/SQL DBMS_LOB.READ procedure. The following sub-sections describe how to initialize a persistent LOB column and how to initialize a BFILE column.

Initializing a Persistent LOB Column

Before you can start writing data to a persistent LOB using the supported programmatic environment interfaces (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 persistent LOB to empty in an `INSERT/UPDATE` statement using the functions `EMPTY_BLOB` for `BLOB`s or `EMPTY_CLOB` for `CLOB`s and `NCLOB`s.

Note: You can use SQL to populate a LOB column with data even if it contains a NULL value.

See Also: Chapter 4, "LOBs in Tables" for more information on initializing LOB columns.

Running 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 *ad_sourcetext* with the character string 'my Oracle',
- Sets *ad_composite*, *ad_finaltext*, and *ad_fltextn* to an empty value,
- Sets *ad_photo* to `NULL`, and
- Initializes *ad_graphic* to point to the file *`my_picture`* located under the logical directory *`my_directory_object`* (see the `CREATE DIRECTORY` statement in *Oracle Database Reference*. for more information about creating a directory object).

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

```
INSERT INTO print_media VALUES (101, 1, EMPTY_BLOB(),
    'my Oracle', EMPTY_CLOB(), EMPTY_CLOB(),
    NULL, NULL, BFILENAME('directory_object', 'my_picture'), NULL, NULL);
```

Similarly, the LOB attributes for the *ad_header* column in `print_media` can be initialized to `NULL` or set to empty as shown in the following.

```
INSERT INTO print_media (product_id, ad_id, ad_header)
    VALUES (101, 1, adheader_typ('AD FOR ORACLE', sysdate,
    'Have Grid', EMPTY_BLOB()));
```

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

See Also:

- "Inserting a Row by Selecting a LOB From Another Table" on page 8-6
- "Inserting a LOB Value Into a Table" on page 8-7
- "Inserting a Row by Initializing a LOB Locator Bind Variable" on page 8-8
- "OCILobLocator Pointer Assignment" on page 6-13 for details on LOB locator semantics in OCI

BFILEs

Before you can access `BFILE` values using LOB APIs, the `BFILE` column or 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: "Accessing BFILEs" on page 15-4 for more information on initializing BFILE columns.

Accessing LOBs

You can access a LOB instance using the following techniques:

- Accessing a LOB Using SQL
- Accessing a LOB Using the Data Interface
- Accessing a LOB Using the Locator Interface

Accessing a LOB Using SQL

Support for columns that use LOB datatypes is built into many SQL functions. This support enables you to use SQL semantics to access LOB columns in SQL. In most cases, you can use the same SQL semantics on a LOB column that you would use on a VARCHAR2 column.

> **See Also:** For details on SQL semantics support for LOBs, see Chapter 9, "SQL Semantics and LOBs".

Accessing a LOB Using the Data Interface

You can select a LOB directly into CHAR or RAW buffers using the LONG-to-LOB API in OCI and PL/SQL. In the following PL/SQL example, AD_FINALTEXT is selected into a VARCHAR buffer final_ad.

```
DECLARE
    final_ad VARCHAR(32767);
BEGIN
    SELECT AD_FINALTEXT INTO final_ad FROM print_media
        WHERE PRODUCT_ID= 2056 and AD_ID= 12001 ;
    DBMS_OUTPUT.PUT_LINE(final_ad);
    /* more calls to read final_ad */
END;
```

> **See Also:** For more details on accessing LOBs using the data interface, see Chapter 13, "Data Interface for Persistent LOBs".

Accessing a LOB Using the Locator Interface

You can access and manipulate a LOB instance by passing the LOB locator to the LOB APIs supplied with the database. An extensive set of LOB APIs is provided with each supported programmatic environment. In OCI, a LOB locator is mapped to a locator pointer which is used to access the LOB value.

> **Note:** In all environments, including OCI, the LOB APIs operate on the LOB value implicitly—there is no need to "dereference" the LOB locator.

See Also:

- Chapter 6, "Overview of Supplied LOB APIs"
- "OCILobLocator Pointer Assignment" on page 6-13 for details on LOB locator semantics in OCI

LOB Restrictions

This section provides details on LOB restrictions.

See Also:

- "Restrictions Removed in Oracle Database 10g Release 1" on page -xxxviii
- "Restrictions Removed in Oracle9i Release 2" on page xxxix
- "Restrictions for LOBs in Partitioned Index-Organized Tables" on page 4-22
- Chapter 11, "Migrating Table Columns from LONGs to LOBs" under "Migrating Applications from LONGs to LOBs" on page 11-10, describes LONG to LOB migration limitations for clustered tables, replication, triggers, domain indexes, and function-based indexes.
- "Unsupported Use of LOBs in SQL" on page 9-11 for restrictions on SQL semantics.

Restrictions on LOB Columns

LOB columns are subject to the following restrictions:

- You cannot specify a LOB as a primary key column.
- 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;
  ```

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 select * from local_table;
UPDATE table1@remote_siteset lobcol = (SELECT lobcol FROM local_table);
DELETE FROM table1@remote_site <WHERE clause involving non_lob_columns>
```

This is the only supported syntax involving LOBs in remote tables. No other usage is supported.

In statements structured like the first three of the preceding examples, only standalone LOB columns are allowed in the select list.

SQL functions and `DBMS_LOB` APIs are not supported for use with remote LOB columns. 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 nonkey columns.
- The following data structures are supported only as temporary instances. You cannot store these instances in database tables:
  - Varray of any LOB type
  - Varray of any type containing a LOB type, such as an ADT with a LOB attribute.
- 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.
- 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 LOB columns in `ANALYZE... COMPUTE` or `ANALYZE... ESTIMATE` statements.

- The first (`INITIAL`) extent of a LOB segment must contain at least three database blocks.
- When creating an `UPDATE` DML trigger, you cannot specify a LOB column in the `UPDATE OF` clause.
- 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.
- 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.
- 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.

Note: 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, the database does not fire the DML trigger.

See Also:

- For details on the INITIAL extent of a LOB segment, see "Restriction on First Extent of a LOB Segment" on page 4-4.
- LOBs in partitioned index-organized tables are also subject to a number of other restrictions. See "Restrictions for LOBs in Partitioned Index-Organized Tables" on page 4-22 for more information.

Restrictions for LOB Operations

Other general LOB restrictions include the following:

- In SQL loader, A field read from a LOBFILE cannot be used as an argument to a clause. See "Database Utilities for Loading Data into LOBs" on page 3-2 for more information.
- Session migration is not supported for BFILEs in shared server (multithreaded server) 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.

3 Managing LOBs: Database Administration

This chapter describes administrative tasks that must be performed to setup, maintain, and use a database that contains LOBs.

This chapter contains these topics:

- Database Utilities for Loading Data into LOBs
- Managing Temporary LOBs
- Managing BFILEs
- Changing Tablespace Storage for a LOB

Database Utilities for Loading Data into LOBs

The following utilities are recommended for bulk loading data into LOB columns as part of database setup or maintenance tasks:

- SQL*Loader
- Oracle DataPump

> **Note:** Application Developers: If you are loading data into a LOB in your application, then using the LOB APIs is recommended. See Chapter 14, "LOB APIs for Basic Operations" for details on APIs that allow you to load LOBs from files.

Using SQL*Loader to Load LOBs

There are two general techniques for using SQL*Loader to load data into LOBs:

- Loading data from a primary data file
- Loading from a secondary data file using LOBFILEs

Consider the following issues when loading LOBs with SQL*Loader:

- 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, then 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, then 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, then internal code is probably batching up more rows in each load call than 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.
- Using LOBFILEs is recommended when loading columns containing XML data in CLOBs or XMLType columns. Whether you perform a direct-path load or a conventional path load with SQL*Loader depends on whether you need to validate XML documents upon loading.
  - If the XML document must be validated upon loading, then use conventional path load.
  - If it is not necessary to ensure that the XML document is valid or you can safely assume that the XML document is valid, then you can perform a direct-path load. Performance is higher when you use direct-path load because the overhead of XML validation is incurred.

  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 *Oracle Database Utilities*.
- 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 old data before loading the new data in its place.

    See Also: For details on using SQL*Loader to load LOBs and other details on SQL*Loader usage, refer to the *Oracle Database Utilities* guide.

Using SQL*Loader to Populate a BFILE Column

This section describes how to load data from files in the file system into a BFILE column.

See Also: "Supported Environments for BFILE APIs" on page 15-2

Note that 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 have already been created. See "Directory Object" on page 15-4 for more information on creating directory objects.

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.

See Also: *Oracle Database Utilities* for details on SQL*Loader syntax.

The following two examples illustrate the loading of BFILES.

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';
```

In the following example only the file name 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:

```
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.

In the following example, the `BFILE` and the `DIRECTORY` object are specified dynamically.

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:

```
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.

Using Oracle DataPump to Transfer LOB Data

You can use Oracle DataPump to transfer LOB data from one database to another.

See Also: For details on using Oracle DataPump, refer to the *Oracle Database Utilities* guide.

Managing Temporary LOBs

The database keeps track of temporary LOBs in each session, and provides a v$ view called `v$temporary_lobs`. From the session, the application can determine which user owns the temporary LOB. As a database administrator, you can use this view to monitor and guide any emergency cleanup of temporary space used by temporary LOBs.

Managing Temporary Tablespace for Temporary LOBs

Temporary tablespace is used to store temporary LOB data. As a database administrator you control data storage resources for temporary LOB data by controlling user access to temporary tablespaces and by the creation of different temporary tablespaces.

See Also: Refer to the *Oracle Database Administrator's Guide* for details on managing temporary tablespaces.

Managing BFILEs

This section describes administrative tasks for managing databases that contain BFILEs.

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 operating system directory path.

- The operating system directory path named in the Oracle DIRECTORY object should not contain any symbolic links in its components.

Setting Maximum Number of Open BFILEs

A limited number of `BFILE`s 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 used. If you want to alter this limit, then 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.

Changing Tablespace Storage for a LOB

As the database administrator, you can use the following techniques to change the default storage for a LOB after the table has been created:

- **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);
```

Part II

Application Design

This part covers issues that you need to consider when designing applications that use LOBs.

This part contains the following chapters:

- Chapter 4, "LOBs in Tables"
- Chapter 5, "Advanced Design Considerations"
- Chapter 6, "Overview of Supplied LOB APIs"
- Chapter 7, "Performance Guidelines"

4

LOBs in Tables

This chapter describes issues specific to tables that contain LOB columns. This chapter includes the following sections:

- Creating Tables That Contain LOBs
- Choosing a LOB Column Datatype
- Selecting a Table Architecture
- LOB Storage
- Indexing LOB Columns
- Manipulating LOBs in Partitioned Tables
- LOBs in Index Organized Tables
- Restrictions for LOBs in Partitioned Index-Organized Tables
- Updating LOBs in Nested Tables

Creating Tables That Contain LOBs

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

- Initializing Persistent LOBs to NULL or Empty
- Initializing Persistent LOB Columns to a Value
- Initializing BFILEs to NULL or a File Name
- LOB Storage, "Defining Tablespace and Storage Characteristics for Persistent LOBs"

Initializing Persistent LOBs to NULL or Empty

You can set a persistent LOB — that is, a LOB column in a table, or a LOB attribute in an object type that you defined— to be `NULL` or empty:

- *Setting a Persistent 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 a Persistent 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, then 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 6, "Overview of Supplied LOB APIs" for more information on supported environments.

Details for these options are given in the following discussions.

Setting a Persistent LOB to NULL

You may want to set a persistent 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 use a `SELECT` statement, such as the following, to determine whether the LOB holds a NULL value:

```
SELECT COUNT (*) FROM print_media WHERE ad_graphic IS NOT NULL;

SELECT COUNT (*) FROM print_media WHERE ad_graphic IS NULL;
```

Note that you cannot call OCI or DBMS_LOB functions on a NULL LOB, so you must then use an SQL `UPDATE` statement to reset the LOB column to a non-NULL (or empty) value.

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`, then there is no locator in the row.

Setting a Persistent LOB to Empty

You can initialize a persistent LOB to EMPTY rather that NULL. Doing so, enables you to obtain a locator for the LOB instance without populating the LOB with data. To set a persistent LOB to EMPTY, use the SQL function `EMPTY_BLOB()` or `EMPTY_CLOB()` in the `INSERT` statement:

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

As an alternative, you can use the RETURNING clause to obtain the LOB locator in one operation rather than calling a subsequent SELECT statement:

```
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

You can initialize the LOBs in `print_media` by using the following `INSERT` statement:

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

This sets the value of *ad_sourcetext*, *ad_fltextn*, *ad_composite*, and *ad_photo* to an empty value, and sets *ad_graphic* to `NULL`.

Initializing Persistent LOB Columns to a Value

Alternatively, LOB *columns*, but not LOB *attributes*, can be initialized to a value. That is — persistent LOB *attributes* differ from persistent LOB *columns* in that LOB attributes cannot be initialized to a value other than `NULL` or empty when the row is inserted into a table.

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

See Also: Chapter 5, "Advanced Design Considerations"

Initializing BFILEs to NULL or a File Name

A BFILE can be initialized to NULL or to a filename. To do so, you can use the BFILENAME() function.

See Also: "BFILENAME() and Initialization" on page 15-6.

Restriction on First Extent of a LOB Segment

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, then it will still work because it is possible for segments in permanent dictionary managed tablespaces to override the default storage setting of the tablespaces.

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, then LOB segments cannot be created in these tablespaces. This is because in these tablespace types, extent sizes are fixed and the default storage setting of the tablespaces is not ignored.

Choosing a LOB Column Datatype

When selecting a datatype, consider the following topics:

- LOBs Compared to LONG and LONG RAW Types
- Storing Varying-Width Character Data in LOBs
- Implicit Character Set Conversions with LOBs

LOBs Compared to LONG and LONG RAW Types

Table 4–1 lists the similarities and differences between LOBs, LONGs, and LONG RAW types.

Table 4–1 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.	In the case of a LONG or LONG RAW the entire value is stored in the table column.
For in-line LOBs, the database will store LOBs that are less than approximately 4,000 bytes of data 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 8 terabytes or more in size depending on your block size.	A LONG or LONG RAW instance is limited to 2 gigabytes in size.
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.
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 *Oracle Database Advanced Replication)*

Storing Varying-Width Character Data in LOBs

Varying-width character data in CLOB and NCLOB datatypes is stored in an internal format that is compatible with UCS2 Unicode character set format. This ensures that there is no storage loss of character data in a varying-width format. Also note the following if you are using LOBs to store varying-width character data:

- You can create tables containing CLOB and NCLOB columns even if you use a varying-width CHAR or NCHAR database character set.
- You can create a table containing a datatype that has a CLOB attribute regardless of whether you use a varying-width CHAR database character set.

Implicit Character Set Conversions with LOBs

For CLOB and NCLOB instances used 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.

The DBMS_LOB.LOADCLOBFROMFILE API, performs an implicit conversion from binary data to character data when loading to a CLOB or NCLOB. With the exception of DBMS_LOB.LOADCLOBFROMFILE, LOB APIs do not perform implicit conversions from binary data to character data.

For example, when you use the DBMS_LOB.LOADFROMFILE API to populate a CLOB or NCLOB, you are populating the LOB with binary data from a BFILE. In this case, you must perform character set conversions on the BFILE data before calling DBMS_LOB.LOADFROMFILE.

See Also: *Oracle 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
  - In-line and Out-of-Line LOB Storage
  - Defining Tablespace and Storage Characteristics for Persistent LOBs
  - LOB Storage Characteristics for LOB Column or Attribute
  - TABLESPACE and LOB Index
    - PCTVERSION
    - CACHE / NOCACHE / CACHE READS
    - LOGGING / NOLOGGING
    - CHUNK
    - ENABLE or DISABLE STORAGE IN ROW Clause

- LOBs in Index Organized Tables
- Manipulating LOBs in Partitioned Tables
- Using Domain Indexing on LOB Columns

LOB Storage

This section summarizes LOB storage characteristics to consider when designing tables with LOB column types.

In-line and Out-of-Line LOB Storage

LOB columns store locators that reference the location of the actual LOB value. Depending on the column properties you specify when you create the table, and depending the size of the LOB, actual LOB values are stored either in the table row (in-line) or outside of the table row (out-of-line).

LOB values are stored out-of-line when any of the following situations apply:

- By default. That is, if you do not specify a LOB parameter for the LOB storage clause when you create the table.
- When you explicitly specify DISABLE STORAGE IN ROW for the LOB storage clause when you create the table.
- When the size of the LOB is greater than 4K bytes, the LOB value for the LOB instance (regardless of the LOB storage properties for the column).

LOB values are stored in-line when any of the following conditions apply:

- When you explicitly specify ENABLE STORAGE IN ROW for the LOB storage clause when you create the table, and the size of the LOB stored in the given row is small, 4K bytes or less.
- When the LOB value is NULL (regardless of the LOB storage properties for the column).

Using the default LOB storage properties (in-line storage) can allow for better database performance; it avoids the overhead of creating and managing out-of-line storage for smaller LOB values. If LOB values stored in your database are frequently small in size, then using in-line storage is recommended.

Note:

- LOB locators are always stored in the row.
- A LOB locator always exists for any LOB instance regardless of the LOB storage properties or LOB value - NULL, empty, or otherwise.
- If the table is created with DISABLE STORAGE IN ROW properties and the LOB holds any data, then a minimum of one chunk of out-of-line storage space is used; even when the size of the LOB is less than the CHUNKSIZE.
- If a LOB column is initialized with `EMPTY_CLOB()` or `EMPTY_BLOB()`, then no LOB value exists, not even NULL. The row holds a LOB locator only. No additional LOB storage is used.
- LOB storage properties do not affect BFILE columns. BFILE data is always stored in operating system files outside the database.

Defining Tablespace and Storage Characteristics for Persistent LOBs

When defining LOBs in a table, you can explicitly indicate the tablespace and storage characteristics for each *persistent LOB* column as shown in the following example:

```
CREATE TABLE ContainsLOB_tab (n NUMBER, c CLOB)
      lob (c) STORE AS SEGNAME (TABLESPACE lobtbs1 CHUNK 4096
                        PCTVERSION 5
                        NOCACHE LOGGING
                        STORAGE (MAXEXTENTS 5)
                       );
```

Note: There are no tablespace or storage characteristics that you can specify for *external* LOBs as they are not stored in the database.

If you need to modify the LOB storage parameters on an existing LOB column, then use the `MODIFY LOB` clause of the `ALTER TABLE` statement.

Note: Only some storage parameters can be modified. For example, you can use the ALTER TABLE ... MODIFY LOB statement to change RETENTION, 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 in the previous example, 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 *Oracle 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` or `RETENTION`
  Note that you can specify either `PCTVERSION` or `RETENTION`, but not both.
- `CACHE/NOCACHE/CACHE READS`
- `LOGGING/NOLOGGING`
- `CHUNK`
- `ENABLE/DISABLE STORAGE IN ROW`
- `STORAGE`
  See the "STORAGE clause" in *Oracle Database SQL Reference* for more information.

For most users, defaults for these storage characteristics will be sufficient. If you want to fine-tune LOB storage, then 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, then 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, then the tablespace of the table is used for the LOB data and index.
- If you specify a tablespace for the LOB data, then both the LOB data and index use the tablespace that was specified.

Tablespace for LOB Index in Non-Partitioned Table

When creating a table, if you specify a tablespace for the LOB index for a non-partitioned table, then 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 tablespace of the table.

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 4–2, "Recommended PCTVERSION Settings" provides some guidelines for determining a suitable PCTVERSION value.

Table 4–2 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%

If your application requires several LOB updates concurrent with heavy reads of LOB columns, then consider using a higher value for PCTVERSION, such as 20%.

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

If persistent LOB instances in your application are created and written just once and are primarily read-only afterward, then updates are infrequent. In this case, consider using a lower value for PCTVERSION, such as 5% or lower.

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

RETENTION

As an alternative to the PCTVERSION parameter, you can specify the RETENTION parameter in the LOB storage clause of the CREATE TABLE or ALTER TABLE statement. Doing so, configures the LOB column to store old versions of LOB data for a *period of time*, rather than using a percentage of the table space. For example:

```
CREATE TABLE ContainsLOB_tab (n NUMBER, c CLOB)
      lob (c) STORE AS SEGNAME (TABLESPACE lobtbs1 CHUNK 4096
```

```
 RETENTION
 NOCACHE LOGGING
 STORAGE (MAXEXTENTS 5)
);
```

The RETENTION parameter is designed for use with Undo features of the database, such as Flashback Versions Query. When a LOB column has the RETENTION property set, old versions of the LOB data are retained for the amount of time specified by the UNDO_RETENTION parameter.

Note the following with respect to the RETENTION parameter:

- Undo SQL is not enabled for LOB columns as it is with other datatypes. You must set the RETENTION property on a LOB column to use Undo SQL on LOB data.
- You cannot set the value of the RETENTION parameter explicitly. The amount of time for retention of LOB versions in determined by the UNDO_RETENTION parameter.
- Usage of the RETENTION parameter is only supported in Automatic Undo Management mode. You must configure your table for use with Automatic Undo Management before you can set RETENTION on a LOB column.
- The LOB storage clause can specify RETENTION or PCTVERSION, but not both.

See Also:

- *Oracle Database Application Developer's Guide - Fundamentals* for more information on using flashback features of the database.
- *Oracle Database SQL Reference* for details on LOB storage clause syntax.

CACHE / NOCACHE / CACHE READS

When creating tables that contain LOBs, use the cache options according to the guidelines in Table 4–3, "When to Use CACHE, NOCACHE, and CACHE READS":

Table 4–3 When to Use CACHE, NOCACHE, and CACHE READS

Cache Mode	Read ...	Written To ...
CACHE READS	Frequently	Once or occasionally
CACHE	Frequently	Frequently
NOCACHE (default)	Once or occasionally	Never

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.

LOGGING / NOLOGGING

[NO] `LOGGING` has a similar application with regard to using LOBs as it does for other table operations. In the usual case, if the [NO]`LOGGING` clause is omitted, then 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, then you will not be able to recover your changes from the log because 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 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 OCILobRead2(), OCILobWrite2(), 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, then 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, then 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, then 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, then make sure they are set in the following manner:

- CHUNK <= NEXT
- CHUNK <= INITIAL

ENABLE or DISABLE STORAGE IN ROW Clause

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`, then 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, then 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.

Indexing LOB Columns

This section discusses different techniques you can use to index LOB columns.

Using Domain Indexing on LOB Columns

You might be able to improve the performance of queries by building indexes specifically attuned to your domain. Extensibility interfaces provided with the database allow for domain indexing, a framework for implementing such domain specific indexes.

Note: You cannot build a B-tree or bitmap index on a LOB column.

See Also: *Oracle Data Cartridge Developer's Guide* for information on building domain specific indexes.

Indexing LOB Columns Using a Text Index

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

See Also: *Oracle interMedia Reference* and *Oracle Text Reference*, for more information regarding Oracle *inter*Media options.

Function-Based Indexes 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 be built on nested tables. However, you can build function-based indexes on LOB columns and varrays.

Like extensible indexes and domain indexes on LOB columns, function-based indexes are also automatically updated when a DML operation is performed on the LOB column. Function-based indexes are also updated when any extensible index is updated.

See Also: *Oracle Database Application Developer's Guide - Fundamentals* for more information on using function-based indexes.

Extensible Indexing on LOB Columns

The database provides *extensible indexing*, a feature which enables you to define new index types as required. This is based on the concept of cooperative indexing where a data cartridge and the database 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 support this structure, the database provides 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, because 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, the database 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 collections, 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.

Extensible indexing functionality enables 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: *Oracle Data Cartridge Developer's Guide*

Oracle Text Indexing Support for XML

You can create Oracle Text indexes on CLOB columns and perform queries on XML data.

See Also:

- *Oracle XML Developer's Kit Programmer's Guide*
- *Oracle Text Reference*
- *Oracle Text Application Developer's Guide*

Manipulating LOBs in Partitioned Tables

You can partition tables that contain LOB columns. As a result, LOBs can take advantage of all of the benefits of partitioning including the following:

- 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 become easier to maintain.
- LOBs can be partitioned into logical groups to speed up operations on LOBs that are accessed as a group.

This section describes some of the ways you can manipulate LOBs in partitioned tables.

Partitioning a Table Containing LOB Columns

LOBs are supported in RANGE partitioned, LIST partitioned, and HASH partitioned tables. Composite heap-organized tables can also have LOBs.

You can partition a table containing LOB columns using the following techniques:

- When the table is created using the `PARTITION BY ...` clause of the `CREATE TABLE` statement.
- Adding a partition to an existing table using the `ALTER TABLE ... ADD PARTITION` clause.

- Exchanging partitions with a table that already has partitioned LOB columns using the `ALTER TABLE ... EXCHANGE PARTITION` clause. Note that `EXCHANGE PARTITION` can only be used when both tables have the same storage attributes, for example, both tables store LOBs out-of-line.

Creating LOB partitions at the same time you create the table (in the `CREATE TABLE` statement) is recommended. If you create partitions on a LOB column when the table is created, then the column can hold LOBs stored either in-line or out-of-line LOBs.

After a table is created, new LOB partitions can only be created on LOB columns that are stored out-of-line. Also, partition maintenance operations, `SPLIT PARTITION` and `MERGE PARTITIONS`, will only work on LOB columns that store LOBs out-of-line. See "Restrictions for LOBs in Partitioned Index-Organized Tables" on page 4-22 for additional information on LOB restrictions.

Note that once a table is created, storage attributes cannot be changed. See "LOB Storage" on page 4-7 for more information about LOB storage attributes.

Creating an Index on a Table Containing Partitioned LOB Columns

To improve the performance of queries, you can create indexes on partitioned LOB columns. For example:

```
CREATE INDEX index_name
   ON table_name (LOB_column_1, LOB_column_2, ...) LOCAL;
```

Note that only domain and function-based indexes are supported on LOB columns. Other types of indexes, such as unique indexes are not supported with LOBs.

Moving Partitions Containing LOBs

You can move a LOB partition into a different tablespace. This is useful if the tablespace is no longer large enough to hold the partition. To do so, use the `ALTER TABLE ... MOVE PARTITION` clause. For example:

```
ALTER TABLE current_table MOVE PARTITION partition_name
   TABLESPACE destination_table_space
   LOB (column_name) STORE AS (TABLESPACE current_tablespace);
```

Splitting Partitions Containing LOBs

You can split a partition containing LOBs into two equally sized partitions using the `ALTER TABLE ... SPLIT PARTITION` clause. Doing so permits you to place one or both new partitions in a new tablespace. For example:

```
ALTER TABLE table_name SPLIT PARTITION partition_name
   AT (partition_range_upper_bound)
   INTO (PARTITION partition_name,
      PARTITION new_partition_name TABLESPACE new_tablespace_name
         LOB (column_name) STORE AS (TABLESPACE tablespace_name)
         ... ;
```

Merging Partitions Containing LOBs

You can merge partitions that contain LOB columns using the `ALTER TABLE ... MERGE PARTITIONS` clause. This technique is useful for reclaiming unused partition space. For example:

```
ALTER TABLE table_name
   MERGE PARTITIONS partition_1, partition_2
   INTO PARTITION new_partition TABLESPACE new_tablespace_name
      LOB (column_name) store as (TABLESPACE tablespace_name)
     ... ;
```

LOBs in Index Organized Tables

Index Organized Tables (IOTs) support internal and external LOB columns. For the most part, SQL DDL, DML, and piece wise operations on LOBs in IOTs produce the same results as those for normal tables. The only exception is the default semantics of LOBs during creation. The main differences are:

- ***Tablespace Mapping***: By default, or unless specified otherwise, the LOB data and index segments will be created in the tablespace in which the primary key index segments of the index organized table are created.
- ***In-line 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, then 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, then SQL will raise an error.

On the other hand, if an overflow segment has been specified, then LOBs in index organized tables will exactly mimic their semantics in conventional tables (see "Defining Tablespace and Storage Characteristics for Persistent LOBs" on page 4-8).

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, then both C2 and C3 would have been stored out of line by default.

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

Restrictions for LOBs in Partitioned Index-Organized Tables

LOB columns are supported in range-, list-, and hash-partitioned index-organized tables with the following restrictions:

- Composite partitioned index-organized tables are not supported.
- Relational and object partitioned index-organized tables (partitioned by range, hash, or list) can hold LOBs stored as follows; however, partition maintenance operations, such as `MOVE`, `SPLIT`, and `MERGE` are not supported with:
  - VARRAY datatypes stored as LOB datatypes
  - Abstract datatypes with LOB attributes
  - Nested tables with LOB types

See Also: Additional restrictions for LOB columns in general are given in "LOB Restrictions" on page 2-8.

Updating LOBs in Nested Tables

To update LOBs in a nested table, you must lock the row containing the LOB explicitly. To do so, you must specify the FOR UPDATE clause in the subquery prior to updating the LOB value.

Note that locking the row of a parent table does not lock the row of a nested table containing LOB columns.

Note: Nested tables containing LOB columns are the only data structures supported for creating collections of LOBs. You cannot create a VARRAY of any LOB datatype.

5

Advanced Design Considerations

This chapter describes design considerations for more advanced application development issues.

This chapter contains these topics:

- LOB Buffering Subsystem
- Opening Persistent LOBs with the OPEN and CLOSE Interfaces
- Read Consistent Locators
- LOB Locators and Transaction Boundaries
- LOBs in the Object Cache
- Terabyte-Size LOB Support
- Interfaces Not Supporting LOBs Greater Than 4 Gigabytes

LOB Buffering Subsystem

The database provides 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 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 buffer in the client 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.*** The LOB buffering subsystem is not a cache. The contents of a LOB buffer are not always the same as the LOB value in the server. Unless you explicitly flush the contents of a LOB 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 5-9).

- ***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 5-13.

  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 Through Updated Locators" on page 5-16.

- ***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, then 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, then 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-enable 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 Oracle assumes it is SQLCS_IMPLICIT.

LOB Buffering Subsystem Usage

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 page size = n x CHUNKSIZE ~= 32K.

A LOB 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.

LOB Buffering Subsystem Usage Scenario

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, the database allocates one or more of the free pages in the page pool to the LOB 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 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 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 buffer, Oracle allocates any available free page(s) from the page pool to the LOB buffer. If there are no free pages available in the page pool, then Oracle reallocates the pages as follows. It ages out the *least recently used* page among the *unmodified* pages in the LOB buffer and reallocates it for the current operation.

If no such page is available in the LOB buffer, then 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 the L1 buffer are dirty, with the remaining 2 containing unmodified data read in from the server. Assume similar conditions in the L2 buffer. Now, for the next buffered operation on L1, Oracle will reallocate the least recently used page from the two unmodified pages in the *L1 buffer*. Once all the 8 pages in the L1 buffer are used up for LOB writes, Oracle can service two more operations on L1 by allocating the two unmodified pages from the *L2 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, then 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 LOBs 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 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 want to update the LOB in the server, then 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.

The technique you use to flush the LOB buffer determines whether data in the buffer is cleared and has performance implications as follows:

- In the default mode, data is retained in the pages that were modified when the flush operation occurs. In this case, when you read or write to the same range of bytes, no round-trip to the server is necessary. Note that flushing the buffer, 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. Flushing the buffer using this technique 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 LOB buffering subsystem 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, or method in PL/SQL

  Note that 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 or write using the locator
  - Flush the LOB
  - Disable the locator for buffering

- Return to the calling function, procedure, or method in PL/SQL

Remember that the database never implicitly flushes the LOB buffer.

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:

    `OCILobRead2(), OCILobWrite2(), 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*** — `OCILobCopy2(), OCILobAppend(), OCILobErase2(), OCILobGetLength2(), OCILobTrim2(), OCILobWriteAppend2()`

    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 OCI example is based on the PM schema included with Oracle Database Sample Schemas.

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

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

   init_OCI();

   -- Establish a savepoint before start of LOB buffering subsystem
operations
   exec_statement("savepoint lbs_savepoint");

   -- Initialize bind variable to BLOB columns from buffered
   -- access:
   exec_statement("select ad_composite into lbs_loc1 from Print_media
       where ad_id = 12001");
   exec_statement("select ad_composite into lbs_loc2 from Print_media
       where ad_id = 12001 for update");
   exec_statement("select ad_composite into lbs_loc2 from Print_media
```

```
    where ad_id = 12001 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;
OCILobRead2(.., 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
       -- client-side buffer.
       -- lbs_loc1 is a read consistent locator.

       -- Write 4K of the LOB throgh lbs_loc2 starting from
       -- offset 1:
       amount = 4096; offset = 1; bufl = 4096;
       buffer = populate_buffer(4096);
       OCILobWrite2(.., 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 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;
       OCILobRead2(.., lbs_loc1, offset, &amount, buffer,
           bufl, ..);

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

```
    -- Write 20K bytes through lbs_loc2 starting from offset
    -- 10K
    amount = 20480; offset = 10240; bufl = 20480;
    buffer = populate_buffer(20480);
    OCILobWrite2(.., lbs_loc2, offset, amount, buffer,
         bufl, ..);

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

    -- The following write through lbs_loc3 will also
    -- result in an error:
    amount = 20000; offset = 1000; bufl = 20000;
    buffer = populate_buffer(20000);
    OCILobWrite2(.., lbs_loc3, offset, amount, buffer,
         bufl, ..);

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

-- The following update through lbs_loc3 will also
-- result in an error
OCILobFileCopy(.., lbs_loc3, lbs_loc2, ..);

if (exception)
    goto exception_handler;
    -- Locators enabled for buffering cannot be used with
    -- operations like Append, Copy, Trim and so on
    -- When done, flush the LOB 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 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 buffer resources.
          exception_handler:
        handle_exception_reporting();
        exec_statement("rollback to savepoint lbs_savepoint");
}
```

Opening Persistent LOBs with the OPEN and CLOSE Interfaces

The OPEN and CLOSE interfaces enable you to explicitly open a persistent LOB instance. When you open a LOB instance with the OPEN interface, the instance remains open until you explicitly close the LOB using the CLOSE interface. The ISOPEN interface enables you to determine whether a persistent LOB is already open.

Note that the open state of a LOB is associated with the LOB instance, not the LOB locator. The locator does not save any information indicating whether the LOB instance that it points to is open.

> **See Also:** "Opening and Closing LOBs" on page 2-2 for general information on situations that you would open a LOB instance.

Index Performance Benefits of Explicitly Opening a LOB

Explicitly opening a LOB instance can benefit performance of a persistent LOB in an indexed column.

If you do not explicitly open the LOB instance, then every modification to the LOB implicitly opens and closes the LOB instance. Any triggers on a domain index are fired each time the LOB is closed. Note that in this case, any domain indexes on the LOB are updated as soon as any modification to the LOB instance is made; the domain index is always valid and can be used at any time.

When you explicitly open a LOB instance, index triggers do not fire until you explicitly close the LOB. Using this technique can increase performance on index columns by eliminating unneeded indexing events until you explicitly close the

LOB. Note that any index on the LOB column is not valid until you explicitly close the LOB.

Working with Explicitly Open LOB Instances

If you explicitly open a LOB instance, then you must close the LOB before you commit the transaction.

Committing a transaction on the open LOB instance will cause an error. When this error occurs, the LOB instance is closed implicitly, any modifications to the LOB instance are saved, and the transaction is committed, but any indexes on the LOB column are not updated. In this situation, you must rebuild your indexes on the LOB column.

If you subsequently rollback the transaction, then the LOB instance is rolled back to its previous state, but the LOB instance is no longer explicitly open.

You must close any LOB instance that you explicitly open:

- Between DML statements that start a transaction, including `SELECT` ... `FOR UPDATE` and `COMMIT`
- Within an autonomous transaction block
- Before the end of a session (when there is no transaction involved)
  If you do not explicitly close the LOB instance, then it is implicitly closed at the end of the session and no index triggers are fired.

Keep track of the open or closed state of LOBs that you explicitly open. The following will cause an error:

- Explicitly opening a LOB instance that is already explicitly open.
- Explicitly closing a LOB instance that is already explicitly closed.

This occurs whether you access the LOB instance using the same locator or different locators.

Read Consistent Locators

Oracle Database provides the same read consistency mechanisms for LOBs as for all other database reads and updates of scalar quantities. Refer to *Oracle 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 selected locator, regardless of the existence of the `FOR UPDATE` clause, becomes a *read consistent locator,* and remains a read consistent locator until the LOB value is updated through that locator. A read consistent locator contains the snapshot environment as of the point in time of the `SELECT` operation.

This has some complex implications. Suppose you have created a read consistent locator (L1) by way of a `SELECT` operation. In reading the value of the persistent 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, then L1 does not see the L2 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*.

You can use the existence of multiple locators to access different transformations of the LOB value. However, in doing so, you must 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 the PRINT_MEDIA table and PL/SQL, three `CLOB` instances 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 `ad_sourcetext` is associated with the locator `clob_selected`.

- In the second operation (at t2), the value in ad_sourcetext is associated with the locator clob_updated. Because there has been no change in the value of ad_sourcetext 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 uses 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 PRINT_MEDIA VALUES (2056, 20010, EMPTY_BLOB(),
    'abcd', EMPTY_CLOB(), EMPTY_CLOB(), 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 ad_sourcetext INTO clob_selected
  FROM Print_media
  WHERE ad_id = 20010;

  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 Print_media SET ad_sourcetext = empty_clob()
      WHERE ad_id = 20010;
  -- 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 ad_sourcetext INTO clob_selected FROM Print_media WHERE
       ad_id = 20010;
  -- 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;
/
```

Updating LOBs Through Updated Locators

When you update the value of the persistent 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 enables 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 a persistent LOB value, the modification is always made to the most current LOB value.

Updating the value of the persistent 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 PRINT_MEDIA in the following example, a `CLOB` locator is created as `clob_selected`. Note the following progressions in the example, from times t1 through t3:

- At the time of the first `SELECT INTO` (at t1), the value in `ad_sourcetext` is associated with the locator `clob_selected`.
- In the second operation (at t2), the value in `ad_sourcetext` is modified through the `SQL UPDATE` statement, without affecting 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 PRINT_MEDIA VALUES (2056, 20020, EMPTY_BLOB(),
    'abcd', EMPTY_CLOB(), EMPTY_CLOB(), 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 ad_sourcetext INTO clob_selected
     FROM Print_media
     WHERE ad_id = 20020;

  -- At time t2:
  SELECT ad_sourcetext INTO clob_updated
     FROM Print_media
     WHERE ad_id = 20020
     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;
/
```

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 a given LOB value.

In the following example, using table PRINT_MEDIA, two CLOBs are created as potential locators:

- `clob_updated`
- `clob_copied`

Note these progressions in the example at times t1 through t5:

- At the time of the first `SELECT INTO` (at t1), the value in `ad_sourcetext` is associated with the locator `clob_updated`.
- The second operation (at time t2) copies the value in `clob_updated` to `clob_copied`. At this time, both locators see the same value. The example demonstrates this with a series of `DBMS_LOB.READ`() calls.
- At time t3, the program uses `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 time 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 PRINT_MEDIA VALUES (2056, 20030, EMPTY_BLOB(),
    'abcd', EMPTY_CLOB(), EMPTY_CLOB(), 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 ad_sourcetext INTO clob_updated FROM PRINT_MEDIA
      WHERE ad_id = 20030
```

```
    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 persistent 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 the L2 snapshot environment at the time of the operation is used.

In the following example, using the table PRINT_MEDIA, three CLOBs are created as potential locators:

- `clob_selected`
- `clob_updated`
- `clob_copied`

Note these progressions in the example at times t1 through t5:

- At the time of the first `SELECT INTO` (at t1), the value in `ad_sourcetext` 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 uses `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, the value associated with `clob_copied` (for example, without the new changes made by `clob_updated`) is inserted. This is demonstrated by the subsequent `DBMS_LOB.READ`() of the value just inserted.

Example

```
INSERT INTO PRINT_MEDIA VALUES (2056, 20020, EMPTY_BLOB(),
    'abcd', EMPTY_CLOB(), EMPTY_CLOB(), 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 ad_sourcetext INTO clob_updated FROM PRINT_MEDIA
      WHERE ad_id = 20020
      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 does not 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 PRINT_MEDIA VALUES (2056, 20022, EMPTY_BLOB(),
    clob_copied, EMPTY_CLOB(), EMPTY_CLOB(), NULL, NULL, NULL, NULL)
    RETURNING ad_sourcetext 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 and Transaction Boundaries

This section discusses the use of LOB locators in transactions and transaction IDs. A basic description of LOB locators and their operations is given in "LOB Locators and BFILE Locators" on page 2-4.

Note the following regarding LOB locators and transactions:

- Locators contain transaction IDs when:

  You Begin the Transaction, Then Select Locator.
  If you begin a transaction and subsequently select a locator, then 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, then 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.

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, then you cannot write to the LOB outside of that particular transaction.
- *Can Write Using Locator*: If the locator *does not* contain a transaction ID, then 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, then you cannot read or write using that locator.
- *Can Read, Not Write Using Locator With Non-Serializable Transactions*: If the transaction is non-serializable, then you can read, but not write outside of that transaction.

The following examples show the relationship between locators and *non-serializable* transactions

Selecting the Locator Outside of the Transaction Boundary

The following scenarios describe techniques for using locators in non-serializable transactions when the locator is selected outside of a transaction.

Scenario:

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.

Scenario:

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 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.

Selecting the Locator Within a Transaction Boundary

The following scenarios describe techniques for using locators in non-serializable transactions when the locator is selected within a transaction.

Scenario:

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 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 5-13 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.

Scenario:

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 is a transaction id in the locator and the transaction was previously committed or rolled back.

> **See Also:** "Read Consistent Locators" on page 5-13 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.

LOB Locators Cannot Span Transactions

Modifying a persistent LOB 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. The `INSERT` or `UPDATE` statement automatically starts a transaction and locks the row. Once this has occurred, the locator cannot 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 can be used to read the LOB value unless you are in a serializable transaction.

> **See Also:** "LOB Locators and Transaction Boundaries" on page 5-24, for more information about the relationship between LOBs and transaction boundaries.

In the following example, a CLOB locator is created: clob_updated

- At the time of the first SELECT INTO (at t1), the value in ad_sourcetext is associated with the locator clob_updated.
- The second operation (at t2), uses the DBMS_LOB.WRITE() function 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 PRINT_MEDIA VALUES (2056, 20010, EMPTY_BLOB(),
    'abcd', EMPTY_CLOB(), EMPTY_CLOB(), 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      ad_sourcetext
     INTO        clob_updated
     FROM        PRINT_MEDIA
     WHERE       ad_id = 20010
     FOR UPDATE;
     read_amount := 10;
     read_offset := 1;
     dbms_lob.read(clob_updated, read_amount, read_offset,
          buffer);
     dbms_output.put_line('clob_updated value: ' || buffer);
     -- This produces the output 'abcd'

     -- At time t2:
```

```
     write_amount := 3;
     write_offset := 5;
     buffer := 'efg';
     dbms_lob.write(clob_updated, write_amount, write_offset,
          buffer);
     read_amount := 10;
     dbms_lob.read(clob_updated, read_amount, read_offset,
         buffer);
     dbms_output.put_line('clob_updated value: ' || buffer);
     -- This produces the output 'abcdefg'

    -- At time t3:
    COMMIT;

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

LOBs in the Object Cache

Consider these object cache issues for internal and external LOB attributes:

- Persistent 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 a persistent 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 (BFILE) 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 object name and filename before reading from the BFILE.

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 5-14 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.

Terabyte-Size LOB Support

Terabyte-size LOBs—LOBs up to a maximum size of 8 to 128 terabytes depending on your database block size—are supported by the following APIs:

- Java using JDBC (Java Database Connectivity)
- PL/SQL using the DBMS_LOB Package
- C using OCI (Oracle Call Interface)

Note that other LOB APIs support LOBs up to a maximum size of 4 gigabytes. See "Interfaces Not Supporting LOBs Greater Than 4 Gigabytes" on page 5-32.

The sections that follow, discuss APIs provided to support terabyte-size LOBs in each of these environments.

Note:

- Programmatic environments other than JDBC, DBMS_LOB, and OCI do not support LOBs larger than 4 gigabytes. For LOB size limits in other environments, see "Interfaces Not Supporting LOBs Greater Than 4 Gigabytes" on page 5-32.
- The database does not support BFILEs larger than 4 gigabytes in any programmatic environment. Any additional file size limit imposed by your operating system also applies to BFILEs.

Maximum Storage Limit for Terabyte-Size LOBs

In supported environments, you can create and manipulate LOBs that are up to the maximum storage size limit for your database configuration. The maximum allowable storage limit for your configuration depends on the database block size setting, the value of the DB_BLOCK_SIZE initialization parameter, and is calculated as (4 gigabytes - 1) times the value of the DB_BLOCK_SIZE parameter. With the current allowable range for the database block size from 2k to 32k, the storage limit ranges from 8 terabytes to 128 terabytes.

See Also: *Oracle Database Administrator's Guide* for details on the DB_BLOCK_SIZE initialization parameter setting for your database installation.

This storage limit applies to all LOB types in environments that support terabyte-size LOBs; however, note that CLOB and NCLOB types are sized in characters while the BLOB type is sized in bytes.

Using Terabyte-Size LOBs with JDBC

You can use terabyte-size LOBs with all LOB APIs included in Oracle JDBC classes.

Using Terabyte-Size LOBs with the DBMS_LOB Package

You can use terabyte-size LOBs with all APIs in the DBMS_LOB PL/SQL package. The DBMS_LOB.GET_STORAGE_LIMIT function returns the storage limit for your database configuration.

See Also: *PL/SQL Packages and Types Reference* for details on the DB_BLOCK_SIZE initialization parameter setting for your database installation.

Using Terabyte-Size LOBs with OCI

The Oracle Call Interface API provides a set of functions specifically for operations on terabyte-size LOBs.

See Also: *Oracle Call Interface Programmer's Guide* for details on OCI functions that support terabyte-size LOBs.

Interfaces Not Supporting LOBs Greater Than 4 Gigabytes

You can create and use LOB instances up to 4 gigabytes in size—"gigabyte LOBs"—in the following programmatic environments:

- COBOL using the Pro*COBOL precompiler
- C/C++ using the Pro*C/C++ precompiler
- Visual Basic using OO4O (Oracle Objects for OLE)

Guidelines for Creating Gigabyte LOBs

To create gigabyte LOBs in supported environments, 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. For example, Solaris 2.5 only allows operating system 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 operating system 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, then extents will become unmanageable 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, then 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.

Creating a Tablespace and Table to Store Gigabyte LOBs

The following example illustrates how to create a tablespace and table to store gigabyte LOBs.

```
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 PRINT_MEDIA_BACKUP
  (PRODUCT_ID NUMBER(6),
   AD_ID NUMBER(6),
   AD_COMPOSITE BLOB,
   AD_SOURCETEXT CLOB,
   AD_FINALTEXT CLOB,
   AD_FLTEXTN NCLOB,
   AD_TEXTDOCS_NTAB TEXTDOC_TAB,
   AD_PHOTO BLOB,
   AD_GRAPHIC BLOB,
   AD_HEADER ADHEADER_TYP)
   NESTED TABLE AD_TEXTDOCS_NTAB STORE AS TEXTDOCS_NESTEDTAB5
   LOB(AD_SOURCETEXT) store as (tablespace lobtbs1 chunk 32768 pctversion 0
                                NOCACHE NOLOGGING
                                storage(initial 100M next 100M maxextents
                                unlimited pctincrease 0));
```

Note the following with respect to this example:

- The storage clause in this example is specified in the CREATE TABLESPACE statement.
- You can specify the storage clause in the CREATE TABLE statement as an alternative.
- The storage clause is not allowed in the CREATE TEMPORARY TABLESPACE statement.
- Setting the PCTINCREASE parameter to 0 is recommended for gigabyte LOBs. For small, or medium size lobs, the default PCTINCREASE value of 50 is recommended as it reduces the number of extent allocations.

6

Overview of Supplied LOB APIs

This chapter discusses the following topics:

- Programmatic Environments That Support LOBs
- Comparing the LOB Interfaces
- Using PL/SQL (DBMS_LOB Package) to Work with LOBs
- Using 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 LOBsUsing Java (JDBC) to Work with LOBs
- Oracle Provider for OLE DB (OraOLEDB)
- Overview of Oracle Data Provider for .NET (ODP.NET)

Programmatic Environments That Support LOBs

Table 6–1 lists the programmatic environments that support LOB functionality.

See Also: APIs for supported LOB operations are described in detail in:

- Chapter 12, "Operations Specific to Persistent and Temporary LOBs"
- Chapter 14, "LOB APIs for Basic Operations"
- Chapter 15, "LOB APIs for BFILE Operations"

Table 6–1 Programmatic Environments That Support LOBs

Language	Precompiler or Interface Program	Syntax Reference	In This Chapter See...
PL/SQL	DBMS_LOB Package	*PL/SQL Packages and Types Reference*	"Using PL/SQL (DBMS_LOB Package) to Work with LOBs" on page 6-7.
C	Oracle Call Interface for C (OCI)	*Oracle Call Interface Programmer's Guide*	"Using OCI to Work with LOBs" on page 6-11.
C++	Oracle Call Interface for C++ (OCCI)	*Oracle C++ Call Interface Programmer's Guide*	"Using C++ (OCCI) to Work with LOBs" on page 6-17.
C/C++	Pro*C/C++ Precompiler	*Pro*C/C++ Programmer's Guide*	"Using C/C++ (Pro*C) to Work with LOBs" on page 6-24.
COBOL	Pro*COBOL Precompiler	*Pro*COBOL Programmer's Guide*	"Using COBOL (Pro*COBOL) to Work with LOBs" on page 6-27.

Table 6–1 (Cont.) Programmatic Environments That Support LOBs

Language	Precompiler or Interface Program	Syntax Reference	In This Chapter See...
Visual Basic	Oracle Objects For OLE (OO4O)	Oracle Objects for OLE (OO4O) is a Windows-based product included with the database. There are no manuals for this product, only online help. Online help is available through the Application Development submenu of the database installation.	"Using Visual Basic (Oracle Objects for OLE (OO4O)) to Work with LOBs" on page 6-31."
Java	JDBC Application Programmatic Interface (API)	*Oracle Database JDBC Developer's Guide and Reference*.	"Using Java (JDBC) to Work with LOBs" on page 6-37.
OLEDB	OraOLEDB, an OLE DB provider for Oracle.	*Oracle Provider for OLE DB Developer's Guide*	

Comparing the LOB Interfaces

Table 6–2 and Table 6–3 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 functionality of the interfaces, with regards to LOBs, is described in the following sections.

Table 6–2 Comparing the LOB Interfaces, 1 of 2

PL/SQL: DBMS_LOB (dbmslob.sql)	C (OCI) (ociap.h)	C++ (OCCI) (occiData.h). Also for Clob and Bfile classes.	Pro*C/C++ and Pro*COBOL
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	OCILobAppend	Blob.append()	APPEND

Table 6–2 (Cont.) Comparing the LOB Interfaces, 1 of 2

PL/SQL: DBMS_LOB (dbmslob.sql)	C (OCI) (ociap.h)	C++ (OCCI) (occiData.h). Also for Clob and Bfile classes.	Pro*C/C++ and Pro*COBOL
N/A [use PL/SQL assign operator]	OCILobAssign		ASSIGN
N/A	OCILobCharSetForm	Clob.getCharsetForm (CLOB only)	N/A
N/A	OCILobCharSetId	Clob.getCharsetId() (CLOB only)	N/A
DBMS_LOB.CLOSE	OCILobClose	Blob.close()	CLOSE
N/A	N/A	Clob.closeStream()	N/A
DBMS_LOB.COPY	OCILobCopy2	Blob.copy()	COPY
N/A	OCILobDisableBuffering	N/A	DISABLE BUFFERING
N/A	OCILobEnableBuffering	N/A	ENABLE BUFFERING
DBMS_LOB.ERASE	OCILobErase2	N/A	ERASE
DBMS_LOB.FILECLOSE	OCILobFileClose	Clob.close()	CLOSE
DBMS_LOB.FILECLOSEALL	OCILobFileCloseAll	N/A	FILE CLOSE ALL
DBMS_LOB.FILEEXISTS	OCILobFileExists	Bfile.fileExists()	DESCRIBE [FILEEXISTS]
DBMS_LOB.GETCHUNKSIZE	OCILobGetChunkSize	Blob.getChunkSize()	DESCRIBE [CHUNKSIZE]
DBMS_LOB.FILEGETNAME	OCILobFileGetName	Bfile.getFileName() and Bfile.getDirAlias()	DESCRIBE [DIRECTORY, FILENAME]
DBMS_LOB.FILEISOPEN	OCILobFileIsOpen	Bfile.isOpen()	DESCRIBE [ISOPEN]
DBMS_LOB.FILEOPEN	OCILobFileOpen	Bfile.open()	OPEN
N/A (use BFILENAME operator)	OCILobFileSetName	Bfile.setName()	FILE SET
N/A	OCILobFlushBuffer	N/A	FLUSH BUFFER
DBMS_LOB.GETLENGTH	OCILobGetLength2	Blob.length()	DESCRIBE [LENGTH]
N/A	OCILobIsEqual	use operator = ( )=/!=	N/A
DBMS_LOB.ISOPEN	OCILobIsOpen	Blob.isOpen()	DESCRIBE [ISOPEN]
DBMS_LOB.LOADFROMFILE	OCILobLoadFromFile2	Use the overloadedcopy() method.	LOAD FROM FILE

Table 6–2 (Cont.) Comparing the LOB Interfaces, 1 of 2

PL/SQL: DBMS_LOB (dbmslob.sql)	C (OCI) (ociap.h)	C++ (OCCI) (occiData.h). Also for Clob and Bfile classes.	Pro*C/C++ and Pro*COBOL
N/A	OCILobLocatorIsInit	Clob.isinitialized()	N/A
DBMS_LOB.OPEN	OCILobOpen	Blob.open	OPEN
DBMS_LOB.READ	OCILobRead2	Blob.read	READ
DBMS_LOB.TRIM	OCILobTrim2	Blob.trim	TRIM
DBMS_LOB.WRITE	OCILobWrite2	Blob.write	WRITEORALOB.
DBMS_LOB.WRITEAPPEND	OCILobWriteAppend2	N/A	WRITE APPEND
DBMS_ LOB.CREATETEMPORARY	OCILobCreateTemporary	N/A	N/A
DBMS_ LOB.FREETEMPORARY	OCILobFreeTemporary	N/A	N/A
DBMS_LOB.ISTEMPORARY	OCILobIsTemporary	N/A	N/A
	OCILobLocatorAssign	use operator = ( ) or copy constructor	N/A

Table 6–3 Comparing the LOB Interfaces, 2 of 2

PL/SQL: DBMS_LOB (dbmslob.sql)	Visual Basic (OO4O)	Java (JDBC	OLEDB
DBMS_LOB.COMPARE	ORALOB.Compare	Use DBMS_LOB.	N/A
DBMS_LOB.INSTR	ORALOB.Matchpos	position	N/A
DBMS_LOB.SUBSTR	N/A	getBytes for BLOBs or BFILEs getSubString for CLOBs	N/A
DBMS_LOB.APPEND	ORALOB.Append	Use length and then putBytes or PutString	N/A
N/A [use PL/SQL assign operator]	ORALOB.Clone	N/A [use equal sign]	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

Table 6–3 (Cont.) Comparing the LOB Interfaces, 2 of 2

PL/SQL: DBMS_LOB (dbmslob.sql)	Visual Basic (OO4O)	Java (JDBC	OLEDB
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
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. DirectoryName ORABFILE. FileName	getDirAlias 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 operator)	DirectoryName 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. 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 and getBinaryStream CLOB: getString and getSubString and getCharacterStream	IRowset::GetData and ISequentialStream::Read
DBMS_LOB.TRIM	ORALOB.Trim	Use DBMS_LOB.	N/A

Table 6–3 (Cont.) Comparing the LOB Interfaces, 2 of 2

PL/SQL: DBMS_LOB (dbmslob.sql)	Visual Basic (OO4O)	Java (JDBC	OLEDB
DBMS_LOB.WRITE	ORALOB.Write	BLOB or BFILE: putBytes and getBinaryOutputStream CLOB: putString and getCharacterOutputStrea m	IRowsetChange::SetData and ISequentialStream::Write
DBMS_LOB.WRITEAPPEND	N/A	Use length and then putString or putBytes	N/A
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: *PL/SQL Packages and Types Reference* for detailed documentation, including parameters, parameter types, return values, and example code.

Provide a LOB Locator Before Running 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 call the routine.

- **Persistent 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, then specify the directory in the correct case. See "Directory Object" on page 15-4 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.

Guidelines for Offset and Amount Parameters in DBMS_LOB Operations

The following guidelines apply to offset and amount parameters used in procedures in the DBMS_LOB PL/SQL package:

- For character data—in all formats, fixed-width and varying-width—the amount and offset parameters are in characters. This applies to operations on CLOB and NCLOB datatypes.
- For binary data, the offset and amount parameters are in bytes. This applies to operations on BLOB datatypes.
- When using the following procedures:
  - DBMS_LOB.LOADFROMFILE
  - DBMS_LOB.LOADBLOBFROMFILE
  - DBMS_LOB.LOADCLOBFROMFILE

  you cannot specify an amount parameter with a value larger than the size of the BFILE you are loading from. To load the entire BFILE with these procedures, you must specify either the exact size of the BFILE, or the maximum allowable storage limit.

See Also:

- "Loading a LOB with Data from a BFILE" on page 14-17
- "Loading a BLOB with Data from a BFILE" on page 14-26
- "Loading a CLOB or NCLOB with Data from a BFILE" on page 14-29

- When using DBMS_LOB.READ, the `amount` parameter can be larger than the size of the data. The amount should be less than or equal to the size of the buffer. The buffer size is limited to 32K.

  See Also: "Reading Data from a LOB" on page 14-52

PL/SQL Functions and Procedures for LOBs

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

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

PL/SQL Functions/Procedures to Modify LOB Values

Table 6–4 PL/SQL: DBMS_LOB Procedures to Modify LOB 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 a persistent LOB
LOADCLOBFROMFILE()	Load character data from a file into a LOB
LOADBLOBFROMFILE()	Load binary data from a file into a LOB

Table 6–4 (Cont.) PL/SQL: DBMS_LOB Procedures to Modify LOB 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 and Procedures for Introspection of LOBs

Table 6–5 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 persistent 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 Operations on Temporary LOBs

Table 6–6 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 6–7 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 object name 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 6–8 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 "Opening Persistent LOBs with the OPEN and CLOSE Interfaces" on page 5-12.

Using OCI to Work with LOBs

Oracle Call Interface (OCI) LOB APIs enable you to access and make changes to LOBs and read data from BFILEs in C. OCI functions for LOBs are discussed in greater detail later in this section.

Setting the CSID Parameter for OCI LOB APIs

If you want to read or write data in 2 byte unicode (UCS2) format, then set the `csid` (character set ID) parameter in `OCILobRead2` and `OCILobWrite2` 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, then it will override the `NLS_LANG` environment variable.

> **See Also:**
>
> - *Oracle Call Interface Programmer's Guide* for information on the `OCIUnicodeToCharSet()` function and details on OCI syntax in general.
> - *Oracle Database Globalization Support Guide* for detailed information about implementing applications in different languages.

Fixed-Width and Varying-Width Character Set Rules for OCI

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

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:
  - `CLOB`s and `NCLOB`s: in characters
  - `BLOB`s and `BFILE`s: 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:
  - `CLOB`s and `NCLOB`s: in characters
  - `BLOB`s and `BFILE`s: in bytes
- **`OCILobRead2`**: 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'.

- **OCILobWrite2**: With client-side character set of varying-width, CLOBs and NCLOBs:
  - *Input amount* is in bytes. The input amount refers to the number of bytes of data in the input buffer 'bufp'.
  - *Output amount* is in characters. The output amount refers to the number of characters written into the server-side CLOB or NCLOB.

Other Operations

For all other LOB operations, irrespective of the client-side character set, the amount parameter is in characters for CLOBs and NCLOBs. These include OCILobCopy2(), OCILobErase2(), OCILobLoadFromFile2(), and OCILobTrim2(). All these operations refer to the amount of LOB data on the server.

See also: *Oracle Database Globalization Support Guide*

NCLOBs

NCLOB are allowed as parameters in methods.

OCILobLoadFromFile2() Amount Parameter

When using OCILobLoadFromFile2() you cannot specify amount larger than the length of the BFILE. To load the entire BFILE, you can pass the value of the system defined constant OCI_LOBMAXSIZE.

OCILobRead2() Amount Parameter

To read to the end of a LOB using OCILobRead2(), you specify an amount equal to the system defined constant OCI_LOBMAXSIZE. See "Streaming Read in OCI" on page 14-52 for more information.

OCILobLocator Pointer Assignment

Special care must be taken when assigning OCILobLocator pointers in an OCI program—using the "=" assignment operator. Pointer assignments create a shallow copy of the LOB. After the pointer assignment, the source and target LOBs point to the same copy of data.

These semantics are different from using LOB APIs, such as `OCILobAssign()` or `OCILobLocatorAssign()` to perform assignments. When the these APIs are used, the locators logically point to independent copies of data after assignment.

For temporary LOBs, before performing pointer assignments, you must ensure that any temporary LOB in the target LOB locator is freed by calling `OCIFreeTemporary()`. In contrast, when `OCILobLocatorAssign()` is used, the original temporary LOB in the target LOB locator variable, if any, is freed automatically before the assignment happens.

LOB Locators in Defines and Out-Bind Variables in OCI

Before you reuse a LOB locator in a define or an out-bind variable in a SQL statement, you must free any temporary LOB in the existing LOB locator buffer using `OCIFreeTemporary()`.

OCI LOB Examples

Further OCI examples are provided in:

- Chapter 12, "Operations Specific to Persistent and Temporary LOBs"
- Chapter 14, "LOB APIs for Basic Operations"
- Chapter 15, "LOB APIs for BFILE Operations"

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 persistent LOBs, see Table 6–9
- To read or examine LOB values, see Table 6–10
- To create or free temporary LOB, or check if Temporary LOB exists, see Table 6–11
- For read only functions on external LOBs (BFILEs), see Table 6–12

- To operate on LOB locators, see Table 6–13
- For LOB buffering, see Table 6–14
- To open and close LOBs, see Table 6–15

OCI Functions to Modify Persistent LOB (BLOB, CLOB, and NCLOB) Values

Table 6–9 OCI Functions to Modify Persistent LOB (BLOB, CLOB, and NCLOB) Values

Function/Procedure	Description
`OCILobAppend()`	Appends LOB value to another LOB.
`OCILobCopy2()`	Copies all or part of a LOB to another LOB.
`OCILobErase2()`	Erases part of a LOB, starting at a specified offset.
`OCILobLoadFromFile2()`	Loads `BFILE` data into a persistent LOB.
`OCILobTrim2()`	Truncates a LOB.
`OCILobWrite2()`	Writes data from a buffer into a LOB, overwriting existing data.
`OCILobWriteAppend2()`	Writes data from a buffer to the end of the LOB.

OCI Functions to Read or Examine Persistent LOB and External LOB (BFILE) Values

Table 6–10 OCI Functions to Read or Examine persistent LOB and external LOB (BFILE) Values

Function/Procedure	Description
`OCILobGetChunkSize()`	Gets the Chunk size used when reading and writing. This works on persistent LOBs and does not apply to external LOBs (BFILEs).
`OCILobGetLength2()`	Returns the length of a LOB or a `BFILE`.
`OCILobRead2()`	Reads a specified portion of a non-null LOB or a `BFILE` into a buffer.

OCI Functions for Temporary LOBs

Table 6–11 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 6–12 OCI Read-Only Functions for BFILES

Function/Procedure	Description
`OCILobFileClose()`	Closes an open `BFILE`.
`OCILobFileCloseAll()`	Closes all open `BFILE`s.
`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 6–13 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 6–14 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 6–15 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

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 set of C++ classes that enable a C++ program to connect to a database, run SQL statements, insert/update values in database tables, retrieve results of a query, run 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 computer as possible. Because 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 computer.

- 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, then the navigational interface performs better than the object interface.
- With respect to ODBC, the OCCI API is simpler to use. Because 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 persistent 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 persistent LOBs

OCCI Classes for LOBs

OCCI provides the following classes that allow you to use different types of LOB instances as objects in your C++ application:

- `Clob` class to access and modify data stored in internal CLOBs and NCLOBs
- `Blob` class to access and modify data stored in internal BLOBs
- `Bfile` class to access and read data stored in external LOBs (BFILEs)

> **See Also:** Syntax information on these classes and details on OCCI in general is available in the *Oracle C++ Call Interface Programmer's Guide*.

Clob Class

The Clob driver implements a CLOB object using an SQL LOB locator. 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 value, for materializing a CLOB value on the client, and getting a substring. Methods in the `ResultSet` and `Statement` interfaces such as `getClob()` and `setClob()` allow you to access SQL CLOB values.

> **See Also:** *Oracle C++ Call Interface Programmer's Guide* for detailed information on the Clob class.

Blob Class

Methods in the `ResultSet` and `Statement` interfaces, such as `getBlob()` and `setBlob()`, allow you to access SQL BLOB values. The `Blob` 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.

> **See Also:**
>
> - *Oracle C++ Call Interface Programmer's Guide* for detailed information on the Blob class methods and details on instantiating and initializing an Blob object in your C++ application.
> - *Oracle Database Globalization Support Guide* for detailed information about implementing applications in different languages.

Bfile Class

The Bfile class enables you to instantiate an Bfile object in your C++ application. You must then use methods of the Bfile class, such as the setName() method, to initialize the Bfile object which associates the object properties with an object of type BFILE in a BFILE column of the database.

> **See Also:** *Oracle C++ Call Interface Programmer's Guide* for detailed information on the Blob class methods and details on instantiating and initializing an Blob object in your C++ application.

Fixed Width Character Set Rules

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

- Clob: offset and amount parameters are always in characters
- Blob: offset and amount parameters are always in bytes
- Bfile: 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:
  - `Clob()`: in characters
  - `Blob()`: in bytes
  - Bfile(): in bytes
- **Amount parameter**: The amount parameter is always as follows:
  - `Clob`: in characters, when referring to a server-side LOB
  - `Blob`: in bytes, when referring to a client-side buffer
  - `Bfile`: 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:
  - `Clob.length()`: in characters
  - `Blob.length()`: in bytes
  - `Bfile.length()`: in bytes
- ***`Clob.read() and Blob.read()`***: With client-side character set of varying-width, `CLOB`s and `NCLOB`s:
  - ***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 OCCI buffer parameter, '`buffer`'.
- **`Clob.write() and Blob.write()`**: With client-side character set of varying-width, `CLOB`s and `NCLOB`s:
  - ***Input amount*** is in bytes. Input amount refers to the number of bytes of data in the OCCI 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 OCCI Operations

For all other OCCI LOB operations, irrespective of the client-side character set, the *amount parameter* is in characters for `CLOB`s and `NCLOB`s. These include the following:

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

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

See also: *Oracle Database Globalization Support Guide*

NCLOBs

- `NCLOB` instances are allowed as parameters in methods
- `NCLOB` instances are allowed as attributes in object types.

Amount Parameter for OCCI LOB copy() Methods

The `copy()` method on `Clob` and `Blob` enables you to load data from a `BFILE`. You can pass one of the following values for the `amount` parameter to this method:

- An amount smaller than the size of the `BFILE` to load a portion of the data
- An amount equal to the size of the `BFILE` to load all of the data
- The `UB4MAXVAL` constant to load all of the `BFILE` data

Note that you cannot specify an amount larger than the length of the `BFILE`.

Amount Parameter for OCCI read() Operations

The `read()` method on an `Clob`, `Blob`, or `Bfile` object, reads data from a `BFILE`. You can pass one of the following values for the amount parameter to specify the amount of data to read:

- An amount smaller than the size of the `BFILE` to load a portion of the data
- An amount equal to the size of the `BFILE` to load all of the data
- `0` (zero) to read until the end of the `BFILE` in streaming mode

Note that you cannot specify an amount larger than the length of the `BFILE`.

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, BLOBs, NCLOBs, and BFILEs

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

- To modify persistent LOBs, see Table 6–16
- To read or examine LOB values, see Table 6–17
- For read only methods on external LOBs (BFILEs), see Table 6–18
- Other LOB OCCI methods are described in Table 6–19
- To open and close LOBs, see Table 6–20

OCCI Methods to Modify Persistent LOB (BLOB, CLOB, and NCLOB) Values

Table 6–16 OCCI Clob and Blob Methods to Modify Persistent LOB (BLOB, CLOB, and NCLOB) Values

Function/Procedure	Description
`Blob/Clob.append()`	Appends C`LOB or BLOB` value to another LOB.
`Blob/Clob.copy()`	Copies all or part of a CLOB or BLOB to another LOB.
`Blob/Clob.copy()`	Loads `BFILE` data into a persistent LOB.
`Blob/Clob.trim()`	Truncates a C`LOB or BLOB`.
`Blob/Clob.write()`	Writes data from a buffer into a LOB, overwriting existing data.

OCCI Methods to Read or Examine Persistent LOB and BFILE Values

Table 6–17 OCCI Blob/Clob/Bfile Methods to Read or Examine persistent LOB and external LOB (BFILE) Values

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

OCCI Read-Only Methods for BFILEs

Table 6–18 OCCI Read-Only Methods for BFILES

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

Other OCCI LOB Methods

Table 6–19 Other OCCI LOB Methods

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

OCCI Methods to Open and Close Internal and External LOBs

Table 6–20 OCCI Methods to Open and Close Internal and External LOBs

Function/Procedure	Description
`Clob/Blob/Bfile.Open()`	Opens a LOB
`Clob/Blob/Bfile.isOpen()`	Sees if a LOB is open
`Clob/Blob/Bfile.Close()`	Closes a LOB

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

You can make changes to an entire persistent 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 persistent LOBs.

Embedded SQL statements allow you to access data stored in `BLOB`s, `CLOB`s, `NCLOB`s, and `BFILE`s. These statements are listed in the following tables, and are discussed in greater detail later in the chapter.

> **See Also:** *Pro*C/C++ 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 run 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

See Also: APIs for supported LOB operations are described in detail in:

- Chapter 12, "Operations Specific to Persistent and Temporary LOBs"
- Chapter 14, "LOB APIs for Basic Operations"
- Chapter 15, "LOB APIs for BFILE Operations"

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 persistent LOBs, see Table 6–21
- To read or examine LOB values, see Table 6–22
- To create or free temporary LOB, or check if Temporary LOB exists, see Table 6–23
- To operate close and 'see if file exists' functions on BFILEs, see Table 6–24
- To operate on LOB locators, see Table 6–25
- For LOB buffering, see Table 6–26
- To open or close LOBs or BFILEs, see Table 6–27

Pro*C/C++ Embedded SQL Statements to Modify Persistent LOB Values

Table 6–21 Pro*C/C++: Embedded SQL Statements to Modify Persistent 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 a persistent 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 for Introspection of LOBs

Table 6–22 Pro*C/C++: Embedded SQL Statements for Introspection of LOBs

Statement	Description
DESCRIBE [CHUNKSIZE]	Gets the Chunk size used when writing. This works for persistent 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 6–23 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 6–24 Pro*C/C++: Embedded SQL Statements for BFILES

Statement	Description
FILE CLOSE ALL	Closes all open `BFILE`s.
DESCRIBE [FILEEXISTS]	Checks whether a `BFILE` exists.
DESCRIBE [DIRECTORY,FILENAME]	Returns the directory object name and filename of a `BFILE`.

Pro*C/C++ Embedded SQL Statements for LOB Locators

Table 6–25 Pro*C/C++ Embedded SQL Statements for LOB Locators

Statement	Description
ASSIGN	Assigns one LOB locator to another.
FILE SET	Sets the directory object name and filename of a `BFILE` in a locator.

Pro*C/C++ Embedded SQL Statements for LOB Buffering

Table 6–26 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 LOBs

Table 6–27 Pro*C/C++ Embedded SQL Statements to Open and Close Persistent 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 persistent 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 persistent LOBs.

Embedded SQL statements allow you to access data stored in `BLOB`s, `CLOB`s, `NCLOB`s, and `BFILE`s. 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 run 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.

> **See Also:** APIs for supported LOB operations are described in detail in:
>
> - Chapter 12, "Operations Specific to Persistent and Temporary LOBs"
> - Chapter 14, "LOB APIs for Basic Operations"
> - Chapter 15, "LOB APIs for BFILE Operations"

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 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 persistent LOBs, see Table 6–28
- To read or examine internal and external LOB values, see Table 6–29
- To create or free temporary LOB, or check LOB locator, see Table 6–30
- To operate close and 'see if file exists' functions on BFILEs, see Table 6–31
- To operate on LOB locators, see Table 6–32
- For LOB buffering, see Table 6–33
- To open or close persistent LOBs or BFILEs, see Table 6–34

Pro*COBOL Embedded SQL Statements to Modify Persistent LOB Values

Table 6–28 Pro*COBOL Embedded SQL Statements to Modify BLOB, CLOB, and NCLOB Values

Statement	Description
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 a persistent 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 for Introspection of LOBs

Table 6–29 Pro*COBOL Embedded SQL Statements for Introspection of LOBs

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 6–30 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 6–31 Pro*COBOL Embedded SQL Statements for BFILES

Statement	Description
FILE CLOSE ALL	Closes all open `BFILE`s.
DESCRIBE [FILEEXISTS]	Checks whether a `BFILE` exists.
DESCRIBE [DIRECTORY, FILENAME]	Returns the directory object name and filename of a `BFILE`.

Pro*COBOL Embedded SQL Statements for LOB Locators

Table 6–32 Pro*COBOL Embedded SQL Statements for LOB Locators Statements

Statement	Description
ASSIGN	Assigns one LOB locator to another.
FILE SET	Sets the directory object name and filename of a `BFILE` in a locator.

Pro*COBOL Embedded SQL Statements for LOB Buffering

Table 6–33 Pro*COBOL 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*COBOL Embedded SQL Statements for Opening and Closing LOBs and BFILEs

Table 6–34 Pro*COBOL Embedded SQL Statements for Opening and CLosing Persistent LOBs and BFILEs

Statement	Description
OPEN	Opens a LOB or BFILE.
DESCRIBE [ISOPEN]	Sees if a LOB or BFILE is open.
CLOSE	Closes a LOB or BFILE.

Using Visual Basic (Oracle Objects for OLE (OO4O)) to Work with LOBs

Oracle Objects for OLE (OO4O) is a set 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 persistent 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: *Oracle*Blob and *Oracle*Clob 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 the database, has no manuals, only online help.

Its online help is available through the Application Development submenu of the database 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, then 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.

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

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

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

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

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

```
OraDyn.MoveNext

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

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

OraConnection.CommitTrans
```

In the 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.

See Also: APIs for supported LOB operations are described in detail in:

- Chapter 12, "Operations Specific to Persistent and Temporary LOBs"
- Chapter 14, "LOB APIs for Basic Operations"
- Chapter 15, "LOB APIs for BFILE Operations"

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 the database, has no manuals, only online help. The OO4O online help is available through the Application Development submenu of the database installation.

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

- To modify persistent LOBs, see Table 6–35
- To read or examine internal and external LOB values, see Table 6–36
- To open and close BFILEs, see Table 6–37
- For LOB buffering, see Table 6–38
- Properties such as to see if LOB is NULL, or to get or set polling amount, see Table 6–39
- For read-only BFILE methods, see Table 6–40
- For BFILE properties, see Table 6–41

OO4O Methods to Modify BLOB, CLOB, and NCLOB Values

Table 6–35 OO4O Methods to Modify BLOB, CLOB, and NCLOB Values

Methods	Description
`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.CopyFromBFile`	Loads `BFILE` data into an internal BLOB
`OraClob.CopyFromBFile`	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 6–36 OO4O Methods to Read or Examine Internal and External LOB Values

Function/Procedure	Description

`OraBlob.Read`
`OraClob.Read`
`OraBFile.Read`
	Reads a specified portion of a non-null BLOB into a buffer
	Reads a specified portion of a non-null CLOB into a buffer
	Reads a specified portion of a non-null BFILE into a buffer

`OraBlob.CopyToFile` `OraClob.CopyToFile`	Reads a specified portion of a non-null BLOB to a file Reads a specified portion of a non-null CLOB to a file

OO4O Methods to Open and Close External LOBs (BFILEs)

Table 6–37 OO4O Methods to Open and Close External LOBs (BFILEs)

Method	Description
`OraBFile.Open`	Opens BFILE.
`OraBFile.Close`	Closes BFILE.

OO4O Methods for Persistent LOB-Buffering

Table 6–38 OO4O Methods for Persistent 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 6–39 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 ■ ORALOB_NEED_DATA There is more data to be read or written ■ ORALOB_NO_DATA There is no more data to be read or written ■ ORALOB_SUCCESS_LOB data read/written successfully
`Size(Read)`	Returns the length of the LOB data

OO4O Read-Only Methods for External Lobs (BFILEs)

Table 6–40 OO4O Read-Only Methods for External LOBs (BFILEs)

Methods	Description
`OraBFile.Close`	Closes an open `BFILE`
`OraBFile.CloseAll`	Closes all open `BFILE`s
`OraBFile.Open`	Opens a `BFILE`
`OraBFile.IsOpen`	Determines if a BFILE is open

OO4O Properties for Operating on External LOBs (BFILEs)

Table 6–41 OO4O Properties for Operating on External LOBs (BFILEs)

Property	Description
`OraBFile.DirectoryName`	Gets/Sets the server side directory object 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 persistent LOB, or to pieces of the beginning, middle or end of a persistent 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 theses 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()`, then the retrieved locator still refers to the original LOB row.

To retrieve the locator for the most current row, you must call `getBLOB()`, `getCLOB()`, or `getBFILE()` on the `OracleResultSet` each time a move operation is made depending on whether the instance is a `BLOB`, `CLOB` or `BFILE`.

JDBC Syntax References and Further Information

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

See Also:

- *Oracle Database JDBC Developer's Guide and Reference* for detailed documentation, including parameters, parameter types, return values, and example code.
- 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 6–42
  - To read or examine BLOB values, see Table 6–43
  - For BLOB buffering, see Table 6–44
  - Temporary BLOBs: Creating, checking if LOB is open, and freeing. See Table 6–52
  - Opening, closing, and checking if BLOB is open, see Table 6–52
  - Trimming BLOBs, see Table 6–55
  - BLOB streaming API, see Table 6–57
- CLOBs:
  - To read or examine CLOB values, see Table 6–46
  - For CLOB buffering, see Table 6–47
  - To modify CLOBs, see Table 6–57
- Temporary CLOBs:
  - Opening, closing, and checking if CLOB is open, see Table 6–53

- Trimming CLOBs, see Table 6–56
- CLOB streaming API, see Table 6–58

- BFILEs:
  - To read or examine BFILEs, see Table 6–48
  - For BFILE buffering, see Table 6–49
  - Opening, closing, and checking if CLOB is open, see Table 6–54
  - BFILE streaming API, see Table 6–59

JDBC oracle.sql.BLOB Methods to Modify BLOB Values

Table 6–42 DBC oracle.sql.BLOB Methods To Modify BLOB Values

Method	Description
`int putBytes(long, byte[])`	Inserts the byte array into the BLOB, starting at the given offset

JDBC oracle.sql.BLOB Methods to Read or Examine BLOB Values

Table 6–43 DBC oracle.sql.BLOB Methods to Read or Examine BLOB Values

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

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

Table 6–44 JDBC oracle.sql.BLOB Methods and Properties for BLOB-Buffering

Method	Description
`public java.io.InputStream getBinaryStream())`	Streams the LOB as a binary stream
`public java.io.OutputStream getBinaryOutputStream()`	Writes to LOB as a binary stream

JDBC oracle.sql.CLOB Methods to Modify CLOB Values

Table 6–45 JDBC oracle.sql.CLOB Methods to Modify CLOB Values

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

JDBC oracle.sql.CLOB Methods to Read or Examine CLOB Value

Table 6–46 JDBC oracle.sql.CLOB Methods to Read or Examine CLOB Values

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

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

Table 6–47 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 6–48 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 file referenced by this BFILE exists
`public void openFile()`	Opens the operating system file referenced by this BFILE
`public void closeFile()`	Closes the operating system file referenced by this BFILE
`public boolean isFileOpen()`	Checks if this BFILE is already open
`public java.lang.String getDirAlias()`	Gets the directory object name 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 6–49 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 Temporary LOB APIs

Oracle Database JDBC drivers contain APIs to create and close temporary LOBs. These APIs can replace workarounds of using the following procedures from the `DBMS_LOB` PL/SQL package in prior releases:

- `DBMS_LOB.createTemporary()`
- `DBMS_LOB.isTemporary()`
- `DBMS_LOB.freeTemporary()`

Table 6–50 JDBC: Temporary BLOB APIs

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

Table 6–51 JDBC: Temporary CLOB APIs

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

JDBC: Opening and Closing LOBs

`oracle.sql.CLOB` class is the Oracle JDBC driver implementation of standard JDBC `java.sql.Clob` interface. Table 6–51 lists the new Oracle extension APIs in `oracle.sql.CLOB` for accessing temporary CLOBs.

Oracle Database 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 implementation of standard JDBC java.sql.Blob interface. Table 6–52 lists the Oracle extension APIs in `oracle.sql.BLOB` that open and close BLOBs. These are new for this release.

Table 6–52 JDBC: Opening and Closing BLOBs

Methods	Description
`public void open(int mode) throws SQLException`	Opens the BLOB
`public boolean isOpen() throws SQLException`	Sees if the BLOB is open
`public void close() throws SQLException`	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 Open

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 implementation of the standard JDBC `java.sql.Clob` interface. Table 6–53 lists the new Oracle extension APIs in `oracle.sql.CLOB` to open and close CLOBs.

Table 6–53 JDBC: Opening and Closing CLOBs

Methods	Description
`public void open(int mode) throws SQLException`	Open the CLOB
`public boolean isOpen() throws SQLExceptio`	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 6–54 lists the new Oracle extension APIs in `oracle.sql.BFILE` for opening and closing BFILEs.

Table 6–54 JDBC API Extensions for Opening and Closing BFILEs

Methods	Description
`public void open() throws SQLException`	Opens the BFILE
`public void open(int mode) throws SQLException`	Opens the BFILE
`public boolean isOpen() throws SQLException`	Checks if the BFILE is open
`public void close() throws SQLException`	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 is 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 lobs
    ResultSet rset = stmt.executeQuery ("select * from basic_lob_table");
    while (rset.next ())
    {
      // Get the lobs
```

```
        BLOB blob = (BLOB) rset.getObject (2);
        CLOB clob = (CLOB) rset.getObject (3);

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

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

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

        // Check if the lobs are 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

Oracle Database JDBC drivers contain APIs to trim persistent LOBs. These APIs replace previous techniques that used `DBMS_LOB.trim()`.

JDBC: Trimming BLOBs

`oracle.sql.BLOB` class is Oracle JDBC driver implementation of the standard JDBC `java.sql.Blob` interface. Table 6–55 lists the new Oracle extension API in `oracle.sql.BLOB` that trims BLOBs.

Table 6–55 JDBC: Trimming BLOBs

Methods	Description
`public void trim(long newlen) throws SQLException`	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 implementation of standard JDBC java.sql.Clob interface. Table 6–56 lists the new Oracle extension API in `oracle.sql.CLOB` that trims CLOBs.

Table 6–56 JDBC: Trimming CLOBs

Methods	Description
`public void trim(long newlen) throws SQLException`	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:** "Trimming LOB Data" on page 14-143, for an example.

JDBC BLOB Streaming APIs

The JDBC interface provided with the database includes LOB streaming APIs that enable you to read from or write to a LOB at the requested position from a Java stream.

The `oracle.sql.BLOB` class implements the standard JDBC `java.sql.Blob` interface. Table 6–57 lists Oracle extensions to the `oracle.sql.BLOB` API that manipulate BLOB content from the requested position.

Table 6–57 JDBC: New BLOB Streaming APIs

Methods	Description
`public java.io.OutputStream getBinaryOutputStream (long pos) throws SQLException`	Writes to the BLOB from a stream
`public java.io.InputStream getBinaryStream(long pos) throws SQLException`	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
```

JDBC CLOB Streaming APIs

The `oracle.sql.CLOB` class is the Oracle JDBC driver implementation of standard JDBC java.sql.Clob interface. Table 6–58 lists the new Oracle extension APIs in `oracle.sql.CLOB` that manipulate the CLOB content from the requested position.

Table 6–58 JDBC: New CLOB Streaming APIs

Methods	Description
`public java.io.OutputStream getAsciiOutputStream (long pos) throws SQLException`	Writes to the CLOB from an ASCII stream
`public java.io.Writer getCharacterOutputStream(long pos) throws SQLException`	Writes to the CLOB from a character stream
`public java.io.InputStream getAsciiStream(long pos) throws SQLException`	Reads from the CLOB as an ASCII stream
`public java.io.Reader getCharacterStream(long pos) throws SQLException`	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 6–59 lists the new Oracle extension APIs in `oracle.sql.BFILE` that reads BFILE content from the requested position.

Table 6–59 JDBC: New BFILE Streaming APIs

Methods	Description
`public java.io.InputStream getBinaryStream(long pos) throws SQLException`	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 is 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')");
```

```
  System.out.println ("Dumping lobs");

  // Select the lobs
  ResultSet rset = stmt.executeQuery ("select * from basic_lob_table");
  while (rset.next ())
  {
    // Get the lobs
    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 lobs again");

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

    // Print the lobs 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.

Oracle Provider for OLE DB (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: *Oracle Provider for OLE DB Developer's Guide*

Overview of Oracle Data Provider for .NET (ODP.NET)

Oracle Data Provider for .NET (ODP.NET) is an implementation of a data provider for the Oracle database. ODP.NET uses Oracle native APIs to offer fast and reliable access to Oracle data and features from any .NET application. ODP.NET also uses and inherits classes and interfaces available in the Microsoft .NET Framework Class Library. The ODP.NET supports the following LOBs as native datatypes with .NET: BLOB, CLOB, NCLOB, and BFILE.

See Also: *Oracle Data Provider for .NET Developer's Guide*

7

Performance Guidelines

This chapter discusses the following topics:

- LOB Performance Guidelines
- Moving Data to LOBs in a Threaded Environment

LOB Performance Guidelines

This section describes performance guidelines for applications that use LOB datatypes.

Performance Guidelines for Small Size LOBs

If most LOBs in your database tables are small in size—8K bytes or less—and only a few rows have LOBs larger than 8K bytes, then use the following guidelines to maximize database performance:

- Use ENABLE STORAGE IN ROW
- Set the DB_BLOCK_SIZE initialization parameter to 8K bytes and use a chunk size of 8K bytes
- See "LOB Storage" on page 4-7 information on tuning other parameters such as CACHE, PCTVERSION, and INITIAL and NEXT for the LOB segment.

General Performance Guidelines

Use the following guidelines to achieve maximum performance with LOBs:

- ***When Possible, Read/Write Large Data Chunks at a Time***: Because 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, then large reads/writes reduce network overhead.

  b. If using the '`NOCACHE`' option, then 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, then 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, then use LOB buffering — see `OCILobEnableBuffering`(), `OCILobDisableBuffering`(), `OCILobFlushBuffer`(), `OCILobWrite2`(), `OCILobRead2`(). Basically, turn on LOB buffering before reading/writing small pieces of LOB data.

  See Also: "LOB Buffering Subsystem" on page 5-2 for more information on LOB buffering.

- ***Use `OCILobRead2()` and `OCILobWrite2()` 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 `OCILobRead2`() and `OCILobWrite2`() with streaming.
- Commit changes frequently.

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, then 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.

- ***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.
- For optimal performance, temporary LOBs use ***reference on read, copy on write semantics***. When a temporary LOB locator is assigned to another locator, the physical LOB data is not copied. Subsequent READ operations using either of the LOB locators refer to the same physical LOB data. On the first WRITE operation after the assignment, the physical LOB data is copied in order to preserve LOB value semantics, that is, to ensure that each locator points to a unique LOB value. This performance consideration mainly applies to the PL/SQL and OCI environments.

  In PL/SQL, reference on read, copy on write semantics are illustrated as follows:

```
LOCATOR1 BLOB;
LOCATOR2 BLOB;
DBMS_LOB.CREATETEMPORARY (LOCATOR1,TRUE,DBMS_LOB.SESSION);

-- LOB data is not copied in this assignment operation:
LOCATOR2 := LOCATOR;
-- These read operations refer to the same physical LOB copy:
DBMS_LOB.READ(LOCATOR1, ...);
DBMS_LOB.GETLENGTH(LOCATOR2, ...);

-- A physical copy of the LOB data is made on WRITE:
DBMS_LOB.WRITE(LOCATOR2, ...);
```

  In OCI, to ensure value semantics of LOB locators and data, `OCILobLocatorAssign()` is used to copy temporary LOB locators as well as the LOB Data. `OCILobLocatorAssign()` does not make a round trip to the server. The physical temporary LOB copy is made when LOB updates happen in the same round trip as the LOB update API as illustrated in the following:

```
OCILobLocator *LOC1;
OCILobLocator *LOC2;
OCILobCreateTemporary(... LOC1, ... TRUE,OCI_DURATION_SESSION);

/* No round-trip is incurred in the following call. */
OCILobLocatorAssign(... LOC1, LOC2);

/* Read operations refer to the same physical LOB copy. */
```

```
OCILobRead2(... LOC1 ...)

/* One round-trip is incurred to make a new copy of the
 * LOB data and to write to the new LOB copy.
 */
OCILobWrite2(... LOC1 ...)

/* LOC2 does not see the same LOB data as LOC1. */
OCILobRead2(... LOC2 ...)
```

If LOB value semantics are not intended, then you can use C pointers to achieve reference semantics as illustrated in the following:

```
OCILobLocator *LOC1;
OCILobLocator *LOC2;
OCILobCreateTemporary(... LOC1, ... TRUE,OCI_DURATION_SESSION);

/* Pointer is copied. LOC1 and LOC2 refer to the same LOB data. */
LOC2 = LOC1;

/* Write to LOC2. */
OCILobWrite2(...LOC2...)

/* LOC1 will see the change made to LOC2. */
OCILobRead2(...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 OCI_OBJECT mode, the database 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, then 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 must 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.

Performance Considerations for SQL Semantics and LOBs

Be aware of the following performance issues when using SQL semantics with LOBs:

- Ensure that your temporary tablespace is large enough to accommodate LOBs stored out-of-line. Persistent LOBs that are greater than 4K bytes in size are stored outside of the LOB column.
- When possible, free unneeded temporary LOB instances. Unless you explicitly free a temporary LOB instance, the LOB remains in existence while your application is executing. More specifically, the instance exists while the scope in which the LOB was declared is executing.

> **See Also:** Chapter 9, "SQL Semantics and LOBs" for details on SQL semantics support for LOBs.

Moving Data to LOBs in a Threaded Environment

There two procedures that you can use to move data to LOBs in a threaded environment, one of which should be avoided.

Procedure to Avoid

The following sequence requires a new connection when using a threaded environment, adversely affects performance, and is not recommended:

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.

Recommended Procedure

Note:

- There is no need to create an empty LOB in this procedure.
- 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.

The recommended 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.

Part III

SQL Access to LOBs

This part describes SQL semantics for LOBs supported in the SQL and PL/SQL environments.

This part includes the following chapters:

- Chapter 8, "DDL and DML Statements with LOBs"
- Chapter 9, "SQL Semantics and LOBs"
- Chapter 10, "PL/SQL Semantics for LOBs"
- Chapter 11, "Migrating Table Columns from LONGs to LOBs"

8

DDL and DML Statements with LOBs

This chapter includes the following topics:

- Creating a Table Containing One or More LOB Columns
- Creating a Nested Table Containing a LOB
- Inserting a Row by Selecting a LOB From Another Table
- Inserting a LOB Value Into a Table
- Inserting a Row by Initializing a LOB Locator Bind Variable
- Updating a LOB with EMPTY_CLOB() or EMPTY_BLOB()
- Updating a Row by Selecting a LOB From Another Table

See Also: For guidelines on how to INSERT into a LOB when binds of more than 4,000 bytes are involved, see the following sections in "Binds of All Sizes in INSERT and UPDATE Operations" on page 13-8.

Creating a Table Containing One or More LOB Columns

This section describes how to create a table containing one or more LOB columns.

When you use functions, `EMPTY_BLOB()` and `EMPTY_CLOB()`, the resulting LOB is initialized, but not populated with data. Also note that LOBs that are empty are not null.

See Also:

Oracle Database 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 persistent LOB columns, and LOB attributes of embedded objects

Scenario

These examples use the following Sample Schemas:

- Human Resources (HR)
- Order Entry (OE)
- Product Media (PM)

Note that the HR and OE schemas must exist before the PM schema is created. For details on these schemas, refer to *Oracle Database Sample Schemas.*

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

```
/*  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_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 Nested Table Containing a LOB

This section describes how to create a nested table containing a LOB.

You must create the object type that contains the LOB attributes before you create a nested table based on that object type. In the example that follows, 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

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 as shown in the following example:

```
/* Create type textdoc_typ as the base type
   for the nested table textdoc_ntab,
   where textdoc_ntab contains a LOB:
*/
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_ntable (
   id number,
   ntab_col textdoc_ntab)
NESTED TABLE ntab_col STORE AS textdoc_nestedtab;

DROP TYPE textdoc_typ force;
DROP TYPE textdoc_ntab;
DROP TABLE textdoc_ntable;
```

See Also:

- "Creating a Table Containing One or More LOB Columns" on page 8-2
- *Oracle Database SQL Reference*, "Chapter 7, SQL Statements" — CREATE TABLE.

Inserting a Row by Selecting a LOB From Another Table

This section describes how to insert a row containing a LOB as SELECT.

Note: Persistent 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 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.

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.

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.

```
/* 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);
```

See Also:

- *Oracle Database SQL Reference*, "Chapter 7, SQL Statements" — INSERT.
- *Oracle Database Sample Schemas* for a description of the PM Schema and the `Print_media` table used in this example.

Inserting a LOB Value Into a Table

This section 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 a persistent 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 value by using the function `EMPTY_BLOB()` as a default predicate. Similarly, a `CLOB` or `NCLOB` column 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');
```

Note that you can also perform this initialization during the `CREATE TABLE` operation. See "Creating a Table Containing One or More LOB Columns" on page 8-2 for more information.

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 Initializing a LOB Locator Bind Variable

This section gives examples of how to insert a row by initializing a LOB locator bind variable.

Preconditions

Before you can insert a row using this technique, the following conditions must be met:

- The table containing the source row must exist.
- The destination table must exist.

For details on creating tables containing LOB columns, see Chapter 4, "LOBs in Tables".

Usage Notes

For guidelines on how to INSERT and UPDATE a row containing a LOB when binds of more than 4,000 bytes are involved, see "Binds of All Sizes in INSERT and UPDATE Operations" on page 13-8.

Syntax

See the following syntax references for details on using this operation in each programmatic environment:

- SQL: *Oracle Database SQL Reference*, "Chapter 7, SQL Statements" — INSERT
- C (OCI): *Oracle Call Interface Programmer's Guide* "Relational Functions" — LOB Functions.
- C++ (OCCI): *Oracle C++ Call Interface Programmer's Guide*
- COBOL (Pro*COBOL) *Pro*COBOL Programmer's Guide* for information on LOBs, usage notes on LOB Statements, and embedded SQL and precompiler directives — INSERT.
- C/C++ (Pro*C/C++): *Pro*C/C++ 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): *Oracle Database JDBC Developer's Guide and Reference* Chapter 7, "Working With LOBs" — Creating and Populating a BLOB or CLOB Column.

Examples

Examples for this use case are provided in the following programmatic environments:

- PL/SQL: Inserting a Row by Initializing a LOB Locator Bind Variable on page 8-9
- C (OCI): Inserting a Row by Initializing a LOB Locator Bind Variable on page 8-10

- C++ (OCCI): No example is provided with this release.

- COBOL (Pro*COBOL): Inserting a Row by Initializing a LOB Locator Bind Variable on page 8-11
- C/C++ (Pro*C/C++): Inserting a Row by Initializing a LOB Locator Bind Variable on page 8-12
- Visual Basic (OO4O): Inserting a Row by Initializing a LOB Locator Bind Variable on page 8-13
- Java (JDBC): Inserting a Row by Initializing a LOB Locator Bind Variable on page 8-14

PL/SQL: Inserting a Row by Initializing a LOB Locator Bind Variable

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/plsql/linsert.sql */

/* inserting a row through an insert statement */

CREATE OR REPLACE PROCEDURE insertLOB_proc (Lob_loc IN BLOB) IS
BEGIN
  /* Insert the BLOB into the row */
  DBMS_OUTPUT.PUT_LINE('------------ LOB INSERT EXAMPLE ------------');
  INSERT INTO print_media (product_id, ad_id, ad_photo)
        values (3106, 60315, Lob_loc);
END;
/
```

C (OCI): Inserting a Row by Initializing a LOB Locator Bind Variable

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/oci/linsert.c */

/* Insert the Locator into table using Bind Variables. */
#include <oratypes.h>
#include <lobdemo.h>
void insertLOB_proc(OCILobLocator *Lob_loc, OCIEnv *envhp,
                    OCIError *errhp, OCISvcCtx *svchp, OCIStmt *stmthp)
{
  int            product_id;
  OCIBind       *bndhp3;
  OCIBind       *bndhp2;
  OCIBind       *bndhp1;

  text          *insstmt =
   (text *) "INSERT INTO Print_media (product_id, ad_id, ad_sourcetext) \
            VALUES (:1, :2, :3)";

  printf ("----------- OCI Lob Insert Demo --------------\n");
  /* Insert the locator into the Print_media table with product_id=3060 */
  product_id = (int)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,
                                (void *) &product_id, (sb4) sizeof(product_id),
                                SQLT_INT, (void *) 0, (ub2 *)0, (ub2 *)0,
                                (ub4) 0, (ub4 *) 0, (ub4) OCI_DEFAULT));

  checkerr (errhp, OCIBindByPos(stmthp, &bndhp1, errhp, (ub4) 2,
                                (void *) &product_id, (sb4) sizeof(product_id),
                                SQLT_INT, (void *) 0, (ub2 *)0, (ub2 *)0,
                                (ub4) 0, (ub4 *) 0, (ub4) OCI_DEFAULT));

  checkerr (errhp, OCIBindByPos(stmthp, &bndhp2, errhp, (ub4) 3,
                                (void *) &Lob_loc, (sb4) 0,  SQLT_CLOB,
```

```
                               (void *) 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));

}
```

COBOL (Pro*COBOL): Inserting a Row by Initializing a LOB Locator Bind Variable

```
      * This file is installed in the following path when you install
      * the database: $ORACLE_HOME/rdbms/demo/lobs/procob/linsert.pco

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

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/proc/linsert.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 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 (OO4O): Inserting a Row by Initializing a LOB Locator Bind Variable

```
' This file is installed in the following path when you install
' the database: $ORACLE_HOME/rdbms/demo/lobs/vb/linsert.bas

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

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/java/linsert.java */

// 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();
    }
  }
}
```

Updating a LOB with EMPTY_CLOB() or EMPTY_BLOB()

This section describes how to UPDATE a LOB with `EMPTY_CLOB()` or `EMPTY_BLOB()`.

> **Note:** Performance improves when you update the LOB with the actual value, instead of using `EMPTY_CLOB()` or `EMPTY_BLOB()`.

Preconditions

Before you write data to a persistent 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 value by using the function `EMPTY_BLOB()` as a default predicate. Similarly, a `CLOB` or `NCLOB` column 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" on page 8-2) or, as in this case, by means of an `INSERT`.

The following example shows a series of updates using the `EMPTY_CLOB` operation to different data types.

```
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;
```

> **See Also:** SQL: *Oracle Database SQL Reference* Chapter 7, "SQL Statements" — UPDATE

Updating a Row by Selecting a LOB From Another Table

This section describes how to use the SQL `UPDATE AS SELECT` statement to update a row containing a LOB column by selecting a LOB from another table.

To use this technique, you must update by means of a reference. For example, the following code updates data from `online_media`:

```
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;
```

9

SQL Semantics and LOBs

This chapter describes SQL semantics that are supported for LOBs. These techniques allow you to use LOBs directly in SQL code and provide an alternative to using LOB-specific APIs for some operations.

This chapter covers the following topics:

- Using LOBs in SQL
- SQL Functions and Operators Supported for Use with LOBs
- Implicit Conversion of LOB Datatypes in SQL
- Unsupported Use of LOBs in SQL
- VARCHAR2 and RAW Semantics for LOBs

See Also: "Performance Considerations for SQL Semantics and LOBs" on page 7-6.

Using LOBs in SQL

You can access CLOB and NCLOB datatypes using SQL VARCHAR2 semantics, such as SQL string operators and functions. (LENGTH functions can be used with BLOB datatypes as well as CLOB and NCLOBs.) These techniques are beneficial in the following situations:

- When performing operations on LOBs that are relatively small in size (up to about 100K bytes).
- After migrating your database from LONG columns to LOB datatypes, any SQL string functions, contained in your existing PL/SQL application, will continue to work after the migration.

SQL semantics are not recommended in the following situations:

- When you need to use advanced features such as random access and piecewise fetch, you should use LOB APIs.
- When performing operations on LOBs that are relatively large in size (greater than 1MB) using SQL semantics can impact performance. Using the LOB APIs is recommended in this situation.

> **Note:** SQL semantics are used with persistent and temporary LOBs. (SQL semantics do not apply to BFILE columns as BFILE is a read-only datatype.)

SQL Functions and Operators Supported for Use with LOBs

Many SQL operators and functions that take VARCHAR2 columns as arguments also accept LOB columns. The following list summarizes which categories of SQL functions and operators are supported for use with LOBs. Details on individual functions and operators are given in Table 9–1.

The following categories of SQL functions and operators are supported for use with LOBs:

- Concatenation
- Comparison
  (Some comparison functions are not supported for use with LOBs.)
- Character functions

- Conversion
  (Some conversion functions are not supported for use with LOBs.)

The following categories of functions are not supported for use with LOBs:

- Aggregate functions
  Note that although pre-defined aggregate functions are not supported for use with LOBs, you can create user-defined aggregate functions to use with LOBs. See the *Oracle Data Cartridge Developer's Guide* for more information on user-defined aggregate functions.
- Unicode functions

Details on individual functions and operators are given in Table 9–1. This table lists SQL operators and functions that take VARCHAR2 types as operands or arguments, or return a VARCHAR2 value, and indicates in the "SQL" column which functions and operators are supported for CLOB and NCLOB datatypes. (The LENGTH function is also supported for the BLOB datatype.)

The DBMS_LOB PL/SQL package supplied with Oracle Database supports using LOBs with most of the functions listed in Table 9–1 as indicated in the "PL/SQL" column.

Note: Operators and functions with "No" indicated in the SQL column of Table 9–1 do not work in SQL queries used in PL/SQL blocks - even though some of these operators and functions are supported for use directly in PL/SQL code.

Implicit Conversion of CLOB to CHAR Types

Functions designated as "CNV" in the SQL or PL/SQL column of Table 9–1 are performed by converting the CLOB to a character datatype, such as VARCHAR2. In the SQL environment, only the first 4K bytes of the CLOB are converted and used in the operation; in the PL/SQL environment, only the first 32K bytes of the CLOB are converted and used in the operation.

Table 9–1 SQL VARCHAR2 Functions and Operators on LOBs

Category	Operator / Function	SQL Example / Comments	SQL	PL/SQL
Concatenation	||, CONCAT()	Select clobCol || clobCol2 from tab;	Yes	Yes
Comparison	= , !=, >, >=, <, <=, <>, ^=	if clobCol=clobCol2 then...	No	Yes
Comparison	IN, NOT IN	if clobCol NOT IN (clob1, clob2, clob3) then...	No	Yes
Comparison	SOME, ANY, ALL	if clobCol < SOME (select clobCol2 from...) then...	No	N/A
Comparison	BETWEEN	if clobCol BETWEEN clobCol2 and clobCol3 then...	No	Yes
Comparison	LIKE [ESCAPE]	if clobCol LIKE '%pattern%' then...	Yes	Yes
Comparison	IS [NOT] NULL	where clobCol IS NOT NULL	Yes	Yes
Character Functions	INITCAP, NLS_ INITCAP	select INITCAP(clobCol) from...	CNV	CNV
Character Functions	LOWER, NLS_LOWER, UPPER, NLS_UPPER	...where LOWER(clobCol1) = LOWER(clobCol2)	Yes	Yes
Character Functions	LPAD, RPAD	select RPAD(clobCol, 20, ' La') from...	Yes	Yes
Character Functions	TRIM, LTRIM, RTRIM	...where RTRIM(LTRIM(clobCol,'ab'), 'xy') = 'cd'	Yes	Yes
Character Functions	REPLACE	select REPLACE(clobCol, 'orig','new') from...	Yes	Yes
Character Functions	SOUNDEX	...where SOUNDEX(clobCOl) = SOUNDEX('SMYTHE')	CNV	CNV
Character Functions	SUBSTR	...where substr(clobCol, 1,4) = 'THIS'	Yes	Yes
Character Functions	TRANSLATE	select TRANSLATE(clobCol, '123abc','NC') from...	CNV	CNV
Character Functions	ASCII	select ASCII(clobCol) from...	CNV	CNV
Character Functions	INSTR	...where instr(clobCol, 'book') = 11	Yes	Yes
Character Functions	LENGTH	...where length(clobCol) != 7;	Yes	Yes

Table 9–1 (Cont.) SQL VARCHAR2 Functions and Operators on LOBs

Category	Operator / Function	SQL Example / Comments	SQL	PL/SQL
Character Functions	NLSSORT	...where NLSSORT (clobCol,'NLS_SORT = German') > NLSSORT ('S','NLS_SORT = German')	CNV	CNV
Character Functions	INSTRB, SUBSTRB, LENGTHB	These functions are supported only for CLOBs that use single-byte character sets. (LENGTHB is supported for BLOBs as well as CLOBs.)	Yes	Yes
Character Functions - Regular Expressions	REGEXP_LIKE	This function searches a character column for a pattern. Use this function in the WHERE clause of a query to return rows matching the regular expression you specify. See the *Oracle Database SQL Reference* for syntax details on SQL functions for regular expressions. See the *Oracle Database Application Developer's Guide - Fundamentals* for information on using regular expressions with the database.	Yes	Yes
Character Functions - Regular Expressions	REGEXP_REPLACE	This function searches for a pattern in a character column and replaces each occurrence of that pattern with the pattern you specify.	Yes	Yes
Character Functions - Regular Expressions	REGEXP_INSTR	This function searches a string for a given occurrence of a regular expression pattern. You specify which occurrence you want to find and the start position to search from. This function returns an integer indicating the position in the string where the match is found.	Yes	Yes
Character Functions - Regular Expressions	REGEXP_SUBSTR	This function returns the actual substring matching the regular expression pattern you specify.	Yes	Yes
Conversion	CHARTOROWID	CHARTOROWID(clobCol)	CNV	CNV
Conversion	HEXTORAW	HEXTORAW(CLOB)	No	CNV
Conversion	CONVERT	select CONVERT(clobCol,'WE8DEC','WE8HP') from...	Yes	CNV
Conversion	TO_DATE	TO_DATE(clobCol)	CNV	CNV
Conversion	TO_NUMBER	TO_NUMBER(clobCol)	CNV	CNV
Conversion	TO_TIMESTAMP	TO_TIMESTAMP(clobCol)	No	CNV

Table 9–1 (Cont.) SQL VARCHAR2 Functions and Operators on LOBs

Category	Operator / Function	SQL Example / Comments	SQL	PL/SQL
Conversion	TO_MULTI_BYTE TO_SINGLE_BYTE	TO_MULTI_BYTE(clobCol) TO_SINGLE_BYTE(clobCol)	CNV	CNV
Conversion	TO_CHAR	TO_CHAR(clobCol)	Yes	Yes
Conversion	TO_NCHAR	TO_NCHAR(clobCol)	Yes	Yes
Conversion	TO_LOB	INSERT INTO... SELECT TO_LOB(longCol)... Note that 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.	N/A	N/A
Conversion	TO_CLOB	TO_CLOB(varchar2Col)	Yes	Yes
Conversion	TO_NCLOB	TO_NCLOB(varchar2Clob)	Yes	Yes
Aggregate Functions	COUNT	select count(clobCol) from...	No	N/A
Aggregate Functions	MAX, MIN	select MAX(clobCol) from...	No	N/A
Aggregate Functions	GROUPING	select grouping(clobCol) from... group by cube (clobCol);	No	N/A
Other Functions	GREATEST, LEAST	select GREATEST (clobCol1, clobCol2) from...	No	CNV
Other Functions	DECODE	select DECODE(clobCol, condition1, value1, defaultValue) from...	CNV	CNV
Other Functions	NVL	select NVL(clobCol,'NULL') from...	Yes	Yes
Other Functions	DUMP	select DUMP(clobCol) from...	No	N/A
Other Functions	VSIZE	select VSIZE(clobCol) from...	No	N/A
Unicode	INSTR2, SUBSTR2, LENGTH2, LIKE2	These functions use UCS2 code point semantics.	No	CNV
Unicode	INSTR4, SUBSTR4, LENGTH4, LIKE4	These functions use UCS4 code point semantics.	No	CNV
Unicode	INSTRC, SUBSTRC, LENGTHC, LIKEC	These functions use complete character semantics.	No	CNV

UNICODE Support

Variations on the INSTR, SUBSTR, LENGTH, and LIKE functions are provided for Unicode support. (These variations are indicated as "Unicode" in the "Category" column of Table 9–1.)

See Also:

- *PL/SQL Packages and Types Reference*
- *Oracle Database Application Developer's Guide - Fundamentals*
- *Oracle Database SQL Reference*
- *Oracle Database Globalization Support Guide*

for a detailed description on the usage of UNICODE functions.

Codepoint Semantics

Codepoint semantics of the INSTR, SUBSTR, LENGTH, and LIKE functions, described in Table 9–1, differ depending on the datatype of the argument passed to the function. These functions use different codepoint semantics depending on whether the argument is a VARCHAR2 or a CLOB type as follows:

- When the argument is a CLOB, UCS2 codepoint semantics are used for all character sets.
- When the argument is a character type, such as VARCHAR2, the default codepoint semantics are used for the given character set:
  - UCS2 codepoint semantics are used for AL16UTF16 and UTF8 character sets.
  - UCS4 codepoint semantics are used for all other character sets, such as AL32UTF8.
- If you are storing character data in a CLOB, then note that the amount and offset parameters for any APIs that read or write data to the CLOB are specified in codepoints. In some character sets, a full character consists one or more codepoints called surrogate pair. In this scenario, you must ensure that the amount or offset you specify does not cut into a full character. This avoids reading or writing a partial character.

Return Values for SQL Semantics on LOBs

The return type of a function or operator that takes a LOB or VARCHAR2 is the same as the datatype of the argument passed to the function or operator.

Functions that take more than one argument, such as CONCAT, return a LOB datatype if one or more arguments is a LOB. For example, CONCAT(CLOB, VARCHAR2) returns a CLOB.

> **See Also:** *Oracle Database SQL Reference* for details on the CONCAT function and the concatenation operator (||).

A LOB instance is always accessed and manipulated through a LOB locator. This is also true for return values: SQL functions and operators return a LOB locator when the return value is a LOB instance.

Any LOB instance returned by a SQL function is a temporary LOB instance. LOB instances in tables (persistent LOBs) are not modified by SQL functions, even when the function is used in the SELECT list of a query.

LENGTH Return Value for LOBs

The return value of the LENGTH function differs depending on whether the argument passed is a LOB or a character string:

- If the input is a character string of length zero, then LENGTH returns NULL.
- For a CLOB of length zero, or an empty locator such as that returned by EMPTY_CLOB(), the LENGTH and DBMS_LOB.GETLENGTH functions return FALSE.

Implicit Conversion of LOB Datatypes in SQL

Some LOB datatypes support implicit conversion and can be used in operations such as cross-type assignment and parameter passing. These conversions are processed at the SQL layer and can be performed in all client interfaces that use LOB types.

Implicit Conversion Between CLOB and NCLOB Datatypes in SQL

The database enables you to perform operations such as cross-type assignment and cross-type parameter passing between CLOB and NCLOB datatypes. The database performs implicit conversions between these types when necessary to preserve properties such as characterset formatting.

Note that, when implicit conversions occur, each character in the source LOB is changed to the character set of the destination LOB, if needed. In this situation, some degradation of performance may occur if the data size is large. When the character set of the destination and the source are the same, there is no degradation of performance.

After an implicit conversion between CLOB and NCLOB types, the destination LOB is implicitly created as a temporary LOB. This new temporary LOB is independent from the source LOB. If the implicit conversion occurs as part of a define operation in a SELECT statement, then any modifications to the destination LOB do not affect the persistent LOB in the table that the LOB was selected from as shown in the following example:

```
SQL> -- check lob length before update
SQL> select dbms_lob.getlength(ad_sourcetext) from Print_media
  2      where product_id=3106 and ad_id = 13001;

DBMS_LOB.GETLENGTH(AD_SOURCETEXT)
---------------------------------
         205

SQL>
SQL> declare
  2   clob1 clob;
  3   amt number:=10;
  4  begin
  5    -- select a clob column into a clob, no implicit convesion
  6    SELECT ad_sourcetext INTO clob1 FROM Print_media
  7      WHERE product_id=3106 and ad_id=13001 FOR UPDATE;
  8
  9    dbms_lob.trim(clob1, amt); -- Trim the selected lob to 10 bytes
 10  end;
 11  /

PL/SQL procedure successfully completed.

SQL> -- Modification is performed on clob1 which points to the
SQL> -- clob column in the table
SQL> select dbms_lob.getlength(ad_sourcetext) from Print_media
  2      where product_id=3106 and ad_id = 13001;

DBMS_LOB.GETLENGTH(AD_SOURCETEXT)
```

```
---------------------------------
         10

SQL>
SQL> rollback;

Rollback complete.

SQL> -- check lob length before update
SQL> select dbms_lob.getlength(ad_sourcetext) from Print_media
  2       where product_id=3106 and ad_id = 13001;

DBMS_LOB.GETLENGTH(AD_SOURCETEXT)
---------------------------------
        205

SQL>
SQL> declare
  2   nclob1 nclob;
  3   amt number:=10;
  4  begin
  5
  6    -- select a clob column into a nclob, implicit convesion occurs
  7    SELECT ad_sourcetext INTO nclob1 FROM Print_media
  8      WHERE product_id=3106 and ad_id=13001 FOR UPDATE;
  9
 10    dbms_lob.trim(nclob1, amt); -- Trim the selected lob to 10 bytes
 11  end;
 12  /

PL/SQL procedure successfully completed.

SQL> -- Modification to nclob1 does not affect the clob in the table,
SQL> -- because nclob1 is a independent temporary LOB

SQL> select dbms_lob.getlength(ad_sourcetext) from Print_media
  2       where product_id=3106 and ad_id = 13001;

DBMS_LOB.GETLENGTH(AD_SOURCETEXT)
---------------------------------
        205
```

See Also:

- "Implicit Conversions Between CLOB and VARCHAR2" on page 10-2 for information on PL/SQL semantics support for implicit conversions between CLOB and VARCHAR2 types.
- "Implicit Character Set Conversions with LOBs" on page 4-6 for more information on implicit character set conversions when loading LOBs from BILEs.
- *Oracle Database SQL Reference* for details on implicit conversions supported for all datatypes.

Unsupported Use of LOBs in SQL

Table 9–2 lists SQL operations that are not supported on LOB columns.

Table 9–2 Unsupported usage of LOBs in SQL

SQL Operations Not Supported	Example of unsupported usage
SELECT DISTINCT	SELECT DISTINCT clobCol from...
SELECT clause ORDER BY	SELECT... ORDER BY clobCol
SELECT clause GROUP BY	SELECT avg(num) FROM... GROUP BY clobCol
UNION, INTERSECT, MINUS (Note that UNION ALL works for LOBs.)	SELECT clobCol1 from tab1 UNION SELECT clobCol2 from tab2;
Join queries	SELECT... FROM... WHERE tab1.clobCol = tab2.clobCol
Index columns	CREATE INDEX clobIndx ON tab(clobCol)...

VARCHAR2 and RAW Semantics for LOBs

The following semantics, used with VARCHAR2 and RAW datatypes, also apply to LOBs:

- Defining a CHAR buffer on a CLOB

  You can define a VARCHAR2 for a CLOB and RAW for a BLOB column. You can also define CLOB and BLOB types for VARCHAR2 and RAW columns.

- Selecting a CLOB column into a CHAR buffer or VARCHAR2

  If a CLOB column is selected into a VARCHAR2 variable, then 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, then a truncation error is thrown and no data is written to the buffer. After successful completion of the SELECT operation, the VARCHAR2 variable holds as a regular character buffer.

  In contrast, when a CLOB column is selected into a local CLOB variable, the CLOB locator is fetched.

- Selecting a BLOB column into a RAW

  When a BLOB column is selected into a RAW variable, the BLOB data is copied into the RAW buffer. If the size of the BLOB exceeds the size of the buffer, then a truncation error is thrown and no data is written to the buffer.

LOBs Returned from SQL Functions

When a LOB is returned from a SQL function, the result returned is a temporary LOB. Your application should view the temporary LOB as local storage for the data returned from the SELECT operation as follows:

- 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 program block at which time, the LOB is freed. These are the same semantics as those for PL/SQL VARCHAR2 variables. At any time, nonetheless, you can use 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).**

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 NULL and IS [NOT] NULL Usage with VARCHAR2s and CLOBs

You can use the IS NULL and IS [NOT] NULL operators with LOB columns. When used with LOBs, these operators determine whether a LOB locator is stored in the row.

Note: In the SQL 92 standard, a character string of length zero is distinct from a null string. The return value of IS NULL differs when you pass a LOB compared to a VARCHAR2:

- When you pass an initialized LOB of length zero to the IS NULL function, zero (FALSE) is returned. These semantics are compliant with the SQL standard.
- When you pass a VARCHAR2 of length zero to the IS NULL function, TRUE is returned.

WHERE Clause Usage with LOBs

SQL functions with LOBs as arguments, except functions that compare LOB values, are allowed in predicates of the WHERE clause. For example, the LENGTH function can be included in the predicate of the WHERE clause:

```
create table t (n number, c clob);
insert into t values (1, 'abc');

select * from t where c is not null;
select * from t where length(c) > 0;
select * from t where c like '%a%';
select * from t where substr(c, 1, 2) like '%b%';
select * from t where instr(c, 'b') = 2;
```

10

PL/SQL Semantics for LOBs

This chapter covers the following topics:

- PL/SQL Statements and Variables
- Implicit Conversions Between CLOB and VARCHAR2
- Explicit Conversion Functions
- PL/SQL CLOB Comparison Rules

PL/SQL Statements and Variables

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.

PL/SQL semantics support is described in the following sections:

- Implicit Conversions Between CLOB and VARCHAR2
- Explicit Conversion Functions
- VARCHAR2 and CLOB in PL/SQL Built-In Functions
- PL/SQL CLOB Comparison Rules

Implicit Conversions Between CLOB and VARCHAR2

Implicit conversions from CLOB to VARCHAR2 and from VARCHAR2 to CLOB datatypes are allowed in PL/SQL. These conversions enable you to perform the following operations in your application:

- 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

Accessing a CLOB as a VARCHAR2 in PL/SQL

The following example illustrates the way CLOB data is accessed when the CLOBs are treated as VARCHAR2s:

```
declare
   myStoryBuf VARCHAR2(4001);
begin
   SELECT AD_SOURCETEXT INTO myStoryBuf FROM PRINT_MEDIA WHERE AD_ID = 12001;
   -- Display Story by printing myStoryBuf directly
end;
/
```

Assigning a CLOB to a VARCHAR2 in PL/SQL

```
declare
myLOB CLOB;
begin
SELECT 'ABCDE' INTO myLOB FROM PRINT_MEDIA WHERE AD_ID = 11001;
-- myLOB is a temporary LOB.
-- Use myLOB as a lob locator
  DBMS_OUTPUT.PUT_LINE('Is temp? '||DBMS_LOB.ISTEMPORARY(myLOB));
end;
/
```

Explicit Conversion Functions

In SQL and PL/SQL, the following explicit conversion functions 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, such as, TO_NUMBER(), see Table 9–1. Conversion function details are explained in Chapter 11, "Migrating Table Columns 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, used 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,"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 VARCHAR2 semantics.

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 print_media WHERE product_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 because myStory is
15 -- longer than 100 bytes
16 myGist := myStory;
17 end;
```

Please note that in line 10 of "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.

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 print_media WHERE product_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 print_media set Story = myStory WHERE product_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 `print_media`.

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` is now used 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.

Freeing Temporary LOBs Automatically and Manually

```
declare
   Story1 CLOB;
   Story2 CLOB;
```

```
   StoryCombined CLOB;
   StoryLower CLOB;
begin
   SELECT Story INTO Story1 FROM print_media WHERE product_ID = 1;
   SELECT Story INTO Story2 FROM print_media WHERE product_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.

11

Migrating Table Columns from LONGs to LOBs

This chapter describes techniques for migrating tables that use LONG datatypes to LOB datatypes. This chapter covers the following topics:

- Benefits of Migrating LONG Columns to LOB Columns
- Preconditions for Migrating LONG Columns to LOB Columns
- Using utldtree.sql to Determine Where Your Application Needs Change
- Converting Tables from LONG to LOB Datatypes
- Migrating Applications from LONGs to LOBs

> **See Also:** The following chapters in this book describe support for LOB datatypes in respective environments:
>
> - Chapter 9, "SQL Semantics and LOBs"
> - Chapter 10, "PL/SQL Semantics for LOBs"
> - Chapter 13, "Data Interface for Persistent LOBs" for PL/SQL and OCI APIs that support LOB datatypes.

Benefits of Migrating LONG Columns to LOB Columns

There are many benefits to migrating table columns from LONG datatypes to LOB datatypes.

> **Note:** You can use the techniques described in this chapter to do either of the following:
>
> - Convert columns of type `LONG` to either `CLOB` or `NCLOB` columns
> - Convert columns of type `LONG RAW` to `BLOB` type columns
>
> Unless otherwise noted, discussions in this chapter regarding "LONG to LOB" conversions apply to both of these datatype conversions.

The following list compares the semantics of LONG and LOB datatypes in various application development scenarios:

- The number of LONG type columns is limited. Any given table can have a maximum of only one LONG type column. The number of LOB type columns in a table is not limited.
- You can use the data interface for LOBs to enable replication of tables that contain LONG or LONG RAW columns. Replication is allowed on LOB columns, but is not supported for LONG and LONG RAW columns. The database omits columns containing LONG and LONG RAW datatypes from replicated tables.

  Note that, if a table is replicated or has materialized views, and its LONG column is changed to LOB, then you may have to manually fix the replicas.

> **Caution:** **Converting LOB datatypes back to LONG datatypes is not supported. Ensure that you do not need to maintain any column as a LONG datatype before converting the column to a LOB type.**

Preconditions for Migrating LONG Columns to LOB Columns

This section describes preconditions that must be met before converting a LONG column to a LOB column.

See Also: "Migrating Applications from LONGs to LOBs" on page 11-10 before converting your table to determine whether any limitations on LOB columns will prevent you from converting to LOBs.

Dropping a Domain Index on a LONG Column Before Converting to a LOB

Any domain index on a LONG column must be dropped before converting the LONG column to LOB column. See "Indexes on Columns Converted from LONG to LOB Datatypes" on page 11-11 for more information.

Preventing Generation of Redo Space on Tables Converted to LOB Datatypes

Generation of redo space can cause performance problems during the process of converting LONG columns. Redo changes for the table are logged during the conversion process 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 column indicate LOGGING. The logging setting (LOGGING or NOLOGGING) for the LOB column is inherited from the tablespace in which the LOB is created.

To prevent generation of redo space during migration, do the following before migrating your table:

1. `ALTER TABLE Long_tab NOLOGGING;`
2. `ALTER TABLE Long_tab MODIFY (long_col CLOB [default <default_val>]) LOB (long_col) STORE AS (NOCACHE NOLOGGING);`

   Note that you must also specify `NOCACHE` when you specify `NOLOGGING` in the `STORE AS` clause.
3. `ALTER TABLE Long_tab MODIFY LOB (long_col) (CACHE);`
4. `ALTER TABLE Long_tab LOGGING;`
5. Make a backup of the tablespaces containing the table and the LOB column.

Using utldtree.sql to Determine Where Your Application Needs Change

You can use the utility, `rdbms/admin/utldtree.sql`, to determine which parts of your application require rewriting when you migrate your table from LONG to LOB column types. This utility enables you to recursively see all objects that are dependent on a given object. For example, you can see all objects which depend on a table with a LONG column. Note that, you will only see objects for which you have permission.

Instructions on how to use `utldtree.sql` are documented in the file itself. Also, `utldtree.sql` is only needed for PL/SQL. For SQL and OCI you do not need to change your applications.

Converting Tables from LONG to LOB Datatypes

This section describes the following techniques for migrating existing tables from LONG to LOB datatypes:

- Using ALTER TABLE to Convert LONG Columns to LOB Columns on page 11-4
- Copying a LONG to a LOB Column Using the TO_LOB Operator on page 11-5
- Online Redefinition of Tables with LONG Columns on page 11-6 where high availability is critical

Using ALTER TABLE to Convert LONG Columns to LOB Columns

You can use the ALTER TABLE statement in SQL to convert a LONG column to a LOB column. To do so, use the following syntax:

```
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 that was created as follows:

```
CREATE TABLE Long_tab (id NUMBER, long_col LONG);
```

then you can change the column `long_col` in table `Long_tab` to datatype `CLOB` using following ALTER TABLE statement:

```
ALTER TABLE Long_tab MODIFY ( long_col CLOB );
```

Note: The `ALTER TABLE` 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.

Note that when using the ALTER TABLE statement to convert a LONG column to a LOB column, only the following options are allowed:

- DEFAULT which enables you to specify a default value for the LOB column.
- The *LOB_storage_clause*, which enables you to specify the LOB storage characteristics for the converted column, can be specified in the MODIFY clause.

Other ALTER TABLE options are not allowed when converting a LONG column to a LOB type column.

Migration Issues

General issues concerning migration include the following:

- 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.
- If you do not specify a default value, then the default value for the LONG column becomes the default value of the LOB column.
- Most of the existing triggers on your table are still usable, however UPDATE OF triggers can cause issues. See "Migrating Applications from LONGs to LOBs" on page 11-10 for more details.

Copying a LONG to a LOB Column Using the TO_LOB Operator

If you do not want to use ALTER TABLE, as described earlier in this section, then you can use the TO_LOB operator on a LONG column to copy it to a LOB column. You can use the CREATE TABLE AS SELECT statement or the INSERT AS SELECT statement with the TO_LOB operator to copy data from a LONG column to a CLOB or NCLOB column, or from a LONG RAW column to a BLOB column. For example, if you have a table with a LONG column that was created as follows:

```
CREATE TABLE Long_tab (id NUMBER, long_col LONG);
```

then you can do the following to copy the column to a LOB column:

```
CREATE TABLE Lob_tab (id NUMBER, clob_col CLOB);
INSERT INTO Lob_tab SELECT id, TO_LOB(long_col) FROM long_tab;
COMMIT;
```

If the INSERT returns an error (because of lack of undo space), then you can incrementally migrate LONG data to the LOB column using the WHERE clause. After you ensure that the data is accurately copied, you can drop the original table and create a view or synonym for the new table using one of the following sequences:

```
DROP TABLE Long_tab;
CREATE VIEW Long_tab (id, long_col) AS SELECT * from Lob_tab;
```

or

```
DROP TABLE Long_tab;
CREATE SYNONYM Long_tab FOR 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 technique, you have to re-create any constraints, triggers, grants and indexes on the new table.

Use of the `TO_LOB` operator is subject to the following limitations:

- You can use TO_LOB to copy data to a LOB column, but not to a LOB attribute of an object type.
- You cannot use TO_LOB with a remote table. For example, the following statements will not work:

  ```
  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.

  To work around this limitation, create the index organized table, and then do an INSERT AS SELECT of the LONG or LONG RAW column using the TO_ LOB operator.
- You cannot use TO_LOB inside any PL/SQL block.

Online Redefinition of Tables with LONG Columns

Tables with LONG and LONG RAW columns can be migrated using online table redefinition. This technique is suitable for migrating LONG columns in database tables where high availability is critical.

To use this technique, you must convert LONG columns to LOB types during the redefinition process as follows:

- Any LONG column must be converted to a CLOB or NCLOB column
- Any LONG RAW column must be converted to a BLOB column

This conversion is performed using the `TO_LOB()` operator in the column mapping of the `DBMS_REDEFINITION.START_REDEF_TABLE()` procedure.

> **Note:** You cannot perform online redefinition of tables with LONG or LONG RAW columns unless you convert the columns to LOB types as described in this section.

General tasks involved in the online redefinition process are given in the following list. Issues specific to converting LONG and LONG RAW columns are called out. See the related documentation referenced at the end of this section for additional details on the online redefinition process that are not described here.

- Create an empty interim table. This table will hold the migrated data when the redefinition process is done. In the interim table:
  - Define a CLOB or NCLOB column for each LONG column in the original table that you are migrating.
  - Define a BLOB column for each LONG RAW column in the original table that you are migrating.
- Start the redefinition process. To do so, call `DBMS_REDEFINITION.START_REDEF_TABLE()` and pass the column mapping using the `TO_LOB()` operator as follows:

```
DBMS_REDEFINITION.START_REDEF_TABLE(
    'schema_name',
    'original_table',
    'interim_table',
    'TO_LOB(long_col_name) lob_col_name',
    'options_flag',
    'orderby_cols');
```

  where *`long_col_name`* is the name of the LONG or LONG RAW column that you are converting in the original table and *`lob_col_name`* is the name of the LOB column in the interim table. This LOB column will hold the converted data.

- Call the `DBMS_REDEFINITION.COPY_TABLE_DEPENDENTS()` procedure as described in the related documentation.
- Call the `DBMS_REDEFINITION.FINISH_REDEF_TABLE()` procedure as described in the related documentation.

The following example demonstrates online redefinition with LOB columns.

```
REM Grant privleges required for online redefinition.
GRANT execute ON DBMS_REDEFINITION TO pm;
GRANT ALTER ANY TABLE TO pm;
GRANT DROP ANY TABLE TO pm;
GRANT LOCK ANY TABLE TO pm;
GRANT CREATE ANY TABLE TO pm;
GRANT SELECT ANY TABLE TO pm;

REM Privileges required to perform cloning of dependent objects.
GRANT CREATE ANY TRIGGER TO pm;
GRANT CREATE ANY INDEX TO pm;

connect pm/pm

drop table cust;
create table cust(c_id   number primary key,
                  c_zip  number,
                  c_name varchar(30) default null,
                  c_long long
                  );
insert into cust values(1, 94065, 'hhh', 'ttt');

-- Creating Interim Table
-- There is no need to specify constraints because they are
-- copied over from the original table.
create table cust_int(c_id   number not null,
                  c_zip  number,
                  c_name varchar(30) default null,
                  c_long clob
                  );

declare
 col_mapping varchar2(1000);
begin
--  map all the columns in the interim table to the original table
 col_mapping :=
               'c_id             c_id  , '||
               'c_zip            c_zip , '||
```

```
               'c_name           c_name, '||
               'to_lob(c_long)   c_long';

 dbms_redefinition.start_redef_table('pm', 'cust', 'cust_int',
                                     col_mapping);
end;
/

declare
 error_count pls_integer := 0;
begin
  dbms_redefinition.copy_table_dependents('pm', 'cust', 'cust_int',
                                          1, true,true,true,false,
                                          error_count);

  dbms_output.put_line('errors := ' || to_char(error_count));
end;
/

exec  dbms_redefinition.finish_redef_table('pm', 'cust', 'cust_int');

-- Drop the interim table
drop table cust_int;

desc cust;

-- The following insert statement fails. This illustrates
-- that the primary key constraint on the c_id column is
-- preserved after migration.

insert into cust values(1, 94065, 'hhh', 'ttt');

select * from cust;
```

See Also: The following related documentation provides additional details on the redefinition process described earlier in this section:

- *Oracle Database Administrator's Guide* gives detailed procedures and examples of redefining tables online.
- *PL/SQL Packages and Types Reference* includes information on syntax and other details on usage of procedures in the DBMS_REDEFINITION package.

Migrating Applications from LONGs to LOBs

This section discusses differences between LONG and LOB datatypes that may impact your application migration plans or require you to modify your application.

Most APIs that work with LONG datatypes in the PL/SQL and OCI environments are enhanced to also work with LOB datatypes. These APIs are collectively referred to as the *data interface for persistent LOBs*, or simply the *data interface*. Among other things, the data interface provides the following benefits:

- Changes needed are minimal in PL/SQL and OCI applications that use tables with columns converted from LONG to LOB datatypes.
- You can work with LOB datatypes in your application without having to deal with LOB locators.

See Also:

- Chapter 13, "Data Interface for Persistent LOBs" for details on PL/SQL and OCI APIs included in the data interface.
- Chapter 9, "SQL Semantics and LOBs" for details on SQL syntax supported for LOB datatypes.
- Chapter 10, "PL/SQL Semantics for LOBs" for details on PL/SQL syntax supported for LOB datatypes.

LOB Columns Not Allowed in Clustered Tables

LOB columns are not allowed in clustered tables, whereas LONGs are allowed. If a table is a part of a cluster, then any LONG or LONG RAW column cannot be changed to a LOB column.

LOB Columns Not Allowed in UPDATE OF Triggers

You cannot have LOB columns in the UPDATE OF list of an UPDATE OF trigger. LONG columns are allowed in such triggers. For example, the following create trigger statement is not valid:

```
create table t(lobcol CLOB);
create trigger trig before/after update of lobcol on t ...;
```

All other triggers work on LOB columns.

Indexes on Columns Converted from LONG to LOB Datatypes

Indexes on any column of the table being migrated must be manually rebuilt after converting any LONG column to a LOB column. This includes function based indexes.

Any function based index on a LONG column will be unusable during the conversion process and must be rebuilt after converting. Application code that uses function based indexing should work without modification after converting.

Note that, any domain indexes on a LONG column must be dropped before converting the LONG column to LOB column. You can rebuild the domain index after converting.

To rebuild an index after converting, use the following steps:

1. Select the index from your original table as follows:

   ```
   SELECT index_name FROM user_indexes WHERE table_name='LONG_TAB';
   ```

   Note: The table name must be capitalized in this query.

   ---

2. For the selected index, use the command:

   ```
   ALTER INDEX <index> REBUILD
   ```

Empty LOBs Compared to NULL and Zero Length LONGs

A LOB column can hold an *empty* LOB. An empty LOB is a LOB locator that is fully initialized, but not populated with data. Because LONG datatypes do not use locators, the "empty" concept does not apply to LONG datatypes.

Both LOB column values and LONG column values, inserted with an initial value of NULL or an empty string literal, have a NULL value. Therefore, application code that uses NULL or zero-length values in a LONG column will function exactly the same after you convert the column to a LOB type column.

In contrast, a LOB initialized to empty has a non-NULL value as illustrated in the following example:

```
CREATE TABLE long_tab(id NUMBER, long_col LONG);
CREATE TABLE lob_tab(id NUMBER, lob_col CLOB);

INSERT INTO long_tab values(1, NULL);

REM     A zero length string inserts a NULL into the LONG column:
INSERT INTO long_tab values(1, '');

INSERT INTO lob_tab values(1, NULL);

REM     A zero length string inserts a NULL into the LOB column:
INSERT INTO lob_tab values(1, '');

REM     Inserting an empty LOB inserts a non-NULL value:
INSERT INTO lob_tab values(1, empty_clob());

DROP TABLE long_tab;
DROP TABLE lob_tab;
```

Overloading with Anchored Types

For applications using anchored types, some overloaded variables resolve to different targets during the conversion to LOBs. For example, given the procedure `p` overloaded with specifications 1 and 2:

```
procedure p(l long) is ...;       -- (specification 1)
procedure p(c clob) is ...;       -- (specification 2)
```

and the procedure call:

```
declare
     var  longtab.longcol%type;
   begin
     ...
   p(var);
     ...
end;
```

Prior to migrating from LONG to LOB columns, this call would resolve to specification 1. Once `longtab` is migrated to LOB columns this call will resolve to specification 2. Note that this would also be true if the parameter type in specification 1 were a CHAR, VARCHAR2, RAW, LONG RAW.

If you have migrated you tables from LONG columns to LOB columns, then you must manually examine your applications and determine whether overloaded procedures must be changed.

Some applications that included overloaded procedures with LOB arguments before migrating may still break. This includes applications that do not use LONG anchored types. For example, given the following specifications (1 and 2) and procedure call for procedure `p`:

```
procedure p(n number) is ...;       -- (1)
procedure p(c clob) is ...;         -- (2)

p('123');             -- procedure call
```

Before migrating, the only conversion allowed was CHAR to NUMBER, so specification 1 would be chosen. After migrating, both conversions are allowed, so the call is ambiguous and raises an overloading error.

Some Implicit Conversions Are Not Supported for LOB Datatypes

PL/SQL permits implicit conversion from NUMBER, DATE, ROW_ID, BINARY_INTEGER, and PLS_INTEGER datatypes to a LONG; however, implicit conversion from these datatypes to a LOB is not allowed.

If your application uses these implicit conversions, then you will have to explicitly convert these types using the TO_CHAR operator for character data or the TO_RAW operator for binary data. For example, if your application has an assignment operation such as:

```
number_var := long_var;  -- The RHS is a LOB variable after converting.
```

then you must modify your code as follows:

```
number_var := TO_CHAR(long_var);
-- Assuming that long_var is of type CLOB after conversion
```

The following conversions are not supported for LOB types:

- BLOB to VARCHAR2, CHAR, or LONG

- CLOB to RAW or LONG RAW

This applies to all operations where implicit conversion takes place. For example if you have a SELECT statement in your application as follows:

```
SELECT long_raw_column INTO my_varchar2 VARIABLE FROM my_table
```

and `long_raw_column` is a BLOB after converting your table, then the SELECT statement will produce an error. To make this conversion work, you must use the TO_RAW operator to explicitly convert the BLOB to a RAW as follows:

```
SELECT TO_RAW(long_raw_column) INTO my_varchar2 VARIABLE FROM my_table
```

The same holds for selecting a CLOB into a RAW variable, or for assignments of CLOB to RAW and BLOB to VARCHAR2.

Part IV

Using LOB APIs

This part provides details on using LOB APIs in supported environments. Examples of LOB API usage are given.

This part includes the following chapters:

- Chapter 12, "Operations Specific to Persistent and Temporary LOBs"
- Chapter 13, "Data Interface for Persistent LOBs"
- Chapter 14, "LOB APIs for Basic Operations"
- Chapter 15, "LOB APIs for BFILE Operations"

12

Operations Specific to Persistent and Temporary LOBs

This chapter discusses LOB operations that differ between persistent and temporary LOB instances. The following topics are covered in this chapter:

- Persistent LOB Operations
- Temporary LOB Operations
- Creating Persistent and Temporary LOBs in PL/SQL

See Also:

- Chapter 14, "LOB APIs for Basic Operations"
  Gives details and examples of API usage for LOB APIs that can be used with either temporary or persistent LOBs.
- Chapter 15, "LOB APIs for BFILE Operations"
  Gives details and examples for usage of LOB APIs that operate on BFILEs.

Persistent LOB Operations

This section describes operations that apply only to persistent LOBs.

Inserting a LOB into a Table

You can insert LOB instances into persistent LOB columns using any of the methods described in Chapter 8, "DDL and DML Statements with LOBs".

Selecting a LOB from a Table

You can select a persistent LOB from a table just as you would any other datatype. In the following example, persistent LOB instances of different types are selected into PL/SQL variables.

```
declare
  blob1 BLOB;
  blob2 BLOB;
  clob1 CLOB;
  nclob1 NCLOB;
begin
  SELECT ad_photo INTO blob1 FROM print_media WHERE Product_id = 2268
        FOR UPDATE;
  SELECT ad_photo INTO blob2 FROM print_media WHERE Product_id = 3106;

  SELECT ad_sourcetext INTO clob1 FROM Print_media
      WHERE product_id=3106 and ad_id=13001 FOR UPDATE;

  SELECT ad_fltextn INTO nclob1 FROM Print_media
      WHERE product_id=3060 and ad_id=11001 FOR UPDATE;

end;
/
show errors;
```

Temporary LOB Operations

This section describes operations that apply only to temporary LOB instances.

Creating and Freeing a Temporary LOB

To create a temporary LOB instance, you must declare a variable of the given LOB datatype and pass the variable to the CREATETEMPORARY API. The temporary LOB instance will exist in your application until it goes out of scope, your session terminates, or you explicitly free the instance. Freeing a temporary LOB instance is recommended to free system resources.

The following example demonstrates how to create and free a temporary LOB in the PL/SQL environment using the DBMS_LOB package.

```
declare
  blob1 BLOB;
  blob2 BLOB;
  clob1 CLOB;
  nclob1 NCLOB;
begin
  -- create temp LOBs
  DBMS_LOB.CREATETEMPORARY(blob1,TRUE, DBMS_LOB.SESSION);
  DBMS_LOB.CREATETEMPORARY(blob2,TRUE, DBMS_LOB.SESSION);
  DBMS_LOB.CREATETEMPORARY(clob1,TRUE, DBMS_LOB.SESSION);
  DBMS_LOB.CREATETEMPORARY(nclob1,TRUE, DBMS_LOB.SESSION);

  -- fill with data
  writeDataToLOB_proc(blob1);
  writeDataToLOB_proc(blob2);

  -- CHAR->LOB conversion
  clob1 := 'abcde';
  nclob1 := TO_NCLOB(clob1);

  -- Other APIs
  call_lob_apis(blob1, blob2, clob1, nclob1);

  -- free temp LOBs
  DBMS_LOB.FREETEMPORARY(blob1);
  DBMS_LOB.FREETEMPORARY(blob2);
  DBMS_LOB.FREETEMPORARY(clob1);
  DBMS_LOB.FREETEMPORARY(nclob1);

end;
/
show errors;
```

Creating Persistent and Temporary LOBs in PL/SQL

The code example that follows illustrates how to create persistent and temporary LOBs in PL/SQL. This code is in the demonstration file:

`$ORACLE_HOME/rdbms/demo/lobs/plsql/lobdemo.sql */`

This demonstration file also calls procedures in separate PL/SQL files that illustrate usage of other LOB APIs. For a list of these files and links to more information about related LOB APIs, see "PL/SQL LOB Demonstration Files" on page A-2.

```
-----------------------------------------------------------------------------
-------------------------  Persistent LOB operations ------------------------
-----------------------------------------------------------------------------

declare
  blob1 BLOB;
  blob2 BLOB;
  clob1 CLOB;
  nclob1 NCLOB;
begin
  SELECT ad_photo INTO blob1 FROM print_media WHERE Product_id = 2268
        FOR UPDATE;
  SELECT ad_photo INTO blob2 FROM print_media WHERE Product_id = 3106;

  SELECT ad_sourcetext INTO clob1 FROM Print_media
      WHERE product_id=3106 and ad_id=13001 FOR UPDATE;

  SELECT ad_fltextn INTO nclob1 FROM Print_media
      WHERE product_id=3060 and ad_id=11001 FOR UPDATE;

  call_lob_apis(blob1, blob2, clob1, nclob1);
  rollback;
end;
/
show errors;

-----------------------------------------------------------------------------
-------------------------  Temporary LOB operations ------------------------
-----------------------------------------------------------------------------

declare
  blob1 BLOB;
  blob2 BLOB;
  clob1 CLOB;
  nclob1 NCLOB;
```

```
begin
  -- create temp LOBs
  DBMS_LOB.CREATETEMPORARY(blob1,TRUE, DBMS_LOB.SESSION);
  DBMS_LOB.CREATETEMPORARY(blob2,TRUE, DBMS_LOB.SESSION);
  DBMS_LOB.CREATETEMPORARY(clob1,TRUE, DBMS_LOB.SESSION);
  DBMS_LOB.CREATETEMPORARY(nclob1,TRUE, DBMS_LOB.SESSION);

  -- fill with data
  writeDataToLOB_proc(blob1);
  writeDataToLOB_proc(blob2);

  -- CHAR->LOB conversion
  clob1 := 'abcde';
  nclob1 := TO_NCLOB(clob1);

  -- Other APIs
  call_lob_apis(blob1, blob2, clob1, nclob1);

  -- free temp LOBs
  DBMS_LOB.FREETEMPORARY(blob1);
  DBMS_LOB.FREETEMPORARY(blob2);
  DBMS_LOB.FREETEMPORARY(clob1);
  DBMS_LOB.FREETEMPORARY(nclob1);

end;
/
show errors;
```

13

Data Interface for Persistent LOBs

This chapter covers the following topics:

- Overview of the Data Interface for Persistent LOBs
- Benefits of Using the Data Interface for Persistent LOBs
- Using the Data Interface for Persistent LOBs in PL/SQL
- Using the Data Interface for Persistent LOBs in OCI

Overview of the Data Interface for Persistent LOBs

The data interface for persistent LOBs includes a set of PL/SQL and OCI APIs that are extended to work with LOB datatypes. These APIs, originally designed for use with legacy datatypes such as LONG, LONG RAW, and VARCHAR2, can also be used with the corresponding LOB datatypes shown in Table 13–1 and Table 13–2. These tables show the legacy datatypes in the "bind or define type" column and the corresponding supported LOB datatype in the "LOB column type" column. You can use the data interface for LOBs to store and manipulate character data and binary data in a LOB column just as if it were stored in the corresponding legacy datatype.

For simplicity, this chapter focuses on character datatypes; however, the same concepts apply to the full set of character and binary datatypes listed in Table 13–1 and Table 13–2.

Table 13–1 Corresponding LONG and LOB Datatypes in SQL and PL/SQL

Bind or Define Type	LOB Column Type	Used For Storing
CHAR	CLOB	Character data
LONG	CLOB	Character data
VARCHAR2	CLOB	Character data
LONG RAW	BLOB	Binary data
RAW	BLOB	Binary data

Table 13–2 Corresponding LONG and LOB Datatypes in OCI

Bind or Define Type	LOB Column Type	Used For Storing
SQLT_AFC(n)	CLOB	Character data
SQLT_CHR	CLOB	Character data
SQLT_LNG	CLOB	Character data
SQLT_VCS	CLOB	Character data
SQLT_BIN	BLOB	Binary data
SQLT_LBI	BLOB	Binary data
SQLT_LVB	BLOB	Binary data

Benefits of Using the Data Interface for Persistent LOBs

Using the data interface for persistent LOBs has the following benefits:

- If your application already uses LONG datatypes, then you can use the same application with LOB datatypes with little or no modification of your existing application required. To do so, just convert LONG datatype columns in your tables to LOB datatype columns as discussed in Chapter 11, "Migrating Table Columns from LONGs to LOBs".
- Performance is better for OCI applications that use sequential access techniques. A piecewise INSERT or fetch using the data interface has comparable performance to using OCI functions like `OCILobRead2()` and `OCILobWrite2()`. Because the data interface allows more than 4k bytes of data to be inserted into a LOB in a single OCI call, a round-trip to the server is saved.
- You can read LOB data in one `OCIStmtFetch()` call, instead of fetching the LOB locator first and then calling `OCILobRead2()`. This improves performance when you want to read LOB data starting at the beginning.

Using the Data Interface for Persistent LOBs in PL/SQL

The data interface enables you to use LONG and LOB datatypes listed in Table 13–1 to perform the following operations in PL/SQL:

- INSERT or UPDATE character data stored in datatypes such as VARCHAR2, CHAR, or LONG into a CLOB column.
- INSERT or UPDATE binary data stored in datatypes such as RAW or LONG RAW into a BLOB column.
- Use the SELECT statement on CLOB columns to select data into a character buffer variable such as CHAR, LONG, or VARCHAR2.
- Use the SELECT statement on BLOB columns to select data into a binary buffer variable such as RAW and LONG RAW.
- Make cross-type assignments (implicit type conversions) between CLOB and VARCHAR2, CHAR, or LONG variables.
- Make cross-type assignments (implicit type conversions) between BLOB and RAW or LONG RAW variables.
- Pass LOB datatypes to functions defined to accept LONG datatypes or pass LONG datatypes to functions defined to accept LOB datatypes. For example,

you can pass a CLOB instance to a function defined to accept another character type, such as VARCHAR2, CHAR, or LONG.

- Use CLOBs with other PL/SQL functions and operators that accept VARCHAR2 arguments such as INSTR and SUBSTR. See "Passing CLOBs to SQL and PL/SQL Built-In Functions" on page 13-6 for a complete list.

Note: When using the data interface for LOBs with the SELECT statement in PL/SQL, 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, then the database throws an exception.

See Also:

- Chapter 9, "SQL Semantics and LOBs" for details on LOB support in SQL statements
- "Some Implicit Conversions Are Not Supported for LOB Datatypes" on page 11-13

Guidelines for Accessing LOB Columns Using the Data Interface in SQL and PL/SQL

This section describes techniques you use to access LOB columns using the data interface for persistent 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, or fetch routine in PL/SQL. Therefore, the amount of data that can be accessed from a LOB column is limited by the maximum character buffer size. PL/SQL supports character buffer sizes up to 32K (32767) bytes. For this reason, LOB data of up to only 32K bytes in size can be accessed by PL/SQL applications using the data interface for persistent LOBs.

If you need to access more than 32K bytes using the data interface, then you must make OCI calls from the PL/SQL code to use the APIs for piecewise insert and fetch.

Use he following are guidelines for using the data interface to access LOB columns:

- INSERT operations

You can INSERT into tables containing LOB columns using regular INSERT statements in the VALUES clause. The field of the LOB column can be a literal, a character datatype, a binary datatype, or a LOB locator.

- UPDATE operations

  LOB columns can be updated as a whole by UPDATE... SET statements. In the SET clause, the new value can be a literal, a character datatype, a binary datatype, or a LOB locator.

- 4000 byte limit on hex to raw and raw to hex conversions

  The database does not do implicit hex to raw or raw to hex conversions on data that is more than 4000 bytes in size. You cannot bind a buffer of character data to a binary datatype column, and you cannot bind a buffer of binary data to a character datatype column if the buffer is over 4000 bytes in size. Attempting to do so will result in your column data being truncated at 4000 bytes.

  For example, you cannot bind a VARCHAR2 buffer to a LONG RAW or a BLOB column if the buffer is more than 4000 bytes in size. Similarly, you cannot bind a RAW buffer to a LONG or a CLOB column if the buffer is more than 4000 bytes in size.

- SELECT operations

  LOB columns can be selected into character or binary buffers in PL/SQL. If the LOB column is longer than the buffer size, then 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

Implicit assignment and parameter passing are supported for LOB columns. For the datatypes listed in Table 13–1 and Table 13–2, you can pass or assign: any character type to any other character type, or any binary type to any other binary type using the data interface for persistent LOBs.

Implicit assignment works for variables declared explicitly and for variables declared by referencing an existing column type using the `%TYPE` attribute as show in the following example. This example assumes that column `long_col` in table `t` has been migrated from a `LONG` to a `CLOB` column.

```
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;
```

Implicit parameter passing is allowed between functions and procedures. For example, you can pass a CLOB to a function or procedure where the formal parameter is defined as a VARCHAR2.

Note: The assigning a VARCHAR2 buffer to a LOB variable is somewhat less efficient than assigning a VARCHAR2 to a LONG variable because the former involves creating a temporary LOB. Therefore, PL/SQL users will see a slight deterioration in the performance of their applications.

Passing CLOBs to SQL and PL/SQL Built-In Functions

Implicit parameter passing is also supported for built-in PL/SQL functions that accept character data. For example, INSTR can accept a CLOB as well as other character data.

Any SQL or PL/SQL built-in function that accepts a VARCHAR2 can accept a CLOB as an argument. Similarly, a VARCHAR2 variable can be passed to any DBMS_LOB API for any parameter that takes a LOB locator.

See Also: Chapter 9, "SQL Semantics and LOBs"

Explicit Conversion Functions

In PL/SQL, the following explicit conversion functions convert other data types to `CLOB` and `BLOB` datatypes as follows:

- `TO_CLOB()` converts `LONG`, `VARCHAR2`, and `CHAR` to `CLOB`
- `TO_BLOB()` converts `LONG RAW` and `RAW` to `BLOB`

Also note that the conversion function `TO_CHAR()` can convert a `CLOB` to a `CHAR` type.

Calling PL/SQL and C Procedures from SQL

When a PL/SQL or C procedure is called from SQL, buffers with more than 4000 bytes of data are not allowed.

Calling PL/SQL and C Procedures from PL/SQL

You can call a PL/SQL or C procedure from PL/SQL. You can pass a CLOB as an actual parameter where CHR is the formal parameter, or vice versa. The same holds for BLOBs and RAWs.

One example of when these cases can arise is when either the formal or the actual parameter is an anchored type, that is, the variable is declared using the `table_name.column_name%type` syntax.

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 a CLOB:***

```
CREATE OR REPLACE PROCEDURE get_lob(table_name IN VARCHAR2, lob INOUT
CLOB) AS
   ...
BEGIN
  ...
END;
/
```

- ***When the formal parameter is a 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 CLOB;
begin
  get_lob('table_name', c);
end;
```

Binds of All Sizes in INSERT and UPDATE Operations

Binds of all sizes are supported for INSERT and UPDATE operations on LOB columns. Multiple binds of any size are allowed in a single INSERT or UPDATE statement.

> **Note:** When you create a table, the length of the default value you specify for any LOB column is restricted to 4,000 bytes.

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 print_media (ad_sourcetext) 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 print_media (ad_photo) VALUES (lpad('a', 5000, 'a'));
```

Restrictions on Binds of More Than 4,000 Bytes

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.

Example: PL/SQL - Using Binds of More Than 4,000 Bytes in INSERT and UPDATE

```
DECLARE
  bigtext VARCHAR2(32767);
  smalltext VARCHAR2(2000);
  bigraw RAW (32767);
BEGIN
  bigtext := LPAD('a', 32767, 'a');
  smalltext := LPAD('a', 2000, 'a');
  bigraw := utl_raw.cast_to_raw (bigtext);

  /* Multiple long binds for LOB columns are allowed for INSERT: */
  INSERT INTO print_media(product_id, ad_id, ad_sourcetext, ad_composite)
    VALUES (2004, 1, bigtext, bigraw);

  /* Single long bind for LOB columns is allowed for INSERT: */
  INSERT INTO print_media (product_id, ad_id, ad_sourcetext)
    VALUES (2005, 2, smalltext);

  bigtext := LPAD('b', 32767, 'b');
  smalltext := LPAD('b', 20, 'a');
  bigraw := utl_raw.cast_to_raw (bigtext);

  /* Multiple long binds for LOB columns are allowed for UPDATE: */
  UPDATE print_media SET ad_sourcetext = bigtext, ad_composite = bigraw,
    ad_finaltext = smalltext;

  /* Single long bind for LOB columns is allowed for UPDATE: */
  UPDATE print_media set ad_sourcetext = smalltext, ad_finaltext = 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 print_media(product_id, ad_id, ad_sourcetext, press_release)
```

```
    VALUES (2030, 3, bigtext, bigtext);

  /* The following is NOT allowed because we are trying to insert
     data into LOB attribute */
  INSERT into print_media(product_id, ad_id, ad_header)
     VALUES (2049, 4, adheader_typ(null, null, null, bigraw));

  /* The following is not allowed because we try to perform INSERT AS
     SELECT data INTO LOB */
  INSERT INTO print_media(product_id, ad_id, ad_sourcetext)
    SELECT 2056, 5, bigtext FROM dual;

END;
/
```

4,000 Byte Result Limit for SQL Operators

The following example illustrates how the result for SQL operators is limited to 4,000 bytes.

```
/* The following command inserts only 4,000 bytes because the result of
 * LPAD is limited to 4,000 bytes */
INSERT INTO print_media(product_id, ad_id, ad_sourcetext)
  VALUES (2004, 5, lpad('a', 5000, 'a'));
select length(ad_sourcetext) from print_media
  where product_id=2004 and ad_id=5;
rollback;

/* 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 print_media(product_id, ad_id, ad_composite)
  VALUES (2004, 5, lpad('a', 5000, 'a'));
select length(ad_composite) from print_media
  where product_id=2004 and ad_id=5;
rollback;
```

Using the Data Interface for LOBs with INSERT, UPDATE, and SELECT Operations

INSERT and UPDATE statements on LOBs are used in the same way as on LONGs. For example:

```
DECLARE
```

```
  ad_buffer VARCHAR2(100);
BEGIN
  INSERT INTO print_media(product_id, ad_id, ad_sourcetext)
    VALUES(2004, 5, 'Source for advertisement 1');
  UPDATE print_media SET ad_sourcetext= 'Source for advertisement 2'
    WHERE product_id=2004 and ad_id=5;
  /* This will get the LOB column if it is up to 100 bytes, otherwise it will
   * raise an exception */
  SELECT ad_sourcetext INTO ad_buffer FROM print_media
    WHERE product_id=2004 and ad_id=5;
END;
/
```

Using the Data Interface for LOBs in Assignments and Parameter Passing

The data interface for LOBs enables implicit assignment and parameter passing as shown in the following 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 OR REPLACE PROCEDURE FOO ( a IN OUT CLOB) IS
BEGIN
  -- Any procedure body
  a := 'abc';
END;
/

CREATE OR REPLACE PROCEDURE BAR (b IN OUT VARCHAR2) IS
BEGIN
```

```
  -- Any procedure body
  b := 'xyz';
END;
/

DECLARE
  a VARCHAR2(100) := '1234567';
  b CLOB;
BEGIN
  FOO(a);
  SELECT clob_col INTO b FROM t;
  BAR(b);
END;
/
```

Using the Data Interface for LOBs with PL/SQL Built-In Functions

This example illustrates the use of CLOBs in PL/SQL built-in functions, using the data interface for LOBs:

```
DECLARE
  my_ad CLOB;
  revised_ad CLOB;
  myGist VARCHAR2(100):= 'This is my gist.';
  revisedGist VARCHAR2(100);
BEGIN
  INSERT INTO print_media (product_id, ad_id, ad_sourcetext)
    VALUES (2004, 5, 'Source for advertisement 1');

  -- select a CLOB column into a CLOB variable
  SELECT ad_sourcetext INTO my_ad FROM print_media
    WHERE product_id=2004 and ad_id=5;

  -- perform VARCHAR2 operations on a CLOB variable
  revised_ad := UPPER(SUBSTR(my_ad, 100, 1));

  -- revised_ad is a temporary LOB
  -- Concat a VARCHAR2 at the end of a CLOB
  revised_ad := revised_ad || myGist;

  -- The following statement will raise an error if my_ad is
  -- longer than 100 bytes
  myGist := my_ad;
END;
/
```

Using the Data Interface for Persistent LOBs in OCI

This section discusses OCI functions included in the data interface for persistent LOBs. These OCI functions work for LOB datatypes exactly the same way as they do for LONG datatypes. Using these functions, you can perform INSERT, UPDATE, fetch, bind, and define operations in OCI on LOBs using the same techniques you would use on other datatypes that store character or binary data.

> **See Also:** "Runtime data allocation and piecewise operations" in the *Oracle Call Interface Programmer's Guide,* for details on OCI APIs.

Binding LOB Datatypes in OCI

You can bind LOB datatypes in the following operations:

- Regular, piecewise, and callback binds for INSERT and UPDATE operations
- Array binds for INSERT and UPDATE operations
- Parameter passing across PL/SQL and OCI boundaries

Piecewise operations can be performed by polling or by providing a callback. To support these operations, the following OCI functions accept the LONG and LOB datatypes listed in Table 13–2.

- `OCIBindByName()` and `OCIBindByPos()`
  These functions create an association between a program variable and a placeholder in the SQL statement or a PL/SQL block for INSERT and UPDATE operations.
- `OCIBindDynamic()`
  You use this call to register callbacks for dynamic data allocation for INSERT and UPDATE operations
- `OCIStmtGetPieceInfo()` and `OCIStmtSetPieceInfo()`
  These calls are used to get or set piece information for piecewise operations.

Defining LOB Datatypes in OCI

The data interface for persistent LOBs allows the following OCI functions to accept the LONG and LOB datatypes listed in Table 13–2.

- `OCIDefineByPos()`
  This call associates an item in a SELECT list with the type and output data buffer.

- `OCIDefineDynamic()`
  This call registers user callbacks for SELECT operations if the OCI_DYNAMIC_FETCH mode was selected in `OCIDefineByPos()` function call.

When you use these functions with LOB types, the LOB data, and not the locator, is selected into your buffer. Note that in OCI, you cannot specify the amount you want to read using the data interface for LOBs. You can only specify the buffer length of your buffer. The database only reads whatever amount fits into your buffer and the data is truncated.

Using Multibyte Charactersets in OCI with the Data Interface for LOBs

When the client characterset is in a multibyte format, functions included in the data interface operate the same way with LOB datatypes as they do for LONG datatypes as follows:

- For a *piecewise* fetch in a multibyte characterset, a multibyte character could be cut in the middle, with some bytes at the end of one buffer and remaining bytes in the next buffer.
- For a *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.

Using OCI Functions to Perform INSERT or UPDATE on LOB Columns

This section discusses the various techniques you can use to perform INSERT or UPDATE operations on LOB columns using the data interface. The operations described in this section assume that you have initialized the OCI environment and allocated all necessary handles.

Simple INSERTs or UPDATEs in One Piece

To perform simple INSERT or UPDATE operations in one piece using the data interface for persistent LOBs, perform the following steps:

1. Call `OCIStmtPrepare()` to prepare the statement in OCI_DEFAULT mode.
2. Call `OCIBindByName()` or `OCIBindbyPos()` to bind a placeholder in OCI_DEFAULT mode to bind a LOB as character data or binary data.
3. Call `OCIStmtExecute()` to do the actual INSERT or UPDATE operation.

Using Piecewise INSERTs and UPDATEs with Polling

To perform piecewise INSERT or UPDATE operations with polling using the data interface for persistent LOBs, do the following steps:

1. Call `OCIStmtPrepare()` to prepare the statement in OCI_DEFAULT mode.
2. Call `OCIBindByName()` or `OCIBindbyPos()` to bind a placeholder in OCI_DATA_AT_EXEC mode to bind a LOB as character data or binary data.
3. Call `OCIStmtExecute()` in default mode. Do each of the following in a loop while the value returned from `OCIStmtExecute()` is OCI_NEED_DATA. Terminate your loop when the value returned from `OCIStmtExecute()` is OCI_SUCCESS.
   - Call `OCIStmtGetPieceInfo()` to retrieve information about the piece to be inserted.
   - Call `OCIStmtSetPieceInfo()` to set information about piece to be inserted.

Piecewise INSERTs and UPDATEs with Callback

To perform piecewise INSERT or UPDATE operations with callback using the data interface for persistent LOBs, do the following steps:

1. Call `OCIStmtPrepare()` to prepare the statement in OCI_DEFAULT mode.
2. Call `OCIBindByName()` or `OCIBindbyPos()` to bind a placeholder in OCI_DATA_AT_EXEC mode to bind the LOB column as character data or binary data.
3. Call `OCIBindDynamic()` to specify the callback.
4. Call `OCIStmtExecute()` in default mode.

Array INSERT and UPDATE Operations

To perform array INSERT or UPDATE operations using the data interface for persistent LOBs, use any of the techniques discussed in this section in conjunction with `OCIBindArrayOfStruct()`, or by specifying the number of iterations (*`iter`*), with *`iter`* value greater than 1, in the `OCIStmtExecute()` call.

Using the Data Interface to Fetch LOB Data in OCI

This section discusses techniques you can use to fetch data from LOB columns in OCI using the data interface for persistent LOBs.

Simple Fetch in One Piece

To perform a simple fetch operation on LOBs in one piece using the data interface for persistent LOBs, do the following:

1. Call `OCIStmtPrepare()` to prepare the SELECT statement in OCI_DEFAULT mode.
2. Call `OCIDefineByPos()` to define a select list position in OCI_DEFAULT mode to define a LOB as character data or binary data.
3. Call `OCIStmtExecute()` to run the SELECT statement.
4. Call `OCIStmtFetch()` to do the actual fetch.

Piecewise Fetch with Polling

To perform a piecewise fetch operation on a LOB column with polling using the data interface for LOBs, do the following steps:

1. Call `OCIStmtPrepare()` to prepare the SELECT statement in OCI_DEFAULT mode.
2. Call `OCIDefinebyPos()` to define a select list position in OCI_DYNAMIC_FETCH mode to define the LOB column as character data or binary data.
3. Call `OCIStmtExecute()` to run the SELECT statement.
4. Call `OCIStmtFetch()` in default mode. Do each of the following in a loop while the value returned from `OCIStmtFetch()` is OCI_NEED_DATA. Terminate your loop when the value returned from `OCIStmtFetch()` is OCI_SUCCESS.
   - Call `OCIStmtGetPieceInfo()` to retrieve information about the piece to be fetched.
   - Call `OCIStmtSetPieceInfo()` to set information about piece to be fetched.

Piecewise with Callback

To perform a piecewise fetch operation on a LOB column with callback using the data interface for persistent LOBs, do the following:

1. Call `OCIStmtPrepare()` to prepare the statement in OCI_DEFAULT mode.
2. Call `OCIDefinebyPos()` to define a select list position in OCI_DYNAMIC_FETCH mode to define the LOB column as character data or binary data.

3. Call `OCIStmtExecute()` to run the SELECT statement.
4. Call `OCIDefineDynamic()` to specify the callback.
5. Call `OCIStmtFetch()` in default mode.

Array Fetch

To perform an array fetch in OCI using the data interface for persistent LOBs, use any of the techniques discussed in this section in conjunction with `OCIDefineArrayOfStruct()`, or by specifying the number of iterations (*`iter`*), with the value of *`iter`* greater than 1, in the `OCIStmtExecute()` call.

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 Out-binds in the "begin foo(:1); end;" Manner.

Here is an example of calling PL/SQL out-binds in the "begin foo(:1);end;" manner:

```
text *sqlstmt = (text *)"BEGIN get_lob(:c); END; " ;
```

Calling PL/SQL Out-binds in the "call foo(:1);" Manner.

Here is an example of calling PL/SQL out-binds 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_NTV_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;
```

Example: C (OCI) - Binds of More than 4,000 Bytes for INSERT and UPDATE

```
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,
press_release)
              VALUES (:1, :2, :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 */
   ub1 buffer[8000];
   text *insert_sql = "INSERT INTO Print_media (ad_sourcetext,press_release)
              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, 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);
   OCIStmtExecute(svchp, stmthp, errhp, 1, 0, OCI_DEFAULT);
}
```

```
void update()
{
/* 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, press_release=: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 update()
{
/* 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, press_release=: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, press_
release)
             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, 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  */
}
```

Using the Data Interface for LOBs in PL/SQL Binds from OCI on LOBs

The data interface for LOBs 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 in out 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 calls the PL/SQL stub.

For the OCI calling program, the following are likely cases:

Calling PL/SQL Out-binds in the "begin foo(:1); end;" Manner

For example:

```
text *sqlstmt = (text *)"BEGIN PKG1.P5 (:c); END; " ;
```

Calling PL/SQL Out-binds 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;
```

Binding LONG Data for LOB Columns in Binds Greater Than 4,000 Bytes

The following example illustrates binding character data for a LOB column:

```
void simple_insert()
{
  word buflen;
  text buf[5000];
  text *insstmt = (text *) "INSERT INTO print_media(product_id, ad_id, ad_
sourcetext) VALUES (2004, 1, :SRCTXT)";

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

  OCIBindByName(stmthp, &bndhp[0], errhp,
                (text *) ":SRCTXT", (sb4) strlen((char *) ":SRCTXT"T),
                (dvoid *) buf, (sb4) sizeof(buf), SQLT_CHR,
                (dvoid *) 0, (ub2 *) 0, (ub2 *) 0,
                (ub4) 0, (ub4 *) 0, (ub4) OCI_DEFAULT);

  memset((void *)buf, (int)'A', (size_t)5000);
  OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                 (const OCISnapshot*) 0, (OCISnapshot*) 0, (ub4) OCI_DEFAULT);
}
```

Binding LONG Data to LOB Columns Using Piecewise INSERT with Polling

The following example illustrates using piecewise `INSERT` with polling using the data interface for LOBs.

```
void piecewise_insert()
{
```

```
  text *sqlstmt = (text *)"INSERT INTO print_media(product_id, ad_id, ad_
sourcetext) VALUES (:1, :2, :3)";
  ub2 rcode;
  ub1 piece, i;
  word product_id = 2004;
  word ad_id = 2;
  word buflen;
  char buf[5000];

  OCIStmtPrepare(stmthp, errhp, sqlstmt, (ub4)strlen((char *)sqlstmt),
                 (ub4) OCI_NTV_SYNTAX, (ub4) OCI_DEFAULT);
  OCIBindByPos(stmthp, &bndhp[0], errhp, (ub4) 1,
               (dvoid *) &product_id, (sb4) sizeof(product_id), SQLT_INT,
               (dvoid *) 0, (ub2 *)0, (ub2 *)0,
               (ub4) 0, (ub4 *) 0, (ub4) OCI_DEFAULT);
  OCIBindByPos(stmthp, &bndhp[1], errhp, (ub4) 2,
               (dvoid *) &ad_id, (sb4) sizeof(ad_id), SQLT_INT,
               (dvoid *) 0, (ub2 *)0, (ub2 *)0,
               (ub4) 0, (ub4 *) 0, (ub4) OCI_DEFAULT);
  OCIBindByPos(stmthp, &bndhp[2], errhp, (ub4) 3,
               (dvoid *) 0, (sb4) 15000, SQLT_LNG,
               (dvoid *) 0, (ub2 *)0, (ub2 *)0,
               (ub4) 0, (ub4 *) 0, (ub4) OCI_DATA_AT_EXEC);

  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 *)buf, (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 *)buf,
                              &buflen, piece, (dvoid *) 0, &rcode))
        {
          DISCARD printf("ERROR: OCIStmtSetPieceInfo: %d \n", retval);
```

```
          break;
        }

      break;
    case OCI_SUCCESS:
      break;
    default:
      DISCARD printf( "oci exec returned %d \n", retval);
      report_error(errhp);
      retval = OCI_SUCCESS;
    } /* end switch */
    if (retval == OCI_SUCCESS)
      break;
  } /* end while(1) */
}
```

Binding LONG Data to LOB Columns Using Piecewise INSERT with Callback

The following example illustrates binding LONG data to LOB columns using a piecewise INSERT with callback:

```
void callback_insert()
{
  word buflen = 15000;
  word product_id = 2004;
  word ad_id = 3;
  text *sqlstmt = (text *) "INSERT INTO print_media(product_id, ad_id, ad_
sourcetext) VALUES (:1, :2, :3)";
  word pos = 3;

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

  OCIBindByPos(stmthp, &bndhp[0], errhp, (ub4) 1,
               (dvoid *) &product_id, (sb4) sizeof(product_id), SQLT_INT,
               (dvoid *) 0, (ub2 *)0, (ub2 *)0,
               (ub4) 0, (ub4 *) 0, (ub4) OCI_DEFAULT);
  OCIBindByPos(stmthp, &bndhp[1], errhp, (ub4) 2,
               (dvoid *) &ad_id, (sb4) sizeof(ad_id), SQLT_INT,
               (dvoid *) 0, (ub2 *)0, (ub2 *)0,
               (ub4) 0, (ub4 *) 0, (ub4) OCI_DEFAULT);
  OCIBindByPos(stmthp, &bndhp[2], errhp, (ub4) 3,
               (dvoid *) 0, (sb4) buflen, SQLT_CHR,
               (dvoid *) 0, (ub2 *)0, (ub2 *)0,
```

```
                (ub4) 0, (ub4 *) 0, (ub4) OCI_DATA_AT_EXEC);

  OCIBindDynamic(bndhp[2], errhp, (dvoid *) (dvoid *) &pos,
                 insert_cbk, (dvoid *) 0, (OCICallbackOutBind) 0);

  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 insert_cbk(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);
  char   buf[5000];

  switch(inpos)
  {
  case 3:
    memset((void *)buf, (int) 'A'+a, (size_t) 5000);
    *bufpp = (dvoid *) buf;
    *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;
}
```

Binding LONG Data to LOB Columns Using an Array INSERT

The following example illustrates binding character data for LOB columns using an array INSERT operation:

```
void array_insert()
{
  word buflen;
  word arrbuf1[5];
  word arrbuf2[5];
  text arrbuf3[5][5000];
  text *insstmt = (text *)"INSERT INTO print_media(product_id, ad_id, ad_
sourcetext) VALUES (:PID, :AID, :SRCTXT)"

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

  OCIBindByName(stmthp, &bndhp[0], errhp,
               (text *) ":PID", (sb4) strlen((char *) ":PID"),
               (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 *) ":AID", (sb4) strlen((char *) ":AID"),
               (dvoid *) &arrbuf2[0], (sb4) sizeof(arrbuf2[0]), SQLT_INT,
               (dvoid *) 0, (ub2 *)0, (ub2 *) 0,
               (ub4) 0, (ub4 *) 0, (ub4) OCI_DEFAULT);

  OCIBindByName(stmthp, &bndhp[2], errhp,
               (text *) ":SRCTXT", (sb4) strlen((char *) ":SRCTXT"),
               (dvoid *) arrbuf3[0], (sb4) sizeof(arrbuf3[0]), SQLT_CHR,
               (dvoid *) 0, (ub2 *) 0, (ub2 *) 0,
               (ub4) 0, (ub4 *) 0, (ub4) OCI_DEFAULT);
```

```
  OCIBindArrayOfStruct(bndhp[0], ERRH, sizeof(arrbuf1[0]),
                       indsk, rlsk, rcsk);
  OCIBindArrayOfStruct(bndhp[1], ERRH, sizeof(arrbuf2[0]),
                       indsk, rlsk, rcsk);
  OCIBindArrayOfStruct(bndhp[1], ERRH, sizeof(arrbuf3[0]),
                       indsk, rlsk, rcsk);

  for (i=0; i<5; i++)
  {
    arrbuf1[i] = 2004;
    arrbuf2[i] = i+4;
    memset((void *)arrbuf3[i], (int)'A'+i, (size_t)5000);
  }
  OCIStmtExecute(svchp, stmthp, errhp, (ub4) 5, (ub4) 0,
                 (const OCISnapshot*) 0, (OCISnapshot*) 0,
                 (ub4) OCI_DEFAULT);

}
```

Selecting a LOB Column into a LONG Buffer Using a Simple Fetch

The following example illustrates selecting a LOB column using a simple fetch:

```
void simple_fetch()
{
  word i, buf1 = 0;
  word retval;
  text buf[15000];
  text *selstmt = (text *) "SELECT AD_SOURCETEXT FROM PRINT_MEDIA WHERE PRODUCT_
ID = 2004";

  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, &defhp[1], errhp, (ub4) 2, (dvoid *) buf,
                   (sb4) sizeof(buf1), (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("buf = %.*s\n", buf2);
  }
}
```

Selecting a LOB Column into a LONG Buffer Using Piecewise Fetch with Polling

The following example illustrates selecting a LOB column into a LONG buffer using a piecewise fetch with polling:

```
void piecewise_fetch()
{
  text buf[15000];
  word buflen=5000;
  word retval;
  text *selstmt = (text *) "SELECT AD_SOURCETEXT FROM PRINT_MEDIA
                  WHERE PRODUCT_ID = 2004 AND AD_ID=2";

  OCIStmtPrepare(stmthp, errhp, sqlstmt,
                 (ub4) strlen((char *)sqlstmt),
                 (ub4) OCI_NTV_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 OCISnapshot*) 0, (OCISnapshot*) 0,
                          (ub4) OCI_DEFAULT);

  retval = OCIStmtFetch(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 *)buf, 5000);
    switch(retval)
    {
    case OCI_NEED_DATA:
```

```
        OCIStmtGetPieceInfo(stmthp, errhp, &hdlptr, &hdltype,
                            &in_out, &iter, &idx, &piece);
        OCIStmtSetPieceInfo(hdlptr, hdltype, errhp,
                            (dvoid *) buf, &buflen, piece,
                            (CONST dvoid *) &indp1, (ub2 *) 0);
        retval = OCI_NEED_DATA;
        break;
      default:
        DISCARD printf("ERROR: piece-wise fetching, %d\n", retval);
        return;
      } /* end switch */
      retval = OCIStmtFetch(stmthp, errhp, (ub4) 1 ,
                            (ub2) OCI_FETCH_NEXT, (ub4) OCI_DEFAULT);
      printf("Data : %s\n"; buf);
    } /* end while */
  }
```

Selecting a LOB Column into a LONG Buffer Using Piecewise Fetch with Callback

The following example illustrates selecting a LONG column into a LOB buffer when using a piecewise fetch with callback:

```
char buf[5000];
void callback_fetch()
{
  text *sqlstmt = (text *) "SELECT AD_SOURCETEXT FROM PRINT_MEDIA WHERE PRODUCT_
ID = 2004 AND AD_ID=3";

  OCIStmtPrepare(stmthp, errhp, sqlstmt, (ub4)strlen((char *)sqlstmt),
                 (ub4) OCI_NTV_SYNTAX, (ub4) OCI_DEFAULT);
  OCIDefineByPos(stmthp, &dfnhp[0], errhp, (ub4) 1,
                 (dvoid *) 0, (sb4)3 * sizeof(buf), SQLT_CHR,
                 (dvoid *) 0, (ub2 *)0, (ub2 *)0,
                 (ub4) OCI_DYNAMIC_FETCH);

  OCIDefineDynamic(dfnhp[0], errhp, (dvoid *) &outpos,
                   (OCICallbackDefine) fetch_cbk);

  OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                 (const OCISnapshot*) 0, (OCISnapshot*) 0,
                 (ub4) OCI_DEFAULT);
  buf[ 4999 ] = '\0';
  printf("Select callback: Last piece: %s\n", buf);
}
```

```
/* ------------------------------------------------------------- */
/* Fetch callback to specify buffers. */
/* ------------------------------------------------------------- */
STATICF sb4 fetch_cbk(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 = 5000;
  switch(outpos)
  {
  case 1:
    a ++;
    *bufpp = (dvoid *) buf;
    *alenpp = &len;
    break;
  default:
    *bufpp = (dvoid *) 0;
    *alenpp = (ub4 *) 0;
    DISCARD printf("ERROR: invalid position number: %d\n", pos);
  }
  *indpp = (dvoid *) 0;
  *rcpp = (ub2 *) 0;

  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;
}
```

Selecting a LOB Column into a LONG Buffer Using an Array Fetch

The following example illustrates selecting a LOB column into a LONG buffer using an array fetch:

```
void array_fetch()
{
  word i;
  text arrbuf[5][5000];
  text *selstmt = (text *) "SELECT AD_SOURCETEXT FROM PRINT_MEDIA WHERE PRODUCT_
ID = 2004 AND AD_ID >=4";

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

  OCIStmtExecute(svchp, stmthp, errhp, (ub4) 0, (ub4) 0,
                 (const OCISnapshot*) 0, (OCISnapshot*) 0, (ub4) OCI_DEFAULT);

  OCIDefineByPos(stmthp, &defhp1, errhp, (ub4) 2,
                   (dvoid *) arrbuf[0], (sb4) sizeof(arrbuf[0]),
                   (ub2) SQLT_CHR, (dvoid *) 0,
                   (ub2 *) 0, (ub2 *) 0, (ub4) OCI_DEFAULT);

  OCIDefineArrayOfStruct(dfnhp[0], ERRH, sizeof(arrbuf[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", arrbuf[0]);
    DISCARD printf("%d, %s\n", arrbuf[1]);
    DISCARD printf("%d, %s\n", arrbuf[2]);
    DISCARD printf("%d, %s\n", arrbuf[3]);
    DISCARD printf("%d, %s\n", arrbuf[4]);
  }
}
```

14

LOB APIs for Basic Operations

This chapter describes APIs that perform basic operations on BLOB, CLOB, and NCLOB datatypes. The operations given in this chapter can be used with either persistent or temporary LOB instances. Note that operations in this chapter do not apply to BFILEs. APIs covered in this chapter are listed in Table 14–1.

See Also:

- Chapter 12, "Operations Specific to Persistent and Temporary LOBs" for information on how to create temporary and persistent LOB instances and other operations specific to temporary or persistent LOBs.
- Chapter 15, "LOB APIs for BFILE Operations" for information on operations specific to BFILE instances.

The following information is given for each operation described in this chapter:

- ***Preconditions*** describe dependencies that must be met and conditions that must exist before calling each operation.
- ***Usage Notes*** provide implementation guidelines such as information specific to a given programmatic environment or datatype.
- ***Syntax*** refers you to the syntax reference documentation for each supported programmatic environment.
- ***Examples*** describe any setup tasks necessary to run the examples given.

Supported Environments

Table 14–1, "Environments Supported for Basic LOB APIs" indicates which programmatic environments are supported for the APIs discussed in this chapter. The first column describes the operation that the API performs. The remaining columns indicate with "Yes" or "No" whether the API is supported in PL/SQL, OCI, OCCI, COBOL, Pro*C, Visual Basic (VB), and JDBC.

Table 14–1 Environments Supported for Basic LOB APIs

Operation	PL/SQL	OCI	OCCI	COBOL	Pro*C	VB	JDBC
Appending One LOB to Another on page 14-4	Yes	Yes	No	Yes	Yes	Yes	Yes
Determining Character Set Form on page 14-13	No	Yes	No	No	No	No	No
Determining Character Set ID on page 14-15	No	Yes	No	No	No	No	No
Determining Chunk Size, See: Writing Data to a LOB on page 14-128	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Comparing All or Part of Two LOBs on page 14-71	Yes	No	No	Yes	Yes	Yes	Yes
Converting a BLOB to a CLOB on page 14-185	Yes	No	No	No	No	No	No
Converting a CLOB to a BLOB on page 14-185	Yes	No	No	No	No	No	No
Copying a LOB Locator on page 14-103	Yes	Yes	No	Yes	Yes	Yes	Yes
Copying All or Part of One LOB to Another LOB on page 14-93	Yes	Yes	No	Yes	Yes	Yes	Yes
Disabling LOB Buffering on page 14-171	No	Yes	No	Yes	Yes	Yes	No
Displaying LOB Data on page 14-42	Yes	Yes	No	Yes	Yes	Yes	Yes
Enabling LOB Buffering on page 14-162	No	No	No	Yes	Yes	Yes	No
Equality: Checking If One LOB Locator Is Equal to Another on page 14-111	No	Yes	No	No	Yes	No	Yes
Erasing Part of a LOB on page 14-154	Yes	Yes	No	Yes	Yes	Yes	Yes
Flushing the Buffer on page 14-167	No	Yes	No	Yes	Yes	No	No

Table 14–1 Environments Supported for Basic LOB APIs (Cont.)

Operation	PL/SQL	OCI	OCCI	COBOL	Pro*C	VB	JDBC
Determining Whether LOB Locator Is Initialized on page 14-117	No	Yes	No	No	Yes	No	No
Length: Determining the Length of a LOB on page 14-86	Yes	Yes	No	Yes	Yes	Yes	Yes
Loading a LOB with Data from a BFILE on page 14-17	Yes	Yes	No	Yes	Yes	Yes	Yes
Loading a BLOB with Data from a BFILE on page 14-26	Yes	No	No	No	No	No	No
Loading a CLOB or NCLOB with Data from a BFILE on page 14-29	Yes	No	No	No	No	No	No
Opening Persistent LOBs with the OPEN and CLOSE Interfaces on page 5-12	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Open: Determining Whether a LOB is Open on page 14-34	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Patterns: Checking for Patterns in a LOB Using INSTR on page 14-79	Yes	No	No	Yes	Yes	No	Yes
Reading a Portion of a LOB (SUBSTR) on page 14-63	Yes	No	No	Yes	Yes	Yes	Yes
Reading Data from a LOB on page 14-52	Yes	Yes	No	Yes	Yes	Yes	Yes
Storage Limit, Determining: Maximum Storage Limit for Terabyte-Size LOBs on page 5-31	Yes	No	No	No	No	No	No
Trimming LOB Data on page 14-143	Yes	Yes	No	Yes	Yes	Yes	Yes
WriteNoAppend, see Appending to a LOB on page 14-120.	No	No	No	No	No	No	No
Writing Data to a LOB on page 14-128	Yes	Yes	Yes	Yes	Yes	Yes	Yes

Appending One LOB to Another

This operation appends one LOB instance to another.

Preconditions

Before you can append one LOB to another, the following conditions must be met:

- Two LOB instances must exist.
- Both instances must be of the same type, for example both `BLOB` or both `CLOB` types.
- You can pass any combination of persistent or temporary LOB instances to this operation.

Usage Notes

Persistent LOBs: You must lock the row you are selecting the LOB from prior to updating a LOB value if you are using the PL/SQL DBMS_LOB package or OCI. While the SQL `INSERT` and `UPDATE` statements implicitly lock the row, locking the row can be done explicitly using the 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 Through Updated Locators" on page 5-16.

Syntax

See the following syntax references for each programmatic environment:

- PL/SQL (DBM_LOB Package): *PL/SQL Packages and Types Reference* "DBMS_LOB" — APPEND
- C (OCI): *Oracle Call Interface Programmer's Guide* "Relational Functions" — OCILobAppend
- C++ (OCCI): *Oracle C++ Call Interface Programmer's Guide*
- COBOL (Pro*COBOL) *Pro*COBOL 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++ 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 > Orablob > METHODS > append

- Java (JDBC): *Oracle Database JDBC Developer's Guide and Reference* for information on creating and populating LOB columns in Java.

Examples

To run the following examples, you must create two LOB instances and pass them when you call the given append operation. Creating a LOB instance is described in Chapter 12, "Operations Specific to Persistent and Temporary LOBs".

Examples for this use case are provided in the following programmatic environments:

- PL/SQL DBMS_LOB Package: Appending One LOB to Another on page 14-5
- C (OCI): Appending One LOB to Another on page 14-6
- C++ (OCCI): No example is provided with this release.
- COBOL (Pro*COBOL): Appending One LOB to Another on page 14-7
- C/C++ (Pro*C/C++): Appending One LOB to Another on page 14-8
- Visual Basic (OO4O): Appending One LOB to Another on page 14-9
- Java (JDBC): Appending One LOB to Another on page 14-10

PL/SQL DBMS_LOB Package: Appending One LOB to Another

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/plsql/lappend.sql */

/* Procedure appendLOB_proc is not part of the DBMS_LOB package: */

/* appending one lob to another */

CREATE OR REPLACE PROCEDURE appendLOB_proc
   (Dest_loc IN OUT BLOB, Src_loc IN OUT BLOB) IS
   /* Note: Dest_loc and Src_loc can be persistent or temporary LOBs */
BEGIN
   DBMS_OUTPUT.PUT_LINE('------------ LOB APPEND EXAMPLE ------------');
   /* 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);
   DBMS_OUTPUT.PUT_LINE('Append succeeded');

EXCEPTION
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('Append failed');
      DBMS_LOB.CLOSE (Dest_loc);
      DBMS_LOB.CLOSE (Src_loc);
END;
/
SHOW ERRORS;
```

C (OCI): Appending One LOB to Another

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/oci/lappend.c */

/* Appending one LOB to another. */
/* This function appends the Source LOB to the end of the Destination LOB */
#include <oratypes.h>
#include <lobdemo.h>
void appendLOB_proc(OCILobLocator *Lob_loc1, OCILobLocator *Lob_loc2,
                    OCIEnv *envhp, OCIError *errhp, OCISvcCtx *svchp,
                    OCIStmt *stmthp)
{
  printf ("----------- OCILobAppend Demo --------------\n");
  /* Opening the LOBs is Optional */
  checkerr (errhp, OCILobOpen(svchp, errhp, Lob_loc2, OCI_LOB_READWRITE));
  checkerr (errhp, OCILobOpen(svchp, errhp, Lob_loc1, 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, Lob_loc2, Lob_loc1));

  /* Closing the LOBs is Mandatory if they have been Opened */
  checkerr (errhp, OCILobClose(svchp, errhp, Lob_loc2));
  checkerr (errhp, OCILobClose(svchp, errhp, Lob_loc1));
  return;
}
```

COBOL (Pro*COBOL): Appending One LOB to Another

```
 * This file is installed in the following path when you install
 * the database: $ORACLE_HOME/rdbms/demo/lobs/procob/lappend.pco

* 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 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_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

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/proc/lappend.pc */

/* 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

```
' This file is installed in the following path when you install
' the database: $ORACLE_HOME/rdbms/demo/lobs/vb/lappend.bas

'Appending one LOB to another
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

OraDyn.MoveNext
OraDyn.Edit
OraAdPhoto1.Append OraAdPhotoClone
OraDyn.Update
```

Java (JDBC): Appending One LOB to Another

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/java/lappend.java */

// 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();
     }
   }
 }
```

Determining Character Set Form

This section describes how to get the character set form of a LOB instance.

Syntax

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* "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.

Example

The example demonstrates how to determine the character set form of the foreign language text (`ad_fltextn`).

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 14-13
- 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

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/oci/lgetchfm.c */
```

```
/* Getting character set form of the foreign language ad text, ad_fltextn */
#include <oratypes.h>
#include <lobdemo.h>
/* This function takes a valid LOB locator and prints the character set form
   of the LOB.
 */
void getCsformLOB_proc(OCILobLocator *Lob_loc, OCIEnv *envhp,
                       OCIError *errhp, OCISvcCtx *svchp, OCIStmt *stmthp)
{
  ub1 charset_form = 0 ;
  printf ("----------- OCILobCharSetForm Demo --------------\n");

  printf (" get the character set form of ad_fltextn\n");
  /* Get the charactersid form of the LOB*/
  checkerr (errhp, OCILobCharSetForm(envhp, errhp, Lob_loc, &charset_form));
  printf(" character Set Form of ad_fltextn is : %d\n", charset_form);

  return;
}
```

Determining Character Set ID

This section describes how to determine the character set ID.

Syntax

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* "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.

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 14-15
- 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

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/oci/lgetchar.c */

/* Getting character set id */
/* This function takes a valid LOB locator and prints the character set id of
```

```
   the LOB. */
#include <oratypes.h>
#include <lobdemo.h>
void getCsidLOB_proc(OCILobLocator *Lob_loc, OCIEnv *envhp,
                     OCIError *errhp, OCISvcCtx *svchp, OCIStmt *stmthp)
{
  ub2 charsetid =0 ;
  printf ("----------- OCILobCharSetID Demo --------------\n");

  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_fltextn is : %d\n", charsetid);

  return;
}
```

Loading a LOB with Data from a BFILE

This operation loads a LOB with data from a BFILE. This procedure can be used to load data into any persistent or temporary LOB instance of any LOB datatype.

See Also:

- The LOADBLOBFROMFILE and LOADCLOBFROMFILE procedures implement the functionality of this procedure and provide improved features for loading binary data and character data. (These 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 Data from a BFILE" on page 14-26 and "Loading a CLOB or NCLOB with Data from a BFILE" on page 14-29 for more information.
- As an alternative to this operation, you can use SQL*Loader to load persistent LOBs with data directly from a file in the file system. See "Using SQL*Loader to Load LOBs" on page 3-2 for more information.

Preconditions

Before you can load a LOB with data from a BFILE, the following conditions must be met:

- The BFILE must exist.
- The target LOB instance must exist.

Usage Notes

Note the following issues regarding this operation.

Use LOADCLOBFROMFILE When Loading Character Data

When you use the DBMS_LOB.LOADFROMFILE procedure to load a CLOB or NCLOB instance, you are loading the LOB with binary data from the BFILE and no implicit character set conversion is performed. For this reason, using the DBMS_LOB.LOADCLOBFROMFILE procedure is recommended when loading character data, see Loading a CLOB or NCLOB with Data from a BFILE on on page 14-29 for more information.

Specifying Amount of BFILE Data to Load

The value you pass for the amount parameter to functions listed in Table 14–2 must be one of the following:

- An amount less than or equal to the actual size (in bytes) of the BFILE you are loading.
- The maximum allowable LOB size (in bytes).—Passing this value, loads the entire BFILE. You can use this technique to load the entire BFILE without determining the size of the BFILE before loading. To get the maximum allowable LOB size, use the technique described in Table 14–2.

Table 14–2 Maximum LOB Size for Load from File Operations

Environment	Function	To pass maximum LOB size, get value of:
DBMS_LOB	`DBMS_LOB.LOADBLOBFROMFILE`	`DBMS_LOB.LOBMAXSIZE`
DBMS_LOB	`DBMS_LOB.LOADCLOBFROMFILE`	`DBMS_LOB.LOBMAXSIZE`
OCI	`OCILobLoadFromFile2` (For LOBs less than 4 gigabytes in size.)	`UB4MAXVAL`
OCI	`OCILobLoadFromFile2` (For LOBs 4 gigabytes and larger in size.)	`UB8MAXVAL`
OCI	`OCILobLoadFromFile` (For LOBs less than 4 gigabytes in size.)	`UB4MAXVAL`
OCI	`OCILobLoadFromFile` (For LOBs 4 gigabytes and larger in size.)	`UB8MAXVAL`

Syntax

See the following syntax references for details on using this operation in each programmatic environment:

- PL/SQL (DBMS_LOB Package): *PL/SQL Packages and Types Reference* "DBMS_LOB" — LOADFROMFILE.
- C (OCI): *Oracle Call Interface Programmer's Guide* "Relational Functions" — LOB Functions, OCILobLoadFromFile.
- C++ (OCCI): *Oracle C++ Call Interface Programmer's Guide*

- COBOL (Pro*COBOL) *Pro*COBOL 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++ 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): *Oracle Database JDBC Developer's Guide and Reference* Chapter 7, "Working With LOBs" — Creating and Populating a BLOB or CLOB Column.

Examples

Examples are provided in the following programmatic environments:

- PL/SQL (DBMS_LOB): Loading a LOB with Data from a BFILE on page 14-19
- C (OCI): Loading a LOB with Data from a BFILE on page 14-20
- C++ (OCCI): No example is provided with this release.
- COBOL (Pro*COBOL): Loading a LOB with Data from a BFILE on page 14-21
- C/C++ (Pro*C/C++): No example is provided with this release.
- Visual Basic (OO4O): Loading a LOB with Data from a BFILE on page 14-22
- Java (JDBC): Loading a LOB with Data from a BFILE on page 14-23

PL/SQL (DBMS_LOB): Loading a LOB with Data from a BFILE

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/plsql/lloaddat.sql */

/* Procedure loadLOBFromBFILE_proc is not part of the DBMS_LOB package: */

/* loading a lob with bfile data */

CREATE OR REPLACE PROCEDURE loadLOBFromBFILE_proc (Dest_loc IN OUT BLOB) IS
   /* Note: Dest_loc can be a persistent or temporary LOB */
   Src_loc        BFILE := BFILENAME('MEDIA_DIR', 'keyboard_logo.jpg');
   Amount         INTEGER := 4000;
BEGIN
```

```
   DBMS_OUTPUT.PUT_LINE('------------ LOB LOADFORMFILE EXAMPLE ------------');
   /* 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(Dest_loc);
   DBMS_LOB.CLOSE(Src_loc);
END;
/
SHOW ERRORS;
```

C (OCI): Loading a LOB with Data from a BFILE

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/oci/lloaddat.c */
#include <oratypes.h>
#include <lobdemo.h>

void loadLOBDataFromBFile_proc(OCILobLocator *Lob_loc, OCILobLocator* BFile_loc,
                               OCIEnv *envhp,
                               OCIError *errhp, OCISvcCtx *svchp,
                               OCIStmt *stmthp)
{
  oraub8         amount= 2000;

  printf ("----------- OCILobLoadFromFile Demo --------------\n");

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

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

  /* Load the data from the graphic file (bfile) into the blob */
  printf (" load the LOB from File\n");
  checkerr (errhp, OCILobLoadFromFile2(svchp, errhp, Lob_loc, BFile_loc,
                                       amount,
                                       (oraub8)1, (oraub8)1));
```

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

  return;
}
```

COBOL (Pro*COBOL): Loading a LOB with Data from a BFILE

```
      * This file is installed in the following path when you install
      * the database: $ORACLE_HOME/rdbms/demo/lobs/procob/lloaddat.pco

       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 (OO4O): Loading a LOB with Data from a BFILE

```
' This file is installed in the following path when you install
' the database: $ORACLE_HOME/rdbms/demo/lobs/vb/lloaddat.bas
```

```
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.ExecuteSQL ("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

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/java/lloaddat.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 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 a BLOB with Data from a BFILE

This procedure loads a BLOB with data from a BFILE. This procedure can be used to load data into any persistent or temporary BLOB instance.

See Also:

- "Loading a LOB with Data from a BFILE" on page 14-17
- To load character data, use DBMS_LOB.LOADCLOBFROMFILE. See "Loading a CLOB or NCLOB with Data from a BFILE" on page 14-29 for more information.
- As an alternative to this operation, you can use SQL*Loader to load persistent LOBs with data directly from a file in the file system. See "Using SQL*Loader to Load LOBs" on page 3-2 for more information.

Preconditions

The following conditions must be met before calling this procedure:

- The target BLOB instance must exist.
- The source BFILE must exist.
- You must open the BFILE. (After calling this procedure, you must close the BFILE at some point.)

Usage Notes

Note the following with respect to this operation:

New Offsets Returned

Using DBMS_LOB.LOADBLOBFROMFILE to load binary data into a BLOB achieves the same result as using DBMS_LOB.LOADFROMFILE, but also returns the new offsets of BLOB.

Specifying Amount of BFILE Data to Load

The value you pass for the amount parameter to the DBMS_LOB.LOADBLOBFROMFILE function must be one of the following:

- An amount less than or equal to the actual size (in bytes) of the BFILE you are loading.

- The maximum allowable LOB size: `DBMS_LOB.LOBMAXSIZE`
  Passing this value causes the function to load the entire BFILE. This is a useful technique for loading the entire BFILE without introspecting the size of the BFILE.

  See Also: Table 14–2, "Maximum LOB Size for Load from File Operations"

Syntax

See *PL/SQL Packages and Types Reference*, "DBMS_LOB" — LOADBLOBFROMFILE procedure for syntax details on this procedure.

Examples

This example is available in PL/SQL only. This API is not provided in other programmatic environments. This example illustrates:

- How to use `LOADBLOBFROMFILE` to load the entire BFILE without getting its length first.
- How to use the return value of the offsets to calculate the actual amount loaded.

PL/SQL: Loading a BLOB with BFILE Data

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/plsql/lldblobf.sql */

CREATE OR REPLACE PROCEDURE loadBLOB_proc (dst_loc IN OUT BLOB) IS
  src_loc     BFILE := bfilename('MEDIA_DIR','keyboard_logo.jpg') ;
  src_offset  NUMBER := 1;
  dst_offset  NUMBER := 1;
  src_osin    NUMBER;
  dst_osin    NUMBER;
  bytes_rd    NUMBER;
  bytes_wt    NUMBER;
BEGIN
  DBMS_OUTPUT.PUT_LINE('------------ LOB LOADBLOBFORMFILE EXAMPLE
------------');
  /* 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 */

END;
/
```

Loading a CLOB or NCLOB with Data from a BFILE

This procedure loads a CLOB or NCLOB with character data from a BFILE. This procedure can be used to load data into a persistent or temporary CLOB or NCLOB instance.

See Also:

- "Loading a LOB with Data from a BFILE" on page 14-17
- To load binary data, use `DBMS_LOB.LOADBLOBFROMFILE`. See "Loading a BLOB with Data from a BFILE" on page 14-26 for more information.
- As an alternative to this operation, you can use SQL*Loader to load persistent LOBs with data directly from a file in the file system. See "Using SQL*Loader to Load LOBs" on page 3-2 for more information.

Preconditions

The following conditions must be met before calling this procedure:

- The target CLOB or NCLOB instance must exist.
- The source BFILE must exist.
- You must open the BFILE. (After calling this procedure, you must close the BFILE at some point.)

Usage Notes

You can specify the character set id of the BFILE when calling this procedure. Doing so, ensures that the character set is properly converted from the BFILE data character set to the destination CLOB or NCLOB character set.

Specifying Amount of BFILE Data to Load

The value you pass for the amount parameter to the `DBMS_LOB.LOADCLOBFROMFILE` function must be one of the following:

- An amount less than or equal to the actual size (in characters) of the BFILE data you are loading.
- The maximum allowable LOB size: `DBMS_LOB.LOBMAXSIZE`
  Passing this value causes the function to load the entire BFILE. This is a useful

technique for loading the entire BFILE without introspecting the size of the BFILE.

Syntax

See *PL/SQL Packages and Types Reference*, "DBMS_LOB" — LOADCLOBFROMFILE procedure for syntax details on this procedure.

Examples

The following examples illustrate different techniques for using this API:

- "PL/SQL: Loading Character Data from a BFILE into a LOB"
- "PL/SQL: Loading Segments of Character Data into Different LOBs"

PL/SQL: Loading Character Data from a BFILE into a LOB

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.

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/plsql/lldclobf.sql */

CREATE OR REPLACE PROCEDURE loadCLOB1_proc (dst_loc IN OUT CLOB) IS
  src_loc     bfile := bfilename('MEDIA_DIR','monitor_3060.txt') ;
  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_OUTPUT.PUT_LINE('------------ LOB LOADCLOBFORMFILE EXAMPLE
------------');
  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) ;

  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;
/
```

PL/SQL: Loading Segments of Character Data into Different LOBs

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.

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/plsql/lldclobs.sql */

CREATE OR REPLACE PROCEDURE loadCLOB2_proc (dst_loc1 IN OUT NCLOB,
                                             dst_loc2 IN OUT NCLOB) IS
  src_loc     bfile := bfilename('MEDIA_DIR','monitor_3060.txt');
  amt         number := 100;
  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_OUTPUT.PUT_LINE('------------ LOB LOADCLOBFORMFILE EXAMPLE
------------');
  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.LOADCLOBFROMFILE(dst_loc1, src_loc, amt, dst_offset, src_offset,
      cs_id, lang_ctx, warning);

  /* 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(' ----------------------------' ) ;


  /* 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);
  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;
/
```

Determining Whether a LOB is Open

This operation determines whether a LOB is open.

Preconditions

The LOB instance must exist before executing this procedure.

Usage Notes

When a LOB is open, it must be closed at some point later in the session.

Syntax

Use the following syntax references for each programmatic environment:

- PL/SQL (DBMS_LOB Package): *PL/SQL Packages and Types Reference* "DBMS_LOB" — OPEN, ISOPEN.
- C (OCI): *Oracle Call Interface Programmer's Guide* "Relational Functions" — LOB Functions — OCILobIsOpen.
- C++ (OCCI): *Oracle C++ Call Interface Programmer's Guide*
- COBOL (Pro*COBOL) *Pro*COBOL 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++ 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): *Oracle Database JDBC Developer's Guide and Reference* Chapter 7, "Working With LOBs" — Creating and Populating a BLOB or CLOB Column.

Examples

Examples are provided in the following programmatic environments:

- PL/SQL (DBMS_LOB Package): Checking If a LOB Is Open on page 14-35
- C (OCI): Checking If a LOB Is Open on page 14-35
- C++ (OCCI): No example is provided with this release.
- COBOL (Pro*COBOL): Checking If a LOB Is Open on page 14-36
- C/C++ (Pro*C/C++): Checking If a LOB Is Open on page 14-38

- Visual Basic (OO4O): No example is provided with this release.
- Java (JDBC): Checking If a LOB Is Open on page 14-39

PL/SQL (DBMS_LOB Package): Checking If a LOB Is Open

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/plsql/lisopen.sql */

/* Procedure lobIsOpen_proc is not part of the DBMS_LOB package: */

/* seeing if lob is open */

CREATE OR REPLACE PROCEDURE lobIsOpen_proc (Lob_loc IN OUT BLOB) IS
   /* Note: Lob_loc can be a persistent or a temporary LOB */
   Retval      INTEGER;
BEGIN
   DBMS_OUTPUT.PUT_LINE('------------ LOB ISOPEN EXAMPLE ------------');
   /* 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. */

   if Retval = 1 THEN
     DBMS_OUTPUT.PUT_LINE('Input locator is open');
   else
     DBMS_OUTPUT.PUT_LINE('Input locator is not open');
   end if;
END;
/
SHOW ERRORS;
```

C (OCI): Checking If a LOB Is Open

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/oci/lisopen.c */

/* Checking if LOB is Open. */
#include <oratypes.h>
#include <lobdemo.h>
void seeIfLOBIsOpen_proc(OCILobLocator *Lob_loc, OCIEnv *envhp,
                         OCIError *errhp, OCISvcCtx *svchp, OCIStmt *stmthp)
```

```
{
  boolean isOpen;

  printf ("----------- OCILobIsOpen Demo --------------\n");
  /* 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 */
  }
  return;
}
```

COBOL (Pro*COBOL): Checking If a LOB Is Open

```
 * This file is installed in the following path when you install
 * the database: $ORACLE_HOME/rdbms/demo/lobs/procob/lisopen.pco

* 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++ (Pro*C/C++): Checking If a LOB Is Open

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/proc/lisopen.pc */

/* 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 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 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

```
/* This file is installed in the following path when you install */
```

```
/* the database: $ORACLE_HOME/rdbms/demo/lobs/java/lisopen.java */

// 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.prepareCall (
        "BEGIN DBMS_LOB.OPEN(?,DBMS_LOB.LOB_READONLY); END;");
    cstmt2.setBLOB(1, blob);
    cstmt2.execute();

    System.out.println("The LOB has been opened with a call to DBMS_LOB.OPEN()");

    // Use the existing cstmt handle to re-query the status of the locator:
    cstmt.setBLOB(2, blob);
    cstmt.execute();
    result = cstmt.getInt(1);
    System.out.println("This result is: " + Integer.toString(result));

    stmt.close();
    cstmt.close();
    cstmt2.close();
    conn.commit();
    conn.close();
     }
     catch (SQLException e)
     {
       e.printStackTrace();
     }
   }
}
```

Displaying LOB Data

This section describes APIs that allow you to read LOB data. You can use this operation to read LOB data into a buffer. This is useful if your application requires displaying large amounts of LOB data or streaming data operations.

Usage Notes

Note the following when using these APIs.

Streaming Mechanism

The most efficient way to read large amounts of LOB data is to use `OCILobRead2`() with the streaming mechanism enabled.

Amount Parameter

The value you pass for the amount parameter is restricted for the APIs described in Table 14–3.

Table 14–3 Maximum LOB Size for Amount Parameter

Environment	Function	Value of amount parameter is limited to:
DBMS_LOB	`DBMS_LOB.READ`	The size of the buffer, 32K bytes.
OCI	`OCILobRead` (For LOBs less than 4 gigabytes in size.)	`UB4MAXVAL` Specifying this amount reads the entier file.
OCI	`OCILobRead2` (For LOBs 4 gigabytes and larger in size.)	`UB8MAXVAL` Specifying this amount reads the entier file.

Syntax

Use the following syntax references for each programmatic environment:

- PL/SQL (DBM_LOB Package): *PL/SQL Packages and Types Reference* "DBMS_LOB" — OPEN, READ, CLOSE.
- C (OCI): *Oracle Call Interface Programmer's Guide* "Relational Functions" — LOB Functions, OCILobOpen, OCILobRead2, OCILobClose.
- C++ (OCCI): *Oracle C++ Call Interface Programmer's Guide*

- COBOL (Pro*COBOL) *Pro*COBOL 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++ 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): *Oracle Database JDBC Developer's Guide and Reference* Chapter 7, "Working With LOBs" — Creating and Populating a BLOB or CLOB Column.

Examples

Examples are provided in the following programmatic environments:

- PL/SQL (DBMS_LOB Package): Displaying LOB Data on page 14-43
- C (OCI): Displaying LOB Data on page 14-44
- C++ (OCCI): No example is provided in this release.
- COBOL (Pro*COBOL): Displaying LOB Data on page 14-46
- C/C++ (Pro*C/C++): Displaying LOB Data on page 14-47
- Visual Basic (OO4O): Displaying LOB Data on page 14-49
- Java (JDBC): Displaying LOB Data on page 14-49

PL/SQL (DBMS_LOB Package): Displaying LOB Data

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/plsql/ldisplay.sql */

/* Procedure displayLOB_proc is not part of the DBMS_LOB package: */

/* displaying lob data */

CREATE OR REPLACE PROCEDURE displayLOB_proc (Lob_loc IN OUT BLOB) IS
    /* Note: Lob_loc can be a persistent or temporary LOB */
Buffer   RAW(1024);
Amount   BINARY_INTEGER := 1024;
Position INTEGER := 1;
BEGIN
```

```
   DBMS_OUTPUT.PUT_LINE('------------ LOB DATA DISPLAY EXAMPLE ------------');
   /* 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(rawtohex(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_LOB.CLOSE (Lob_loc);
END;
/

SHOW ERRORS;
```

C (OCI): Displaying LOB Data

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/oci/ldisplay.c */

/* Displaying LOB data. This example reads the entire contents of a CLOB
   piecewise into a buffer using the standard polling method, processing
   each buffer piece after every READ operation until the entire CLOB
   has been read. */
#include <oratypes.h>
#include <lobdemo.h>
void displayLOB_proc(OCILobLocator *Lob_loc, OCIEnv *envhp,
                     OCIError *errhp, OCISvcCtx *svchp, OCIStmt *stmthp)
{
  oraub8 amt;
  ub4 offset;
  sword retval;
  boolean done;
  ub1 bufp[MAXBUFLEN];
  ub4 buflen;
  ub1 piece;
```

```
 printf ("----------- LOB Data Display Demo --------------\n");
 /* Open the CLOB */
 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((void *)bufp, '\0', MAXBUFLEN);
 done = FALSE;
 piece = OCI_FIRST_PIECE;
 while (!done)
 {
   retval = OCILobRead2(svchp, errhp, Lob_loc, &amt, NULL, offset,
                        (void *) bufp, buflen, piece, (void *)0,
                        (OCICallbackLobRead2) 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 in bytes.
      */
     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 in bytes.
      */
     piece = OCI_NEXT_PIECE;
     break;
   default:
     checkerr (errhp, retval);
     done = TRUE;
     break;
   }
 } /* while */
```

```
  /* Closing the CLOB is mandatory if you have opened it */
   printf(" close the lob \n");
   checkerr (errhp, OCILobClose(svchp, errhp, Lob_loc));
}
```

COBOL (Pro*COBOL): Displaying LOB Data

```
      * This file is installed in the following path when you install
      * the database: $ORACLE_HOME/rdbms/demo/lobs/procob/ldisplay.pco

     * 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++ (Pro*C/C++): Displaying LOB Data

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/proc/ldisplay.pc */

/* 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 (OO4O): Displaying LOB Data

```
' This file is installed in the following path when you install
' the database: $ORACLE_HOME/rdbms/demo/lobs/vb/ldisplay.bas

'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 chnunk
Loop Until OraAdSourceText.Status <> ORALOB_NEED_DATA
```

Java (JDBC): Displaying LOB Data

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/java/ldisplay.java */

// 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

This section describes how to read data from LOBs.

Usage Notes

Note the following when using this operation.

Streaming Read in OCI

The most efficient way to read of large amounts of LOB data is to use `OCILobRead2()` with the streaming mechanism enabled using polling or callback. To do so, specify the starting point of the read using the offset parameter and use the symbol `OCI_LOBMAXSIZE` for the amount, as follows:

```
ub8  char_amt =  20;
ub8  byte_amt = 20;
ub4  offset = 1000;

OCILobRead2(svchp, errhp, locp, &byte_amt, &char_amt, offset, bufp, bufl,
OCI_ONE_PIECE,
0, 0, 0, 0)
```

When using *polling mode*, be sure to look at the value of the '`amount`' parameter after each `OCILobRead2`() call to see how many bytes were read into the buffer because 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 because 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 run `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, then 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

Use the following syntax references for each programmatic environment:

- PL/SQL (DBM_LOB Package): *PL/SQL Packages and Types Reference* "DBMS_LOB" — OPEN, GETCHUNKSIZE, READ, CLOSE.
- C (OCI): *Oracle Call Interface Programmer's Guide* Chapter 7, " "Relational Functions" — LOB Functions, OciLobOpen, OCILobRead2, OciLobClose.
- C++ (OCCI): *Oracle C++ Call Interface Programmer's Guide*
- COBOL (Pro*COBOL) *Pro*COBOL 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++ 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): *Oracle Database JDBC Developer's Guide and Reference* Chapter 7, "Working With LOBs" — Creating and Populating a BLOB or CLOB Column.

Examples

Examples are provided in the following programmatic environments:

- PL/SQL (DBMS_LOB Package): Reading Data from a LOB on page 14-54
- C (OCI): Reading Data from a LOB on page 14-54
- COBOL (Pro*COBOL): Reading Data from a LOB on page 14-58
- COBOL (Pro*COBOL): Reading Data from a LOB on page 14-58
- C (OCI): Reading Data from a LOB on page 14-54
- Visual Basic (OO4O): Reading Data from a LOB on page 14-60
- Java (JDBC): Reading Data from a LOB on page 14-61

PL/SQL (DBMS_LOB Package): Reading Data from a LOB

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/plsql/lread.sql */

/* Procedure readLOB_proc is not part of the DBMS_LOB package: */

/* reading data from lob */

CREATE OR REPLACE PROCEDURE readLOB_proc (Lob_loc IN OUT BLOB) IS
    /* Note: Lob_loc can be a persistent or a temporary LOB */
    Buffer            RAW(32767);
    Amount            BINARY_INTEGER := 32767;
    Position          INTEGER := 2;
BEGIN
   DBMS_OUTPUT.PUT_LINE('------------ LOB READ EXAMPLE ------------');
   /* 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);

   DBMS_OUTPUT.PUT_LINE(RAWTOHEX(substr(Buffer, 1, 200)));
END;
/

SHOW ERRORS;
```

C (OCI): Reading Data from a LOB

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/oci/lread.c */

/* Reading LOB data. 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 <oratypes.h>
```

```
#include <lobdemo.h>


static void pollingRead(OCILobLocator *Lob_loc, OCIEnv *envhp,
                        OCIError *errhp, OCISvcCtx *svchp, OCIStmt *stmthp)
{
  oraub8 amt;
  oraub8 offset;
  sword retval;
  ub1 bufp[MAXBUFLEN];
  oraub8 buflen;
  boolean done;
  ub1 piece;

  buflen = (oraub8)MAXBUFLEN;
  /* Open the CLOB */
  printf(" call OCILobOpen\n");
  checkerr (errhp, OCILobOpen(svchp, errhp, Lob_loc, OCI_LOB_READONLY));

  printf ("  ---------- OCILobRead one-piece mode  -------------\n");
  offset=1;
  amt = 5;
  checkerr(errhp, OCILobRead2(svchp, errhp, Lob_loc, NULL, &amt, offset,
                              (void *)bufp, buflen, OCI_ONE_PIECE, (void *)0,
                              (OCICallbackLobRead2) 0,
                              (ub2) 0, (ub1) SQLCS_IMPLICIT));
  printf ("  amt read= %d\n", (ub4)amt);
  printf ("  ---------- OCILobRead polling mode  -------------\n");
  /* Setting the amt to the buffer length.  Note here that amt is in chars
     since we are using a CLOB  */
  amt = 0;

 /* Process the data in pieces */
  printf("  process the data in pieces\n");
  offset = 1;
  memset((void *)bufp, '\0', MAXBUFLEN);
  done = FALSE;
  piece = OCI_FIRST_PIECE;

  while (!done)
  {
  retval = OCILobRead2(svchp, errhp, Lob_loc, NULL, &amt, offset,
                       (void *) bufp, buflen, piece, (void *)0,
                       (OCICallbackLobRead2) 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 in bytes. */
      printf("  amt read=%d in the last call\n", (ub4)amt);
      done = TRUE;
     break;
    case OCI_ERROR:
      /*   report_error();        this function is not shown here */
      done = TRUE;
      break;
    case OCI_NEED_DATA:
      printf("  amt read=%d\n", (ub4)amt);
      piece = OCI_NEXT_PIECE;
      break;
    default:
      (void) printf("Unexpected ERROR: OCILobRead2() LOB.\n");
       done = TRUE;
       break;
    }
  }
  /* Closing the CLOB is mandatory if you have opened it */
  checkerr (errhp, OCILobClose(svchp, errhp, Lob_loc));
}

struct somestruct
{
  ub4 count;
};

sb4 lobCallback(dvoid *ctxp, CONST dvoid *bufp, oraub8 len, ub1 piece,
                dvoid **changed_bufpp, oraub8 *changed_lenp)
{
  struct somestruct *ctx = (struct somestruct *)ctxp;
  printf("  In callback, count = %d, len passed in=%d, piece=%d\n",
         ctx->count++,(ub4)len, piece);

  return OCI_CONTINUE;
}

static void callbackRead(OCILobLocator *Lob_loc, OCIEnv *envhp,
                         OCIError *errhp, OCISvcCtx *svchp, OCIStmt *stmthp)
{
  oraub8 amt;
```

```
  oraub8 offset;
  sword retval;
  ub1 bufp[MAXBUFLEN];
  oraub8 buflen;
  boolean done;

  struct somestruct ctx;

  printf ("  ---------- OCILobRead Callback mode  -------------\n");

  /* Open the CLOB */
  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 chars
     since we are using a CLOB  */
  amt = 0;
  buflen =(oraub8) sizeof(bufp);

 /* Process the data in pieces */
  printf("  process the data in pieces\n");
  offset = 1;
  memset((void *)bufp, '\0', MAXBUFLEN);
  done = FALSE;

  ctx.count = 0;
  checkerr (errhp, OCILobRead2(svchp, errhp, Lob_loc, NULL, &amt, offset,
                      (void *) bufp, buflen, OCI_FIRST_PIECE,
                      (void *)&ctx, lobCallback,
                      (ub2) 0, (ub1) SQLCS_IMPLICIT));

  printf("  total amt read=%d in OCILobRead2 callback mode\n", (ub4)amt);
  /* Closing the CLOB is mandatory if you have opened it */
  checkerr (errhp, OCILobClose(svchp, errhp, Lob_loc));
}

void readLOB_proc(OCILobLocator *Lob_loc, OCIEnv *envhp,
                  OCIError *errhp, OCISvcCtx *svchp, OCIStmt *stmthp)

{
  printf ("----------- OCILobRead Demo  --------------\n");
  /* One-piece and Polling mode */
  pollingRead(Lob_loc, envhp, errhp, svchp, stmthp);
  /* Callback mode */
  callbackRead(Lob_loc, envhp, errhp, svchp, stmthp);
}
```

COBOL (Pro*COBOL): Reading Data from a LOB

```
 * This file is installed in the following path when you install
 * the database: $ORACLE_HOME/rdbms/demo/lobs/procob/lread.pco

* 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

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/proc/lread.pc */

/* 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

```
' This file is installed in the following path when you install
' the database: $ORACLE_HOME/rdbms/demo/lobs/vb/lread.bas

'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

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/java/lread.java */

// 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 a LOB (SUBSTR)

This section describes how to read a portion of a LOB using SUBSTR.

Syntax

Use the following syntax references for each programmatic environment:

- PL/SQL (DBM_LOB Package): *PL/SQL Packages and Types Reference* "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 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++ 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 > Orablob > PROPERTIES > offset, chunksize, and >OBJECTS > Oraclob > METHODS > read
- Java (JDBC): *Oracle Database JDBC Developer's Guide and Reference* Chapter 7, "Working With LOBs" — Creating and Populating a BLOB or CLOB Column.

Examples

Examples are provided in the following programmatic environments:

- PL/SQL (DBMS_LOB Package): Reading a Portion of the LOB (substr) on page 14-64
- 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 14-64
- C/C++ (Pro*C/C++): Reading a Portion of the LOB (substr) on page 14-66
- Visual Basic (OO4O): Reading a Portion of the LOB (substr) on page 14-67
- Java (JDBC): Reading a Portion of the LOB (substr) on page 14-68

PL/SQL (DBMS_LOB Package): Reading a Portion of the LOB (substr)

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/plsql/lsubstr.sql */

/* Procedure substringLOB_proc is not part of the DBMS_LOB package: */

/* reading portion of lob (substr) */

CREATE OR REPLACE PROCEDURE substringLOB_proc (Lob_loc IN OUT BLOB) IS
    /* Note: Lob_loca can be a persistent or a temporary LOB */
    Amount            BINARY_INTEGER := 32767;
    Position          INTEGER := 3;
    Buffer            RAW(32767);
BEGIN
    DBMS_OUTPUT.PUT_LINE('------------ LOB SUBSTR EXAMPLE ------------');
    /* 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);
    DBMS_OUTPUT.PUT_LINE(rawtohex(substr(buffer,1,200)));
END;
/

SHOW ERRORS;

/* For persistent LOBs, DBMS_LOB.SUBSTR can be used in a SQL statement too.
   In the following SQL statement, 200 is the amount to read
   and 1 is the starting offset from which to read: */
SELECT DBMS_LOB.SUBSTR(ad_sourcetext, 200, 1) FROM print_media
WHERE product_id = 2268;
```

COBOL (Pro*COBOL): Reading a Portion of the LOB (substr)

```
       * This file is installed in the following path when you install
       * the database: $ORACLE_HOME/rdbms/demo/lobs/procob/lsubstr.pco

      * 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 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.
     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)

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/proc/lsubstr.pc */

/* 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)

```
' This file is installed in the following path when you install
' the database: $ORACLE_HOME/rdbms/demo/lobs/vb/lsubstr.bas

'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)

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/java/lsubstr.java */

// 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

This section describes how to compare all or part of two LOBs.

Syntax

Use the following syntax references for each programmatic environment:

- PL/SQL (DBM_LOB Package): *PL/SQL Packages and Types Reference* "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 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++ 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): *Oracle Database JDBC Developer's Guide and Reference* Chapter 7, "Working With LOBs" — Creating and Populating a BLOB or CLOB Column.

Examples

Examples are provided in the following programmatic environments:

- PL/SQL (DBMS_LOB Package): Comparing All or Part of Two LOBs on page 14-72
- 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 14-72
- C/C++ (Pro*C/C++): Comparing All or Part of Two LOBs on page 14-74
- Visual Basic (OO4O): Comparing All or Part of Two LOBs on page 14-76
- Java (JDBC): Comparing All or Part of Two LOBs on page 14-76

PL/SQL (DBMS_LOB Package): Comparing All or Part of Two LOBs

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/plsql/lcompare.sql */

/* Procedure compareTwoLOBs_proc is not part of the DBMS_LOB package: */

/* comparing all or part of lob */

CREATE OR REPLACE PROCEDURE compareTwoLOBs_proc
    (Lob_loc1 IN OUT BLOB, Lob_loc2 IN OUT BLOB) IS
    /* Note: Lob_loc1 and Lob_loc2 can be persistent or temporary LOBs */
    Amount              INTEGER := 32767;
    Retval              INTEGER;
BEGIN
    DBMS_OUTPUT.PUT_LINE('------------ LOB COMPARE EXAMPLE ------------');
    /* 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 LOBs: */
    retval := DBMS_LOB.COMPARE(Lob_loc1, Lob_loc2, Amount, 1, 1);
    IF retval = 0 THEN
       DBMS_OUTPUT.PUT_LINE('LOBs are equal');
    ELSE
       DBMS_OUTPUT.PUT_LINE('LOBs are not equal');
    END IF;
    /* Closing the LOB is mandatory if you have opened it: */
    DBMS_LOB.CLOSE (Lob_loc1);
    DBMS_LOB.CLOSE (Lob_loc2);
END;
/

SHOW ERRORS;
```

COBOL (Pro*COBOL): Comparing All or Part of Two LOBs

```
      * This file is installed in the following path when you install
      * the database: $ORACLE_HOME/rdbms/demo/lobs/procob/lcompare.pco

      * 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
                SELECT AD_COMPOSITE INTO :BLOB1
                FROM PRINT_MEDIA PM WHERE PM.PRODUCT_ID = 2268  AND AD_ID =
21001
             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 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++): Comparing All or Part of Two LOBs

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/proc/lcompare.pc */

/* Comparing all or part of two 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 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

```
' This file is installed in the following path when you install
' the database: $ORACLE_HOME/rdbms/demo/lobs/vb/lcompare.bas

'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, OraAdPhoto1 as OraBLOB, OraAdPhotoClone as OraBLOB

Set OraDyn = OraDb.CreateDynaset(
   "SELECT * FROM Print_media ORDER BY product_id, ad_id", ORADYN_DEFAULT)
Set OraAdPhoto1 = OraDyn.Fields("ad_photo").Value
'Clone it for future reference
Set OraAdPhotoClone = OraAdPhoto1.Clone

'Lets go to the next row and compare LOBs
OraDyn.MoveNext

MsgBox CBool(OraAdPhotot1.Compare(OraAdPhototClone, OraAdPhotoClone.size, 1, 1))
```

Java (JDBC): Comparing All or Part of Two LOBs

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/java/lcompare.java */

// 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 a LOB Using INSTR

This section describes how to see if a pattern exists in a LOB using INSTR.

Syntax

Use the following syntax references for each programmatic environment:

- PL/SQL (DBM_LOB Package): *PL/SQL Packages and Types Reference* "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 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++ 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): *Oracle Database JDBC Developer's Guide and Reference* Chapter 7, "Working With LOBs" — Creating and Populating a BLOB or CLOB Column.

Examples

Examples are provided in the following programmatic environments:

- PL/SQL (DBMS_LOB Package): Checking for Patterns in a LOB (instr) on page 14-80
- 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 a LOB (instr) on page 14-80
- C/C++ (Pro*C/C++): Checking for Patterns in a LOB (instr) on page 14-82
- Visual Basic (OO4O): No example is provided with this release.
- Java (JDBC): Checking for Patterns in a LOB (instr) on page 14-83

PL/SQL (DBMS_LOB Package): Checking for Patterns in a LOB (instr)

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/plsql/linstr.sql */

/* Procedure instringLOB_proc is not part of the DBMS_LOB package: */

/* seeing if pattern exists in lob (instr) */

CREATE OR REPLACE PROCEDURE instringLOB_proc (Lob_loc IN OUT BLOB) IS
   /* Note: Lob_loccan be a persistent or temporary LOB */
   Pattern        RAW(30) := hextoraw('aabb');
   Position       INTEGER := 0;
   Offset         INTEGER := 1;
   Occurrence     INTEGER := 1;
BEGIN
   DBMS_OUTPUT.PUT_LINE('------------ LOB INSTR EXAMPLE ------------');
   /* 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;
/

SHOW ERRORS;
```

COBOL (Pro*COBOL): Checking for Patterns in a LOB (instr)

```
      * This file is installed in the following path when you install
      * the database: $ORACLE_HOME/rdbms/demo/lobs/procob/linstr.pco

     * 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 a LOB (instr)

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/proc/linstr.pc */

/* 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;
  instringLOB_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

Java (JDBC): Checking for Patterns in a LOB (instr)

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/java/linstr.java */

// 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

This section describes how to determine the length of a LOB.

Syntax

Use the following syntax references for each programmatic environment:

- PL/SQL (DBM_LOB Package): *PL/SQL Packages and Types Reference* "DBMS_ LOB" — GETLENGTH
- C (OCI): *Oracle Call Interface Programmer's Guide* "Relational Functions" — LOB Functions, OCILobGetLength2.
- C++ (OCCI): *Oracle C++ Call Interface Programmer's Guide*
- COBOL (Pro*COBOL) *Pro*COBOL 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++ 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): *Oracle Database JDBC Developer's Guide and Reference* Chapter 7, "Working With LOBs" — Creating and Populating a BLOB or CLOB Column.

Examples

Examples are provided in the following programmatic environments:

- PL/SQL (DBMS_LOB Package): Determining the Length of a LOB on page 14-87
- C (OCI): Determining the Length of a LOB on page 14-87
- C++ (OCCI): No example is provided with this release.
- COBOL (Pro*COBOL): Determining the Length of a LOB on page 14-88
- C/C++ (Pro*C/C++): Determining the Length of a LOB on page 14-89
- Visual Basic (OO4O): Determining the Length of a LOB on page 14-90
- Java (JDBC): Determining the Length of a LOB on page 14-91

PL/SQL (DBMS_LOB Package): Determining the Length of a LOB

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/plsql/llength.sql */

/* Procedure getLengthLOB_proc is not part of the DBMS_LOB package: */

/* getting the length of a lob */

CREATE OR REPLACE PROCEDURE getLengthLOB_proc (Lob_loc IN OUT BLOB) IS
   /* Note: Lob_loc can be a persistent or temporary LOB */
   Length      INTEGER;
BEGIN
   DBMS_OUTPUT.PUT_LINE('------------ LOB GETLENGTH EXAMPLE ------------');
   /* 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;
/
SHOW ERRORS;
```

C (OCI): Determining the Length of a LOB

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/oci/llength.c */

/* Getting the length of a LOB */
#include <oratypes.h>
#include <lobdemo.h>

/* This function gets the length of the LOB */
void getLengthLOB_proc(OCILobLocator *Lob_loc, OCIEnv *envhp,
                       OCIError *errhp, OCISvcCtx *svchp, OCIStmt *stmthp)
```

```
{
  oraub8 length;

  printf("----------- OCILobGetLength Demo --------------\n");
  /* 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_fltextn.\n");
  checkerr (errhp, OCILobGetLength2(svchp, errhp, Lob_loc, &length));

  /* Length is undefined if the LOB is NULL or undefined */
  printf(" Length of LOB is %d\n",(ub4)length);

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

  return;
}
```

COBOL (Pro*COBOL): Determining the Length of a LOB

```
  * This file is installed in the following path when you install
  * the database: $ORACLE_HOME/rdbms/demo/lobs/procob/llength.pco

 * 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 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++): Determining the Length of a LOB

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/proc/llength.pc */

/* Getting the length 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 getLengthLOB_proc()
{
  OCIClobLocator *Lob_loc;
  unsigned int Length;

  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 READ ONLY;
  /* Get the Length: */
  EXEC SQL LOB DESCRIBE :Lob_loc GET LENGTH INTO :Length;
  /* If the LOB is NULL or unitialized, then Length is Undefined: */
  printf("Length is %d characters\n", Length);
  /* Closing the LOB is mandatory if you have Opened it: */
  EXEC SQL LOB CLOSE :Lob_loc;
  EXEC SQL FREE :Lob_loc;
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  getLengthLOB_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

Visual Basic (OO4O): Determining the Length of a LOB

```
' This file is installed in the following path when you install
' the database: $ORACLE_HOME/rdbms/demo/lobs/vb/llength.bas

'Getting the length 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, ad_id", ORADYN_DEFAULT)
Set OraAdPhoto1 = OraDyn.Fields("ad_photo").Value

'Display out size of the lob:
MsgBox "Length of the lob is " & OraAdPhoto1.Size
```

Java (JDBC): Determining the Length of a LOB

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/java/llength.java */

//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

This section describes how to copy all or part of a LOB to another LOB. These APIs copy an amount of data you specify from a source LOB to a destination LOB.

Usage Notes

Note the following issues when using this API.

Specifying Amount of Data to Copy

The value you pass for the `amount` parameter to the `DBMS_LOB.COPY` function must be one of the following:

- An amount less than or equal to the actual size of the data you are loading.
- The maximum allowable LOB size: `DBMS_LOB.LOBMAXSIZE`
  Passing this value causes the function to read the entire LOB. This is a useful technique for reading the entire LOB without introspecting the size of the LOB.

Note that for character data, the amount is specified in characters, while for binary data, the amount is specified in bytes.

Locking the Row Prior to Updating

If you plan to update a LOB value, then you must lock the row containing the LOB prior to updating. 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 Through Updated Locators" on page 5-16.

Syntax

See the following syntax references for each programmatic environment:

- PL/SQL (DBM_LOB Package): *PL/SQL Packages and Types Reference* "DBMS_LOB" — COPY
- C (OCI): *Oracle Call Interface Programmer's Guide* "Relational Functions" — LOB Functions, OCILobCopy2
- C++ (OCCI): *Oracle C++ Call Interface Programmer's Guide*

- COBOL (Pro*COBOL) *Pro*COBOL 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++ 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): *Oracle Database JDBC Developer's Guide and Reference* Chapter 7, "Working With LOBs" — Creating and Populating a BLOB or CLOB 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 14-94
- C (OCI): Copying All or Part of One LOB to Another LOB on page 14-95
- COBOL (Pro*COBOL): Copying All or Part of One LOB to Another LOB on page 14-96
- C/C++ (Pro*C/C++): Copying All or Part of a LOB to Another LOB on page 14-98
- Visual Basic (OO4O): Copying All or Part of One LOB to Another LOB on page 14-100
- Java (JDBC): Copying All or Part of One LOB to Another LOB on page 14-100

PL/SQL (DBMS_LOB Package): Copying All or Part of One LOB to Another LOB

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/plsql/lcopy.sql */

/* Procedure copyLOB_proc is not part of the DBMS_LOB package: */

/* copying all or part of a lob to another lob */

CREATE OR REPLACE PROCEDURE copyLOB_proc
   (Dest_loc IN OUT BLOB, Src_loc IN OUT BLOB) IS
```

```
   /* Note: Dest_loc and Src_loc can be persistent or temporary LOBs */
   Amount       NUMBER := 1;
   Dest_pos     NUMBER := 3;
   Src_pos      NUMBER := 2;
BEGIN
   DBMS_OUTPUT.PUT_LINE('------------ LOB COPY EXAMPLE ------------');
   /* 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);
   DBMS_OUTPUT.PUT_LINE('Copy succeeded');
EXCEPTION
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('Copy failed');
      DBMS_LOB.CLOSE(Dest_loc);
      DBMS_LOB.CLOSE(Src_loc);
END;
/
SHOW ERRORS;
```

C (OCI): Copying All or Part of One LOB to Another LOB

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/oci/lcopy.c */

/* This function copies part of the Source LOB into a specified position
   in the destination LOB
 */
#include <oratypes.h>
#include <lobdemo.h>

void copyAllPartLOB_proc(OCILobLocator *Lob_loc1, OCILobLocator *Lob_loc2,
                    OCIEnv *envhp, OCIError *errhp, OCISvcCtx *svchp,
                    OCIStmt *stmthp)
{
  oraub8 Amount = 100;                                 /* <Amount to Copy> */
  oraub8 Dest_pos = 100;                  /*<Position to start copying into> */
```

```
  oraub8 Src_pos = 1;                    /* <Position to start copying from> */
  printf ("----------- OCILobCopy Demo --------------\n");

  /* Opening the LOBs is Optional */
  printf (" open the destination locator (optional)\n");
  checkerr (errhp, OCILobOpen(svchp, errhp, Lob_loc2, OCI_LOB_READWRITE));
  printf (" open the source locator (optional)\n");
  checkerr (errhp, OCILobOpen(svchp, errhp, Lob_loc1, OCI_LOB_READONLY));

  printf (" copy the lob (amount) from the source to destination\n");
  checkerr (errhp, OCILobCopy2(svchp, errhp, Lob_loc2, Lob_loc1,
                             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, Lob_loc2));
  checkerr (errhp, OCILobClose(svchp, errhp, Lob_loc1));

  return;
}
```

COBOL (Pro*COBOL): Copying All or Part of One LOB to Another LOB

```
     * This file is installed in the following path when you install
     * the database: $ORACLE_HOME/rdbms/demo/lobs/procob/lcopy.pco

    * 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 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++): Copying All or Part of a LOB to Another LOB

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/proc/lcopy.pc */

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

void copyLOB_proc()
{
  OCIBlobLocator *Dest_loc, *Src_loc;
  int Amount = 5;
  int Dest_pos = 10;
  int Src_pos = 1;

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  /* Allocate the LOB locators: */
  EXEC SQL ALLOCATE :Dest_loc;
  EXEC SQL ALLOCATE :Src_loc;
  /* Select the LOBs: */
  EXEC SQL SELECT 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

```
' This file is installed in the following path when you install
' the database: $ORACLE_HOME/rdbms/demo/lobs/vb/lcopy.bas

'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

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/java/lcopy.java */

// 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

This section describes how to copy a LOB locator. Note that different locators may point to the same or different data, or to current or outdated data.

Syntax

Use the following syntax references for each programmatic environment:

- PL/SQL (DBM_LOB Package): Refer to "Read Consistent Locators" on page 5-13 for information on assigning one lob locator to another.
- C (OCI): *Oracle Call Interface Programmer's Guide* "Relational Functions" — LOB Functions, OciLobAssign, OciLobIsEqual.
- C++ (OCCI): *Oracle C++ Call Interface Programmer's Guide*
- COBOL (Pro*COBOL) *Pro*COBOL 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++ 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): *Oracle Database JDBC Developer's Guide and Reference* Chapter 7, "Working With LOBs" — Creating and Populating a BLOB or CLOB Column.

Examples

Examples are provided in the following programmatic environments:

- PL/SQL (DBMS_LOB Package): Copying a LOB Locator on page 14-104
- C (OCI): Copying a LOB Locator on page 14-104

- C++ (OCCI): No example is provided with this release.

- COBOL (Pro*COBOL): Copying a LOB Locator on page 14-105
- C/C++ (Pro*C/C++): Copying a LOB Locator on page 14-107
- Visual Basic (OO4O): Copying a LOB Locator on page 14-108
- Java (JDBC): Copying a LOB Locator on page 14-109

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 in "Read Consistent Locators" on page 5-13.

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/plsql/lcopyloc.sql */

/* Procedure lobAssign_proc is not part of the DBMS_LOB package. */

/* copying a lob locator */

CREATE OR REPLACE PROCEDURE lobAssign_proc (Lob_loc1 IN OUT BLOB) IS
   /* Note that Lob_loc1 can be a persistent or temporary LOB */
  Lob_loc2    BLOB;
BEGIN
   DBMS_OUTPUT.PUT_LINE('------------ LOB ASSIGN EXAMPLE ------------');
   /* 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;
/
SHOW ERRORS;
```

C (OCI): Copying a LOB Locator

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/oci/lcopyloc.c */

/* Copying a LOB locator */
#include <oratypes.h>
#include <lobdemo.h>
void assignLOB_proc(OCILobLocator *Lob_loc1, OCILobLocator *Lob_loc2,
                    OCIEnv *envhp, OCIError *errhp, OCISvcCtx *svchp,
                    OCIStmt *stmthp)
```

```
{
  boolean       isEqual;
  printf ("----------- OCILobAssign Demo --------------\n");

  /* Assign Lob_loc1 to Lob_loc2 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, OCILobLocatorAssign(svchp, errhp, Lob_loc1, &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.
   */

  /* Call OCI to see if the two locators are Equal */

  printf (" check if Lobs are Equal.\n");
  checkerr (errhp, OCILobIsEqual(envhp, Lob_loc1, Lob_loc2, &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 */
  return;
}
```

COBOL (Pro*COBOL): Copying a LOB Locator

```
  * This file is installed in the following path when you install
  * the database: $ORACLE_HOME/rdbms/demo/lobs/procob/lcopyloc.pco

 * 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 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++): Copying a LOB Locator

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/proc/lcopyloc.pc */

/* Copying a LOB locator */
#include <oci.h>
#include <stdio.h>
#include <sqlca.h>

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

void lobAssign_proc()
{
  OCIBlobLocator *Lob_loc1, *Lob_loc2;

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  EXEC SQL ALLOCATE :Lob_loc1;
  EXEC SQL ALLOCATE :Lob_loc2;
  EXEC SQL SELECT 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

```
' This file is installed in the following path when you install
' the database: $ORACLE_HOME/rdbms/demo/lobs/vb/lcopyloc.bas

'Copying a LOB locator
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

OraDyn.MoveNext

'Copy 1000 bytes of LOB values OraAdPhotoClone to OraAdPhotol at OraAdPhotol
'offset 100:
OraDyn.Edit
OraAdPhoto1.Copy OraAdPhotoClone, 1000, 100
OraDyn.Update
```

Java (JDBC): Copying a LOB Locator

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/java/lcopyloc.java */

// 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

This section describes how to determine whether one LOB locator is equal to another. If two locators are equal, then this means that they refer to the same version of the LOB data.

See Also:

- Table 14–1, "Environments Supported for Basic LOB APIs" on page 14-2
- "Read Consistent Locators" on page 5-13

Syntax

Use the following syntax references for each programmatic environment:

- C (OCI): *Oracle Call Interface Programmer's Guide* "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++ 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): *Oracle Database JDBC Developer's Guide and Reference* Chapter 7, "Working With LOBs" — Creating and Populating a BLOB or CLOB Column.

Examples

Examples are provided in the following programmatic environments:

- C (OCI): Checking If One LOB Locator Is Equal to Another on page 14-111
- 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 14-113
- Visual Basic (OO4O): No example is provided with this release.
- Java (JDBC): Checking If One LOB Locator Is Equal to Another on page 14-115

C (OCI): Checking If One LOB Locator Is Equal to Another

```
/* This is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/oci/lequal.c */

/* Seeing if One LOB locator is Equal to Another */
#include <oratypes.h>
#include <lobdemo.h>
void locatorIsEqual_proc(OCILobLocator *Lob_loc1, OCILobLocator *Lob_loc2,
                         OCIEnv *envhp, OCIError *errhp, OCISvcCtx *svchp,
                         OCIStmt *stmthp)
{
  boolean       isEqual;
  OCILobLocator *Lob_loc3;

  printf ("----------- OCILobIsEqual Demo --------------\n");


  /* Call OCI to see if the two locators are Equal: */
  printf (" check if Lob1 and Lob2 are Equal.\n");
  checkerr (errhp, OCILobIsEqual(envhp, Lob_loc1, Lob_loc2, &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");
  }

  (void)OCIDescriptorAlloc((void *) envhp, (void **) &Lob_loc3,
                             (ub4)OCI_DTYPE_LOB, (size_t) 0, (void **) 0);

  /* Assign Lob_loc1 to Lob_loc2 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, OCILobLocatorAssign(svchp, errhp, Lob_loc1, &Lob_loc3));

  /* 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 Lob1 and Lob3 are Equal.\n");
  checkerr (errhp, OCILobIsEqual(envhp, Lob_loc1, Lob_loc3, &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");
  }

  OCIDescriptorFree((void *)Lob_loc3, (ub4)OCI_DTYPE_LOB);

  return;
}
```

C/C++ (Pro*C/C++): Checking If One LOB Locator Is Equal to Another

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/proc/lequal.pc */

/* 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

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/java/lequal.java */

// 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();
    }
  }
}
```

Determining Whether LOB Locator Is Initialized

This section describes how to determine whether a LOB locator is initialized.

> **See Also:** Table 14–1, "Environments Supported for Basic LOB APIs" on page 14-2

Syntax

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* "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++ 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.

Examples

Examples are provided in the following programmatic environments:

- PL/SQL (DBMS_LOB Package): No example is provided with this release.
- C (OCI): Determining Whether a LOB Locator Is Initialized on page 14-118
- C (OCCI)): No example is provided with this release.
- COBOL (Pro*COBOL): No example is provided with this release.
- C/C++ (Pro*C/C++): Determining Whether a LOB Locator Is Initialized on page 14-119
- Visual Basic (OO4O): No example is provided with this release.
- Java (JDBC): No example is provided with this release.

C (OCI): Determining Whether a LOB Locator Is Initialized

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/oci/linit.c */

/* Seeing if a LOB locator is initialized */
#include <oratypes.h>
#include <lobdemo.h>

void isInitializedLOB_proc(OCILobLocator *Lob_loc1, OCILobLocator *Lob_loc2,
                           OCIEnv *envhp, OCIError *errhp, OCISvcCtx *svchp,
                           OCIStmt *stmthp)
{
  boolean       isInitialized;
  printf ("----------- OCILobLocatorIsInit Demo --------------\n");
  /* 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);

  return;
}
```

C/C++ (Pro*C/C++): Determining Whether a LOB Locator Is Initialized

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/proc/linit.pc */

/* 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 through 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;
}
```

Appending to a LOB

This section describes how to write-append the contents of a buffer to a LOB.

> **See Also:** Table 14–1, "Environments Supported for Basic LOB APIs" on page 14-2

Usage Notes

Note the following issues regarding usage of this API.

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`, then data must be provided through callbacks or polling.

- If a callback function is defined in the cbfp parameter, then this callback function will be called 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 `OCILobWriteAppend2()` returns the `OCI_NEED_DATA` error code. The application must call `OCILobWriteAppend2()` 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 Through Updated Locators" on page 5-16.

Syntax

Use the following syntax references for each programmatic environment:

- PL/SQL (DBM_LOB Package): *PL/SQL Packages and Types Reference* "DBMS_ LOB" — WRITEAPPEND
- C (OCI): *Oracle Call Interface Programmer's Guide* "Relational Functions" — LOB Functions, OCILobWriteAppend2
- C++ (OCCI): *Oracle C++ Call Interface Programmer's Guide*
- COBOL (Pro*COBOL) *Pro*COBOL 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++ 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): *Oracle Database JDBC Developer's Guide and Reference* Chapter 7, "Working With LOBs" — Creating and Populating a BLOB or CLOB Column.

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 14-121

- C (OCI): Writing to the End of (Appending to) a LOB on page 14-123
- C++ (OCCI): No example is provided with this release.
- COBOL (Pro*COBOL): Writing to the End of (Appending to) a LOB on page 14-124
- C/C++ (Pro*C/C++): Writing to the End of (Appending to) a LOB on page 14-125
- Visual Basic (OO4O): No example is provided with this release.
- Java (JDBC): Writing to the End of (Append-Write to) a LOB on page 14-126

PL/SQL (DBMS_LOB Package): Writing to the End of (Appending to) a LOB

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/plsql/lwriteap.sql */

/* Procedure lobWriteAppend_proc is not part of the DBMS_LOB package: */

/* writing to the end of lob (write append) */

CREATE OR REPLACE PROCEDURE lobWriteAppend_proc (Lob_loc IN OUT BLOB) IS
   /* Note: Lob_loc can be a persistent or temporary LOB */
   Buffer     RAW(32767);
   Amount     Binary_integer := 4;
BEGIN
   DBMS_OUTPUT.PUT_LINE('------------ LOB WRITEAPPEND EXAMPLE ------------');
   /* Fill the buffer with data... */
   Buffer := hextoraw('ab2daa44');
   /* 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;
/
SHOW ERRORS;
```

C (OCI): Writing to the End of (Appending to) a LOB

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/oci/lwriteap.c */

/* Write-appending to a LOB */
#include <oratypes.h>
#include <lobdemo.h>
void writeAppendLOB_proc(OCILobLocator *Lob_loc, OCIEnv *envhp,
                  OCIError *errhp, OCISvcCtx *svchp, OCIStmt *stmthp)
{
  oraub8 amt;
  oraub8 offset;
  sword retval;
  ub1 bufp[MAXBUFLEN];
  oraub8 buflen;

  printf ("----------- OCILobWriteAppend Demo --------------\n");
  /* Open the CLOB: */
  checkerr (errhp, (OCILobOpen(svchp, errhp, Lob_loc, OCI_LOB_READWRITE)));

  /* Setting the amt to the buffer length.  Note here that amt is in chars
     since we are using a CLOB: */
  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, OCILobWriteAppend2(svchp, errhp, Lob_loc, NULL,
                              &amt, (void *)bufp, buflen,
                              OCI_ONE_PIECE, (void *)0,
                              (OCICallbackLobWrite2) 0,
                              0, SQLCS_IMPLICIT));

  /* Closing the CLOB is mandatory if you have opened it: */
  checkerr (errhp, OCILobClose(svchp, errhp, Lob_loc));
  return;
}
```

COBOL (Pro*COBOL): Writing to the End of (Appending to) a LOB

```
* This file is installed in the following path when you install
* the database: $ORACLE_HOME/rdbms/demo/lobs/procob/lwriteap.pco

* 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

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/proc/lwriteap.pc */

/* 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

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/java/lwriteap.java */

// 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

This section describes how to write the contents of a buffer to a LOB.

See Also:

- Table 14–1, "Environments Supported for Basic LOB APIs" on page 14-2
- Reading Data from a LOB on page 14-52

Usage Notes

Note the following issues regarding usage of this API.

Stream Write

The most efficient way to write large amounts of LOB data is to use `OCILobWrite2()` with the streaming mechanism enabled using polling or a callback. If you know how much data will be written to the LOB, then specify that amount when calling `OCILobWrite2()`. This ensures that LOB data on the disk is contiguous. 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 run `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, then no extra or excess versioning is incurred or duplicated. If it is appropriate for your application, then 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 Through Updated Locators" on page 5-16.

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 <= the buffer `length` parameter
- The `length` of the buffer should be ((`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

Use the following syntax references for each programmatic environment:

- PL/SQL (DBM_LOB Package): *PL/SQL Packages and Types Reference* "DBMS_LOB" — WRITE
- C (OCI): *Oracle Call Interface Programmer's Guide* "Relational Functions" — LOB Functions, OCILobWrite2.
- C++ (OCCI): *Oracle C++ Call Interface Programmer's Guide*
- COBOL (Pro*COBOL) *Pro*COBOL 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++ 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): *Oracle Database JDBC Developer's Guide and Reference* Chapter 7, "Working With LOBs" — Creating and Populating a BLOB or CLOB Column.

Examples

Examples are provided in the following programmatic environments:

- PL/SQL (DBMS_LOB Package): Writing Data to a LOB on page 14-131
- C (OCI): Writing Data to a LOB on page 14-132
- COBOL (Pro*COBOL): Writing Data to a LOB on page 14-135
- C/C++ (Pro*C/C++): Writing Data to a LOB on page 14-137
- Visual Basic (OO4O):Writing Data to a LOB on page 14-140

- Java (JDBC): Writing Data to a LOB on page 14-141

PL/SQL (DBMS_LOB Package): Writing Data to a LOB

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/plsql/lwrite.sql */

/* Procedure writeDataToLOB_proc is not part of the DBMS_LOB package: */

/* writing data to a lob */

CREATE or REPLACE PROCEDURE writeDataToLOB_proc (Lob_loc IN OUT BLOB) IS
   /* Note: Lob_loc can be a persistent or temporary LOB */
   Buffer          RAW(32767);
   Amount          BINARY_INTEGER := 10;
   OptimalAmount   BINARY_INTEGER;
   Position        INTEGER := 1;
   i               INTEGER;
   Chunk_size      INTEGER;
BEGIN
   DBMS_OUTPUT.PUT_LINE('------------ LOB WRITE EXAMPLE ------------');
   /* For persistent LOBs, for each DBMS_LOB call,
    * we write data in multiples of chunksize,
    * and write on chunk boundaries. This ensures best performance */
    Chunk_size := DBMS_LOB.GETCHUNKSIZE(Lob_loc);
    OptimalAmount := (Amount/Chunk_size) * Chunk_size;
    if OptimalAmount = 0 then
      OptimalAmount := Amount;
    end if;

   /* Fill a buffer */
   Buffer := hextoraw(lpad('4', Amount*4, '4'));
   /* Opening the LOB is optional: */
   DBMS_LOB.OPEN (Lob_loc, DBMS_LOB.LOB_READWRITE);
   FOR i IN 1..3 LOOP
      Amount := OptimalAmount;
      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;
```

```
/
SHOW ERRORS;
```

C (OCI): Writing Data to a LOB

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/oci/lwrite.c */

/* 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. */
#include <oratypes.h>
#include <lobdemo.h>
void writeLOB_proc(OCILobLocator *Lob_loc, OCIEnv *envhp,
                   OCIError *errhp, OCISvcCtx *svchp, OCIStmt *stmthp,
                   oraub8 amt_to_write)
{
  /* <total amount of data to write to the CLOB in bytes> */
  oraub8 amt;
  oraub8 offset;
  unsigned int remainder, nbytes;
  boolean last;
  ub1 bufp[MAXBUFLEN];
  sword err;
  oraub8 lob_len;

  printf ("----------- OCILobWrite Demo --------------\n");
  /* Open the CLOB */
  checkerr (errhp, (OCILobOpen(svchp, errhp, Lob_loc, OCI_LOB_READWRITE)));
  if (amt_to_write > MAXBUFLEN)
    nbytes = MAXBUFLEN;   /* We will use streaming through standard polling */
  else
    nbytes = (unsigned int)amt_to_write;  /* Only a single write is required */

  /* Fill in all a's */
  memset((void *)bufp, 'a', MAXBUFLEN);

  /* Fill the buffer with nbytes worth of data */
  remainder = (unsigned int)(amt_to_write - nbytes);
```

```
/* Setting Amount to 0 streams the data until use specifies OCI_LAST_PIECE */
amt = 0;
offset = 1;

/* check lob length before update */
checkerr (errhp, OCILobGetLength2(svchp, errhp, Lob_loc, &lob_len));
printf(" Lob length before update = %d\n", (ub4)lob_len);


if (0 == remainder)
{
  printf(" writing the Lob data in one-piece mode\n");
  amt = (oraub8)nbytes;
  /* Here, (amt_to_write <= MAXBUFLEN ) so we can write in one piece */
  checkerr (errhp, OCILobWrite2(svchp, errhp, Lob_loc, &amt, NULL,
                                offset, (void *)bufp, nbytes,
                                OCI_ONE_PIECE, (void *)0,
                                (OCICallbackLobWrite2)0,
                                0, SQLCS_IMPLICIT));
}
else
{
  printf(" writing the Lob data in streaming polling mode\n");
  /* Here (amt_to_write > MAXBUFLEN ) so we use streaming
     through standard polling */
  /* write the first piece.  Specifying first initiates polling. */
  err =  OCILobWrite2(svchp, errhp, Lob_loc, &amt, NULL,
                      offset, (void *)bufp, nbytes,
                      OCI_FIRST_PIECE, (void *)0,
                      (OCICallbackLobWrite2) 0,
                      0, SQLCS_IMPLICIT);

  printf("  1st call. amt returned = %d\n", (ub4)amt);
  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 = OCILobWrite2(svchp, errhp, Lob_loc, &amt, NULL,
                         offset, (void *)bufp, nbytes,
                         OCI_LAST_PIECE, (void *)0,
                         (OCICallbackLobWrite2) 0,
                         0, SQLCS_IMPLICIT);
      printf("  last call. amt returned = %d\n", (ub4)amt);
      if (err != OCI_SUCCESS)
        checkerr(errhp, err);
    }
    else
    {
      err = OCILobWrite2(svchp, errhp, Lob_loc, &amt, NULL,
                         offset, (void *)bufp, nbytes,
                         OCI_NEXT_PIECE, (void *)0,
                         (OCICallbackLobWrite2) 0,
                         0, SQLCS_IMPLICIT);
      printf("  subsequent call. amt returned = %d\n", (ub4)amt);

      if (err != OCI_NEED_DATA)
        checkerr (errhp, err);
    }
    /* Determine how much is left to write */
    remainder = remainder - nbytes;
  } while (!last);
}

/* check lob length after update */
checkerr (errhp, OCILobGetLength2(svchp, errhp, Lob_loc, &lob_len));
printf(" Lob length after update = %d\n", (ub4)lob_len);

/* At this point, (remainder == 0) */

/* Closing the LOB is mandatory if you have opened it */
checkerr (errhp, OCILobClose(svchp, errhp, Lob_loc));

return;
}
```

COBOL (Pro*COBOL): Writing Data to a LOB

```
 * This file is installed in the following path when you install
 * the database: $ORACLE_HOME/rdbms/demo/lobs/procob/lwrite.pco

* 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

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/proc/lwrite.pc */

/* 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;   /* Using streaming through 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 use streaming through
      /* standard polling to 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

```
' This file is installed in the following path when you install
' the database: $ORACLE_HOME/rdbms/demo/lobs/vb/lwrite.bas

'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

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/java/lwrite.java */

//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

This section describes how to trim a LOB to the size you specify.

> **See Also:** Table 14–1, "Environments Supported for Basic LOB APIs" on page 14-2

Usage Notes

Note the following issues regarding usage of this API.

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 Through Updated Locators" on page 5-16.

Syntax

Use the following syntax references for each programmatic environment:

- PL/SQL (DBM_LOB Package): *PL/SQL Packages and Types Reference* "DBMS_LOB" — TRIM
- C (OCI): *Oracle Call Interface Programmer's Guide* "Relational Functions" — LOB Functions, OCILobTrim2.

- C++ (OCCI): *Oracle C++ Call Interface Programmer's Guide*
- COBOL (Pro*COBOL) *Pro*COBOL 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++ 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): *Oracle Database JDBC Developer's Guide and Reference* Chapter 7, "Working With LOBs" — Creating and Populating a BLOB or CLOB Column.

Examples

Examples are provided in the following programmatic environments:

- PL/SQL (DBMS_LOB Package): Trimming LOB Data on page 14-144
- C (OCI): Trimming LOB Data on page 14-145
- C++ (OCCI): No example is provided with this release.
- COBOL (Pro*COBOL): Trimming LOB Data on page 14-146
- C/C++ (Pro*C/C++): Trimming LOB Data on page 14-147
- Visual Basic (OO4O): Trimming LOB Data on page 14-149
- Java (JDBC): Trimming LOB Data on page 14-149

PL/SQL (DBMS_LOB Package): Trimming LOB Data

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/plsql/ltrim.sql */

/* Procedure trimLOB_proc is not part of the DBMS_LOB package: */

/* trimming lob data */

CREATE OR REPLACE PROCEDURE trimLOB_proc (Lob_loc IN OUT BLOB) IS
    /* Note: Lob_loc can be a persistent or temporary LOB */
BEGIN
   DBMS_OUTPUT.PUT_LINE('------------ LOB TRIM EXAMPLE ------------');
```

```
   /* Opening the LOB is optional: */
   DBMS_LOB.OPEN (Lob_loc, DBMS_LOB.LOB_READWRITE);
   /* Trim the LOB data: */
   DBMS_LOB.TRIM(Lob_loc,3);
   /* Closing the LOB is mandatory if you have opened it: */
   DBMS_LOB.CLOSE (Lob_loc);
   DBMS_OUTPUT.PUT_LINE('Trim succeeded');
/* Exception handling: */
EXCEPTION
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('Trim failed');
END;
/
SHOW ERRORS;
```

C (OCI): Trimming LOB Data

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/oci/ltrim.c */

/* Trimming LOB data */
#include <oratypes.h>
#include <lobdemo.h>
void trimLOB_proc(OCILobLocator *Lob_loc, OCIEnv *envhp,
                  OCIError *errhp, OCISvcCtx *svchp, OCIStmt *stmthp)
{
  oraub8 trimLength;

  printf ("----------- OCILobTrim Demo --------------\n");

  /* Open the CLOB */
  checkerr (errhp, (OCILobOpen(svchp, errhp, Lob_loc, OCI_LOB_READWRITE)));

  /* Trim the LOB to its new length */
  trimLength = 200;                      /* <New truncated length of the LOB>*/

  printf (" trim the lob to %d bytes\n", (ub4)trimLength);
  checkerr (errhp, OCILobTrim2(svchp, errhp, Lob_loc, trimLength ));

  /* Closing the CLOB is mandatory if you have opened it */
  checkerr (errhp, OCILobClose(svchp, errhp, Lob_loc));
```

```
}
```

COBOL (Pro*COBOL): Trimming LOB Data

```
      * This file is installed in the following path when you install
      * the database: $ORACLE_HOME/rdbms/demo/lobs/procob/ltrim.pco

     * 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

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/proc/ltrim.pc */

/* 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

```
' This file is installed in the following path when you install
' the database: $ORACLE_HOME/rdbms/demo/lobs/vb/ltrim.bas

'Trimming LOB data
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

OraDyn.Edit
OraAdPhoto1.Trim 10
OraDyn.Update
```

Java (JDBC): Trimming LOB Data

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/java/ltrim.java */

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

      // Show the original lob length
      System.out.println ("Open the lobs");
      System.out.println ("blob.length()="+blob.length());
      System.out.println ("clob.length()="+clob.length());

      // Trim the lobs
      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 lobs");
      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`:

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/java/oldltrim.java */
```

```
// 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

This section describes how to erase part of a LOB.

> **See Also:** Table 14–1, "Environments Supported for Basic LOB APIs" on page 14-2

Usage Notes

Note the following issues regarding usage of this API.

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 Through Updated Locators" on page 5-16.

Syntax

Use the following syntax references for each programmatic environment:

- PL/SQL (DBM_LOB Package): *PL/SQL Packages and Types Reference* "DBMS_LOB" — ERASE
- C (OCI): *Oracle Call Interface Programmer's Guide* "Relational Functions" — LOB Functions, OCILobErase2.
- C++ (OCCI): *Oracle C++ Call Interface Programmer's Guide*
- COBOL (Pro*COBOL) *Pro*COBOL 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++ 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): *Oracle Database JDBC Developer's Guide and Reference* Chapter 7, "Working With LOBs" — Creating and Populating a BLOB or CLOB Column.

Examples

Examples are provided in the following programmatic environments:

- PL/SQL (DBMS_LOB Package): Erasing Part of a LOB on page 14-155
- C (OCI): Erasing Part of a LOB on page 14-156

- C++ (OCCI): No example is provided with this release.

- COBOL (Pro*COBOL): Erasing Part of a LOB on page 14-156
- C/C++ (Pro*C/C++): Erasing Part of a LOB on page 14-158
- Visual Basic (OO4O): Erasing Part of a LOB on page 14-159
- Java (JDBC): Erasing Part of a LOB on page 14-159

PL/SQL (DBMS_LOB Package): Erasing Part of a LOB

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/plsql/lerase.sql */

/* Procedure eraseLOB_proc is not part of the DBMS_LOB package: */

/* erasing part of a lob */

CREATE OR REPLACE PROCEDURE eraseLOB_proc (Lob_loc IN OUT BLOB) IS
   /* Note: Lob_loc can be a persistent or temporary LOB */
   Amount         INTEGER := 3000;
BEGIN
   DBMS_OUTPUT.PUT_LINE('------------ LOB ERASE EXAMPLE ------------');
   /* Opening the LOB is optional: */
   DBMS_LOB.OPEN (Lob_loc, DBMS_LOB.LOB_READWRITE);
   /* Erase the data: */
   DBMS_LOB.ERASE(Lob_loc, Amount, 4);
   /* Closing the LOB is mandatory if you have opened it: */
   DBMS_LOB.CLOSE (Lob_loc);
   DBMS_OUTPUT.PUT_LINE('Erase succeeded');
/* Exception handling: */
EXCEPTION
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('Erase failed');
END;
/
SHOW ERRORS;
```

C (OCI): Erasing Part of a LOB

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/oci/lerase.c */

/* Erasing part of a LOB (persistent LOBs) */
#include <oratypes.h>
#include <lobdemo.h>
void eraseLOB_proc(OCILobLocator *Lob_loc, OCIEnv *envhp,
                   OCIError *errhp, OCISvcCtx *svchp, OCIStmt *stmthp)
{
  oraub8 amount = 300;
  oraub8 offset = 10;

  printf ("----------- OCILobErase Demo --------------\n");
  /* Open the CLOB: */
  checkerr (errhp, (OCILobOpen(svchp, errhp, Lob_loc, OCI_LOB_READWRITE)));

  /* Erase the data starting at the specified Offset: */
  printf(" erase %d bytes at offset %d from the Lob\n", (ub4)amount,
(ub4)offset);
  checkerr (errhp, OCILobErase2(svchp, errhp, Lob_loc, &amount, offset ));

  /* Closing the CLOB is mandatory if you have opened it: */
  checkerr (errhp, OCILobClose(svchp, errhp, Lob_loc));

  return;
}
```

COBOL (Pro*COBOL): Erasing Part of a LOB

```
      * This file is installed in the following path when you install
      * the database: $ORACLE_HOME/rdbms/demo/lobs/procob/lerase.pco

     * 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

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/proc/lerase.pc */

/* 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

```
' This file is installed in the following path when you install
' the database: $ORACLE_HOME/rdbms/demo/lobs/vb/lerase.bas

'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 begining from the 100th byte:
OraDyn.Edit
OraAdPhoto1.Erase 10, 100
OraDyn.Update
```

Java (JDBC): Erasing Part of a LOB

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/java/lerase.java */

// 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

This section describes how to enable LOB buffering.

> **See Also:** Table 14–1, "Environments Supported for Basic LOB APIs" on page 14-2

Usage Notes

Enable LOB buffering when you are 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 5-2.

Syntax

Use the following syntax references for each programmatic environment:

- PL/SQL: This API is not available in any supplied PL/SQL packages.
- C (OCI): *Oracle Call Interface Programmer's Guide* "Relational Functions" — LOB Functions, OCIEnableLobBuffering, OCIDisableLobBuffering, OCIFlushBuffer
- C++ (OCCI): *Oracle C++ Call Interface Programmer's Guide*
- COBOL (Pro*COBOL) *Pro*COBOL 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++ 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.

Examples

Examples are provided in the following programmatic environments:

- C (OCI): No example is provided with this release. Using this API is similar to that described in the example, "Disabling LOB Buffering" on page 14-171.
- C++ (OCCI): No example is provided with this release.
- COBOL (Pro*COBOL): Enabling LOB Buffering on page 14-163
- C/C++ (Pro*C/C++): Enabling LOB Buffering on page 14-165
- Visual Basic (OO4O): Enabling LOB Buffering on page 14-166

COBOL (Pro*COBOL): Enabling LOB Buffering

```
 * This file is installed in the following path when you install
 * the database: $ORACLE_HOME/rdbms/demo/lobs/procob/lenbuf.pco

* 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

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/proc/lenbuf.pc */

/* 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 (OO4O): Enabling LOB Buffering

```
' This file is installed in the following path when you install
' the database: $ORACLE_HOME/rdbms/demo/lobs/vb/lenbuf.bas

'Enabling LOB buffering (persistent LOBs)
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

'Enable buffering:
OraAdPhoto1.EnableBuffering
```

Flushing the Buffer

This section describes how to flush the LOB buffer.

See Also: Table 14–1, "Environments Supported for Basic LOB APIs" on page 14-2

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 5-2.

Syntax

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* "Relational Functions" — LOB Functions, OCIEnableLobBuffering, OCIDisableLobBuffering, OCIFlushBuffer.
- C++ (OCCI): *Oracle C++ Call Interface Programmer's Guide*
- COBOL (Pro*COBOL) *Pro*COBOL 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++ 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.

Examples

Examples are provided in the following programmatic environments:

- PL/SQL (DBMS_LOB Package): No example is provided with this release.
- C (OCI): No example is provided with this release. Using this API is similar to that described in the example, "Disabling LOB Buffering" on page 14-171.
- C++ (OCCI): No example is provided with this release.
- COBOL (Pro*COBOL): Flushing the Buffer on page 14-168
- C/C++ (Pro*C/C++): Flushing the Buffer on page 14-170
- Visual Basic (OO4O): No example is provided with this release.
- Java (JDBC): No example is provided with this release.

COBOL (Pro*COBOL): Flushing the Buffer

```
  * This file is installed in the following path when you install
  * the database: $ORACLE_HOME/rdbms/demo/lobs/procob/lflbuf.pco

 * 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 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++): Flushing the Buffer

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/proc/lflbuf.pc */

/* Flushing the LOB Buffer (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 flushBufferingLOB_proc()
{
  OCIBlobLocator *Lob_loc;
  int Amount = BufferLength;
  int multiple, Position = 1;

  /* Datatype equivalencing is mandatory for this datatype: */
  char Buffer[BufferLength];
  EXEC SQL VAR Buffer is RAW(BufferLength);
  EXEC SQL WHENEVER SQLERROR DO Sample_Error();

  /* Allocate and Initialize the LOB: */
  EXEC SQL ALLOCATE :Lob_loc;
  EXEC SQL SELECT Sound INTO :Lob_loc
             FROM Multimedia_tab WHERE Clip_ID = 1 FOR UPDATE;

  /* Enable use of the LOB Buffering Subsystem: */
  EXEC SQL LOB ENABLE BUFFERING :Lob_loc;
  memset((void *)Buffer, 0, BufferLength);
```

```
  for (multiple = 0; multiple < 8; multiple++)
    {
      /* Write data to the LOB: */
      EXEC SQL LOB WRITE ONE :Amount
                    FROM :Buffer INTO :Lob_loc AT :Position;
      Position += BufferLength;
    }
  /* Flush the contents of the buffers and Free their resources: */
  EXEC SQL LOB FLUSH BUFFER :Lob_loc FREE;
  /* Turn off use of the LOB Buffering Subsystem: */
  EXEC SQL LOB DISABLE BUFFERING :Lob_loc;
  /* Release resources held by the Locator: */
  EXEC SQL FREE :Lob_loc;
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  flushBufferingLOB_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

Disabling LOB Buffering

This section describes how to disable LOB buffering.

> **See Also:** Table 14–1, "Environments Supported for Basic LOB APIs" on page 14-2

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 5-2

Syntax

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* "Relational Functions" — LOB Functions, OCIEnableLobBuffering, OCIDisableLobBuffering, OCIFlushBuffer
- C++ (OCCI): *Oracle C++ Call Interface Programmer's Guide*
- COBOL (Pro*COBOL) *Pro*COBOL 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++ 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.

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 14-173
- C++ (OCCI): No example is provided with this release.
- COBOL (Pro*COBOL): Disabling LOB Buffering on page 14-174
- C/C++ (Pro*C/C++): Disabling LOB Buffering on page 14-176

- Visual Basic (OO4O): Disabling LOB Buffering on page 14-177
- Java (JDBC): No example is provided with this release.

C (OCI): Disabling LOB Buffering

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/oci/ldisbuf.c */

/* Disabling LOB buffering (persistent LOBs) */
#include <oratypes.h>
#include <lobdemo.h>
void LOBBuffering_proc(OCILobLocator *Lob_loc, OCIEnv *envhp,
                       OCIError *errhp, OCISvcCtx *svchp, OCIStmt *stmthp)
{
  ub4 amt;
  ub4 offset;
  sword retval;
  ub1 bufp[MAXBUFLEN];
  ub4 buflen;
  printf ("----------- OCI LOB Buffering Demo --------------\n");

  /* Open the CLOB: */
  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, (void *)bufp, buflen,
                                OCI_ONE_PIECE, (void *)0,
                                (sb4 (*)(void*,void*,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 CLOB is mandatory if you have opened it: */
  checkerr (errhp, OCILobClose(svchp, errhp, Lob_loc));

  return;
}
```

COBOL (Pro*COBOL): Disabling LOB Buffering

```
      * This file is installed in the following path when you install
      * the database: $ORACLE_HOME/rdbms/demo/lobs/procob/ldisbuf.pco

     * 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

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/proc/ldisbuf.pc */

/* 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

```
' This file is installed in the following path when you install
' the database: $ORACLE_HOME/rdbms/demo/lobs/vb/ldisbuf.bas

'Disabling LOB buffering (persistent LOBs)
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
'Disable buffering:
OraAdPhoto1.DisableBuffering
```

Determining Whether a LOB instance Is Temporary

This section describes how to determine whether a LOB instance is temporary.

> **See Also:** Table 14–1, "Environments Supported for Basic LOB APIs" on page 14-2

Syntax

Use the following syntax references for each programmatic environment:

- PL/SQL (DBMS_LOB): *PL/SQL Packages and Types Reference* "DBMS_LOB" — ISTEMPORARY, FREETEMPORARY
- C (OCI): *Oracle Call Interface Programmer's Guide* "Relational Functions" — LOB Functions, OCILobIsTemporary
- COBOL (Pro*COBOL) *Pro*COBOL 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++ 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): *Oracle Database JDBC Developer's Guide and Reference* Chapter 7, "Working With LOBs" — Creating and Populating a BLOB or CLOB Column.

Examples

Examples are provided in the following programmatic environments:

- PL/SQL (DBMS_LOB Package): Determining Whether a LOB Is Temporary on page 14-178
- C (OCI): Determining Whether a LOB Is Temporary on page 14-179
- COBOL (Pro*COBOL): Determining Whether a LOB Is Temporary on page 14-180
- C/C++ (Pro*C/C++): Determining Whether a LOB Is Temporary on page 14-182
- Visual Basic (OO4O): No example is provided with this release.

- Java (JDBC): Determining Whether a BLOB Is Temporary on page 14-183
- Java (JDBC): Determining Whether a CLOB Is Temporary on page 14-184.

PL/SQL (DBMS_LOB Package): Determining Whether a LOB Is Temporary

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/plsql/listemp.sql */

/* Procedure isTempLob_proc is not part of the DBMS_LOB package: */

/* seeing if lob is temporary. */

CREATE or REPLACE PROCEDURE isTempLob_proc(Lob_loc IN OUT BLOB) IS
   /* Note: Lob_loc can be persistent or temporary LOB */
BEGIN
   DBMS_OUTPUT.PUT_LINE('------------ LOB ISTEMPORARY EXAMPLE ------------');
   /* Check to make sure that the locator is pointing to a temporary LOB */
    IF DBMS_LOB.ISTEMPORARY(Lob_loc) = 1 THEN
        DBMS_OUTPUT.PUT_LINE('Input locator is a temporary LOB locator');
    ELSE
        /* Print an error: */
        DBMS_OUTPUT.PUT_LINE('Input locator is not a temporary LOB locator');
    END IF;
END;
/
SHOW ERRORS;
```

C (OCI): Determining Whether a LOB Is Temporary

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/oci/listemp.c */

/* Checking if a LOB is temporary.
   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: */
#include <oratypes.h>
```

```
#include <lobdemo.h>
void isTempLOBAndFree_proc(OCILobLocator *Lob_loc, OCIEnv *envhp, OCIError
*errhp,
                           OCISvcCtx *svchp, OCIStmt *stmthp)
{
  boolean is_temp;
  is_temp = FALSE;
  printf ("-----------  OCILobIsTemporary and OCILobFreeTemporary Demo \
--------------\n");
  checkerr (errhp, OCILobIsTemporary(envhp, errhp, Lob_loc, &is_temp));

  if(is_temp)
  {
      checkerr(errhp, (OCILobFreeTemporary(svchp, errhp, Lob_loc)));
      printf("Temporary LOB freed\n");

  }
  else
  {
      printf("locator is not a temporary LOB locator\n");
  }

}
```

COBOL (Pro*COBOL): Determining Whether a LOB Is Temporary

```
      * This file is installed in the following path when you install
      * the database: $ORACLE_HOME/rdbms/demo/lobs/procob/listemp.pco

     * Checking if a LOB is temporary.
       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 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 Whether a LOB Is Temporary

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/proc/listemp.pc */

/* Checking if a LOB is temporary.
#include <oci.h>
#include <stdio.h>
#include <sqlca.h>

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

void lobIsTemp_proc()
{
  OCIBlobLocator *Temp_loc;
  int isTemporary = 0;

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  /* Allocate and Create the Temporary LOB: */
  EXEC SQL ALLOCATE :Temp_loc;
  EXEC SQL LOB CREATE TEMPORARY :Temp_loc;
  /* Determine if the Locator is a Temporary LOB Locator: */
  EXEC SQL LOB DESCRIBE :Temp_loc GET ISTEMPORARY INTO :isTemporary;

  /* 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): Determining Whether 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:

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/java/listempb.java */

/**
* Checking if a BLOB is temporary.
* 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): Determining Whether a CLOB Is Temporary

To determine whether 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. These two methods are defined as follows:

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/java/listempc.java */

/**
 * Checking if LOB is temporary.
 * 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);
```

Converting a BLOB to a CLOB

You can convert a BLOB instance to a CLOB using the PL/SQL procedure DBMS_LOB.CONVERTTOCLOB. This technique is convenient if you have character data stored in binary format that you want to store in a CLOB. You specify the character set of the binary data when calling this procedure. See *PL/SQL Packages and Types Reference* for details on syntax and usage of this procedure.

Converting a CLOB to a BLOB

You can convert a CLOB instance to a BLOB instance using the PL/SQL procedure DBMS_LOB.CONVERTTOBLOB. This technique is a convenient way to convert character data to binary data using LOB APIs. See *PL/SQL Packages and Types Reference* for details on syntax and usage of this procedure.

15

LOB APIs for BFILE Operations

This chapter describes APIs for operations that use BFILEs. APIs covered in this chapter are listed in Table 15–1.

The following information is given for each operation described in this chapter:

- ***Usage Notes*** provide implementation guidelines such as information specific to a given programmatic environment or datatype.
- ***Syntax*** refers you to the syntax reference documentation for each supported programmatic environment.
- ***Examples*** describe any setup tasks necessary to run the examples given.

> **Note:** LOB APIs do not support loading data into BFILEs. See Using SQL*Loader to Load LOBs on page 3-2 for details on techniques for loading data into BFILEs.

Supported Environments for BFILE APIs

Table 15–1, "Environments Supported for BFILE APIs" indicates which programmatic environments are supported for the APIs discussed in this chapter. The first column describes the operation that the API performs. The remaining columns indicate with "Yes" or "No" whether the API is supported in PL/SQL, OCI, COBOL, Pro*C, Visual Basic (VB), and JDBC.

Table 15–1 Environments Supported for BFILE APIs

Operation	PL/SQL	OCI	COBOL	Pro*C	VB	JDBC
Inserting a Row Containing a BFILE on page 15-148	Yes	Yes	Yes	Yes	Yes	Yes
Loading a LOB with BFILE Data on page 15-13	Yes	Yes	Yes	Yes	Yes	Yes
Opening a BFILE with FILEOPEN on page 15-28	Yes	Yes	No	No	No	Yes
Opening a BFILE with OPEN on page 15-21	Yes	Yes	Yes	Yes	Yes	Yes
Determining Whether a BFILE Is Open Using ISOPEN on page 15-32	Yes	Yes	Yes	Yes	Yes	Yes
Determining Whether a BFILE Is Open with FILEISOPEN on page 15-41	Yes	Yes	No	No	No	Yes
Displaying BFILE Data on page 15-46	Yes	Yes	Yes	Yes	Yes	Yes
Reading Data from a BFILE on page 15-56	Yes	Yes	Yes	Yes	Yes	Yes
Reading a Portion of BFILE Data Using SUBSTR on page 15-66	Yes	No	Yes	Yes	Yes	Yes
Comparing All or Parts of Two BFILES on page 15-73	Yes	No	Yes	Yes	Yes	Yes
Checking If a Pattern Exists in a BFILE Using INSTR on page 15-82	Yes	No	Yes	Yes	No	Yes
Determining Whether a BFILE Exists on page 15-89	Yes	Yes	Yes	Yes	Yes	Yes
Getting the Length of a BFILE on page 15-97	Yes	Yes	Yes	Yes	Yes	Yes
Assigning a BFILE Locator on page 15-105	Yes	Yes	Yes	Yes	No	Yes

Table 15–1 Environments Supported for BFILE APIs(Cont.)

Operation	PL/SQL	OCI	COBOL	Pro*C	VB	JDBC
Getting Directory Object Name and Filename of a BFILE on page 15-111	Yes	Yes	Yes	Yes	Yes	Yes
Updating a BFILE by Initializing a BFILE Locator on page 15-119	Yes	Yes	Yes	Yes	Yes	Yes
Closing a BFILE with FILECLOSE on page 15-127	Yes	Yes	No	No	Yes	Yes
Closing a BFILE with CLOSE on page 15-131	Yes	Yes	Yes	Yes	Yes	Yes
Closing All Open BFILEs with FILECLOSEALL on page 15-139	Yes	Yes	Yes	Yes	Yes	Yes

Accessing BFILEs

To access 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 6, "Overview of Supplied LOB APIs" for information about supported environments for accessing BFILEs.

Directory Object

The `DIRECTORY` object facilitates administering access and usage of `BFILE` datatypes (see `CREATE DIRECTORY` in *Oracle Database SQL Reference)*. A `DIRECTORY` object specifies a *logical alias name* for a physical directory on the database server file system under which the file to be accessed is located. You can access a file in the server file system only if granted the required access privilege on `DIRECTORY` object.

Initializing a BFILE Locator

The `DIRECTORY` object also provides the flexibility to manage the locations of the files, instead of forcing you to hard code the absolute path names of physical files in your applications. A directory object name is used in conjunction with the `BFILENAME`() function, in SQL and PL/SQL, or the `OCILobFileSetName`(), in OCI for initializing a `BFILE` locator.

Note: The database 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, then 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 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 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 OCILobFileSetName() 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;
```

Note: The database does not expand environment variables specified in the DIRECTORY object or filename of a BFILE locator. For example, specifying:

```
BFILENAME('WORK_DIR', '$MY_FILE')
```

where `MY_FILE` is an environment variable defined in the operating system, is not valid.

BFILENAME() and Initialization

`BFILENAME()` is a built-in function that you use to initialize a `BFILE` column to point to an 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

Characteristics of the BFILE Datatype

Using the `BFILE` datatype has the following advantages:

- If your need for a particular `BFILE` is temporary and scoped just within the module on which you are working, then you can use the `BFILE` related APIs on the variable without ever having to associate this with a column in the database.

- Because 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 15-13).

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';
```

then the directory object 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 *Oracle Database SQL Reference*. Oracle Database 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 a DIRECTORY Object

READ permission on the DIRECTORY object enables 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, then 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 responsibility of the DBA 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 *Oracle Database 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 *Oracle Database SQL Reference* for information about the following SQL DML statements that provide security for `BFILE`s:

- `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 a DIRECTORY object to a data file directory. A DIRECTORY object 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 the DIRECTORY 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, then 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, then 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 run 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.
- Use caution when revoking a 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 run 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 (Multithreaded Server) Mode

The database does not support session migration for BFILE datatypes in shared server (multithreaded server) mode. This implies that operations on open BFILE instances 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), then 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 operates like any other automatic variable. With respect to file operations, it operates 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

Note the following guidelines when working with BFILEs:

- Open and close a file from the 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 the object to the database. If an object contains a `BFILE`, then 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 the DIRECTORY object name and filename before INSERT or UPDATE of a BFILE. It is an error to `INSERT or UPDATE` a `BFILE` without indicating a directory object name 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 object name 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`.

- Before using SQL to insert or update a row with a `BFILE`, you must initialize the `BFILE` to one of the following:
  - `NULL` (not possible if using an OCI bind variable)
  - A directory object name and filename

Loading a LOB with BFILE Data

This section describes how to load a LOB with data from a BFILE.

> **See Also:** Table 15–1, "Environments Supported for BFILE APIs" on page 15-2, for a list of operations on BFILEs and APIs provided for each programmatic environment.

Preconditions

The following preconditions must exist before calling this procedure:

- The source BFILE instance must exist.
- The destination LOB instance must exist.

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 Data from a BFILE" on page 14-26 and "Loading a CLOB or NCLOB with Data from a BFILE" on page 14-29 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 or NCLOB with Data from a BFILE on page 14-29).

> **See Also:** *Oracle Database Globalization Support Guide* for character set conversion issues.

Amount Parameter

Note the following with respect to the `amount` parameter:

- `DBMS_LOB.LOADFROMFILE`
  If you want to load the entire BFILE, then pass the constant `DBMS_LOB.LOBMAXSIZE`. If you pass any other value, then it must be less than or equal to the size of the BFILE.
- `OCILobLoadFromFile()`
  If you want to load the entire BFILE, then you can pass the constant `UB4MAXVAL`. If you pass any other value, then it must be less than or equal to the size of the BFILE.
- `OCILobLoadFromFile2()`
  If you want to load the entire BFILE, then you can pass the constant `UB8MAXVAL`. If you pass any other value, then it must be less than or equal to the size of the BFILE.

> **See Also:** Table 14–2, "Maximum LOB Size for Load from File Operations" on page 14-18 for details on the maximum value of the amount parameter.

Syntax

Use the following syntax references for each programmatic environment:

- PL/SQL (DBMS_LOB) *(PL/SQL Packages and Types Reference):* "DBMS_LOB" — LOADFROMFILE
- C (OCI) (*Oracle Call Interface Programmer's Guide*): Chapter 7, "LOB and File Operations", for usage notes and examples. "Relational Functions" — LOB Functions, OCILobLoadFromFile2.
- COBOL (Pro*COBOL) *Pro*COBOL 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++ 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) *Oracle Database JDBC Developer's Guide and Reference)*: Chapter 7, "Working With LOBs" — Creating and Populating a BLOB or CLOB Column.

Examples

Examples are provided in the following programmatic environments:

- PL/SQL (DBMS_LOB): Loading a LOB with BFILE Data on page 15-15
- C (OCI): Loading a LOB with BFILE Data on page 15-16
- COBOL (Pro*COBOL): Loading a LOB with BFILE Data on page 15-16
- C/C++ (Pro*C/C++): Loading a LOB with BFILE Data on page 15-18
- Visual Basic (OO4O): Loading a LOB with BFILE Data on page 15-20

PL/SQL (DBMS_LOB): Loading a LOB with BFILE Data

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/plsql/lloaddat.sql */

/* Procedure loadLOBFromBFILE_proc is not part of the DBMS_LOB package: */

/* loading a lob with bfile data */

CREATE OR REPLACE PROCEDURE loadLOBFromBFILE_proc (Dest_loc IN OUT BLOB) IS
   /* Note: Dest_loc can be a persistent or temporary LOB */
   Src_loc        BFILE := BFILENAME('MEDIA_DIR', 'keyboard_logo.jpg');
   Amount         INTEGER := 4000;
BEGIN
   DBMS_OUTPUT.PUT_LINE('------------ LOB LOADFORMFILE EXAMPLE ------------');
   /* 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(Dest_loc);
   DBMS_LOB.CLOSE(Src_loc);
END;
/
SHOW ERRORS;
```

C (OCI): Loading a LOB with BFILE Data

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/oci/lloaddat.c */
#include <oratypes.h>
#include <lobdemo.h>

void loadLOBDataFromBFile_proc(OCILobLocator *Lob_loc, OCILobLocator* BFile_loc,
                               OCIEnv *envhp,
                               OCIError *errhp, OCISvcCtx *svchp,
                               OCIStmt *stmthp)
{
  oraub8         amount= 2000;

  printf ("----------- OCILobLoadFromFile Demo --------------\n");

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

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

  /* Load the data from the graphic file (bfile) into the blob */
  printf (" load the LOB from File\n");
  checkerr (errhp, OCILobLoadFromFile2(svchp, errhp, Lob_loc, BFile_loc,
                                       amount,
                                       (oraub8)1, (oraub8)1));

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

  return;
}
```

COBOL (Pro*COBOL): Loading a LOB with BFILE Data

```
 * This file is installed in the following path when you install
      * the database: $ORACLE_HOME/rdbms/demo/lobs/procob/floadlob.pco
```

```
* 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

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/proc/floadlob.pc */

/* 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

```
' This file is installed in the following path when you install
' the database: $ORACLE_HOME/rdbms/demo/lobs/vb/floadlob.bas

'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
```

Opening a BFILE with OPEN

This section describes how to open a BFILE using the OPEN function.

Note: You can also open a BFILE using the FILEOPEN function; however, using the OPEN function is recommended for new development. Using the FILEOPEN function is described in Opening a BFILE with FILEOPEN on page 15-28.

See Also: Table 15–1, "Environments Supported for BFILE APIs" on page 15-2, for a list of operations on BFILEs and APIs provided for each programmatic environment.

Syntax

Use the following syntax references for each programmatic environment:

- PL/SQL(DBMS_LOB) *(PL/SQL Packages and Types Reference): "DBMS_LOB"* — OPEN
- C (OCI) (*Oracle Call Interface Programmer's Guide)*: Chapter 7, "LOB and File Operations", for usage notes. "Relational Functions" — LOB Functions, OCILobOpen, OCILobClose
- COBOL (Pro*COBOL) *Pro*COBOL 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++ 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) *(Oracle Database JDBC Developer's Guide and Reference):* Chapter 7, "Working With LOBs" — Creating and Populating a BLOB or CLOB Column.

Scenario

These examples open an image in operating system file `ADPHOTO_DIR`.

Examples

Examples are provided in the following six programmatic environments:

- PL/SQL (DBMS_LOB): Opening a BFILE with OPEN on page 15-22
- C (OCI): Opening a BFILE with OPEN on page 15-22
- C/C++ (Pro*C/C++): Opening a BFILE with OPEN on page 15-24
- COBOL (Pro*COBOL): Opening a BFILE with OPEN on page 15-23
- Visual Basic (OO4O) Opening a BFILE with OPEN on page 15-25
- Java (JDBC): Opening a BFILE with OPEN on page 15-26

PL/SQL (DBMS_LOB): Opening a BFILE with OPEN

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/plsql/fopen.sql */

/* 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('MEDIA_DIR', 'keyboard_logo.jpg');
BEGIN
   DBMS_OUTPUT.PUT_LINE('------------ BFILE OPEN EXAMPLE ------------');
   /* 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

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/oci/fopen.c */

/* Opening a BFILE with OPEN.  */
#include <oratypes.h>
#include <lobdemo.h>
void BfileLobOpen_proc(OCILobLocator *Bfile_loc, OCIEnv *envhp,
```

```
                        OCIError *errhp, OCISvcCtx *svchp, OCIStmt *stmthp)
{
     printf ("----------- OCILobOpen BFILE Demo --------------\n");
     checkerr(errhp, OCILobOpen(svchp, errhp, Bfile_loc,
                                (ub1)OCI_FILE_READONLY));
     /* ... Do some processing. */
     checkerr(errhp, OCILobClose(svchp, errhp, Bfile_loc));
}
```

COBOL (Pro*COBOL): Opening a BFILE with OPEN

```
 * This file is installed in the following path when you install
      * the database: $ORACLE_HOME/rdbms/demo/lobs/procob/fopen.pco

     * Opening a BFILE with OPEN.
       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 object 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 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++): Opening a BFILE with OPEN

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/proc/fopen.pc */

/* Opening a BFILE using OPEN.
   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

```
' This file is installed in the following path when you install
' the database: $ORACLE_HOME/rdbms/demo/lobs/vb/fopen.bas

'Opening a BFILE using OPEN.
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

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/java/fopen.java */

// Opening a BFILE with OPEN.
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();
    }
  }
}
```

Opening a BFILE with FILEOPEN

This section describes how to open a BFILE using the FILEOPEN function.

Note: The FILEOPEN function is not recommended for new application development. The OPEN function is recommended for new development. See "Opening a BFILE with OPEN" on page 15-21 for details on using OPEN.

See Also: Table 15–1, "Environments Supported for BFILE APIs" on page 15-2, for a list of operations on BFILEs and APIs provided for each programmatic environment.

Usage Notes

While you can continue to use the older `FILEOPEN` form, Oracle *strongly recommends* that you switch to using `OPEN`, because this facilitates future extensibility.

Syntax

Use the following syntax references for each programmatic environment:

- PL/SQL (DBMS_LOB) *(PL/SQL Packages and Types Reference):* "DBMS_LOB" — FILEOPEN, FILECLOSE
- C (OCI) (*Oracle Call Interface Programmer's Guide*): Chapter 7, "LOB and File Operations, for usage notes. "Relational Functions" — LOB Functions, OCILobFileOpen, OCILobFileClose, OCILobFileSetName
- 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) (*Oracle Database JDBC Developer's Guide and Reference*): Chapter 7, "Working With LOBs" — Creating and Populating a BLOB or CLOB Column.

Scenario

These examples open `keyboard_photo3060` in operating system file `ADPHOTO_DIR`.

Examples

Examples are provided in the following four programmatic environments:

- PL/SQL (DBMS_LOB): Opening a BFILE with FILEOPEN on page 15-29
- C (OCI): Opening a BFILE with FILEOPEN on page 15-29
- COBOL (Pro*COBOL): No example is provided with this release.
- C/C++ (Pro*C/C++): No example is provided with this release.
- Java (JDBC): Opening a BFILE with FILEOPEN on page 15-30

PL/SQL (DBMS_LOB): Opening a BFILE with FILEOPEN

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/plsql/ffilopen.sql */

/* Opening a BFILE with FILEOPEN  */
/* Procedure openBFILE_procOne is not part of DBMS_LOB package: */
CREATE OR REPLACE PROCEDURE openBFILE_procOne IS
   file_loc    BFILE := BFILENAME('MEDIA_DIR', 'keyboard_logo.jpg');
BEGIN
   DBMS_OUTPUT.PUT_LINE('------------ LOB FILEOPEN EXAMPLE ------------');
   /* 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

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/oci/ffilopen.c */

/* Opening a BFILE with FILEOPEN  */
#include <oratypes.h>
#include <lobdemo.h>
void BfileFileOpen_proc(OCILobLocator *Bfile_loc, OCIEnv *envhp,
                        OCIError *errhp, OCISvcCtx *svchp, OCIStmt *stmthp)
{
```

```
  printf ("----------- OCILobFileOpen Demo --------------\n");
  checkerr(errhp, OCILobFileOpen(svchp, errhp, Bfile_loc,
                                 (ub1)OCI_FILE_READONLY));
  /* ... Do some processing. */
  checkerr(errhp, OCILobFileClose(svchp, errhp, Bfile_loc));
}
```

Java (JDBC): Opening a BFILE with FILEOPEN

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/java/ffilopen.java */

// 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();
    }
  }
}
```

Determining Whether a BFILE Is Open Using ISOPEN

This section describes how to determine whether a BFILE is open using ISOPEN.

Note: This function (ISOPEN) is recommended for new application development. The older FILEISOPEN function, described in "Determining Whether a BFILE Is Open with FILEISOPEN" on page 15-41, is not recommended for new development.

See Also: Table 15–1, "Environments Supported for BFILE APIs" on page 15-2, for a list of operations on BFILEs and APIs provided for each programmatic environment.

Syntax

Use the following syntax references for each programmatic environment:

- PL/SQL (DBMS_LOB) *(PL/SQL Packages and Types Reference):* "DBMS_LOB" — ISOPEN
- C (OCI) (*Oracle Call Interface Programmer's Guide)*: Chapter 7, "LOB and File Operations" for usage notes. "Relational Functions" — LOB Functions, OCILobFileIsOpen
- COBOL (Pro*COBOL) *Pro*COBOL 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++ 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) *(Oracle Database JDBC Developer's Guide and Reference):* Chapter 7, "Working With LOBs" — Creating and Populating a BLOB or CLOB Column.

Examples

Examples are provided in the following six programmatic environments:

- PL/SQL (DBMS_LOB): Determining Whether a BFILE Is Open with ISOPEN on page 15-32
- C (OCI): Determining Whether a BFILE Is Open with ISOPEN on page 15-34
- COBOL (Pro*COBOL): Determining Whether a BFILE Is Open with ISOPEN on page 15-34
- C/C++ (Pro*C/C++): Determining Whether a BFILE Is Open with ISOPEN on page 15-36
- Visual Basic (OO4O): Determining Whether a BFILE Is Open with ISOPEN on page 15-37
- Java (JDBC): Determining Whether a BFILE Is Open with ISOPEN on page 15-38

PL/SQL (DBMS_LOB): Determining Whether a BFILE Is Open with ISOPEN

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/plsql/fisopen.sql */

/* Checking if the BFILE is open with ISOPEN */
/* Procedure seeIfOpenBFILE_procTwo is not part of DBMS_LOB package: */

CREATE OR REPLACE PROCEDURE seeIfOpenBFILE_procTwo IS
   file_loc     BFILE;
   RetVal       INTEGER;
BEGIN
   DBMS_OUTPUT.PUT_LINE('------------ BFILE ISOPEN EXAMPLE ------------');
   /* 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;
/
```

```
SHOW ERRORS;
```

C (OCI): Determining Whether a BFILE Is Open with ISOPEN

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/oci/fisopen.c */

/* Checking if the BFILE is Open with ISOPEN. */
#include <oratypes.h>
#include <lobdemo.h>
void BfileIsOpen_proc(OCILobLocator *Bfile_loc, OCIEnv *envhp,
                      OCIError *errhp, OCISvcCtx *svchp, OCIStmt *stmthp)
{
  boolean flag;

   printf ("----------- OCILobIsOpen Demo --------------\n");
  /* Allocate the locator descriptor */
  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));
}
```

COBOL (Pro*COBOL): Determining Whether a BFILE Is Open with ISOPEN

```
 * This file is installed in the following path when you install
      * the database: $ORACLE_HOME/rdbms/demo/lobs/procob/fisopen.pco
```

```
* Checking if BFILE is open with ISOPEN
  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 object 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 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 Whether a BFILE Is Open with ISOPEN

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/proc/fisopen.pc */

/* Checking if the BFILE is open with ISOPEN.
   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): Determining Whether a BFILE Is Open with ISOPEN

```
' This file is installed in the following path when you install
' the database: $ORACLE_HOME/rdbms/demo/lobs/vb/fisopen.bas

' Checking if the BFILE is open with ISOPEN
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): Determining Whether a BFILE Is Open with ISOPEN

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/java/fisopen.java */

// Checking if the BFILE is open with ISOPEN

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();
         }
      }
   }
```

Determining Whether a BFILE Is Open with FILEISOPEN

This section describes how to determine whether a BFILE is OPEN using the FILEISOPEN function.

Note: The FILEISOPEN function is not recommended for new application development. The ISOPEN function is recommended for new development. See Determining Whether a BFILE Is Open Using ISOPEN on page 15-32 for details on using ISOPEN.

See Also: Table 15–1, "Environments Supported for BFILE APIs" on page 15-2, for a list of operations on BFILEs and APIs provided for each programmatic environment.

Usage Notes

While you can continue to use the older `FILEISOPEN` form, Oracle *strongly recommends* that you switch to using `ISOPEN`, because this facilitates future extensibility.

Syntax

Use the following syntax references for each programmatic environment:

- PL/SQL(DBMS_LOB) *(PL/SQL Packages and Types Reference):* "DBMS_LOB" — FILEISOPEN
- C (OCI) (*Oracle Call Interface Programmer's Guide*): Chapter 7, "LOB and File Operations" for usage notes. "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) (*Oracle Database JDBC Developer's Guide and Reference*): Chapter 7, "Working With LOBs" — Creating and Populating a BLOB or CLOB Column.

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): Determining Whether a BFILE Is Open with FILEISOPEN on page 15-42
- C (OCI): Determining Whether a BFILE Is Open with FILEISOPEN on page 15-43
- C/C++ (Pro*C/C++): No example is provided with this release.
- COBOL (Pro*COBOL): No example is provided with this release.
- Java (JDBC): Determining Whether a BFILE Is Open with FILEISOPEN on page 15-43

PL/SQL (DBMS_LOB): Determining Whether a BFILE Is Open with FILEISOPEN

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/plsql/ffisopen.sql */

/* Checking if the BFILE is OPEN with FILEISOPEN.
   Procedure seeIfOpenBFILE_procOne is not part of DBMS_LOB package: */

CREATE OR REPLACE PROCEDURE seeIfOpenBFILE_procOne IS
   file_loc      BFILE;
   RetVal       INTEGER;
BEGIN
   DBMS_OUTPUT.PUT_LINE('------------ BFILE FILEISOPEN EXAMPLE ------------');
   /* 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;
/
SHOW ERRORS;
```

C (OCI): Determining Whether a BFILE Is Open with FILEISOPEN

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/oci/ffisopen.c */

/* Checking if the BFILE is open with FILEISOPEN. */
#include <oratypes.h>
#include <lobdemo.h>
void BfileFileIsOpen_proc(OCILobLocator *Bfile_loc, OCIEnv *envhp,
                          OCIError *errhp, OCISvcCtx *svchp, OCIStmt *stmthp)
{
  boolean flag;

  printf ("----------- OCILobFileIsOpen Demo --------------\n");
  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));
}
```

Java (JDBC): Determining Whether a BFILE Is Open with FILEISOPEN

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/java/ffisopen.java */
```

```
// Checking if a BFILE is open with FILEISOPEN

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();
        }
      }
    }
```

Displaying BFILE Data

This section describes how to display BFILE data.

> **See Also:** Table 15–1, "Environments Supported for BFILE APIs" on page 15-2, for a list of operations on BFILEs and APIs provided for each programmatic environment.

Syntax

Use the following syntax references for each programmatic environment:

- PL/SQL (DBMS_LOB) *(PL/SQL Packages and Types Reference):* "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. "Relational Functions" — LOB Functions, OCILobFileOpen, OCILobRead2
- COBOL (Pro*COBOL) *Pro*COBOL 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++ 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) (*Oracle Database JDBC Developer's Guide and Reference)*: Chapter 7, "Working With LOBs" — Creating and Populating a BLOB or CLOB Column.

Examples

Examples are provided in six programmatic environments:

- PL/SQL (DBMS_LOB): Displaying BFILE Data on page 15-47
- C (OCI): Displaying BFILE Data on page 15-47
- COBOL (Pro*COBOL): Displaying BFILE Data on page 15-49
- C/C++ (Pro*C/C++): Displaying BFILE Data on page 15-51
- Visual Basic (OO4O): Displaying BFILE Data on page 15-53

- Java (JDBC): Displaying BFILE Data on page 15-53

PL/SQL (DBMS_LOB): Displaying BFILE Data

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/plsql/fdisplay.sql */

/* Displaying BFILE data.  */
/* Procedure displayBFILE_proc is not part of DBMS_LOB package: */

CREATE OR REPLACE PROCEDURE displayBFILE_proc IS
   file_loc BFILE := BFILENAME('MEDIA_DIR', 'monitor_3060.txt');
   Buffer   RAW(1024);
   Amount   BINARY_INTEGER := 200;
   Position INTEGER        := 1;
BEGIN
   DBMS_OUTPUT.PUT_LINE('------------ BFILE DISPLAY EXAMPLE ------------');
   /* 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(substr(utl_raw.cast_to_varchar2(Buffer), 1, 250));
      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;
/
SHOW ERRORS;
```

C (OCI): Displaying BFILE Data

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/oci/fdisplay.c */
```

```
/* Displaying BFILE data. */
#include <oratypes.h>
#include <lobdemo.h>
void BfileDisplay_proc(OCILobLocator *Bfile_loc, OCIEnv *envhp,
                       OCIError *errhp, OCISvcCtx *svchp, OCIStmt *stmthp)
{
   /* Assume all handles passed as input to this routine have been
      allocated and initialized */
   ub1 bufp[MAXBUFLEN];
   oraub8 buflen, amt, offset;
   boolean done;
   ub4 retval;
   ub1 piece;

   printf ("----------- OCI BFILE Display Demo --------------\n");
   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((void *)bufp, '\0', MAXBUFLEN);
   done = FALSE;
   piece = OCI_FIRST_PIECE;
   while (!done)
   {
     retval = OCILobRead2(svchp, errhp, Bfile_loc,
                          &amt, NULL, offset, (void *) bufp,
                          buflen, piece, (void *)0,
                          (OCICallbackLobRead2)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 in bytes. */
         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 in bytes. */
          piece = OCI_NEXT_PIECE;
          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));
}
```

COBOL (Pro*COBOL): Displaying BFILE Data

```
 * This file is installed in the following path when you install
      * the database: $ORACLE_HOME/rdbms/demo/lobs/procob/fdisplay.pco

     * Displaying BFILE data.
       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 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 BFILE Data

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/proc/fdisplay.pc */

/* Displaying BFILE data.
   This example reads the entire contents of a BFILE piecewise into a
   buffer using a streaming mechanism through 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

```
' This file is installed in the following path when you install
' the database: $ORACLE_HOME/rdbms/demo/lobs/vb/fdisplay.bas

' Displaying BFILE data.
Dim MySession As OraSession
Dim OraDb As OraDatabase

Dim OraDyn As OraDynaset, OraAdGraphio 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

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/java/fdisplay.java */

// Displaying BFILE data.

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

This section describes how to read data from a BFILE.

> **See Also:** Table 15–1, "Environments Supported for BFILE APIs" on page 15-2, for a list of operations on BFILEs and APIs provided for each programmatic environment.

Usage Notes

Note the following when using this operation.

Streaming Read in OCI

The most efficient way to read of large amounts of LOB data is to use `OCILobRead2()` with the streaming mechanism enabled using polling or callback. To do so, specify the starting point of the read using the offset parameter and use the symbol `OCI_LOBMAXSIZE` for the amount, as follows:

```
ub8  amount =  OCI_LOBMAXSIZE;
ub4  offset = 1000;
OCILobRead2(svchp, errhp, locp, &amount, offset, bufp, bufl, 0, 0, 0, 0)
```

When using *polling mode*, be sure to look at the value of the `amount` parameter after each `OCILobRead2()` call to see how many bytes were read into the buffer because 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 because the entire buffer may not be filled with data (see the *Oracle Call Interface Programmer's Guide.*)

Amount Parameter

- When calling `DBMS_LOB.READ`, the amount parameter can be larger than the size of the data; however, the amount parameter should be less than or equal to the size of the buffer. In PL/SQL, the buffer size is limited to 32K.
- When calling `OCILobRead2`, you can pass a value of 0 (zero) for the amount parameter to read to the end of the LOB.

Syntax

Use the following syntax references for each programmatic environment:

- PL/SQL (DBMS_LOB) *(PL/SQL Packages and Types Reference):* "DBMS_LOB" — READ
- C (OCI) (*Oracle Call Interface Programmer's Guide*): Chapter 7, "LOB and File Operations" for usage notes. "Relational Functions" — LOB Functions, OCILobRead2
- COBOL (Pro*COBOL) *Pro*COBOL 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++ 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) *(Oracle Database JDBC Developer's Guide and Reference):* Chapter 7, "Working With LOBs" — Creating and Populating a BLOB or CLOB Column.

Examples

Examples are provided in the following programmatic environments:

- PL/SQL (DBMS_LOB): Reading Data from a BFILE on page 15-57
- C (OCI): Reading Data from a BFILE on page 15-58
- COBOL (Pro*COBOL): Reading Data from a BFILE on page 15-60
- C/C++ (Pro*C/C++): Reading Data from a BFILE on page 15-61
- Visual Basic (OO4O): Reading Data from a BFILE on page 15-62
- Java (JDBC): Reading Data from a BFILE on page 15-63

PL/SQL (DBMS_LOB): Reading Data from a BFILE

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/plsql/fread.sql */

/* Reading data from a BFILE. */
/* Procedure readBFILE_proc is not part of DBMS_LOB package: */
```

```
CREATE OR REPLACE PROCEDURE readBFILE_proc IS
   file_loc      BFILE;
   Amount        INTEGER := 32767;
   Position      INTEGER := 1;
   Buffer        RAW(32767);
BEGIN
   DBMS_OUTPUT.PUT_LINE('------------ BFILE READ EXAMPLE ------------');
   /* 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;
/
show errors;
```

C (OCI): Reading Data from a BFILE

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/oci/fread.c */

/* Reading data from a BFILE. */
#include <oratypes.h>
#include <lobdemo.h>
void BfileRead_proc(OCILobLocator *Bfile_loc, OCIEnv *envhp,
                        OCIError *errhp, OCISvcCtx *svchp, OCIStmt *stmthp)
{
   ub1 bufp[MAXBUFLEN];
   oraub8 buflen, amt, offset;
   ub4 retval;
   ub1 piece;

   boolean done;

   printf ("----------- OCILobRead BFILE Demo --------------\n");
   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((void *)bufp, '\0', MAXBUFLEN);
   piece = OCI_FIRST_PIECE;
   done = FALSE;

  while (!done)
  {
    retval = OCILobRead2(svchp, errhp, Bfile_loc, &amt, NULL, offset,
                         (void *) bufp, buflen, piece, (void *)0,
                         (OCICallbackLobRead2) 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 is in bytes. */
      printf(" amt read=%d in the last call\n", (ub4)amt);
      done = TRUE;
     break;
    case OCI_ERROR:
      /*   report_error();        this function is not shown here */
      done = TRUE;
      break;
    case OCI_NEED_DATA:
      printf(" amt read=%d\n", (ub4)amt);
      piece = OCI_NEXT_PIECE;
      break;
    default:
      (void) printf("Unexpected ERROR: OCILobRead2() LOB.\n");
       done = TRUE;
       break;
    }
  }

  /* Closing the BFILE is mandatory if you have opened it */
  checkerr (errhp, OCILobFileClose(svchp, errhp, Bfile_loc));
}
```

COBOL (Pro*COBOL): Reading Data from a BFILE

```
* This file is installed in the following path when you install
     * the database: $ORACLE_HOME/rdbms/demo/lobs/procob/fread.pco

    * Reading data from a BFILE.
      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

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/proc/fread.pc */

/* Reading data from 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);
}

#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

```
' This file is installed in the following path when you install
' the database: $ORACLE_HOME/rdbms/demo/lobs/vb/fread.bas

' Reading data from a BFILE

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

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/java/fread.java */

// Reading data from a BFILE.

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 Using SUBSTR

This section describes how to read portion of BFILE data using SUBSTR.

> **See Also:** Table 15–1, "Environments Supported for BFILE APIs" on page 15-2, for a list of operations on BFILEs and APIs provided for each programmatic environment.

Syntax

Use the following syntax references for each programmatic environment:

- PL/SQL (DBMS_LOB) *(PL/SQL Packages and Types Reference):* "DBMS_LOB" — SUBSTR
- C (OCI): A syntax reference is not applicable in this release.
- COBOL (Pro*COBOL) *Pro*COBOL 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++ 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) *(Oracle Database JDBC Developer's Guide and Reference)*: Chapter 7, "Working With LOBs" — Creating and Populating a BLOB or CLOB Column.

Examples

Examples are provided in these five programmatic environments:

- PL/SQL (DBMS_LOB): Reading a Portion of BFILE Data Using SUBSTR on page 15-67
- C (OCI): No example is provided with this release.
- COBOL (Pro*COBOL): Reading a Portion of BFILE Data Using SUBSTR on page 15-67
- C/C++ (Pro*C/C++): Reading a Portion of BFILE Data Using SUBSTR on page 15-69

- Visual Basic (OO4O): Reading a Portion of BFILE Data Using SUBSTR on page 15-70
- Java (JDBC): Reading a Portion of BFILE Data Using SUBSTR on page 15-71

PL/SQL (DBMS_LOB): Reading a Portion of BFILE Data Using SUBSTR

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/plsql/freadprt.sql */

/* Reading portion of a BFILE data using substr. */
/* 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
   DBMS_OUTPUT.PUT_LINE('------------ LOB SUBSTR EXAMPLE ------------');
   /* 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;
/
show errors;
```

COBOL (Pro*COBOL): Reading a Portion of BFILE Data Using SUBSTR

```
 * This file is installed in the following path when you install
      * the database: $ORACLE_HOME/rdbms/demo/lobs/procob/freadprt.pco

     * Reading portion of a BFILE data using substr.
       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 Using SUBSTR

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/proc/freadprt.pc */

/* Reading portion of a BFILE data 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 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 (OO4O): Reading a Portion of BFILE Data Using SUBSTR

```
' This file is installed in the following path when you install
' the database: $ORACLE_HOME/rdbms/demo/lobs/vb/freadprt.bas

' Reading portion of a BFILE data using substr.
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 Using SUBSTR

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/java/freadprt.java */

// Reading a portion of a BFILE data using substr.

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

This section describes how to compare all or parts of two BFILES.

> **See Also:** Table 15–1, "Environments Supported for BFILE APIs" on page 15-2, for a list of operations on BFILEs and APIs provided for each programmatic environment.

Syntax

Use the following syntax references for each programmatic environment:

- PL/SQL(DBMS_LOB) *(PL/SQL Packages and Types Reference):* "DBMS_LOB" — COMPARE
- C (OCI): A syntax reference is not applicable in this release.
- COBOL (Pro*COBOL) *Pro*COBOL 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++ 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) *(Oracle Database JDBC Developer's Guide and Reference):* Chapter 7, "Working With LOBs" — Creating and Populating a BLOB or CLOB Column.

Examples

Examples are provided in these five programmatic environments:

- PL/SQL (DBMS_LOB): Comparing All or Parts of Two BFILES on page 15-74
- C (OCI): No example is provided with this release.
- COBOL (Pro*COBOL): Comparing All or Parts of Two BFILES on page 15-74
- C/C++ (Pro*C/C++): Comparing All or Parts of Two BFILES on page 15-76
- Visual Basic (OO4O): Comparing All or Parts of Two BFILES on page 15-78
- Java (JDBC): Comparing All or Parts of Two BFILES on page 15-79

PL/SQL (DBMS_LOB): Comparing All or Parts of Two BFILES

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/plsql/fpattern.sql */

/* Checking if a pattern exists in a BFILE using instr
/* 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('MEDIA_DIR', 'keyboard.jpg');
   file_loc2     BFILE;
   Retval         INTEGER;
BEGIN
   DBMS_OUTPUT.PUT_LINE('------------ LOB COMPARE EXAMPLE ------------');
   /* 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;
/
SHOW ERRORS;
```

COBOL (Pro*COBOL): Comparing All or Parts of Two BFILES

```
 * This file is installed in the following path when you install
      * the database: $ORACLE_HOME/rdbms/demo/lobs/procob/fcompare.pco

     * Comparing all or parts of two BFILES.
       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

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/proc/fcompare.pc */

/* Comparing all or parts of two BFILES.
   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

```
' This file is installed in the following path when you install
' the database: $ORACLE_HOME/rdbms/demo/lobs/vb/fcompare.bas

'Comparing all or parts of two BFILES.
'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

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/java/fcompare.java */

// Comparing all or parts of two BFILES.

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 in a BFILE Using INSTR

This section describes how to determine whether a pattern exists in a BFILE using INSTR.

> **See Also:** Table 15–1, "Environments Supported for BFILE APIs" on page 15-2, for a list of operations on BFILEs and APIs provided for each programmatic environment.

Syntax

Use the following syntax references for each programmatic environment:

- PL/SQL (DBMS_LOB) *(PL/SQL Packages and Types Reference):* "DBMS_LOB" — INSTR
- C (OCI): A syntax reference is not applicable in this release.
- COBOL (Pro*COBOL) *Pro*COBOL 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++ 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) *(Oracle Database JDBC Developer's Guide and Reference):* Chapter 7, "Working With LOBs" — Creating and Populating a BLOB or CLOB Column.

Examples

These examples are provided in the following four programmatic environments:

- PL/SQL (DBMS_LOB): Checking If a Pattern Exists in a BFILE Using INSTR on page 15-83
- C (OCI): No example is provided with this release.
- COBOL (Pro*COBOL): Checking If a Pattern Exists in a BFILE Using INSTR on page 15-83
- C/C++ (Pro*C/C++): Checking If a Pattern Exists in a BFILE Using INSTR on page 15-85
- Visual Basic (OO4O): No example is provided with this release.
- Java (JDBC): Checking If a Pattern Exists in a BFILE Using INSTR on page 15-87

PL/SQL (DBMS_LOB): Checking If a Pattern Exists in a BFILE Using INSTR

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/plsql/fcompare.sql */

/* Comparing all or parts of two BFILES.  */
/* 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
   DBMS_OUTPUT.PUT_LINE('------------ BFILE INSTR EXAMPLE ------------');
   /* 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);
   /*  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;
/
SHOW ERRORS;
```

COBOL (Pro*COBOL): Checking If a Pattern Exists in a BFILE Using INSTR

```
 * This file is installed in the following path when you install
      * the database: $ORACLE_HOME/rdbms/demo/lobs/procob/fpattern.pco

     * Checking if a pattern exists in a BFILE using instr
       IDENTIFICATION DIVISION.
       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 in a BFILE Using INSTR

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/proc/fpattern.pc */

/* Checking if a pattern exists in a BFILE using instr
   Pro*C lacks an equivalent embedded SQL form of the DBMS_LOB.INSTR()
   function. However, like SUBSTR() and COMPARE(), Pro*C/C++ can call
   DBMS_LOB.INSTR() from within an anonymous PL/SQL block as shown here: */

#include <sql2oci.h>
#include <stdio.h>
#include <string.h>
#include <sqlca.h>

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

#define PatternSize 5
```

```
void instringBFILE_proc()
{
  OCIBFileLocator *Lob_loc;
  unsigned int Position = 0;
  int 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 occurrance of the pattern starting from the
     beginning of the BFILE using PL/SQL: */
  EXEC SQL EXECUTE
    BEGIN
      :Position := DBMS_LOB.INSTR(:Lob_loc, :Pattern, 1, 1);
    END;
  END-EXEC;
  /* Close the BFILE: */
  EXEC SQL LOB CLOSE :Lob_loc;
  if (0 == Position)
    printf("Pattern not found\n");
  else
    printf("The pattern occurs at %d\n", Position);
  EXEC SQL FREE :Lob_loc;
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
```

```
  instringBFILE_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

Java (JDBC): Checking If a Pattern Exists in a BFILE Using INSTR

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/java/fpattern.java */

// Checking if a pattern exists in a BFILE 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 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();
    }
  }
}
```

Determining Whether a BFILE Exists

This procedure determines whether a BFILE locator points to a valid BFILE instance.

> **See Also:** Table 15–1, "Environments Supported for BFILE APIs" on page 15-2, for a list of operations on BFILEs and APIs provided for each programmatic environment.

Syntax

Use the following syntax references for each programmatic environment:

- PL/SQL (DBMS_LOB) *PL/SQL Packages and Types Reference):* "DBMS_LOB" — FILEEXISTS
- C (OCI) *Oracle Call Interface Programmer's Guide)*: Chapter 7, "LOB and File Operations" for usage notes. "Relational Functions" — LOB Functions, OCILobFileExists
- COBOL (Pro*COBOL) *Pro*COBOL 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++ 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) *Oracle Database JDBC Developer's Guide and Reference:* Chapter 7, "Working With LOBs" — Creating and Populating a BLOB or CLOB Column.

Examples

The examples are provided in the following six programmatic environments:

- PL/SQL (DBMS_LOB): Determining Whether a BFILE Exists on page 15-90
- C (OCI): Determining Whether a BFILE Exists on page 15-90
- COBOL (Pro*COBOL): Determining Whether a BFILE Exists on page 15-91
- C/C++ (Pro*C/C++): Determining Whether a BFILE Exists on page 15-93

- Visual Basic (OO4O): Determining Whether a BFILE Exists on page 15-94
- Java (JDBC): Determining Whether a BFILE Exists on page 15-95

PL/SQL (DBMS_LOB): Determining Whether a BFILE Exists

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/plsql/fexists.sql */

/* Checking if a BFILE exists */
/* Procedure seeIfExistsBFILE_proc is not part of DBMS_LOB package:  */

CREATE OR REPLACE PROCEDURE seeIfExistsBFILE_proc IS
   file_loc      BFILE;
BEGIN
   DBMS_OUTPUT.PUT_LINE('------------ BFILE FILEEXISTS EXAMPLE ------------');
   /* Select the LOB: */
   SELECT ad_graphic INTO File_loc FROM print_media
      WHERE product_id = 3060 AND ad_id = 11001;

   /* 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;
/
SHOW ERRORS;
```

C (OCI): Determining Whether a BFILE Exists

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/oci/fexists.c */

/* Checking if a BFILE exists */
```

```
#include <oratypes.h>
#include <lobdemo.h>
void BfileExists_proc(OCILobLocator *Bfile_loc, OCIEnv *envhp,
                        OCIError *errhp, OCISvcCtx *svchp, OCIStmt *stmthp)
{
   boolean is_exist;

   printf ("----------- OCILobFileExists Demo --------------\n");
   checkerr (errhp, OCILobFileExists(svchp, errhp, Bfile_loc, &is_exist));

   if (is_exist == TRUE)
   {
     printf("File exists\n");
   }
   else
   {
     printf("File does not exist\n");
   }
}
```

COBOL (Pro*COBOL): Determining Whether a BFILE Exists

```
 * This file is installed in the following path when you install
      * the database: $ORACLE_HOME/rdbms/demo/lobs/procob/fexists.pco

     * Checking if a BFILE exists.
       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 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 Whether a BFILE Exists

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/proc/fexists.pc */

/* Checking if a BFILE exists. */

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

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

void seeIfBFILEExists_proc()
{
  OCIBFileLocator *Lob_loc;
  unsigned int Exists = 0;

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  EXEC SQL ALLOCATE :Lob_loc;
  EXEC SQL SELECT 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): Determining Whether a BFILE Exists

```
' This file is installed in the following path when you install
' the database: $ORACLE_HOME/rdbms/demo/lobs/vb/fexists.bas

'Checking if a BFILE exists.
'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 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 OraAdGraphic = OraParameters("MyAdGraphic").Value

If OraAdGraphic.Exists Then
    'Process the data
Else
   'Do error processing
End If
OraDb.Connection.CommitTrans
```

Java (JDBC): Determining Whether a BFILE Exists

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/java/fexists.java */

// Checking if a BFILE exists.

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 object for this BFILE:
           System.out.println("Directory object: " + 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

This section describes how to get the length of a BFILE.

> **See Also:** Table 15–1, "Environments Supported for BFILE APIs" on page 15-2, for a list of operations on BFILEs and APIs provided for each programmatic environment.

Syntax

Use the following syntax references for each programmatic environment:

- PL/SQL (DBMS_LOB) *(PL/SQL Packages and Types Reference):* "DBMS_LOB" — GETLENGTH
- C (OCI) (*Oracle Call Interface Programmer's Guide)*: Chapter 7, "LOB and File Operations". "Relational Functions" — LOB Functions, OCILobGetLength2
- COBOL (Pro*COBOL) *Pro*COBOL 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++ 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 > OraBfile > Examples
- Java (JDBC) *Oracle Database JDBC Developer's Guide and Reference*: Chapter 7, "Working With LOBs" — Creating and Populating a BLOB or CLOB Column.

Examples

The examples are provided in six programmatic environments:

- PL/SQL (DBMS_LOB): Getting the Length of a BFILE on page 15-97
- C (OCI): Getting the Length of a BFILE on page 15-98
- COBOL (Pro*COBOL): Getting the Length of a BFILE on page 15-99
- C/C++ (Pro*C/C++): Getting the Length of a BFILE on page 15-100
- Visual Basic (OO4O): Getting the Length of a BFILE on page 15-102
- Java (JDBC): Getting the Length of a BFILE on page 15-103

PL/SQL (DBMS_LOB): Getting the Length of a BFILE

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/plsql/flength.sql */

/* Getting the length of a BFILE. */
/* Procedure getLengthBFILE_proc is not part of DBMS_LOB package: */

CREATE OR REPLACE PROCEDURE getLengthBFILE_proc IS
   file_loc      BFILE;
   Length       INTEGER;
BEGIN
   DBMS_OUTPUT.PUT_LINE('------------ BFILE LENGTH EXAMPLE ------------');
   /* 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;
/
SHOW ERRORS;
```

C (OCI): Getting the Length of a BFILE

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/oci/flength.c */

/* Getting the length of a BFILE.  */
/* Select the lob/bfile from table Print_media */
#include <oratypes.h>
#include <lobdemo.h>
```

```
void BfileLength_proc(OCILobLocator *Bfile_loc, OCIEnv *envhp,
                      OCIError *errhp, OCISvcCtx *svchp, OCIStmt *stmthp)
{
   oraub8 len;

   printf ("----------- OCILobGetLength BFILE Demo --------------\n");
   checkerr (errhp, OCILobFileOpen(svchp, errhp, Bfile_loc,
                                   (ub1) OCI_FILE_READONLY));

   checkerr (errhp, OCILobGetLength2(svchp, errhp, Bfile_loc, &len));

   printf("Length of bfile = %d\n", (ub4)len);

   checkerr (errhp, OCILobFileClose(svchp, errhp, Bfile_loc));
}
```

COBOL (Pro*COBOL): Getting the Length of a BFILE

```
 * This file is installed in the following path when you install
      * the database: $ORACLE_HOME/rdbms/demo/lobs/procob/flength.pco

     * Getting the length of a BFILE.
       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

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/proc/flength.pc */

/* Getting the length of 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 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 unitialized, then Length is Undefined: */
  printf("Length is %d bytes\n", Length);
  /* Close the BFILE: */
  EXEC SQL LOB CLOSE :Lob_loc;
  EXEC SQL FREE :Lob_loc;
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  getLengthBFILE_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

Visual Basic (OO4O): Getting the Length of a BFILE

```
' This file is installed in the following path when you install
' the database: $ORACLE_HOME/rdbms/demo/lobs/vb/flength.bas

'Getting the length of a BFILE.
'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

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/java/flength.java */

// Getting the length of a BFILE.

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 object for this BFILE:
      System.out.println("Directory object: " + 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();
   }
  }
}
```

Assigning a BFILE Locator

This section describes how to assign one BFILE locator to another.

> **See Also:** Table 15–1, "Environments Supported for BFILE APIs" on page 15-2, for a list of operations on BFILEs and APIs provided for each programmatic environment.

Syntax

Use the following syntax references for each programmatic environment:

- SQL *(Oracle Database SQL Reference):* Chapter 7, "SQL Statements" — CREATE PROCEDURE
- PL/SQL (DBMS_LOB package): Refer to Chapter 5, "Advanced Design Considerations" 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. "Relational Functions" — LOB Functions, OCILobLocatorAssign
- COBOL (Pro*COBOL) *Pro*COBOL 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++ 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*) Oracle Database JDBC Developer's Guide and Reference:* Chapter 7, "Working With LOBs" — Creating and Populating a BLOB or CLOB Column.

Examples

The examples are provided in the following five programmatic environments:

- PL/SQL: Assigning a BFILE Locator on page 15-100
- C (OCI): Assigning a BFILE Locator on page 15-106
- COBOL (Pro*COBOL): Assigning a BFILE Locator on page 15-107
- C/C++ (Pro*C/C++): Assigning a BFILE Locator on page 15-108
- Visual Basic: An example is not provided with this release.

- Java (JDBC): Assigning a BFILE Locator on page 15-109

PL/SQL: Assigning a BFILE Locator

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/plsql/fcopyloc.sql */

/* Copying a LOB locator for a BFILE. */
/* Procedure BFILEAssign_proc is not part of DBMS_LOB package:   */

CREATE OR REPLACE PROCEDURE BFILEAssign_proc IS
   file_loc1    BFILE := BFILENAME('MEDIA_DIR', 'keyboard_logo.jpg');
   file_loc2    BFILE;
BEGIN
   DBMS_OUTPUT.PUT_LINE('------------ BFILE ASSIGN EXAMPLE ------------');
   /*
   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;
/
SHOW ERRORS;
```

C (OCI): Assigning a BFILE Locator

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/oci/fcopyloc.c */

/* Copying a LOB locator for a BFILE.  */
#include <oratypes.h>
#include <lobdemo.h>
void BfileAssign_proc(OCILobLocator *Bfile_loc1, OCILobLocator *Bfile_loc2,
                      OCIEnv *envhp, OCIError *errhp, OCISvcCtx *svchp,
                      OCIStmt *stmthp)
{
```

```
    printf ("----------- OCI BFILE Assign Demo --------------\n");

    checkerr(errhp, OCILobLocatorAssign(svchp, errhp, Bfile_loc1, &Bfile_loc2));
}
```

COBOL (Pro*COBOL): Assigning a BFILE Locator

```
* This file is installed in the following path when you install
     * the database: $ORACLE_HOME/rdbms/demo/lobs/procob/fcopyloc.pco

    * Copying a LOB locator for a BFILE.
      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 SQLERROR DO PERFORM SQL-ERROR END-EXEC.
          EXEC SQL CONNECT :USERID END-EXEC.

     * Allocate and initialize the BFILE locator:
          EXEC SQL ALLOCATE :BFILE1 END-EXEC.
          EXEC SQL ALLOCATE :BFILE2 END-EXEC.

          EXEC SQL WHENEVER NOT FOUND GOTO END-OF-BFILE END-EXEC.
          EXEC SQL
               SELECT 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 SQLERRMC.
    EXEC SQL ROLLBACK WORK RELEASE END-EXEC.
    STOP RUN.
```

C/C++ (Pro*C/C++): Assigning a BFILE Locator

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/proc/fcopyloc.pc */

/* Copying a LOB locator for 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 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): Assigning a BFILE Locator

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/java/fcopyloc.java */

// Copying a LOB locator for a BFILE.

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();
      }
    }
  }
```

Getting Directory Object Name and Filename of a BFILE

This section describes how to get the directory object name and filename of a BFILE.

> **See Also:** Table 15–1, "Environments Supported for BFILE APIs" on page 15-2, for a list of operations on BFILEs and APIs provided for each programmatic environment.

Syntax

Use the following syntax references for each programmatic environment:

- PL/SQL (DBMS_LOB) *(PL/SQL Packages and Types Reference):* "DBMS_LOB" — FILEGETNAME
- C (OCI) (*Oracle Call Interface Programmer's Guide)*: Chapter 7, "LOB and File Operations" for usage notes. "Relational Functions" — LOB Functions, OCILobFileGetName
- COBOL (Pro*COBOL) *Pro*COBOL 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++ Programmer's Guide*): "Large Objects (LOBs)", "LOB Statements", "Embedded SQL Statements and Directives" — LOB DESCRIBE ...GET DIRECTORY ...
- Java (JDBC) *Oracle Database JDBC Developer's Guide and Reference:* Chapter 7, "Working With LOBs" — Creating and Populating a BLOB or CLOB Column.

Examples

Examples of this procedure are provided in the following programmatic environments:

- PL/SQL (DBMS_LOB): Getting Directory Object Name and Filename on page 15-112
- C (OCI): Getting Directory Object Name and Filename on page 15-112
- COBOL (Pro*COBOL): Getting Directory Object Name and Filename on page 15-113
- C/C++ (Pro*C/C++): Getting Directory Object Name and Filename on page 15-114
- Visual Basic (OO4O): Getting Directory Object Name and Filename on page 15-115

- Java (JDBC): Getting Directory Object Name and Filename on page 15-116

PL/SQL (DBMS_LOB): Getting Directory Object Name and Filename

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/plsql/fgetdir.sql */

/* Getting the directory object and filename of a BFILE*/

CREATE OR REPLACE PROCEDURE getNameBFILE_proc IS
   file_loc        BFILE;
   DirAlias_name   VARCHAR2(30);
   File_name       VARCHAR2(40);
BEGIN
   DBMS_OUTPUT.PUT_LINE('------------ BFILE FILEGETNAME EXAMPLE ------------');
   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);
   /* DirAlias_name and File_name now store the directory object and filename */
END;
/
SHOW ERRORS;
```

C (OCI): Getting Directory Object Name and Filename

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/oci/fgetdir.c */

/* Getting the directory object and filename */
#include <oratypes.h>
#include <lobdemo.h>
void BfileGetDir_proc(OCILobLocator *Bfile_loc, OCIEnv *envhp,
                      OCIError *errhp, OCISvcCtx *svchp, OCIStmt *stmthp)
{
   OraText dir_alias[32];
   OraText filename[256];
   ub2 d_length = 32;
   ub2 f_length = 256;
```

```
   printf ("----------- OCILobFileGetName Demo --------------\n");
   checkerr(errhp, OCILobFileGetName(envhp, errhp, Bfile_loc,
                       dir_alias, &d_length, filename, &f_length));

   dir_alias[d_length] = '\0';
   filename[f_length] = '\0';
   printf("Directory object : [%s]\n", dir_alias);
   printf("File name : [%s]\n", filename);
}
```

COBOL (Pro*COBOL): Getting Directory Object Name and Filename

```
 * This file is installed in the following path when you install
      * the database: $ORACLE_HOME/rdbms/demo/lobs/procob/fgetdir.pco

     * Getting the directory object and filename
       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 object 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 Object Name and Filename

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/proc/fgetdir.pc */

/* Getting the directory object and filename */

#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 Object Name and Filename

```
' This file is installed in the following path when you install
' the database: $ORACLE_HOME/rdbms/demo/lobs/vb/fgetdir.bas
```

```
'Getting the directory object and filename
'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 object and filename:
MsgBox " Directory object is " & OraAdGraphic1.DirectoryName &
   " Filename is " & OraAdGraphic1.filename

OraDb.Connection.CommitTrans
```

Java (JDBC): Getting Directory Object Name and Filename

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/java/fgetdir.java */

// Getting the directory object and filename

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 object for this BFILE:
            System.out.println("Directory object: " + 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 by Initializing a BFILE Locator

This section describes how to UPDATE a BFILE by initializing a BFILE locator.

> **See Also:** Table 15–1, "Environments Supported for BFILE APIs" on page 15-2, for a list of operations on BFILEs and APIs provided for each programmatic environment.

Syntax

Use the following syntax references for each programmatic environment:

- PL/SQL (DBMS_LOB): See the *(Oracle Database SQL Reference)*, Chapter 7, "SQL Statements" — UPDATE
- C (OCI) (*Oracle Call Interface Programmer's Guide)*: Chapter 7, "LOB and File Operations" for usage notes. "Relational Functions" — LOB Functions, OCILobFileSetName
- COBOL (Pro*COBOL) *Pro*COBOL Programmer's Guide* for information on LOBs, usage notes on LOB Statements, and embedded SQL and precompiler directives — ALLOCATE. See also *(Oracle Database SQL Reference), Chapter 7, "SQL Statements"* — UPDATE
- C/C++ (Pro*C/C++) (*Pro*C/C++ Programmer's Guide)*: "Large Objects (LOBs)", "LOB Statements", "Embedded SQL Statements and Directives". See also *(Oracle Database 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) *Oracle Database JDBC Developer's Guide and Reference:* Chapter 7, "Working With LOBs" — Creating and Populating a BLOB or CLOB Column.

Examples

The examples are provided in six programmatic environments:

- PL/SQL: Updating a BFILE by Initializing a BFILE Locator on page 15-119
- C (OCI): Updating a BFILE by Initializing a BFILE Locator on page 15-120
- COBOL (Pro*COBOL): Updating a BFILE by Initializing a BFILE Locator on page 15-121

- C/C++ (Pro*C/C++): Updating a BFILE by Initializing a BFILE Locator on page 15-123
- Visual Basic (OO4O): Updating a BFILE by Initializing a BFILE Locator on page 15-124
- Java (JDBC): Updating a BFILE by Initializing a BFILE Locator on page 15-125

PL/SQL: Updating a BFILE by Initializing a BFILE Locator

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/plsql/fupdate.sql */

/* Updating a BFILE by initializing a BFILE locator. */
/* Procedure updateUseBindVariable_proc is not part of DBMS_LOB package: */

CREATE OR REPLACE PROCEDURE updateBFILEColumn_proc IS
   File_loc  BFILE;
BEGIN
   DBMS_OUTPUT.PUT_LINE('------------ BFILE UPDATE EXAMPLE ------------');
   SELECT ad_graphic INTO File_loc
      FROM Print_media
         WHERE product_id = 3060 AND ad_id = 11001;

   UPDATE Print_media SET ad_graphic = File_loc
         WHERE product_id = 3060 AND ad_id = 11001;

END;
/
```

C (OCI): Updating a BFILE by Initializing a BFILE Locator

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/oci/fupdate.c */

/* Updating a BFILE by initializing a BFILE locator. */
#include <oratypes.h>
#include <lobdemo.h>
void BfileUpdate_proc(OCILobLocator *Bfile_loc, OCIEnv *envhp,
                      OCIError *errhp, OCISvcCtx *svchp, OCIStmt *stmthp)
```

```
{
  OCIBind *bndhp, *bndhp2;

  text  *updstmt =
    (text *) "UPDATE Print_media SET ad_graphic = :Lob_loc \
              WHERE product_id = 3107 AND ad_id = 13002";

  OraText *Dir = (OraText *)"MEDIA_DIR",
          *Name = (OraText *)"keyboard_logo.jpg";

  printf ("----------- OCI BFILE Update Demo --------------\n");
  /* Prepare the SQL statement: */
  checkerr (errhp, OCIStmtPrepare(stmthp, errhp, updstmt,  (ub4)
                                  strlen((char *) updstmt),
                                  (ub4) OCI_NTV_SYNTAX, (ub4)OCI_DEFAULT));

  checkerr (errhp, OCILobFileSetName(envhp, errhp, &Bfile_loc,
                                     Dir, (ub2)strlen((char *)Dir),
                                     Name,(ub2)strlen((char *)Name)));

  checkerr (errhp, OCIBindByPos(stmthp, &bndhp, errhp, (ub4) 1,
                                (void *) &Bfile_loc, (sb4) 0,  SQLT_BFILE,
                                (void *) 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));
  printf("Bfile column updated \n");
}
```

COBOL (Pro*COBOL): Updating a BFILE by Initializing a BFILE Locator

```
 * This file is installed in the following path when you install
      * the database: $ORACLE_HOME/rdbms/demo/lobs/procob/fupdate.pco

     * Updating a BFILE by initializing a BFILE locator.
       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

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/proc/fupdate.pc */

/* Updating a BFILE by initializing a BFILE locator. */

#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 (OO4O): Updating a BFILE by Initializing a BFILE Locator

```
' This file is installed in the following path when you install
' the database: $ORACLE_HOME/rdbms/demo/lobs/vb/fupdate.bas

'Updating a BFILE by initializing a BFILE locator.

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 object and filename
'MsgBox " Directory object is " & OraAdGraphic1.DirectoryName & " Filename is "
& OraAdGraphic1.filename

OraDb.Connection.CommitTrans
```

Java (JDBC): Updating a BFILE by Initializing a BFILE Locator

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/java/fupdate.java */

// Updating a BFILE by initializing a BFILE locator.

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

This section describes how to close a BFILE with FILECLOSE.

Note: This function (FILECLOSE) is not recommended for new development. For new development, use the CLOSE function instead. See "Closing a BFILE with CLOSE" on page 15-131 for more information.

See Also: Table 15–1, "Environments Supported for BFILE APIs" on page 15-2, for a list of operations on BFILEs and APIs provided for each programmatic environment.

Syntax

Use the following syntax references for each programmatic environment:

- PL/SQL (DBMS_LOB)*(PL/SQL Packages and Types Reference):* "DBMS_LOB" — FILEOPEN, FILECLOSE
- C (OCI) (*Oracle Call Interface Programmer's Guide)*: Chapter 7, "LOB and File Operations" for usage notes. "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*) Oracle Database JDBC Developer's Guide and Reference:* Chapter 7, "Working With LOBs" — Creating and Populating a BLOB or CLOB Column.

Examples

- PL/SQL (DBMS_LOB): Closing a BFILE with FILECLOSE on page 15-127
- C (OCI): Closing a BFILE with FILECLOSE on page 15-127
- 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): This operation is not supported in Visual Basic. Instead use Visual Basic (OO4O): Closing a BFile with CLOSE as described on page 15-136.

- Java (JDBC): Closing a BFile with FILECLOSE on page 15-129

PL/SQL (DBMS_LOB): Closing a BFILE with FILECLOSE

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/plsql/fclose_f.sql */

/* Closing a BFILE with FILECLOSE.
   Procedure closeBFILE_procOne is not part of DBMS_LOB package: */

CREATE OR REPLACE PROCEDURE closeBFILE_procOne IS
   file_loc      BFILE := BFILENAME('MEDIA_DIR', 'keyboard_logo.jpg');
BEGIN
   DBMS_OUTPUT.PUT_LINE('------------ BFILE FILECLOSE EXAMPLE ------------');
   DBMS_LOB.FILEOPEN(file_loc, DBMS_LOB.FILE_READONLY);
   /* ...Do some processing. */
   DBMS_LOB.FILECLOSE(file_loc);
END;
/
SHOW ERRORS;
```

C (OCI): Closing a BFILE with FILECLOSE

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/oci/fclose_f.c */

/* Closing a BFILE with FILECLOSE. */
#include <oratypes.h>
#include <lobdemo.h>
void BfileFileClose_proc(OCILobLocator *Bfile_loc, OCIEnv *envhp,
                        OCIError *errhp, OCISvcCtx *svchp, OCIStmt *stmthp)

{
   printf ("----------- OCILobFileOpen Demo --------------\n");
   checkerr(errhp, OCILobFileOpen(svchp, errhp, Bfile_loc,
                                  (ub1) OCI_FILE_READONLY));

   checkerr(errhp, OCILobFileClose(svchp, errhp, Bfile_loc));
}
```

Java (JDBC): Closing a BFile with FILECLOSE

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/java/fclose_f.java */

// Closing a BFILE with FILECLOSE.

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

This section describes how to close a BFILE with the CLOSE function.

Note: This function (CLOSE) is recommended for new application development. The older FILECLOSE function, is not recommended for new development.

See Also: Table 15–1, "Environments Supported for BFILE APIs" on page 15-2, for a list of operations on BFILEs and APIs provided for each programmatic environment.

Usage Notes

Opening and closing a BFILE is mandatory. You must close the instance at some point later in the session.

See Also:

- Opening a BFILE with OPEN on page 15-21
- Determining Whether a BFILE Is Open Using ISOPEN on page 15-32

Syntax

Use the following syntax references for each programmatic environment:

- PL/SQL (DBMS_LOB) *(PL/SQL Packages and Types Reference):* "DBMS_LOB" — CLOSE
- C (OCI) (*Oracle Call Interface Programmer's Guide)*: Chapter 7, "LOB and File Operations" for usage notes. "Relational Functions" — LOB Functions, OCILobClose
- COBOL (Pro*COBOL) *Pro*COBOL 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++ 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) *Oracle Database JDBC Developer's Guide and Reference:* Chapter 7, "Working With LOBs" — Creating and Populating a BLOB or CLOB Column.

Examples

- PL/SQL (DBMS_LOB): Closing a BFILE with CLOSE on page 15-132
- C (OCI): Closing a BFile with CLOSE on page 15-133
- COBOL (Pro*COBOL): Closing a BFILE with CLOSE on page 15-133
- C/C++ (Pro*C/C++): Closing a BFile with CLOSE on page 15-135
- Visual Basic (OO4O): Closing a BFile with CLOSE on page 15-136
- Java (JDBC): Closing a BFile with CLOSE on page 15-136

PL/SQL (DBMS_LOB): Closing a BFILE with CLOSE

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/plsql/fclose_c.sql */

/* Closing a BFILE with CLOSE.
   Procedure closeBFILE_procTwo is not part of DBMS_LOB package: */

CREATE OR REPLACE PROCEDURE closeBFILE_procTwo IS
   file_loc      BFILE := BFILENAME('MEDIA_DIR', 'keyboard_logo.jpg');
BEGIN
   DBMS_OUTPUT.PUT_LINE('------------ BFILE CLOSE EXAMPLE ------------');
   DBMS_LOB.OPEN(file_loc, DBMS_LOB.LOB_READONLY);
   /* ...Do some processing. */
   DBMS_LOB.CLOSE(file_loc);
END;
/
SHOW ERRORS;
```

C (OCI): Closing a BFile with CLOSE

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/oci/fclose_c.c */

/* Closing a BFILE with CLOSE.  */
#include <oratypes.h>
#include <lobdemo.h>
void BfileLobClose_proc(OCILobLocator *Bfile_loc, OCIEnv *envhp,
                        OCIError *errhp, OCISvcCtx *svchp, OCIStmt *stmthp)
{
   printf ("----------- OCILobOpen Demo --------------\n");
   checkerr(errhp, OCILobOpen(svchp, errhp, Bfile_loc,
                     (ub1) OCI_LOB_READONLY));

   checkerr(errhp, OCILobClose(svchp, errhp, Bfile_loc));
}
```

COBOL (Pro*COBOL): Closing a BFILE with CLOSE

```
 * This file is installed in the following path when you install
      * the database: $ORACLE_HOME/rdbms/demo/lobs/procob/fclose_c.pco

    * Closing a BFILE with CLOSE.
      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

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/proc/fclose_c.pc */

/* Closing a BFILE with CLOSE.
   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

```
' This file is installed in the following path when you install
' the database: $ORACLE_HOME/rdbms/demo/lobs/vb/fclose_c.bas

'Closing a BFILE with CLOSE.

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

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/java/fclose_c.java */

// Closing a BFILE with CLOSE.

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

This section describes how to close all open BFILEs.

You are responsible for closing any BFILE instances at some point before your program terminates. For example, you must close any open BFILE instance before the termination of a PL/SQL block or OCI program.

You must close open BFILE instances even in cases where an exception or unexpected termination of your application occurs. In these cases, if a BFILE instance is not closed, then it is still considered open by the database. Ensure that your exception handling strategy does not allow BFILE instances to remain open in these situations.

See Also:

- Table 15–1, "Environments Supported for BFILE APIs" on page 15-2, for a list of operations on BFILEs and APIs provided for each programmatic environment.
- "Setting Maximum Number of Open BFILEs" on page 3-7

Syntax

Use the following syntax references for each programmatic environment:

- PL/SQL (DBMS_LOB) *(PL/SQL Packages and Types Reference):* "DBMS_LOB" — FILECLOSEALL
- C (OCI) (*Oracle Call Interface Programmer's Guide)*: Chapter 7, "LOB and File Operations" for usage notes. "Relational Functions" — LOB Functions, OCILobFileCloseAll
- COBOL (Pro*COBOL) *Pro*COBOL 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++ 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) *Oracle Database JDBC Developer's Guide and Reference:* Chapter 7, "Working With LOBs" — Creating and Populating a BLOB or CLOB Column.

Examples

- PL/SQL (DBMS_LOB): Closing All Open BFiles on page 15-140
- C (OCI): Closing All Open BFiles on page 15-140
- COBOL (Pro*COBOL): Closing All Open BFiles on page 15-141
- C/C++ (Pro*C/C++): Closing All Open BFiles on page 15-143
- Visual Basic (OO4O): Closing All Open BFiles on page 15-144
- Java (JDBC): Closing All Open BFiles on page 15-145

PL/SQL (DBMS_LOB): Closing All Open BFiles

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/plsql/fclosea.sql */

/* Closing all open BFILEs.
   Procedure closeAllOpenFilesBFILE_proc is not part of DBMS_LOB package: */

CREATE OR REPLACE PROCEDURE closeAllOpenBFILEs_proc IS
BEGIN
   DBMS_OUTPUT.PUT_LINE('------------ BFILE CLOSEALL EXAMPLE ------------');
   /* Close all open BFILEs: */
   DBMS_LOB.FILECLOSEALL;
END;
/
SHOW ERRORS;
```

C (OCI): Closing All Open BFiles

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/oci/fclosea.c */

/* Closing all open BFILEs. */
#include <oratypes.h>
```

```
#include <lobdemo.h>
void BfileCloseAll_proc(OCILobLocator *Bfile_loc1, OCILobLocator *Bfile_loc2,
                        OCIEnv *envhp, OCIError *errhp, OCISvcCtx *svchp,
                        OCIStmt *stmthp)
{
   printf ("----------- OCILobFileCloseAll Demo --------------\n");
   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));
}
```

COBOL (Pro*COBOL): Closing All Open BFiles

```
 * This file is installed in the following path when you install
      * the database: $ORACLE_HOME/rdbms/demo/lobs/procob/fclosea.pco

     * Closing all open BFILEs.
       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

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/proc/fclosea.pc */

/* Closing all open BFILEs. */

#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 (OO4O): Closing All Open BFiles

```
' This file is installed in the following path when you install
' the database: $ORACLE_HOME/rdbms/demo/lobs/vb/fclosea.bas

'Closing all open BFILEs.

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

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/java/fclosea.java */

// Closing all open BFILEs.

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();
         }
      }
   }
```

Inserting a Row Containing a BFILE

This section describes how to INSERT a row containing a BFILE by initializing a BFILE locator.

See Also:

- Table 15–1, "Environments Supported for BFILE APIs" on page 15-2, for a list of operations on BFILEs and APIs provided for each programmatic environment.

Usage Notes

You must initialize the BFILE locator bind variable to NULL or a directory object and filename before issuing the INSERT statement.

Syntax

See the following syntax references for each programmatic environment:

- SQL(*Oracle Database SQL Reference*, Chapter 7 "SQL Statements" — INSERT
- C (OCI) *Oracle Call Interface Programmer's Guide*: Chapter 7, "LOB and File Operations". "Relational Functions" — LOB Functions.
- COBOL (Pro*COBOL) *Pro*COBOL Programmer's Guide* for information on LOBs, usage notes on LOB Statements, embedded SQL, and precompiler directives. See also *Oracle Database SQL Reference*, for related information on the SQL INSERT statement.
- C/C++ (Pro*C/C++) *Pro*C/C++ Programmer's Guide*: "Large Objects (LOBs)", "LOB Statements", "Embedded SQL Statements and Directives" — LOB FILE SET. See also (*Oracle Database 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) *Oracle Database JDBC Developer's Guide and Reference*: Chapter 7, "Working With LOBs" — Creating and Populating a BLOB or CLOB Column.

Examples

Examples in the following programmatic environments are provided:

- PL/SQL (DBMS_LOB): Inserting a Row Containing a BFILE on page 15-149
- C (OCI): Inserting a Row Containing a BFILE on page 15-149
- COBOL (Pro*COBOL): Inserting a Row Containing a BFILE on page 15-150
- C/C++ (Pro*C/C++): Inserting a Row Containing a BFILE on page 15-152
- Visual Basic (OO4O): Inserting a Row Containing a BFILE on page 15-153
- Java (JDBC): Inserting a Row Containing a BFILE on page 15-154

PL/SQL (DBMS_LOB): Inserting a Row Containing a BFILE

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/plsql/finsert.sql */

/* Inserting  row containing a BFILE by initializing a BFILE locator */

CREATE OR REPLACE PROCEDURE insertBFILE_proc IS
  /* Initialize the BFILE locator: */
  Lob_loc  BFILE := BFILENAME('MEDIA_DIR', 'keyboard_logo.jpg');
BEGIN
    DBMS_OUTPUT.PUT_LINE('------------ BFILE INSERT EXAMPLE ------------');
    INSERT INTO print_media
    (product_id, ad_id, ad_graphic) VALUES (3106, 13002, Lob_loc);
END;
/
```

C (OCI): Inserting a Row Containing a BFILE

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/oci/finsert.c */

/* Inserting a row by initializing a BFILE Locator. */
#include <oratypes.h>
#include <lobdemo.h>
void BfileInsert_proc(OCILobLocator *Bfile_loc, OCIEnv *envhp,
                      OCIError *errhp, OCISvcCtx *svchp, OCIStmt *stmthp)

{
```

```
  text  *insstmt =
     (text *) "INSERT INTO Print_media (product_id, ad_id, ad_graphic) \
        VALUES (2056, 60315, :Lob_loc)";
  OCIBind *bndhp;
  OraText *Dir = (OraText *)"MEDIA_DIR", *Name = (OraText *)"keyboard_logo.jpg";

  printf ("----------- OCI BFILE Insert Demo --------------\n");
  /* Prepare the SQL statement: */
  checkerr (errhp, OCIStmtPrepare(stmthp, errhp, insstmt,  (ub4)
                                  strlen((char *) insstmt),
                                  (ub4) OCI_NTV_SYNTAX, (ub4)OCI_DEFAULT));

  checkerr (errhp, OCILobFileSetName(envhp, errhp, &Bfile_loc,
                                     Dir, (ub2)strlen((char *)Dir),
                                     Name,(ub2)strlen((char *)Name)));
  checkerr (errhp, OCIBindByPos(stmthp, &bndhp, errhp, (ub4) 1,
                                (void *) &Bfile_loc, (sb4) 0,  SQLT_BFILE,
                                (void *) 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));
}
```

COBOL (Pro*COBOL): Inserting a Row Containing a BFILE

```
 * This file is installed in the following path when you install
      * the database: $ORACLE_HOME/rdbms/demo/lobs/procob/finsert.pco

     * Inserting a row containing a BFILE by initializing a BFILE
       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 object 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

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/proc/finsert.pc */

/* Inserting a row containing a BFILE by initializing 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 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

```
' This file is installed in the following path when you install
' the database: $ORACLE_HOME/rdbms/demo/lobs/vb/finsert.bas

' Inserting a row containing a BFILE by initializing a BFILE.

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

```
/* This file is installed in the following path when you install */
/* the database: $ORACLE_HOME/rdbms/demo/lobs/java/finsert.java */

// Inserting a row containing a BFILE by initializing 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 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();
    }
  }
}
```

A

LOB Demonstration Files

This appendix describes files distributed with the database that demonstrate how LOBs are used in supported programmatic environments.

- PL/SQL LOB Demonstration Files
- OCI LOB Demonstration Files
- Pro*COBOL LOB Demonstration Files
- Pro*C LOB Demonstration Files
- Visual Basic OO4O LOB Demonstration Files
- Java LOB Demonstration Files

PL/SQL LOB Demonstration Files

The following table lists PL/SQL demonstration files. These files are installed in $ORACLE_HOME/rdbms/demo/lobs/plsql/.

Table A–1 PL/SQL Demonstration Examples

File Name	Description	Usage Information
fclose_c.sql	Closing a BFILE with CLOSE	Closing a BFILE with CLOSE on page 15-131
fclose_f.sql	Closing a BFILE with FILECLOSE	Closing a BFILE with FILECLOSE on page 15-127
fclosea.sql	Closing all open BFILEs	Closing All Open BFILEs with FILECLOSEALL on page 15-139
fcompare.sql	Comparing all or parts of two BFILES	Comparing All or Parts of Two BFILES on page 15-73
fcopyloc.sql	Copying a LOB locator for a BFILE	Assigning a BFILE Locator on page 15-105
fdisplay.sql	Displaying BFILE data	Displaying BFILE Data on page 15-46
fexists.sql	Checking if a BFILE exists	Determining Whether a BFILE Exists on page 15-89
ffilopen.sql	Opening a BFILE with FILEOPEN	Opening a BFILE with FILEOPEN on page 15-28
ffisopen.sql	Checking if the BFILE is OPEN with FILEISOPEN	Determining Whether a BFILE Is Open with FILEISOPEN on page 15-41
fgetdir.sql	Getting the directory object name and filename of a BFILE	Getting Directory Object Name and Filename of a BFILE on page 15-111
finsert.sql	Inserting row containing a BFILE by initializing a BFILE locator	Inserting a Row Containing a BFILE on page 15-148
fisopen.sql	Checking if the BFILE is open with ISOPEN	Determining Whether a BFILE Is Open Using ISOPEN on page 15-32
flength.sql	Getting the length of a BFILE	Getting the Length of a BFILE on page 15-97
floadlob.sql	Loading a LOB with BFILE data	Loading a LOB with BFILE Data on page 15-13
fopen.sql	Opening a BFILE with OPEN	Opening a BFILE with OPEN on page 15-21

Table A–1 PL/SQL Demonstration Examples

File Name	Description	Usage Information
fpattern.sql	Checking if a pattern exists in a BFILE using instr	Checking If a Pattern Exists in a BFILE Using INSTR on page 15-82
fread.sql	Reading data from a BFILE	Reading Data from a BFILE on page 15-56
freadprt.sql	Reading portion of a BFILE data using substr	Reading a Portion of BFILE Data Using SUBSTR on page 15-66
fupdate.sql	Updating a BFILE by initializing a BFILE locator	Updating a BFILE by Initializing a BFILE Locator on page 15-119
lappend.sql	Appending one LOB to another	Appending One LOB to Another on page 14-4
lcompare.sql	Comparing all or part of LOB	Comparing All or Part of Two LOBs on page 14-71
lcopy.sql	Copying all or part of a LOB to another LOB	Copying All or Part of One LOB to Another LOB on page 14-93
lcopyloc.sql	Copying a LOB locator	Copying a LOB Locator on page 14-103
ldisplay.sql	Displaying LOB data	Displaying LOB Data on page 14-42
lerase.sql	Erasing part of a LOB	Erasing Part of a LOB on page 14-154
linsert.sql	Inserting a row by initializing LOB locator bind variable	Inserting a Row by Initializing a LOB Locator Bind Variable on page 8-8
linstr.sql	Seeing if pattern exists in LOB (instr)	Patterns: Checking for Patterns in a LOB Using INSTR on page 14-79
lisopen.sql	Seeing if LOB is open	Determining Whether a LOB is Open on page 14-34
listemp.sql	Seeing if LOB is temporary	Determining Whether a LOB instance Is Temporary on page 14-178
lldblobf.sql	Using DBMS_LOB.LOADBLOBFROMFILE to load a BLOB with data from a BILE	Loading a BLOB with Data from a BFILE on page 14-26
lldclobf.sql	Using DBMS_LOB.LOADCLOBFROMFILE to load a CLOB or NCLOB with data from a BILE	Loading a CLOB or NCLOB with Data from a BFILE on page 14-29
lldclobs.sql	Using DBMS_LOB.LOADCLOBFROMFILE to load segments of a stream of data from a BFILE into different CLOBs	Loading a CLOB or NCLOB with Data from a BFILE on page 14-29
llength.sql	Getting the length of a LOB	Length: Determining the Length of a LOB on page 14-86

Table A–1 PL/SQL Demonstration Examples

File Name	Description	Usage Information
lloaddat.sql	Loading a LOB with BFILE data	Loading a LOB with Data from a BFILE on page 14-17
lobuse.sql	Examples of LOB API usage.	Creating Persistent and Temporary LOBs in PL/SQL on page 12-4
lread.sql	Reading data from LOB	Reading Data from a LOB on page 14-52
lsubstr.sql	Reading portion of LOB (substr)	Reading a Portion of a LOB (SUBSTR) on page 14-63
ltrim.sql	Trimming LOB data	Trimming LOB Data on page 14-143
lwrite.sql	Writing data to a LOB	Writing Data to a LOB on page 14-128
lwriteap.sql	Writing to the end of LOB (write append)	Appending to a LOB on page 14-120

OCI LOB Demonstration Files

The following table lists OCI demonstration files. These files are installed in $ORACLE_HOME/rdbms/demo/lobs/oci/.

Table A–2 OCI Demonstration Examples

File Name	Description	Usage Information
fclose_c.c	Closing a BFILE with CLOSE	Closing a BFILE with CLOSE on page 15-131
fclose_f.c	Closing a BFILE with FILECLOSE	Closing a BFILE with FILECLOSE on page 15-127
fclosea.c	Closing all open BFILEs	Closing All Open BFILEs with FILECLOSEALL on page 15-139
fcopyloc.c	Copying a LOB locator for a BFILE	Assigning a BFILE Locator on page 15-105
fdisplay.c	Displaying BFILE data	Displaying BFILE Data on page 15-46
fexists.c	Checking if a BFILE exists	Determining Whether a BFILE Exists on page 15-89
ffilopen.c	Opening a BFILE with FILEOPEN	Opening a BFILE with FILEOPEN on page 15-28
ffisopen.c	Checking if the BFILE is OPEN with FILEISOPEN	Determining Whether a BFILE Is Open with FILEISOPEN on page 15-41

Table A–2 OCI Demonstration Examples

File Name	Description	Usage Information
fgetdir.c	Getting the directory object name and filename of a BFILE	Getting Directory Object Name and Filename of a BFILE on page 15-111
finsert.c	Inserting row containing a BFILE by initializing a BFILE locator	Inserting a Row Containing a BFILE on page 15-148
fisopen.c	Checking if the BFILE is open with ISOPEN	Determining Whether a BFILE Is Open Using ISOPEN on page 15-32
flength.c	Getting the length of a BFILE	Getting the Length of a BFILE on page 15-97
floadlob.c	Loading a LOB with BFILE data	Loading a LOB with BFILE Data on page 15-13
fopen.c	Opening a BFILE with OPEN	Opening a BFILE with OPEN on page 15-21
fread.c	Reading data from a BFILE	Reading Data from a BFILE on page 15-56
fupdate.c	Updating a BFILE by initializing a BFILE locator	Updating a BFILE by Initializing a BFILE Locator on page 15-119
lappend.c	Appending one LOB to another	Appending One LOB to Another on page 14-4
lcopy.c	Copying all or part of a LOB to another LOB	Copying All or Part of One LOB to Another LOB on page 14-93
lcopyloc.c	Copying a LOB locator	Copying a LOB Locator on page 14-103
ldisbuf.c	Disabling LOB buffering (persistent LOBs)	Disabling LOB Buffering on page 14-171
ldisplay.c	Displaying LOB data	Displaying LOB Data on page 14-42
lequal.c	Seeing if one LOB locator is equal to another	Equality: Checking If One LOB Locator Is Equal to Another on page 14-111
lerase.c	Erasing part of a LOB	Erasing Part of a LOB on page 14-154
lgetchar.c	Getting character set id	Determining Character Set ID on page 14-15
lgetchfm.c	Getting character set form of the foreign language ad text, ad_fltextn	Determining Character Set Form on page 14-13
linit.c	Seeing if a LOB locator is initialized	Determining Whether LOB Locator Is Initialized on page 14-117

Table A–2 OCI Demonstration Examples

File Name	Description	Usage Information
linsert.c	Inserting a row by initializing LOB locator bind variable	Inserting a Row by Initializing a LOB Locator Bind Variable on page 8-8
lisopen.c	Seeing if LOB is open	Determining Whether a LOB is Open on page 14-34
listemp.c	Seeing if LOB is temporary	Determining Whether a LOB instance Is Temporary on page 14-178
llength.c	Getting the length of a LOB	Length: Determining the Length of a LOB on page 14-86
lloaddat.c	Loading a LOB with BFILE data	Loading a LOB with Data from a BFILE on page 14-17
lread.c	Reading data from LOB	Reading Data from a LOB on page 14-52
ltrim.c	Trimming LOB data	Trimming LOB Data on page 14-143
lwrite.c	Writing data to a LOB	Writing Data to a LOB on page 14-128
lwriteap.c	Writing to the end of LOB (write append)	Appending to a LOB on page 14-120

Pro*COBOL LOB Demonstration Files

The following table lists Pro*COBOL demonstration files. These files are installed in $ORACLE_HOME/rdbms/demo/lobs/procob/.

Table A–3 Pro*COBOL Demonstration Examples

File Name	Description	Usage Information
fclose_c.pco	Closing a BFILE with CLOSE	Closing a BFILE with CLOSE on page 15-131
fclosea.pco	Closing all open BFILEs	Closing All Open BFILEs with FILECLOSEALL on page 15-139
fcompare.pco	Comparing all or parts of two BFILES	Comparing All or Parts of Two BFILES on page 15-73
fcopyloc.pco	Copying a LOB locator for a BFILE	Assigning a BFILE Locator on page 15-105
fdisplay.pco	Displaying BFILE data	Displaying BFILE Data on page 15-46

*Table A–3 Pro*COBOL Demonstration Examples*

File Name	Description	Usage Information
fexists.pco	Checking if a BFILE exists	Determining Whether a BFILE Exists on page 15-89
fgetdir.pco	Getting the directory object name and filename of a BFILE	Getting Directory Object Name and Filename of a BFILE on page 15-111
finsert.pco	Inserting row containing a BFILE by initializing a BFILE locator	Inserting a Row Containing a BFILE on page 15-148
fisopen.pco	Checking if the BFILE is open with ISOPEN	Determining Whether a BFILE Is Open Using ISOPEN on page 15-32
flength.pco	Getting the length of a BFILE	Getting the Length of a BFILE on page 15-97
floadlob.pco	Loading a LOB with BFILE data	Loading a LOB with BFILE Data on page 15-13
fopen.pco	Opening a BFILE with OPEN	Opening a BFILE with OPEN on page 15-21
fpattern.pco	Checking if a pattern exists in a BFILE using instr	Checking If a Pattern Exists in a BFILE Using INSTR on page 15-82
fread.pco	Reading data from a BFILE	Reading Data from a BFILE on page 15-56
freadprt.pco	Reading portion of a BFILE data using substr	Reading a Portion of BFILE Data Using SUBSTR on page 15-66
fupdate.pco	Updating a BFILE by initializing a BFILE locator	Updating a BFILE by Initializing a BFILE Locator on page 15-119
lappend.pco	Appending one LOB to another	Appending One LOB to Another on page 14-4
lcompare.pco	Comparing all or part of LOB	Comparing All or Part of Two LOBs on page 14-71
lcopy.pco	Copying all or part of a LOB to another LOB	Copying All or Part of One LOB to Another LOB on page 14-93
lcopyloc.pco	Copying a LOB locator	Copying a LOB Locator on page 14-103
ldisbuf.pco	Disabling LOB buffering (persistent LOBs)	Disabling LOB Buffering on page 14-171

*Table A–3 Pro*COBOL Demonstration Examples*

File Name	Description	Usage Information
ldisplay.pco	Displaying LOB data	Displaying LOB Data on page 14-42
lenbuf.pco	Enabling LOB buffering	Enabling LOB Buffering on page 14-162
lerase.pco	Erasing part of a LOB	Erasing Part of a LOB on page 14-154
lflbuf.pco	Flushing the LOB buffer (persistent LOBs)	Flushing the Buffer on page 14-167
linsert.pco	Inserting a row by initializing LOB locator bind variable	Inserting a Row by Initializing a LOB Locator Bind Variable on page 8-8
linstr.pco	Seeing if pattern exists in LOB (instr)	Patterns: Checking for Patterns in a LOB Using INSTR on page 14-79
lisopen.pco	Seeing if LOB is open	Determining Whether a LOB is Open on page 14-34
listemp.pco	Seeing if LOB is temporary	Determining Whether a LOB instance Is Temporary on page 14-178
llength.pco	Getting the length of a LOB	Length: Determining the Length of a LOB on page 14-86
lloaddat.pco	Loading a LOB with BFILE data	Loading a LOB with Data from a BFILE on page 14-17
lread.pco	Reading data from LOB	Reading Data from a LOB on page 14-52
lsubstr.pco	Reading portion of LOB (substr)	Reading a Portion of a LOB (SUBSTR) on page 14-63
ltrim.pco	Trimming LOB data	Trimming LOB Data on page 14-143
lwrite.pco	Writing data to a LOB	Writing Data to a LOB on page 14-128
lwriteap.pco	Writing to the end of LOB (write append)	Appending to a LOB on page 14-120

Pro*C LOB Demonstration Files

The following table lists Pro*C demonstration files. These files are installed in $ORACLE_HOME/rdbms/demo/lobs/proc/.

Table A–4 Pro*C Demonstration Examples

File Name	Description	Usage Information
fclose_c.pc	Closing a BFILE with CLOSE	Closing a BFILE with CLOSE on page 15-131
fclosea.pc	Closing all open BFILEs	Closing All Open BFILEs with FILECLOSEALL on page 15-139
fcompare.pc	Comparing all or parts of two BFILES	Comparing All or Parts of Two BFILES on page 15-73
fcopyloc.pc	Copying a LOB locator for a BFILE	Assigning a BFILE Locator on page 15-105
fdisplay.pc	Displaying BFILE data	Displaying BFILE Data on page 15-46
fexists.pc	Checking if a BFILE exists	Determining Whether a BFILE Exists on page 15-89
fgetdir.pc	Getting the directory object name and filename of a BFILE	Getting Directory Object Name and Filename of a BFILE on page 15-111
finsert.pc	Inserting row containing a BFILE by initializing a BFILE locator	Inserting a Row Containing a BFILE on page 15-148
fisopen.pc	Checking if the BFILE is open with ISOPEN	Determining Whether a BFILE Is Open Using ISOPEN on page 15-32
flength.pc	Getting the length of a BFILE	Getting the Length of a BFILE on page 15-97
floadlob.pc	Loading a LOB with BFILE data	Loading a LOB with BFILE Data on page 15-13
fopen.pc	Opening a BFILE with OPEN	Opening a BFILE with OPEN on page 15-21
fpattern.pc	Checking if a pattern exists in a BFILE using instr	Checking If a Pattern Exists in a BFILE Using INSTR on page 15-82

*Table A–4 Pro*C Demonstration Examples*

File Name	Description	Usage Information
fread.pc	Reading data from a BFILE	Reading Data from a BFILE on page 15-56
freadprt.pc	Reading portion of a BFILE data using substr	Reading a Portion of BFILE Data Using SUBSTR on page 15-66
fupdate.pc	Updating a BFILE by initializing a BFILE locator	Updating a BFILE by Initializing a BFILE Locator on page 15-119
lappend.pc	Appending one LOB to another	Appending One LOB to Another on page 14-4
lcompare.pc	Comparing all or part of LOB	Comparing All or Part of Two LOBs on page 14-71
lcopy.pc	Copying all or part of a LOB to another LOB	Copying All or Part of One LOB to Another LOB on page 14-93
lcopyloc.pc	Copying a LOB locator	Copying a LOB Locator on page 14-103
ldisbuf.pc	Disabling LOB buffering (persistent LOBs)	Disabling LOB Buffering on page 14-171
ldisplay.pc	Displaying LOB data	Displaying LOB Data on page 14-42
lenbuf.pc	Enabling LOB buffering	Enabling LOB Buffering on page 14-162
lequal.pc	Seeing if one LOB locator is equal to another	Equality: Checking If One LOB Locator Is Equal to Another on page 14-111
lerase.pc	Erasing part of a LOB	Erasing Part of a LOB on page 14-154
lflbuf.pc	Flushing the LOB buffer (persistent LOBs)	Flushing the Buffer on page 14-167
linit.pc	Seeing if a LOB locator is initialized	Determining Whether LOB Locator Is Initialized on page 14-117
linsert.pc	Inserting a row by initializing LOB locator bind variable	Inserting a Row by Initializing a LOB Locator Bind Variable on page 8-8

Table A–4 Pro*C Demonstration Examples

File Name	Description	Usage Information
linstr.pc	Seeing if pattern exists in LOB (instr)	Patterns: Checking for Patterns in a LOB Using INSTR on page 14-79
lisopen.pc	Seeing if LOB is open	Determining Whether a LOB is Open on page 14-34
listemp.pc	Seeing if LOB is temporary	Determining Whether a LOB instance Is Temporary on page 14-178
llength.pc	Getting the length of a LOB	Length: Determining the Length of a LOB on page 14-86
lread.pc	Reading data from LOB	Reading Data from a LOB on page 14-52
lsubstr.pc	Reading portion of LOB (substr)	Reading a Portion of a LOB (SUBSTR) on page 14-63
ltrim.pc	Trimming LOB data	Trimming LOB Data on page 14-143
lwrite.pc	Writing data to a LOB	Writing Data to a LOB on page 14-128
lwriteap.pc	Writing to the end of LOB (write append)	Appending to a LOB on page 14-120

Visual Basic OO4O LOB Demonstration Files

The following table lists Visual Basic OO4O demonstration files. These files are installed in $ORACLE_HOME/rdbms/demo/lobs/vb/.

Table A–5 Visual Basic OO4O Demonstration Examples

File Name	Description	Usage Information
fclose_c.bas	Closing a BFILE with CLOSE	Closing a BFILE with CLOSE on page 15-131
fclosea.bas	Closing all open BFILEs	Closing All Open BFILEs with FILECLOSEALL on page 15-139
fcompare.bas	Comparing all or parts of two BFILES	Comparing All or Parts of Two BFILES on page 15-73
fdisplay.bas	Displaying BFILE data	Displaying BFILE Data on page 15-46

Table A–5 Visual Basic OO4O Demonstration Examples

File Name	Description	Usage Information
fexists.bas	Checking if a BFILE exists	Determining Whether a BFILE Exists on page 15-89
fgetdir.bas	Getting the directory object name and filename of a BFILE	Getting Directory Object Name and Filename of a BFILE on page 15-111
finsert.bas	Inserting row containing a BFILE by initializing a BFILE locator	Inserting a Row Containing a BFILE on page 15-148
fisopen.bas	Checking if the BFILE is open with ISOPEN	Determining Whether a BFILE Is Open Using ISOPEN on page 15-32
flength.bas	Getting the length of a BFILE	Getting the Length of a BFILE on page 15-97
floadlob.bas	Loading a LOB with BFILE data	Loading a LOB with BFILE Data on page 15-13
fopen.bas	Opening a BFILE with OPEN	Opening a BFILE with OPEN on page 15-21
fread.bas	Reading data from a BFILE	Reading Data from a BFILE on page 15-56
freadprt.bas	Reading portion of a BFILE data using substr	Reading a Portion of BFILE Data Using SUBSTR on page 15-66
fupdate.bas	Updating a BFILE by initializing a BFILE locator	Updating a BFILE by Initializing a BFILE Locator on page 15-119
lappend.bas	Appending one LOB to another	Appending One LOB to Another on page 14-4
lcompare.bas	Comparing all or part of LOB	Comparing All or Part of Two LOBs on page 14-71
lcopy.bas	Copying all or part of a LOB to another LOB	Copying All or Part of One LOB to Another LOB on page 14-93
lcopyloc.bas	Copying a LOB locator	Copying a LOB Locator on page 14-103
ldisbuf.bas	Disabling LOB buffering (persistent LOBs)	Disabling LOB Buffering on page 14-171
ldisplay.bas	Displaying LOB data	Displaying LOB Data on page 14-42
lenbuf.bas	Enabling LOB buffering	Enabling LOB Buffering on page 14-162
lerase.bas	Erasing part of a LOB	Erasing Part of a LOB on page 14-154

Table A–5 Visual Basic OO4O Demonstration Examples

File Name	Description	Usage Information
linsert.bas	Inserting a row by initializing LOB locator bind variable	Inserting a Row by Initializing a LOB Locator Bind Variable on page 8-8
llength.bas	Getting the length of a LOB	Length: Determining the Length of a LOB on page 14-86
lloaddat.bas	Loading a LOB with BFILE data	Loading a LOB with Data from a BFILE on page 14-17
lread.bas	Reading data from LOB	Reading Data from a LOB on page 14-52
lsubstr.bas	Reading portion of LOB (substr)	Reading a Portion of a LOB (SUBSTR) on page 14-63
ltrim.bas	Trimming LOB data	Trimming LOB Data on page 14-143
lwrite.bas	Writing data to a LOB	Writing Data to a LOB on page 14-128

Java LOB Demonstration Files

The following table lists Java demonstration files. These files are installed in $ORACLE_HOME/rdbms/demo/lobs/java/.

Table A–6 Java Demonstration Examples

File Name	Description	Usage Information
fclose_c.java	Closing a BFILE with CLOSE	Closing a BFILE with CLOSE on page 15-131
fclose_f.java	Closing a BFILE with FILECLOSE	Closing a BFILE with FILECLOSE on page 15-127
fclosea.java	Closing all open BFILEs	Closing All Open BFILEs with FILECLOSEALL on page 15-139
fcompare.java	Comparing all or parts of two BFILES	Comparing All or Parts of Two BFILES on page 15-73
fcopyloc.java	Copying a LOB locator for a BFILE	Assigning a BFILE Locator on page 15-105
fdisplay.java	Displaying BFILE data	Displaying BFILE Data on page 15-46

Table A–6 Java Demonstration Examples

File Name	Description	Usage Information
fexists.java	Checking if a BFILE exists	Determining Whether a BFILE Exists on page 15-89
ffilopen.java	Opening a BFILE with FILEOPEN	Opening a BFILE with FILEOPEN on page 15-28
ffisopen.java	Checking if the BFILE is OPEN with FILEISOPEN	Determining Whether a BFILE Is Open with FILEISOPEN on page 15-41
fgetdir.java	Getting the directory object name and filename of a BFILE	Getting Directory Object Name and Filename of a BFILE on page 15-111
finsert.java	Inserting row containing a BFILE by initializing a BFILE locator	Inserting a Row Containing a BFILE on page 15-148
fisopen.java	Checking if the BFILE is open with ISOPEN	Determining Whether a BFILE Is Open Using ISOPEN on page 15-32
flength.java	Getting the length of a BFILE	Getting the Length of a BFILE on page 15-97
fopen.java	Opening a BFILE with OPEN	Opening a BFILE with OPEN on page 15-21
fpattern.java	Checking if a pattern exists in a BFILE using instr	Checking If a Pattern Exists in a BFILE Using INSTR on page 15-82
fread.java	Reading data from a BFILE	Reading Data from a BFILE on page 15-56
freadprt.java	Reading portion of a BFILE data using substr	Reading a Portion of BFILE Data Using SUBSTR on page 15-66
fupdate.java	Updating a BFILE by initializing a BFILE locator	Updating a BFILE by Initializing a BFILE Locator on page 15-119
lappend.java	Appending one LOB to another	Appending One LOB to Another on page 14-4
lcompare.java	Comparing all or part of LOB	Comparing All or Part of Two LOBs on page 14-71
lcopy.java	Copying all or part of a LOB to another LOB	Copying All or Part of One LOB to Another LOB on page 14-93
lcopyloc.java	Copying a LOB locator	Copying a LOB Locator on page 14-103

Table A–6 Java Demonstration Examples

File Name	Description	Usage Information
ldisplay.java	Displaying LOB data	Displaying LOB Data on page 14-42
lequal.java	Seeing if one LOB locator is equal to another	Equality: Checking If One LOB Locator Is Equal to Another on page 14-111
lerase.java	Erasing part of a LOB	Erasing Part of a LOB on page 14-154
linsert.java	Inserting a row by initializing LOB locator bind variable	Inserting a Row by Initializing a LOB Locator Bind Variable on page 8-8
linstr.java	Seeing if pattern exists in LOB (instr)	Patterns: Checking for Patterns in a LOB Using INSTR on page 14-79
lisopen.java	Seeing if LOB is open	Determining Whether a LOB is Open on page 14-34
listempb.java	Seeing if LOB is temporary	Determining Whether a LOB instance Is Temporary on page 14-178
listempc.java	Seeing if LOB is temporary	Determining Whether a LOB instance Is Temporary on page 14-178
llength.java	Getting the length of a LOB	Length: Determining the Length of a LOB on page 14-86
lloaddat.java	Loading a LOB with BFILE data	Loading a LOB with Data from a BFILE on page 14-17
lread.java	Reading data from LOB	Reading Data from a LOB on page 14-52
lsubstr.java	Reading portion of LOB (substr)	Reading a Portion of a LOB (SUBSTR) on page 14-63
ltrim.java	Trimming LOB data	Trimming LOB Data on page 14-143
lwrite.java	Writing data to a LOB	Writing Data to a LOB on page 14-128
lwriteap.java	Writing to the end of LOB (write append)	Appending to a LOB on page 14-120
oldtrim.java	Old way of trimming LOB data	Trimming LOB Data on page 14-143

Glossary

BFILE

A Large Object datatype that is a binary file residing in the file system, outside of the database datafiles and tablespace. Note that the BFILE datatype is also referred to as an *external LOB* in some documentation.

Binary Large Object (BLOB)

A Large Object datatype that has content consisting of binary data and is typically used to hold unstructured data. The BLOB datatype is included in the category **Persistent LOBs** because it resides in the database.

BLOB

Pronounced "bee-lob." See Binary Large Object.

Character Large Object (CLOB)

The LOB datatype that has content consisting of character data in the database character set. A CLOB may be indexed and searched by the interMedia Text search engine.

CLOB

Pronounced "see-lob." See Character Large Object.

external LOB

A Large Object datatype that is stored outside of the database tablespace. The BFILE datatype is the only external LOB datatype. See also BFILE.

internal persistent LOB

A large object (LOB) that is stored in the database in a BLOB/CLOB/NCLOB column.

introspect

To examine attributes or value of an object.

Large Objects (LOBs)

Large Objects include the following SQL datatypes: BLOB, CLOB, NCLOB, and BFILE. These datatypes are designed for storing data that is large in size. See also BFILE, Binary Large Object, Character Large Object, and National Character Large Object.

LOB

See Large Objects.

LOB attribute

A large object datatype that is a field of an abstract datatype. For example a `CLOB` field of an `object` type.

LOB value

The actual data stored by the Large Object. For example, if a BLOB stores a picture, then the value of the BLOB is the data that makes up the image.

national character Large Object

The LOB datatype that has content consisting of character data in the database national character set. A CLOB may be indexed and searched by the interMedia Text search engine.

NCLOB

Pronounced "en-see-lob." See national character Large Object.

persistent LOB

A `BLOB`, `CLOB`, or `NCLOB` that is stored in the database. A persistent LOB instance can be selected out of a table and used within the scope of your application. The ACID properties of the instance are maintained just like any other column type. Persistent LOBs are sometimes also referred to as *internal persistent LOBs* or just, *internal LOBs*.

A persistent LOB can exist as a field of an abstract datatype as well as an instance in a LOB-type column. For example a `CLOB` attribute of an instance of type `object`.

See also *temporary LOB* and *external LOB*.

tablespace

A database storage unit that groups related logical structures together.

temporary LOB

A BLOB, CLOB, or NCLOB that is accessible and persists only within, the application scope in which it is declared. A temporary LOB does not exist in database tables.

Index

A

abstract datatypes and LOBs, 1-7
accessing a LOB
  using the LOB APIs, 2-7
accessing external LOBs, 15-4
amount, 15-56
amount parameter
  used with BFILEs, 15-14
appending
  writing to the end of a LOB
    internal persistent LOBs, 14-120
assigning OCILobLocator pointers, 6-13

B

BFILE class, See JDBC
BFILE-buffering, See JDBC
BFILEs
  accessing, 15-4
  converting to CLOB or NCLOB, 15-13
  creating an object in object cache, 5-29
  datatype, 1-7
  DBMS_LOB read-only procedures, 6-11
  DBMS_LOB, offset and amount parameters in bytes, 6-8
  hard links and symbolic links not allowed, 3-6
  locators, 2-4
  maximum number of open, 3-7, 15-97
  multithreaded server, 2-11
  multithreaded server mode, 15-11
  not affected by LOB storage properties, 4-8
  OCI functions to read/examine values, 6-15, 6-23
  OCI read-only functions, 6-16, 6-23
  opening and closing using JDBC, 6-47
  operating system files, and, 3-6
  Oracle Objects for OLE (OO4O)
    opening/closing methods, 6-36
    properties, 6-37
    read-only methods, 6-37
  Pro*C/C++ precompiler statements, 6-26
  Pro*COBOL precompiler embedded SQL statements, 6-30
  reading with DBMS_LOB, 6-10
  rules for using, 3-6
  security, 15-7
  storage devices, 1-5
  storing any operating system file, 1-7
  streaming APIs, 6-55
  using JDBC to read/examine, 6-42
  using Pro*C/C++ precompiler to open and close, 6-27
bind variables, used with LOB locators in OCI, 6-14
binds
  See also INSERT statements and UPDATE statements
Blob class, 6-19
BLOB-buffering, See JDBC
BLOBs
  class, See JDBC
  datatype, 1-7
  DBMS_LOB, offset and amount parameters in bytes, 6-8
  modify using DBMS_LOB, 6-9
  using JDBC to modify, 6-40
  using JDBC to read/examine BLOB values, 6-40

using oracle.sql.BLOB methods to modify, 6-40
buffering
disable
internal persistent LOBs, 14-171
enable
internal persistent LOBs, 14-162
flush
internal persistent LOBs, 14-167
LOB buffering subsystem, 5-5

C

C++, See Pro*C/C++ precompiler
C, See OCI
CACHE / NOCACHE, 4-12
caches
object cache, 5-29
callback, 14-52, 14-120, 15-56
catalog views
v$temporary_lobs, 3-6
character data
varying width, 4-5
character set ID
getting the
internal persistent LOBs, 14-15
See CSID parameter
charactersets
multibyte, LONG and LOB datatypes, 13-14
CHUNK, 4-14
chunksize, 14-128
multiple of, to improve performance, 14-52
CHUNKSIZE and LOB storage properties, 4-8
CLOB class, See JDBC
CLOB-buffering, See JDBC
CLOBs
columns
varying- width character data, 4-5
datatype, 1-7
varying-width columns, 4-5
DBMS_LOB, offset and amount parameters in characters, 6-8
modify using DBMS_LOB, 6-9
opening and closing using JDBC, 6-46
reading/examining with JDBC, 6-41
using JDBC to modify, 6-41
Clone method, See Oracle Objects for OLE (OO4O)
closing
all open BFILEs, 15-139
BFILEs with CLOSE, 15-131
BFILEs with FILECLOSE, 15-127
clustered tables, 11-10
COBOL, See Pro*COBOL precompiler
comparing
all or part of two LOBs
internal persistent LOBs, 14-71
all or parts of two BFILEs, 15-73
conventional path load, 3-3
conversion
explicit functions for PL/SQL, 10-3
conversion, implicit from CLOB to character type, 9-3
conversions
character set, 15-13
from binary data to character set, 15-13
implicit, between CLOB and VARCHAR2, 10-2
converting
to CLOB, 10-3
copy semantics, 1-6
internal LOBs, 8-6
copying
all or part of a LOB to another LOB
internal persistent LOBs, 14-93
LOB locator
internal persistent LOBs, 14-103
LOB locator for BFILE, 15-105
CSID parameter
setting OCILobRead and OCILobWrite to OCI_UCS2ID, 6-11

D

data interface for persistent LOBs, 13-1
multibyte charactersets, 13-14
DBMS_LOB
updating LOB with bind variable, 5-22
WRITE()
passing hexadecimal string to, 14-129
DBMS_LOB functions on a NULL LOB restriction, 4-2
DBMS_LOB package

available LOB procedures/functions, 6-3, 6-5
for temporary LOBs, 6-10
functions/procedures to modify BLOB, CLOB, and NCLOB, 6-9
functions/procedures to read/examine internal and external LOBs, 6-10
LOADFROMFILE(), 15-14
multithreaded server, 2-11
multithreaded server mode, 15-11
offset and amount parameter guidelines, 6-8
open and close, JDBC replacements for, 6-44
opening/closing internal and external LOBs, 6-11
provide LOB locator before invoking, 6-7
read-only functions/procedures for BFILEs, 6-11
to work with LOBs, using, 6-7
WRITE()
guidelines, 14-129
DBMS_LOB.GETLENGTH return value, 9-8
DBMS_LOB.isTemporary(), previous workaround for JDBC, 14-184
DBMS_LOB.READ, 15-56
directories
catalog views, 15-9
guidelines for usage, 15-10
ownership and privileges, 15-8
DIRECTORY name specification, 15-7
DIRECTORY object, 15-4
catalog views, 15-9
getting the alias and filename, 15-111
guidelines for usage, 15-10
names on Windows NT, 15-7
naming convention, 15-7
READ permission on object not individual files, 15-8
rules for using, 3-6
symbolic links, 3-6
symbolic links, and, 3-6
directory objects, 15-4
direct-path load, 3-3
displaying
LOB data for internal persistent LOBs, 14-42
domain indexing on LOB columns, 4-15

E

embedded SQL statements, See Pro*C/C++ precompiler and Pro*COBOL precompiler
empty LOBs
creating using JDBC, 6-60
JDBC, 6-60
EMPTY_BLOB() and EMPTY_CLOB, LOB storage properties for, 4-8
EMPTY_CLOB()/BLOB()
to initialize internal LOB
equal
one LOB locator to another
internal persistent LOBs, 14-111
erasing
part of LOB
internal persistent LOBs, 14-154
error message documentation, database, i-xxix
examples
repercussions of mixing SQL DML with DBMS_LOB, 5-17
updated LOB locators, 5-19
updating a LOB with a PL/SQL variable, 5-22
existence
check for BFILE, 15-89
extensible indexes, 4-16
external callout, 5-7
external LOBs (BFILEs)
See BFILEs
external LOBs (BFILEs), See BFILEs

F

FILECLOSEALL(), 15-10
flushing
LOB buffer, 5-6
flushing buffer, 5-2
FOR UPDATE clause
LOB locator, 5-14
function-based indexes on LOB columns, 4-16

H

hard links, rules with BFILEs, 3-6
hexadecimal string
passing to DBMS_LOB.WRITE(), 14-129

I

implicit assignment and parameter passing for LOB columns, 13-5
implicit conversion of CLOB to character type, 9-3
Improved LOB Usability, Accessing LOBs Using SQL Character Functions, 9-2
indexes
  function-based, 4-16
  rebuilding after LONG-to-LOB migration, 11-11
indexes on LOB columns
  bitmap index not supported, 4-16
  B-tree index not supported, 4-16
  domain indexing, 4-15
indexes, restrictions, 11-11
index-organized tables, restrictions for LOB columns, 4-22
initializing
  during CREATE TABLE or INSERT, 8-7
  internal LOB attributes cannot be initialized to a value, 4-3
  internal LOBs, See LOBs
    internal LOBs
  using EMPTY_CLOB(), EMPTY_BLOB()
initializing a LOB column to a non-null value, 4-3
in-line storage, 4-7
INSERT statements
  binds of greater than 4000 bytes, 13-8
inserting
  a row by initializing a LOB locator
    internal persistent LOBs, 8-8
  a row by initializing BFILE locator, 15-148
interfaces for LOBs, see programmatic environments
IS NULL return value for LOBs, 9-13
IS NULL usage with LOBs, 9-13

J

Java, See JDBC
JDBC
  available LOB methods/properties, 6-5
  BFILE class
  BFILE streaming APIs, 6-55
  BFILE-buffering, 6-43
  BLOB and CLOB classes
  calling DBMS_LOB package, 6-38
  changing internal LOBs with Java using objects oracle.sql.BLOB/CLOB, 6-37
  checking if BLOB is temporary, 14-183
  checking if CLOB is temporary, 14-184
  CLOB streaming APIs, 6-53
  empty LOBs, 6-60
  encapsulating locators
  methods/properties for BLOB-buffering, 6-41
  methods/properties for CLOB-buffering, 6-42
  modifying BLOB values, 6-40
  modifying CLOB values, 6-41
  newStreamLob.java, 6-55
  opening and closing BFILEs, 6-47
  opening and closing CLOBs, 6-46
  opening and closing LOBs, 6-44
  reading internal LOBs and external LOBs (BFILEs) with Java, 6-38
  reading/examining BLOB values, 6-40
  reading/examining CLOB values, 6-41
  reading/examining external LOB (BFILE) values, 6-42
  referencing LOBs, 6-38
  streaming APIs for LOBs, 6-52
  syntax references, 6-39
  trimming LOBs, 6-51
  using OracleResultSet to reference LOBs, 6-38
  using OUT parameter from OraclePreparedStatement to reference LOBs, 6-38
  writing to empty LOBs, 6-60
JDBC and Empty LOBs, 6-60

L

LBS, See Lob Buffering Subsystem (LBS)
length
  an internal persistent LOB, 14-86
  getting BFILE, 15-97
LENGTH return value for LOBs, 9-8
loading
  a LOB with BFILE data, 15-13
  LOB with data from a BFILE, 14-17
LOB, 5-27
LOB attributes
  defined, 1-7

LOB buffering
BLOB-buffering with JDBC, 6-41
buffer-enabled locators, 5-8
example, 5-5
flushing the buffer, 5-6
flushing the updated LOB through LBS, 5-7
guidelines, 5-2
OCI example, 5-9
OCI functions, 6-17
OCILobFlushBuffer(), 5-7
Oracle Objects for OLE (OO4O)
methods for internal LOBs, 6-36
physical structure of buffer, 5-4
Pro*C/C++ precompiler statements, 6-27
Pro*COBOL precompiler statements, 6-30
usage notes, 5-4
LOB Buffering SubSystem (LBS)
LOB Buffering Subsystem (LBS)
advantages, 5-2
buffer-enabled locators, 5-7
buffering example using OCI, 5-9
example, 5-5
flushing
updated LOB, 5-7
flushing the buffer, 5-6
guidelines, 5-2
saving the state of locator to avoid reselect, 5-8
usage, 5-4
LOB columns
initializing internal LOB to a value, 4-3
initializing to contain locator, 2-4
initializing to NULL or Empty, 4-2
LOB locator
copy semantics, 1-6
external LOBs (BFILEs), 1-6
internal LOBs, 1-6
out-bind variables in OCI, 6-14
reference semantics, 1-6
LOB locators, always stored in row, 4-8
LOB storage
format of varying width character data, 4-5
in-line and out-of-line storage properties, 4-7
LOBs, 5-29
abstract datatypes, members of, 1-7
attributes and abstract datatypes, 1-7
attributes and object cache, 5-29
buffering
caveats, 5-2
pages can be aged out, 5-7
buffering subsystem, 5-2
buffering usage notes, 5-4
datatypes versus LONG, 1-3
external (BFILEs), 1-5
flushing, 5-2
in partitioned tables, 4-18
in the object cache, 5-29
interfaces, See programmatic environments
interMEDIA, 1-8
internal
creating an object in object cache, 5-29
internal LOBs
CACHE / NOCACHE, 4-12
CHUNK, 4-14
ENABLE | DISABLE STORAGE IN ROW, 4-14
initializing, 15-56
introduced, 1-4
locators, 2-4
locking before updating, 14-4, 14-93, 14-121, 14-129, 14-143, 14-154
LOGGING / NOLOGGING, 4-13
Oracle Objects for OLE (OO4O), modifying methods, 6-35
PCTVERSION, 4-10
setting to empty, 4-3
tablespace and LOB index, 4-10
tablespace and storage characteristics, 4-8
transactions, 1-4
locators, 2-4, 5-14
object cache, 5-29
piecewise operations, 5-17
read consistent locators, 5-14
reason for using, 1-2
setting to contain a locator, 2-4
setting to NULL, 4-2
tables
creating indexes, 4-19
moving partitions, 4-19
splitting partitions, 4-20
unstructured data, 1-3

updated LOB locators, 5-16
varying-width character data, 4-6
locators, 2-4
BFILEs, 15-11
guidelines, 15-12
two rows can refer to the same file, 15-11
buffer-enabled, 5-8
external LOBs (BFILEs), 2-4
LOB, cannot span transactions, 5-27
multiple, 5-14
OCI functions, 6-16, 6-23
Pro*COBOL precompiler statements, 6-30
providing in Pro*COBOL precompiler, 6-27
read consistent, 5-6, 5-9, 5-14, 5-22, 5-27
read consistent locators, 5-14
read consistent, updating, 5-14
reading and writing to a LOB using, 5-25
saving the state to avoid reselect, 5-8
see if LOB locator is initialized
internal persistent LOBs, 14-117
selecting within a transaction, 5-26
selecting without current transaction, 5-25
setting column to contain, 2-4
transaction boundaries, 5-24
updated, 5-6, 5-16, 5-22
updating, 5-27
LOGGING
migrating LONG-to-LOBs, 11-3
LOGGING / NOLOGGING, 4-13
LONG versus LOB datatypes, 1-3
LONG-to-LOB Migration, 11-2
LONG-to-LOB migration
ALTER TABLE, 11-4
clustered tables, 11-10
LOGGING, 11-3
NULLs, 11-11
rebuilding indexes, 11-11
replication, 11-2
triggers, 11-11
lONG-to-LOB migration
PL/SQL, 13-4

M

migrating
LONG to LOBs, see LONG-to-LOB, 11-2
LONG-to-LOB using ALTER TABLE, 11-4
LONG-to-LOBs, constraints maintained, 11-5
LONG-to-LOBs, indexing, 11-11
multibyte character sets, using with the data interface for LOBs, 13-14
multithreaded server
BFILEs, 2-11, 15-11

N

national language support
NCLOBs, 1-7
NCLOB parameters allowed as attributes, ii-xl
NCLOBs
datatype, 1-7
DBMS_LOB, offset and amount parameters in characters, 6-8
modify using DBMS_LOB, 6-9
NewStreamLob.java, 6-55
NOCOPY, using to pass temporary LOB parameters by reference, 7-3
IS, 9-13
NULL LOB value, LOB storage for, 4-8
NULL LOB values, LOB storage properties for, 4-7
NULL LOB, restrictions calling OCI and DBMS_LOB functions, 4-2
NULL usage with LOBs, 9-13

O

object cache, 5-29
creating an object in, 5-29
LOBs, 5-29
OCCI
compared to other interfaces, 6-3
LOB functionality, 6-17
OCCI Bfile class, 6-23
OCCI Blob class
read, 6-20
write, 6-20
OCCI Clob class, 6-18
read, 6-20
write, 6-20
OCI

available LOB functions, 6-3
character set rules, fixed-width and varying-width, 6-12
functions for BFILEs, 6-16, 6-23
functions for temporary LOBs, 6-16, 6-23
functions to modify internal LOB values, 6-15, 6-22
functions to open/close internal and external LOBs, 6-17, 6-24
functions to read or examine internal and external LOB values, 6-15, 6-23
LOB buffering example, 5-9
LOB locator functions, 6-16, 6-23
Lob-buffering functions, 6-17
NCLOB parameters, 6-13, 6-21
OCILobFileGetLength
CLOB and NCLOB input and output length, 6-12
OCILobRead
varying-width CLOB and NCLOB input and amount amounts, 6-12
OCILobWrite
varying-width CLOB and NCLOB input and amount amounts, 6-13, 6-20
offset and amount parameter rules
fixed-width character sets, 6-19
setting OCILobRead, OCILobWrite to OCI_UCS2ID, 6-11
using to work LOBs, 6-11
OCI functions on a NULL LOB restriction, 4-2
OCILobAssign(), 5-3
OCILobFileSetName(), 15-7, 15-12
OCILobFlushBuffer(), 5-7
OCILobLoadFromFile(), 15-14
OCILobLocator in assignment "=" operations, 6-13
OCILobLocator, out-bind variables, 6-14
OCILobRead
BFILEs, 15-56
OCILobRead(), 14-42, 14-52, 15-56
OCILobWriteAppend(), 14-121
OCIObjectFlush(), 15-12
OCIObjectNew(), 15-12
OCISetAttr(), 15-12
offset parameter, in DBMS_LOB operations, 6-8
OLEDB, 6-60
OO4O, See Oracle Objects for OLE (OO4O)
open
checking for open BFILEs with FILEISOPEN(), 15-41
checking if BFILE is open with ISOPEN, 15-32
open, determining whether a LOB is open, 14-34
opening
BFILEs using FILEOPEN, 15-28
BFILEs with OPEN, 15-21
opening and closing LOBs
using JDBC, 6-44
ORA-17098
empty LOBs and JDBC, 6-60
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, 6-5
internal LOB buffering, 6-36
methods and properties to access data stored in BLOBs, CLOBs, NCLOBs, and BFILEs, 6-33
modifying internal LOBs, 6-35
opening/closing external LOBs (BFILEs), 6-36
OraBfile example
OraBlob example
OraBlob, OraClob, and OraBfile encapsulate locators, 6-32
properties for operating on external LOBs (BFILEs), 6-37
properties for operating on LOBs, 6-36
reading/examining internal LOB and external LOB (BFile) values, 6-35
read-only methods for external LOBs (BFILEs), 6-37
syntax reference, 6-31
using Clone method to retain locator independent of dynaset, 6-32
OraclePreparedStatement, See JDBC
OracleResultSet, See JDBC
oracle.sql.BFILE
BFILE-buffering, 6-43
JDBC methods to read/examine BFILEs, 6-42
oracle.sql.BLOB
for modifying BLOB values, 6-40
reading/examining BLOB values, 6-40

See JDBC
oracle.sql.BLOBs
BLOB-buffering
oracle.sql.CLOB
CLOB-buffering
JDBC methods to read/examine CLOB values, 6-41
modifying CLOB values, 6-41
oracle.sql.CLOBs
See JDBC
OraOLEDB, 6-60
out-of-line storage, 4-7

P

partitioned index-organized tables
restrictions for LOB columns, 4-22
pattern
check if it exists in BFILE using instr, 15-82
see if it exists IN LOB using (instr)
internal persistent LOBs, 14-79
PCTINCREASE parameter, recommended value for LOBs, 5-33
PCTVERSION, 4-10
performance
guidelines
reading/writing large data chunks, 7-2
reading/writing large data chunks, temporary LOBs, 7-4
PL/SQL, 6-2
and LOBs, semantics changes, 10-2
changing locator-data linkage, 10-5
CLOB variables in, 10-4
CLOB variables in PL/SQL, 10-4
CLOB versus VARCHAR2 comparison, 10-6
CLOBs passed in like VARCHAR2s, 10-3
defining a CLOB Variable on a VARCHAR, 10-3
freeing temporary LOBs automatically and manually, 10-5
using in LONG-to-LOB migration, 13-4
polling, 14-52, 14-120, 15-56
Pro*C/C++ precompiler
available LOB functions, 6-3
LOB buffering, 6-27
locators, 6-26
modifying internal LOB values, 6-25
opening and closing internal LOBs and external LOBs (BFILEs), 6-27
providing an allocated input locator pointer, 6-24
reading or examining internal and external LOB values, 6-26
statements for BFILEs, 6-26
statements for temporary LOBs, 6-26
Pro*COBOL precompiler
available LOB functions, 6-3
LOB buffering, 6-30
locators, 6-30
modifying internal LOB values, 6-29
providing an allocated input locator, 6-27
reading or examining internal and external LOBs, 6-29
statements for BFILEs, 6-30
temporary LOBs, 6-29
programmatic environments
available functions, 6-3
compared, 6-3
programmatic environments for LOBs, 6-2

R

read consistency
LOBs, 5-14
read consistent locators, 5-6, 5-9, 5-14, 5-22, 5-27
reading
data from a LOB
internal persistent LOBs, 14-52
large data chunks, performance guidelines, 7-2
large data chunks, temporary LOBs, 7-4
portion of BFILE data using substr, 15-66
portion of LOB using substr
internal persistent LOBs, 14-63
small amounts of data, enable buffering, 14-162
reference semantics, 8-6
BFILEs enables multiple BFILE columns for each record, 15-6
replication, 11-2
restrictions
binds of more than 4000 bytes, 13-8

cannot call OCI or DBMS_LOB functions on a NULL LOB, 4-2
clustered tables, 11-10
indexes, 11-11
index-organized tables and LOBs, 4-22
on LOBs, 2-8
removed, ii-xxxix
replication, 11-2
triggers, 11-11
RETURNING clause, using with INSERT to initialize a LOB, 4-3
round-trips to the server, avoiding, 5-2, 5-9
rules for using directory objects and BFILEs, 3-6

S

security
BFILEs, 15-7
BFILEs using SQL DDL, 15-9
BFILEs using SQL DML, 15-9
SELECT statement
read consistency, 5-14
semantics
copy-based for internal LOBs, 8-6
reference based for BFILEs, 15-6
SESSION_MAX_OPEN_FILES parameter, 3-7
setting
internal LOBs to empty, 4-3
LOBs to NULL, 4-2
overrides for NLS_LANG variable
SQL
Character Functions, improved, 9-2
features where LOBs cannot be used, 9-11
SQL DDL
BFILE security, 15-9
SQL DML
BFILE security, 15-9
SQL functions on LOBs
return type, 9-8
return value, 9-8
temporary LOBs returned, 9-8
SQL semantics and LOBs, 9-11
SQL semantics supported for use with LOBs, 9-2
SQL*Loader
conventional path load, 3-3
direct-path load, 3-3
streaming, 14-42
do not enable buffering, when using, 14-162
write, 14-128
streaming APIs
NewStreamLob.java, 6-55
using JDBC and BFILEs, 6-55
using JDBC and CLOBs, 6-53
using JDBC and LOBs, 6-52
symbolic links, rules with DIRECTORY objects and BFILEs, 3-6
system owned object, See DIRECTORY object

T

temporary BLOB
checking if temporary using JDBC, 14-183
temporary CLOB
checking if temporary using JDBC, 14-184
temporary LOBs
checking if LOB is temporary, 14-178
DBMS_LOB available functions/procedures, 6-10
OCI functions, 6-16, 6-23
Pro*C/C++ precompiler embedded SQL statements, 6-26
Pro*COBOL precompiler statements, 6-29
returned from SQL functions, 9-8
TO_BLOB(),TO_CHAR(), TO_NCHAR(), 10-3
TO_CLOB()
converting VARCHAR2,NVARCHAR2,NCLOB to CLOB, 10-3
TO_NCLOB(), 10-3
transaction boundaries
LOB locators, 5-24
transaction IDs, 5-25
transactions
external LOBs do not participate in, 1-6
IDs of locators, 5-24
internal LOBs participate in database transactions, 1-4
LOB locators cannot span, 5-27
locators with non-serializable, 5-25
locators with serializable, 5-25
migrating from, 5-7

triggers
LONG-to-LOB migration, 11-11
trimming
LOB data
internal persistent LOBs, 14-143
trimming LOBs using JDBC, 6-51

U

UCS2 Unicode character set
varying width character data, 4-5
UNICODE
VARCHAR2 and CLOBs support, 9-7
unstructured data, 1-3
UPDATE statements
binds of greater than 4000 bytes, 13-8
updated locators, 5-6, 5-16, 5-22
updating
avoid the LOB with different locators, 5-19
LOB values using one locator, 5-19
LOB values, read consistent locators, 5-14
LOB with PL/SQL bind variable, 5-22
LOBs using SQL and DBMS_LOB, 5-17
locators, 5-27
locking before, 14-93
locking prior to, 14-4, 14-143, 14-154

V

VARCHAR2
accessing CLOB data when treated as, 10-2
also RAW, applied to CLOBs and BLOBs, 9-11
defining CLOB variable on, 10-3
VARCHAR2, using SQL functions and operators with LOBs, 9-2
VARRAY
LOB restriction, 2-9
varying-width character data, 4-5
views on DIRECTORY object, 15-9
Visual Basic, See Oracle Objects for OLE(OO4O)

W

WHERE Clause Usage with LOBs, 9-13
writing
data to a LOB
internal persistent LOBs, 14-128
large data chunks, performance guidelines, 7-2
large data chunks, temporary LOBs, 7-4
singly or piecewise, 14-120
small amounts of data, enable buffering, 14-162