

Oracle8i

アプリケーション開発者ガイド ラージ・オブジェクト

リリース 8.1

2000年2月

ORACLE®

Oracle8i アプリケーション開発者ガイド ラージ・オブジェクト, リリース 8.1

原本名 : Oracle8i Application Developer's Guide - Large Objects (LOBs), Release 2 (8.1.6)

原本著者 : Shelley Higgins, Susan Kotsovolos, Den Raphaely

原本協力者 : Geeta Arora, Sandeepan Banerjee, Thomas Chang, Chandrasekharan Iyer, Ramkumar Krishnan, Dan Mullen, Visar Nimani, Anindo Roy, Rosanne Toohey, Guhan Viswana, Jeya Balaji, Maria Chien, Christian Shay, Ali Shehade, Sundaram Vedala, Eric Wan, Joyce Yang

グラフィック・デザイナー : Valerie Moore, Charles Keller

Copyright © 1996, 1999, Oracle Corporation. All rights reserved.

Printed in Japan.

制限付権利の説明

プログラム (ソフトウェアおよびドキュメントを含む) の使用、複製または開示は、オラクル社との契約に記載された制約条件に従うものとします。著作権、特許権およびその他の知的財産権に関する法律により保護されています。

当プログラムのリバース・エンジニアリング等は禁止されています。

このドキュメントの情報は、予告なしに変更されることがあります。オラクル社は本ドキュメントの無謬性を保証しません。

* オラクル社とは、Oracle Corporation (米国オラクル) または日本オラクル株式会社 (日本オラクル) を指します。

危険な用途への使用について

オラクル社製品は、原子力、航空産業、大量輸送、医療あるいはその他の危険が伴うアプリケーションを用途として開発されておりません。オラクル社製品を上述のようなアプリケーションに使用することについての安全確保は、顧客各位の責任と費用により行ってください。万一かかる用途での使用によりクレームや損害が発生いたしましても、日本オラクル株式会社と開発元である Oracle Corporation (米国オラクル) およびその関連会社は一切責任を負いかねます。当プログラムを米国防総省の米国政府機関に提供する際には、『Restricted Rights』と共に提供してください。この場合次の Notice が適用されます。

Restricted Rights Notice

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

このドキュメントに記載されているその他の会社名および製品名は、あくまでその製品および会社を識別する目的にのみ使用されており、それぞれの所有者の商標または登録商標です。

目次

はじめに	xxxvii
このマニュアルについて	xxxviii
機能範囲および可用性	xxxviii
LOB の新機能	xxxviii
このマニュアルの新規事項	xxxix
関連マニュアル	xl
このマニュアルの構成	xlii
このマニュアルの表記規則	xliv
図の解釈方法	xlv
ユースケース図の要素	xlviii
1 概要	
LOB を使用する理由	1-2
非構造化データ	1-2
LOB データ型によるインターネット・アプリケーションのサポート	1-2
LONG を使用しない理由	1-3
LOB によるセマンティクスの制御	1-4
LOB による <i>interMedia</i> の使用	1-4
LOB のデモ・ディレクトリ	1-5
互換性および移行の問題	1-5
このマニュアルでの <i>Multimedia_Tab</i> を使用した例	1-6
参照	1-6
2 基本コンポーネント	
LOB データ型	2-2

内部 LOB	2-2
外部 LOB (BFILE)	2-2
内部 LOB で使用するコピー・セマンティクスおよび外部 LOB で使用する参照セマンティクス ...	2-3
可変幅文字データ	2-4
LOB ロケータ	2-5
LOB の値およびロケータ	2-5
LOB ロケータの操作	2-6
LOB を含む表の作成	2-8
内部 LOB を NULL または空として初期化	2-8
内部 LOB 列を値に初期化	2-10
外部 LOB を NULL またはファイル名に初期化	2-10

3 LOB プログラム環境

LOB で操作される 6 つのプログラム環境	3-2
6 つの LOB インタフェースの比較	3-3
LOB の作業に対する PL/SQL (DBMS_LOB パッケージ) の使用	3-6
DBMS_LOB ルーチン起動前の LOB ロケータの提供	3-6
DBMS_LOB ルーチンをコールできないクライアント PL/SQL プロシージャ	3-6
オフセット・パラメータおよび量パラメータ: 固定幅と可変幅、DBMS_LOB パッケージに対する 文字またはバイト	3-7
DBMS_LOB.LOADFROMFILE: BFILE より小さいサイズの量パラメータの指定	3-7
DBMS_LOB.READ: データ・サイズより大きい量パラメータ	3-7
PL/SQL: BLOB、CLOB、NCLOB および BFILE を操作するファンクションおよび プロシージャ	3-7
PL/SQL: BLOB、CLOB および NCLOB の値を変更するファンクション / プロシージャ	3-8
PL/SQL: 内部および外部 LOB 値の読み込みまたはテストを行うファンクション / プロシージャ ...	3-8
PL/SQL: テンポラリ LOB を操作するファンクション / プロシージャ	3-9
PL/SQL: BFILE 固有の読み込み専用ファンクション / プロシージャ	3-9
PL/SQL: 内部および外部 LOB をオープンおよびクローズするファンクション / プロシージャ ...	3-9
C (OCI) を使用した LOB の作業	3-11
UCS2 における読み込み / 書き込みのための CSID パラメータの OCI_UCS2ID への設定	3-11
オフセット・パラメータおよび量パラメータ: 固定幅と可変幅、OCI での文字またはバイト	3-11
OCILobLoadFromFile: 量パラメータに BFILE より短い値を指定する	3-13
OCILobRead: 量パラメータに 4GB -1 の値を指定する	3-13
OCI LOB の例	3-13

OCI の詳細	3-14
OCI: BLOB、CLOB、NCLOB および BFILE を操作する関数	3-14
OCI: 内部 LOB (BLOB、CLOB および NCLOB) の値を変更する関数	3-14
OCI: 内部 LOB および外部 LOB (BFILE) 値の読み込みまたはテストを行う関数	3-15
OCI: テンポラリ LOB を操作する関数	3-15
OCI: BFILE 固有の読み込み専用関数	3-15
OCI: LOB ロケータ関数	3-16
OCI: LOB バッファリング関数	3-16
OCI: 内部および外部 LOB をオープンおよびクローズする関数	3-16
プロシージャ例 : main() および seeIfLOBIsOpen	3-17
C/C++ (Pro*C) を使用した LOB の作業	3-21
LOB を表す入力ロケータ・ポインタの割当て	3-21
Pro*C/C++: BLOB、CLOB、NCLOB および BFILE を操作する文	3-21
Pro*C/C++: 内部 LOB (BLOB、CLOB および NCLOB) の値を変更する埋込み SQL 文	3-22
Pro*C/C++: 内部および外部 LOB 値の読み込みまたはテストを行う埋込み SQL 文	3-22
Pro*C/C++: テンポラリ LOB を操作する埋込み SQL 文	3-23
Pro*C/C++: BFILE 固有の埋込み SQL 文	3-23
Pro*C/C++: LOB ロケータ埋込み SQL 文	3-23
Pro*C/C++: LOB バッファリング埋込み SQL 文	3-24
Pro*C/C++: 内部 LOB および外部 LOB (BFILE) をオープンおよびクローズするための 埋込み SQL 文	3-24
COBOL (Pro*COBOL) を使用した LOB の作業	3-25
LOB を表す入力ロケータ・ポインタの割当て	3-25
Pro*COBOL: BLOB、CLOB、NCLOB および BFILE を操作する文	3-26
Pro*COBOL: 内部 LOB (BLOB、CLOB および NCLOB) の値を変更する埋込み SQL 文	3-26
Pro*COBOL: 内部および外部 LOB 値の読み込みまたはテストを行う埋込み SQL 文	3-27
Pro*COBOL: テンポラリ LOB を操作する埋込み SQL 文	3-27
Pro*COBOL: BFILE 固有の埋込み SQL 文	3-27
Pro*COBOL: LOB ロケータ埋込み SQL 文	3-28
Pro*COBOL: LOB バッファリング埋込み SQL 文	3-28
Pro*COBOL: 内部 LOB および外部 LOB (BFILE) をオープンおよびクローズするための 埋込み SQL 文	3-28
Visual Basic (Oracle Objects for OLE (OO4O)) を使用した LOB の作業	3-29
OO4O 構文の参照および詳細	3-29
OraBlob、OraClob および OraBfile オブジェクト・インタフェースによるロケータの カプセル化	3-30

OraBlob および OraBfile の例	3-30
OO4O: BLOB、CLOB、NCLOB および BFILE に格納されたデータにアクセスするメソッド およびプロパティ	3-31
OO4O: 内部 LOB (BLOB、CLOB および NCLOB) の値を変更するメソッド	3-32
OO4O: 内部および外部 LOB 値の読み込みまたはテストを行うメソッド	3-33
OO4O: 外部 LOB (BFILE) をオープンおよびクローズするメソッド	3-33
OO4O: 内部 LOB バッファリング・メソッド	3-34
OO4O: LOB プロパティ	3-34
OO4O: 外部 LOB (BFILE) 固有の読み込み専用メソッド	3-35
OO4O: 外部 LOB (BFILE) での操作のためのプロパティ	3-35
Java (JDBC) を使用した LOB の作業	3-36
Java での内部 LOB の変更	3-36
Java を使用した内部 LOB および外部 LOB (BFILE) の読み込み	3-36
DBMS_LOB パッケージのコール	3-36
LOB の参照	3-37
JDBC 構文の参照および詳細	3-37
LOB での操作のための JDBC メソッド	3-38
JDBC: BLOB 値を変更する oracle.sql.BLOB メソッド	3-38
JDBC: BLOB 値の読み込みまたはテストを行う oracle.sql.BLOB メソッド	3-38
JDBC: BLOB バッファリングのための oracle.sql.BLOB メソッドおよびプロパティ	3-39
JDBC: CLOB 値を変更する oracle.sql.CLOB メソッド	3-39
JDBC: CLOB 値を読み込みまたはテストを行う oracle.sql.CLOB メソッド	3-40
JDBC: CLOB バッファリングのための oracle.sql.CLOB メソッドおよびプロパティ	3-40
JDBC: 外部 LOB (BFILE) 値の読み込みまたはテストを行う oracle.sql.BFILE メソッド	3-41
JDBC: BFILE バッファリングのための oracle.sql.BFILE メソッドおよびプロパティ	3-42
JDBC: 8.1.x および今後のリリースで使用できない OracleBlob および OracleClob	3-42

4 LOB の管理

LOB を使用する前に必要な DBA アクション	4-2
オープンできる BFILE 数の上限の設定	4-2
LOB に対する基本操作での SQL DML の使用	4-2
LOB 表領域の記憶域の変更	4-3
テンポラリ LOB の管理	4-4
LOB ロード時の SQL*Loader の使用	4-5
LOBFIELDS	4-5

SQL*Loader を使用した行内および行外データの内部 LOB へのロード	4-6
SQL*Loader パフォーマンス : 内部 LOB へのロード	4-6
行内 LOB データのロード	4-7
既定サイズ・フィールド内の行内 LOB データのロード	4-7
デリミタ付きフィールド内の行内 LOB データのロード	4-8
Length-Value Pair フィールド内の行内 LOB データのロード	4-8
行外 LOB データのロード	4-10
1 ファイルにつき 1 つの LOB のロード	4-10
既定サイズ・フィールド内の行外 LOB データのロード	4-11
デリミタ付きフィールド内の行外 LOB データのロード	4-12
Length-Value Pair フィールド内の行外 LOB データのロード	4-13
ヒントをロードする SQL*Loader LOB	4-14
LOB の制限	4-15
削除された制限	4-17

5 高度なトピック

読取り一貫性のあるロケータ	5-2
読取り一貫性のあるロケータになる、SELECT されたロケータ	5-2
LOB の更新および読取り一貫性	5-3
読取り一貫性のあるロケータを使用した更新の例	5-3
更新済ロケータを介した更新済 LOB	5-6
SQL DML および DBMS_LOB を使用した LOB の更新例	5-6
同じ LOB 値を更新するために 1 つのロケータを使用する例	5-8
PL/SQL (DBMS_LOB) バインド変数を使用した LOB の更新の例	5-10
複数のトランザクションにまたがることはできない LOB ロケータ	5-13
トランザクションにまたがらないロケータの例	5-13
LOB ロケータとトランザクション境界	5-15
ロケータがトランザクション ID を含む場合	5-15
ロケータがトランザクション ID を含まない場合	5-15
トランザクション ID: ロケータを使用した LOB に対する読込みおよび書込み	5-15
シリアライズ可能でない例: 現在のトランザクションを持たないロケータを選択する	5-16
シリアライズ可能でない例: トランザクション内でロケータを選択する	5-17
オブジェクト・キャッシュ内の LOB	5-18
LOB バッファリング・サブシステム	5-19
LOB バッファリングのメリット	5-19

LOB バッファリングの使用上のガイドライン	5-19
LOB バッファリングの使用についての注意	5-21
LOB バッファのフラッシュ	5-23
更新済 LOB のフラッシュ	5-24
バッファリング対応のロケータ	5-25
再選択を避けるためのロケータ状態の保存	5-25
OCI: LOB バッファリングの例	5-26
LOB への参照を含む VARRAY の作成	5-28

6 FAQ

LOB データ型へのデータ型の変換	6-3
どのような長さのデータでも LOB 列に挿入または更新できますか？	6-3
データが 64KB を超える場合、LONG を LOB にコピーできますか？	6-3
一般的な質問	6-4
トリガーのある LOB 列が更新されているかどうかを判断する方法は？	6-4
LOB データの読み込みおよびロード: 量パラメータのサイズは？	6-4
索引構成表 (IOT) および LOB	6-6
行内記憶域は索引構成表の LOB に使用できますか？	6-6
LOB ロケータの初期化	6-7
EMPTY_BLOB() および EMPTY_CLOB() はいつ使用するのですか？	6-7
Java で EMPTY_BLOB() を使用して BLOB 属性を初期化する方法は？	6-8
JDBC、JPublisher および LOB	6-8
JDBC を使用して空の LOB ロケータを持つ行を表に挿入する方法は？	6-8
JPublisher を使用して EMPTY_BLOB() に setData を実行する方法は？	6-9
JDBC: OracleBlob および OracleClob は Oracle8i で使用できますか？	6-10
8.1 JDBC Thin Driver で LOB を操作する方法は？	6-10
LOB へ書き込む場合、SELECT に FOR UPDATE 句は必要ですか？	6-11
LOB およびデータの LOB へのロード	6-12
1MB のファイルを CLOB 列にロードする方法は？	6-12
JDBC Driver を使用してロードする場合に BLOB および CLOB のパフォーマンスを向上させる 方法は？	6-12
LOB 索引付け	6-16
LOB 索引は LOB データと同じ表領域内に作成されますか？	6-16
索引付け: DELETE で BLOB 列は削除されますが、BFILE 列は削除されないのはなぜですか？ ..	6-16
LOB 索引について問合せができるのはどのビューですか？	6-16

LOB 記憶域および領域の問題	6-18
LOB 表領域と ENABLE STORAGE IN ROW を指定するとどうなりますか？	6-18
BFILE と BLOB にイメージを格納する場合のメリットおよびデメリットは何ですか？	6-18
DISABLE STORAGE IN ROW はいつ指定する必要がありますか？	6-19
4KB 未満の BLOB は表データと同じセグメントに移動されますか？ 4KB を超える BLOB は 指定されたセグメントに移動されますか？	6-19
4KB の BLOB は行内に格納されますか？	6-20
LOB 列が NULL ではなく EMPTY_CLOB() または EMPTY_BLOB() の場合に、LOB ロケータは どのように格納されますか？ この場合、追加のデータ・ブロックが使用されますか？	6-21
他のデータベース・システムからの移行	6-22
Oracle8i では、異なる LOB 型の間での暗黙的な LOB 変換はできますか？	6-22
パフォーマンス	6-23
ディスク・アレイ、UNIX および Oracle において Veritas File System を使用した場合に LOB ロードのパフォーマンスを向上させる方法は？	6-23
DBMS_LOB.SUBSTR を使用した場合と DBMS_LOB.READ を使用した場合とでは、 パフォーマンスに違いがありますか？	6-24
LOB パフォーマンスのチューニングに関するホワイト・ペーパーまたはガイドラインが ありますか？	6-24
全体の読み込みにチャンクを使用する必要があるのはいつですか？	6-25
LOB を行内に格納してもよいでしょうか？ またその場合の時期はいつですか？	6-25
4GB を超える LOB をデータベースに格納する方法は？	6-26

7 モデリングおよび設計

データ型の選択	7-2
LOB 型と LONG 型および LONG RAW 型との比較	7-2
キャラクタ・セットの変換：可変幅文字データの処理	7-3
表アーキテクチャの選択	7-4
LOB 記憶域	7-5
LOB 列内の NULL 値の格納場所	7-5
内部 LOB に対する表領域および記憶特性の定義	7-5
LOB 列または属性の LOB 記憶特性	7-6
TABLESPACE および LOB 索引	7-7
PCTVERSION	7-7
CACHE / NOCACHE / CACHE READS	7-8
LOGGING / NOLOGGING	7-9
CHUNK	7-10

ENABLE DISABLE STORAGE IN ROW	7-11
GB の LOB の作成方法	7-12
例:GB の LOB を格納するための表領域および表の作成	7-12
LOB ロケータおよびトランザクション境界	7-13
INSERT および UPDATE での 4,000 バイトを超えるバインド	7-14
LOB の INSERT および UPDATE で現在可能な 4,000 バイトを超えるバインド	7-14
4,000 バイトを超えるバインド (HEX から RAW または RAW から HEX への変換なし)	7-14
SQL 演算子の結果に対する 4,000 バイトの制限	7-15
4,000 バイトを超えるバインド: 制限事項	7-16
例: PL/SQL - INSERT および UPDATE での 4,000 バイトを超えるバインドの使用	7-16
例: PL/SQL - 4,000 バイトを超えるバインド (HEX から RAW / RAW から HEX への変換が サポートされていないため挿入は未サポート)	7-17
例: PL/SQL - データに SQL 演算子が含まれる場合に 4,000 バイトを超えるバインドの結果を 4,000 バイトに制限	7-18
内部 LOB 用の Open、Close および IsOpen インタフェース	7-19
索引構成表 (IOT) 内の LOB	7-22
LOB 列を含む索引構成表 (IOT) の例	7-22
パーティション表内の LOB の操作	7-24
LOB データを含む表の作成およびパーティション化	7-26
LOB 列を含む表の索引の作成	7-28
LOB データを含むパーティションの交換	7-28
LOB データを含む表へのパーティションの追加	7-29
LOB を含むパーティションの移動	7-29
LOB を含むパーティションの分割	7-29
LOB 列の索引付け	7-30
パフォーマンスの最適化のための最善策	7-31
SQL*Loader の使用	7-31
パフォーマンスの最適化のためのガイドライン	7-31
スレッド環境における LOB へのデータの移動	7-32

8 サンプル・アプリケーション

サンプル・アプリケーション	8-2
マルチメディア・コンテンツ収集システム	8-2
アプリケーションへのオブジェクト・リレーショナル設計の適用	8-4
Multimedia_tab 表の構造	8-5

9 内部永続 LOB

ユーสケース・モデル:内部永続 LOB	9-2
LOB を含む表を作成する 3 つの方法	9-6
使用上の注意	9-7
1 つ以上の LOB 列を含む表の作成	9-8
用途	9-8
使用上の注意	9-8
構文	9-9
使用例	9-9
例	9-10
SQL: 1 つ以上の LOB 列を含む表の作成	9-10
LOB 属性を持つオブジェクト型を含む表の作成	9-13
用途	9-13
使用上の注意	9-13
構文	9-14
使用例	9-14
例	9-15
SQL: LOB 属性を持つオブジェクト型を含む表の作成	9-15
LOB を含む NESTED TABLE の作成	9-18
用途	9-18
使用上の注意	9-18
構文	9-18
使用例	9-19
例	9-20
SQL: LOB を含む NESTED TABLE の作成	9-20
1 つ以上の LOB 値を行に挿入する 3 つの方法	9-21
使用上の注意	9-22
EMPTY_CLOB() または EMPTY_BLOB() を使用した LOB 値の挿入	9-23
用途	9-24
使用上の注意	9-24
構文	9-24
使用例	9-25
例	9-25
SQL: EMPTY_CLOB()/EMPTY_BLOB() を使用した値の挿入	9-25
Java (JDBC) : EMPTY_CLOB()/EMPTY_BLOB() を使用した値の挿入	9-26

別の表からの LOB の選択による行の挿入	9-27
用途	9-27
使用上の注意	9-27
構文	9-28
使用例	9-28
例	9-28
SQL: 別の表からの LOB の選択による行の挿入	9-28
初期化した LOB ロケータ・バインド変数を使用した行の挿入	9-29
用途	9-29
使用上の注意	9-29
構文	9-30
使用例	9-30
例	9-30
PL/SQL (DBMS_LOB パッケージ) : 初期化した LOB ロケータ・バインド変数を使用した 行の挿入	9-31
C (OCI) : 初期化した LOB ロケータ・バインド変数を使用した行の挿入	9-31
COBOL (Pro*COBOL) : 初期化した LOB ロケータ・バインド変数を使用した行の挿入	9-33
C/C++ (Pro*C) : 初期化した LOB ロケータ・バインド変数を使用した行の挿入	9-34
Visual Basic (OO4O) : 初期化した LOB ロケータ・バインド変数を使用した行の挿入	9-35
Java (JDBC) : 初期化した LOB ロケータ・バインド変数を使用した行の挿入	9-36
内部 LOB (BLOB、CLOB、NCLOB) へのデータのロード	9-38
用途	9-38
使用上の注意および例	9-39
構文	9-39
使用例	9-39
LOB への BFILE データのロード	9-40
用途	9-41
使用上の注意	9-41
構文	9-41
使用例	9-42
例	9-42
PL/SQL (DBMS_LOB パッケージ) : LOB への BFILE データのロード	9-42
C (OCI) : LOB への BFILE データのロード	9-43
COBOL (Pro*COBOL) : LOB への BFILE データのロード	9-45
C/C++ (Pro*C) : LOB への BFILE データのロード	9-46

Visual Basic (OO4O) :LOB への BFILE データのロード	9-47
Java (JDBC) :LOB への BFILE データのロード	9-48
LOB がオープンしているかどうかの確認	9-51
用途	9-51
使用上の注意	9-51
構文	9-51
使用例	9-52
例	9-52
PL/SQL (DBMS_LOB パッケージ) :LOB がオープンしているかどうかの確認	9-52
C (OCI) :LOB がオープンしているかどうかの確認	9-53
COBOL (Pro*COBOL) :LOB がオープンしているかどうかの確認	9-54
C/C++ (Pro*C) :LOB がオープンしているかどうかの確認	9-55
Java (JDBC) :LOB がオープンしているかどうかの確認	9-56
LOB への LONG のコピー	9-59
用途	9-59
使用上の注意	9-60
構文	9-60
使用例	9-60
例	9-61
SQL: LONG の LOB へのコピー	9-61
LOB のチェックアウト	9-64
用途	9-64
使用上の注意	9-65
構文	9-65
使用例	9-65
例	9-66
PL/SQL (DBMS_LOB パッケージ) :LOB のチェックアウト	9-66
C (OCI) :LOB のチェックアウト	9-67
COBOL (Pro*COBOL) :LOB のチェックアウト	9-69
C/C++ (Pro*C) :LOB のチェックアウト	9-71
Visual Basic (OO4O) :LOB のチェックアウト	9-72
Java (JDBC) :LOB のチェックアウト	9-73
LOB のチェックイン	9-75
用途	9-75
使用上の注意	9-76

構文	9-76
使用例	9-76
例	9-77
PL/SQL (DBMS_LOB パッケージ) : LOB のチェックイン	9-77
C (OCI) : LOB のチェックイン	9-78
COBOL (Pro*COBOL) : LOB のチェックイン	9-81
C/C++ (Pro*C) : LOB のチェックイン	9-83
Visual Basic (OO4O) : LOB のチェックイン	9-86
Java (JDBC) : LOB のチェックイン	9-87
LOB データの表示	9-89
用途	9-90
使用上の注意	9-90
構文	9-90
使用例	9-90
例	9-91
PL/SQL (DBMS_LOB パッケージ) : LOB データの表示	9-91
C (OCI) : LOB データの表示	9-92
COBOL (Pro*COBOL) : LOB データの表示	9-94
C/C++ (Pro*C) : LOB データの表示	9-95
Visual Basic (OO4O) : LOB データの表示	9-97
Java (JDBC) : LOB データの表示	9-97
LOB からのデータの読み込み	9-99
用途	9-100
使用上の注意	9-100
構文	9-101
使用例	9-102
例	9-102
PL/SQL (DBMS_LOB パッケージ) : LOB からのデータの読み込み	9-102
C (OCI) : LOB からのデータの読み込み	9-103
COBOL (Pro*COBOL) : LOB からのデータの読み込み	9-105
C/C++ (Pro*C/C++) : LOB からのデータの読み込み	9-107
Visual Basic (OO4O) : LOB からのデータの読み込み	9-108
Java (JDBC) : LOB からのデータの読み込み	9-108
LOB の一部の読み込み (substr)	9-110
用途	9-111

使用上の注意	9-111
構文	9-111
使用例	9-112
例	9-112
PL/SQL (DBMS_LOB パッケージ) : LOB の一部の読込み (substr)	9-112
COBOL (Pro*COBOL) : LOB の一部の読込み (substr)	9-113
C/C++ (Pro*C/C++) : LOB の一部の読込み (substr)	9-114
Visual Basic (OO4O) : LOB の一部の読込み (substr)	9-115
Java (JDBC) : LOB の一部の読込み (substr)	9-116
2つのLOBの全体または一部の比較	9-118
用途	9-118
使用上の注意	9-119
構文	9-119
使用例	9-119
例	9-120
PL/SQL (DBMS_LOB パッケージ) : 2つのLOBの全体または一部の比較	9-120
COBOL (Pro*COBOL) : 2つのLOBの全体または一部の比較	9-121
C/C++ (Pro*C/C++) : 2つのLOBの全体または一部の比較	9-122
Visual Basic (OO4O) : 2つのLOBの全体または一部の比較	9-123
Java (JDBC) : 2つのLOBの全体または一部の比較	9-124
LOB内のパターンの有無の確認 (instr)	9-126
用途	9-127
使用上の注意	9-127
構文	9-127
使用例	9-127
例	9-128
PL/SQL (DBMS_LOB パッケージ) : LOB内のパターンの有無の確認 (instr)	9-128
COBOL (Pro*COBOL) : LOB内のパターンの有無の確認 (instr)	9-129
C/C++ (Pro*C/C++) : LOB内のパターンの有無の確認 (instr)	9-130
Java (JDBC) : LOB内のパターンの有無の確認 (instr)	9-131
LOBの長さの取得	9-133
用途	9-133
使用上の注意	9-134
構文	9-134
使用例	9-134

例	9-134
PL/SQL (DBMS_LOB パッケージ) : LOB の長さの取得	9-135
C (OCI) : LOB の長さの取得	9-135
COBOL (Pro*COBOL) : LOB の長さの取得	9-137
C/C++ (Pro*C/C++) : LOB の長さの取得	9-138
Visual Basic (OO4O) : LOB の長さの取得	9-139
Java (JDBC) : LOB の長さの取得	9-139
別の LOB への LOB の全体または一部のコピー	9-141
用途	9-141
使用上の注意	9-142
構文	9-142
使用例	9-143
例	9-143
PL/SQL (DBMS_LOB パッケージ) : 別の LOB への LOB の全体または一部のコピー	9-143
C (OCI) : 別の LOB への LOB の全体または一部のコピー	9-144
COBOL (Pro*COBOL) : 別の LOB への LOB の全体または一部のコピー	9-146
C/C++ (Pro*C/C++) : 別の LOB への LOB の全体または一部のコピー	9-148
Visual Basic (OO4O) : 別の LOB への LOB の全体または一部のコピー	9-149
Java (JDBC) : 別の LOB への LOB の全体または一部のコピー	9-149
LOB ロケータのコピー	9-152
用途	9-152
使用上の注意	9-152
構文	9-152
使用例	9-153
例	9-153
PL/SQL (DBMS_LOB パッケージ) : LOB ロケータのコピー	9-154
C (OCI) : LOB ロケータのコピー	9-154
COBOL (Pro*COBOL) : LOB ロケータのコピー	9-156
C/C++ (Pro*C/C++) : LOB ロケータのコピー	9-157
Visual Basic (OO4O) : LOB ロケータのコピー	9-158
Java (JDBC) : LOB ロケータのコピー	9-158
2 つの LOB ロケータが等しいかどうかの確認	9-160
用途	9-160
使用上の注意	9-160
構文	9-160

使用例	9-161
例	9-161
C (OCI) : 2つの LOB ロケータが等しいかどうかの確認	9-161
C/C++ (Pro*C/C++) : 2つの LOB ロケータが等しいかどうかの確認	9-163
Java (JDBC) : 2つの LOB ロケータが等しいかどうかの確認	9-164
LOB ロケータが初期化されているかどうかの確認	9-167
用途	9-167
使用上の注意	9-168
構文	9-168
使用例	9-168
例	9-168
C (OCI) : LOB ロケータが初期化されているかどうかの確認	9-169
C/C++ (Pro*C/C++) : LOB ロケータが初期化されているかどうかの確認	9-170
キャラクタ・セット ID の取得	9-172
用途	9-172
使用上の注意	9-172
構文	9-173
使用例	9-173
例	9-173
C (OCI) : キャラクタ・セット ID の取得	9-173
キャラクタ・セット・フォームの取得	9-175
用途	9-175
使用上の注意	9-175
構文	9-176
使用例	9-176
例	9-176
C (OCI) : キャラクタ・セット・フォームの取得	9-176
他の LOB への LOB の追加	9-178
用途	9-179
使用上の注意	9-179
構文	9-179
使用例	9-180
例	9-180
PL/SQL (DBMS_LOB パッケージ) : 他の LOB への LOB の追加	9-180
C (OCI) : 他の LOB への LOB の追加	9-181

COBOL (Pro*COBOL) : 他の LOB への LOB の追加	9-183
C/C++ (Pro*C/C++) : 他の LOB への LOB の追加	9-184
Visual Basic (OO4O) : 他の LOB への LOB の追加	9-185
Java (JDBC) : 他の LOB への LOB の追加	9-185
LOB への追加の書込み	9-188
用途	9-188
使用上の注意	9-189
構文	9-189
使用例	9-190
例	9-190
PL/SQL (DBMS_LOB パッケージ) : LOB への追加の書込み	9-190
C (OCI) : LOB への追加の書込み	9-191
COBOL (Pro*COBOL) : LOB への追加の書込み	9-192
C/C++ (Pro*C/C++) : LOB への追加の書込み	9-194
Java (JDBC) : LOB への追加の書込み	9-195
LOB へのデータの書込み	9-197
用途	9-198
使用上の注意	9-198
構文	9-199
使用例	9-200
例	9-200
PL/SQL (DBMS_LOB パッケージ) : LOB へのデータの書込み	9-201
C (OCI) : LOB へのデータの書込み	9-202
COBOL (Pro*COBOL) : LOB へのデータの書込み	9-205
C/C++ (Pro*C/C++) : LOB へのデータの書込み	9-208
Visual Basic (OO4O) : LOB へのデータの書込み	9-210
Java (JDBC) : LOB へのデータの書込み	9-211
LOB データの切捨て	9-213
用途	9-214
使用上の注意	9-214
構文	9-214
使用例	9-215
例	9-215
PL/SQL (DBMS_LOB パッケージ) : LOB データの切捨て	9-215
C (OCI) : LOB データの切捨て	9-216

COBOL (Pro*COBOL) :LOB データの切捨て	9-217
C/C++ (Pro*C/C++) :LOB データの切捨て	9-218
Visual Basic (OO4O) :LOB データの切捨て	9-220
Java (JDBC) :LOB データの切捨て	9-221
LOB の一部の消去	9-223
用途	9-224
使用上の注意	9-224
構文	9-224
使用例	9-225
例	9-225
PL/SQL (DBMS_LOB パッケージ) :LOB の一部の消去	9-225
C (OCI) :LOB の一部の消去	9-226
COBOL (Pro*COBOL) :LOB の一部の消去	9-227
C/C++ (Pro*C/C++) :LOB の一部の消去	9-228
Visual Basic (OO4O) :LOB の一部の消去	9-229
Java (JDBC) :LOB の一部の消去	9-230
LOB バッファリングの使用可能化	9-232
用途	9-233
使用上の注意	9-233
構文	9-233
使用例	9-234
例	9-234
C (OCI) :LOB バッファリングの使用可能化	9-234
COBOL (Pro*COBOL) :LOB バッファリングの使用可能化	9-234
C/C++ (Pro*C/C++) :LOB バッファリングの使用可能化	9-236
Visual Basic (OO4O) :LOB バッファリングの使用可能化	9-237
バッファのフラッシュ	9-238
用途	9-239
使用上の注意	9-239
構文	9-239
使用例	9-240
例	9-240
C (OCI) :バッファのフラッシュ	9-240
COBOL (Pro*COBOL) :バッファのフラッシュ	9-240
C/C++ (Pro*C/C++) :バッファのフラッシュ	9-242

Visual Basic (OO4O) :バッファのフラッシュ	9-243
LOB バッファリングの使用禁止化	9-244
用途	9-245
使用上の注意	9-245
構文	9-245
使用例	9-246
例	9-246
C (OCI) :LOB バッファリングの使用禁止化	9-246
COBOL (Pro*COBOL) :LOB バッファリングの使用禁止化	9-248
C/C++ (Pro*C/C++) :LOB バッファリングの使用禁止化	9-250
Visual Basic (OO4O) :LOB バッファリングの使用禁止化	9-251
LOB または LOB データ全体を更新する 3 つの方法	9-252
EMPTY_CLOB() または EMPTY_BLOB() を使用した LOB の更新	9-254
用途	9-255
使用上の注意	9-255
構文	9-255
使用例	9-255
例	9-255
SQL: EMPTY_CLOB() または EMPTY_BLOB() を使用した LOB の更新	9-256
別の表からの LOB の選択による行の更新	9-257
用途	9-257
使用上の注意	9-257
構文	9-257
使用例	9-258
例	9-258
SQL: 別の表からの LOB の選択による行の更新	9-258
初期化した LOB ロケータ・バインド変数を使用した更新	9-259
用途	9-259
使用上の注意	9-259
構文	9-259
使用例	9-260
例	9-260
SQL: 初期化した LOB ロケータ・バインド変数を使用した更新	9-260
C (OCI) :初期化した LOB ロケータ・バインド変数を使用した更新	9-261
COBOL (Pro*COBOL) :初期化した LOB ロケータ・バインド変数を使用した更新	9-262

C/C++ (Pro*C/C++) : 初期化した LOB ロケータ・バインド変数を使用した更新	9-264
Visual Basic (OO4O) : 初期化した LOB ロケータ・バインド変数を使用した更新	9-265
Java (JDBC) : 初期化した LOB ロケータ・バインド変数を使用した更新	9-265
LOB を含む表の行の削除	9-267
用途	9-267
使用上の注意	9-267
構文	9-268
使用例	9-268
例	9-268
SQL: LOB の削除	9-268

10 テンポラリ LOB

ユースケース・モデル: 内部テンポラリ LOB	10-2
プログラム環境	10-6
ロケータ	10-6
IN 値として使用できるテンポラリ LOB ロケータ	10-6
テンポラリ LOB および内部永続 LOB に使用できる関数	10-7
一時表領域へのテンポラリ LOB データの格納	10-7
テンポラリ LOB の存続期間	10-8
メモリー操作	10-8
ロケータおよびセマンティクス	10-9
テンポラリ LOB 固有の機能	10-10
テンポラリ LOB におけるセキュリティ上の問題	10-12
NOCOPY 制限事項	10-12
テンポラリ LOB の管理	10-13
テンポラリ LOB の作成	10-14
用途	10-14
使用上の注意	10-14
構文	10-15
使用例	10-15
例	10-15
PL/SQL (DBMS_LOB パッケージ) : テンポラリ LOB の作成	10-16
C (OCI) : テンポラリ LOB の作成	10-16
COBOL (Pro*COBOL) : テンポラリ LOB の作成	10-18
C/C++ (Pro*C/C++) : テンポラリ LOB の作成	10-20

LOB がテンポラリ LOB であるかどうかの確認	10-22
用途	10-22
使用上の注意	10-22
構文	10-22
使用例	10-23
例	10-23
PL/SQL (DBMS_LOB パッケージ) : LOB がテンポラリ LOB であるかどうかの確認	10-24
C (OCI) : LOB がテンポラリ LOB であるかどうかの確認	10-24
COBOL (Pro*COBOL) : LOB がテンポラリ LOB であるかどうかの確認	10-25
C/C++ (Pro*C/C++) : LOB がテンポラリ LOB であるかどうかの確認	10-26
テンポラリ LOB の解放	10-28
用途	10-28
使用上の注意	10-28
構文	10-29
使用例	10-29
例	10-29
PL/SQL (DBMS_LOB パッケージ) : テンポラリ LOB の解放	10-30
C (OCI) : テンポラリ LOB の解放	10-30
COBOL (Pro*COBOL) : テンポラリ LOB の解放	10-31
C/C++ (Pro*C/C++) : テンポラリ LOB の解放	10-32
テンポラリ LOB への BFILE データのロード	10-33
用途	10-33
使用上の注意	10-34
構文	10-34
使用例	10-34
例	10-34
PL/SQL (DBMS_LOB パッケージ) : テンポラリ LOB への BFILE データのロード	10-35
C (OCI) : テンポラリ LOB への BFILE データのロード	10-35
COBOL (Pro*COBOL) : テンポラリ LOB への BFILE データのロード	10-37
C/C++ (Pro*C/C++) : テンポラリ LOB への BFILE データのロード	10-38
テンポラリ LOB がオープンしているかどうかの確認	10-40
用途	10-40
使用上の注意	10-40
構文	10-40
使用例	10-41

例	10-41
PL/SQL: テンポラリ LOB がオープンしているかどうかの確認	10-41
C (OCI) : テンポラリ LOB がオープンしているかどうかの確認	10-42
COBOL (Pro*COBOL) : テンポラリ LOB がオープンしているかどうかの確認	10-43
C/C++ (Pro*C/C++) : テンポラリ LOB がオープンしているかどうかの確認	10-44
テンポラリ LOB データの表示	10-46
用途	10-47
使用上の注意	10-47
構文	10-47
使用例	10-47
例	10-47
PL/SQL (DBMS_LOB パッケージ) : テンポラリ LOB データの表示	10-48
C (OCI) : テンポラリ LOB データの表示	10-49
COBOL (Pro*COBOL) : テンポラリ LOB データの表示	10-52
C/C++ (Pro*C/C++) : テンポラリ LOB データの表示	10-54
テンポラリ LOB からのデータの読み込み	10-56
用途	10-57
使用上の注意	10-57
構文	10-57
使用例	10-58
例	10-58
PL/SQL (DBMS_LOB パッケージ) : テンポラリ LOB からのデータの読み込み	10-58
C (OCI) : テンポラリ LOB からのデータの読み込み	10-59
COBOL (Pro*COBOL) : テンポラリ LOB からのデータの読み込み	10-62
C/C++ (Pro*C/C++) : テンポラリ LOB からのデータの読み込み	10-64
テンポラリ LOB の一部の読み込み (substr)	10-67
用途	10-68
使用上の注意	10-68
構文	10-68
使用例	10-68
例	10-68
PL/SQL (DBMS_LOB パッケージ) : テンポラリ LOB の一部の読み込み (substr)	10-69
COBOL (Pro*COBOL) : テンポラリ LOB の一部の読み込み (substr)	10-69
C/C++ (Pro*C/C++) : テンポラリ LOB の一部の読み込み (substr)	10-71
2つの (テンポラリ) LOB の全体または一部の比較	10-74

用途	10-74
使用上の注意	10-75
構文	10-75
使用例	10-75
例	10-75
PL/SQL (DBMS_LOB パッケージ) : 2つの (テンポラリ) LOB の全体または一部の比較	10-76
COBOL (Pro*COBOL) : 2つの (テンポラリ) LOB の全体または一部の比較	10-77
C/C++ (Pro*C/C++) : 2つの (テンポラリ) LOB の全体または一部の比較	10-79
テンポラリ LOB 内のパターンの有無の確認 (instr)	10-81
用途	10-81
使用上の注意	10-82
構文	10-82
使用例	10-82
例	10-82
PL/SQL (DBMS_LOB パッケージ) : テンポラリ LOB 内のパターンの有無の確認 (instr)	10-83
COBOL (Pro*COBOL) : テンポラリ LOB 内のパターンの有無の確認 (instr)	10-84
C/C++ (Pro*C/C++) : テンポラリ LOB 内のパターンの有無の確認 (instr)	10-86
テンポラリ LOB の長さの取得	10-88
用途	10-89
使用上の注意	10-89
構文	10-89
使用例	10-89
例	10-90
PL/SQL (DBMS_LOB パッケージ) : テンポラリ LOB の長さの取得	10-90
C (OCI) : テンポラリ LOB の長さの取得	10-91
COBOL (Pro*COBOL) : テンポラリ LOB の長さの取得	10-93
C/C++ (Pro*C/C++) : テンポラリ LOB の長さの取得	10-95
(テンポラリ) LOB の全体または一部の他へのコピー	10-97
用途	10-97
使用上の注意	10-98
構文	10-98
使用例	10-98
例	10-98
PL/SQL (DBMS_LOB パッケージ) : (テンポラリ) LOB の全体または一部の他へのコピー ..	10-99
C (OCI) : (テンポラリ) LOB の全体または一部の他へのコピー	10-100

COBOL (Pro*COBOL) : (テンポラリ) LOB の全体または一部の他へのコピー	10-103
C/C++ (Pro*C/C++) : (テンポラリ) LOB の全体または一部の他へのコピー	10-105
テンポラリ LOB の LOB ロケータのコピー	10-107
用途	10-107
使用上の注意	10-107
構文	10-108
使用例	10-108
例	10-108
PL/SQL (DBMS_LOB パッケージ) : テンポラリ LOB の LOB ロケータのコピー	10-109
C (OCI) : テンポラリ LOB の LOB ロケータのコピー	10-110
COBOL (Pro*COBOL) : テンポラリ LOB の LOB ロケータのコピー	10-112
C/C++ (Pro*C/C++) : テンポラリ LOB の LOB ロケータのコピー	10-114
テンポラリ LOB ロケータが他と等しいかどうかの確認	10-116
用途	10-116
使用上の注意	10-116
構文	10-117
使用例	10-117
例	10-117
C (OCI) : テンポラリ LOB の LOB ロケータが他と等しいかどうかの確認	10-118
C/C++ (Pro*C/C++) : テンポラリ LOB の LOB ロケータが他と等しいかどうかの確認	10-119
テンポラリ LOB の LOB ロケータが初期化されているかどうかの確認	10-121
用途	10-121
使用上の注意	10-121
構文	10-122
使用例	10-122
例	10-122
C (OCI) : テンポラリ LOB の LOB ロケータが初期化されているかどうかの確認	10-123
C/C++ (Pro*C/C++) : テンポラリ LOB の LOB ロケータが初期化されているかどうかの 確認	10-123
テンポラリ LOB のキャラクタ・セット ID の取得	10-125
用途	10-125
使用上の注意	10-126
構文	10-126
使用例	10-126
例	10-126

C (OCI) : テンポラリ LOB のキャラクタ・セット ID の取得	10-127
テンポラリ LOB のキャラクタ・セット・フォームの取得	10-128
用途	10-128
使用上の注意	10-129
構文	10-129
使用例	10-129
例	10-129
C (OCI) : テンポラリ LOB のキャラクタ・セット・フォームの取得	10-130
(テンポラリ) LOB の他への追加	10-131
用途	10-131
使用上の注意	10-132
構文	10-132
使用例	10-132
例	10-132
PL/SQL (DBMS_LOB パッケージ) : (テンポラリ) LOB の他への追加	10-133
C (OCI) : (テンポラリ) LOB の他への追加	10-133
COBOL (Pro*COBOL) : (テンポラリ) LOB の他への追加	10-136
C/C++ (Pro*C/C++) : (テンポラリ) LOB の他への追加	10-138
テンポラリ LOB への追加での書込み	10-140
用途	10-141
使用上の注意	10-141
構文	10-141
使用例	10-141
例	10-141
PL/SQL (DBMS_LOB パッケージ) : テンポラリ LOB への追加での書込み	10-142
C (OCI) : テンポラリ LOB への追加での書込み	10-143
COBOL (Pro*COBOL) : テンポラリ LOB への追加での書込み	10-144
C/C++ (Pro*C/C++) : テンポラリ LOB への追加での書込み	10-146
テンポラリ LOB へのデータの書込み	10-148
用途	10-149
使用上の注意	10-149
構文	10-150
使用例	10-150
例	10-150
PL/SQL (DBMS_LOB パッケージ) : テンポラリ LOB へのデータの書込み	10-151

C (OCI) : テンポラリ LOB へのデータの書込み	10-151
COBOL (Pro*COBOL) : テンポラリ LOB へのデータの書込み	10-154
C/C++ (Pro*C/C++) : テンポラリ LOB へのデータの書込み	10-155
テンポラリ LOB データの切捨て	10-158
用途	10-159
使用上の注意	10-159
構文	10-159
使用例	10-159
例	10-159
PL/SQL (DBMS_LOB パッケージ) : テンポラリ LOB データの切捨て	10-160
C (OCI) : テンポラリ LOB データの切捨て	10-160
COBOL (Pro*COBOL) : テンポラリ LOB データの切捨て	10-162
C/C++ (Pro*C/C++) : テンポラリ LOB データの切捨て	10-164
テンポラリ LOB の一部の消去	10-166
用途	10-166
使用上の注意	10-167
構文	10-167
使用例	10-167
例	10-167
PL/SQL (DBMS_LOB パッケージ) : テンポラリ LOB の一部の消去	10-168
C (OCI) : テンポラリ LOB の一部の消去	10-168
COBOL (Pro*COBOL) : テンポラリ LOB の一部の消去	10-171
C/C++ (Pro*C/C++) : テンポラリ LOB の一部の消去	10-173
テンポラリ LOB に対する LOB バッファリングの使用可能化	10-175
用途	10-175
使用上の注意	10-175
構文	10-176
使用例	10-176
例	10-176
C (OCI) : テンポラリ LOB に対する LOB バッファリングの使用可能化	10-177
COBOL (Pro*COBOL) : テンポラリ LOB に対する LOB バッファリングの使用可能化	10-178
C/C++ (Pro*C/C++) : テンポラリ LOB に対する LOB バッファリングの使用可能化	10-180
テンポラリ LOB に対するバッファのフラッシュ	10-182
用途	10-182
使用上の注意	10-182

構文	10-182
使用例	10-183
例	10-183
C (OCI) : テンポラリ LOB に対するバッファのフラッシュ	10-183
COBOL (Pro*COBOL) : テンポラリ LOB に対するバッファのフラッシュ	10-185
C/C++ (Pro*C/C++) : テンポラリ LOB に対するバッファのフラッシュ	10-186
テンポラリ LOB に対する LOB バッファリングの使用禁止化	10-188
用途	10-188
使用上の注意	10-188
構文	10-189
使用例	10-189
例	10-189
C (OCI) : テンポラリ LOB に対する LOB バッファリングの使用禁止化	10-190
COBOL (Pro*COBOL) : テンポラリ LOB に対する LOB バッファリングの使用禁止化	10-191
C/C++ (Pro*C/C++) : テンポラリ LOB に対する LOB バッファリングの使用禁止化	10-193

11 外部 LOB (BFILE)

ユースケース・モデル:外部 LOB (BFILE)	11-2
外部 LOB (BFILE) へのアクセス	11-5
ディレクトリ・オブジェクト	11-5
BFILE ロケータの初期化	11-5
データベース・レコードへのオペレーティング・システム・ファイルの対応付け	11-6
BFILENAME() および値の初期化	11-7
ディレクトリ名の指定	11-8
BFILE セキュリティ	11-9
所有権および権限	11-9
ディレクトリ・オブジェクトに対する読取り権限	11-9
BFILE セキュリティ用の SQL DDL	11-10
BFILE セキュリティ用の SQL DML	11-10
ディレクトリのカタログ・ビュー	11-10
DIRECTORY 使用のガイドライン	11-11
マルチスレッド・サーバー (MTS)・モードの BFILE	11-12
外部 LOB (BFILE) ロケータ	11-12
BFILE を含む表を作成する 3 つの方法	11-14
1 つ以上の BFILE 列を含む表の作成	11-15

用途	11-15
使用上の注意	11-15
構文	11-15
使用例	11-16
例	11-16
SQL: 1 つ以上の BFILE 列を含む表の作成	11-16
BFILE 属性を持つオブジェクト型の表の作成	11-18
用途	11-18
使用上の注意	11-18
構文	11-18
使用例	11-19
例	11-19
SQL: BFILE 属性を持つオブジェクト型の表の作成	11-19
BFILE を含む NESTED TABLE を持つ表の作成	11-21
用途	11-21
使用上の注意	11-21
構文	11-21
使用例	11-22
例	11-22
SQL: BFILE を含む NESTED TABLE を持つ表の作成	11-22
BFILE を含む行を挿入する 3 つの方法	11-23
BFILENAME() を使用した行の挿入	11-24
用途	11-25
使用上の注意	11-25
構文	11-26
使用例	11-26
例	11-27
SQL: BFILENAME() を使用した行の挿入	11-27
C (OCI) : BFILENAME() を使用した行の挿入	11-27
COBOL (Pro*COBOL) : BFILENAME() を使用した行の挿入	11-28
C/C++ (Pro*C/C++) : BFILENAME() を使用した行の挿入	11-29
Visual Basic (OO4O) : BFILENAME() を使用した行の挿入	11-30
Java (JDBC) : BFILENAME() を使用した行の挿入	11-30
別の表からの BFILE の選択による BFILE 行の挿入	11-32
用途	11-32

使用上の注意	11-32
構文	11-32
使用例	11-33
例	11-33
SQL: 別の表からの BFILE の選択による BFILE を含む行の挿入	11-33
初期化した BFILE ロケータを使用した BFILE を含む行の挿入	11-34
用途	11-35
使用上の注意	11-35
構文	11-35
使用例	11-36
例	11-36
PL/SQL: 初期化した BFILE ロケータを使用した BFILE を含む行の挿入	11-36
C (OCI) : 初期化した BFILE ロケータを使用した BFILE を含む行の挿入	11-36
COBOL (Pro*COBOL) : 初期化した BFILE ロケータを使用した BFILE を含む行の挿入	11-37
C/C++ (Pro*C/C++) : 初期化した BFILE ロケータを使用した BFILE を含む行の挿入	11-39
Visual Basic (OO4O) : 初期化した BFILE ロケータを使用した BFILE を含む行の挿入	11-39
Java (JDBC) : 初期化した BFILE ロケータを使用した BFILE を含む行の挿入	11-40
外部 LOB (BFILE) へのデータのロード	11-42
用途	11-42
使用上の注意	11-43
構文	11-43
使用例	11-43
例	11-44
BFILE へのデータのロード : ファイル名だけの動的な指定	11-44
BFILE へのデータのロード : ファイル名および DIRECTORY オブジェクトの動的な指定	11-45
LOB への BFILE データのロード	11-46
用途	11-47
使用上の注意	11-47
構文	11-48
使用例	11-48
例	11-49
PL/SQL (DBMS_LOB パッケージ) : LOB への BFILE データのロード	11-49
C (OCI) : LOB への BFILE データのロード	11-49
COBOL (Pro*COBOL) : LOB への BFILE データのロード	11-51
C/C++ (Pro*C/C++) : LOB への BFILE データのロード	11-52

Visual Basic (OO4O) :LOB への BFILE データのロード	11-54
Java (JDBC) :LOB への BFILE データのロード	11-54
BFILE をオープンする 2 つの方法	11-55
推奨事項 : OPEN を使用した BFILE のオープン	11-56
BFILE の最大オープン数の指定 : SESSION_MAX_OPEN_FILES	11-56
FILEOPEN を使用した BFILE のオープン	11-57
用途	11-57
使用上の注意	11-58
構文	11-58
使用例	11-58
例	11-58
PL/SQL: FILEOPEN を使用した BFILE のオープン	11-59
C (OCI) : FILEOPEN を使用した BFILE のオープン	11-59
Visual Basic (OO4O) : FILEOPEN を使用した BFILE のオープン	11-60
Java (JDBC) : FILEOPEN を使用した BFILE のオープン	11-60
OPEN を使用した BFILE のオープン	11-62
用途	11-62
使用上の注意	11-63
構文	11-63
使用例	11-63
例	11-63
PL/SQL: OPEN を使用した BFILE のオープン	11-64
C (OCI) : OPEN を使用した BFILE のオープン	11-64
COBOL (Pro*COBOL) : OPEN を使用した BFILE のオープン	11-65
C/C++ (Pro*C/C++) : OPEN を使用した BFILE のオープン	11-66
Visual Basic (OO4O) : OPEN を使用した BFILE のオープン	11-67
Java (JDBC) : OPEN を使用した BFILE のオープン	11-67
BFILE のオープンを確認する 2 つの方法	11-69
推奨事項 : OPEN を使用した BFILE のオープン	11-69
BFILE の最大オープン数の指定 : SESSION_MAX_OPEN_FILES	11-69
FILEISOPEN を使用した BFILE のオープンの確認	11-71
用途	11-71
使用上の注意	11-71
構文	11-71
使用例	11-72

例	11-72
PL/SQL (DBMS_LOB パッケージ) : FILEISOPEN を使用した BFILE のオープンの確認	11-72
C (OCI) : FILEISOPEN を使用した BFILE のオープンの確認	11-73
Visual Basic (OO4O) : FILEISOPEN を使用した BFILE のオープンの確認	11-74
Java (JDBC) : FILEISOPEN を使用した BFILE のオープンの確認	11-75
ISOPEN を使用した BFILE のオープンの確認	11-77
用途	11-77
使用上の注意	11-77
構文	11-77
使用例	11-78
例	11-78
PL/SQL (DBMS_LOB パッケージ) : ISOPEN を使用した BFILE のオープンの確認	11-79
C (OCI) : ISOPEN を使用した BFILE のオープンの確認	11-79
COBOL (Pro*COBOL) : ISOPEN を使用した BFILE のオープンの確認	11-81
C/C++ (Pro*C/C++) : ISOPEN を使用した BFILE のオープンの確認	11-82
Visual Basic (OO4O) : ISOPEN を使用した BFILE のオープンの確認	11-83
Java (JDBC) : ISOPEN を使用した BFILE のオープンの確認	11-83
BFILE データの表示	11-86
用途	11-86
使用上の注意	11-87
構文	11-87
使用例	11-87
例	11-87
PL/SQL: BFILE データの表示	11-88
C (OCI) : BFILE データの表示	11-88
COBOL (Pro*COBOL) : BFILE データの表示	11-91
C/C++ (Pro*C/C++) : BFILE データの表示	11-93
Visual Basic (OO4O) : BFILE データの表示	11-94
Java (JDBC) : BFILE データの表示	11-95
BFILE からのデータの読み込み	11-97
用途	11-97
使用上の注意	11-98
構文	11-98
使用例	11-99
例	11-99

PL/SQL (DBMS_LOB パッケージ) : BFILE からのデータの読み込み	11-100
C (OCI) : BFILE からのデータの読み込み	11-100
COBOL (Pro*COBOL) : BFILE からのデータの読み込み	11-102
C/C++ (Pro*C/C++) : BFILE からのデータの読み込み	11-103
Visual Basic (OO4O) : BFILE からのデータの読み込み	11-104
Java (JDBC) : BFILE からのデータの読み込み	11-105
BFILE データの一部の読み込み (substr)	11-107
用途	11-107
使用上の注意	11-108
構文	11-108
使用例	11-108
例	11-108
PL/SQL (DBMS_LOB パッケージ) : BFILE データの一部の読み込み (substr)	11-109
COBOL (Pro*COBOL) : BFILE データの一部の読み込み (substr)	11-109
C/C++ (Pro*C/C++) : BFILE データの一部の読み込み (substr)	11-110
Visual Basic (OO4O) : BFILE データの一部の読み込み (substr)	11-111
Java (JDBC) : BFILE データの一部の読み込み (substr)	11-112
2 つの BFILE の全体または一部の比較	11-114
用途	11-115
使用上の注意	11-115
構文	11-115
使用例	11-116
例	11-116
PL/SQL (DBMS_LOB パッケージ) : 2 つの BFILE の全体または一部の比較	11-116
COBOL (Pro*COBOL) : 2 つの BFILE の全体または一部の比較	11-117
C/C++ (Pro*C/C++) : 2 つの BFILE の全体または一部の比較	11-119
Visual Basic (OO4O) : 2 つの BFILE の全体または一部の比較	11-120
Java (JDBC) : 2 つの BFILE の全体または一部の比較	11-121
BFILE 内のパターンの有無の確認 (instr)	11-123
用途	11-124
使用上の注意	11-124
構文	11-124
使用例	11-125
例	11-125
PL/SQL (DBMS_LOB パッケージ) : BFILE 内のパターンの有無の確認 (instr)	11-125

COBOL (Pro*COBOL) : BFILE 内のパターンの有無の確認 (instr)	11-126
C/C++ (Pro*C/C++) : BFILE 内のパターンの有無の確認 (instr)	11-127
Java (JDBC) : BFILE 内のパターンの有無の確認 (instr)	11-129
BFILE が存在するかどうかの確認	11-131
用途	11-132
使用上の注意	11-132
構文	11-132
使用例	11-133
例	11-133
PL/SQL (DBMS_LOB パッケージ) : BFILE が存在するかどうかの確認	11-133
C (OCI) : BFILE が存在するかどうかの確認	11-134
COBOL (Pro*COBOL) : BFILE が存在するかどうかの確認	11-135
C/C++ (Pro*C/C++) : BFILE が存在するかどうかの確認	11-136
Visual Basic (OO4O) : BFILE が存在するかどうかの確認	11-137
Java (JDBC) : BFILE が存在するかどうかの確認	11-138
BFILE の長さの取得	11-140
用途	11-141
使用上の注意	11-141
構文	11-141
使用例	11-142
例	11-142
PL/SQL (DBMS_LOB パッケージ) : BFILE の長さの取得	11-142
C (OCI) : BFILE の長さの取得	11-143
COBOL (Pro*COBOL) : BFILE の長さの取得	11-144
C/C++ (Pro*C/C++) : BFILE の長さの取得	11-145
Visual Basic (OO4O) : BFILE の長さの取得	11-146
Java (JDBC) : BFILE の長さの取得	11-147
BFILE 用の LOB ロケータのコピー	11-149
用途	11-150
使用上の注意	11-150
構文	11-150
使用例	11-150
例	11-150
PL/SQL: BFILE 用の LOB ロケータのコピー	11-151
C (OCI) : BFILE 用の LOB ロケータのコピー	11-151

COBOL (Pro*COBOL) : BFILE 用の LOB ロケータのコピー	11-153
C/C++ (Pro*C/C++) : BFILE 用の LOB ロケータのコピー	11-154
Java (JDBC) : BFILE 用の LOB ロケータのコピー	11-154
BFILE の LOB ロケータが初期化されているかどうかの確認	11-156
用途	11-156
使用上の注意	11-157
構文	11-157
使用例	11-157
例	11-157
C (OCI) : BFILE の LOB ロケータが初期化されているかどうかの確認	11-158
C/C++ (Pro*C/C++) : BFILE の LOB ロケータが初期化されているかどうかの確認	11-159
BFILE の LOB ロケータが他と等しいかどうかの確認	11-161
用途	11-162
使用上の注意	11-162
構文	11-162
使用例	11-162
例	11-163
C (OCI) : BFILE の LOB ロケータが他と等しいかどうかの確認	11-163
C/C++ (Pro*C/C++) : BFILE の LOB ロケータが他と等しいかどうかの確認	11-165
Java (JDBC) : BFILE の LOB ロケータが他と等しいかどうかの確認	11-166
ディレクトリ別名およびファイル名の取得	11-168
用途	11-169
使用上の注意	11-169
構文	11-169
使用例	11-169
例	11-170
PL/SQL (DBMS_LOB パッケージ) : ディレクトリ別名およびファイル名の取得	11-170
C (OCI) : ディレクトリ別名およびファイル名の取得	11-170
COBOL (Pro*COBOL) : ディレクトリ別名およびファイル名の取得	11-172
C/C++ (Pro*C/C++) : ディレクトリ別名およびファイル名の取得	11-173
Visual Basic (OO4O) : ディレクトリ別名およびファイル名の取得	11-174
Java (JDBC) : ディレクトリ別名およびファイル名の取得	11-175
BFILE を含む行を更新する 3 つの方法	11-177
BFILENAME() を使用した BFILE の更新	11-178
使用上の注意	11-179

構文	11-179
使用例	11-180
例	11-180
SQL: BFILENAME() を使用した BFILE の更新	11-180
別の表からの BFILE の選択による BFILE の更新	11-181
用途	11-181
使用上の注意	11-181
構文	11-181
使用例	11-182
例	11-182
SQL: 別の表からの BFILE の選択による BFILE の更新	11-182
初期化した BFILE ロケータを使用した BFILE の更新	11-183
用途	11-184
使用上の注意	11-184
構文	11-184
使用例	11-184
例	11-185
PL/SQL: 初期化した BFILE ロケータを使用した BFILE の更新	11-185
C (OCI) : 初期化した BFILE ロケータを使用した BFILE の更新	11-185
COBOL (Pro*COBOL) : 初期化した BFILE ロケータを使用した BFILE の更新	11-186
C/C++ (Pro*C/C++) : 初期化した BFILE ロケータを使用した BFILE の更新	11-188
Visual Basic (OO4O) : 初期化した BFILE ロケータを使用した BFILE の更新	11-188
Java (JDBC) : 初期化した BFILE ロケータを使用した BFILE の更新	11-189
BFILE をクローズする 2 つの方法	11-191
FILECLOSE を使用した BFILE のクローズ	11-193
用途	11-193
使用上の注意	11-194
構文	11-194
使用例	11-194
例	11-194
PL/SQL (DBMS_LOB パッケージ) : FILECLOSE を使用した BFILE のクローズ	11-195
C (OCI) : FILECLOSE を使用した BFILE のクローズ	11-195
Visual Basic (OO4O) : FILECLOSE を使用した BFILE のクローズ	11-196
Java (JDBC) : FILECLOSE を使用した BFILE のクローズ	11-196
CLOSE を使用した BFILE のクローズ	11-198

用途	11-198
使用上の注意	11-199
構文	11-199
使用例	11-199
例	11-199
PL/SQL (DBMS_LOB パッケージ) : CLOSE を使用した BFILE のクローズ	11-200
C (OCI) : CLOSE を使用した BFILE のクローズ	11-200
COBOL (Pro*COBOL) : CLOSE を使用した BFILE のクローズ	11-201
C/C++ (Pro*C/C++) : CLOSE を使用した BFILE のクローズ	11-202
Visual Basic (OO4O) : CLOSE を使用した BFILE のクローズ	11-203
Java (JDBC) : CLOSE を使用した BFILE のクローズ	11-203
オープン中のすべての BFILE のクローズ	11-205
用途	11-206
使用上の注意	11-206
構文	11-206
使用例	11-207
例	11-207
PL/SQL (DBMS_LOB パッケージ) : オープン中のすべての BFILE のクローズ	11-207
C (OCI) : オープン中のすべての BFILE のクローズ	11-207
COBOL (Pro*COBOL) : オープン中のすべての BFILE のクローズ	11-208
C/C++ (Pro*C/C++) : オープン中のすべての BFILE のクローズ	11-210
Visual Basic (OO4O) : オープン中のすべての BFILE のクローズ	11-211
Java (JDBC) : オープン中のすべての BFILE のクローズ	11-211
BFILE を含む表の行の削除	11-214
用途	11-214
使用上の注意	11-214
構文	11-215
使用例	11-215
例	11-215
SQL: 表からの行の削除	11-215

索引

はじめに

このマニュアルでは、Oracle8i におけるアプリケーション開発のうち、ラージ・オブジェクト（LOB）に関連する機能について説明します。このマニュアルの情報は、すべてのプラットフォームで実行する Oracle Server のバージョンに適用されますが、システム固有の情報は含まれていません。

内容は次のとおりです。

- このマニュアルについて
- 機能範囲および可用性
- LOB の新機能
- このマニュアルの新規事項
- 関連マニュアル
- このマニュアルの構成
- このマニュアルの表記規則
- 図の解釈方法
- ユースケース図の要素

このマニュアルについて

このマニュアルは、LOB を使用する新しいアプリケーションを開発するプログラマの他に、すでにこのテクノロジーを実装して、さらに新機能を活用しようとしているプログラマを対象としています。

マルチメディア・データのような、構造化されていないデータの重要性が高まりつつあることを受けて、Oracle アプリケーション開発者用のドキュメント・セットの中で独立したマニュアルとして提供されるようになりました。

機能範囲および可用性

このマニュアルでは、Oracle8i および Oracle8i Enterprise Edition 製品の特徴および機能について説明します。

Oracle8i および Oracle8i Enterprise Edition の基本機能は同じです。ただし、最新機能の中には、Enterprise Edition のみで使用可能であり、オプションのものもあります。たとえば、オブジェクト機能を使用するには、Enterprise Edition およびオブジェクト・オプションが必要です。

必要なオプション

LOB の取扱いについては特別な制限はありません。制限の詳細は、第4章「LOB の管理」を参照してください。次のオプションが必要です。

- Oracle Partitioning Option: パーティション表で LOB を使用します。
- Oracle オブジェクト・オプション: オブジェクト型で LOB を使用します。

Oracle8i と Oracle8i Enterprise Edition の相違点、および使用可能な機能とオプションの詳細は、次の URL からダウンロード可能な『Oracle8i: A Family of Database Products』を参照してください。

<http://www.oracle.com/database/availability/>

LOB の新機能

Oracle8i リリース 8.1.6 で導入された新機能

Oracle8i リリース 8.1.6 で導入された LOB の新機能には、次のものが含まれます。

- LOB 列に対する CACHE READS オプションの追加
- 内部 LOB にバインドするバインド変数に対する 4,000 バイトの制限の削除

Oracle8i リリース 8.1.5 で導入された機能

Oracle8i リリース 8.1.5 で導入された LOB の新機能には、次のものが含まれます。

- テンポラリ LOB
- 可変幅の CLOB および NCLOB のサポート
- パーティション表内の LOB のサポート
- LOB に対する新しい API (open/close/isopen、writeappend、getchunksizes)
- 非パーティション索引構成表内の LOB のサポート
- LOB への LONG の値のコピー

このマニュアルの新規事項

このマニュアルは、今回のリリースから、次のように変更されました。

- **再編成**: このマニュアルは、この章の後半に記載するとおり、再編成されています。たとえば、前版のマニュアルの第 1 章の内容は、複数の新しい章に分割されました。
- **FAQ に関する章の追加**: 第 6 章「FAQ」が、新しく追加されました。
- **LOB を使用する理由**: 第 1 章では、LOB の必要性および LOB の効果について説明しています。
- **ユースケース**: アプリケーション開発者用のドキュメント・セットのユースケースを統一するために、各ユースケースは同様の構造（用途、使用上の注意、構文、使用例および例）になっています。ユースケースのマスター表は、各ユースケースに使用可能なプログラム環境の例を含むように更新されています。
- **構文参照**: 第 9 章、第 10 章および第 11 章のユースケースについては、プログラム環境ごとに非常に詳細な構文参照があり、より詳細な構文情報を参照するための適切なマニュアル、章および項、またはオンライン・メニューについて記載しています。

- **注意事項の追加**: このマニュアルに追加された注意事項は、次のとおりです。
 - 第7章の「GBのLOBの作成方法」
 - 第3章「LOBプログラム環境」の「JDBC: 8.1.x および今後のリリースで使用できない OracleBlob および OracleClob」
 - 第5章「高度なトピック」の「LOBへの参照を含む VARRAY の作成」
 - 削除された制限は、第4章「LOBの管理」にリストを記載しています。詳細は、第7章「モデリングおよび設計」の「INSERT および UPDATE での 4,000 バイトを超えるバインド」で例をあげて説明しています。
 - 第10章「テンポラリLOB」の「テンポラリLOBへのデータの書込み」の項の「DBMS_LOB.WRITE() を使用したデータのテンポラリ BLOB への書込み」に、DBMS_LOB.WRITE に関するガイドラインが追加されています。
 - CACHE READS が、LOB の記憶域オプションとして追加されました。これは、第7章「モデリングおよび設計」の「LOB記憶域」にある「CACHE / NOCACHE / CACHE READS」で説明しています。このオプションの使用による、リリース 8.1 からのダウングレードへの影響については、これら注意事項を参照してください。
 - 第10章「テンポラリLOB」の「NOCOPY 制限事項」に、NOCOPY 制限およびガイドラインに対する参照が追加されています。
 - TO_LOB 関数について、NCLLOB ではなく CLOB にデータをコピーするために TO_LOB が使用されることをユーザーに注意するための注意事項が第9章「内部永続LOB」の「LOBへのLONGのコピー」に追加されています。

関連マニュアル

構文および実装の詳細は、次のマニュアルを参照してください。

- 『Oracle8i PL/SQL パッケージ・プロシージャ・リファレンス』
PL/SQL は、オラクル社が開発した、SQL のプロシージャ型拡張機能です。この高水準プログラム言語の詳細は、このマニュアルを参照してください。
- 『Oracle8i コール・インタフェース・プログラマーズ・ガイド』
Oracle Call Interface (OCI) について説明しています。OCI を使用すると、Oracle Server にアクセスする C または C++ の第3世代言語 (3GL) アプリケーションを作成できます。
- 『Oracle8i Pro*C/C++ プリコンパイラ・プログラマーズ・ガイド』
オラクル社は、プリコンパイラの Pro* シリーズも提供しています。これを使用することで、ご使用のアプリケーション・プログラムに SQL および PL/SQL を組み込みます。

- 『Oracle8i Pro*COBOL プリコンパイラ・プログラマーズ・ガイド』
Pro*COBOL プリコンパイラを使用することで、Oracle Server にアクセスするための COBOL プログラムに SQL および PL/SQL を組み込めます。
- 『Oracle プリコンパイラ・プログラマーズ・ガイド』 および 『Oracle プリコンパイラ・ガイド Pro*FORTRAN サプリメント』
Oracle Server にアクセスする FORTRAN プリコンパイラ・プログラミングについては、このマニュアルを参照してください。
- 『Programmer's Guide to SQL*Module for Ada』
Oracle Server へのアクセスを Ada でプログラミングする場合は、このマニュアルを参照してください。
- Java: Oracle 8i では、データベース内で Java を使用できます。Oracle Java ドキュメント・セットには、次のマニュアルが含まれています。
 - 『Oracle8i Enterprise JavaBeans と CORBA 開発者ガイド』
 - 『Oracle8i JDBC 開発者ガイドおよびリファレンス』
 - 『Oracle8i Java 開発者ガイド』
 - 『Oracle8i JPublisher ユーザーズ・ガイド』
 - 『Oracle8i Java スタアド・プロシージャ 開発者ガイド』

マルチメディア

Oracle のマルチメディア・テクノロジー開発環境には、様々な方法でアクセスできます。

- 『Oracle8i データ・カートリッジ開発者ガイド』
データベースと統合する自己完結型アプリケーションを作成するには、Oracle の拡張フレームワークについて、このマニュアルを参照してください。
- Oracle 自体の *interMedia* アプリケーションを使用するには、次のマニュアルを参照してください。
 - 『Oracle8i *interMedia* Audio、Image、Video ユーザーズ・ガイドおよびリファレンス』
 - 『Oracle8i *interMedia* Audio、Image、Video Java Client ユーザーズ・ガイドおよびリファレンス』
 - 『Oracle8i *interMedia* Locator ユーザーズ・ガイドおよびリファレンス』
 - 『Oracle8i *interMedia* Web』

基本的な参照

- 『Oracle8i SQL リファレンス』 および 『Oracle8i 管理者ガイド』
SQL 情報については、これらのマニュアルを参照してください。
- 『Oracle8i レプリケーション・ガイド』
LOB データとの Oracle レプリケーションの情報については、このマニュアルを参照してください。
- 『Oracle8i 概要』
Oracle の基本概念については、このマニュアルを参照してください。

このマニュアルの構成

このマニュアルは、次の 11 章で構成されています。各章の内容は次のとおりです。

第 1 章「概要」

この章では、非構造化データの必要性および LOB 使用のメリットについて説明します。CLOB による国際化を促進するための LOB の使用、および LONG のかわりに LOB を使用するメリットについて説明します。この章では、LOB のデモ・ファイルおよび提供された LOB のサンプル・スクリプトの記載場所についても説明します。

第 2 章「基本コンポーネント」

この章では、内部永続 LOB とテンポラリ LOB、および外部 LOB (BFILE) を含む LOB データ型について説明します。また、LOB を NULL または空に初期化する必要性についても説明します。LOB ロケータおよびその使用方法についても説明します。

第 3 章「LOB プログラム環境」

この章では、LOB を操作するために使用する、次の 6 つのプログラム環境について説明します。また、使用可能な LOB 関連のメソッドまたはプロシージャのリストも記載します。

- **DBMS_LOB パッケージ**を使用した PL/SQL
(『Oracle8i PL/SQL パッケージ・プロシージャ・リファレンス』を参照)
- **Oracle Call Interface (OCI)** を使用した C
(『Oracle8i コール・インタフェース・プログラマーズ・ガイド』を参照)
- **Pro*C/C++ プリコンパイラ**を使用した C++
(『Oracle8i Pro*C/C++ プリコンパイラ・プログラマーズ・ガイド』を参照)
- **Pro*COBOL プリコンパイラ**を使用した COBOL
(『Oracle8i Pro*COBOL プリコンパイラ・プログラマーズ・ガイド』を参照)

- **Oracle Objects For OLE (OO4O)** を使用した Visual Basic (付属のオンライン・マニュアルを参照)
- **JDBC アプリケーション・プログラマーズ・インタフェース (API)** を使用した Java (『Oracle8i JDBC 開発者ガイドおよびリファレンス』を参照)

第4章 「LOB の管理」

この章では、SQL*Loader の使用方法、LOB で作業する前に必要な DBA アクション、および LOB 制限について説明します。

第5章 「高度なトピック」

この章では、他のすべての章に関係する高度なトピックについて説明します。特に、読取り一貫性のあるロケータ、LOB のバッファリング・サブシステムおよびオブジェクト・キャッシュ内の LOB を中心に説明します。

第6章 「FAQ」

この章では、LOB に関連して、よくある質問およびそれに対応する回答を記載しています。

第7章 「モデリングおよび設計」

この章では、データ型の選択に関連した問題について説明し、LONG と LONGRAW のプロパティを比較します。また、表のアーキテクチャ設計の基準について説明し、表領域と記憶域の問題、参照セマンティクスとコピー・セマンティクス、索引構成表およびパーティション表についても説明します。さらに、LOB 列の索引作成およびパフォーマンス向上のための最善策についても説明します。

第8章 「サンプル・アプリケーション」

この章では、サンプル・マルチメディア事例およびソリューションを記載しています。Multimedia_tab 表の形式でのマルチメディア・アプリケーション・アーキテクチャの設計、関連オブジェクト、型および参照を記載しています。

第9章 「内部永続 LOB」

この章では、内部永続 LOB に関連する基本操作について、第8章に概要を示したシナリオに沿って、関連する問題とともに説明します。また、統一モデリング言語 (Unified Modeling Language: UML) の表記について、ユースケースの点から紹介します。各基本操作は、ユースケースで説明されています。UML については、このマニュアルでは詳しく説明していませんが、このマニュアルで使用する規則については、「はじめに」の後半で説明しています。各プログラム環境で、できるだけ同じ例を使用して説明しています。

第10章「テンポラリ LOB」

この章では、第9章と同じ形式で、テンポラリ LOB の新機能について説明します。新しい API とそれに伴う問題について、詳細に説明します。テンポラリ LOB の Visual Basic (OO4O) サンプル・スクリプトおよび Java (JDBC) サンプル・スクリプトは、このリリースでは提供されていませんが、今後のリリースでは使用可能です。

第11章「外部 LOB (BFILE)」

この章では、外部 LOB (BFILE) について説明します。この章も、第9章、第10章と同じ形式で説明します。各操作は、ユースケースとして扱われ、使用可能な各プログラム環境に適合したコード例を記載しています。

このマニュアルの表記規則

このマニュアルで使用する表記およびフォーマットの規則は、次のとおりです。

[]

大カッコは、中に含まれている項目がオプションであることを示します。カッコは入力しないでください。

{ }

中カッコは、複数の項目を囲み、その中の1つのみが必要であることを示します。

|

縦棒線は、中カッコ内の項目を分割します。また、複数の値を関数のパラメータに渡すことを示すためにも使用します。

...

コード例における省略記号は、その説明内容に関係のないコードを省略していることを意味します。

固定幅フォント

SQL または C のコード例は、固定幅フォントで示します。

イタリック

イタリックは、OCI パラメータ、OCI ルーチン名、ファイル名およびデータ・フィールドを示します。

大文字

大文字は、SELECT、UPDATE などの SQL キーワードを示します。

このマニュアルでは、ある情報に対して読者の注意を促すために、特別なテキスト・フォーマットを使用しています。太字テキストの見出しで始まる段落は、特別な意味を持つ場合があります。次の各段落では、それぞれの見出しで示される各種の情報を説明します。

注意：共通の問題を避けたり概念の理解を深めるために、情報に特別な注意を払う必要があることを示します。

警告：アプリケーションが正しく動作するために注意して行う必要があること、および行ってはならないことを OCI プログラマに示します。

参照：説明する項目についての追加情報が記載されているこのマニュアルの他の項、または他のマニュアルを示します。

図の解釈方法

このマニュアルの第 9 章、第 10 章および第 11 章で使用されているユースケース図は、統一モデリング言語 (UML) に基づいています。

ビジュアル・モデリングを使用する理由

アプリケーション開発者の会議などでは、黒板やフリップ・チャートに問題の説明と解決策の概要を表すスケッチを描き始める人がいます。これは、ソフトウェア開発に伴う複雑な関係を説明するには、図形とテキストを組み合わせる説明することが一番簡単であることを開発者が直感的に認識しているためです。このような会議では、参加者がこのスケッチをコピーして、後でコードを開発するときこのスケッチを参考にするということがよくあります。

統一モデリング言語

このプロセスで問題になるのは、議論されている問題を適切に表す表記法を図の作成者が考え出す必要があるということです。ほとんどの問題は参加者がすでに熟知しているものであり、また図の線や端点が何を意味するかがわからなければその場で質問することができません。ただし、ここでさらに次のような問題が発生します。会議に参加していない開発メンバーはどうなるのでしょうか？ 会議に参加したメンバーでも、後になってメモの基になった論理がわからなくなってしまうことがあります。

このような問題に対処するため、『アプリケーション開発者ガイド』では統一モデリング言語 (UML) で定義されたグラフィック表記を採用しています。UML は、ソフトウェア・システムのモデリング用に作成され広く使用されている業界標準です。UML については、このマニュアルでは詳しく説明しません。ただし、このマニュアルで使用する規則については説明しています。

イラストおよび図

ソフトウェアのドキュメントには、今までにも常に図が含まれていました。それでは、ソフトウェア開発のモデリングに使用する UML ベースの図と、これまで様々な項目を表示するために使用されてきた図ではどこが違うのでしょうか？このマニュアルでは、この 2 種類を次のように区別しています。

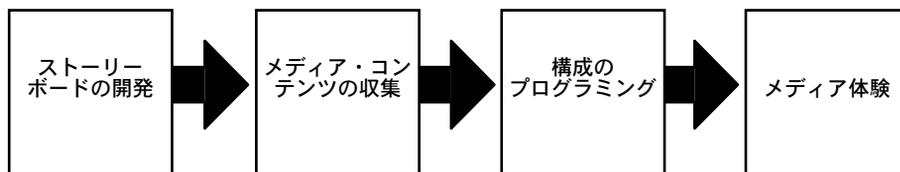
- イラスト：テクノロジーをより理解しやすくするために使用します。
- 図：ソフトウェアの実際のモデリングに使用します。

この 2 種類の違いは、図のタイトルで区別します。次の例では、図という用語を使用して説明します。

イラストの例

図 1 は、マルチメディア・アプリケーションの作成に伴う大まかなステップを示したものです。この図は、組織としての観点からソフトウェア開発を計画する際には有効ですが、実際のコーディングの際には役に立ちません。

図 1 イラストの例：マルチメディア・オーサリング処理

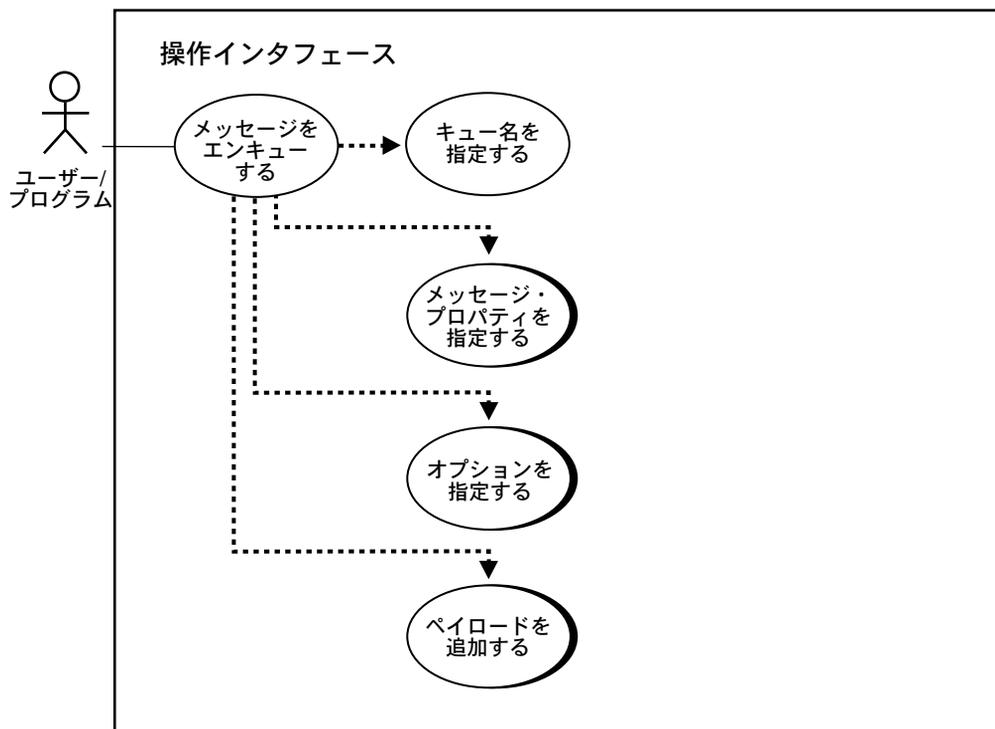


ユースケース図の例

注意：次のユースケース図は、ラージ・オブジェクト (LOB) の機能ではなく、アドバンスト・キューイングの機能を説明しています。便宜上、今後のリリースでは、このユースケース図の例は、ラージ・オブジェクト (LOB) の機能を説明するものとします。

図 1 とは対照的に、図 2 は、Oracle アドバンスト・キューイングを使用してメッセージをエンキューするために必要な操作について説明しています。キュー名の指定、メッセージ・プロパティの指定、および様々なオプションからの指定を行い、その後メッセージ・ペイロードを追加する必要があります。この図には、最後の 3 つの影付きの楕円で示したような補足的な図が追加されます。

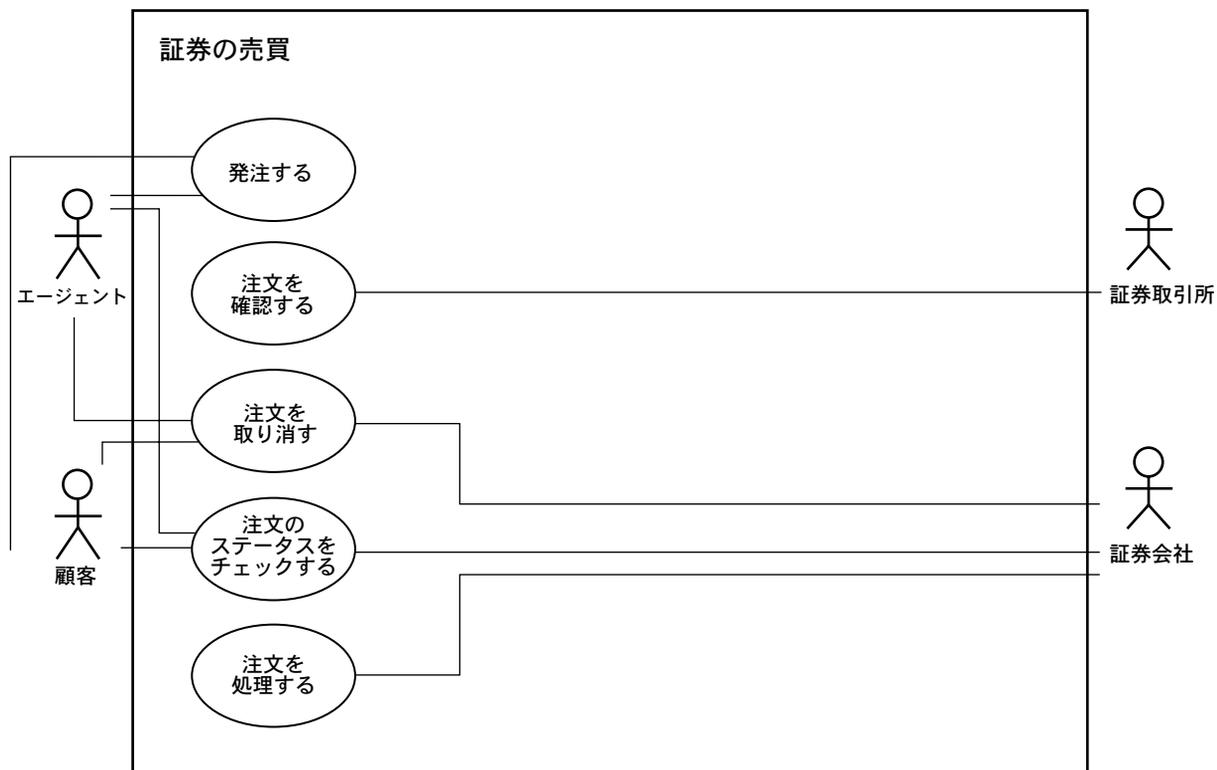
図2 ユースケース図：メッセージのエンキュー



ユースケース図の要素

一般に、ユースケースは、アプリケーションのシナリオ全体を構成する一連のアクティビティを記述するために使用されます。

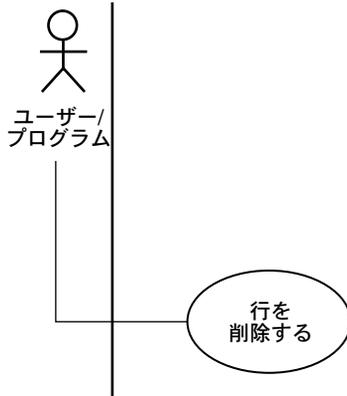
図3 ユースケース



次の項では、異なるケースで適用されるユースケース図の解釈方法について説明します。

図形要素

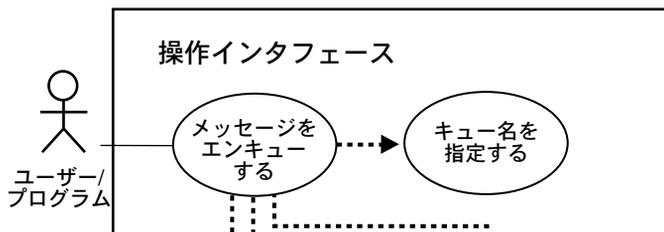
説明



1次ユースケースは、それぞれアクター（人を表すマーク）によって開始します。アクターは、ユーザー、アプリケーションまたはサブプログラムのどれでもかまいません。

アクターは、ユースケース・アクションを囲む楕円（吹出し）として示される1次ユースケースに接続されます。

1次ユースケース全体は、ユースケース・モデル図によって記述されます。



1次ユースケースを完了するには、他の操作が必要になる場合があります。この図では、

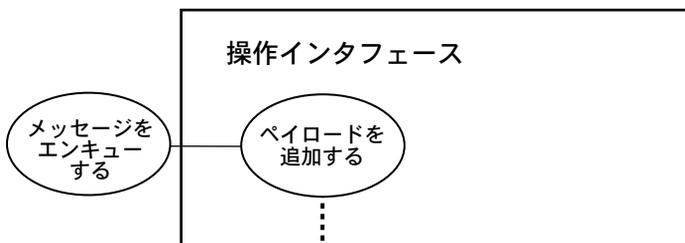
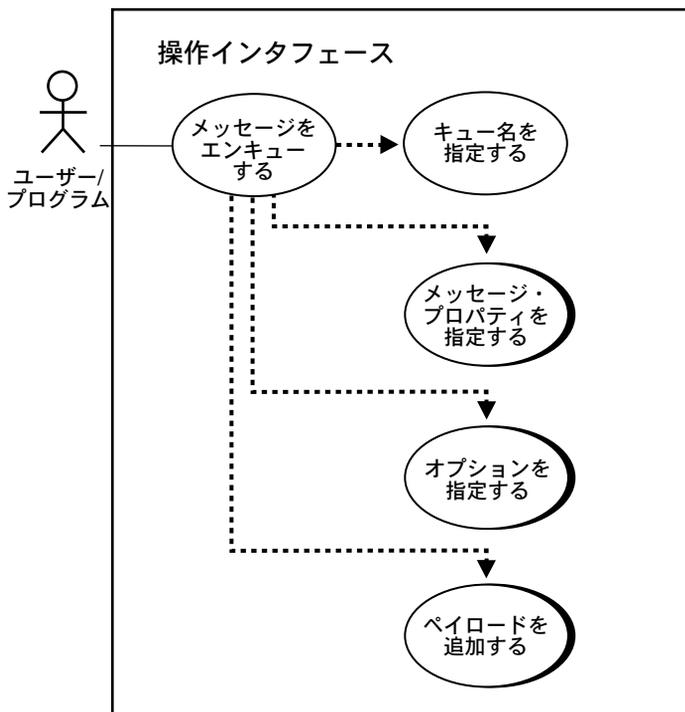
- キュー名を指定する

が、次に示すオペレーションを完了するために必要なサブオペレーション（2次ユース・ケース）の1つです。

- メッセージをエンキューする

1次ユース・ケースから、必要な他のオペレーション（ここでは省略されています）に向かって下向きの線が伸びています。

図形要素



説明

影付きの2次ユース・ケースは展開されます（自分自身のユースケース図によって記述されます）。これには次の2つの理由があります。

- (a) オペレーション・ロジックが理解しやすくなる。
- (b) すべてのオペレーションおよびサブオペレーションを同一ページ内に収めることができない。

この例では、

- メッセージ・プロパティを指定する
- オプションを指定する
- ペイロードを追加する

というアクションは、すべてさらに詳しいユースケース図に展開されます。

この図には、展開されたユースケース図が示されています。標準的な図は、通常、アクターから始まりますが、ここではユースケース自体がサブオペレーションへの出発点になっています。

この例では、

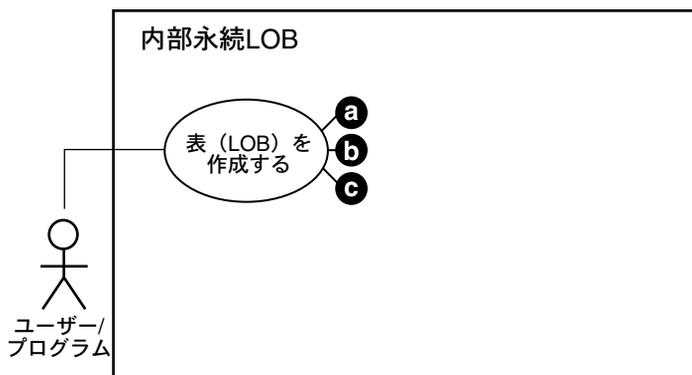
- ペイロードを追加する

の展開ビューが、次のオペレーションの構成要素を表しています。

- メッセージをエンキューする

図形要素

説明

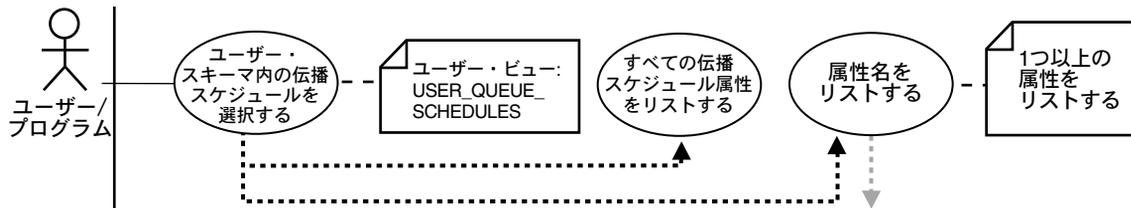


この表記 (a、b、c) は、LOB を含む表の作成に3つの異なる方法があることを示します。



これは注釈ボックスの使用方法の1つを示したもので、LOB を含む表の3つの作成方法のどれを表したものが区別できます。

図形要素

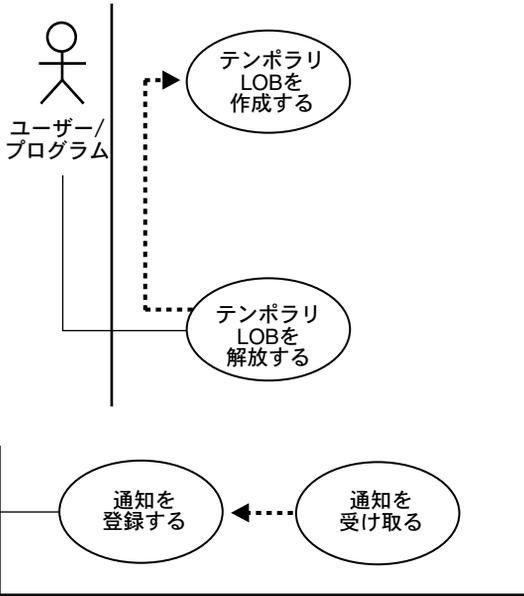


説明

この図では、注釈ボックスの表記でよく使用されるものを2つ示します。

- (a) 代替名を表す場合、この例では、「ユーザー・スキーマ内の伝播スケジュールを選択する」というアクションは、`USER_QUEUE_SCHEDULES` というビューによって表されます。
- (b) 「属性名をリストする」というアクションには、ユーザーに対する注釈が付けられています。注釈には、伝播スケジュール属性をすべてリストしない場合は、属性を少なくとも1つリストする必要があることが示されています。

図形要素



説明

ユースケース図の破線の矢印は、依存性を示します。この例では、

- テンポラリ LOB を解放するには、まず
- テンポラリ LOB を作成する必要があることを示しています。

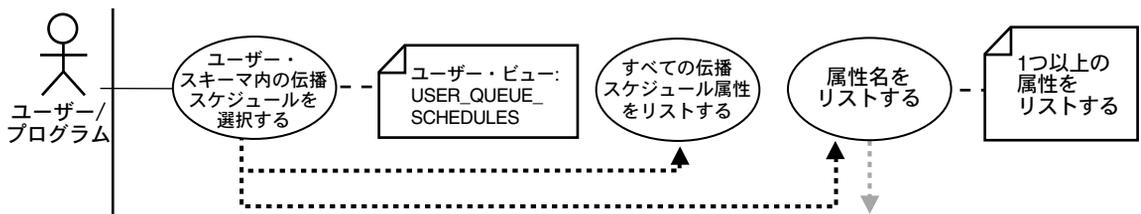
矢印の先は、最初に行う必要がある操作を示すということを理解しておく必要があります。

ユースケースとそのサブオペレーションは複雑な関係でリンクする可能性があります。

このコールバックの例では、

- 通知を登録する
 - 通知を受け取る
- 必要があります。

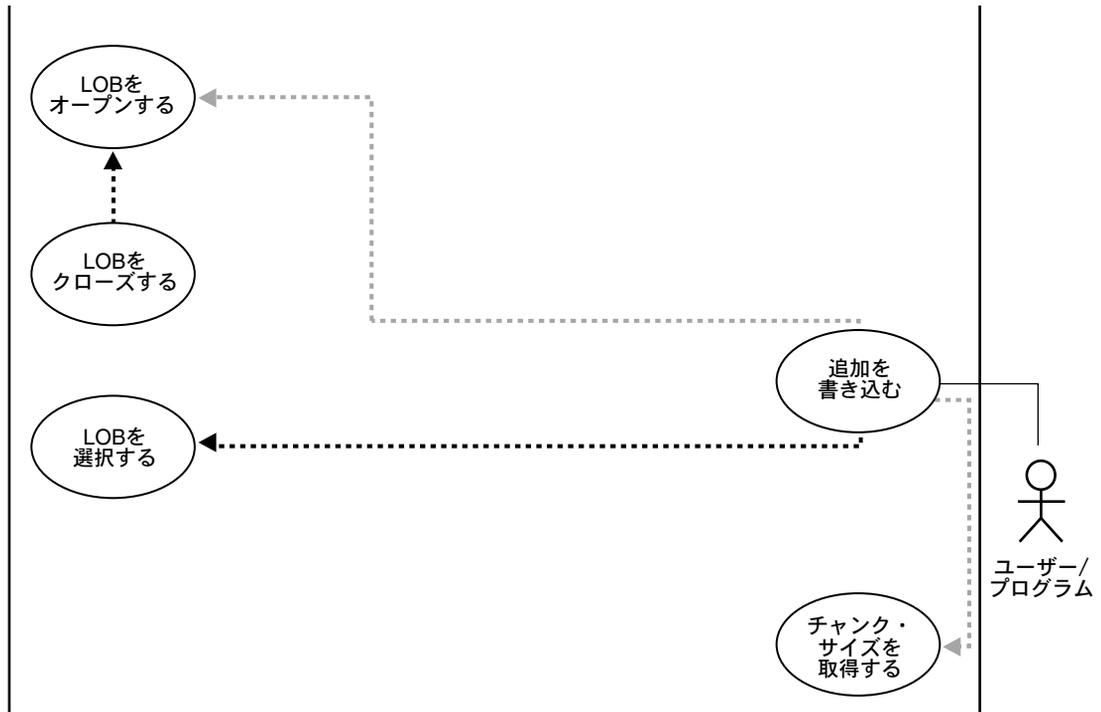
図形要素



説明

この例では、OR 条件の分岐パスが示されています。ビューを起動する際に、すべての属性のリストを選択するか、1つ以上の属性を表示するかを選択できます。ユーザーがどの属性を参照可能にするかを指定できるということが、灰色の矢印によって示されています。

図形要素



説明

線の付いたオペレーションがすべて必須というわけではありません。黒の破線および矢印は、ユースケースを完了するにはその矢印の宛先のオペレーションを実行する必要があることを示しますが、オプションのアクションは灰色の破線および矢印で示されています。

この例では、LOB に対して WRITEAPPEND を実行するには、まず「LOB を選択する」必要があります。

これを支援するオペレーションとして、「LOB をオープンする」または「チャンク・サイズを取得する」（あるいはその両方）を選択することもできます。

ただし、「LOB をオープンする」場合は、後で「LOB をクローズする」必要があります。

図形要素**説明**

内部テンポラリLOB (1/2)

次ページへ続く

ユースケース・モデル図は、内部テンポラリLOBなどの特定ドメイン内のユースケースをすべて要約したものです。この図は1ページに収まらないほど複雑になることがあります。

その場合は、図を2つの部分に分割します。この分割は順序を意味するものではないため注意してください。

場合によっては、単にページが長くなりすぎるという理由で図を分割することがあります。このようなときのために、このマーカーが用意されています。

1

概要

この章の内容は次のとおりです。

- LOB を使用する理由
 - 非構造化データ
 - LOB データ型によるインターネット・アプリケーションのサポート
 - LONG を使用しない理由
 - LOB によるセマンティクスの制御
 - LOB による interMedia の使用
- LOB のデモ・ディレクトリ
 - デモ・ディレクトリの位置
- 互換性および移行の問題
- このマニュアルでの Multimedia_Tab を使用した例

LOB を使用する理由

アプリケーションの発展により、ますます豊富なセマンティクスを扱うようになったため、様々な種類のデータ（単純な構造化データ、複雑な構造化データ、半構造化データ、非構造化データなど）を処理する必要性が増大しています。伝統的に、リレーショナル・モデルは単純な構造化データ（単純な表などのデータ）の処理を効率的に行ってきました。Oracle では、アプリケーションが複雑な構造化データ（コレクション、参照、ユーザー定義型など）を処理できるように、オブジェクト・リレーショナル機能が追加されています。このキューイング・テクノロジーによって、メッセージおよびその他の半構造化データを処理します。LOB は、非構造化データをサポートするように設計されています。

非構造化データ

標準コンポーネントに分解されない非構造化データ

非構造化データが標準コンポーネントに分解されることはありません。従業員に関するデータは、名前（文字列）、ID（数値）、給与などに構造化できます。ただし、写真をとりあげてみると、そのデータは実際は 0（ゼロ）および 1 の長いストリームで構成されていることがわかります。この 0（ゼロ）および 1 は、ディスプレイでその写真が見られるように、ピクセルをオンまたはオフに切り替えるために使用されますが、データベース記憶域に関しては、それより細かい構造に分解されることはありません。

サイズが大きい非構造化データ

テキスト、図形画像、静止ビデオ・クリップ、フル・モーション・ビデオ、サウンドウェーブ・フォームなどの非構造化データは、サイズが大きくなる傾向があります。通常、従業員記録は数百バイトですが、わずかなマルチメディア・データが、その何千倍もの大きさになる場合があります。

データベースからのアクセスが必要なシステム・ファイルの非構造化データ

オペレーティング・システム・ファイルにマルチメディア・データが存在する場合があります。このようなマルチメディア・データには、データベースからのアクセスが適しています。

LOB データ型によるインターネット・アプリケーションのサポート

インターネットおよび内容が豊富なアプリケーションの普及に伴い、データベースは、次のことを実現するためのデータ型のサポートが不可欠になっています。

- 非構造化データの格納
- 大量の非構造化データの最適化
- データベースの内外で、大量の非構造化データにアクセスするための一定の方法の提供

サポートされる 2 種類の LOB

Oracle8i では、次の 2 種類の LOB をサポートしています。

- 表の行内に、または別の表領域（BLOB、CLOB および NCLOB）のいずれかでデータベースに格納される LOB
- オペレーティング・システム・ファイル（BFILE など）として格納される LOB

LONG を使用しない理由

Oracle7 では、大量の非構造化データを格納するほとんどのアプリケーションが、LONG または LONG RAW のデータ型を使用していました。

Oracle8i での LOB データ型のサポートは、次の点で、Oracle7 での LONG および LONG RAW のサポートよりも優れています。

- **LOB 容量:** Oracle8i を使用すると、最大 4GB のデータを LOB に格納できます。これは、LONG および LONG RAW のデータ型が格納できる 2GB のデータの 2 倍です。
- **表あたりの LOB 列数:** Oracle8i の表は、複数の LOB 列を持つことができます。同一の表内の各 LOB 列は、異なる型を指定できます。Oracle7 リリース 7.3 では、表は単一の LONG または LONG RAW の列に制限されています。
- **ランダムなピース単位のアクセス:** LOB はデータへのランダム・アクセスをサポートしますが、LONG は逐次アクセスのみサポートします。さらに、サーバー側からクライアントへ LOB を送信する処理速度を短縮するために、LOB をチャンクに分割し、クライアントへの 1 回のラウンドトリップで送信することができます。

LOB 型の列

LOB（BLOB、CLOB、NCLOB または BFILE）型の列には、「ロケータ」といわれる値または参照を格納できます。これは行外または外部ファイルに格納されたラージ・オブジェクトの位置を指定します。

LOB 型の列は、ロケータを格納するだけではない LOB 型の列では、LOB ロケータは行内に格納されます。ただし、ユーザー定義の SQL データ定義言語（DDL）によっては、Oracle8i は、4KB より小さい LOB を表の行内に格納できます。LOB が約 4KB より大きくなると、Oracle8i は、LOB を表から別のセグメントに移動し、さらに別の表領域に移動します。このため、Oracle8i は、LOB ロケータのみではなく、LOB データを行内に格納します。

BLOB、CLOB および NCLOB のデータは、データベース内で行外に格納されます。BFILE データは、データベース外のオペレーティング・システム・ファイルに格納されます。Oracle8i は、LOB へのアクセスおよびその操作のためのプログラム・インタフェースおよび PL/SQL サポートを提供します。

LOB によるセマンティクスの制御

SQL に関しては、Oracle8i LOB に常駐するデータは、不透明で問合せはできません。LOB の部分にアクセスまたは操作するファンクション（オブジェクト型メソッドを含む）を書き込むことはできます。このように、ラージ・オブジェクトに常駐するデータの構造およびセマンティクスは、アプリケーション開発者が指定できます。

たとえば、従業員の履歴書を LOB 文字として格納するとします。このような場合、履歴書構造のアプリケーション固有の情報を利用して、従業員が以前勤めていた会社名を取り出すなどの、履歴書を解析するルーチンの書込みができます。また、*interMedia Text*（コンテキスト）を使用して、履歴書のキーワードを索引参照することもできます。

LOB による *interMedia* の使用

LOB によって、マルチメディア・データを格納するインフラストラクチャがデータベース内に提供されますが、Oracle8i でも、ごく一般的に使用されるマルチメディア型のための追加機能が開発者に提供されています。マルチメディア型には、テキスト、イメージ、ロケータおよびオーディオまたはビデオ・データがあります。

Oracle8i では、データ・カートリッジともいわれる特定のデータ型のコレクションである *interMedia* をバンドルしています。テキスト・データ、空間位置、イメージ、オーディオおよびビデオ・データはすべてサポートされます。ユーザーは、豊富な SQL 問合せを使用してこれらの型のオブジェクトへアクセスし、その内容を操作（イメージの切捨て）し、内容の読み込み、書き込み、およびあるフォーマットから別のフォーマットへデータの変換を行うことができます。

また、データ・カートリッジは Oracle8i のインフラストラクチャを使用して、オブジェクト型、メソッド、およびデータベースにこの特定の型のデータを表すために必要な LOB を定義します。

Oracle8i のデータ・カートリッジは、事前に定義された一連のオブジェクトおよび操作を提供します。これによって、前述の型を使用したアプリケーション開発が容易になります。詳細は、次の URL を参照してください。

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

LOB のデモ・ディレクトリ

LOB のデモンストレーション・サンプル・スクリプトは、現在、このマニュアルの第 9 章、第 10 章および第 11 章に記載されています。このスクリプトの大半はテスト済で、正常に実行することが確認されています。第 8 章で説明されているサンプル・マルチメディア・スキーマの構文については、次の項目を参照してください。

- 9-8 ページの「1 つ以上の LOB 列を含む表の作成」
- 10-14 ページの「テンポラリ LOB の作成」
- 11-15 ページの「1 つ以上の BFILE 列を含む表の作成」

前述のスキーマに対する SQL セットアップ構文は、次のファイルの Oracle8i のデモ・ディレクトリでも提供されています。

- lobdemo.sql
- adloci.sql

デモ・ディレクトリの位置

デモ・スクリプトは、Oracle8i インストールで使用可能です。プログラムの位置、名前および可用性は、プラットフォームによって異なります。プラットフォーム固有のドキュメントを参照してください。UNIX および Windows NT に関しては次のとおりです。

- **UNIX:** UNIX ワークステーションでは、プログラムは ORACLE_HOME/rdbms/demo ディレクトリにインストールされています。
- **Windows NT:** Windows NT マシンでは、プログラムは ORACLE_HOME¥Oci¥Samples ディレクトリ (OCI コード例の場合) にあります。

互換性および移行の問題

次の LOB に関連する互換性および移行の問題の詳細は、『Oracle8i 移行ガイド』を参照してください。次の章および項は、『Oracle8i 移行ガイド リリース 8.1』の章および項です。

- 第 9 章の「互換性および相互運用性の問題」の項の「データ型」の「CLOB および NCLOB の可変幅キャラクタ・セット」を参照してください。

- 定義済の CACHE READ を伴うダウングレードについては、第 13 章の「非互換性の削除」の項の「データ型」にある「LOB の CACHE READ 指定の使用の中断」を参照してください。
- ダウングレード - パーティション表からの LOB 列の削除については、第 13 章の「非互換性の削除」の項の「データ型」にある「パーティション表からの LOB 列の削除」を参照してください。
- ダウングレード - 索引構成表内の LOB および VARRAY については、第 13 章の「非互換性の削除」の項の「スキーマ・オブジェクト」にある「索引構成表の LOB および VARRAY の使用の中断」を参照してください。
- ダウングレード - CLOB または NCLOB の可変幅キャラクタ・セットについては、第 13 章の「非互換性の削除」の項の「データ型」にある「可変幅のキャラクタ・セットのあるデータベースの表から CLOB および NCLOB を削除」を参照してください。

このマニュアルでの Multimedia_Tab を使用した例

マルチメディア・データは、教育、エンターテイメント、セキュリティおよびその他の産業用の Web ページ、CD-ROM、フィルムおよびテレビでの使用が増大しています。通常、マルチメディア・データは大きく、オーディオ、ビデオ、スクリプト、履歴書、図形、写真などで構成されます。このデータの大半が構造化されていません。

LOB は、大量の非構造化データを処理するように設計されています。「非構造化データ」については、この章の最初で説明しています。

サンプル・アプリケーションについては、Multimedia_tab といわれるマルチメディア表に基づいて、第 8 章「サンプル・アプリケーション」で詳しく説明します。このマニュアルに記載されているすべての例は、Multimedia_tab 表に基づいています。この表は、必要に応じて変更または拡張されますが、これについては該当する項で説明します。

参照

LOB の詳細は、次の URL を参照してください。

<http://technet.oracle.com/products>

基本コンポーネント

この章の内容は次のとおりです。

- LOB データ型
 - 内部 LOB
 - 外部 LOB (BFILE)
 - 内部 LOB で使用するコピー・セマンティクスおよび外部 LOB で使用する参照セマンティクス
 - 可変幅文字データ
- LOB ロケータ
 - LOB の値およびロケータ
 - LOB ロケータの操作
- LOB を含む表の作成
 - 内部 LOB を NULL または空として初期化
 - 内部 LOB 列を値に初期化
 - 外部 LOB を NULL またはファイル名に初期化

注意: この章で示す例は、第 8 章「サンプル・アプリケーション」で説明する Multimedia_tab スキーマおよび Multimedia_tab 表を基にしています。

LOB データ型

Oracle8i では、LOB を、データベースに関連する格納場所によって、**内部 LOB** および**外部 LOB** の 2 種類に分けています。外部 LOB は、**BFILE** (バイナリ・ファイル) とも呼ばれます。このマニュアルで LOB 作業について説明する場合、特に指定がなければ、その内容は内部および外部両方の LOB に適用できます。

内部 LOB には、さらに**永続 LOB** および**テンポラリ LOB** の 2 種類があります。

内部 LOB

内部 LOB は、その名前のとおりデータベース表領域内に格納され、領域を最適化し、効率的なアクセスを提供します。内部 LOB はコピー・セマンティクスを使用し、サーバーのトランザクション・モデルに使用されます。内部 LOB は、トランザクションまたはメディア障害が発生してもリカバリ可能です。また、内部 LOB 値の変更は、コミットまたはロールバックできます。すなわち、データベース・オブジェクトの使用に関するすべての ACID プロパティは、内部 LOB の使用にも適用されます。

内部 LOB データ型

内部 LOB のインスタンスを定義するには、次の 3 つの SQL データ型があります。

- **BLOB**: 非構造化バイナリ (ロー)・データで構成される値を持つ LOB
- **CLOB**: Oracle8i データベース用に定義されたデータベース・キャラクタ・セットに対応する文字データで構成される値を持つ LOB
- **NCLOB**: Oracle8i データベース用に定義された各国語キャラクタ・セットに対応する文字データで構成される値を持つ LOB

外部 LOB (BFILE)

外部 LOB (BFILE) は、データベース表領域外のオペレーティング・システム・ファイル内に格納される大規模なバイナリ・データ・オブジェクトです。このオペレーティング・システム・ファイルでは、参照セマンティクスを使用します。BFILE は、ハード・ディスクなどの既存の 2 次記憶装置のみでなく、CD-ROM、フォト CD、DVD などの 3 次ブロック記憶装置に格納することもできます。

BFILE データ型は、データベース・サーバーのファイル・システム上に置かれた大規模なファイルに対する、読取り専用のバイト・ストリーム・アクセスを可能にします。

Oracle Server では、基礎となるサーバーのオペレーティング・システムが、これらのオペレーティング・システム・ファイルへのストリーム・モード・アクセスをサポートしている場合、BFILE にアクセスできます。

注意:

- 外部 LOB は、トランザクションでは関係しません。LOB の整合性および耐久性は、オペレーティング・システムで管理されるファイル・システムによってサポートされる必要があります。
 - 単一の BFILE を複数のデバイスに格納することはできません。たとえば、複数のディスク・アレイにまたがってストライプ化することはできません。
-
-

外部 LOB データ型

外部 SQL の LOB のインスタンスを宣言するためのデータ型は次の 1 種類です。

- **BFILE**: バイナリ (ロー)・データで構成され、データベース表領域外のサーバー側オペレーティング・システム・ファイル内に格納される値を持つ LOB

内部 LOB で使用するコピー・セマンティクスおよび外部 LOB で使用する参照セマンティクス

- コピー・セマンティクス: LOB ロケータと値の両方がコピーされます。
- 参照セマンティクス: LOB ロケータのみコピーされます。

コピー・セマンティクス

内部 LOB (BLOB、CLOB、NCLOB) は、永続 LOB かテンポラリ LOB かにかかわらず、コピー・セマンティクスを使用します。

LOB を、同じ表内の他の行の LOB とともに挿入または更新する場合、各行が異なる LOB 値のコピーを持つように、その LOB 値はコピーされます。

表内の行の LOB が、他の行または同じ行ではあるが他の列にある別の LOB にコピーされた場合、LOB ロケータのみでなく、実際の LOB 値もコピーされるように、内部 LOB はコピー・セマンティクスを持ちます。この場合、2 つの異なる LOB ロケータおよび 2 つの LOB 値のコピーが存在することになります。

参照セマンティクス

外部 LOB (BFILE) は、参照セマンティクスを使用します。表内の行の BFILE が別の BFILE にコピーされた場合、実際の BFILE データではなく、BFILE ロケータのみがコピーされます。つまり、実際のオペレーティング・システム・ファイルはコピーされません。

可変幅文字データ

次の表を作成できます。

- 可変幅 CHAR/NCHAR データベース・キャラクタ・セットを使用する場合でも、CLOB/NCLOB 列を含む表
- 可変幅 CHAR データベース・キャラクタ・セットを使用しているかどうかにかかわらず、CLOB 属性を持つ型を含む表

次の表は作成できません。

- オブジェクト型の属性としての NCLOB を含む表

可変幅キャラクタ・セット用の 2 バイトの Unicode を使用して格納される CLOB / NCLOB 値

データベース CHAR/NCHAR キャラクタ・セットが可変幅の場合、CLOB/NCLOB 値は、固定幅 2 バイトの Unicode キャラクタ・セットを使用してデータベースに格納されます。

- **データの挿入** データを CLOB に挿入するときは、データ入力は可変幅キャラクタ・セットにすることができます。この可変幅文字データは、暗黙的に Unicode に変換されてから、データベースに格納されます。
- **LOB の読み込み** 逆に、LOB 値の読み込み時には、格納された Unicode 値が、クライアントまたはサーバーのいずれかでユーザーが要求したキャラクタ・セット（可変幅の場合が多い）に変換されます。

Oracle では、Unicode への変換および Unicode からの変換もすべて、暗黙的に行われることに注意してください。

NCLOB は、固定幅データを格納します。

CLOB には、すべての LOB 操作（読み込み、書き込み、切捨て、消去、比較など）を実行できます。CLOB へのアクセスが可能なプログラム環境はすべて、CHAR/NCHAR キャラクタ・セットが可変幅であるデータベース内の CLOB で機能します。これには、SQL、PL/SQL、OCI、PRO*C、DBMS_LOB などが含まれます。

可変幅の CLOB データには、パラメータを文字かバイトのどちらで指定するかを考慮する必要があります。

参照：

[第 3 章「LOB プログラム環境」](#) の次の項を参照してください。

- [3-7 ページの「オフセット・パラメータおよび量パラメータ：固定幅と可変幅、DBMS_LOB パッケージに対する文字またはバイト」](#)
- [3-11 ページの「オフセット・パラメータおよび量パラメータ：固定幅と可変幅、OCI での文字またはバイト」](#)

LOB ロケータ

LOB の値およびロケータ

LOB 値の行内記憶域

LOB に格納されたデータを LOB の値といいます。内部 LOB の値は、他の行データとともに行内に格納できる場合とできない場合があります。DISABLE STORAGE IN ROW を設定しない場合、内部 LOB の値が約 4000 バイトより小さいと、その値は行内に格納されます。それ以外の場合は、行外に格納されます。LOB はオブジェクトとしては大きくなる傾向があるため、アプリケーションに大小の LOB が混在する場合にのみ、行内記憶域が効果的です。

7-11 ページの「[ENABLE | DISABLE STORAGE IN ROW](#)」で記述しているとおり、LOB 値は、一度でも約 4000 バイトを超えると自動的に行の外に移されます。

LOB ロケータ

内部 LOB の値が格納されている場所にかかわらず、ロケータは行内に格納されます。LOB ロケータは、LOB 値の実際の位置へのポインタとして考えることができます。BFILE ロケータが外部 LOB を示すロケータであるのに対して、LOB ロケータは内部 LOB を示すロケータです。単にロケータという場合は、LOB ロケータと BFILE ロケータの両方を指します。

- **内部 LOB ロケータ** 内部 LOB については、データベース表領域に格納される LOB の値に対するロケータが LOB 列に格納されます。指定された行におけるそれぞれの LOB 列 / 属性には、個別の LOB ロケータ、およびデータベース表領域内に格納された個別の LOB 値のコピーがあります。
- **外部 LOB ロケータ** 外部 LOB (BFILE) については、外部オペレーティング・システム・ファイルに対する BFILE ロケータが LOB 列に格納されます。指定された行におけるそれぞれの BFILE 列 / 属性には、個別の BFILE ロケータがあります。ただし、2つの異なる行では、同じオペレーティング・システム・ファイルを指す BFILE ロケータを含むことができます。

LOB ロケータの操作

ロケータを含む LOB 列 / 属性の設定

- **内部 LOB** : 6 つのプログラム環境インタフェース¹ (PL/SQL、OCI、Pro*C、Pro*COBOL、Visual Basic または Java) のいずれかを介してデータを内部 LOB に書き込む前に、LOB 列 / 属性を NULL 以外にする必要があります。つまり、ロケータを含める必要があります。これは、BLOB に対しては `EMPTY_BLOB()` 関数、CLOB および NCLOB に対しては `EMPTY_CLOB()` 関数を使用して、INSERT/UPDATE 文で内部 LOB を空に初期化することによって行います。

参照 : 9-23 ページの「[EMPTY_CLOB\(\) または EMPTY_BLOB\(\) を使用した LOB 値の挿入](#)」を参照してください。

- **外部 LOB** : 6 つのプログラム環境インタフェースのいずれかを介して外部 LOB (BFILE) へのアクセスを開始する前に、BFILE 列 / 属性を NULL 以外にする必要があります。BFILENAME() 関数を使用して、外部オペレーティング・システム・ファイルを指す BFILE 列を初期化できます。

参照 : 11-24 ページの「[BFILENAME\(\) を使用した行の挿入](#)」を参照してください。

`EMPTY_BLOB()` または `EMPTY_CLOB()` 関数の単独の起動では例外は発生しません。ただし、空に設定された LOB ロケータを使用して、PL/SQL の `DBMS_LOB` または OCI ルーチンで LOB 値をアクセスまたは操作すると、例外が発生します。

空の LOB ロケータが有効となるのは、INSERT 文の VALUES 句や UPDATE 文の SET 句などです。

次の INSERT 文では、次のことを行います。

- story に「JFK interview」という文字列を移入します。
- flsub、frame および sound を空の値に設定します。
- photo を NULL に設定します。
- さらに、music を初期化して、「JFK_interview」ファイルを指すようにします。このファイルは論理ディレクトリ `AUDIO_DIR` の下にあります (『Oracle8i リファレンス・マニュアル』の CREATE DIRECTORY 文を参照)。

文字列は、インスタンス用のデフォルトのキャラクタ・セットを使用して挿入されます。

¹ **注意** : NULL が含まれていても LOB 属性である限り、データを LOB 列に移入するために SQL を使用できます。ただし、NULL の LOB には、6 つのプログラム環境インタフェースのいずれも使用できません。

Multimedia_tab 表の定義は、第 8 章「サンプル・アプリケーション」を参照してください。

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

同様に、Multimedia_tab 中の Map_typ 列の LOB 属性は NULL に初期化するか、次に示すように空に設定することができます。リテラルでは、LOB オブジェクト属性を初期化できないことに注意してください。

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

ロケータによる LOB へのアクセス

LOB の SELECT LOB に対して SELECT を実行すると、LOB 値のかわりにロケータが戻されます。次の PL/SQL の断片部分では、story の LOB ロケータを選択し、プログラム・ブロック内で定義された PL/SQL ロケータ変数 Image1 内に、それを配置します。LOB 値の操作に PL/SQL の DBMS_LOB ファンクションを使用する場合は、ロケータを使用して LOB を参照します。

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

OCI の場合には、ロケータは LOB 値の操作に使用されるロケータ・ポインタにマップされます。OCI の LOB インタフェースについては、第 3 章「LOB プログラム環境」および『Oracle8i コール・インタフェース・プログラマーズ・ガイド』を参照してください。

LOB ロケータ、トランザクション境界の使用、および読取り一貫性のあるロケータについては、第 5 章「高度なトピック」を参照してください。

LOB を含む表の作成

LOB を含む表を作成する際は、次の項に示すガイドラインを使用してください。

- 内部 LOB を NULL または空として初期化
- 内部 LOB 列を値に初期化
- 外部 LOB を NULL またはファイル名に初期化
- 表領域および記憶特性の定義については、第 7 章「モデリングおよび設計」、「内部 LOB に対する表領域および記憶特性の定義」を参照してください。

内部 LOB を NULL または空として初期化

内部 LOB（表内の LOB 列、またはユーザーが定義したオブジェクト型の LOB 属性）を、NULL または空に設定できます。

- 内部 LOB を NULL に設定 : NULL に設定された LOB にロケータはありません。NULL 値はロケータではなく、表内の行に格納されます。これは、他のすべてのデータ型と同じプロセスです。
- 内部 LOB を空に設定 : 逆に、表に格納された空の LOB は長さ 0（ゼロ）の LOB で、ロケータを持っています。したがって、空の LOB 列 / 属性から SELECT すると、OCI または PL/SQL (DBMS_LOB) など、6 つのプログラム環境のいずれかを介して、LOB にデータを移入するためのロケータを入手できます。詳細は、第 3 章「LOB プログラム環境」を参照してください。

これらのオプションについては、次に詳しく説明します。

次に説明するとおり、外部 LOB (BFILE) は、NULL またはファイル名に初期化できます。

内部 LOB を NULL に設定

INSERT するときに LOB データがない場合、または後で次のような SELECT 文を発行する場合（あるいはその両方の場合）、行を挿入するときに内部 LOB の値を NULL に設定する必要があります。

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

この文により、録音済のナレーション・セグメントをすべて検索できます。また、

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

この文により、録音する必要があるセグメントをチェックできます。

OCI または DBMS_LOB 関数は NULL LOB ではコールできない ただし、この方法には、後で SQL UPDATE 文を発行して、内部 LOB については NULL の LOB 列を EMPTY_BLOB(), EMPTY_CLOB() または値 ('Denzel Washington' など) に、外部 LOB についてはファイル名などに設定し直す必要があるというデメリットがあります。

つまり、NULL に設定された LOB では、6つのプログラム環境 (OCI、PL/SQL (DBMS_LOB) など) の関数はいずれもコールできません。これらの関数はロケータがある場合にのみ有効です。LOB 列が NULL の場合は、行にロケータがないため無効となります。

内部 LOB を空に設定

内部 LOB 列を NULL に設定しない場合、INSERT 文で関数 EMPTY_BLOB() または EMPTY_CLOB() を使用して LOB 値を空にできます。

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

RETURNING 句を使用してその後すぐに OCI または PL/SQL DBMS_LOB ファンクションをコールして LOB にデータを移入すると、(次の SELECT に必要な、処理の往復が避けられるため) 効果的です。

```
DECLARE
    Lob_loc BLOB;
BEGIN
    INSERT INTO a_table VALUES (EMPTY_BLOB()) RETURNING blob_col INTO Lob_loc;
    /* ロケータ Lob_loc を使用して、BLOB にデータを移入します。 */
END;
```

Multimedia_tab 表を使用した例

次の INSERT 文を使用して、Multimedia_tab 中の LOB を初期化できます。

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

これによって、story、flsub、frame および sound の値が空に設定され、photo および music が NULL に設定されます。

内部 LOB 列を値に初期化

また、LOB 列は、値に初期化することもできます (LOB 属性はできません)。つまり、内部 LOB 属性は、LOB 属性が NULL または空以外の値には初期化できないという点で、LOB 列とは異なります。

LOB 列は、4KB より大きいデータを含む値に初期化できることに注意してください。詳細は、[第7章「モデリングおよび設計」](#)を参照してください。

外部 LOB を NULL またはファイル名に初期化

外部 LOB (BFILE) は、BFILENAME() 関数を介して NULL またはファイル名に初期化できます。

詳細は、11-5 ページの「[BFILE ロケータの初期化](#)」を参照してください。

LOB プログラム環境

この章の内容は次のとおりです。

- LOB で操作される 6 つのプログラム環境
- 6 つの LOB インタフェースの比較
- LOB の作業に対する PL/SQL (DBMS_LOB パッケージ) の使用
- C (OCI) を使用した LOB の作業
- C/C++ (Pro*C) を使用した LOB の作業
- COBOL (Pro*COBOL) を使用した LOB の作業
- Visual Basic (Oracle Objects for OLE (OO4O)) を使用した LOB の作業
- Java (JDBC) を使用した LOB の作業

注意: この章で示す例は、第 8 章「サンプル・アプリケーション」で説明するマルチメディア・スキーマおよび Multimedia_tab 表を基にしています。

LOBで操作される6つのプログラム環境

Oracle8iでは、LOBでの操作用に6つの環境（言語）を提供しています。表3-1「LOBの6つのプログラム環境」に、これらを示します。

表 3-1 LOB の 6 つのプログラム環境

言語	プリコンパイラまたは インタフェース・プロ グラム	構文についての参照先	この章にある参照先
PL/SQL	DBMS_LOB パッケージ	『Oracle8i PL/SQL パッケージ・プロシージャ・リファレンス』	3-6 ページの「LOBの作業に対する PL/SQL (DBMS_LOB パッケージ) の使用」
C	Oracle Call Interface (OCI)	『Oracle8i コール・インタフェース・プログラマーズ・ガイド』	3-11 ページの「C (OCI) を使用した LOB の作業」
C++	Pro*C/C++ プリコンパイラ	『Oracle8i Pro*C/C++ プリコンパイラ・プログラマーズ・ガイド』	3-21 ページの「C/C++ (Pro*C) を使用した LOB の作業」
COBOL	Pro*COBOL プリコンパイラ	『Oracle8i Pro*COBOL プリコンパイラ・プログラマーズ・ガイド』	3-25 ページの「COBOL (Pro*COBOL) を使用した LOB の作業」
Visual Basic	Oracle Objects For OLE (OO4O)	Oracle Objects for OLE (OO4O) は、Oracle8i Client for Windows NT に含まれる Windows ベースの製品です。 この製品に対するマニュアルはありません。オンライン・ヘルプのみです。オンライン・ヘルプは、Oracle8i インストールの「Application Development」サブメニューから利用できます。	3-29 ページの「Visual Basic (Oracle Objects for OLE (OO4O)) を使用した LOB の作業」
Java	JDBC アプリケーション・プログラム・インタフェース (API)	『Oracle8i SQLJ 開発者ガイドおよびリファレンス』および『Oracle8i JDBC 開発者ガイドおよびリファレンス』	3-36 ページの「Java (JDBC) を使用した LOB の作業」

6 つの LOB インタフェースの比較

表 3-2 「LOB を使用するときのインタフェースの比較」に、LOB で操作するために使用する使用可能なファンクションおよびメソッドを示し、6 つの LOB インタフェースを比較します。

表 3-2 LOB を使用するときのインタフェースの比較

PL/SQL: DBMS_LOB (dbmslob.sql)	OCI (ociap.h)	Pro*C および Pro*COBOL	Visual Basic (OO4O)	Java (JDBC)
DBMS_LOB.COMPARE	利用不可	利用不可	ORALOB.Compare	DBMS_LOB.COMPARE を使用
DBMS_LOB.INSTR	利用不可	利用不可	ORALOB.Matchpos	position
DBMS_LOB.SUBSTR	利用不可	利用不可	利用不可	BLOB または BFILE 用の getBytes CLOB 用の getSubString
DBMS_LOB.APPEND	OCILobAppend	APPEND	ORALOB.Append	length の後で putBytes または PutString を使用
利用不可 (PL/SQL 割当て 演算子を使用)	OCILobAssign	ASSIGN	ORALOB.Clone	利用不可 (等号を使用)
利用不可	OCILobCharSetForm	利用不可	利用不可	利用不可
利用不可	OCILobCharSetId	利用不可	利用不可	利用不可
DBMS_LOB.CLOSE	OCILobClose	CLOSE	利用不可	DBMS_LOB.CLOSE を使用
DBMS_LOB.COPY	OCILobCopy	COPY	ORALOB.Copy	read および write を使用
利用不可	OCILobDisableBuffering	DISABLE BUFFERING	ORALOB. DisableBuffering	利用不可
利用不可	OCILobEnableBuffering	ENABLE BUFFERING	ORALOB. EnableBuffering	利用不可
DBMS_LOB.ERASE	OCILobErase	ERASE	ORALOB.Erase	DBMS_LOB.ERASE を使用
DBMS_LOB.FILECLOSE	OCILobFileClose	CLOSE	ORABFILE.Close	closeFile
DBMS_LOB.FILECLOSE ALL	OCILobFileCloseAll	FILE CLOSE ALL	ORABFILE.CloseAll	DBMS_LOB.FILECLOSE ALL を使用
DBMS_LOB.FILEEXISTS	OCILobFileExists	DESCRIBE [FILEEXISTS]	ORABFILE.Exist	fileExists

表 3-2 LOB を使用するときのインタフェースの比較 (続き)

PL/SQL: DBMS_LOB (dbmslob.sql)	OCI (ociap.h)	Pro*C および Pro*COBOL	Visual Basic (OO4O)	Java (JDBC)
DBMS_LOB.GETCHUNK SIZE	OCILobGetChunkSize	DESCRIBE [CHUNKSIZE]	利用不可	getChunkSize
DBMS_LOB.FILEGET NAME	OCILobFileGetName	DESCRIBE [DIRECTORY, FILENAME]	ORABFILE. DirectoryName ORABFILE. FileName	getDirAlias getName
DBMS_LOB.FILEISOPEN	OCILobFileIsOpen	DESCRIBE [ISOPEN]	ORABFILE.IsOpen	DBMS_LOB.ISOPEN を使 用
DBMS_LOB.FILEOPEN	OCILobFileOpen	OPEN	ORABFILE.Open	openFile
利用不可 (BFILENAME 演 算子を使用)	OCILobFileSetName	FILE SET	DirectoryName FileName	BFILENAME を使用
利用不可	OCILobFlushBuffer	FLUSH BUFFER	ORALOB.FlushBuffer	利用不可
DBMS_LOB.GETLENGTH	OCILobGetLength	DESCRIBE [LENGTH]	ORALOB.Size	length
利用不可	OCILobIsEqual	利用不可	利用不可	equals
DBMS_LOB.ISOPEN	OCILobIsOpen	DESCRIBE [ISOPEN]	ORALOB.IsOpen	DBMS_LOB.ISOPEN を使 用
DBMS_LOB.LOADFROM FILE	OCILobLoadFromFile	LOAD FROM FILE	ORALOB. CopyFromBfile	read の後で write を使用
利用不可 (常に初期化)	OCILobLocatorIsInit	利用不可	利用不可	利用不可
DBMS_LOB.OPEN	OCILobOpen	OPEN	ORALOB.open	DBMS_LOB.OPEN を使 用
DBMS_LOB.READ	OCILobRead	READ	ORALOB.Read	BLOB または BFILE: getBytes および getBinaryStream CLOB: getString および getSubString および getCharacterStream
DBMS_LOB.TRIM	OCILobTrim	TRIM	ORALOB.Trim	DBMS_LOB.TRIM を使用

表 3-2 LOB を使用するときのインタフェースの比較 (続き)

PL/SQL: DBMS_LOB (dbmslob.sql)	OCI (ociap.h)	Pro*C および Pro*COBOL	Visual Basic (OO4O)	Java (JDBC)
DBMS_LOB.WRITE	OCILobWrite	WRITEORA LOB.	ORALOB.Write	BLOB または BFILE: putBytes および getBinaryOutputStream CLOB: putString および getCharacterOutputStream
DBMS_LOB.WRITE APPEND	OCILobWriteAppend	WRITE APPEND	利用不可	length の後で putString または putBytes を使用
DBMS_LOB. CREATETEMPORARY	OCILobCreateTemporary		利用不可	
DBMS_LOB. FREETEMPORARY	OCILobFree Temporary		利用不可	
DBMS_LOB.ISTEMPORARY	OCILobIsTemporary		利用不可	
	OCILobLocatorAssign		利用不可	

次の項で、これらの各インタフェースを詳しく説明します。

LOB の作業に対する PL/SQL (DBMS_LOB パッケージ) の使用

PL/SQL DBMS_LOB パッケージは、次の操作のために使用できます。

- **内部永続 LOB およびテンポラリ LOB:** 全体またはピース単位の読み込みおよび変更操作
- **BFILE:** 読み込み操作

参照: パラメータ、パラメータ・タイプ、戻り値およびサンプル・コードの詳細は、『Oracle8i PL/SQL パッケージ・プロシージャ・リファレンス』を参照してください。

DBMS_LOB ルーチン起動前の LOB ロケータの提供

次に詳しく説明するように、DBMS_LOB ルーチンは LOB ロケータに基づいて機能します。DBMS_LOB ルーチンを正常に完了させるには、データベース表領域または外部ファイル・システムですでに存在する LOB を表す入力ロケータを用意してから、ルーチンを起動する必要があります。

- **内部 LOB:** SQL を使用して LOB 列を含む表を定義します。その後、SQL を使用してこれらの LOB 列のロケータを初期化または移入できます。
- **外部 LOB:** アクセスする外部 LOB を含む有効物理ディレクトリにマップする、DIRECTORY オブジェクトを定義します。これらのファイルは存在し、処理のための Oracle Server に対する読み込み権限を持っている必要があります。オペレーティング・システムがパス名を大 / 小文字区別している場合には、正しい大 / 小文字でディレクトリを指定してください。詳細は、11-5 ページの「**ディレクトリ・オブジェクト**」を参照してください。

一度 LOB を定義して作成すると、SELECT を実行して LOB ロケータをローカルの PL/SQL LOB 変数に割り当てることができます。また、この変数を、LOB 値にアクセスするための DBMS_LOB への入力パラメータとして使用できます。

次の項では、各 DBMS_LOB ルーチンで提供されている例によって、前述の操作を解説します。

DBMS_LOB ルーチンをコールできないクライアント PL/SQL プロシージャ

クライアント側の PL/SQL プロシージャは、DBMS_LOB パッケージ・ルーチンをコールできません。

ただし、サーバー側の PL/SQL プロシージャまたは Pro*C/C++ 中の無名ブロックを使用して、DBMS_LOB パッケージ・ルーチンをコールすることはできます。

オフセット・パラメータおよび量パラメータ： 固定幅と可変幅、DBMS_LOB パッケージに対する文字またはバイト

DBMS_LOB パッケージ（固定幅および可変幅のキャラクタ・セット両方）について、次の規則が適用されます。

- CLOB および NCLOB: オフセット・パラメータおよび量パラメータは、常に文字で示されます。
- BLOB および BFILES: オフセット・パラメータおよび量パラメータは、常にバイトで示されます。

DBMS_LOB.LOADFROMFILE: BFILE より小さいサイズの量パラメータの指定

DBMS_LOB.LOADFROMFILE を使用する場合、量パラメータは BFILE より大きいサイズに指定することはできません。

DBMS_LOB.READ: データ・サイズより大きい量パラメータ

DBMS_LOB.READ を使用する場合、量パラメータはデータより大きいサイズに指定できません。PL/SQL では、量はバッファのサイズ以下である必要があり、バッファ・サイズは 32KB に制限されます。

PL/SQL: BLOB、CLOB、NCLOB および BFILE を操作するファンクションおよびプロシージャ

BLOB、CLOB、NCLOB および BFILE で操作される PL/SQL ファンクションおよびプロシージャは、次のとおりです。

- 内部 LOB の値の変更は、[表 3-3](#) を参照してください。
- LOB 値の読み込みまたはテストは、[表 3-4](#) を参照してください。
- テンポラリ LOB の作成、解放またはチェックは、[表 3-5](#) を参照してください。
- 外部 LOB (BFILE) における読み込み専用ファンクションは、[表 3-6](#) を参照してください。
- LOB のオープン、クローズ、または LOB がオープンしているかどうかのチェックは、[表 3-7](#) を参照してください。

PL/SQL: BLOB、CLOB および NCLOB の値を変更するファンクション/プロシージャ

表 3-3 PL/SQL: BLOB、CLOB および NCLOB の値を変更する DBMS_LOB プロシージャ

ファンクション/プロシージャ	説明
APPEND ()	LOB 値を別の LOB に追加します。
COPY ()	LOB の全体または一部を別の LOB にコピーします。
ERASE ()	指定のオフセットから開始して、LOB の一部を消去します。
LOADFROMFILE ()	BFILE データを内部 LOB にロードします。
TRIM ()	指定された長さまで LOB 値を切り捨てます。
WRITE ()	指定されたオフセットから LOB にデータを書き込みます。
WRITEAPPEND ()	データを LOB の最後に書き込みます。

PL/SQL: 内部および外部 LOB 値の読み込みまたはテストを行うファンクション/プロシージャ

表 3-4 PL/SQL: 内部および外部 LOB の値の読み込みまたはテストを行う DBMS_LOB プロシージャ

ファンクション/プロシージャ	説明
COMPARE ()	2 つの LOB の値を比較します。
GETCHUNKSIZE ()	読み込みおよび書き込み用にチャンク・サイズを取得します。これは内部 LOB にも適用され、外部 LOB (BFILE) には適用されません。
GETLENGTH ()	LOB 値の長さを取得します。
INSTR ()	LOB におけるパターンの n 番目の出現位置を戻します。
READ ()	指定されたオフセットから LOB のデータを読み込みます。
SUBSTR ()	指定されたオフセットから LOB 値の一部を戻します。

PL/SQL: テンポラリ LOB を操作するファンクション/プロシージャ

表 3-5 PL/SQL: テンポラリ LOB を操作する DBMS_LOB プロシージャ

ファンクション/プロシージャ	説明
CREATETEMPORARY ()	テンポラリ LOB を作成します。
ISTEMPORARY ()	LOB ロケータがテンポラリ LOB を参照するかどうかをチェックします。
FREETEMPORARY ()	テンポラリ LOB を解放します。

PL/SQL: BFILE 固有の読み込み専用ファンクション/プロシージャ

表 3-6 PL/SQL: BFILE 固有の DBMS_LOB 読み込み専用プロシージャ

ファンクション/プロシージャ	説明
FILECLOSE ()	ファイルをクローズします。 ¹
FILECLOSEALL ()	オープンしていたすべてのファイルをクローズします。
FILEEXISTS ()	ファイルがサーバー上に存在するかどうかをチェックします。
FILEGETNAME ()	ディレクトリ別名およびファイル名を取得します。
FILEISOPEN ()	入力 BFILE ロケータを使用して、ファイルがオープンされたかどうかをチェックします。 ²
FILEOPEN ()	ファイルをオープンします。 ³

¹ CLOSE () でも可能です。

² ISOPEN () でも可能です。

³ OPEN () でも可能です。

PL/SQL: 内部および外部 LOB をオープンおよびクローズするファンクション/プロシージャ

表 3-7 PL/SQL: 内部および外部 LOB をオープンおよびクローズする DBMS_LOB プロシージャ

ファンクション/プロシージャ	説明
OPEN ()	LOB をオープンします。
ISOPEN ()	LOB がオープンしているかどうかをチェックします。
CLOSE ()	LOB をクローズします。

次の章では、特定の LOB 操作 (LOB を含む行の INSERT など) について、これらのプロシージャをさらに詳しく説明します。

- 第 9 章「内部永続 LOB」
- 第 10 章「テンポラリ LOB」
- 第 11 章「外部 LOB (BFILE)」

将来のリリースでは、Oracle8i のソフトウェア CD-ROM の /rdbms/demo ディレクトリから、対応する PL/SQL のサンプル・スクリプトにアクセスできるようになります。

C (OCI) を使用した LOB の作業

Oracle Call Interface (OCI) を使用すると、次のようにして内部 LOB の全体、または内部 LOB の初め、中、終わりの部分を変更できます。

- 内部および外部 LOB (BFILE) からの読み込み
- 内部 LOB への書き込み

また、OCI には、次のことを行うために使用できる関数が含まれます。

- 内部 LOB (BLOB、CLOB、NCLOB) および外部 LOB (BFILE) に格納されたデータへのアクセス
- 内部 LOB (BLOB、CLOB、NCLOB) の変更

これらの関数については、表 3-8 ~ 表 3-14 を参照してください。詳細は、この項の後半で説明します。

UCS2 における読み込み / 書き込みのための CSID パラメータの OCI_UCS2ID への設定

データを 2 バイトの Unicode (UCS2) フォーマットで読み込みまたは書き込みする場合、OCILOBRead および OCILOBWrite の `csid` (キャラクタ・セット ID) パラメータを `OCI_UCS2ID` に設定します。`csid` パラメータは、バッファ・パラメータ用のキャラクタ・セット ID を示します。`csid` パラメータは、すべてのキャラクタ・セット ID に設定できません。`csid` パラメータが設定されていると、環境変数 `NLS_LANG` より優先されます。

参照:

- パラメータ、パラメータ・タイプ、戻り値、サンプル・コードなどの詳細は、『Oracle8i コール・インタフェース・プログラマーズ・ガイド』を参照してください。
- 異なる言語におけるアプリケーションの実装の詳細は、『Oracle8i NLS ガイド』を参照してください。

オフセット・パラメータおよび量パラメータ: 固定幅と可変幅、OCI での文字またはバイト

固定幅キャラクタ・セットの規則

OCI では、固定幅のクライアント側キャラクタ・セットに対して、次の規則が適用されます。

- CLOB および NCLOB: オフセット・パラメータおよび量パラメータは、常に文字で示されます。

- BLOB および BFILES: オフセット・パラメータおよび量パラメータは、常にバイトで示されます。

可変幅キャラクタ・セットの規則

可変幅のクライアント側キャラクタ・セットに対しては、次の規則が適用されます。

- **オフセット・パラメータ**: クライアント側キャラクタ・セットが可変幅かどうかにかかわらず、オフセット・パラメータは常に次のように表されます。
 - CLOB および NCLOB: 文字で表されます。
 - BLOB および BFILE: バイトで表されます。
- **量パラメータ**: 量パラメータは、常に次のように表されます。
 - サーバー側 LOB を参照する場合: 文字で表されます。
 - クライアント側バッファを参照する場合: バイトで表されます。
- **OCILOBFileGetLength**: クライアント側キャラクタ・セットが可変幅かどうかにかかわらず、出力の長さは次のように表されます。
 - CLOB および NCLOB: 文字で表されます。
 - BLOB および BFILE: バイトで表されます。
- **OCILOBRead**: 可変幅のクライアント側キャラクタ・セット、CLOB および NCLOB では次のようになります。
 - **入力量**は、文字で表されます。入力量は、サーバー側の CLOB または NCLOB から読み込まれる文字数を表します。
 - **出力量**は、バイトで表されます。出力量は、バッファ `bufp` に読み込まれたバイト数を表します。
- **OCILOBWrite**: 可変幅のクライアント側キャラクタ・セット、CLOB および NCLOB では次のようになります。
 - **入力量**は、バイトで表されます。入力量は、入力バッファ `bufp` にあるデータのバイト数を表します。
 - **出力量**は、文字で表されます。出力量は、サーバー側の CLOB または NCLOB に書き込まれた文字数を表します。

他の操作

他のすべての LOB 操作については、クライアント側キャラクタ・セットかどうかにかかわらず、量パラメータは CLOB および NCLOB については文字で表されます。これには、OCILOBCopy、OCILOBErase、OCILOBLoadFromFile および OCILOBTrim が含まれます。これらの操作はすべて、サーバー上の LOB データの量に関する操作です。

参照: 詳細は、『Oracle8i NLS ガイド』を参照してください。

NCLOB

- NCLOB パラメータは、メソッド中で使用できます。
- NCLOB パラメータは、オブジェクト型の属性としては使用できません。

OCILobLoadFromFile: 量パラメータに BFILE より短い値を指定する

OCILobLoadFromFile を使用する場合、量パラメータに、BFILE の長さより長い値を指定することはできません。

OCILobRead: 量パラメータに 4GB -1 の値を指定する

OCILobRead では、量パラメータを 4GB -1 に指定できます。こうすると、LOB の終わりまでが読み込まれます。

OCI LOB の例

OCI の例は、次の章でも記載されています。

- 第 9 章「内部永続 LOB」
- 第 10 章「テンポラリ LOB」
- 第 11 章「外部 LOB (BFILE)」

将来のリリースでは、Oracle8i のソフトウェア CD-ROM から、これらのすべてのサンプル・スクリプトにアクセスできるようになります。Oracle8i のソフトウェア CD-ROM では、次の使用可能な OCI スクリプトを参照できます。

- /ORACLE_HOME/rdbms/demo/demolb.c
- /ORACLE_HOME/rdbms/demo/demolb2.c
- /ORACLE_HOME/rdbms/demo/demolbs.c

NT では、次のスクリプトを参照できます。

- %ORACLE_HOME%\Oci\Samples\demolb.c など

これ以外の OCI のデモ・スクリプトについては、『Oracle8i コール・インタフェース・プログラマーズ・ガイド』の付録 B「OCI デモ・プログラム」を参照してください。

OCI の詳細

OCI の詳細および機能は、次の Web サイトを参照してください。

- <http://www.oracle.com/products> から、アプリケーション・サーバーを選択してください。検索行に OCI と入力すると、OCI に関する情報を入手できます。
- OCI の機能については、<http://technet.oracle.com/Products/Programmer> を参照してください。

OCI: BLOB、BLOB、NCLOB および BFILE を操作する関数

BLOB、BLOB、NCLOB および BFILE で操作される OCI 関数は、次のとおりです。

- 内部 LOB の変更は、[表 3-8](#) を参照してください。
- LOB 値の読み込みまたはテストは、[表 3-9](#) を参照してください。
- テンポラリ LOB の作成や解放、またはテンポラリ LOB が存在するかどうかのチェックは、[表 3-10](#) を参照してください。
- 外部 LOB (BFILE) における読み込み専用関数については、[表 3-11](#) を参照してください。
- LOB ロケータでの操作は、[表 3-12](#) を参照してください。
- LOB バッファリングについては、[表 3-13](#) を参照してください。
- LOB のオープンおよびクローズは、[表 3-14](#) を参照してください。

OCI: 内部 LOB (BLOB、CLOB および NCLOB) の値を変更する関数

表 3-8 OCI: 内部 LOB (BLOB、CLOB および NCLOB) の値を変更する関数

関数/プロシージャ	説明
OCILOBAppend()	LOB 値を別の LOB に追加します。
OCILOBCopy()	LOB の全体または一部を別の LOB にコピーします。
OCILOBErase()	指定されたオフセットから開始して、LOB の一部を消去します。
OCILOBLoadFromFile()	BFILE データを内部 LOB にロードします。
OCILOBTrim()	LOB を切り捨てます。
OCILOBWrite()	バッファのデータを LOB に書き込み、既存データを上書きします。
OCILOBWriteAppend()	データをバッファから LOB の終りに書き込みます。

OCI: 内部 LOB および外部 LOB (BFILE) 値の読み込みまたはテストを行う関数

表 3-9 OCI: 内部 LOB および外部 LOB (BFILE) の値の読み込みまたはテストを行う関数

関数 / プロシージャ	説明
OCILOBGetChunkSize()	読み込みおよび書き込み用にチャンク・サイズを取得します。これは内部 LOB に適用され、外部 LOB (BFILE) には適用されません。
OCILOBGetLength()	LOB または BFILE の長さを戻します。
OCILOBRead()	NULL でない LOB または BFILE の指定部分をバッファに読み込みます。

OCI: テンポラリ LOB を操作する関数

表 3-10 OCI: テンポラリ LOB を操作する関数

関数 / プロシージャ	説明
OCILOBCreateTemporary()	テンポラリ LOB を作成します。
OCILOBIsTemporary()	テンポラリ LOB が存在するかどうかをチェックします。
OCILOBFreeTemporary()	テンポラリ LOB を解放します。

OCI: BFILE 固有の読み込み専用関数

表 3-11 OCI: BFILE 固有の読み込み専用関数

関数 / プロシージャ	説明
OCILOBFileClose()	オープンしている BFILE をクローズします。
OCILOBFileCloseAll()	オープンしているすべての BFILE をクローズします。
OCILOBFileExists()	BFILE が存在するかどうかをチェックします。
OCILOBFileGetName()	BFILE の名前を戻します。
OCILOBFileIsOpen()	BFILE がオープンしているかどうかをチェックします。
OCILOBFileOpen()	BFILE をオープンします。

OCI: LOB ロケータ関数

表 3-12 OCI: LOB ロケータ関数

関数 / プロシージャ	説明
OCILobAssign()	LOB ロケータを別の LOB ロケータに割り当てます。
OCILobCharSetForm()	LOB のキャラクタ・セット・フォームを戻します。
OCILobCharSetId()	LOB のキャラクタ・セット ID を戻します。
OCILobFileSetName()	BFILE の名前をロケータに設定します。
OCILobIsEqual()	2 つの LOB ロケータが同じ LOB を参照しているかどうかをチェックします。
OCILobLocatorIsInit()	LOB ロケータが初期化されているかどうかをチェックします。

OCI: LOB バッファリング関数

表 3-13 OCI: LOB バッファリング関数

関数 / プロシージャ	説明
OCILobDisableBuffering()	バッファリング・サブシステムを使用禁止にします。
OCILobEnableBuffering()	これ以降の LOB データの読み込み / 書き込みに、LOB バッファリング・サブシステムを使用します。
OCILobFlushBuffer()	LOB バッファリング・サブシステムへの変更を、データベース (サーバー) へフラッシュします。

OCI: 内部および外部 LOB をオープンおよびクローズする関数

表 3-14 OCI: 内部および外部 LOB をオープンおよびクローズする関数

ファンクション / プロシージャ	説明
OCILobOpen()	LOB をオープンします。
OCILobIsOpen()	LOB がオープンしているかどうかをチェックします。
OCILobClose()	LOB をクローズします。

プロシージャ例 : main() および seelfLOBIsOpen

このマニュアルの後半部分で OCI の例について作業する場合は、次のような main() 関数が利用できます。ここでは、seelfLOBIsOpen を例に説明します。

```
int main(char *argv, int argc)
{
    /* OCI ハンドルの使用を宣言します。 */
    OCIEnv      *envhp;
    OCIserver    *srvhp;
    OCISvcCtx    *svchp;
    OCIError     *errhp;
    OCISession   *authp;
    OCIStmt      *stmthp;
    OCILobLocator *Lob_loc;

    /* OCI 環境を作成および初期化します。 */
    (void) OCIEnvCreate(&envhp, (ub4)OCI_DEFAULT, (dvoid *)0,
                      (dvoid * (*)(dvoid *, size_t)) 0,
                      (dvoid * (*)(dvoid *, dvoid *, size_t))0,
                      (void (*)(dvoid *, dvoid *))0,
                      (size_t) 0, (dvoid **) 0);

    /* エラー・ハンドルを割り当てます。 */
    (void) OCIHandleAlloc((dvoid *) envhp, (dvoid **) &errhp, OCI_HTYPE_ERROR,
                          (size_t) 0, (dvoid **) 0);

    /* サーバー・コンテキストを割り当てます。 */
    (void) OCIHandleAlloc((dvoid *) envhp, (dvoid **) &srvhp, OCI_HTYPE_SERVER,
                          (size_t) 0, (dvoid **) 0);

    /* サービス・コンテキストを割り当てます。 */
    (void) OCIHandleAlloc((dvoid *) envhp, (dvoid **) &svchp, OCI_HTYPE_SVCCTX,
                          (size_t) 0, (dvoid **) 0);

    /* Oracle データベースに連結します。 */
    (void) OCIserverAttach(srvhp, errhp, (text *)"", strlen(""), 0);

    /* サービス・コンテキストのサーバー・コンテキスト属性を設定します。 */
    (void) OCIAttrSet ((dvoid *) svchp, OCI_HTYPE_SVCCTX,
                      (dvoid *)srvhp, (ub4) 0,
                      OCI_ATTR_SERVER, (OCIError *) errhp);

    /* セッション・ハンドルを割り当てます。 */
    (void) OCIHandleAlloc((dvoid *) envhp,
                          (dvoid **)&authp, (ub4) OCI_HTYPE_SESSION,
                          (size_t) 0, (dvoid **) 0);
}
```

```
/* セッション・ハンドルのユーザー名を設定します。*/
(void) OCIAttrSet((dvoid *) authp, (ub4) OCI_HTYPE_SESSION,
                 (dvoid *) "samp", (ub4) 4,
                 (ub4) OCI_ATTR_USERNAME, errhp);
/* セッション・ハンドルのパスワードを設定します。*/
(void) OCIAttrSet((dvoid *) authp, (ub4) OCI_HTYPE_SESSION,
                 (dvoid *) "samp", (ub4) 4,
                 (ub4) OCI_ATTR_PASSWORD, errhp);

/* セッションを認証し、開始します。*/
checkerr(errhp, OCISessionBegin (svchp, errhp, authp, OCI_CRED_RDBMS,
                               (ub4) OCI_DEFAULT));

/* サービス・コンテキストのセッション属性を設定します。*/
(void) OCIAttrSet((dvoid *) svchp, (ub4) OCI_HTYPE_SVCCTX,
                 (dvoid *) authp, (ub4) 0,
                 (ub4) OCI_ATTR_SESSION, errhp);

/* ----- この時点で、有効なセッションが作成されます。 -----*/
printf ("user session created \n");

/* 文ハンドルを割り当てます。*/
checkerr(errhp, OCIHandleAlloc( (dvoid *) envhp, (dvoid **) &stmthp,
                               OCI_HTYPE_STMT, (size_t) 0, (dvoid **) 0));

/* ===== これ以降はサンプルのプロシージャ・コールです。 =====*/

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

return 0;
}

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

    switch (status)
    {
        case OCI_SUCCESS:
            break;
        case OCI_SUCCESS_WITH_INFO:
            (void) printf("Error - OCI_SUCCESS_WITH_INFO\n");
    }
}
```

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

/* ロケータ変数にロケータを選択します。*/

sb4 select_frame_locator(Lob_loc, errhp, svchp, stmthp)
OCILobLocator *Lob_loc;
OCIError      *errhp;
OCISvcCtx     *svchp;
OCISstmt      *stmthp;
{
    text      *sqlstmt =
        (text *) "SELECT Frame FROM Multimedia_tab WHERE Clip_ID=1";
    OCIDefine *defnp1;

    checkerr (errhp, OCISstmtPrepare(stmthp, errhp, sqlstmt,
        (ub4) strlen((char *) sqlstmt),
        (ub4) OCI_MTV_SYNTAX, (ub4) OCI_DEFAULT));

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

```
/* SELECT を実行し、1 行をフェッチします。*/
checkerr(errhnp, OCISstmtExecute(svchp, stmthp, errhnp, (ub4) 1, (ub4) 0,
                                (CONST OCISnapshot*) 0, (OCISnapshot*) 0,
                                (ub4) OCI_DEFAULT));

return (0);
}

void seeIfLOBIsOpen(envhnp, errhnp, svchp, stmthp)
OCIEnv    *envhnp;
OCIError  *errhnp;
OCISvcCtx *svchp;
OCISstmt  *stmthp;
{
    OCILobLocator *Lob_loc;
    int isOpen;

    /* ロケータ・リソースを割り当てます。*/
    (void) OCIDescriptorAlloc((dvoid *)envhnp, (dvoid **)&Lob_loc,
                              (ub4)OCI_DTYPE_LOB, (size_t)0, (dvoid **)0);

    /* ロケータを選択します。*/
    (void)select_frame_locator(Lob_loc, errhnp, svchp, stmthp);

    /* LOB がオープンしているかどうかを確認します。*/
    checkerr (errhnp, OCILobIsOpen(svchp, errhnp, Lob_loc, &isOpen));

    if (isOpen)
    {
        printf(" Lob is Open\n");
        /* LOB がオープンされている場合の処理が行われます。*/
    }
    else
    {
        printf(" Lob is not Open\n");
        /* LOB がオープンされていない場合の処理が行われます。*/
    }

    /* ロケータが保持しているリソースを解放します。*/
    (void) OCIDescriptorFree((dvoid *) Lob_loc, (ub4) OCI_DTYPE_LOB);
    return;
}
```

C/C++ (Pro*C) を使用した LOB の作業

埋込み SQL を使用すると、内部 LOB の全体、または LOB の初め、中、終りの部分を変更できます。内部および外部 LOB の両方に読み込み目的のアクセスが可能で、内部 LOB には書き込みも可能です。

埋込み SQL 文によって、BLOB、CLOB、NCLOB および BFILE に格納されたデータにアクセスできます。これらの文について、表 3-15 ～表 3-21 に示します。詳細は、この章の後半で説明します。

参照： 構文、ホスト変数、ホスト変数型およびサンプル・コードの詳細は、『Oracle8i Pro*C/C++ プリコンパイラ・プログラマーズ・ガイド』を参照してください。

LOB を表す入力ロケータ・ポインタの割当て

PL/SQL のロケータとは異なり、Pro*C/C++ のロケータはロケータ・ポインタにマップされ、LOB または BFILE 値を参照するために使用できます。

埋込み SQL LOB 文が正常に実行するには、次の処理が必要です。

1. 文を実行する前に、データベース表領域または外部ファイル・システムに存在する LOB を表す入力ロケータ・ポインタを割り当てます。
2. LOB ロケータを LOB ロケータ・ポインタ変数に SELECT します。
3. 埋込み SQL LOB 文の中でこの変数を使用し、LOB 値にアクセスし操作します。

各埋込み SQL LOB 文の例は、次の章でも記載されています。

- [第 9 章「内部永続 LOB」](#)
- [第 10 章「テンポラリ LOB」](#)
- [第 11 章「外部 LOB \(BFILE\)」](#)

また、将来のリリースでは、Oracle8i のソフトウェア CD-ROM の /rdbams/demo ディレクトリから、これらのサンプル・スクリプトにアクセスできるようになります。

Pro*C/C++: BLOB、CLOB、NCLOB および BFILE を操作する文

BLOB、CLOB および NCLOB を操作する Pro*C 文は、次のとおりです。

- 内部 LOB の変更は、[表 3-15](#) を参照を参照してください。

- LOB 値の読み込みまたはテストは、表 3-16 を参照してください。
- テンポラリ LOB の作成または解放、またはテンポラリ LOB が存在するかどうかのチェックは、表 3-17 を参照してください。
- BFILE におけるクローズおよびファイルの有無は、表 3-18 を参照してください。
- LOB ロケータでの操作は、表 3-19 を参照してください。
- LOB バッファリングについては、表 3-20 を参照してください。
- LOB または BFILE のオープンまたはクローズは、表 3-21 を参照してください。

Pro*C/C++: 内部 LOB (BLOB、CLOB および NCLOB) の値を変更する埋込み SQL 文

表 3-15 Pro*C/C++: 内部 LOB (BLOB、CLOB および NCLOB) の値を変更する埋込み SQL 文

文	説明
APPEND	ある LOB 値を別の LOB に追加します。
COPY	LOB の全体または一部を別の LOB にコピーします。
ERASE	指定のオフセットから開始して、LOB の一部を消去します。
LOAD FROM FILE	内部 LOB の指定されたオフセットに BFILE データをロードします。
TRIM	LOB を切り捨てます。
WRITE	LOB の指定されたオフセットにバッファのデータを書き込みます。
WRITE APPEND	バッファのデータを LOB の最後に書き込みます。

Pro*C/C++: 内部および外部 LOB 値の読み込みまたはテストを行う埋込み SQL 文

表 3-16 Pro*C/C++: 内部および外部 LOB の値の読み込みまたはテストを行う埋込み SQL 文

文	説明
DESCRIBE [CHUNKSIZE]	書き込み用にチャンク・サイズを取得します。これは内部 LOB のみに適用されます。外部 LOB (BFILE) には適用されません。
DESCRIBE [LENGTH]	LOB または BFILE の長さを戻します。

表 3-16 Pro*C/C++: 内部および外部 LOB の値の読み込みまたはテストを行う埋込み SQL 文

文	説明
READ	NULL 以外の LOB または BFILE の指定部分をバッファに読み込みます。

Pro*C/C++: テンポラリ LOB を操作する埋込み SQL 文

表 3-17 Pro*C/C++: テンポラリ LOB を操作する埋込み SQL 文

文	説明
CREATE TEMPORARY	テンポラリ LOB を作成します。
DESCRIBE [ISTEMPORARY]	LOB ロケータがテンポラリ LOB を参照しているかどうかをチェックします。
FREE TEMPORARY	テンポラリ LOB を解放します。

Pro*C/C++: BFILE 固有の埋込み SQL 文

表 3-18 Pro*C/C++: BFILE 固有の埋込み SQL 文

文	説明
FILE CLOSE ALL	オープンしているすべての BFILE をクローズします。
DESCRIBE [FILEEXISTS]	BFILE が存在するかどうかをチェックします。
DESCRIBE [DIRECTORY, FILENAME]	ディレクトリ別名または BFILE のファイル名 (あるいはその両方) を戻します。

Pro*C/C++: LOB ロケータ埋込み SQL 文

表 3-19 Pro*C/C++: LOB ロケータ埋込み SQL 文

文	説明
ASSIGN	LOB ロケータを別の LOB ロケータに割り当てます。
FILE SET	ディレクトリ別名と BFILE のファイル名をロケータに設定します。

Pro*C/C++: LOB バッファリング埋込み SQL 文

表 3-20 Pro*C/C++: LOB バッファリング埋込み SQL 文

文	説明
DISABLE BUFFERING	バッファリング・サブシステムを使用禁止にします。
ENABLE BUFFERING	これ以降の LOB データの読み込み / 書き込みに、LOB バッファリング・サブシステムを使用します。
FLUSH BUFFER	LOB バッファリング・サブシステムへの変更を、データベース (サーバー) へフラッシュします。

Pro*C/C++: 内部 LOB および外部 LOB (BFILE) をオープンおよびクローズするための埋込み SQL 文

表 3-21 Pro*C/C++: 内部 LOB および外部 LOB (BFILE) をオープンおよびクローズするための埋込み SQL 文

文	説明
OPEN	LOB または BFILE をオープンします。
DESCRIBE [ISOPEN]	LOB または BFILE がオープンしているかどうかをチェックします。
CLOSE	LOB または BFILE をクローズします。

COBOL (Pro*COBOL) を使用した LOB の作業

埋め込み SQL を使用すると、内部 LOB の全体、または内部 LOB の初め、中、終りの部分を変更できます。内部および外部 LOB の両方に読み込み目的のアクセスが可能で、内部 LOB には書き込みも可能です。

埋め込み SQL 文によって、BLOB、CLOB、NCLOB および BFILE に格納されたデータにアクセスできます。これらの文について、表 3-22 ～表 3-28 に示します。詳細は、このマニュアルの後半で説明します。

LOB を表す入力ロケータ・ポインタの割当て

PL/SQL のロケータとは異なり、Pro*COBOL のロケータはロケータ・ポインタにマップされ、LOB または BFILE 値を参照するために使用されます。埋め込み SQL LOB 文が正常に実行するには、次の処理が必要です。

1. 文を実行する前に、データベース表領域または外部ファイル・システムに存在する LOB を表す入力ロケータ・ポインタを割り当てます。
2. LOB ロケータを LOB ロケータ・ポインタ変数に SELECT します。
3. 埋め込み SQL LOB 文でこの変数を使用し、LOB 値にアクセスし操作します。

各埋め込み SQL LOB 文の例は、次の章でも記載されています。

- [第 9 章「内部永続 LOB」](#)
- [第 10 章「テンポラリ LOB」](#)
- [第 11 章「外部 LOB \(BFILE\)」](#)

また、将来のリリースでは、Oracle8i のソフトウェア CD-ROM の /rdbms/demo ディレクトリから、これらのサンプル・スクリプトにアクセスできるようになります。

Pro*COBOL インタフェースが必要な機能を提供していない場合は、C を使用して OCI をコールできます。このようなプログラムはオペレーティング・システムによって異なるため、ここでは例を示しません。

参照： 構文、ホスト変数、ホスト変数型およびサンプル・コードの詳細は、『Oracle8i Pro*COBOL プリコンパイラ・プログラマーズ・ガイド』を参照してください。

Pro*COBOL: BLOB、CLOB、NCLOB および BFILE を操作する文

BLOB、CLOB、NCLOB および BFILE を操作する Pro*COBOL 文は、次のとおりです。

- 内部 LOB の変更は、表 3-22 を参照してください。
- 内部および外部 LOB の値の読み込みまたは調査は、表 3-23 を参照してください。
- テンポラリ LOB の作成、解放、または LOB ロケータのチェックは、表 3-24 を参照してください。
- BFILE におけるクローズおよびファイルの有無は、表 3-25 を参照してください。
- LOB ロケータでの操作は、表 3-26 を参照してください。
- LOB バッファリングについては、表 3-27 を参照してください。
- 内部 LOB または BFILE のオープンまたはクローズは、表 3-28 を参照してください。

Pro*COBOL: 内部 LOB (BLOB、CLOB および NCLOB) の値を変更する埋込み SQL 文

表 3-22 Pro*COBOL: BLOB、CLOB および NCLOB の値を変更する埋込み SQL 文

文	説明
APPEND	LOB 値を別の LOB に追加します。
COPY	LOB の全体または一部を別の LOB にコピーします。
ERASE	指定のオフセットから開始して、LOB の一部を消去します。
LOAD FROM FILE	内部 LOB の指定されたオフセットに BFILE データをロードします。
TRIM	LOB を切り捨てます。
WRITE	LOB の指定されたオフセットにバッファのデータを書き込みます。
WRITE APPEND	バッファのデータを LOB の最後に書き込みます。

Pro*COBOL: 内部および外部 LOB 値の読み込みまたはテストを行う埋込み SQL 文

表 3-23 Pro*COBOL: 内部および外部 LOB 値の読み込みまたはテストを行う埋込み SQL 文

文	説明
DESCRIBE [CHUNKSIZE]	書き込み用のチャンク・サイズを取得します。
DESCRIBE [LENGTH]	LOB または BFILE の長さを戻します。
READ	NULL 以外の LOB または BFILE の指定部分をバッファに読み込みます。

Pro*COBOL: テンポラリ LOB を操作する埋込み SQL 文

表 3-24 Pro*COBOL: テンポラリ LOB を操作する埋込み SQL 文

文	説明
CREATE TEMPORARY	テンポラリ LOB を作成します。
DESCRIBE [ISTEMPORARY]	LOB ロケータがテンポラリ LOB を参照しているかどうかをチェックします。
FREE TEMPORARY	テンポラリ LOB を解放します。

Pro*COBOL: BFILE 固有の埋込み SQL 文

表 3-25 Pro*COBOL: BFILE 固有の埋込み SQL 文

文	説明
FILE CLOSE ALL	オープンしているすべての BFILE をクローズします。
DESCRIBE [FILEEXISTS]	BFILE が存在するかどうかをチェックします。
DESCRIBE [DIRECTORY, FILENAME]	ディレクトリ別名または BFILE のファイル名 (あるいはその両方) を戻します。

Pro*COBOL: LOB ロケータ埋込み SQL 文

表 3-26 Pro*COBOL: LOB ロケータ文の埋込み SQL 文

文	説明
ASSIGN	LOB ロケータを別の LOB ロケータに割り当てます。
FILE SET	ディレクトリ別名と BFILE のファイル名をロケータに設定します。

Pro*COBOL: LOB バッファリング埋込み SQL 文

表 3-27 Pro*COBOL: LOB バッファリング埋込み SQL 文

文	説明
DISABLE BUFFERING	バッファリング・サブシステムを使用禁止にします。
ENABLE BUFFERING	後続の LOB データの読み込み / 書き込みに、LOB バッファリング・サブシステムを使用します。
FLUSH BUFFER	LOB バッファリング・サブシステムへの変更を、データベース (サーバー) へフラッシュします。

Pro*COBOL: 内部 LOB および外部 LOB (BFILE) をオープンおよびクローズするための埋込み SQL 文

表 3-28 Pro*COBOL: 内部 LOB および外部 LOB (BFILE) をオープンおよびクローズするための埋込み SQL 文

文	説明
OPEN	LOB または BFILE をオープンします。
DESCRIBE [ISOPEN]	LOB または BFILE がオープンしているかどうかをチェックします。
CLOSE	LOB または BFILE をクローズします。

Visual Basic (Oracle Objects for OLE (OO4O)) を使用した LOB の作業

次のオブジェクト・インタフェースの1つを使用すると、Oracle Objects for OLE (OO4O) API を介して、内部 LOB の全体、または内部 LOB の初め、中、終わりの部分に対する変更ができます。

- **OraBlob**: データベース内の BLOB データ型において操作を実行するためのメソッドを提供します。
- **OraClob**: データベース内の CLOB データ型において操作を実行するためのメソッドを提供します。
- **OraBFile**: オペレーティング・システム・ファイル内に格納されている BFILE データにおける操作を実行するためのメソッドを提供します。

注意: OracleBlob および OracleClob は使用しないでください。今後は使用できません。

OO4O 構文の参照および詳細

構文

OO4O 構文については、OO4O オンライン・ヘルプを参照してください。

Oracle Objects for OLE (OO4O) は、Oracle8i Client for Windows NT に含まれる Windows ベースの製品です。マニュアルはなく、オンライン・ヘルプのみです。

オンライン・ヘルプは、Oracle8i インストールの「Application Development」サブメニューから利用できます。「Help Topic」メニューから特定のメソッドおよびプロパティを表示するには、「Contents」タブから、「OO4O Automation Server」>「Methods」または「Properties」を選択します。

詳細

OO4O の詳細は、次の Web サイトを参照してください。

<http://technet.oracle.com>

OO4O または Oracle Objects for OLE についての項目を検索してください。

OraBlob、OraClob および OraBfile オブジェクト・インタフェースによるロケータのカプセル化

これらのインタフェースは LOB ロケータをカプセル化するため、ユーザーは、ロケータではなく、提供されているメソッドおよびプロパティを使用して操作することで、状態情報を取得できます。

ダイナセットの一部として取り出され LOB ロケータを表す OraBlob および OraClob オブジェクト

OraBlob オブジェクトおよび OraClob オブジェクトがダイナセットの一部として取り出された場合、これらのオブジェクトはダイナセット・カレント行の LOB ロケータを表します。移動操作でダイナセット・カレント行が変わると、OraBlob オブジェクトおよび OraClob オブジェクトは新しいカレント行の LOB ロケータを表します。

ダイナセット移動からロケータの独立を維持するための Clone メソッドの使用

ダイナセットに移動操作を行っても OraBlob オブジェクトおよび OraClob オブジェクトの LOB ロケータが影響を受けないようにするためには、Clone メソッドを使用します。このメソッドは、OraBlob オブジェクトおよび OraClob オブジェクトを戻します。また、これらのオブジェクトを PL/SQL バインド・パラメータとして使用することもできます。

OraBlob および OraBfile の例

次の例に、OraBlob および OraBfile の使用方法を示します。機能および例の詳細は、第 9 章、第 10 章および第 11 章を参照してください。

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

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

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

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

```
Set OraMyBFile = OraParameters("mybfile").value  
' 次の行に進みます。  
OraDyn.MoveNext
```

```
OraDyn.Edit  
'OraSound1 データを BFILE のデータで更新します。  
OraSound1.CopyFromBFile OraMyBFile  
OraDyn.Update
```

```
OraDyn.MoveNext  
' 次の行に進みます。  
OraDyn.Edit  
' OraSoundClone で表された最初の行の LOB データを追加して、OraSound1 を更新します。  
OraSound1.Append OraSoundClone  
OraDyn.Update
```

```
OraConnection.CommitTrans
```

この例では、次のことを表しています。

OraSound1: ダイナセット内のカレント行のロケータを表します。

OraSoundClone: 最初の行のロケータを表します。

カレント行における変更 (OraDyn.MoveNext など) は、次のことを意味します。

OraSound1: 2 番目の行のロケータを表します。

OraSoundClone: 最初の行にあるロケータを表します。OraDyn 行ナビゲーションにかかわらず、OraSoundClone は最初の行のロケータのみを参照します。

OraMyBFile: OraDatabase.ExecuteSQL を実行することによって、PL/SQL プロシージャを実行した結果として、PL/SQL OUT パラメータから取得されたロケータを参照します。

注意: SQL の実行によって取得された LOB は、トランザクションの存続期間のみ有効です。このため、BEGINTRANS および COMMITTRANS を使用して、トランザクションの存続期間を指定します。

OO4O: BLOB、CLOB、NCLOB および BFILE に格納されたデータにアクセスするメソッドおよびプロパティ

Oracle Objects for OLE (OO4O) には、BLOB、CLOB、NCLOB および BFILE に格納されたデータにアクセスするために使用できるメソッドおよびプロパティが含まれます。これらのメソッドおよびプロパティについて、表 3-29 ~ 表 3-35 に示します。詳細は、次の章で説明します。

- 第 9 章「内部永続 LOB」
- 第 11 章「外部 LOB (BFILE)」

また、将来のリリースでは、Oracle8i のソフトウェア CD-ROM の /rdbms/demo ディレクトリから、これらのスクリプトにアクセスできるようになります。

参照： パラメータ、パラメータ・タイプ、戻り値、サンプル・コードなどの詳細は、OO4O オンライン・ヘルプを参照してください。Oracle Objects for OLE (OO4O) は、Oracle8i Client for Windows NT に含まれる Windows ベースの製品です。マニュアルはなく、オンライン・ヘルプのみです。オンライン・ヘルプは、Oracle8i インストールの「Application Development」サブメニューから利用できます。

BLOB、CLOB、NCLOB および BFILE を操作する OO4O メソッドおよびプロパティは、次のとおりです。

- 内部 LOB の変更は、表 3-29 を参照してください。
- 内部および外部 LOB の値の読み込みまたはテストは、表 3-30 を参照してください。
- BFILE のオープンおよびクローズは、表 3-31 を参照してください。
- LOB バッファリングについては、表 3-32 を参照してください。
- LOB が NULL かどうかのチェック、またはポーリング量の取得または設定をするプロパティは、表 3-33 を参照してください。
- 読み込み専用 BFILE メソッドについては、表 3-34 を参照してください。
- BFILE プロパティについては、表 3-35 を参照してください。

OO4O: 内部 LOB (BLOB、CLOB および NCLOB) の値を変更するメソッド

表 3-29 OO4O: 内部 LOB (BLOB、CLOB および NCLOB) の値を変更するメソッド

メソッド	説明
OraBlob.Append	BLOB 値を別の LOB に追加します。
OraClob.Append	CLOB または NCLOB 値を別の LOB に追加します。
OraBlob.Copy	BLOB の一部を別の LOB にコピーします。
OraClob.Copy	CLOB または NCLOB の一部を別の LOB にコピーします。

表 3-29 OO4O: 内部 LOB (BLOB、CLOB および NCLOB) の値を変更するメソッド

メソッド	説明
OraBlob.Erase	指定のオフセットから開始して、BLOB の一部を消去します。
OraClob.Erase	指定のオフセットから開始して、CLOB または NCLOB の一部を消去します。
OraBlob.CopyFromBFile	BFILE データを内部 BLOB にロードします。
OraClob.CopyFromBFile	BFILE データを内部 CLOB または NCLOB にロードします。
OraBlob.Trim	BLOB を切り捨てます。
OraClob.Trim	CLOB または NCLOB を切り捨てます。
OraBlob.CopyFromFile	データをファイルから BLOB に書き込みます。
OraClob.CopyFromFile	データをファイルから CLOB または NCLOB に書き込みます。
OraBlob.Write	データを BLOB に書き込みます。
OraClob.Write	データを CLOB または NCLOB に書き込みます。

OO4O: 内部および外部 LOB 値の読み込みまたはテストを行うメソッド

表 3-30 OO4O: 内部および外部 LOB 値の読み込みまたはテストを行うメソッド

ファンクション/プロシージャ	説明
OraBlob.Read	NULL でない BLOB の指定部分をバッファに読み込みます。
OraClob.Read	NULL でない CLOB の指定部分をバッファに読み込みます。
OraBFile.Read	NULL でない BFILE の指定部分をバッファに読み込みます。
OraBlob.CopyToFile	NULL でない BLOB の指定部分をファイルに読み込みます。
OraClob.CopyToFile	NULL でない CLOB の指定部分をファイルに読み込みます。

OO4O: 外部 LOB (BFILE) をオープンおよびクローズするメソッド

表 3-31 OO4O: 外部 LOB (BFILE) をオープンおよびクローズするメソッド

メソッド	説明
OraBFile.Open	BFILE をオープンします。
OraBFile.Close	BFILE をクローズします。

0040: 内部 LOB バッファリング・メソッド

表 3-32 0040: 内部 LOB バッファリング・メソッド

メソッド	説明
OraBlob.FlushBuffer	BLOB バッファリング・サブシステムへの変更を、データベース (サーバー) へフラッシュします。
OraClob.FlushBuffer	CLOB バッファリング (サーバー)・サブシステムへの変更を、データベース (サーバー) へフラッシュします。
OraBlob.EnableBuffering	BLOB 操作のバッファリングを使用可能にします。
OraClob.EnableBuffering	CLOB 操作のバッファリングを使用可能にします。
OraBlob.DisableBuffering	BLOB 操作のバッファリングを使用禁止にします。
OraClob.DisableBuffering	CLOB 操作のバッファリングを使用禁止にします。

0040: LOB プロパティ

表 3-33 0040: LOB プロパティ

プロパティ	説明
IsNull (Read)	LOB が NULL であることを示します。
PollingAmount (Read/Write)	読み込み / 書き込みポーリング操作の合計を取得 / 設定します。
Offset (Read/Write)	読み込み / 書き込み操作のオフセットを取得 / 設定します。デフォルトでは 1 に設定されています。
Status (Read)	ポーリング状態を戻します。戻される値は次のとおりです。 <ul style="list-style-type: none"> ■ ORALOB_NEED_DATA: 読み込みまたは書き込みするデータが残っています。 ■ ORALOB_NO_DATA: 読み込みまたは書き込みするデータは残っていません。 ■ ORALOB_SUCCESS LOB: データは正常に読み込み / 書き込みされました。
Size (Read)	LOB データの長さを戻します。

OO4O: 外部 LOB (BFILE) 固有の読み専用メソッド

表 3-34 OO4O: 外部 LOB (BFILE) 固有の読み専用メソッド

メソッド	説明
<code>OraBFile.Close</code>	オープンしている BFILE をクローズします
<code>OraBFile.CloseAll</code>	オープンしているすべての BFILE をクローズします
<code>OraBFile.Open</code>	BFILE をオープンします
<code>OraBFile.IsOpen</code>	BFILE がオープンしているかどうかを判断します。

OO4O: 外部 LOB (BFILE) での操作のためのプロパティ

表 3-35 OO4O: 外部 LOB (BFILE) での操作のためのプロパティ

プロパティ	説明
<code>OraBFile.DirectoryName</code>	サーバー側のディレクトリ別名を取得 / 設定します。
<code>OraBFile.FileName (Read/Write)</code>	サーバー側のファイル名を取得 / 設定します。
<code>OraBFile.Exists</code>	BFILE が存在するかどうかをチェックします。

Java (JDBC) を使用した LOB の作業

Java での内部 LOB の変更

次のオブジェクトを介して JDBC API を使用すると、Java における内部 LOB の全体、または内部 LOB の初め、中、終わりの部分に対する変更ができます。

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

また、これらのオブジェクトは、JDBC 2.0 の仕様に従って `java.sql.Blob` および `java.sql.Clob` インタフェースを実装できます。この実装では、`java.sql.Blob` が予測される場合は `oracle.sql.BLOB` を使用でき、`java.sql.Clob` が予測される場合は `oracle.sql.CLOB` を使用できます。

Java を使用した内部 LOB および外部 LOB (BFILE) の読み込み

JDBC インタフェースでは、Java を使用して内部および外部 LOB (BFILE) の両方を読み込むことができます。

BLOB、CLOB および BFILE クラス

- **BLOB および CLOB クラス** JDBC: これらのクラスは BLOB および CLOB のデータ型を含むデータベースでラージ・オブジェクトを操作するメソッドを提供します。
- **BFILE クラス** JDBC: このクラスはデータベース内の BFILE データを操作するメソッドを提供します。

BLOB、CLOB および BFILE クラスは LOB ロケータをカプセル化するため、ユーザーは、ロケータではなく、提供されているメソッドおよびプロパティを使用して操作することで、状態情報を取得します。

DBMS_LOB パッケージのコール

これらのクラスによって提供されていないすべての LOB 機能は、PL/SQL DBMS_LOB パッケージへコールすることによってアクセスできます。この技法は、このマニュアルの中の例で繰り返し使用されています。

LOB の参照

次の 2 つの方法で、前述のすべての LOB を参照できます。

- `OracleResultSet` の列として
- `OraclePreparedStatement` の「OUT」型 PL/SQL パラメータとして

OracleResultSet の使用 : カレント行の LOB ロケータを表す取り出された BLOB および CLOB オブジェクト

`OracleResultSet` の一部として BLOB オブジェクトと CLOB オブジェクトが取り出された場合、これらのオブジェクトは現在選択されている行の LOB ロケータを表します。

移動操作 (`rset.next()` など) によってカレント行が変更されても、取り出されたロケータは元の LOB 行を参照したままです。

最新のカレント行のロケータを取り出すには、移動操作を行うたびに `OracleResultSet` 上で `getXXXX()` をコールする必要があります (XXXX は BLOB、CLOB または BFILE です)。

JDBC 構文の参照および詳細

構文

参照 :

- パラメータ、パラメータ・タイプ、戻り値、サンプル・コードなどの詳細は、『Oracle8i JDBC 開発者ガイドおよびリファレンス』を参照してください。
- 『Oracle8i SQLJ 開発者ガイドおよびリファレンス』も参照してください。

詳細

JDBC の詳細は、次の Web サイトを参照してください。

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

LOB での操作のための JDBC メソッド

BLOB、CLOB および BFILE を操作する JDBC メソッドは、次のとおりです。

- BLOB
 - BLOB 値の変更は、表 3-36 を参照してください。
 - BLOB 値の読み込みまたはテストは、表 3-37 を参照してください。
 - BLOB バッファリングについては、表 3-38 を参照してください
- CLOB
 - CLOB の変更は、表 3-39 を参照してください
 - CLOB 値の読み込みまたはテストは、表 3-40 を参照してください。
 - CLOB バッファリングについては、表 3-41 を参照してください。
- BFILE
 - BFILE の読み込みまたはテストは、表 3-42 を参照してください。
 - BFILE バッファリングについては、表 3-43 を参照してください。

JDBC: BLOB 値を変更する oracle.sql.BLOB メソッド

表 3-36 JDBC: BLOB 値を変更する oracle.sql.BLOB メソッド

メソッド	説明
<code>int putBytes(long, byte[])</code>	指定されたオフセットから、バイト配列を LOB に挿入します。

JDBC: BLOB 値の読み込みまたはテストを行う oracle.sql.BLOB メソッド

表 3-37 JDBC: BLOB 値の読み込みまたはテストを行う oracle.sql.BLOB メソッド

メソッド	説明
<code>byte[] getBytes(long, int)</code>	指定されたオフセットから、バイトの配列として LOB の内容を取得します。
<code>long position(byte[], long)</code>	LOB 内の指定されたバイト配列を、指定されたオフセットで検索します。
<code>long position(Blob, long)</code>	指定された BLOB を LOB 内で検索します。

表 3-37 JDBC: BLOB 値の読み込みまたはテストを行う oracle.sql.BLOB メソッド

メソッド	説明
public boolean equals(java.lang.Object)	この LOB を別の LOB と比較します。LOB ロケータを相互に比較します。
public long length()	LOB の長さを戻します。
public int getChunkSize()	LOB の ChunkSize を戻します。

JDBC: BLOB バッファリングのための oracle.sql.BLOB メソッドおよびプロパティ

表 3-38 JDBC: BLOB バッファリングのための oracle.sql.BLOB メソッドおよびプロパティ

メソッド	説明
public java.io.InputStream getBinaryStream()	LOB をバイナリ・ストリームとして流すストリームを戻します。
public java.io.OutputStream getBinaryOutputStream()	バイナリ・ストリームとして LOB に書き込むストリームを戻します。

JDBC: CLOB 値を変更する oracle.sql.CLOB メソッド

表 3-39 JDBC: CLOB 値を変更する oracle.sql.CLOB メソッド

メソッド	説明
int putString(long, java.lang.String)	指定されたオフセットから LOB に文字列を挿入します。
int putChars(long, char[])	指定されたオフセットから LOB に文字配列を挿入します。

JDBC: CLOB 値を読み込みまたはテストを行う oracle.sql.CLOB メソッド

表 3-40 JDBC: CLOB 値を読み込みまたはテストを行う oracle.sql.CLOB メソッド

メソッド	説明
java.lang.String getSubString(long, int)	LOB の部分文字列を文字列として戻します。
int getChars(long, int, char[])	LOB のサブセットを文字配列中に読み込みます。
long position(java.lang.String , long)	指定された文字列を LOB 内の指定されたオフセットから検索します。
longposition(oracle.jdbc2 .Clob, long)	指定された CLOB を LOB 内の指定されたオフセットから検索します。
boolean equals(java.lang.Object)	この LOB を他の LOB 別と比較します。
long length()	LOB の長さを戻します。
int getChunkSize()	LOB のチャンクサイズを戻します。

JDBC: CLOB バッファリングのための oracle.sql.CLOB メソッドおよびプロパティ

表 3-41 CLOB バッファリングのための JDBC oracle.sql.CLOB メソッドおよびプロパティ

メソッド	説明
java.io.InputStream getAsciiStream()	LOB を ASCII ストリームとして読み込みます。
java.io.OutputStream getAsciiOutputStream()	ASCII ストリームから LOB に書き込みます。
java.io.Reader getCharacterStream()	LOB を文字ストリームとして読み込みます。
java.io.Writer getCharacterOutputStream()	文字ストリームから LOB に書き込みます。

JDBC: 外部 LOB (BFILE) 値の読み込みまたはテストを行う oracle.sql.BFILE メソッド

表 3-42 JDBC: 外部 LOB (BFILE) 値の読み込みまたはテストを行う oracle.sql.BFILE メソッド

メソッド	説明
<code>byte[] getBytes(long, int)</code>	指定されたオフセットから、バイトの配列として BFILE の内容を取得します。
<code>int getBytes(long, int, byte[])</code>	BFILE のサブセットをバイト配列に読み込みます。
<code>long position(oracle.sql.BFILE, long)</code>	指定されたオフセットから、LOB 内の指定された BFILE の内容の 1 番目のものを検索します。
<code>long position(byte[], long)</code>	指定されたオフセットから、BFILE 内の指定されたバイト配列の 1 番目のものを検索します。
<code>boolean equals(java.lang.Object)</code>	この BFILE を他の BFILE と比較します。ロケータ・バイトを相互に比較します。
<code>long length()</code>	BFILE の長さを戻します。
<code>boolean fileExists()</code>	この BFILE が参照しているオペレーティング・システム (OS) ・ファイルが存在するかどうかをチェックします。
<code>public void openFile()</code>	この BFILE が参照している OS ファイルをオープンします。
<code>public void closeFile()</code>	この BFILE が参照している OS ファイルをクローズします。
<code>public boolean isFileOpen()</code>	この BFILE がすでにオープンしているかどうかをチェックします。
<code>public java.lang.String getDirAlias()</code>	この BFILE のディレクトリ別名を取得します。
<code>public java.lang.String getName()</code>	この BFILE が参照しているファイル名を取得します。

JDBC: BFILE バッファリングのための oracle.sql.BFILE メソッドおよびプロパティ

表 3-43 JDBC: CLOB バッファリングのための oracle.sql.BFILE メソッドおよびプロパティ

メソッド	説明
<code>public java.io.InputStream getBinaryStream()</code>	BFILE をバイナリ・ストリームとして読み込みます。

JDBC: 8.1.x および今後のリリースで使用できない OracleBlob および OracleClob

OracleBlob および OracleClob は、LOB データにアクセスするために JDBC 8.0.x ドライバで使用される Oracle の特別な機能です。リリース 8.1.x および今後のリリースでは、OracleBlob および OracleClob は使用できません。

OracleBlob または OracleClob を使用して LOB にアクセスすると、次の標準エラー・メッセージが表示されます。たとえば、Oracle8i リリース 8.1.5 の JDBC Thin Driver で LOB を操作すると、次のエラーが表示されます。

```
"Dumping lob java.sql.SQLException: ORA-03115: サポートされていないネットワークのデータ型または表現があります。"
```

これらのサポートされていない機能、代替および改善された JDBC メソッドの説明は、『Oracle8i JDBC 開発者ガイドおよびリファレンス』を参照してください。

Java での LOB の作業の詳細は、Oracle8i に付属している LOB のサンプル・コードを参照するか、www.oracle.com/java/jdbc から LOB 例を取得してください。

この章の内容は次のとおりです。

- LOB を使用する前に必要な DBA アクション
- LOB に対する基本操作での SQL DML の使用
- LOB 表領域の記憶域の変更
- テンポラリ LOB の管理
- LOB ロード時の SQL*Loader の使用
 - SQL*Loader を使用した行内および行外データの内部 LOB へのロード
 - ヒントをロードする SQL*Loader LOB
- LOB の制限
- 削除された制限

注意: この章で示す例は、第 8 章「サンプル・アプリケーション」で説明するマルチメディア・スキーマおよび Multimedia_tab 表を基にしています。

LOB を使用する前に必要な DBA アクション

オープンできる BFILE 数の上限の設定

1 回のセッションにつき同時にオープンできる BFILE の数には制限があります。初期化パラメータ `SESSION_MAX_OPEN_FILES` は、1 回のセッションで同時にオープンできるファイル数の上限を定義します。

このパラメータに対するデフォルト値は 10 です。デフォルト値を使用していれば、1 回のセッションにつき 10 までのファイルを同時にオープンできます。この制限を変更する場合、データベース管理者が `init.ora` ファイルのこのパラメータ値を変更します。たとえば、次のように設定します。

```
SESSION_MAX_OPEN_FILES=20
```

クローズされていないファイル数が `SESSION_MAX_OPEN_FILES` 値を超えると、そのセッションでは、それ以上のファイルをオープンできなくなります。すべてのオープン・ファイルをクローズするには、`FILECLOSEALL` コールを使用してください。

LOB に対する基本操作での SQL DML の使用

SQL データ操作言語 (DML) には、`INSERT`、`UPDATE`、`DELETE` など基本的な操作が含まれています。これらの操作を使用すると、Oracle RDBMS 内で内部 LOB の値全体を変更できます。

- **内部 LOB:** 内部 LOB の一部を操作する場合は、[第 3 章「LOB プログラム環境」](#) に説明されている、複雑な要件を処理するためのインタフェースを使用する必要があります。ユースケース例については、[第 9 章「内部永続 LOB」](#) の次の項を参照してください。
 - `INSERT`:
 - [9-23 ページの「EMPTY_CLOB\(\) または EMPTY_BLOB\(\) を使用した LOB 値の挿入」](#)
 - [9-27 ページの「別の表からの LOB の選択による行の挿入」](#)
 - [9-29 ページの「初期化した LOB ロケータ・バインド変数を使用した行の挿入」](#)
 - `UPDATE`:
 - [9-254 ページの「EMPTY_CLOB\(\) または EMPTY_BLOB\(\) を使用した LOB の更新」](#)
 - [9-257 ページの「別の表からの LOB の選択による行の更新」](#)

- 9-259 ページの「初期化した LOB ロケータ・バインド変数を使用した更新」
- DELETE:
 - 9-267 ページの「LOB を含む表の行の削除」
- 外部 LOB (BFILE) : Oracle8i では、外部 LOB の読み専用操作をサポートしています。詳細は、第 11 章「外部 LOB (BFILE)」を参照してください。
 - INSERT:
 - 11-24 ページの「BFILENAME() を使用した行の挿入」
 - 11-32 ページの「別の表からの BFILE の選択による BFILE 行の挿入」
 - 11-34 ページの「初期化した BFILE ロケータを使用した BFILE を含む行の挿入」
 - UPDATE: 次のメソッドを使用して、BFILE に対して更新または書き込みができます。
 - 11-181 ページの「BFILENAME() を使用した BFILE の更新」
 - 11-181 ページの「別の表からの BFILE の選択による BFILE の更新」
 - 11-183 ページの「初期化した BFILE ロケータを使用した BFILE の更新」
 - DELETE:
 - 11-214 ページの「BFILE を含む表の行の削除」

LOB 表領域の記憶域の変更

表の作成後、LOB のデフォルト記憶域を変更できます。

Oracle8 リリース 8.0.4.3

Oracle8 リリース 8.0.4.3 で表領域 A から表領域 B へ CLOB 列を移動させるには、次の文を実行します。

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

ただし、次のエラー・メッセージが戻されます。

```
ORA-02210: ALTER TABLE にオプションが指定されていません。
```

Oracle8i

- ALTER TABLE... MODIFY の使用 : LOB 表領域の記憶域を次のとおり変更できます。

注意： 既存の LOB 列を変更するための ALTER TABLE 構文は、LOB ... STORE AS 句ではなく、MODIFY LOB 句を使用します。LOB...STORE AS 句は、新しく追加された LOB 列に対してのみ使用されます。

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

- **ALTER TABLE ... MOVE の使用：**Oracle8i では、LOB 表領域の記憶域を変更するために ALTER TABLE 文の MOVE 句も使用できます。次に例を示します。

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

テンポラリ LOB の管理

テンポラリ LOB の管理およびセキュリティの問題は、[第 10 章「テンポラリ LOB」](#)の次の項を参照してください。

- 10-13 ページの「[テンポラリ LOB の管理](#)」
- 10-12 ページの「[テンポラリ LOB におけるセキュリティ上の問題](#)」

LOB ロード時の SQL*Loader の使用

LOB をバルクロードするために SQL*Loader を使用できます。LOB 型をロードする SQL*Loader 制御ファイル・データ定義言語 (DDL) の使用に関する詳細は、『Oracle8i ユーティリティ・ガイド』の第 5 章「SQL*Loader 制御ファイル・リファレンス」の「LOB のロード」を参照してください。

LOB にロードされたデータは大きくなる傾向があるため、このデータを残りのデータとは別に行外に置く必要がある場合があります。LOBFILES を使用することによって、長いデータを分割できます。

LOBFILES

LOBFILES は、LOB ロードを容易にする単純なデータ・ファイルです。LOBFILE は、レコードの概念がない LOBFILE の主データ・ファイルとは区別されています。LOBFILE のデータは次のいずれかです。

- 既定サイズ・フィールド (固定長フィールド)
- デリミタ付きフィールド (TERMINATED BY または ENCLOSED BY)

注意: PRESERVE BLANKS 句は、LOBFILE から読み込まれたフィールドには適用されません。

- Length-Value Pair フィールド (可変長フィールド) - VARRAW、VARCHAR または VARCHARC ローダー・データ型は、このフィールド型からロードするために使用されません。
- ファイル全体の内容を読み込める単一の LOB フィールド

注意: LOBFILE から読み込まれたフィールドは、句 (たとえば、NULLIF 句) への引数としては使用できません。

SQL*Loader を使用した行内および行外データの内部 LOB へのロード

次の項では、異なる方法でフォーマットされた行内および行外データを内部 LOB にロードする手順を説明します。

- 行内 LOB データのロード
 - 既定サイズ・フィールド内の行内 LOB データのロード
 - デリミタ付きフィールド内の行内 LOB データのロード
 - Length-Value Pair フィールド内の行内 LOB データのロード
- 行外 LOB データのロード
 - 1 ファイルにつき 1 つの LOB のロード
 - 既定サイズ・フィールド内の行外 LOB データのロード
 - デリミタ付きフィールド内の行外 LOB データのロード
 - Length-Value Pair フィールド内の行外 LOB データのロード

次の項目についても説明します。

- ヒントをロードする SQL*Loader LOB

SQL*Loader パフォーマンス : 内部 LOB へのロード

前述の方法でデータを内部 LOB にロードする際の関連パフォーマンスについては、[表 4-1 「SQL*Loader パフォーマンス : 内部 LOB へのデータのロード」](#) を参照してください。

表 4-1 SQL*Loader パフォーマンス : 内部 LOB へのデータのロード

行内または行外のデータのロード方法	関連パフォーマンス
既定サイズ・フィールド	最も高い
デリミタ付きフィールド	やや低い
Length-Value Pair フィールド	高い
1 ファイルにつき 1 つの LOB	高い

行内 LOB データのロード

- 既定サイズ・フィールド内の行内 LOB データのロード
- デリミタ付きフィールド内の行内 LOB データのロード
- Length-Value Pair フィールド内の行内 LOB データのロード

既定サイズ・フィールド内の行内 LOB データのロード

これは、非常に高速で簡単な LOB のロード方法です。ただし、ロードされる LOB は、同一サイズであるとは限りません。

注意： この問題を解決する方法の 1 つは、LOB データに空白スペースを埋め込み、特定のデータ・フィールド内のすべての LOB を同じ長さにする事です。後続の空白スペースの切捨てについては、『Oracle8i ユーティリティ・ガイド』の第 5 章「SQL*Loader 制御ファイル・リファレンス」の「空白とタブの切捨て」を参照してください。

このフォーマットを使用して LOB をロードするには、ロードするデータ型として CHAR または RAW のいずれかを使用します。次に例を示します。

制御ファイル

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

データ・ファイル (sample.dat)

```
007 Once upon a time
```

注意： 空白を 1 つ挿入することによって、Story の初めから Clip_ID (007) を分割できます。Story の長さは 15 バイトです。

Story を含むデータ・フィールドが空の場合、NULL の LOB ではなく空の LOB が作成されます。NULL の LOB は、DEFAULTIF directive ではなく NULLIF directive が使用される場合に生成されます。また、CHAR 以外のローダー・データ型を使用して LOB をロードすることも可能です。BLOB をロードする場合は、RAW データ型を使用してください。

デリミタ付きフィールド内の行内 LOB データのロード

異なるサイズの LOB を同じ列（データ・ファイル・フィールド）にロードすることは問題ではありません。このように融通がきくようになりますが、そのかわりにパフォーマンスが低下してしまいます。このフォーマットでロードするのは少し時間がかかります。ローダーはデータの中をスキャンし、デリミタ文字列を探すためです。次に例を示します。

制御ファイル

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

データ・ファイル (sample1.dat)

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

Length-Value Pair フィールド内の行内 LOB データのロード

VARCHAR (『Oracle8i ユーティリティ・ガイド』を参照)、VARCHARC、VARRAW の各データ型を使用して、この方法で編成された LOB データをロードすることができます。この方法でロードすると、前述の方法よりパフォーマンスは向上しますが、柔軟性は多少低下します。すなわち、ロードする前に各 LOB に対する LOB 長を知っておく必要があります。次に例を示します。

制御ファイル

```
LOAD DATA
INFILE 'sample2.dat' "str '<endrec>\n'"
INTO TABLE Multimedia_tab
FIELDS TERMINATED BY ','
(
Clip_ID INTEGER EXTERNAL (3),
Story VARCHARC (3, 500)
)
```

データ・ファイル (sample2.dat)

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

注意:

- Story は CLOB 列に対応するフィールドです。制御ファイルでは VARCHARC (3, 500) として記述され、長さフィールドは 3 バイトで、最大サイズは 500 バイトです。これによって、最初の 3 バイトで LOB データの長さが検索できることがローダーに伝えられます。
 - VARCHARC の長さサブフィールドは 0 (ゼロ) です (値サブフィールドは空です)。その結果、LOB インスタンスは空に初期化されます。
 - データ・ファイルの最後の文字は、必ずライン・フィールドにしてください。
-
-

行外 LOB データのロード

- 1 ファイルにつき 1 つの LOB のロード
- 既定サイズ・フィールド内の行外 LOB データのロード
- デリミタ付きフィールド内の行外 LOB データのロード
- Length-Value Pair フィールド内の行外 LOB データのロード

前述のとおり、LOB データは非常に大きくなる場合があるため、LOB を二次的なデータ・ファイルからロードすることは合理的です。

LOBFILE では、LOB データ・インスタンスはまだフィールド（既定サイズ、デリミタ付き、Length-Value Pair）の中にあることになっています。しかし、これらのフィールドはレコードの中に編成されていません（レコードという概念は LOBFILES にはありません）。したがって、レコードを扱う処理オーバーヘッドは回避されます。このタイプのデータ編成は、LOB のロードに理想的です。

1 ファイルにつき 1 つの LOB のロード

各 LOBFILE には、単一の LOB が含まれています。次に例を示します。

制御ファイル

```
LOAD DATA
INFILE 'sample3.dat'
INTO TABLE Multimedia_tab
REPLACE
FIELDS TERMINATED BY ','
(
  Clip_ID          INTEGER EXTERNAL(5),
  ext_FileName     FILLER CHAR(40),
  Story            LOBFILE(ext_FileName) TERMINATED BY EOF
)
```

データ・ファイル (sample3.dat)

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

2 次データ・ファイル (FirstStory.txt)

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

2 次データ・ファイル (SecondStory.txt)

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

注意:

- STORY は、ファイル名が ext_FileName フィールドに格納されているファイルで LOB データを検索できることをローダーに伝えます。
 - TERMINATED BY EOF は、LOB がファイル全体をまたがることをローダーに伝えます。
 - 詳細は、『Oracle8i ユーティリティ・ガイド』を参照してください。
-
-

既定サイズ・フィールド内の行外 LOB データのロード

制御ファイル内で、特定の列にロードされる LOB のサイズが指定されます。ロードの間、その特定の列にロードされるどの LOB データも指定されたサイズであることが想定されています。フィールドのサイズが既定されていると、データパーサーのパフォーマンスが向上します。ただし、すべての LOB が同一サイズであると保証することは困難です。次に例を示します。

制御ファイル

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

データ・ファイル (sample4.dat)

```
007,
008,
```

2 次データ・ファイル (FirstStory1.txt)

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

注意： SQL*Loader は、CHAR データ型を使用して FirstStory.txt LOBFILE から 2000 バイトのデータをロードします。現在のロード・セッションの間で最後にロードされたバイトの次のバイトからロードします。

デリミタ付きフィールド内の行外 LOB データのロード

LOBFILE ファイル内の LOB データ・インスタンスは、デリミタ付きです。このフォーマットでは、同じ列に異なるサイズの LOB をロードしても問題ありません。このように融通がきくようになりますが、そのかわりにパフォーマンスが低下してしまいます。このフォーマットでロードするのは少し時間がかかります。ローダーはデータの中をスキャンし、デリミタ文字列を探すためです。次に例を示します。

制御ファイル

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

データ・ファイル (sample5.dat)

```
007,
008,
```

2 次データ・ファイル (FirstStory2.txt)

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

注意: TERMINATED BY 句は LOB を終了する文字列を指定します。

また、ENCLOSED BY 句を使用することもできます。ENCLOSED BY 句によって、LOBFILE 中の LOB の相対的な位置指定がやや柔軟になります。LOBFILE 内の LOB は、必ずしも連続する必要はありません。

Length-Value Pair フィールド内の行外 LOB データのロード

LOBFILE 内の各 LOB の前には長さがあります。VARCHAR (『Oracle8 ユーティリティ・ガイド』を参照)、VARCHARC または VARRAW の各データ型を使用して、この方法で編成された LOB データをロードすることができます。Length-Value Pair 指定の LOB をロードするための制御可能な構文は、非常に単純です。

このような方法でロードすると、前述の方法よりパフォーマンスは向上しますが、柔軟性は多少低下します。すなわち、ロードする前に各 LOB の長さを知っておく必要があります。次に例を示します。

制御ファイル

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

データ・ファイル (sample6.dat)

```
007,
008,
```

2 次データ・ファイル (FirstStory3.txt)

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

注意： VARCHARC (4, 2000) は、LOBFILE 内の LOB が Length-Value Pair フォーマットであり、最初の 4 バイトは長さとして解析する必要があることをローダーに伝えます。max_length 部分 (2000) は、ローダーにフィールドの最大長についてヒントを与えます。

- 0031 は、次の 31 文字が指定された LOB に属することをローダーに伝えます。
 - 0000 は空の LOB になります (NULL の LOB ではありません)。
-
-

ヒントをロードする SQL*Loader LOB

- 特定の LOB のロードに失敗しても、その LOB を含むレコードが拒否されることはありません。ただし、そのレコードには空の LOB が含まれます。
- LOBFILE からロードする場合は、LOB 型列に対応するフィールドの最大長を指定してください。ただし、最大長が指定されると、メモリー使用率を最適化するヒントになります。最大長を指定するときは、実際の最大長を低く見積もらないことが特に重要です。

参照： 詳細は、『Oracle8i ユーティリティ・ガイド』を参照してください。

LOB の制限

LOB の使用には、いくつかの制限があります。

- **サポートされていない分散 LOB** SELECT 句および WHERE 句では、リモート・ロケータを使用できません。これには、DBMS_LOB パッケージ・ファンクションの使用も含まれます。さらに、LOB 属性の有無にかかわらず、リモート表のオブジェクトの参照もできません。

無効な操作 たとえば、次の操作は無効です。

```
- 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;
```

有効な操作 リモート表の LOB 列に対する次の操作は有効です。

```
- 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...
- UPDATE table1@remote...
- DELETE table1@remote...
```

- **LOB をサポートしていない表タイプおよび句**

LOB は、次の表タイプおよび句ではサポートされません。

- クラスタ化表および LOB は、クラスタ・キーにはなることはできません。
- GROUP BY、ORDER BY、SELECT DISTINCT、集約および JOINS

ただし、UNION ALL は、LOB がある表では使用できます。UNION、MINUS および SELECT DISTINCT は、オブジェクト型が MAP 関数または ORDER 関数を持つ場合は、LOB 属性で使用できます。

- 索引構成表

ただし、LOB は非パーティション化索引構成表ではサポートされます。

- VARRAY

- 表を作成する場合に、NCLOB をオブジェクト型で属性として使用することはできませんが、NCLOB パラメータをメソッドで使用することはできます。NCLOB は、固定幅データを格納します。

■ ANALYZE および ESTIMATE

LOB は、ANALYZE...COMPUTE または ESTIMATE STATISTICS 文ではサポートされていません。

■ トリガー本体

次の条件を満たしている場合は、LOB 列または属性をトリガー本体に使用できます。一般的に、トリガーにバインドされた `:new` 値および `:old` LOB 値は読み込み専用で、LOB に書き込めません。詳細は、次のとおりです。

- a. 行前トリガーと行後トリガーの中では、
 - * どちらのトリガーでも、LOB の `:old` 値を読み込みます。
 - * LOB の `:new` 値は、行後トリガーの中でのみ読み込みます。
- b. ビューの INSTEAD OF トリガーでは、`:new` 値および `:old` 値の両方を読み込みます。
- c. OF 句の中では LOB 列を指定できません (BFILE は、基礎となる表を更新しなくても、変更できることに注意してください)。
- d. OCI 関数または DBMS_LOB ルーチンを使用してオブジェクト列上の LOB 値または LOB 属性を更新している場合、列や属性を含む表に定義されているトリガーをその関数やルーチンで起動することはできません。

参照： ドメイン・インデックスに対するトリガーの起動については、『Oracle8i データ・カートリッジ開発者ガイド』を参照してください。

■ クライアント側の PL/SQL プロシージャ

これは、DBMS_LOB パッケージ・ルーチンをコールしない場合があります。ただし、サーバー側の PL/SQL プロシージャまたは Pro*C/C++ の無名ブロックを使用して DBMS_LOB パッケージ・ルーチンをコールすることはできます。

■ 外部 LOB (BFILE) の読み込み専用サポート

Oracle8i では、外部 LOB の読み込み専用操作をサポートしています。外部 LOB を更新または書き込みする場合は、ユーザーのニーズに合ったクライアント側のアプリケーションを開発する必要があります。

■ CACHE / NOCACHE / CACHE READS

このリリースでは、CACHE READS LOB がサポートされています。CACHE READS LOB を使用し、8.0 または 8.1.5 にダウングレードする場合、ご使用の CACHE READS LOB は警告を生成し、CACHE LOGGING LOB になります。ご使用の LOB を CACHE LOGGING にしない場合は、後で明示的に LOB の記憶域特性を変更できます。

詳細は、7-8 ページの「[CACHE / NOCACHE / CACHE READS](#)」を参照してください。

削除された制限

次の制限が削除されました。

4,000 バイトより大きいデータのバインディング

Oracle8i は、現在 INSERT および UPDATE 文で、内部 LOB 列への 4,000 バイトより大きいデータのバインディングをサポートしています。

- 表に、LONG および LOB 列がある場合、LONG または LOB 列のいずれかに対して 4,000 バイトより大きいデータをバインドできますが、同一の文で両方の列に対してバインドすることはできません。
- ADT では、LOB 属性に対してどのようなサイズのデータもバインドできません。前回のリリースからのこの制限は、今回も適用されます。LOB 属性には、まず空の LOB ロケータを挿入し、次にプログラム環境インタフェースの 1 つを使用して LOB の内容を変更してください。
- INSERT AS SELECT 操作では、LOB 列にはどのような長さのデータのバインドもできません。前回のリリースからのこの制限は、今回も適用されます。

高度なトピック

この章では、後の章で説明されているユースケースを補足し、詳細を説明します。一度ユースケースに目を通しておくと、ここで説明されているトピックが理解しやすくなるでしょう。

- 読取り一貫性のあるロケータ
 - 読取り一貫性のあるロケータになる、SELECT されたロケータ
 - 更新済ロケータを介した更新済 LOB
 - SQL DML および DBMS_LOB を使用した LOB の更新例
 - 同じ LOB 値を更新するために1つのロケータを使用する例
 - PL/SQL (DBMS_LOB) バインド変数を使用した LOB の更新の例
 - 複数のトランザクションにまたがることはできない LOB ロケータ
 - LOB ロケータとトランザクション境界
- オブジェクト・キャッシュ内の LOB
- LOB バッファリング・サブシステム
 - LOB バッファリングのメリット
 - LOB バッファリングの使用上のガイドライン
 - LOB バッファリングの使用についての注意
 - OCI: LOB バッファリングの例
- LOB への参照を含む VARRAY の作成

注意: この章に示す例は、第8章「サンプル・アプリケーション」で説明するマルチメディア・スキーマおよび Multimedia_tab 表を基にしています。

読取り一貫性のあるロケータ

Oracle では、スカラー量に対する他のデータベースの読み込みおよび更新処理と同様の、読取り一貫性メカニズムを LOB に対して提供しています。読取り一貫性に関する一般的な説明は、『Oracle8i 概要』を参照してください。ただし、LOB ロケータの場合、読取り一貫性にはいくつかの特別な用途があるため、正しく理解しておく必要があります。

読取り一貫性のあるロケータになる、SELECT されたロケータ

FOR UPDATE 句の有無にかかわらず、SELECT されたロケータは読取り一貫性のあるロケータとなり、LOB 値がそのロケータによって更新されるまでは読取り一貫性のあるロケータとして存在します。読取り一貫性のあるロケータには、SELECT 実行時点のスナップショット環境が含まれます。

これには複雑な意味があります。たとえば、SELECT 操作によって読取り一貫性のあるロケータ (L1) を作成したとします。L1 によって内部 LOB 値を読み込むときは、次の点に注意してください。

- SELECT 文に FOR UPDATE が含まれていても、SELECT 文実行時点の LOB が読み込まれます。
- 同じトランザクションの別のロケータ L2 によって LOB 値が更新されても、L1 では L2 の更新が認識されません。
- L1 では、別のトランザクションによって LOB にコミットされた更新も認識されません。
- 読取り一貫性のあるロケータ L1 が別のロケータ L2 にコピーされた場合 (たとえば、2 つのロケータ変数の PL/SQL 割当て L2:= L1 によって)、L2 は L1 と同様に読取り一貫性のあるロケータとなり、読み込まれるデータは L1 に対する SELECT 実行時点のデータとなります。

複数のロケータが存在することを利用して、LOB 値の異なる変換にアクセスできます。ただし、この場合、どのロケータでどの値にアクセスしているかを常に理解しておくように注意します。

LOB の更新および読取り一貫性

読取り一貫性のあるロケータを使用した更新の例

SELECT の実行時期に関係なく同じ LOB 値を提供する読取り一貫性のあるロケータ

次のコードは、読取り一貫性と更新の関係を簡単な例で示しています。第 8 章「サンプル・アプリケーション」で定義した `Multimeia_tab` および PL/SQL を使用して、次の 3 つの CLOB がロケータとして作成されます。

- `clob_selected`
- `clob_update`
- `clob_copied`

次のコードでは、`t1` から `t6` までが次のように処理されます。

- 最初の `SELECT INTO` の実行時 (`t1`) に、`story` 内の値がロケータ `clob_selected` に対応付けられます。
- 次の操作 (`t2`) では、`story` 内の値がロケータ `clob_updated` に対応付けられます。`t1` と `t2` では `story` の値が変わらないため、`clob_selected` と `clob_updated` の両方のロケータは、異なる時点でのスナップショットを反映しているにもかかわらず、同じ値を持つ読取り一貫性のあるロケータとなります。
- 3 つ目の操作 (`t3`) では、`clob_selected` の値が `clob_copied` にコピーされます。この時点で、3 つのロケータが同じ値になります。例では、一連の `DBMS_LOB.READ()` コールがこれを示しています。
- `t4` で、プログラムは `DBMS_LOB.WRITE()` を使用して `clob_updated` 内の値を変更します。`DBMS_LOB.READ()` が新しい値を示します。
- ただし、`clob_selected` を介した値の `DBMS_LOB.READ()` は (`t5`)、これが読取り一貫性のあるロケータであることを示します。`SELECT` の発行時と同じ値が、引き続き参照されます。
- 同様に、`clob_copied` を介した値の `DBMS_LOB.READ()` は (`t6`)、これが読取り一貫性のあるロケータであることを示します。`clob_selected` と同じ値が、引き続き参照されます。

例

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

COMMIT;

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

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

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

    -- t3時:
    clob_copied := clob_selected;
    -- 割り当てた後、clob_copiedおよびclob_selectedは、SELECT INTO clob_selectedの
    -- 実行時と同じスナップショットを持ちます。

    -- clob_selectedおよびclob_copiedからの読み込みを行うと、同じLOB値が戻されます。
    -- また、clob_updatedも、clob_selectedと同じLOB値を参照します。
    read_amount := 10;
    read_offset := 1;
    dbms_lob.read(clob_selected, read_amount, read_offset,
        buffer);
    dbms_output.put_line('clob_selected value: ' || buffer);
    -- 出力 'abcd' が生成されます。

    read_amount := 10;
    dbms_lob.read(clob_copied, read_amount, read_offset, buffer);
```

```
dbms_output.put_line('clob_copied value: ' || buffer);
-- 出力 'abcd' が生成されます。

read_amount := 10;
dbms_lob.read(clob_updated, read_amount, read_offset, buffer);
dbms_output.put_line('clob_updated value: ' || buffer);
-- 出力 'abcd' が生成されます。

-- t4時:
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);
-- 出力 'abcdefg' が生成されます。

-- t5時:
read_amount := 10;
dbms_lob.read(clob_selected, read_amount, read_offset,
              buffer);
dbms_output.put_line('clob_selected value: ' || buffer);
-- 出力 'abcd' が生成されます。

-- t6時:
read_amount := 10;
dbms_lob.read(clob_copied, read_amount, read_offset, buffer);
dbms_output.put_line('clob_copied value: ' || buffer);
-- 出力 'abcd' が生成されます。

END;
/
```

更新済ロケータを介した更新済 LOB

LOB ロケータ (L1) によって内部 LOB の値を更新するとき、L1 (ロケータ自体) は更新され、ロケータ L1 による LOB 値の操作が完了した時点での、現行スナップショット環境を持ちます。このため、L1 は更新済ロケータと呼ばれます。この操作によって、LOB 値に対して行った変更を、同じロケータ L1 による次の読み込み時に参照できます。

注意： 単に LOB 値を読み込むためにロケータを使用した場合、ロケータ内のスナップショット環境は更新されません。PL/SQL DBMS_LOB パッケージまたは OCI LOB API によってロケータを介して LOB 値を変更した場合に限り、更新されます。

別のトランザクションによってコミットされた更新は、そのトランザクションがコミット読み込みトランザクションであり、他のトランザクションのコミット後に L1 を使用して LOB 値を更新する場合にのみ、L1 によって認識されます。

注意： 内部 LOB の値を更新すると、最新の LOB 値が常に変更されます。

OCI LOB API または PL/SQL DBMS_LOB パッケージなどの使用可能なメソッドによって内部 LOB の値を更新することは、LOB 値を更新して、新しい LOB 値を参照するロケータを再選択するということです。

SQL による LOB 値の更新は、単なる UPDATE 文にすぎません。ロケータが UPDATE 文による変更を認識できるようにするには、LOB ロケータを再選択するか、UPDATE 文の RETURNING 句を使用するかをいずれかを実行してください。LOB ロケータを再選択しない場合、または RETURNING 句を使用しない場合、LOB 値が更新されていない状態での最新の値を読み込むことになります。このため、OCI を使用した SQL DML と DBMS_LOB の区分的操作を混合しないようにしてください。

参照： 詳細は、『Oracle8i PL/SQL ユーザーズ・ガイドおよびリファレンス』を参照してください。

SQL DML および DBMS_LOB を使用した LOB の更新例

先に定義した Multimedia_tab 表を使用して、CLOB ロケータを作成します。

- clob_selected

次の PL/SQL (DBMS_LOB) コード例では、t1 から t3 までが次のように処理されます。

- 最初の SELECT INTO の実行時 (t1) に、story 内の値がロケータ clob_selected に対応付けられます。

- 次の操作 (t2) では、story 内の値が clob_selected ロケータをバイパスして、SQL UPDATE 文によって変更されます。ロケータは元の SELECT 時点での LOB 値を参照します。つまり、ロケータは SQL UPDATE 文によって実行された更新を認識しません。例では、後続の DBMS_LOB.READ() コールがこれを示しています。
- 3つ目の操作 (t3) では、LOB 値がロケータ clob_selected に再選択されます。これによって、ロケータは最新のスナップショット環境に更新され、先の SQL UPDATE 文によって行われた変更を認識できるようになります。このため、次の DBMS_LOB.READ() では、LOB 値が空である (データがない) ことについてエラーが戻されます。

例

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

COMMIT;

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

BEGIN

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

    read_amount := 10;
    read_offset := 1;
    dbms_lob.read(clob_selected, read_amount, read_offset,
        buffer);
    dbms_output.put_line('clob_selected value: ' || buffer);
    -- 出力 'abcd' が生成されます。

    -- t2時:
    UPDATE Multimedia_tab SET story = empty_clob()
        WHERE clip_id = 1;
    -- ほとんどの現行 LOB 値が空であるにもかかわらず、clob_selected は SELECT の
    -- 実行時と同じ LOB 値を参照します。
```

```
read_amount := 10;
dbms_lob.read(clob_selected, read_amount, read_offset,
  buffer);
dbms_output.put_line('clob_selected value: ' || buffer);
-- 出力 'abcd' が生成されます。

-- t3 時:
SELECT story INTO clob_selected FROM Multimedia_tab WHERE
  clip_id = 1;
-- SELECT によって、clob_selected がほとんどの現行 LOB 値を参照できます。

read_amount := 10;
dbms_lob.read(clob_selected, read_amount, read_offset,
  buffer);
-- エラー : ORA-01403: データが見つかりません。
END;
/
```

同じ LOB 値を更新するために 1 つのロケータを使用する例

注意：異なるロケータを使用して同じ LOB を更新しないでください。同じ LOB 値を変更する場合、1 つのロケータを使用することで多くの問題を回避できます。

先に定義した Multimedia_tab 表を使用して、次の 2 つの CLOB をロケータとして作成します。

- clob_updated
- clob_copied

次の PL/SQL (DBMS_LOB) コード例では、t1 から t5 ままでが次のように処理されます。

- 最初の SELECT INTO の実行時 (t1) に、story 内の値がロケータ clob_selected に対応付けられます。
- 次の操作 (t2) では、clob_updated の値が clob_copied にコピーされます。この時点では、両方のロケータが同じ値を参照します。例では、一連の DBMS_LOB.READ() コールがこれを示しています。

- この時点で (t3)、プログラムは DBMS_LOB.WRITE() を使用して clob_updated 内の値を変更します。DBMS_LOB.READ() が新しい値を示します。
- ただし、clob_copied を介した値の DBMS_LOB.READ() は (t4)、clob_updated から割り当てられた時点 (t2) の LOB の値を参照しています。
- clob_updated が clob_copied に割り当てられた時点 (t5) で初めて、clob_copied は clob_updated による更新があったことを認識します。

例

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

COMMIT;

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

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

-- t2時 :
clob_copied := clob_updated;
-- 割り当てた後、clob_copied および clob_updated は同じ LOB 値を参照します。

read_amount := 10;
read_offset := 1;
dbms_lob.read(clob_updated, read_amount, read_offset, buffer);
dbms_output.put_line('clob_updated value: ' || buffer);
-- 出力 'abcd' が生成されます。

read_amount := 10;
```

```
dbms_lob.read(clob_copied, read_amount, read_offset, buffer);
dbms_output.put_line('clob_copied value: ' || buffer);
-- 出力 'abcd' が生成されます。

-- t3時:
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);
-- 出力 'abcdefg' が生成されます。

-- t4時:
read_amount := 10;
dbms_lob.read(clob_copied, read_amount, read_offset, buffer);
dbms_output.put_line('clob_copied value: ' || buffer);
-- 出力 'abcd' が生成されます。

-- t5時:
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);
-- 出力 'abcdefg' が生成されます。
END;
/
```

PL/SQL (DBMS_LOB) バインド変数を使用した LOB の更新の例

別の内部 LOB を更新するためのソースとして LOB ロケータを使用する場合 (SQL INSERT 文または UPDATE 文、DBMS_LOB.COPY() ルーチンなど)、ソース LOB ロケータ内のスナップショット環境によって、ソースとして使用される LOB 値が決まります。ソース・ロケータ (たとえば L1) が読取り一貫性のあるロケータの場合、L1 の SELECT 実行時点の LOB 値が使用されます。ソース・ロケータ (たとえば L2) が更新済ロケータの場合、更新時における L2 のスナップショット環境に対応付けられた LOB 値が使用されます。

先に定義した Multimedia_tab 表を使用して、次の3つの CLOB をロケータとして作成します。

- clob_selected
- clob_updated
- clob_copied

次のコード例では、t1 から t5 ままでが次のように処理されます。

- 最初の SELECT INTO の実行時 (t1) に、story 内の値がロケータ clob_updated に対応付けられます。
- 次の操作 (t2) では、clob_updated の値が clob_copied にコピーされます。この時点では、両方のロケータが同じ値を参照します。
- この時点で (t3)、プログラムは DBMS_LOB.WRITE () を使用して clob_updated 内の値を変更します。DBMS_LOB.READ () が新しい値を示します。
- ただし、clob_copied を介した値の DBMS_LOB.READ は (t4)、clob_copied が clob_updated による変更を参照しないことを示します。
- このため (t5 の時点で)、clob_copied を INSERT 文の値のソースとして使用する場合、clob_copied と対応付けられた値が挿入されます (clob_updated による新規の変更は反映されません)。これは、その後の、挿入されたばかりの値に対する DBMS_LOB.READ () によってわかります。

例

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

```
COMMIT;
```

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

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

read_amount := 10;
read_offset := 1;
dbms_lob.read(clob_updated, read_amount, read_offset, buffer);
dbms_output.put_line('clob_updated value: ' || buffer);
-- 出力 'abcd' が生成されます。

-- t2時 :
clob_copied := clob_updated;

-- t3時 :
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);
-- 出力 'abcdefg' が生成されます。
-- clob_copiedは、clob_updatedより前に行われた書込みを参照しないことに
-- 注意してください。

-- t4時 :
read_amount := 10;
dbms_lob.read(clob_copied, read_amount, read_offset, buffer);
dbms_output.put_line('clob_copied value: ' || buffer);
-- 出力 'abcd' が生成されます。

-- t5時 :
-- INSERT文では、clob_updatedによる変更を含まないLOB値のclob_copiedビューが
-- 使用されます。
INSERT INTO Multimedia_tab VALUES (2, clob_copied, EMPTY_CLOB(), NULL,
      EMPTY_BLOB(), EMPTY_BLOB(), NULL, NULL, NULL, NULL)
      RETURNING story INTO clob_selected;
```

```
read_amount := 10;
dbms_lob.read(clob_selected, read_amount, read_offset,
             buffer);
dbms_output.put_line('clob_selected value: ' || buffer);
-- 出力 'abcd' が生成されます。
END;
/
```

複数のトランザクションにまたがることはできない LOB ロケータ

DBMS_LOB、OCI、SQL INSERT 文または UPDATE 文を使用して、LOB ロケータによって内部 LOB の値を更新すると、読取り一貫性のあるロケータが更新済ロケータに変更されます。さらに、INSERT 文や UPDATE 文によって、トランザクションが自動的に開始され、行がロックされます。一度これが発生すると、ロケータを現行トランザクション以外で使用して、LOB 値を変更できなくなる可能性があります。つまり、データの書込みに使用する LOB ロケータを複数のトランザクションにまたがって使用することはできません。ただし、シリアライズ可能トランザクション内でなければ、ロケータを使用して LOB 値を読み込むことができます。

参照： LOB とトランザクション境界の関係については、5-15 ページの「[LOB ロケータとトランザクション境界](#)」を参照してください。

先に定義した Multimedia_tab 表を使用して、CLOB ロケータ clob_updated を作成します。

- SELECT INTO の最初の操作 (t1) では、story 内の値がロケータ clob_updated に対応付けられます。
- 次の操作 (t2) では、DBMS_LOB.WRITE() コマンドを使用して clob_updated 内の値を変更します。DBMS_LOB.READ() が、新しい値を示します。
- COMMIT 文 (t3) によって現行のトランザクションが終了します。
- トランザクションを終了すると (t4)、clob_updated ロケータが別のトランザクション (コミット済) を参照することになるため、次の DBMS_LOB.WRITE() 操作が失敗します。これは、戻されるエラーによって通知されます。この LOB ロケータをさらに DBMS_LOB (および OCI) の変更操作で使用するには、再選択する必要があります。

トランザクションにまたがらないロケータの例

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

COMMIT;
```

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

BEGIN
    -- t1時:
    SELECT      story
    INTO        clob_updated
    FROM        Multimedia_tab
    WHERE       clip_id = 1
    FOR UPDATE;
    read_amount := 10;
    read_offset := 1;
    dbms_lob.read(clob_updated, read_amount, read_offset,
                 buffer);
    dbms_output.put_line('clob_updated value: ' || buffer);
    -- 出力 'abcd' が生成されます。

    -- t2時:
    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);
    -- 出力 'abcd' が生成されます。

    -- t3時:
    COMMIT;

    -- t4時:
    dbms_lob.write(clob_updated, write_amount, write_offset,
                  buffer);
    -- エラー: ORA-22990: LOB ロケータは複数のトランザクションにまたがることはできません。
END;
/
```

LOB ロケータとトランザクション境界

LOB ロケータおよび LOB 操作の基本については、第 2 章「基本コンポーネント」を参照してください。

この項では、トランザクションでの LOB ロケータの使用、およびトランザクション ID について説明します。

ロケータがトランザクション ID を含む場合

- トランザクションを開始してからロケータを選択した場合：トランザクションを開始してからロケータを選択すると、ロケータにはトランザクション ID が含まれます。明示的にトランザクションを開始しなくても、暗黙的にトランザクションが開始されている場合もあることに注意してください。たとえば、SELECT ... FOR UPDATE は、暗黙的にトランザクションを開始します。このような場合、ロケータはトランザクション ID を含んでしまいます。

ロケータがトランザクション ID を含まない場合

- ユーザーが、トランザクションの外からロケータを選択した場合：トランザクション外のロケータを選択しても、ロケータはトランザクション ID を含みません。
- ロケータは、DML 文実行の前に選択されるとトランザクション ID を含みません。トランザクション ID は、最初の DML 文が実行されるまで割り当てられません。このような DML 文の前に選択されたロケータは、トランザクション ID を含みません。

トランザクション ID: ロケータを使用した LOB に対する読み込みおよび書き込み

ロケータがトランザクション ID を含んでいるかどうかにかかわらず、常にロケータを使用して LOB データを読み込むことができます。

- ロケータを使用して書き込みできない場合：ロケータがトランザクション ID を含む場合、その特定のトランザクション外で LOB に書き込むことはできません。
- ロケータを使用して書き込みできる場合：ロケータがトランザクション ID を含まない場合、トランザクションを明示的または暗黙的に開始した後で、LOB に書き込むことができます。
- シリアライズ可能なトランザクションがあるロケータを使用して読み込みまたは書き込みできない場合：ロケータが古いトランザクションのトランザクション ID を含み、現行のトランザクションがシリアライズ可能な場合、そのロケータを使用して読み込みまたは書き込みはできません。
- シリアライズ可能でないトランザクションがあるロケータを使用して読み込みできるが、書き込みできない場合：トランザクションがシリアライズ可能でない場合、そのトランザクション外で読み込むことはできますが、書き込むことはできません。

次の例で、ロケータとシリアライズ可能でないトランザクションの関係を示します。

シリアライズ可能でない例：現在のトランザクションを持たないロケータを選択する

ケース 1

1. 現行トランザクションを持たないロケータを選択します。この時点ではロケータはトランザクション ID を含みません。
2. トランザクションを開始します。
3. ロケータを使用して LOB からデータを読み込みます。
4. トランザクションをコミットまたはロールバックします。
5. ロケータを使用して LOB からデータを読み込みます。
6. トランザクションを開始します。ロケータはトランザクション ID を含みません。
7. ロケータを使用してデータを LOB に書き込みます。この操作は、ロケータが書き込みの前にトランザクション ID を含まないため有効です。このコールの後、ロケータはトランザクション ID を含むようになります。

ケース 2

1. 現行トランザクションを持たないロケータを選択します。この時点ではロケータはトランザクション ID を含みません。
2. トランザクションを開始します。ロケータはトランザクション ID を含みません。
3. ロケータを使用して LOB からデータを読み込みます。ロケータはトランザクション ID を含みません。
4. ロケータを使用してデータを LOB に書き込みます。この操作は、ロケータが書き込みの前にトランザクション ID を含まないため有効です。このコールの後、ロケータはトランザクション ID を含むようになります。続けて LOB の読み込みまたは書き込み（あるいはその両方）を行うことができます。
5. トランザクションをコミットまたはロールバックします。ロケータは引き続きトランザクション ID を含みます。
6. ロケータを使用して LOB からデータを読み込みます。この操作は有効です。
7. トランザクションを開始します。ロケータはすでに前のトランザクションの ID を含んでいます。

8. ロケータを使用してデータを LOB に書き込みます。ロケータが現行トランザクションに一致するトランザクション ID を含まないため、この書き込み操作は失敗します。

シリアル化可能でない例：トランザクション内でロケータを選択する

ケース 3

1. トランザクションの中でロケータを選択します。この時点では、ロケータはトランザクション ID を含んでいます。
2. トランザクションを開始します。ロケータは前のトランザクションの ID を含んでいます。
3. ロケータを使用して LOB からデータを読み込みます。ロケータの中のトランザクション ID は現在のトランザクションに一致していませんが、この操作は有効です。

参照： ロケータを使用した LOB データの読み込みの詳細は、5-2 ページの「[読取り一貫性のあるロケータ](#)」を参照してください。

4. ロケータを使用してデータを LOB に書き込みます。ロケータの中のトランザクション ID が現在のトランザクションに一致していないため、この操作は失敗します。

ケース 4

1. トランザクションを開始します。
2. ロケータを選択します。ロケータがトランザクションの中で選択されたため、トランザクション ID が含まれています。
3. ロケータを使用して LOB の読み込みまたは書き込み（あるいはその両方）を行います。これらの操作は有効です。
4. トランザクションをコミットまたはロールバックします。ロケータは引き続きトランザクション ID を含みます。
5. ロケータを使用して LOB からデータを読み込みます。ロケータの中にトランザクション ID があり、トランザクションはすでにコミットまたはロールバックされていますが、この操作は有効です。

参照： ロケータを使用した LOB データの読み込みの詳細は、5-2 ページの「[読取り一貫性のあるロケータ](#)」を参照してください。

6. ロケータを使用してデータを LOB に書き込みます。ロケータの中のトランザクション ID はすでにコミットまたはロールバックされたトランザクション用であるため、この操作は失敗します。

オブジェクト・キャッシュ内の LOB

- **内部 LOB 属性:** オブジェクト・キャッシュ内にオブジェクトを作成すると、LOB 属性は空に設定される

内部 LOB 属性を持つオブジェクト・キャッシュ内にオブジェクトを作成すると、その LOB 属性は暗黙的に空に設定されます。この空の LOB ロケータを使用して、データを書き込むことはできません。最初にオブジェクトをフラッシュし、その後、表に行を挿入し、空の LOB (長さ 0 (ゼロ) の LOB) を作成する必要があります。オブジェクトがオブジェクト・キャッシュ内でリフレッシュされ (OCI_PIN_LATEST を使用)、実際の LOB ロケータが属性に読み込まれると、OCI LOB API をコールして LOB にデータを書き込むことができます。

- **外部 LOB 属性:** オブジェクト・キャッシュ内にオブジェクトを作成すると、BFILE 属性は NULL に設定される

外部 LOB (BFILE) 属性を持つオブジェクトを作成すると、BFILE は NULL に設定されます。ファイルからの読み込みを始める前に、有効なディレクトリ別名とファイル名を使用して BFILE を更新しておく必要があります。

LOB ロケータ属性を持つオブジェクト・キャッシュ内で、あるオブジェクトを別のオブジェクトにコピーすると、LOB ロケータのみがコピーされます。これら 2 つの異なるオブジェクトの LOB 属性には、まったく同一の LOB 値を参照する、同一のロケータが含まれることとなります。ターゲット・オブジェクトがフラッシュされたときにのみ、LOB 値の個別の、物理的なコピーが作成されます。これは、ソースの LOB 値とは別の値となります。

参照: ロケータの 1 つを使用して書き込みが実行された場合、各オブジェクトでどのバージョンの LOB 値が参照されるかについては、5-3 ページの「[読取り一貫性のあるロケータを使用した更新の例](#)」を参照してください。

したがって、コピーのターゲットになった LOB を変更するには、**ターゲット・オブジェクト** をフラッシュしてリフレッシュし、ロケータ属性を介して LOB に書き込む必要があります。

LOB バッファリング・サブシステム

Oracle8i では、LOB バッファリング・サブシステム (LBS) を提供しており、DataCartridge、Web サーバー、その他のクライアント・ベースのアプリケーションなど、1 つ以上の LOB の内容をクライアントのアドレス空間にバッファリングする必要のある OCI ベースの高度なアプリケーションに対応しています。クライアント側のバッファリング・サブシステムに必要なメモリー量は、最大使用時で 512KB です。これは、バッファリングが使用可能な LOB での 1 回の読み込みまたは書き込み操作に対して指定できる最大サイズでもありません。

LOB バッファリングのメリット

バッファリングは、LOB の特定の部分に一連の小規模な読み込みおよび書き込みを実行する（繰り返し行われる場合が多い）クライアント・アプリケーションについて、次の点で特に有効です。

- バッファリングによってサーバーへの遅延書き込みが可能になります。LOB バッファへの書き込みをクライアントのアドレス空間に数回分バッファリングし、最後にサーバーへフラッシュできます。これによって、クライアント・アプリケーションとサーバー間のネットワーク・ラウンドトリップ数が減り、LOB 更新の全体的なパフォーマンスが向上します。
- バッファリングによって、サーバー上の LOB の総更新回数が減り、LOB のバージョン数とロギング量が減ります。これによって、全体的な LOB パフォーマンスおよびディスク使用率が向上します。

LOB バッファリングの使用上のガイドライン

バッファリングされた LOB 操作については次の点に注意する必要があります。

- Oracle8i で提供されるのは、単純なバッファリング・サブシステムで、キャッシュではありません。Oracle8i では、LOB バッファの内容とサーバーの LOB 値の同期は保証されていません。LOB バッファの内容を明示的にフラッシュしなければ、サーバーの実際の LOB に反映された、バッファリングを利用した書き込み結果を参照できません。
- バッファリングされた LOB 操作のエラー・リカバリは、ユーザーの責任で行う必要があります。実際の LOB の更新には遅れがあるため、バッファリングされた特定の読み込みまたは書き込み操作のエラー・レポートは、次のサーバー・ベースの LOB へのアクセスまで行われません。
- バッファリングされた LOB 操作が行われるトランザクションは、他のユーザー・セッションに移行できません。LBS はシングル・ユーザー、単一スレッドのシステムであるためです。

- Oracle8i では、バッファリングされた LOB 操作に対するトランザクション・サポートは保証されていません。バッファリングされた LOB 更新に対してトランザクション・セマンティクスを保証するには、アプリケーションで論理セーブポイントを保持して、エラーの際には、LOB に対するすべての変更がロールバックされるようにする必要があります。2つの論理セーブポイントの間に、バッファリングされた LOB の更新を常にラップする必要があります (5-26 ページの「OCI: LOB バッファリングの例」を参照)。
- バッファリングされた書込みを使用して LOB の更新を開始した後は、どのトランザクションでも、バッファリング・サブシステムを使用しない同一のトランザクション内で、他の操作によって同じ LOB が更新されないことを確認する必要があります。

この状況は、サーバー・ベースの LOB を更新する SQL 文を使用することで発生する可能性があります。Oracle8i ではこのような操作を区別できないため、防止することができません。これは、アプリケーションの正確さと整合性に影響します。
- LOB に対するバッファリング操作は、通常の場合と同様にロケータを介して実行されます。バッファリングを使用可能にしたロケータは、そのロケータを使用した LOB への書込み操作が実行されるまでは、読取り一貫性のある LOB を提供します。

参照： 5-2 ページの「[読取り一貫性のあるロケータ](#)」を参照してください。

バッファリングされた書込みに使用されることによってロケータが更新済ロケータになった後は、バッファリング・サブシステムを通してわかるように、常に最新バージョンの LOB へのアクセスが提供されます。この更新済ロケータについて、バッファリングによって発生するもう 1 つの重要な点は、それ以降の LOB へのバッファリングされた書込みが、この更新済ロケータからしか実行できないことです。Oracle8i では、バッファリングを使用可能にしたその他のロケータを介して LOB への書込みを行おうとすると、エラーが戻されます。

参照： 5-6 ページの「[更新済ロケータを介した更新済 LOB](#)」を参照してください。

- バッファリングを使用可能にした更新済ロケータは、PL/SQL プロシージャへの IN パラメータとして渡せます。ただし、IN OUT または OUT パラメータを渡すと、更新済ロケータを戻そうとした場合と同様に、エラーが発生します。
- バッファリングを使用可能にした更新済ロケータを別のロケータに割り当てることはできません。ロケータの割当ては、OCILOBAssign() の使用、PL/SQL 変数の割当て、オブジェクトが LOB 属性を含む場合は OCIObjectCopy() の使用などによって発生します。バッファリングを使用可能にしている読取り一貫性のあるロケータを、バッファリングを使用可能にしていないロケータに割り当てると、対象となるロケータに対してバッファリング機能がオンになります。同様に、バッファリングを使用可能にしていないロケータを、バッファリングを使用可能にしているロケータに割り当てると、対象となるロケータに対してバッファリング機能がオフになります。

また、もともとバッファリングを使用可能にしているロケータに対して SELECT を発行した場合、ロケータは新しいロケータ値で上書きされ、バッファリング機能はオフになります。

- バッファリングされた書込みを使用した LOB 値への追加は、これらの書込みを開始するオフセットが、BLOB（または CLOB/NCLOB）の最後から 1 バイト（または 1 文字）の場合にのみ可能です。言い換えると、バッファリング・サブシステムでは、サーバー・ベースの LOB 内に対してバイト値 0（ゼロ）の充てん文字または空白の作成が必要になるような追加はサポートされていません。
- CLOB の場合、Oracle8i では、クライアント側のロケータ・バインド変数のキャラクタ・セット・フォームとサーバーの LOB が同じである必要があります。ほとんどの OCI LOB プログラムでは、この条件が必要です。ロケータがリモート・データベースから SELECT されると、OCI プログラムが現在アクセスしているデータベースからのキャラクタ・セット・フォームと異なっている場合がありますが、これは例外です。この場合は、エラーが戻されます。ユーザーによってキャラクタ・セット・フォームが入力されていない場合、これは SQLCS_IMPLICIT とみなされます。

LOB バッファリングの使用についての注意

LOB バッファの物理構造

各ユーザー・セッションは、次のような構造になっています。

- 各ユーザー・セッションには、16 ページの固定ページ・プールがあります。このページは、そのセッションからバッファリング・モードでアクセスされるすべての LOB によって共有されます。
- 各ページ・サイズは、最大 32KB（文字ではない）の固定サイズです。ページ・サイズは、 $n \times \text{CHUNKSIZE} = \text{約 } 32\text{KB}$ です。

LOB バッファは、1 ページ以上のこのようなページで構成されます。1 セッションあたりの最大ページ数は 16 ページです。バッファリングされた読み込みまたは書き込み操作に指定できるサイズの最大値は 512KB で、それぞれの環境によって、読み込み / 書き込みの最大値が小さくなる場合があります。

LOB バッファリング・システム (LBS) の使用例

LOB は、固定サイズの論理リージョンに分割されることを考慮してください。各ページはこれらの固定サイズ・リージョンの1つにマップされ、基本的には、メモリー内コピーにもマップされます。Oracle8i では、入力オフセット、および読みまたは書き込み操作に指定した量に従って、ページ・プールから1ページ以上の空きページがLOBのバッファに割り当てられます。空きページとは、バッファリングされた読みまたは書き込み操作によって、読みまたは書き込みが行われていないページです。

たとえば、ページ・サイズを32KBとします。

- 入力オフセットが1000で、指定された読み / 書き込みの量が30000の場合、Oracle8iでは、LOBの最初の32KBのリージョンがLOBバッファ内のページに読み込まれます。
- 入力オフセットが33000で、読み / 書き込みの量が30000の場合、LOBの次の32KBのリージョンがページに読み込まれます。
- 入力オフセットが1000で、読み / 書き込みの量が35000の場合、LOBのバッファは2ページになります。最初の部分はLOBの1から32KBまでのリージョンに、次の部分は32KB + 1から64KBまでのリージョンにマップされます。

ページとLOBリージョンの間のこのマッピングは、Oracle8iによって別のリージョンがページにマップされるまでの一時的なものです。LOBのバッファ内で一杯になっていてもう使用できないLOBリージョンにアクセスしようとする、Oracle8iでは、ページ・プールから任意の使用可能な空きページがLOBのバッファに割り当てられます。ページ・プールに使用可能なページがない場合は、Oracle8iでは、次のようにページを再配置します。LOBバッファ内の未修正ページのうち、LRUページがページ・アウトされ、現在の操作に再配置されます。

LOBバッファ内にそのようなページがない場合は、同じセッションにある、バッファリングされた別のLOBの未修正ページのうち、LRUページがページ・アウトされます。それでも使用可能なページがない場合は、ページ・プールのすべてのページが使用済である（変更されている）ことを示しています。その場合、現在アクセス中のLOBまたは別のLOBをフラッシュする必要があります。Oracle8iでは、この状態をエラーとしてユーザーに通知します。Oracle8iでは、使用済ページが暗黙的にフラッシュされたり、再配置されることはありません。これらのページは、ユーザーが明示的にフラッシュしたり、LOBのバッファリング機能を使用禁止にして破棄できます。

前述の説明をわかりやすくするために、バッファリング・モードでL1およびL2の2つのLOBにアクセス中で、各LOBには8ページのバッファがある場合を考えてみます。L1のバッファの8ページのうち6ページが使用済で、残りの2ページには、サーバーから読み込まれた未修正データが入っているとします。L2のバッファの状態も同様であるとします。ここで、L1の次のバッファリング操作のために、Oracle8iによって、L1のバッファの未修正の2ページからLRUページが再配置されます。L1のバッファの8ページすべてがLOB書き込みのために使用されると、Oracle8iはLRUの方針を使用して、L2のバッファから未修正の2ページを再配置することによって、L1にさらに2つの操作を提供できます。ただし、L1またはL2でさらにバッファリング操作を実行しようとする、Oracle8iではエラーが戻されます。

バッファがすべて使用済で、バッファリングされたLOBに別の読み込みまたは書き込みを行うと、次のエラーが発生します。

ORA-22280: その操作に使用可能なバッファは、これ以上ありません。

これには、2つの原因が考えられます。

1. バッファ・プールにあるすべてのバッファが、以前の操作で使用された。
この場合、LOBの更新に使用されているロケータを介してLOBをフラッシュします。
2. 以前バッファリングされた更新操作がないのにLOBをフラッシュしようとしている。
この場合、バッファをフラッシュする前に、バッファリングが可能なロケータを介してLOBに書き込みます。

LOBバッファのフラッシュ

フラッシュとは、一連のプロセスを指します。ロケータを介してバッファ内のLOBにデータを書き込むと、そのロケータは、更新済ロケータになります。いったん、更新済ロケータを介してバッファのLOBデータを更新すると、フラッシュ・コールは、次のことを行います。

- LOBのバッファの使用済ページをサーバー・ベースのLOBに書き込み、それによってLOB値を更新します。
- 更新済ロケータをリセットして、読取り一貫性のあるロケータにします。
- フラッシュ済バッファを解放するか、バッファのページの状態を使用済から変更されていない状態に戻します。

フラッシュ後、ロケータは、読取り一貫性のあるロケータとなり、別のロケータに割り当てることができます (L2 := L1)。

たとえば、L1およびL2の2つのロケータがあるとします。両方とも読取り一貫性のあるロケータであり、サーバーのLOBデータの状態と一貫性があるとします。バッファに書き込みをしてLOBを更新すると、L1は更新済ロケータになります。L1およびL2は、現在、別のバージョンのLOB値を参照しています。サーバーのLOBを更新する場合、L2で得た読取り一貫性のある状態を維持するためにL1を使用する必要があります。フラッシュ操作によって、フラッシュに使用したロケータに新しいスナップショット環境が書き込まれます。重要な点は、LOBバッファをフラッシュするとき、更新済ロケータ (L1) を使用する必要があることです。読取り一貫性のあるロケータをフラッシュしようとする、エラーが発生します。

ここで、LOBバッファのデータに何が発生するのかという問題があります。2つの可能性があります。デフォルト・モードの場合、フラッシュ操作では、変更したページのデータが維持されます。この場合、同じバイト範囲に対する読み込みまたは書き込みでは、サーバーへのラウンドトリップは必要ありません。この状況でのフラッシュでは、バッファ・データはクリアされないことに注意してください。また、フラッシュ済バッファに占有されているメモリーは、クライアントのアドレス領域に戻されません。

注意： 変更されていないページは、必要に応じてページ・アウトされません。

2つ目のケースでは、OCILOBFlushBuffer() のフラグ・パラメータを OCI_LOB_BUFFER_FREE に設定して、バッファ・ページを解放すると、メモリーは、クライアントのアドレス領域に戻されます。この場合のフラッシュとは、サーバーのLOB値を更新し、読取り一貫性のあるロケータを戻し、バッファ・ページを解放することです。

更新済LOBのフラッシュ

LBSを使用して更新したLOBは、次の場合にフラッシュする必要があります。

- トランザクションをコミットする前
- 現行のトランザクションから別のトランザクションに移行する前
- LOBのバッファリング操作を使用禁止にする前
- 外部コールアウトの実行から PL/SQL のファンクション / プロシージャ / メソッドのコールに戻る前

注意： 外部コールアウトが、PL/SQL ブロックから呼び出され、ロケータがパラメータとして渡される場合、バッファリングを使用可能にする呼出しを含めてすべてのバッファリング操作をコールアウトの内部で行う必要があります。次の手順に従うことをお勧めします。

- 外部コールアウトを呼出します。
- ロケータをバッファリング可能にします。
- ロケータを使用して読み込み / 書き込みを行います。
- LOBをフラッシュします。
- ロケータをバッファリング使用禁止にします。
- PL/SQLの呼出し元のファンクション / プロシージャ / メソッドに戻ります。

Oracle8i では、LOB は暗黙的にはフラッシュされないことに注意してください。

バッファリング対応のロケータ

バッファリング対応のロケータを使用できる場合と使用できない場合があることに注意してください。

■ バッファリング対応ロケータを使用できる場合

- **OCI:** バッファリングを使用可能にしたロケータは、次の OCI API がある場合にのみ使用できます。

```
OCILOBRead(), OCILOBWrite(), OCILOBAssign(), OCILOBIsEqual(),  
OCILOBLocatorIsInit(), OCILOBCharSetId(), OCILOBCharSetForm()
```

■ バッファリング対応ロケータを使用できない場合

次の OCI API を、バッファリングを使用可能にしたロケータと一緒に使用すると、エラーが戻されます。

- **OCI:** OCILOBCopy(), OCILOBAppend(), OCILOBErase(), OCILOBGetLength(), OCILOBTrim(), OCILOBWriteAppend()

これらの API は、バッファリングが使用可能になっていないロケータとともに使用したときに、そのロケータが示す LOB がその他のロケータを介してバッファリング・モードですでにアクセスされている場合にも、エラーを戻します。

- **PL/SQL (DBMS_LOB) :** 入力 LOB ロケータでバッファリングを使用可能にすると、DBMS_LOB API からエラーが戻されます。
- その他のすべてのロケータの場合と同様に、LOB バッファリングを使用可能にしたロケータは、トランザクションにまたがることはできません。

再選択を避けるためのロケータ状態の保存

さらに LOB バッファに書き込む前に、LOB の現在の状態を保存するとします。LOB バッファリングの使用中に更新を実行すると、既存のバッファへの書き込みで、サーバーへのラウンドトリップは不要なため、ロケータのスナップショット環境はリフレッシュされません。これは、LOB バッファリングを使用しないで直接 LOB を更新する場合にはあてはまりません。その場合、更新するたびにサーバーへのラウンドトリップが必要となるため、ロケータのスナップショットはリフレッシュされます。

LOB バッファを使用して書き込まれた LOB の状態を保存するには、次の手順に従います。

1. LOB をフラッシュして、LOB およびロケータ (L1) のスナップショット環境を更新します。この時点では、ロケータ (L1) の状態と LOB は同じです。
2. フラッシュおよび更新に使用されたロケータ (L1) を別のロケータ (L2) に割り当てます。この時点では、2つのロケータの状態 (L1 および L2) と LOB はすべて同じです。

これで、L2 は読取り一貫性のあるロケータになり、フラッシュが行われるまでは L1 を介して行われた変更にはアクセスできますが、フラッシュ後はアクセスできません。この割り当てによって、ロケータを L2 に選択し直すためにサーバーへはアクセスせずに済みます。

OCI: LOB バッファリングの例

Multimedia_tab スキーマに基づく、次の OCI プログラム用の疑似コードは、前述の概念を説明しています。

```
OCI_BLOB_buffering_program()
{
    int          amount;
    int          offset;
    OCILobLocator lbs_loc1, lbs_loc2, lbs_loc3;
    void         *buffer;
    int          buf1;

    -- 標準 OCI 初期化操作 - サーバーにログインし、バインド変数の
    -- 作成や初期化などを行います。

    init_OCI();

    -- LBS 操作の開始前にセーブポイントを確立します。
    exec_statement("savepoint lbs_savepoint");

    -- バッファリングでのアクセスから、BLOB 列へのバインド変数を初期化します。
    exec_statement("select frame into lbs_loc1 from Multimedia_tab
        where clip_id = 12");
    exec_statement("select frame into lbs_loc2 from Multimedia_tab
        where clip_id = 12 for update");
    exec_statement("select frame into lbs_loc2 from Multimedia_tab
        where clip_id = 12 for update");

    -- バッファリング・モードでの LOB へのアクセスに対して、ロケータを使用可能にします。
    OCILobEnableBuffering(lbs_loc1);
    OCILobEnableBuffering(lbs_loc2);
    OCILobEnableBuffering(lbs_loc3);

    -- オフセット 1 から開始して、lbs_loc1 を介して 4KB 読み込みます。
    amount = 4096; offset = 1; buf1 = 4096;
    OCILobRead(..., lbs_loc1, offset, &amount, buffer, buf1,
        ..);
    if (exception)
        goto exception_handler;
    -- これによって、LOB の最初の 32KB が、サーバーから LOB の
    -- クライアント側バッファにあるページ (page_A といいます) に読み込まれます。
    -- lbs_loc1 は、読取り一貫性のあるロケータです。

    -- オフセット 1 から開始して、lbs_loc2 を介して 4K の LOB を書き込みます。
    amount = 4096; offset = 1; buf1 = 4096;
    buffer = populate_buffer(4096);
    OCILobWrite(..., lbs_loc2, offset, amount, buffer,
```

```
    buf1, ..);

if (exception)
    goto exception_handler;
-- これによって、LOBの最初の32KBが、サーバーからLOBの
-- バッファにある新しいページ (page_B といいます) に読み込まれ、
-- このページの内容が入力バッファの内容で変更されます。
-- lbs_loc2 は更新済ロケータです。

-- オフセット10Kから開始して、lbs_loc1 を介して20KBを読み込みます。
amount = 20480; offset = 10240;
OCILOBRead(.., lbs_loc1, offset, &amount, buffer,
    buf1, ..);

if (exception)
    goto exception_handler;
-- page_A からユーザー・バッファに直接読み込みます。
-- データがすでにクライアント側のバッファにあるため、サーバーへの
-- ラウンドトリップは発生しません。

-- オフセット10Kから開始して、lbs_loc2 を介して20KBを書き込みます。
amount = 20480; offset = 10240; buf1 = 20480;
buffer = populate_buffer(20480);
OCILOBWrite(.., lbs_loc2, offset, amount, buffer,
    buf1, ..);

if (exception)
    goto exception_handler;
-- この時点で、ユーザー・バッファの内容が page_B に書き込まれます。
-- この場合、サーバーへのラウンドトリップは発生しません。
-- これによって、サーバーでの新しいバージョンのLOBの作成、ログへの
-- REDO の書き込みが回避されます。

-- 次のような lbs_loc3 を介した書き込みでもエラーが発生します。
amount = 20000; offset = 1000; buf1 = 20000;
buffer = populate_buffer(20000);
OCILOBWrite(.., lbs_loc3, offset, amount, buffer,
    buf1, ..);

if (exception)
    goto exception_handler;
-- 2つのロケータを使用して、バッファリング・システムを介して
-- バッファリングされたLOBを更新できません。

-- 次のような lbs_loc3 を介した更新でもエラーが発生します。
OCILOBFileCopy(.., lbs_loc3, lbs_loc2, ..);
```

```
if (exception)
    goto exception_handler;
-- バッファリングを使用可能にしたロケータを、Append、Copy、Trimなど
-- の操作と一緒に使用することはできません。
-- これを行った場合、LOB のバッファをサーバーにフラッシュします。
OCILobFlushBuffer(..., lbs_loc2, OCI_LOB_BUFFER_NOFREE);

if (exception)
    goto exception_handler;
-- これによって、LOB バッファにあるすべての変更されたページがフラッシュ
-- され、lbs_loc2 が更新済ロケータから読取り一貫性のあるロケータにリセット
-- されます。変更されたページはバッファに残り、メモリーを解放しません。
-- これらのページは、必要に応じてページ・アウトできます。

-- バッファリング・モードでの LOB へのアクセスに対して、ロケータを使用禁止にします。*/
OCILobDisableBuffering(lbs_loc1);
OCILobDisableBuffering(lbs_loc2);
OCILobDisableBuffering(lbs_loc3);

if (exception)
    goto exception_handler;
-- これによって、3つのロケータがバッファリング使用禁止になり、LOB の
-- バッファ・リソースが解放されます。
exception_handler:
handle_exception_reporting();
exec_statement("rollback to savepoint lbs_savepoint");
}
```

LOB への参照を含む VARRAY の作成

LOB、または LOB への参照も、VARRAY を使用して作成できます。LOB への参照を含む VARRAY を作成するには、次を参照してください。

Multimedia_tab 表には、すでに MAP_TYP 型の MAP_OBJ 列があります。Multimedia_tab 表の詳細は、第 8 章「サンプル・アプリケーション」を参照してください。MAP_OBJ 列は、DRAWING という名前の BLOB 列を含みます。

関連する型および Multimedia_tab 表を作成する構文については、9-10 ページの「[SQL: 1 つ以上の LOB 列を含む表の作成](#)」を参照してください。

例

1つのマルチメディア・クリップに複数のマップ・オブジェクトを格納する必要があるとします。これを行うには、次の手順に従います。

1. REF MAP_TYP 型の VARRAY を定義します。

次に例を示します。

```
CREATE TYPE MAP_TYP_ARR AS  
    VARRAY(10) OF REF MAP_TYP;
```

2. 配列型の列を Multimedia_tab に定義します。

次に例を示します。

```
CREATE TABLE MULTIMEDIA_TAB ( .....etc. [list all columns here]  
    ... MAP_OBJ_ARR MAP_TYP_ARR)  
    VARRAY MAP_OBJ_ARR STORE AS LOB MAP_OBJ_ARR_STORE;
```


この章では、次のようなよくある質問（FAQ）と、それに対する回答を示します。

- LOB データ型へのデータ型の変換
 - どのような長さのデータでも LOB 列に挿入または更新できますか？
 - データが 64KB を超える場合、LONG を LOB にコピーできますか？
- 一般的な質問
 - トリガーのある LOB 列が更新されているかどうかを判断する方法は？
 - LOB データの読み込みおよびロード:量パラメータのサイズは？
- 索引構成表 (IOT) および LOB
 - 行内記憶域は索引構成表の LOB に使用できますか？
- LOB ロケータの初期化
 - EMPTY_BLOB() および EMPTY_CLOB() はいつ使用するのですか？
 - Java で EMPTY_BLOB() を使用して BLOB 属性を初期化する方法は？
- JDBC、JPublisher および LOB
 - JDBC を使用して空の LOB ロケータを持つ行を表に挿入する方法は？
 - JDBC: OracleBlob および OracleClob は Oracle8i で使用できますか？
 - 8.1 JDBC Thin Driver で LOB を操作する方法は？
 - LOB へ書き込む場合、SELECT に FOR UPDATE 句は必要ですか？
- LOB およびデータの LOB へのロード
 - 1MB のファイルを CLOB 列にロードする方法は？

-
- JDBC Driver を使用してロードする場合に BLOB および CLOB のパフォーマンスを向上させる方法は？
 - LOB 索引付け
 - LOB 索引は LOB データと同じ表領域内に作成されますか？
 - 索引付け : DELETE で BLOB 列は削除されますが、BFILE 列は削除されないのはなぜですか？
 - LOB 索引について問合せができるのはどのビューですか？
 - LOB 記憶域および領域の問題
 - LOB 表領域と ENABLE STORAGE IN ROW を指定するとどうなりますか？
 - BFILE と BLOB にイメージを格納する場合のメリットおよびデメリットは何ですか？
 - DISABLE STORAGE IN ROW はいつ指定する必要がありますか？
 - 4KB 未満の BLOB は表データと同じセグメントに移動されますか？4KB を超える BLOB は指定されたセグメントに移動されますか？
 - 4KB の BLOB は行内に格納されますか？
 - LOB 列が NULL ではなく EMPTY_CLOB() または EMPTY_BLOB() の場合に、LOB ロケータはどのように格納されますか？この場合、追加のデータ・ブロックが使用されますか？
 - 他のデータベース・システムからの移行
 - Oracle8i では、異なる LOB 型の間での暗黙的な LOB 変換はできますか？
 - パフォーマンス
 - ディスク・アレイ、UNIX および Oracle において Veritas File System を使用した場合に LOB ロードのパフォーマンスを向上させる方法は？
 - DBMS_LOB.SUBSTR を使用した場合と DBMS_LOB.READ を使用した場合とでは、パフォーマンスに違いがありますか？
 - LOB パフォーマンスのチューニングに関するホワイト・ペーパーまたはガイドラインがありますか？
 - 全体の読みみにチャンクを使用する必要があるのはいつですか？
 - LOB を行内に格納してもよいでしょうか？またその場合の時期はいつですか？
 - LOB パフォーマンスのチューニングに関するホワイト・ペーパーまたはガイドラインがありますか？
 - 4GB を超える LOB をデータベースに格納する方法は？

LOB データ型へのデータ型の変換

どのような長さのデータでも LOB 列に挿入または更新できますか？

質問

どのような長さのデータでも LOB 列に対して挿入または更新できますか？ 現在でも 4KB に制限されていますか？ LOB 属性にはどのような制限がありますか？

回答

現在は、LOB 列を挿入または更新する場合に、4KB の制限はありません。

LOB 属性については、次の 2 つのステップを実行する必要があります。

1. RETURNING 句を使用して、空の LOB に INSERT します。
2. OCILobWrite をコールして、すべてのデータを書き込みます。

データが 64KB を超える場合、LONG を LOB にコピーできますか？

質問

例：次の SQL を使用して、Long を LOB にコピーします。

```
INSERT INTO Multimedia_tab (clip_id,sound) SELECT id, TO_LOB(SoundEffects)
```

これは、LONG または LONGRAW が 64KB を超える場合でも有効ですか？

回答

はい。LONG にあるすべてのデータは、サイズに関係なく LOB にコピーされます。

一般的な質問

トリガーのある LOB 列が更新されているかどうかを判断する方法は？

質問

LOB 列にトリガーが必要なプロジェクトに取り組んでいます。この列が更新された場合、いくつかの条件をチェックする必要があります。この LOB 列に対する値が NEW にあるかどうかは、どのようにチェックするのですか？ BLOB は NULL と比較できないため、NULL は無効になってしまいます。

回答

トリガー内に UPDATING 句を使用して、LOB 列が更新されているかどうかを判断できます。

```
CREATE OR REPLACE TRIGGER.....  
...  
    IF UPDATING('lobcol')  
    THEN .....  
...  
...
```

注意：前述の例は、トップ・レベルの LOB 列に対してのみ有効です。

LOB データの読み込みおよびロード：量パラメータのサイズは？

質問

前回のリリースの『アプリケーション開発者ガイド ラージ・オブジェクト』には、次のように記載されていました。

「LOB 値を読み込むとき、LOB の最後を超えて読み込んでもエラーにはなりません。開始オフセットと LOB のデータ量にかかわらず、通常 4GB を指定できます。読み込む量を決定するために、サーバーへの OCILobGetLength() コールを繰り返し、LOB 値の長さを判断する必要はありません。」

さらに、DBMS_LOB.LOADFROMFILE() プロシージャの項では、次のように記述されています。

「ソース BFILE のデータの長さを超える値を指定してもエラーにはなりません。そのため、src_offset から BFILE の終わりまでのコピーを指定できます。」

それでも、次のコードはエラーを戻します。

```
declare
  cursor c is
    select id, text from bfiles;
  v_clob      clob;
begin
  for j in c
  loop
    Dbms_Lob.FileOpen ( j.text, Dbms_Lob.File_ReadOnly );
    insert into clobs ( id, text )
      values ( j.id, empty_clob() )
      returning text into v_clob;
    Dbms_Lob.LoadFromFile
      (
        dest_lob => v_clob,
        src_lob  => j.text,
        amount   => 4294967296, /* = 4Gb */
        dest_offset => 1,
        src_offset  => 1
      );
    Dbms_Lob.FileClose ( j.text );
  end loop;
  commit;
end;
/
```

表示されるエラー・メッセージは次のとおりです。

ORA-21560: 引数3がNULLまたは無効、範囲外です。

量パラメータの値を1減らして4294967295にすると、次のエラー・メッセージが表示されます。

ORA-22993: 指定された入力量は実際のソース量を超えています。

なぜエラーになるのかを教えてください。

回答

■ PL/SQL:

- DBMS_LOB.LOADFROMFILE では、量パラメータを BFILE のサイズを超える値に指定することはできません。そのため、前述のコード例ではエラーが戻ります。
- DBMS_LOB.READ では、量パラメータをデータのサイズより大きい値にできます。ただし、PL/SQL ではバッファのサイズが 32KB に制限され、また、データ量はバッファのサイズ以下である必要があるため、データ量は 32KB に制限されます。

PL/SQL では、データ量がバッファ・サイズを超える場合、エラーが戻ることにご注意してください。ub4 変数の制限は 4GB -1 であるため、常にデータ量を 4GB -1 以下に指定する必要があります。

- **OCI:** OCILobLoadFromFile では、データ量を BFILE の長さより大きい値に指定できません。ただし、OCILobRead では、データ量を 4GB -1 に指定できます。この場合、LOB の終わりまで読み込まれます。

索引構成表 (IOT) および LOB

行内記憶域は索引構成表の LOB に使用できますか？

質問

行内記憶域は索引構成表の LOB に使用できますか？

回答

索引構成表の LOB では、表がオーバーフロー・セグメント付きで作成されている場合のみ、行内 LOB 記憶域を使用できます。

LOB ロケータの初期化

EMPTY_BLOB() および EMPTY_CLOB() はいつ使用するのですか？

質問

EMPTY_BLOB() および EMPTY_CLOB() は、いつ使用する必要がありますか？ これまで、CLOB または BLOB を挿入するたびに、まず EMPTY_CLOB() または EMPTY_BLOB() で、LOB ロケータを初期化する必要があると思っていました。

回答

Oracle8i リリース 8.1 では、データが 4KB 未満である限り、INSERT 文を使用して、データで LOB を初期化できます。INSERT 文が有効であったのは、このためです。UPDATE 文を介して、4KB 未満のデータで LOB を更新することもできます。LOB が 4KB を超える場合、次のステップを実行します。

1. EMPTY_BLOB() または EMPTY_CLOB() を使用して LOB を初期化する表に挿入し、RETURNING 句を使用してロケータを取り戻します。
2. LOB 属性に対しては、ocilobwrite() をコールしてデータ全体を LOB に書き込みます。LOB 属性以外のものについては、INSERT 文ですべてのデータを挿入できます。

次のことに注意してください。

- 4KB 未満の制限は削除されているため、INSERT 文を使用して 4KB 以上のデータを LOB に挿入できます。さらに、UPDATE 文を使用して、LOB 列の文を更新することもできます。ただし、オブジェクト型の一部である LOB 属性はデータで初期化できず、EMPTY_BLOB()/EMPTY_CLOB() を使用する必要があります。
- LOB データを挿入または更新するときに 4KB 以上を使用できる場合でも、LOB のデフォルト値として 4KB 以上は使用できません。
- データまたは EMPTY_BLOB()/EMPTY_CLOB() を使用した LOB 値の初期化は、データの格納方法に影響します。LOB 値が約 4KB 未満の場合、(ユーザーが DISABLE STORAGE IN ROW を指定しない場合に限り) その値は行内に格納されます。LOB 値が 4KB よりも大きくなると、行外に移動されます。

Java で EMPTY_BLOB() を使用して BLOB 属性を初期化する方法は？

質問

Java から、BLOB 属性を持つ完全なオブジェクトを Oracle8i のオブジェクト表に挿入する必要があります。これを行う場合の問題は、BLOB 属性を EMPTY_BLOB() で初期化する必要があります。Java で、EMPTY_BLOB() を使用して BLOB 属性を初期化する方法はありますか？

現在、次のことを行っています。

まず、NULL のオブジェクトを BLOB 属性に挿入します。その後 EMPTY_BLOB() でオブジェクトを更新し、これを再選択します。次に BLOB ロケータを取得し、最後に BLOB に書き込みます。

有効な方法はこれだけでしょうか？ Custom Datum インタフェース実装の toDatum メソッドで、BLOB を直接初期化する方法はありますか？

回答

同等の SQLJ は次のとおりです。

```
BLOB myblob = null;
#sql { select empty_blob() into :myblob from dual } ;
```

BLOB を NULL に初期化する必要がある場合は、常にコード内に myblob を使用してください。

「JDBC、JPublisher および LOB」の「JPublisher を使用して EMPTY_BLOB() に setData を実行する方法は？」の質問および回答も参照してください。

JDBC、JPublisher および LOB

JDBC を使用して空の LOB ロケータを持つ行を表に挿入する方法は？

質問

JDBC を使用して、空の LOB ロケータを持つ行を表に挿入できますか？

回答

EMPTY_BLOB() は、JDBC でも使用できます。

```
Statement stmt = conn.createStatement() ;
try {
    stmt.execute ("insert into lobtable values (empty_blob())");
}
catch{ ...}
```

次のような例もあります。

```
stmt.execute ("drop table lobtran_table");
stmt.execute ("create table lobtran_table (b1 blob, b2 blob, c1 clob,
        c2 clob, f1 bfile, f2 bfile)");
stmt.execute ("insert into lobtran_table values
        ('01010101010101010101010101010101010101010101', empty_blob(),
        'onetwothreefour', empty_clob(),
        bfilename('TEST_DIR','tkpjobLOB11.dat'),
        bfilename ('TEST_DIR','tkpjobLOB12.dat'))");
```

JPublisher を使用して EMPTY_BLOB() に setData を実行する方法は？

質問

Jpublisher を使用して、どのように EMPTY_BLOB() に setData を実行するのですか？ JDBC で処理された SQL 文ではなく、Java 文の EMPTY_BLOB() および EMPTY_CLOB() のようなものはありますか？ JPublisher を使用して EMPTY_BLOB() に setData を実行するには、どのような方法がありますか？

回答

JPublisher で空の LOB を作成する方法の 1 つは、次のとおりです。

```
BLOB b1 = new BLOB(conn, null) ;
```

データ列の set メソッドでは、b1 を使用できます。

JDBC: OracleBlob および OracleClob は Oracle8i で使用できますか？

質問

OracleBlob および OracleClob は、Oracle8i で使用できますか？

回答

OracleBlob および OracleClob は、LOB データにアクセスするために JDBC 8.0.x ドライバで使用される Oracle 固有の機能です。リリース 8.1 および将来のリリースでは、OracleBlob および OracleClob は使用できません。

OracleBlob または OracleClob を使用して LOB にアクセスすると、標準エラー・メッセージが表示されます。たとえば、Oracle8i リリース 8.1 の JDBC Thin Driver で LOB を操作しようとする、次のエラーが表示されます。

```
"Dumping lob java.sql.SQLException: ORA-03115: サポートされていないネットワークのデータ型または表現があります。"
```

これらのサポートされていない機能、代替および改善された JDBC メソッドについては、『Oracle8i JDBC 開発者ガイドおよびリファレンス』を参照してください。

Java での LOB に対する作業の詳細は、Oracle8i に付属の LOB のサンプル・コードを参照するか、<http://www.oracle.com/java/jdbc> から LOB 例を取得してください。

8.1 JDBC Thin Driver で LOB を操作する方法は？

質問

8.1 JDBC Thin Driver で LOB を操作しようとする、次のようなエラーが表示されました。

```
Dumping lobs
java.sql.SQLException: ORA-03115: サポートされていないネットワークのデータ型または表現があります。
at oracle.jdbc.ttc7.TTIoer.processError(TTIoer.java:181)
at oracle.jdbc.ttc7.Odscrarr.receive(Compiled Code)
at oracle.jdbc.ttc7.TTC7Protocol.describe(Compiled Code)
at oracle.jdbc.ttc7.TTC7Protocol.parseExecuteDescribe(TTC7Protocol.java: 516)
at oracle.jdbc.driver.OracleStatement.doExecuteQuery(OracleStatement.java:1002)
at oracle.jdbc.driver.OracleStatement.doExecute(OracleStatement.java:1163)
at oracle.jdbc.driver.OracleStatement.doExecuteWithTimeout(OracleStatement.java:1211)
at oracle.jdbc.driver.OracleStatement.executeQuery(OracleStatement.java: 201)
at LobExample.main(Compiled Code)
-----
```

使用したコードは、8.0 に付属の LobExample.java です。これは、本来 OCI8 のサンプルです。唯一の違いは、8.1 インスタンスに対して 8.1 Thin Driver を使用していることです。

回答

間違ったサンプルを使用しています。OracleBlob および OracleClob はサポートされていないため、有効ではありません。Oracle8i に付属の LobExample サンプルを使用してください。これは、<http://www.oracle.com/java/jdbc> から取得できます。

LOB へ書き込む場合、SELECT に FOR UPDATE 句は必要ですか？

質問

CLOB を書き込む Java ストアド・プロシージャを実行していますが、次の例外が発生します。

```
ORA-22920: LOB 値を含む行がロックされていません。
```

```
ORA-06512: 708 行 "SYS.DBMS_LOB"
```

```
ORA-06512: 1 行
```

SELECT 文に FOR UPDATE 句を追加すると、この例外は発生しません。

SELECT に FOR UPDATE 句を指定する必要があることを、『Oracle8i JDBC 開発者ガイドおよびリファレンス』に反映する必要があると思います。特に、第7章「LOB および BFILE を使用した作業」の「BLOB および CLOB ロケータの取得」および「BLOB 列または CLOB 列の作成および移入」の記事は、更新する必要があると思います。

回答

これは特に JDBC の問題というわけではありません。LOB の動作の問題です。autoCommit は、デフォルトで FALSE に設定されているため、このことは JSP で証明されています。クライアント側で autoCommit を FALSE に設定した場合にも、同じ例外が発生します。FOR UPDATE 句を使用した場合はロックが明示的に取得されるため、例外は発生しません。

LOB およびデータの LOB へのロード

1MB のファイルを CLOB 列にロードする方法は？

質問

ディスクに格納されている 1MB のファイルを表の CLOB 列に挿入するには、どのようにすればいいのですか？ DBMS_LOB.LOADFROMFILE が有効であると思うのですが、ドキュメントには、これは BFILE にのみ有効であると記述されています。どうすればよいでしょうか。

回答

SQL*Loader を使用できます。『Oracle8i ユーティリティ・ガイド』、またはこのマニュアルの 4-5 ページの「LOB ロード時の SQL*Loader の使用」を参照してください。

loadfromfile() を使用してデータを CLOB にロードできますが、そのデータはロー・データとして BFILE から転送されます。キャラクタ・セット変換は実行されません。キャラクタ・セット変換が必要な場合は、loadfromfile() をコールする前にユーザー自身が実行してください。

コールバックで OCILobWrite() を使用してください。コールバックは、オペレーティング・システム (OS) からファイルを読み込み、データをデータベースのキャラクタ・セットに変換できます (OS ファイルのキャラクタ・セットと異なる場合)。その後、CLOB にデータを書き込みます。

JDBC Driver を使用してロードする場合に BLOB および CLOB のパフォーマンスを向上させる方法は？

質問

BLOB および CLOB に関するパフォーマンスに問題があります。JDBC Driver を使用して BLOB および CLOB にデータをロードするのに、非常に時間がかかります。

回答

実際、JDBC Thin Driver を使用してデータを LOB に挿入するには時間がかかります。これは、まだ DBMS_LOB パッケージが使用されており、各 LOB 操作に対する完全な JDBC CallableStatement 実行のオーバーヘッドが追加されるためです。

JDBC OCI および JDBC のサーバー側の内部ドライバでは、システム固有の LOB API が使用されるため、挿入がより高速になります。JDBC Driver 実装からの余分なオーバーヘッドはありません。

LOB データへのアクセスおよび操作には、InputStream および OutputStream の使用をお勧めします。LOB のストリーム・アクセスを使用することによって、JDBC Driver は LOB データのバッファリングを適切に処理し、ネットワークのラウンドトリップの数を削減します。また、各データベース処理で、LOB 固有のチャンク・サイズの倍数のデータ・サイズが使用されることが保証されます。

OutputStream を使用して BLOB にデータを書き込む例を次に示します。

```
/*
 * この例では、GIF ファイル john.gif が BLOB に書き込まれます。
 */
import java.sql.*;
import java.io.*;
import java.util.*;

//Oracle JDBC Driver パッケージをインポートすると、コードがより読みやすくなります。
import oracle.jdbc.driver.*;

// 新しい CLOB および BLOB クラスに必要です。
import oracle.sql.*;

public class LobExample
{
    public static void main (String args [])
        throws Exception
    {
        // Oracle JDBC Driver を登録します。
        DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

        // データベースに接続します。
        // データベース名は、接続 URL の @ 符号の後に入力できます。
        Connection conn =
            DriverManager.getConnection ("jdbc:oracle:oci8:@", "scott", "tiger");

        // 自動コミットがオフの場合、高速になります。
        conn.setAutoCommit (false);

        // 文を作成します。
        Statement stmt = conn.createStatement ();

        try
        {
            stmt.execute ("drop table persons");
        }
        catch (SQLException e)
        {
            // 表が存在しない場合、この時点で例外が発生する可能性があります。
        }
    }
}
```

```
// BLOB および CLOB を含む表を作成します。
stmt.execute ("create table persons (name varchar2 (30), picture blob)");

// 表を移入します。
stmt.execute ("insert into persons values ('John', EMPTY_BLOB())");

// BLOB を選択します。
ResultSet rset = stmt.executeQuery ("select picture from persons where name =
'John'");
if (rset.next ())
{
    // 表から BLOB ロケータを取得します。
    BLOB blob = ((OracleResultSet)rset).getBLOB (1);

    // john.gif ファイルのファイル・ハンドラを宣言します。
    File binaryFile = new File ("john.gif");

    // GIF ファイルの内容を読み込むための FileInputStream オブジェクトを作成します。
    FileInputStream istream = new FileInputStream (binaryFile);

    // BLOB をストリームとして書き込むための OutputStream オブジェクトを作成します。
    OutputStream ostream = blob.getBinaryOutputStream ();

    // 一時バッファを作成します。
    byte[] buffer = new byte[1024];
    int length = 0;

    // read() メソッドを使用して、GIF ファイルをバイト配列バッファに読み込み、
    // write() メソッドを使用して、そのファイルを BLOB に書き込みます。
    while ((length = istream.read(buffer)) != -1)
        ostream.write(buffer, 0, length);

    // InputStream および OutputStream をクローズします。
    istream.close();
    ostream.close();

    // BLOB サイズをチェックします。
    System.out.println ("Number of bytes written = "+blob.length());
}

// すべてのリソースをクローズします。
rset.close();
```

```
        stmt.close();  
        conn.close();  
    }  
}
```

DBMS_LOB.WRITE() のかわりに DBMS_LOB.LOADFROMFILE() を使用すると、パフォーマンスがさらに向上します。

DBMS_LOB.LOADFROMFILE() を使用するには、LOB に書き込むデータがサーバー側のファイルにある必要があります。

LOB 索引付け

LOB 索引は LOB データと同じ表領域内に作成されますか？

質問

LOB 索引は LOB データと同じ表領域内に作成されますか？

回答

LOB 索引は LOB 列に作成され、LOB データを索引付けします。LOB 索引はロケータと同じ表領域に存在します。

索引付け : DELETE で BLOB 列は削除されますが、BFILE 列は削除されないのはなぜですか？

質問

promotion 列は BFILE として定義および索引付けされますが、たとえば行が DELETE された場合、promotion 列が BLOB として定義されている場合は、Word ドキュメントは削除されます。ただし、列が BFILE として定義されている場合は削除されません。これはなぜですか？

回答

BFILE データに対する索引は作成されていません。内部永続 LOB は自動的にデータベースでバックアップされますが、外部 BFILE はバックアップされないことにも注意してください。また、内部永続 LOB に対する変更は、将来のリカバリのために REDO ログに格納されます。

LOB 索引について問合せができるのはどのビューですか？

質問

LOB 索引について問合せができるのはどのビューですか？

回答

- 内部永続 LOB
 - ALL_INDEXES ビュー : カレント・ユーザーがどの方法でも変更できるすべての索引が含まれます。LOB 索引は改名、再作成または変更できないため、このビューに LOB 索引は表示されません。

- DBA_INDEXES ビュー: 存在するすべての索引が含まれます。LOB 索引についての情報を検索するには、このビューを問い合わせてください。
- USER_INDEXES ビュー: ユーザーが所有するすべての索引が含まれます。ビューに問い合わせているユーザーと作成したユーザーが同じである場合、LOB 索引がこのビューに表示されます。

- **テンポラリ LOB**

テンポラリ LOB に対しては、LOB 索引情報は V\$SORT_USAGE ビューから取り出せません。

次に例を示します。

```
SELECT USER#, USERNAME, SEGTYPE, EXTENTS, BLOCKS
       FROM v$sort_usage, v$session
       WHERE SERIAL#=SESSION_NUM;
```

LOB 記憶域および領域の問題

LOB 表領域と ENABLE STORAGE IN ROW を指定するとどうなりますか？

質問

LOB TABLESPACE と ENABLE STORAGE IN ROW を同時に指定するとどうなりますか？

回答

LOB 値の長さが約 4KB より小さい場合、データは表の行内に格納されます。約 4KB よりも大きくなると、LOB 値は指定された表領域に移動されます。

BFILE と BLOB にイメージを格納する場合のメリットおよびデメリットは何ですか？

質問

BFILE と BLOB にイメージを格納する場合のメリットおよびデメリットは何ですか？

回答

基本的には次のことがいえます。

- セキュリティ
 - BFILE は、オペレーティング・システム (OS) と同様に安全性が保証されていません。
- 機能
 - BLOB と異なり、BFILE は標準データベース API から書き込めません。
 - 最も重要な機能の 1 つに、BLOB はトランザクションで使用でき、リカバリ可能ですが BFILE ではできません。
- パフォーマンス
 - ほぼ同じです。
 - バッファ・キャッシュのサイズを増やすと、BLOB のパフォーマンスが大幅に向上します。

- BLOB は、Oracle のキャッシュ内に存在するように構成でき、これによって、繰返し / 複数の読み込みが高速化されます。
- BLOB のピース単位 / 非連続的なアクセスは、BFILE の場合より高速です。
- 管理性
 - BFILE ロケータのみが Oracle BACKUP に格納されます。別のバックアップを実行して、BFILE ロケータがポイントする OS ファイルを保存する必要があります。BLOB データは、残りのデータベース・データとともにバックアップされます。
- 記憶域
 - BLOB にファイル・データを格納するために必要な表領域の量は、LOB 索引のために、ファイル自体に必要な量よりも大きくなります。この LOB 索引によって、BLOB 値のピース単位のランダム・アクセスに対する BLOB のパフォーマンスが向上します。

DISABLE STORAGE IN ROW はいつ指定する必要がありますか？

質問

フル・テーブル・スキャンを含む多くの UPDATE または SELECT が予測される場合、DISABLE STORAGE IN ROW を常に指定する必要がありますか？

回答

他の表データが頻繁に更新または選択される場合は、DISABLE STORAGE IN ROW を使用してください。LOB データが頻繁に更新または選択される場合は、使用しないでください。

4KB 未満の BLOB は表データと同じセグメントに移動されますか？ 4KB を超える BLOB は指定されたセグメントに移動されますか？

質問

BLOB に対してセグメントおよび表領域を指定し、ENABLE STORAGE IN ROW を指定して USER LOBS を参照した場合、BLOB は IN_ROW として定義され、セグメントが指定されています。これは何を意味していますか？ 4KB 以下のすべての BLOB は表データと同じセグメントに移動しますが、4KB を超えるものは指定したセグメントに移動するのですか？

回答

はい。

4KB の BLOB は行内に格納されますか？

質問

『Oracle8i SQL リファレンス』の第7章には、次のように記載されています。

ENABLE STORAGE IN ROW - LOB 値の長さが、約 4000 バイトからシステム制御情報分を引いた長さより小さい場合、LOB 値を行に格納することを指定します。これはデフォルト値です。

行内 LOB が 4KB を超える場合、次のどちらが正しいでしょうか？

1. 最初の 4KB が構造データに格納され、残りは別の場所に格納される
 2. LOB 全体が別の場所に格納される
- 2 番目が正しいと思うのですが、どうでしょうか？

回答

そのとおりです。正しいのは 2 番目です。いくつかのメタ情報が行内に格納されるため、LOB 値へのアクセスが高速になります。ただし、LOB 値が約 4KB を超えると、LOB 値全体が別の場所に格納されます。

1. BLOB ロケータに NULL 値がある場合、次を実行したとします。

```
INSERT INTO blob_table (key, blob_column) VALUES (1, null);
```

この場合、BLOB 値に対するポインタがないため、他の NULL 値と同様に、領域は使用されないと考えられます。

2. BLOB に NULL がある場合、次を実行したとします。

```
INSERT INTO blob_table (key, blob_column) VALUES (1, empty_blob());
```

この場合、2 番目が正しくなります。少なくともチャンク・サイズの領域が必要です。BLOB に NULL 値を使用する場合と、空の文字列を使用する場合とは区別されます。

LOB 列が NULL ではなく EMPTY_CLOB() または EMPTY_BLOB() の場合に、LOB ロケータはどのように格納されますか？ この場合、追加のデータ・ブロックが使用されますか？

質問

LOB 列が NULL ではなく、EMPTY_CLOB() または EMPTY_BLOB() の場合、LOB ロケータはどのように行に格納されますか？ また、この場合、追加のデータ・ブロックが使用されますか？

答

7-5 ページの「[LOB 記憶域](#)」も参照してください。

DISABLE STORAGE IN ROW 属性の LOB 列を含む表を作成する、単純なテストを実行できます。NULL の LOB を含む行を数千挿入してください。

Oracle8i は、数千ものチャンクを消費して NULL を格納しないことに注意してください。

他のデータベース・システムからの移行

Oracle8i では、異なる LOB 型の間での暗黙的な LOB 変換はできますか？

質問

異なる LOB 型の間での暗黙的な LOB 変換は行われませんか？たとえば、PL/SQL では、次の問合せはできません。

```
INSERT INTO t VALUES ('abc');  
WHERE t CONTAINS a CLOB column.....
```

Oracle8i でもこの制限があるのでしょうか。この制限は Oracle8 の PL/SQL に存在しますが、挿入するデータが 4KB 未満である限り、ユーザーは SQL で INSERT 文を発行できると思います。この 4KB 制限は、SQL では削除されていると理解しています。

回答

Oracle8i では、PL/SQL 制限は削除されました。現在は、4KB を超えるデータを挿入できます。

パフォーマンス

ディスク・アレイ、UNIX および Oracle において Veritas File System を使用した場合に LOB ロードのパフォーマンスを向上させる方法は？

質問 1

250MB のビデオ・コンテンツをデータベースの BLOB 列に移入しようとする、ロードに 70 秒以上かかります。UNIX のコピーを使用した場合の 15 秒の転送時間と比較すると、これは許容範囲外です。この状況を改善するにはどうしたらよいでしょうか？

BLOB はパーティション化された表領域に格納されており、NOLOGGING、NOCACHE オプションはパフォーマンスを最大限にするように指定されています。

パーティション表領域およびパーティション記憶域に対する INITIAL および NEXT エクステントを 300MB に定義し、データのロード時のオーバーヘッドが最小になるように、MINEXTENTS を 1 に設定しています。

チャンク・サイズは、Oracle の最大値である 32768 バイトに設定しています。

db_block_buffers に対する INIT.ORA パラメータは、上下に変更しました。

前述の処理をすべて行っても、ロード時間はほとんど変わらず、70 ~ 75 秒で一定しています。これらの設定ではほとんど効果がありません。

回答 1

まず、I/O 記憶デバイスおよびバスを調べてください。

質問 2

I/O デバイス/バス: 4 つの SUN AS5200 ディスク・アレイをデータ記憶域 (BLOB が書き込まれたデバイス) に対して使用していました。この配列のディスクは、(9+9) の 4 つのストライプを持つ RAID (0+1) です。Veritas VxFS 3.2.1 は、すべてのディスクにおけるファイル・システムです。

異なるデバイスを使用する効果を測定するために、BLOB に対する表領域を /tmp で定義しました。/tmp とは、スワップ領域です。

当然、BLOB のロードには 14 秒 (毎分 1.07GB のデータ転送率) しかかかりませんでした。これからは、UNIX のコピー程ではなくても、パフォーマンス率の差が小さくなっていることがわかります。

このことがきっかけで、ディスク・アレイの表領域に BLOB をロードする場合に、何が発生するかを詳しく調査するようになりました。SAR 出力によって、I/O に対する長い待機、メモリーの大量消費、CPU の高サイクル、および常に一貫している 70 秒のロード時間がわかりました。これを解決するにはどのような方法がありますか？

回答 2

Veritas QuickIO オプションのインストール: 明らかに、Veritas、UNIX および Oracle の操作の総合的な問題です。これについては、サポート用のドキュメントを提供しています。Oracle と、ディスク・アレイにおける Veritas File System で納得できるパフォーマンスを得るには、**Veritas QuickIO オプションのインストール**をお勧めします。

最終的な注意: 通常、LOB の書込みが遅い場合、問題は、Oracle の LOB の書込み方法ではありません。前述の場合、UNIX のファイル・キャッシングを使用する Veritas File System を使用しているため、パフォーマンスは非常に悪くなります。

UNIX のキャッシングを使用禁止にすると、パフォーマンスはシステム固有のファイルのコピーで向上します。

DBMS_LOB.SUBSTR を使用した場合と DBMS_LOB.READ を使用した場合とでは、パフォーマンスに違いがありますか？

質問

DBMS_LOB.SUBSTR を使用した場合と DBMS_LOB.READ を使用した場合とでは、パフォーマンスに違いがありますか？

回答

DBMS_LOB.SUBSTR は SQL 文で使用できる関数です。パフォーマンスの違いはありません。

LOB パフォーマンスのチューニングに関するホワイト・ペーパーまたはガイドラインがありますか？

質問

LOB パフォーマンスのチューニングに関するホワイト・ペーパーまたはガイドラインがありますか？

回答

7-4 ページの「[表アーキテクチャの選択](#)」を参照してください。7-31 ページの「[パフォーマンスの最適化のための最善策](#)」も参照してください。

LOB パフォーマンスに関する情報を掲載した Web サイトがありましたが、これは古い情報です。更新される予定なので定期的にチェックしてください。

全体の読み込みにチャンクを使用する必要があるのはいつですか？

質問

全体の読み込みにチャンクを使用する必要があるのはいつですか？

回答

LOB の 2 つ以上のチャンクを読み込む場合、ポーリングまたはコールバックを介したストリーム・メカニズムの `OCILOBRead` を使用してください。1 つのチャンクに収まる一部の LOB のみを読み込む必要がある場合、そのチャンクのみを読み込んでください。それより多くを読み込むと、余分なネットワーク・オーバーヘッドが発生します。

LOB を行内に格納してもよいでしょうか？ またその場合の時期はいつですか？

質問

LOB を行内に格納してもよいでしょうか？ また、その場合の時期はいつですか？

回答

LOB の行内格納はデフォルトで、ほとんどの場合お勧めします。Oracle8i では、LOB 値が約 4KB 未満の場合に、LOB が行内に格納されます。そのため、値を行外に格納するよりもパフォーマンスが向上します。LOB が 4KB よりも大きくなった場合、LOB 値は別の記憶域セグメントに移動されますが、LOB 値をすばやく検索するためのメタ情報は行内に格納されます。そのため、多くの場合、行内格納によってパフォーマンスが最大になります。

ただし、フル・テーブル・スキャン、複数行アクセス（レンジ・スキャン）、LOB 列以外の列の多数の更新 / 選択など、多くのベース表処理を実行する場合、LOB は行内格納しないでください。

4GB を超える LOB をデータベースに格納する方法は？

質問

4GB を超える LOB をデータベースに格納する方法は？

答

4GB を超える LOB を格納するには、次の 2 つの方法があります。

- LOB を圧縮して、4GB 以内に収めます。
- 別々の LOB 列または別々の LOB 行として、LOB を 4GB のチャンクに分割します。

モデリングおよび設計

この章の内容は次のとおりです。

- データ型の選択
 - LOB 型と LONG 型および LONG RAW 型との比較
 - キャラクタ・セットの変換：可変幅文字データの処理
- 表アーキテクチャの選択
 - LOB 列内の NULL 値の格納場所
 - 内部 LOB に対する表領域および記憶特性の定義
 - LOB 列または属性の LOB 記憶特性
 - TABLESPACE および LOB 索引
 - GB の LOB の作成方法
- LOB ロケータおよびトランザクション境界
- INSERT および UPDATE での 4,000 バイトを超えるバインド
- 内部 LOB 用の Open、Close および IsOpen インタフェース
- 索引構成表 (IOT) 内の LOB
- パーティション表内の LOB の操作
- LOB 列の索引付け
- パフォーマンスの最適化のための最善策
 - スレッド環境における LOB へのデータの移動

注意：この章で示す例は、第 8 章「サンプル・アプリケーション」で説明するマルチメディア・スキーマおよび Multimedia_tab 表を基にしています。

データ型の選択

LOB 型と LONG 型および LONG RAW 型との比較

LOB は、LONG 型および LONG RAW 型と似ていますが、次の点で異なります。

表 7-1 LOB と LONG RAW

LOB データ型	LONG および LONG RAW データ型
単一の行に複数の LOB を格納できます。	1 行あたり 1 つの LONG または LONG RAW し か格納できません。
LOB は、ユーザー定義のデータ型の属性にで きます。	LONG と LONG RAW のどちらも、ユーザー定 義データ型の属性にはできません。
LOB ロケータのみ表の列に格納されます。 BLOB データおよび CLOB データは、別々の表 領域に格納でき、BFILE データは外部ファイ ルとして格納されます。 行内 LOB では、表列の中に約 4KB までの データが格納されます。	LONG または LONG RAW の場合、値全体が表 の列に格納されます。
LOB 列にアクセスすると、ロケータが戻され ます。	LONG または LONG RAW にアクセスすると、 値全体が戻ります。
LOB の最大サイズは、4GB です。BFILE の最 大サイズは、オペレーティング・システムに よって異なりますが、4GB を超えることはあ りません。有効なアクセス可能範囲は、 1 ~ (2 ³² -1) です。	これに対して、LONG または LONG RAW は、 2GB までです。
ランダムにピース単位でデータを操作する場 合は、LOB を使用の方が柔軟性があります。 LOB はランダム・オフセットでアクセスでき ます。	LONG データまたは LONG RAW データを使用 して、ランダムにピース単位でデータを操作 する場合は、柔軟性はなりません。LONG で は、目的の位置にアクセスするためには、先 頭からアクセスする必要があります。
ローカルおよび分散環境の両方で LOB をレプ リケートできます。	ローカルおよび分散環境の両方でのレプリ ケーションは、LONG も LONG RAW も使用で きません (『Oracle8i レプリケーション・ガイ ド』を参照)。

既存の LONG 列は、TO_LOB() 関数を使用して LOB に変換できます (9-59 ページの「LOB への LONG のコピー」を参照)。

Oracle8i は、LOB から LONG への再変換をサポートしないことに注意してください。

キャラクタ・セットの変換：可変幅文字データの処理

OCI (Oracle Call Interface) または OCI 機能にアクセスするプログラム環境を使用する場合、キャラクタ・セットの変換は1つのキャラクタ・セットから別のキャラクタ・セットに翻訳するとき、暗黙的に実行されます。

ただし、バイナリ・データからキャラクタ・セットへの場合は、暗黙の変換は実行されません。loadfromfile 操作を使用して CLOB または NCLOB に移入する場合は、BFILE のバイナリ・データを LOB に移入することになります。この場合、loadfromfile を実行する前に、キャラクタ・セットの変換を BFILE データ上で実行しておく必要があります。

参照： キャラクタ・セットの変換の詳細は、『Oracle8i NLS ガイド』を参照してください。

表アーキテクチャの選択

表を設計する場合、次の設計基準を考慮してください。

- LOB 記憶域
 - LOB 列内の NULL 値の格納場所
 - 内部 LOB に対する表領域および記憶特性の定義
 - LOB 列または属性の LOB 記憶特性
 - TABLESPACE および LOB 索引
 - * PCTVERSION
 - * CACHE / NOCACHE / CACHE READS
 - * LOGGING / NOLOGGING
 - * CHUNK
 - * ENABLE | DISABLE STORAGE IN ROW
 - GB の LOB の作成方法
- 索引構成表 (IOT) 内の LOB
- パーティション表内の LOB の操作
- LOB 列の索引付け

LOB 記憶域

LOB 列内の NULL 値の格納場所

NULL LOB 列の記憶域 : NULL 値の格納

LOB 列が NULL の場合、情報の格納にデータ・ブロックは使用されません。NULL 値は、他の NULL 値と同様に、行内に格納されます。これは、LOB に対して `DISABLE STORAGE IN ROW` を指定する場合も同じです。

EMPTY_CLOB() または EMPTY_BLOB() 列の記憶域 : LOB ロケータの格納

LOB 列が `EMPTY_CLOB()` または `EMPTY_BLOB()` で初期化される場合、NULL のかわりに LOB ロケータが行内に格納されます。追加の記憶域は使用されません。

- `DISABLE STORAGE IN ROW`: 1 バイトのデータを含む LOB がある場合は、行内に LOB ロケータがあります。これは、LOB が `ENABLE` または `DISABLE STORAGE IN ROW` として作成されたかにかかわらず真です。さらに、LOB 列が `DISABLE STORAGE IN ROW` として作成された場合、チャンク・サイズ分のデータ・ブロックが、1 バイトのデータの格納に使用されます。
- `ENABLE STORAGE IN ROW`: LOB 列が `ENABLE STORAGE IN ROW` として作成された場合、1 バイトのデータの格納に、行内の 1 バイトの記憶域しか余分に使用されません。LOB 列を `ENABLE STORAGE IN ROW` で作成し、格納するデータ量が列（約 4,000 バイト）に収まらない場合、データの格納にチャンク・サイズの複数倍が使用されます。

内部 LOB に対する表領域および記憶特性の定義

表内に LOB を定義する場合、各内部 LOB に対して表領域および記憶特性を明示的に指定できます。

次に例を示します。

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

外部 LOB はデータベース内に格納されないため、外部 LOB に対する追加の表領域または記憶特性はありません。

後で LOB 記憶域パラメータを変更する場合は、ALTER TABLE 文の MODIFY LOB 句を使用します。

注意： いくつかの記憶域パラメータのみ変更できます。たとえば、ALTER TABLE ...MODIFY LOB 文を使用して、PCTVERSION、CACHE/NO CACHE LOGGING/NO LOGGING および STORAGE 句を変更できます。

また、ALTER TABLE ...MOVE 文を介して TABLESPACE を変更することもできます。

ただし、一度表が作成されると、チャンク・サイズまたは ENABLE/DISABLE STORAGE IN ROW の設定を変更できません。

LOB データ・セグメント名の割当て

前述の例に示されているように、LOB データ・セグメントに名前を指定すると、作業環境がより明確になります。LOB データ・ディクショナリ・ビュー USER_LOBS、ALL_LOBS、DBA_LOBS (『Oracle8i リファレンス・マニュアル』を参照) を問い合わせる場合、システムが生成した名前ではなく、選択した LOB データ・セグメントが表示されます。

LOB 列または属性の LOB 記憶特性

LOB 列または LOB 属性に対して指定できる LOB 記憶特性は次のとおりです。

- TABLESPACE
- PCTVERSION
- CACHE/NOCACHE/CACHE READS
- LOGGING/NOLOGGING
- CHUNK
- ENABLE/DISABLE STORAGE IN ROW
- STORAGE

詳細は、『Oracle8i SQL リファレンス』の「STORAGE 句」を参照してください。

ほとんどのユーザーには、これらの記憶特性のデフォルトで十分です。LOB 記憶域をチューニングするには、次のガイドラインを考慮する必要があります。

TABLESPACE および LOB 索引

LOB に対するパフォーマンスを最適化するには、LOB の記憶域を、その LOB を含む表に使用されるものとは別の表領域に指定します。多くの LOB が頻繁にアクセスされる場合、デバイスでの競合を避けるために、各 LOB 列または属性に個別の表領域を指定すると効果的です。

LOB 索引は、LOB 記憶域に密接に対応付けられている内部構造体です。これは、ユーザーが LOB 索引を削除または再構築できないことを示します。

注意： LOB 索引は変更できません。

システムは、LOB 記憶域内でのユーザー指定に従って、LOB データおよび LOB 索引に使用する表領域を決定します。

- LOB データに表領域を指定しない場合は、LOB データおよび索引には表の表領域が使用されます。
- LOB データに表領域を指定する場合は、LOB データと索引の両方に指定された表領域が使用されます。

非パーティション表の LOB 索引に対する表領域

Oracle 8i で表を作成するときに非パーティション表の LOB 索引に対して表領域を指定する場合、表領域の指定は無視され、LOB 索引は LOB データとともに配置されます。パーティション化された LOB に、LOB 索引構文は含まれません。

LOB 記憶域セグメントごとに別々の表領域を指定すると、表の表領域での競合を削減できません。

PCTVERSION

LOB が変更されると、新しいバージョンの LOB ページが作成され、前のバージョンの LOB 値の一貫読み込みがサポートされます。

PCTVERSION は、使用中の LOB データ領域全体のうち、旧バージョンの LOB データ・ページが占める割合です。LOB 領域内で旧バージョンの LOB データ・ページが、この PCTVERSION に指定した量を超えると、すぐに旧バージョンを初期化し直して再使用します。PCTVERSION は、使用中の LOB データ・ブロック全体のうち旧バージョンの LOB データが使用できる割合を表します。

デフォルト : 10 (%) 最小値 : 0 (%) 最大値 : 100 (%)

PCTVERSION に設定する値を決定するには、どれくらいの頻度で LOB が更新され、どれくらいの頻度で更新された LOB を読み込むかを考慮する必要があります。

表 7-2 「推奨する PCTVERSION 設定」に、適切な PCTVERSION 値を決定するガイドラインを示します。

表 7-2 推奨する PCTVERSION 設定

LOB 更新パターン	LOB 読み込みパターン	PCTVERSION
XX% の LOB データを更新する	更新された LOB を読み込む	XX%
XX% の LOB データを更新する	LOB を読み込むが、更新された LOB は読み込まない	0%
XX% の LOB データを更新する	LOB および更新されていない LOB の両方を読み込む	XX%
LOB を更新しない	LOB を読み込む	0%

例 1:

LOB を大量に読み込むと同時に、LOB の更新を時々行う場合を考えます。

PCTVERSION = 20% に設定します。

PCTVERSION をデフォルトの 2 倍の値に設定すると、旧バージョンのデータ・ページに使用できる空きページが増えます。大量の問合せを行う場合、一貫性のある LOB の読み込みが必要となるため、旧バージョンの LOB ページを保持しておく必要があります。この場合、Oracle は空きページを積極的に再使用しないため、LOB 記憶域は大きくなります。

例 2:

LOB を作成し、書き込みを 1 回のみ行い、その後は主に読み込み専用で使用します。更新はほとんど行わない場合を考えます。

PCTVERSION = 5% 以下に設定します。

LOB の更新をほとんど行わず、更新する部分が非常に少ない場合は、LOB データの旧バージョンの使用に必要な領域が少なくなります。既存の LOB が読み込み専用とわかっている場合は、データの旧バージョンにはページが必要ないため、PCTVERSION を 0% に設定しても問題ありません。

CACHE / NOCACHE / CACHE READS

LOB を含む表を作成する場合、表 7-3 「CACHE、NOCACHE および CACHE READS の使用時期」のガイドラインに従ってキャッシュ・オプションを使用します。

表 7-3 CACHE、NOCACHE および CACHE READS の使用時期

キャッシュ・モード	読み込み	書き込み
CACHE	頻繁	頻繁
NOCACHE (デフォルト)	1 回または時々	なし
CACHE READS	頻繁	1 回または時々

CACHE / NOCACHE / CACHE READS: LOB 値およびバッファ・キャッシュ

- CACHE: Oracle は、アクセスを高速化するために、バッファ・キャッシュに LOB ページを置きます。
- NOCACHE: LOB_storage_clause のパラメータとして NOCACHE が指定されると、LOB 値がバッファ・キャッシュに入れられるか、またはバッファ・キャッシュに入れられ、LRU リスト内の最も前に使用されたものの後に配置されます。
- CACHE READS: LOB 値は、読み込み中のみバッファ・キャッシュに入れられ、書き込み中には入れられません。

リリース 8.1.5 または 8.0.X へのダウングレード

リリース 8.1.6 で LOB に対して CACHE READS を設定し、リリース 8.1.5 または 8.0.x にダウングレードする場合、CACHE READS LOB は警告を生成し、CACHE LOGGING LOB になります。

LOB を CACHE LOGGING にしない場合、後で明示的に LOB の記憶特性を変更できます。たとえば、LOB を NOCACHE にする場合は、ALTER TABLE を使用して NOCACHE に確実に変更します。

LOGGING / NOLOGGING

[NO] LOGGING は、LOB の使用に関しては他の表操作と同様に適用されます。通常、[NO] LOGGING 句の省略は、NO LOGGING、LOGGING のどちらも指定されておらず、表または表のパーティションのロギング属性のデフォルトとして、置かれている表領域のロギング属性を使用するということです。

LOB には、CACHE の指定によっては他にも指定方法があります。

- **CACHE が指定され**、[NO] LOGGING 句が省略されると、LOGGING が自動的に実装されず (CACHE NOLOGGING がないためです)。
- **CACHE が指定されず**、[NO] LOGGING 句が省略されると、デフォルトとして表およびパーティション表と同様に処理されます。[NO] LOGGING 値は LOB 値の置かれている表領域から取得されます。

ただし、次の事項を考慮する必要があります。

LOB は、常に LOB 索引ページに対して UNDO を生成する

LOGGING または NOLOGGING が設定されているかどうかにかかわらず、古い LOB データはバージョンごとに格納されるため、LOB では LOB データ・ページのロールバック情報 (UNDO) は生成されません。LOB について作成されるロールバック情報は、LOB の索引ページの変更専用であるため、サイズが小さくなります。

LOGGING が設定されている場合、LOB データ・ページの完全な REDO が生成される

NOLOGGING は、メディア・リカバリを必要としない場合に使用されます。そのため、ディスク / テープ / 記憶域などのメディアに障害が発生した場合、変更のログは作成されていないため、ログから変更をリカバリできません。

NOLOGGING が効果を発揮するのは、大量のロードまたは挿入です。たとえば、LOB にデータをロードしている場合、REDO を必要とせず、ロードに失敗した場合にロードを簡単に再開できる場合、LOB データ・セグメントの記憶特性を NOCACHE NOLOGGING に設定します。これによって、データの初期ロードのパフォーマンスが向上します。データのロードが完了すると、ALTER TABLE を使用して、LOB データ・セグメントの LOB 記憶特性を、通常の LOB 操作に対する希望の特性 (CACHE または NOCACHELOGGING) に設定できます。

注意： CACHE を設定すると、LOGGING も行われます。

CHUNK

CHUNK には、一度にアクセスする LOB データのブロック数 (LOB 値への一度のアクセスで OCILobRead(), OCILobWrite(), DBMS_LOB.READ() または DBMS_LOB.WRITE() を介して読み込みまたは書き込みを行うブロックの数) を設定します。

注意： CHUNK のデフォルト値は 1 つの Oracle ブロックで、プラットフォームによって変化しません。

一度に 1 ブロックの LOB データにしかアクセスしない場合には、CHUNK を 1 ブロックのサイズに設定します。たとえば、データベース・ブロック・サイズが 2KB の場合には、CHUNK を 2KB に設定します。

CHUNK よりも大きいサイズに INITIAL および NEXT を設定する

LOB の記憶特性を明示的に指定する場合、LOB データ・セグメントの記憶域の INITIAL および NEXT が、チャンク・サイズより大きく設定されていることを確認してください。たとえば、データベース・ブロックのサイズが 2KB で、CHUNK を 8KB に指定する場合、INITIAL および NEXT が 8KB よりかなり大きいこと（16KB 以上）を確認してください。

INITIAL、NEXT または LOB CHUNK サイズの値を指定する場合は、次のことを確認してください。

- `CHUNK <= NEXT`
および
- `CHUNK <= INITIAL`

ENABLE | DISABLE STORAGE IN ROW

ENABLE | DISABLE STORAGE IN ROW 句を使用して、LOB を行内または行外に格納するかどうかを指定します。

注意： これは一度指定すると、変更できない場合があります。ENABLE STORAGE IN ROW から DISABLE STORAGE IN ROW に、およびその逆にも変更できません。

デフォルトは、ENABLE STORAGE IN ROW です。

小さい LOB (ENABLE または DISABLE STORAGE) と大きい LOB (ENABLE STORAGE)

行内に格納される LOB データの最大サイズは、最大 VARCHAR サイズ (4,000 バイト) です。この最大値には、LOB 値および制御情報が含まれます。LOB を行内に格納するように指定した場合、LOB 値および制御情報が 4,000 バイトを超えると、LOB 値は自動的に行外に移動されます。

このため、次のようなガイドラインが必要になります。

- **小さい LOB:** LOB が小さい場合 (4,000 バイト未満の場合)、LOB データを行外に格納すると、パフォーマンスが低下します。ただし、LOB を行内に格納すると、行のサイズが大きくなります。これは、ユーザーがフル・テーブル・スキャン、複数行アクセス (レンジ・スキャン)、LOB 列以外の列への多数の UPDATE/SELECT など、多くのベース表処理を行っている場合のパフォーマンスに影響します。

- **大きい LOB:** LOB データが 4,000 バイト未満にはならない場合 (すべての LOB が大きい場合) は、次の理由からデフォルトの `ENABLE STORAGE IN ROW` が最適です。
 - * LOB データは、4,000 を超えると自動的に行外に移動されます (LOB データはもともと大きいと想定しています)。
 - * LOB データを行外に格納しても、制御情報は行内に格納されているため、パフォーマンスは多少向上します。

GB の LOB の作成方法

Oracle8i では、LOB の上限は 4GB です。サイズが GB の LOB を作成するには、次のガイドラインに従って、LOB 記憶域用の表領域のすべての使用可能な領域を使用します。

- **単一データ・ファイルのサイズ制限:** 各オペレーティング・システム (OS) に対する単一データ・ファイルのサイズには制限があります。たとえば、Solaris 2.5 では、OS ファイルは 2GB 以下に制限されています。このため、Oracle データベースが実行している OS ファイルの最大許容サイズよりも LOB が大きくなった場合、表領域にさらに多くのデータ・ファイルを追加してください。
- **PCT INCREASE パラメータを 0 (ゼロ) に設定する:** LOB 記憶域句の `PCTINCREASE` パラメータでは、新しいエクステント・サイズのパーセントを指定します。LOB が表領域内でピース単位で格納される場合、多数の新しいエクステントがそのプロセスで作成されます。エクステントのサイズが毎回デフォルト値の 50% ずつ増加し続けると、エクステントが管理不可能なほど大きくなり、最終的に表領域内の領域が無駄になります。そのため、`PCTINCREASE` パラメータを 0 (ゼロ) または小さい値に設定する必要があります。
- **MAXEXTENTS を適切な値または UNLIMITED に設定する:** LOB 記憶域句の `MAXEXTENTS` を、LOB の計画されたサイズに合う適切な値に設定するか、または安全のため `UNLIMITED` に設定する必要があります。
- **大きいエクステント・サイズを使用する:** 作成されたすべての新しいエクステントに対して、Oracle8i は、ヘッダー、およびエクステントの他のメタデータに対するロールバック情報を生成します。エクステントの数が多の場合、ロールバック・セグメントが過剰に供給されます。これを避けるには、大きなエクステント・サイズ (100MB など) を選択してエクステント作成の頻度を削減するか、またはより頻繁にトランザクションをコミットして、ロールバック・セグメントの領域を再使用します。

例 : GB の LOB を格納するための表領域および表の作成

サイズが GB の LOB を格納できる表領域および表の作成の実際の例を、次に示します。マルチメディア表のビデオ・フレームが非常に大きくなる (GB) と予測される場合は、[第 8 章「サンプル・アプリケーション」](#) のマルチメディア・アプリケーションの例を参照してください。

```
CREATE TABLESPACE lobtbs1 datafile '/your/own/data/directory/lobtbs_1.dat' size
2000M reuse online nologging default storage (maxextents unlimited);
CREATE TABLESPACE lobtbs1 add datafile '/your/own/data/directory/lobtbs_2.dat' size
2000M reuse;
ALTER TABLESPACE lobtbs1 add datafile '/your/own/data/directory/lobtbs_2.dat' size
1000M reuse;

CREATE TABLE Multimedia_tab (
  Clip_ID      NUMBER NOT NULL,
  Story        CLOB default EMPTY_CLOB(),
  FLSub        NCLOB default EMPTY_CLOB(),
  Photo        BFILE default NULL,
  Frame        BLOB default EMPTY_BLOB(),
  Sound        BLOB default EMPTY_BLOB(),
  Voiced_ref   REF Voiced_typ,
  InSeg_ntab   InSeg_tab,
  Music        BFILE default NULL,
  Map_obj      Map_typ
  Comments     LONG
)
NESTED TABLE   InSeg_ntab STORE AS InSeg_nestedtab
LOB(Frame) store as (tablespace lobtbs1 chunk 32768 pctversion 0 NOCACHE
NOLOGGING
storage(initial 100M next 100M maxextents unlimited pctincrease 0));
```

LOB ロケータおよびトランザクション境界

LOB ロケータおよび操作の基本的な説明は、[第2章「基本コンポーネント」](#)を参照してください。

LOB ロケータのトランザクション境界および読取り一貫性のあるロケータの使用の詳細は、[第5章「高度なトピック」](#)を参照してください。

INSERT および UPDATE での 4,000 バイトを超えるバインド

LOB の INSERT および UPDATE で現在可能な 4,000 バイトを超えるバインド

今回のリリースでは、LOB の INSERT および UPDATE での 4,000 バイトを超えるデータのバインドがサポートされます。以前のリリースでは、この機能は LONG 列のみで可能でした。現在、LOB 列への INSERT または UPDATE については、次のとおりバインドできます。

- OCIBindByPos()、OCIBindByName() を使用した 4GB までのデータのバインド
- PL/SQL バインドを使用した 32,767 バイトまでのデータのバインド

1 行に複数の LOB を持つことができるため、同一の INSERT または UPDATE 文で、それらの各 LOB に対して、4GB までのデータをバインドできます。つまり、単一の文で、4,000 バイトを超える複数のバインドが可能です。

注意： LOB に指定するデフォルト値の長さは、これまでどおり 4,000 バイトに制限されています。

一時表領域が十分大きいことを確認する 4,000 バイトを超えるデータの LOB 列へのバインドでは、一時表領域の領域が使用されます。そのため、一時表領域が、LOB に対するすべてのバインド長の合計を保持するのに十分大きいことを確認してください。

一時表領域が拡張可能な場合、既存の領域が完全に使用された後、一時表領域が自動的に拡張されます。次の文を使用して、拡張可能な一時表領域を作成します。

```
CREATE TABLESPACE .. AUTOEXTEND ON ... TEMPORARY ..;
```

4,000 バイトを超えるバインド (HEX から RAW または RAW から HEX への変換なし)

Multimedia_tab 表については、[第 8 章「サンプル・アプリケーション」](#)を参照してください。次の例では、Comments という追加の列が使用されます。CREATE TABLE 構文に次の行を追加して、Comments 列を Multimedia_tab 表に追加する必要があります。

```
Comments LONG -- stores the comments of viewers on this clip
```

4,000 バイトを超えるデータでは、HEX から RAW や、RAW から HEX などの暗黙的な変換は行われません。

```

declare
  charbuf varchar(32767);
  rawbuf raw(32767);
begin
  charbuf := lpad('a', 12000, 'a');
  rawbuf := utl_raw.cast_to_raw(charbuf);

```

表 7-4 「4,000 バイトを超えるバインド: 可能な INSERT および UPDATE 操作」では、前述の例で可能な INSERT 操作および不可能な INSERT 操作が説明されています。これは、UPDATE 操作でも同じです。

表 7-4 4,000 バイトを超えるバインド: 可能な INSERT および UPDATE 操作

可能な INSERT/UPDATE	不可能な INSERT/UPDATE
<pre> INSERT INTO Multimedia_tab (story, sound) VALUES (charbuf, rawbuf); </pre>	<pre> INSERT INTO Multimedia_tab(sound) VALUES (charbuf); </pre> <p>この文では、HEX から RAW への暗黙的な変換が行われないため、処理されません。</p> <pre> INSERT INTO Multimedia_tab(story) VALUES (rawbuf); </pre> <p>この文では、HEX から RAW への暗黙的な変換が行われないため、処理されません。</p> <pre> INSERT INTO Multimedia_tab(sound) VALUES (utl_raw.cast_to_raw(charbuf)); </pre> <p>これは、utl_raw.cast_to_raw() 演算子と 4,000 バイトを超えるバインドを組み合わせることができないため、処理されません。</p>

SQL 演算子の結果に対する 4,000 バイトの制限

4,000 バイトを超えるデータを BLOB または CLOB にバインドし、そのデータが SQL 演算子で構成される場合、結果のサイズは最大 4,000 バイトに制限されます。

次の文では、LPAD の結果が 4,000 バイトに制限されているため、4,000 バイトのみ挿入されます。

```

INSERT INTO Multimedia_tab (story) VALUES (lpad('a', 5000, 'a'));

```

次の文では、LPAD の結果が 4,000 バイトに制限され、HEX から RAW への暗黙的な変換によって 2,000 バイトの RAW データに変換されるため、2,000 バイトのみ挿入されます。

```
INSERT INTO Multimedia_tab (sound) VALUES (lpad('a', 5000, 'a'));
```

4,000 バイトを超えるバインド : 制限事項

4,000 バイトを超えるバインドに対する制限事項を次に示します。

- 表に LONG 列と LOB 列の両方がある場合、LONG 列または LOB 列のいずれかに 4,000 バイトを超えるデータをバインドできますが、同一の文で両方にバインドすることはできません。
- どのようなサイズのデータも ADT の LOB 属性にバインドできません。この制限は、以前のリリースから存在しています。LOB 属性に対しては、まず空の LOB ロケータを挿入してから、OCILOB* 関数を使用して LOB の内容を変更します。
- INSERT AS SELECT 操作では、どのような長さのデータも LOB 列にバインドできません。この制限は、以前のリリースから存在しています。

例 : PL/SQL - INSERT および UPDATE での 4,000 バイトを超えるバインドの使用

```
CREATE TABLE foo (a INTEGER );
DECLARE
    bigtext    VARCHAR(32767);
    smalltext  VARCHAR(2000);
    bigraw     RAW (32767);
BEGIN
    bigtext    := LPAD('a', 32767, 'a');
    smalltext  := LPAD('a', 2000, 'a');
    bigraw     := utlraw.cast_to_raw (bigtext);

    /* 次は許可されています。*/
    INSERT INTO Multimedia_tab(clip_id, story, frame, comments)
        VALUES (1,bigtext, bigraw,smalltext);
    /* 次は許可されています。*/
    INSERT INTO Multimedia_tab (clip_id, story, comments)
        VALUES (2,smalltext, bigtext);

    bigtext    := LPAD('b', 32767, 'b');
    smalltext  := LPAD('b', 20, 'a');
    bigraw     := utlraw.cast_to_raw (bigtext);

    /* 次は許可されています。*/
    UPDATE Multimedia_tab SET story = bigtext, frame = bigraw,
        comments = smalltext;
```

```
/* 次は許可されています。*/
    UPDATE Multimedia_tab set story = smalltext, comments = bigtext;

/* LONG 列および LOB 列に 4,000 バイトを超えるデータを挿入しようとしているため、
次は許可されません。*/
    INSERT INTO Multimedia_tab (clip_id, story, comments)
        VALUES (5, bigtext, bigtext);

/* LOB 属性にデータを挿入しようとしているため、次は許可されません。*/
    INSERT into Multimedia_tab (clip_id,map_obj)
        VALUES (10,map_typ(NULL, NULL, NULL, NULL, NULL,bigtext, NULL));

/* INSERT AS SELECT データ INTO LOB 操作を実行しようとしているため、
次は許可されません。*/
    INSERT INTO Multimedia_tab (story) AS SELECT bigtext FROM foo;
END;
```

例 : PL/SQL - 4,000 バイトを超えるバインド (HEX から RAW / RAW から HEX への変換がサポートされていないため挿入は未サポート)

```
/* 4,000 バイトを超えるデータの暗黙的な変換 (RAW から HEX、RAW から HEX など)
は行われません。そのため、次のケースは正常に動作しません。 */

declare
    charbuf   varchar(32767);
    rawbuf    raw(32767);
begin
    charbuf := lpad ('a', 12000, 'a');
    rawbuf  := utl_raw.cast_to_raw(charbuf);

/* 次は許可されています。*/
    INSERT INTO Multimedia_tab (story, sound) VALUES (charbuf, rawbuf);

/* HEX から RAW への変換は行われなため、次は許可されません。*/
    INSERT INTO Multimedia_tab (sound) VALUES (charbuf);

/* RAW から HEX への変換は行われなため、次は許可されません。*/
    INSERT INTO Multimedia_tab (story) VALUES (rawbuf);
```

```
/* utl_raw.cast_to_raw() 演算子を 4,000 バイトを超えるバインドと結合できないため、
   次は許可されません。*/
INSERT INTO Multimedia_tab (sound) VALUES (utl_raw.cast_to_raw(charbuf));

end;
/
```

例 : PL/SQL - データに SQL 演算子が含まれる場合に 4,000 バイトを超えるバインドの結果を 4,000 バイトに制限

4,000 バイトを超えるデータを BLOB または CLOB にバインドし、データは実際に SQL 演算子で構成される場合、結果のサイズが 4,000 バイトに制限されます。

For example,

```
/* LPAD の結果が 4,000 バイトに制限されているため、次のコマンドでは、4,000 バイト
   のみが挿入されます。*/
```

```
INSERT INTO Multimedia_tab (story) VALUES (lpad('a', 5000, 'a'));
```

```
/* LPAD の結果が 4,000 バイトに制限され、HEX から RAW の暗黙的な変換によって
   2,000 バイトの RAW データに変換されるため、次のコマンドでは、2,000 バイト
   のみが挿入されます。*/
```

```
INSERT INTO Multimedia_tab (sound) VALUES (lpad('a', 5000, 'a'));
```

内部LOB用の Open、Close および IsOpen インタフェース

これらのインタフェースを使用すると、内部LOBをオープンおよびクローズし、内部LOBがオープンされているかどうかをテストできます。

Open/Close API ですべてのLOB操作をラップする必要はありません。LOBをあらかじめオープンしなくてもLOBに書き込む既存のアプリケーションに対して、この機能を追加しても影響はありません。これらのコールはリリース 8.0 には存在しないためです。

オープン性はLOBに対応付けられ、ロケータには対応付けられていないことに注意してください。ロケータは、ロケータが参照しているLOBがオープンしているかどうかに関する情報は格納しません。

Open/Close コール内でのLOB操作のラップ

- Open/Close コール操作内でLOB操作をラップしない場合:LOBを変更するとLOBは自動的にオープン、クローズされ、ドメイン・インデックス上の任意のトリガーが起動されます。この場合、LOB上のすべてのドメイン・インデックスは、LOBを変更するとすぐに更新されることに注意してください。したがって、LOBドメイン・インデックスは常に有効で、いつでも使用可能です。
- Open/Close 操作内でLOB操作をラップする場合:LOBが変更されても、そのたびにトリガーが起動されることはありません。そのかわりに、ドメイン・インデックス上のトリガーはCloseコールで起動されます。たとえば、Closeをコールするまでドメイン・インデックスが更新されないようにアプリケーションを設計できます。ただし、これは、LOB上の任意のドメイン・インデックスがOpen/Closeコールの間では有効にならないことを示します。

オープンLOB値がクローズされた場合のトランザクション

オープンしたLOB値をクローズする必要があるトランザクションの定義は、次のいずれかであることに注意してください。

- 「トランザクション (SELECT ... FOR UPDATE を含む) を開始する DML 文」と COMMIT との間
- 自律型トランザクション・ブロックの中

トランザクションがないときにオープンしているLOBは、セッションが終わる前にクローズする必要があります。セッションの終わりにオープンしているLOBがある場合、そのオープン性が破棄され、ドメイン・インデックス上のトリガーは起動されません。

トランザクションのコミット前におけるオープン中のすべての LOB のクローズ

トランザクションによってオープンされたすべての LOB をクローズする前にトランザクションをコミットすると、エラーが発生します。エラーが戻ると、LOB のオープン性は破棄されていますが、トランザクションは正常にコミットされます。

このため、トランザクションの LOB および非 LOB データに対するすべての変更はコミットされますが、ドメイン・インデックスに対してトリガーは起動されません。

注意： LOB への変更は、COMMIT がエラーを戻しても破棄されません。

トランザクション・ロールバック時には、そのトランザクションに対してまだオープンしているすべての LOB のオープン性は破棄されます。オープン性が破棄されるということは、LOB がクローズできなくなり、ドメイン・インデックス上でトリガーは起動されないということです。

同じ LOB の 2 回のオープンまたはクローズの禁止

異なるロケータまたは同じロケータを使用して、同じ LOB を 2 回オープン / クローズしてもエラーになります。

例 1: トランザクションにおける Open/Close コールの適切な使用方法

次に、トランザクションの内外において LOB に対する Open/Close コールの適切な使用方法を示します。

```
DECLARE
  Lob_loc1 CLOB;
  Lob_loc2 CLOB;
  Buffer   VARCHAR2(32767);
  Amount  BINARY_INTEGER := 32767;
  Position INTEGER := 1;
BEGIN
  /* LOB を選択します。 */
  SELECT Story INTO Lob_loc1 FROM Multimedia_tab WHERE Clip_ID = 1;

  /* 次の文によって LOB がトランザクションの外でオープンされるため、セッションが
  終わる前にその LOB をクローズする必要があります。 */
  DBMS_LOB.OPEN(Lob_loc1, DBMS_LOB.LOB_READONLY);
  /* 次の文によってトランザクションが開始されます。Lob_loc1 および Lob_loc2 が同じ
  LOB を参照していることに注意してください。 */
  SELECT Story INTO Lob_loc2 FROM Multimedia_tab WHERE Clip_ID = 1 for update;
  /* LOB がこのトランザクションでオープンされていないため、次の LOB のオープン
  操作は許可されます。 */
```

```
DBMS_LOB.OPEN(Lob_loc2, DBMS_LOB.LOB_READWRITE);
/* バッファにLOBに書き込むデータを充填します。*/
buffer := 'A good story';
Amount := 12;
/* バッファをLOBに書き込みます。*/
DBMS_LOB.WRITE(Lob_loc2, Amount, Position, Buffer);
/* LOBをオープンしている場合、そのLOBをクローズする必要があります。*/
DBMS_LOB.CLOSE(Lob_loc2);
/* COMMITによってトランザクションが終了します。トランザクションでオープンされた
すべてのLOBがクローズされているため、これは許可されます。*/
COMMIT;
/* 次の文によって、トランザクション開始前にオープンされたLOBがクローズされます。*/
DBMS_LOB.CLOSE(Lob_loc1);
END;
```

例 2: トランザクションにおける Open/Close コールの不適切な使用方法

次に、LOB に対する Open/Close コールの不適切な使用方法について、LOB をオープンしたトランザクションのコミットが、どのようにエラーを戻すのかを示します。

```
DECLARE
  Lob_loc CLOB;
BEGIN
  /* FOR UPDATE 句によってトランザクションが開始されることに注意してください。*/
  SELECT Story INTO Lob_loc FROM Multimedia_tab WHERE Clip_ID = 1 for update;
  DBMS_LOB.OPEN(Lob_loc, DBMS_LOB.LOB_READONLY);
  /* このトランザクションに対応付けられているLOBがオープンされているため、
  COMMITすると、エラーが戻されます。*/
  COMMIT;
END;
```

索引構成表 (IOT) 内の LOB

索引構成表 (IOT) は現在、内部 LOB 列および外部 LOB 列をサポートしています。索引構成表内の LOB に対する SQL DDL、DML およびピース単位操作は、従来型表内の操作と同じ動作になります。作成中の LOB のデフォルトの動作のみが異なります。主な違いは次のとおりです。

- **表領域のマッピング**: デフォルトで、または別に指定されていない限り、LOB のデータおよび索引セグメントは、索引構成表の主キー索引セグメントが作成された表領域の中に作成されます。
- **行内記憶域と行外記憶域**: デフォルトで、オーバーフロー・セグメントなしで作成された索引構成表内のすべての LOB は、行外に格納されます。つまり、索引構成表がオーバーフロー・セグメントなしで作成された場合、この表内の LOB のデフォルト記憶域属性は `DISABLE STORAGE IN ROW` になります。このような LOB に対して無理に `ENABLE STORAGE IN ROW` 句を指定しようとすると、SQL はエラーを表示します。

逆に、オーバーフロー・セグメントが指定されている場合、索引構成表内の LOB は従来型表の場合と同様に動作します (7-5 ページの「[内部 LOB に対する表領域および記憶特性の定義](#)」を参照)。

LOB 列を含む索引構成表 (IOT) の例

次の例について考えてみます。

```
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)));
```

これらの文を実行すると、次の要素を持つ索引構成表 `iotlob_tab` が作成されます。

- 表領域 `iot_ts` 内の主キー索引セグメント

- 表領域 `iot_o_ts` 内のオーバーフロー・データ・セグメント
- オーバーフロー・データ・セグメント内に明示的に格納される、C3 から始まる列
- 表領域 `lob_ts` 内の BLOB (列 C2) データ・セグメント
- 表領域 `lobidx_ts` 内の BLOB (列 C2) 索引セグメント
- 表領域 `iot_ts` 内の CLOB (列 C3) データ・セグメント
- 表領域 `iot_ts` 内の CLOB (列 C3) 索引セグメント
- IOT にオーバーフロー・セグメントがあるために行内に格納される CLOB (列 C3)
- 明示的に行外に強制格納される BLOB (列 C2)

注意： オーバーフローが指定されていない場合、C2 と C3 の両方がデフォルトで行外に格納されます。

BFILE や可変幅文字 LOB などの他の LOB 機能も索引構成表でサポートされ、その使用方法も従来型表の場合と同じです。

注意： パーティション化された索引構成表内の LOB は、将来のリリースでサポートされる予定です。

パーティション表内の LOB の操作

LOB を含む表をパーティション化できます。この結果、パーティション化のすべての利点を LOB でも利用できます。たとえば、LOB セグメントをいくつかの表領域に分散して、I/O 負荷を均衡化させ、バックアップおよびリカバリの管理をより簡単にできます。パーティション表内の LOB も、メンテナンスが簡単になります。

この項では、パーティション表内の LOB の操作方法をいくつか説明します。

[第 8 章「サンプル・アプリケーション」](#)で説明するマルチメディア・アプリケーション・サンプルの延長として、ドキュメンタリのプロデューサーが歴代の米国大統領に関連したクリップを製作しているとします。このクリップには、大統領の写真に演説および BGM が付いています。写真は PhotoLib_Tab アーカイブからとったものです。最も効率よく使用できるように、大統領の写真は、[図 7-1](#) に示す構造に従ってデータベースにロードされています。

[表 7-5 「Multimedia_tab 列」](#)に、Multimedia_tab の列について示します。

図 7-1 PHOTO_REF 参照を含む Multimedia_tab 表の構造

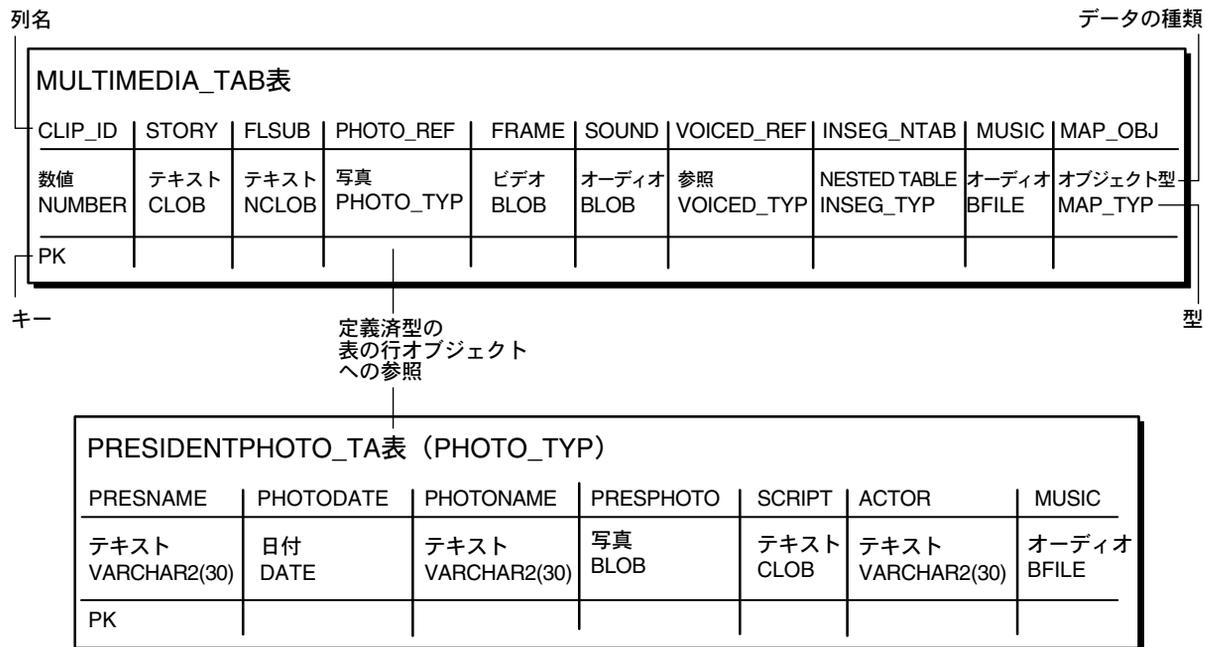


表 7-5 Multimedia_tab 列

列名	説明
PRESNAME	大統領の名前が含まれます。ドキュメンタリのプロデューサは、これを使用して特定の大統領に関して編成するクリップ用のデータを選択できます。PRESNAME は一意の値を保持するため、主キーとしても選択されます。
PRESPHOTO	大統領が写っている写真が含まれます。このカテゴリには、写真が登場する以前の大統領の肖像画および彫像の写真も含まれています。
PHOTODATE	写真の撮影日が含まれます。写真の登場する以前の大統領の場合、PHOTODATE には肖像画が描かれたり彫像が彫られた日付が含まれます。 この列はパーティション・キーに選択されており、パーティションの追加、および大統領の最初の任期終了日などのような指定された日付に基づいたデータの MERGE および SPLIT をより簡単に行うために使用されます。これについては、この項で後述します。
PHOTONAME	写真の名前が含まれます。この名前の例としては、「ブッシュ大統領の国際連合での演説、1990年6月」のように詳しいものも、「フランクリン・ルーズベルト、就任式」のような簡単なものもあります。
SCRIPT	写真に関連して書かれたテキストが含まれます。ここには、写真に写っているイベントの解説や、大統領の演説文などのテキストを入れます。
ACTOR	スクリプトを読む俳優の名前が含まれます。
MUSIC	写真を表示している間に演奏される BGM が含まれます。

LOB データを含む表の作成およびパーティション化

指定された大統領に対応付けられた写真を分離するため、大統領ごとに、それぞれの任期終了日によってパーティションが作成されます。たとえば、2 期努めた大統領には 2 つのパーティションが作成されます。1 つ目のパーティションは 1 期目の終了日で区切られ、2 つ目のパーティションは 2 期目の終了日で区切られます。

注意: 次の例では、拡張要素 1 は大統領の最初の任期を参照し、2 は大統領の 2 期目の任期を参照します。たとえば、GeorgeWashington1_part は、ジョージ・ワシントンの 1 期目のために作成されたパーティションを参照し、RichardNixon2_part はリチャード・ニクソンの 2 期目のためのパーティションを参照します。

注意： 一部の例が正しく動作するためには、次のように、データ構造を設定する必要があります。

```
CONNECT system/manager
GRANT CREATE TABLESPACE, DROP TABLESPACE TO scott;
CONNECT scott/tiger
CREATE TABLESPACE EarlyPresidents_tbs DATAFILE
'disk1:moredata01' SIZE 1M;
CREATE TABLESPACE EarlyPresidentsPhotos_tbs DATAFILE
'disk1:moredata99' SIZE 1M;
CREATE TABLESPACE EarlyPresidentsScripts_tbs DATAFILE
'disk1:moredata03' SIZE 1M;
CREATE TABLESPACE RichardNixon1_tbs DATAFILE
'disk1:moredata04' SIZE 1M;
CREATE TABLESPACE Post1960PresidentsPhotos_tbs DATAFILE
'disk1:moredata05' SIZE 1M;
CREATE TABLESPACE Post1960PresidentsScripts_tbs DATAFILE
'disk1:moredata06' SIZE 1M;
CREATE TABLESPACE RichardNixon2_tbs DATAFILE
'disk1:moredata07' SIZE 1M;
CREATE TABLESPACE GeraldFord1_tbs DATAFILE
'disk1:moredata97' SIZE 1M;
CREATE TABLESPACE RichardNixonPhotos_tbs DATAFILE
'disk1:moredata08' SIZE 2M;
CREATE TABLESPACE RichardNixonBigger2_tbs DATAFILE
'disk1:moredata48' SIZE 2M;
CREATE TABLE Mirrorlob_tab(
    PresName VARCHAR2(30),
    PhotoDate DATE,
    PhotoName VARCHAR2(30),
    PresPhoto BLOB,
    Script CLOB,
    Actor VARCHAR2(30),
    Music BFILE);
```

```
CREATE TABLE Presidentphoto_tab(PresName VARCHAR2(30), PhotoDate DATE,
    PhotoName VARCHAR2(30), PresPhoto BLOB,
    Script CLOB, Actor VARCHAR2(30), Music BFILE)
STORAGE (INITIAL 100K NEXT 100K PCTINCREASE 0)
LOB (PresPhoto) STORE AS (CHUNK 4096)
LOB (Script) STORE AS (CHUNK 2048)
PARTITION BY RANGE (PhotoDate)
(PARTITION GeorgeWashington1_part
```

```
/* ワシントンの第1期終了日に写真を使用します。*/
VALUES LESS THAN (TO_DATE('19-mar-1792', 'DD-MON-YYYY'))
TABLESPACE EarlyPresidents_tbs
LOB (PresPhoto) store as (TABLESPACE EarlyPresidentsPhotos_tbs)
LOB (Script) store as (TABLESPACE EarlyPresidentsScripts_tbs),
PARTITION GeorgeWashington2_part
/* ワシントンの第2期終了日に写真を使用します。*/
VALUES LESS THAN (TO_DATE('19-mar-1796', 'DD-MON-YYYY'))
TABLESPACE EarlyPresidents_tbs
LOB (PresPhoto) store as (TABLESPACE EarlyPresidentsPhotos_tbs)
LOB (Script) store as (TABLESPACE EarlyPresidentsScripts_tbs),
PARTITION JohnAdams1_part
/* アダムスの唯一の任期終了日に写真を使用します。*/
VALUES LESS THAN (TO_DATE('19-mar-1800', 'DD-MON-YYYY'))
TABLESPACE EarlyPresidents_tbs
LOB (PresPhoto) store as (TABLESPACE EarlyPresidentsPhotos_tbs)
LOB (Script) store as (TABLESPACE EarlyPresidentsScripts_tbs),
/* 大統領が介在しています。*/
PARTITION RichardNixon1_part
/* ニクソンの第1期終了日に写真を使用します。*/
VALUES LESS THAN (TO_DATE('20-jan-1972', 'DD-MON-YYYY'))
TABLESPACE RichardNixon1_tbs
LOB (PresPhoto) store as (TABLESPACE Post1960PresidentsPhotos_tbs)
LOB (Script) store as (TABLESPACE Post1960PresidentsScripts_tbs)
);
```

LOB 列を含む表の索引の作成

大統領の名前または写真の名前によってレコードにアクセスする問合せのパフォーマンスを改善するため、UNIQUE のローカル索引を作成します。

```
CREATE UNIQUE INDEX PresPhoto_idx
ON PresidentPhoto_tab (PresName, PhotoName, Photodate) LOCAL;
```

LOB データを含むパーティションの交換

Oracle 8.0 から 8.1 へのアップグレードの一部として、ビル・クリントンの 1 期目の写真を含む既存の非パーティション化表から該当するパーティションにデータを交換します。

```
ALTER TABLE PresidentPhoto_tab EXCHANGE PARTITION RichardNixon1_part
WITH TABLE Mirrorlob_tab INCLUDING INDEXES;
```

LOB データを含む表へのパーティションの追加

リチャード・ニクソンの 2 期目を処理するために、新しいパーティションが `PresidentPhoto_tab` に追加されます。

```
ALTER TABLE PresidentPhoto_tab ADD PARTITION RichardNixon2_part
VALUES LESS THAN (TO_DATE('20-jan-1976', 'DD-MON-YYYY'))
TABLESPACE RichardNixon2_tbs
LOB (PresPhoto) store as (TABLESPACE Post1960PresidentsPhotos_tbs)
LOB (Script) store as (TABLESPACE Post1960PresidentsScripts_tbs);
```

LOB を含むパーティションの移動

リチャード・ニクソンの 2 期目には非常に多くの写真があるため、彼の 2 期目の情報を含むパーティションは十分ではなくなっています。データ・パーティションを移動し、`PresidentPhoto_tab` のデータ・パーティションと対応する LOB パーティションを別の表領域に移動することにしました。該当する `Script` の LOB パーティションは元の表領域に残したままとします。

```
ALTER TABLE PresidentPhoto_tab MOVE PARTITION RichardNixon2_part
TABLESPACE RichardNixonBigger2_tbs
LOB (PresPhoto) STORE AS (TABLESPACE RichardNixonPhotos_tbs);
```

LOB を含むパーティションの分割

リチャード・ニクソンが 2 期目に再当選したときに、彼の任期の予期される最終日（1976 年 1 月 20 日）の区切りでパーティションが表に追加されました（前述の例を参照）。ニクソンは 1974 年 8 月 9 日に辞職したため、このパーティションはジェラルド・フォードが残りの任期を務めた事実を反映するように分割する必要があります。

```
ALTER TABLE PresidentPhoto_tab SPLIT PARTITION RichardNixon2_part
AT (TO_DATE('09-aug-1974', 'DD-MON-YYYY'))
INTO (PARTITION RichardNixon2_part,
PARTITION GeraldFord1_part TABLESPACE GeraldFord1_tbs
LOB (PresPhoto) STORE AS (TABLESPACE Post1960PresidentsPhotos_tbs)
LOB (Script) STORE AS (TABLESPACE Post1960PresidentsScripts_tbs));
```

LOB を含むパーティションのマージ

ドキュメンタリプロデューサーは、ジョージ・ワシントンの肖像画または彫像の写真を検索しましたが、見つかった写真の数は 2 つの任期をパーティションに分けるには十分な数ではありませんでした。このため、彼の 2 つのパーティションは `GeorgeWashington8Years_part` という名前の 1 つのパーティションにマージすることになりました。

```
ALTER TABLE PresidentPhoto_tab
MERGE PARTITIONS GeorgeWashington1_part, GeorgeWashington2_part
INTO PARTITION GeorgeWashington8Years_part TABLESPACE EarlyPresidents_tbs
LOB (PresPhoto) store as (TABLESPACE EarlyPresidentsPhotos_tbs)
LOB (Script) store as (TABLESPACE EarlyPresidentsScripts_tbs);
```

LOB 列の索引付け

LOB 列には、B-tree またはビットマップ索引を構築できません。ただし、アプリケーションおよびそのアプリケーションがどのように LOB 列を使用しているかによって、ドメイン専用チューニングされた索引を構築し、問合せのパフォーマンスを向上できる場合もあります。Oracle8i の拡張性のあるインタフェースによって、そのようなドメイン固有の索引を実装するための枠組みである、ドメイン・インデックス作成機能を利用できます。

参照： ドメイン固有の索引作成の詳細は、『Oracle8i データ・カートリッジ開発者ガイド』を参照してください。

LOB 列の内容の性質によっては、Oracle8i *interMedia* オプションの 1 つを使用して索引を作成することもできます。たとえば、テキスト・ドキュメントが CLOB 列に格納されている場合、テキスト索引 (Oracle が提供) を作成して、CLOB 列に対するテキストベースの問合せのパフォーマンスを向上させることができます。

参照： Oracle の *interMedia* オプションの詳細は、『Oracle8i *interMedia* Audio, Image, Video ユーザーズ・ガイドおよびリファレンス』および『Oracle8i *interMedia* Text リファレンス』を参照してください。

パフォーマンスの最適化のための最善策

SQL*Loader の使用

SQL*Loader を使用して、LOB をバルクロードできます。

参照：

- SQL*Loader の詳細は、4-5 ページの「[LOB ロード時の SQL*Loader の使用](#)」を参照してください。
- SQL*Loader の総合的な説明については、『Oracle8i ユーティリティ・ガイド』を参照してください。

パフォーマンスの最適化のためのガイドライン

次のガイドラインを使用して、LOB でのパフォーマンスを最適化します。

- **できるだけ一度に大きいデータのチャンクの読み込み/書き込みを行う：**LOB のサイズは大きい場合、LOB 値の大きいデータ・チャンクを一度に読み込みおよび書き込みすることによって、パフォーマンスが最適化されます。これは次のような場合に便利です。
 - a. クライアント側から LOB にアクセスしており、クライアントがサーバーと異なるノードにある場合、大量の読み込み / 書き込み時のネットワーク・オーバーヘッドが削減されます。
 - b. NOCACHE オプションを使用する場合、少量の読み込み / 書き込みを行うたびに I/O が発生します。大量の読み込み / 書き込みを行うことで I/O が削減されます。
 - c. LOB に書き込みを行うと、新しいバージョンの LOB CHUNK が作成されます。そのため、一度の書き込み量が少ないと、書き込みが行われるたびに新しいバージョンを作成するコストがかかります。ロギングがオンになっている場合、CHUNK は REDO ログにも格納されます。
- **LOB バッファリングを使用して、小さいチャンクのデータの読み込み/書き込みを行う：**クライアント側の小さい LOB データの読み込み / 書き込みを行う必要がある場合、LOB バッファリングを使用します。OCILOBEnableBuffering()、OCILOBDisableBuffering()、OCILOBFlushBuffer()、OCILOBWrite()、OCILOBRead() を参照してください。基本的に、小さい LOB データの読み込み / 書き込みを行う前には、LOB バッファリングをオンにします。

参照： LOB バッファリングの詳細は、5-19 ページの「[LOB バッファリング・サブシステム](#)」を参照してください。

- **コールバックとともに OCILobRead() および OCILobWrite() を使用する**: データが、LOB へおよび LOB からストリーム形式で送られます。入力時に、量パラメータに書込み全体の長さが設定されていることを確認してください。できるだけ、LOB チャンクの倍数で読み込みおよび書き込みを行います。
- **LOB に対してチェックアウト/チェックイン・モデルを使用する**: LOB は次の操作に対して最適化されます。
 - a. LOB 値全体を置換する SQL の UPDATE
 - b. クライアントに LOB データ全体をコピーし、その LOB データをクライアント側で変更し、LOB データ全体をデータベースに再度コピーする操作。これは、ストリームのある OCILobRead() および OCILobWrite() を使用して行われます。

スレッド環境における LOB へのデータの移動

不適切な手順

スレッド環境を使用する場合に新規の接続が必要となる次の手順は、パフォーマンスに悪影響を及ぼすため、適切ではありません。

1. 空の (NULL でない) LOB を作成します。
2. 空の LOB を使用して INSERT します。
3. 入力直後の行を SELECT-FOR-UPDATE します。
4. データを LOB に移動させます。
5. COMMIT します。これで SELECT-FOR-UPDATE ロックが解放され、LOB データが永続になります。

適切な手順

次のことに注意してください。

- 空の LOB を作成する必要はありません。
- INSERT/UPDATE 文に RETURNING 句を使用することで、ロックされた LOB ロケータを戻すことができます。これによって、ステップ 3 の SELECT-FOR-UPDATE を行う必要がなくなります。

このため、次の手順の実行をお勧めします。

1. 空の LOB を INSERT し、LOB ロケータを戻します。
2. このロケータを使用してデータを LOB に移動させます。
3. COMMIT します。これで SELECT-FOR-UPDATE ロックが解放され、LOB データが永続になります。

また、LOB 属性ではなく LOB 列の 4,000 バイトを超えるデータを直接挿入できます。

サンプル・アプリケーション

この章の内容は次のとおりです。

- [マルチメディア・コンテンツ収集システム](#)
- [アプリケーションへのオブジェクト・リレーショナル設計の適用](#)
- [Multimedia_tab 表の構造](#)

サンプル・アプリケーション

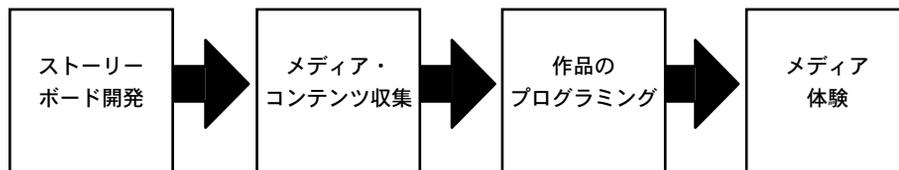
Oracle8i は、最大で 4GB のバイナリまたは文字データを保持できる LOB（ラージ・オブジェクト）をサポートしています。これは、アプリケーション開発者にとってどのような意味があるのでしょうか。

次の例について考えてみます。

マルチメディア・コンテンツ収集システム

マルチメディア・データが使用されるメディア・チャネルの種類は増加の一途をたどっています。中でもフィルム、テレビ、Web ページ、CD-ROM は最も普及しているメディアです。これらの異なるチャネルでは、メディアがどのように処理されるかは様々です（対話性、物理的な環境、情報の構造など）。このような違いはあっても、特に内容の組立てなどのマルチメディア・オーサリング処理は、高い類似性があります。

図 8-1 マルチメディア・オーサリング処理



たとえば、複雑なドキュメンタリを作成するテレビ局、テレビ用の広告を作成する広告代理店、Web 用の対話型ゲームを専門とするソフトウェア制作会社は、データベース管理システムを有効に使用してマルチメディア・データを収集および構成します。彼らはそれぞれ高度な編集ソフトウェアを使用してこれらの要素から特定の製品を作成するのですが、このような作業は複雑なため、マルチメディア要素を適切にグループ化するための、構成前のアプリケーションが必要になります。

たとえば、フィルムの制作について考えてみます。編成の基本的な単位としてクリップを使用するアプリケーションが考えられます。すべてのクリップは、次のメディアの種類のうち1つ以上を含むことができます。

- 文字テキスト（ストーリーボード、台本、字幕スーパーなど）
- 画像（写真、ビデオ・フレームなど）
- 描画（地図など）

- 音声（音響効果、音楽、インタビューなど）

このアプリケーションは編集する前のものであるため、クリップ内の要素の関係（写真と音声の同期など）、およびクリップ間の要素の関係（クリップの順序など）の正確な定義はされません。

このアプリケーションを使用すると、複数の編集者が作業して、異なる種類のマルチメディア・データを同時に格納、取出し、操作できます。ここでは、材料の一部を社内のデータベースから集めると想定します。また、専門業者からデータを購入およびダウンロードする可能性もあります。

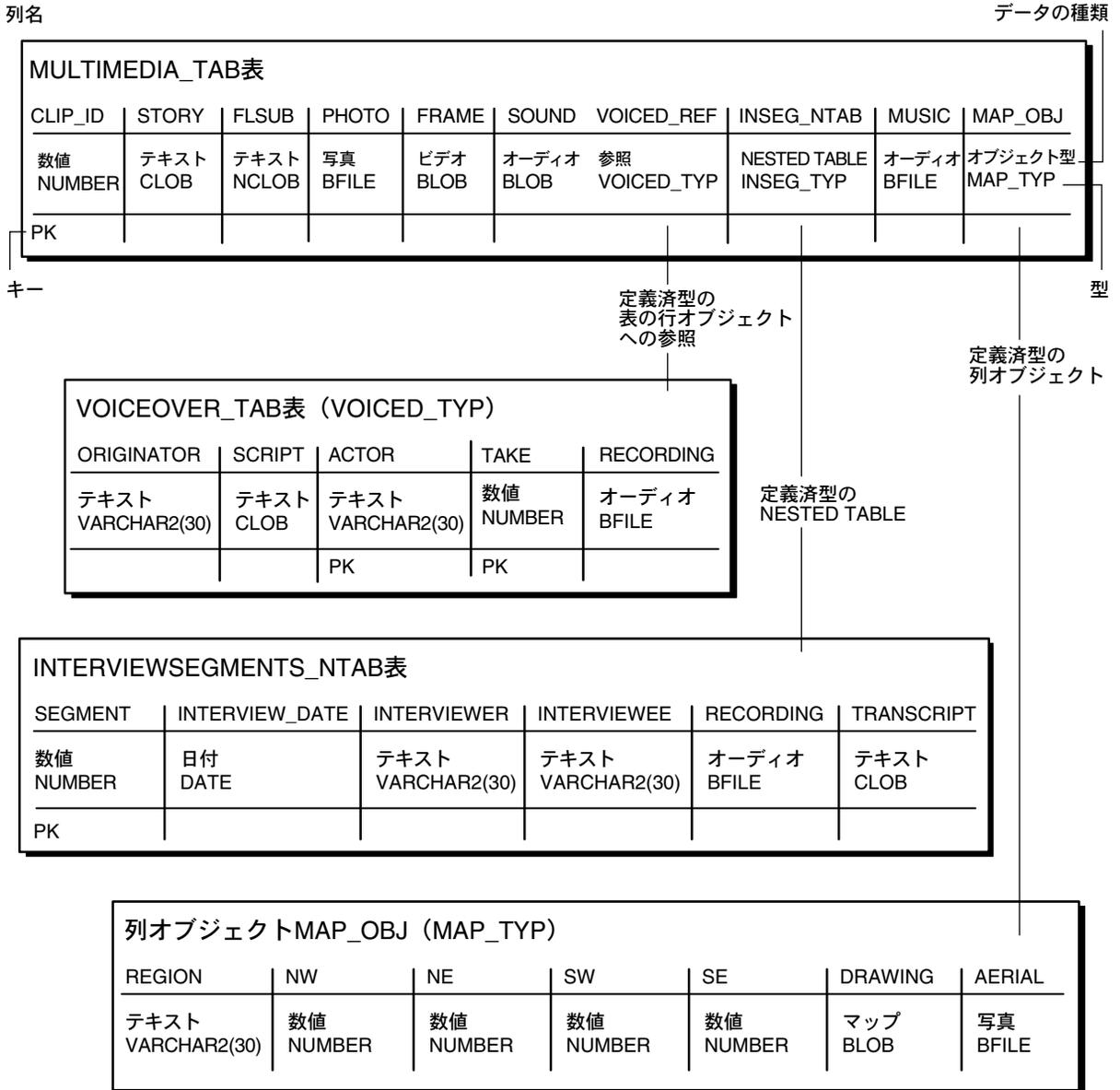
注意

この章の目的は、現実的なアプリケーションを作成することではなく、LOB を使用した作業で知っておく必要があるすべてのことを説明することです。したがって、この技術の説明に必要なアプリケーションの実装のみを行います。たとえば、扱うマルチメディアの種類は限定されています。LOB を操作するための、クライアント側のアプリケーションを作成することが目的ではありません。また、よいパフォーマンスを得るために LOB ファイルのディスク・ストライプ化を実装する必要があるなどの配置上の問題も扱いません。

図 8-2 「MULTIMEDIA_TAB 表のスキーマ計画」を参照してください。

アプリケーションへのオブジェクト・リレーショナル設計の適用

図 8-2 MULTIMEDIA_TAB 表のスキーマ計画



Multimedia_tab 表の構造

図 8-3 MULTIMEDIA_TAB 表のスキーマ計画

列名	MULTIMEDIA_TAB表									データの種類	
	CLIP_ID	STORY	FLSUB	PHOTO	FRAME	SOUND	VOICED_REF	INSEG_NTAB	MUSIC	MAP_OBJ	
	数値 NUMBER	テキスト CLOB	テキスト NCLOB	写真 BFILE	ビデオ BLOB	オーディオ BLOB	参照 VOICED_TYP	NESTED TABLE INSEG_TYP	オーディオ BFILE	オブジェクト型 MAP_TYP	
キー	PK										型

図 8-3 「MULTIMEDIA_TAB 表のスキーマ計画」に、MULTIMEDIA_TAB 表の構造を示します。その列を、次に説明します。

- **CLIP_ID:** すべての行（クリップ・オブジェクト）はそのクリップを識別する番号を持つ必要があります。この数字は便宜上、Oracle の番号 SEQUENCER で生成されるものを使用し、クリップの最終的な順序には関係ありません。
- **STORY:** アプリケーション・デザインは、すべてのクリップがクリップを記述するストーリーボードとしてのテキストを持つことを要求します。テキストの長さまたはフォーマットを制限しないため、CLOB データ型を使用します。
- **FLSUB:** 字幕スーパーには様々な使用方法があります。たとえば耳が不自由な人のための字幕としての使用、タイトルとしての使用、注意を引くためのオーバーレイなどとして使用します。完全なアプリケーションはこのような種類のデータそれぞれについて列を持っていますが、ここでは例として外国語の字幕スーパーという特定のケースのみを考え、NCLOB データ型を使用します。
- **PHOTO:** 写真は明らかにマルチメディア製品の主要項目です。PhotoLib_tab アーカイブに写真のライブラリが格納されていると想定します。この種の大規模データベースは定期的に更新される 3 次記憶装置に格納されるため、写真用の列は BFILE データ型を使用します。

- **FRAME:** 要素を動的なメディア・ソースから抽出してさらに処理することも必要になります。たとえば、VRML ゲームの作成者およびアニメーション作成者は個々のセルに関心を持つことが多いでしょう。このアプリケーションでは、ケネディ暗殺のフィルムで実行されたように、対象となるフィルムまたはビデオを、フレームごとに分析する必要性を取り上げています。また、ソースは永続的な記憶域にあり、アプリケーションでは個々のフレームを BLOB として格納できることを想定しています。
- **SOUND:** 音響効果で使用される BLOB 列です。
- **VOICED_REF:** この列を使用すると、表の中の特定の行を参照できます。この表は、`Voiced_typ` 型である必要があります。このアプリケーションでは、これは `VoiceOver_tab` 表の中の行の参照です。この表の目的は、ナレータのコメントとして使用される音声記録を格納することです。たとえば、本人が死んでしまった、または外国人であるため、音声記録を作成することができず、そのかわりに本人が語ったり書いたりした言葉を俳優が読み上げるなどという場合が考えられます。

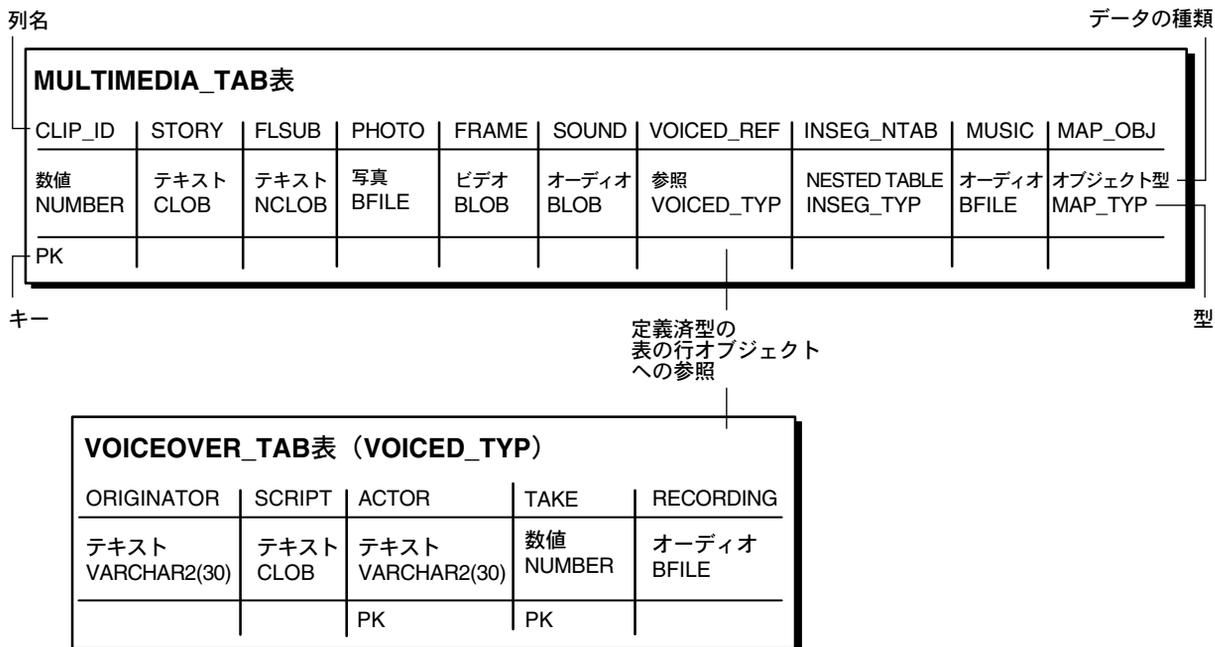
この構造を使用すると、アプリケーション制作者はこれまで説明してきた様々な異なる戦略が使用できます。アプリケーションは、資料をアーカイブのソースから行にロードするのではなく、データを参照するのみです。つまり、アプリケーション内の別の表から、または別のアプリケーションからも、同じデータを参照できるということです。唯一の条件は、参照は同じ型の表である必要があるということです。つまり、`Voiced_ref` の参照は、`Voiced_typ` 型に適合するどの表の行オブジェクトも参照できます。

`Voiced_typ` は LOB データ型を組み合わせて使用していることに注意してください。

- CLOB は、俳優が読む台本を格納します。
- BFILE は、音声記録を格納します。

図 8-4 「[VOICED_REF 参照を含むスキーマ設計](#)」に、`VoiceOver_tab` 表の `Voiced_typ` 行を参照する `VOIDED_REF` 列を示します。

図 8-4 VOICED_REF 参照を含むスキーマ設計



- **INSEG_NTAB:** LOB の VARRAY を格納することはできませんが、アプリケーション制作者は NESTED TABLE を使用して単一の行にマルチメディア要素の変数値を格納できます。この例では、事前定義済の InSeg_typ 型の NESTED TABLE、InSeg_ntab を使用して 0（ゼロ）、1つ、または多くのインタビューのセグメントを所定のクリップに格納できます。たとえば、この機能を使用すると、同じテーマに関連のある複数のインタビューのセグメントが異なる時間に起きた場合でも、1つに集めることができます。

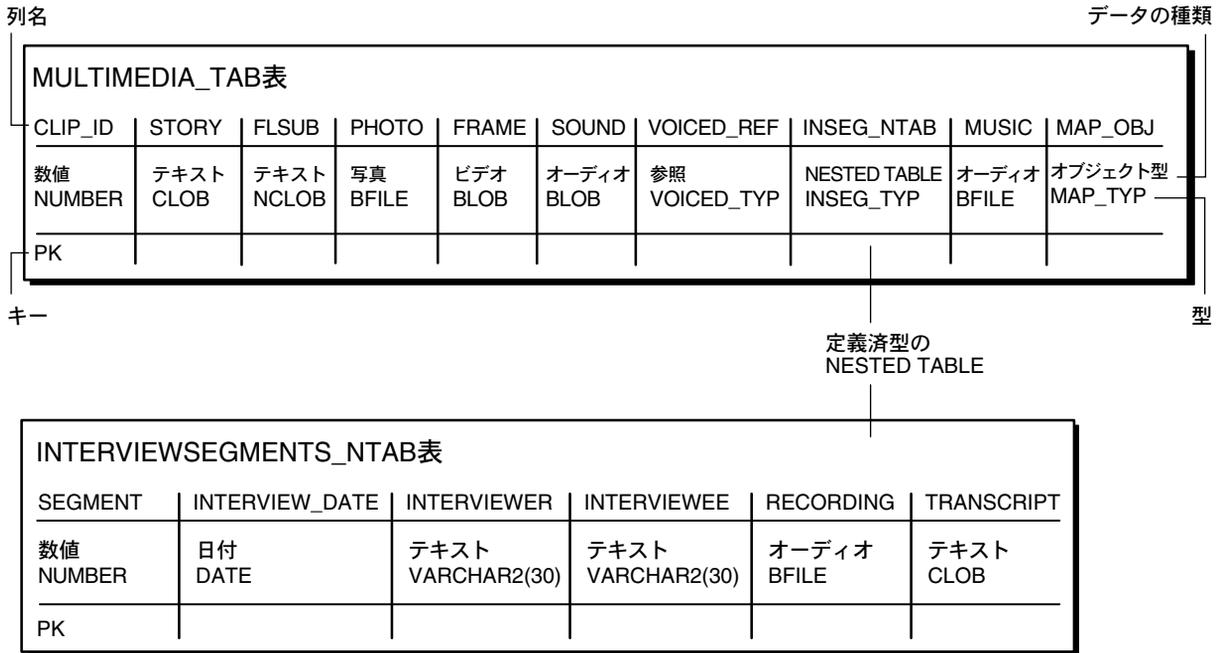
図 8-5 「NESTED TABLE、INSEG_NTAB の論理和のスキーマ設計」を参照してください。

この例では、NESTED TABLE の interviewsegments_ntab は、2 種類の LOB データ型を使用しています。

- BFILE は、インタビューの音声記録を格納します。
- CLOB は、台本を格納します。

このようなセグメントは非常に長くなるため、LOB が 4GB を超えることができないことに注意する必要があります。

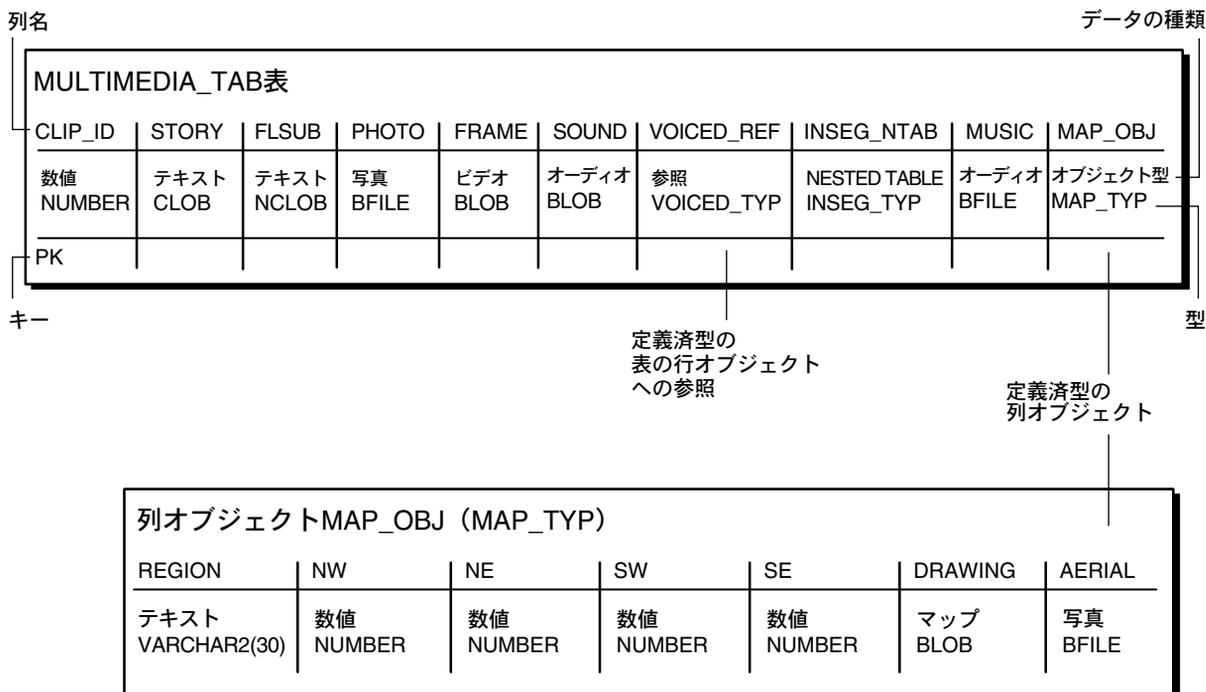
図 8-5 NESTED TABLE、INSEG_NTAB の論理和のスキーマ設計



- **MUSIC:** 音楽を扱う機能は、どのマルチメディア管理システムでも基本的な要件です。この例では、BFILE データ型を使用して、音声をオペレーティング・システム・ファイルとして格納します。
- **MAP_OBJ:** マルチメディア・アプリケーションは多くの異なる種類の描画を扱うことができる必要があります。たとえば、アニメ、図、芸術作品などです。このアプリケーションでは、クリップは、オブジェクト型 MAP_TYP の列オブジェクト MAP_OBJ としてマップを含む想定になっています。この場合、オブジェクトは行の中に埋め込まれた値に含まれます。

このアプリケーションの定義によると、MAP_TYP がその構造に含んでいる LOB は、描画用の BLOB のみです。ただし、REF 型および NESTED TABLE の場合のように、オブジェクト型が含む LOB の数には制限がありません。図 8-6 「列オブジェクト MAP_OBJ の論理和のスキーマ設計」を参照してください。

図 8-6 列オブジェクト MAP_OBJ の論理和のスキーマ設計



参照： LOB のその他の例については、次のマニュアルを参照してください。

- 『Oracle8i interMedia Audio、Image、Video ユーザーズ・ガイドおよびリファレンス』
- 『Oracle8i interMedia Audio、Image、Video Java Client ユーザーズ・ガイドおよびリファレンス』
- 『Oracle8i interMedia Locator ユーザーズ・ガイドおよびリファレンス』
- 『Oracle8i interMedia Web』
- 『Oracle8i interMedia Text 移行ガイド』
- 『Oracle8i interMedia Text リファレンス』

内部永続 LOB

ユースケース・モデル

この章では、内部永続 LOB を使用する操作について、ユースケースに沿って説明します。LOB に対する操作（LOB へのデータの書き込みなど）を、その操作名ごとにユースケースの点から説明します。表 9-1「ユースケース・モデル: 内部永続 LOB – 基本操作」に、すべてのユースケースを示します。

ユースケース・モデルの図形概要

「ユースケース・モデル図: 内部永続 LOB (1/2)」では、すべてのユースケースを 1 つの図にまとめています。

個々のユースケース

内部永続 LOB の個々のユースケースは、次のように説明されています。

- **ユースケース図**: ユースケースを表す図（図の見方については、「はじめに」の「図の解釈方法」を参照）
- **用途**: LOB に関するこのユースケースの用途
- **使用上の注意**: LOB 操作の実装に有効なガイドライン。
- **構文**: LOB 関連のアクティビティの実行に使用する主な構文
- **使用例**: ユースケース実装の例を、仮想マルチメディア・アプリケーションの点から表現した使用例。第 8 章「サンプル・アプリケーション」を参照してください。
- **例**: 各プログラム環境でのユースケースの例。これらは、第 8 章「サンプル・アプリケーション」で説明するマルチメディア・アプリケーションおよび Multimedia_tab 表を基にしています。

ユースケース・モデル: 内部永続 LOB

表 9-1 「ユースケース・モデル: 内部永続 LOB – 基本操作」の「+」は、特定のユースケースで、プログラム環境の例が提供されているものを示します。「S」は、SQL がそのユースケースおよび該当するプログラム環境に直接使用されていることを示します。

この表では、プログラム環境を次の略称で表しています。

- P – DBMS_LOB パッケージを使用した PL/SQL
- O – OCI (Oracle Call Interface) を使用した C
- B – Pro*COBOL プリコンパイラを使用した COBOL
- C – Pro*C/C++ プリコンパイラを使用した C/C++
- V – OO4O (Oracle Objects for OLE) を使用した Visual Basic
- J – JDBC (Java Database Connectivity) を使用した Java
- S – SQL

表 9-1 ユースケース・モデル: 内部永続 LOB – 基本操作

ユースケースおよびページ	プログラム環境の例					
	P	O	B	C	V	J
9-6 ページの「LOB を含む表を作成する 3 つの方法」						
9-8 ページの「1 つ以上の LOB 列を含む表の作成」	S	S	S	S	S	S
9-13 ページの「LOB 属性を持つオブジェクト型を含む表の作成」	S	S	S	S	S	S
9-18 ページの「LOB を含む NESTED TABLE の作成」	S	S	S	S	S	S
(LOB への参照を含む VARRAY の作成については、第 5 章「高度なトピック」を参照)	S	S	S	S	S	S
9-21 ページの「1 つ以上の LOB 値を行に挿入する 3 つの方法」						
9-23 ページの「EMPTY_CLOB() または EMPTY_BLOB() を使用した LOB 値の挿入」	S	S	S	S	S	+
9-27 ページの「別の表からの LOB の選択による行の挿入」	S	S	S	S	S	S
9-29 ページの「初期化した LOB ロケータ・バインド変数を使用した行の挿入」	S	+	+	+	+	+
9-38 ページの「内部 LOB (BLOB、CLOB、NCLOB) へのデータのロード」	+					
9-40 ページの「LOB への BFILE データのロード」	+	+	+	+	+	+
9-51 ページの「LOB がオープンしているかどうかの確認」	+	+	+	+		+
9-59 ページの「LOB への LONG のコピー」	S	S	S	S	S	S

ユースケースおよびページ	プログラム環境の例					
	P	O	B	C	V	J
9-64 ページの「LOB のチェックアウト」	+	+	+	+	+	+
9-75 ページの「LOB のチェックイン」	+	+	+	+	+	+
9-89 ページの「LOB データの表示」	+	+	+	+	+	+
9-99 ページの「LOB からのデータの読み込み」	+	+	+	+	+	+
9-110 ページの「LOB の一部の読み込み (substr)」	+		+	+	+	+
9-118 ページの「2 つの LOB の全体または一部の比較」	+		+	+	+	+
9-126 ページの「LOB 内のパターンの有無の確認 (instr)」	+		+	+		+
9-133 ページの「LOB の長さの取得」	+	+	+	+	+	+
9-141 ページの「別の LOB への LOB の全体または一部のコピー」	+	+	+	+	+	+
9-152 ページの「LOB ロケータのコピー」	+	+	+	+	+	+
9-160 ページの「2 つの LOB ロケータが等しいかどうかの確認」		+		+		+
9-167 ページの「LOB ロケータが初期化されているかどうかの確認」		+		+		
9-172 ページの「キャラクタ・セット ID の取得」		+				
9-175 ページの「キャラクタ・セット・フォームの取得」		+				
9-178 ページの「他の LOB への LOB の追加」	+	+	+	+	+	+
9-188 ページの「LOB への追加の書き込み」	+	+	+	+		+
9-197 ページの「LOB へのデータの書き込み」	+	+	+	+	+	+
9-213 ページの「LOB データの切捨て」	+	+	+	+	+	+
9-223 ページの「LOB の一部の消去」	+	+	+	+	+	+
9-232 ページの「LOB バッファリングの使用可能化」				+	+	
9-238 ページの「バッファのフラッシュ」		+	+	+		
9-244 ページの「LOB バッファリングの使用禁止化」		+	+	+	+	
9-252 ページの「LOB または LOB データ全体を更新する 3 つの方法」						
9-254 ページの「EMPTY_CLOB() または EMPTY_BLOB() を使用した LOB の更新」	S	S	S	S	S	S
9-257 ページの「別の表からの LOB の選択による行の更新」	S	S	S	S	S	S
9-259 ページの「初期化した LOB ロケータ・バインド変数を使用した更新」	S	+	+	+	+	+
9-267 ページの「LOB を含む表の行の削除」	S	S	S	S	S	S

図 9-1 ユースケース・モデル図：内部永続LOB (1/2)

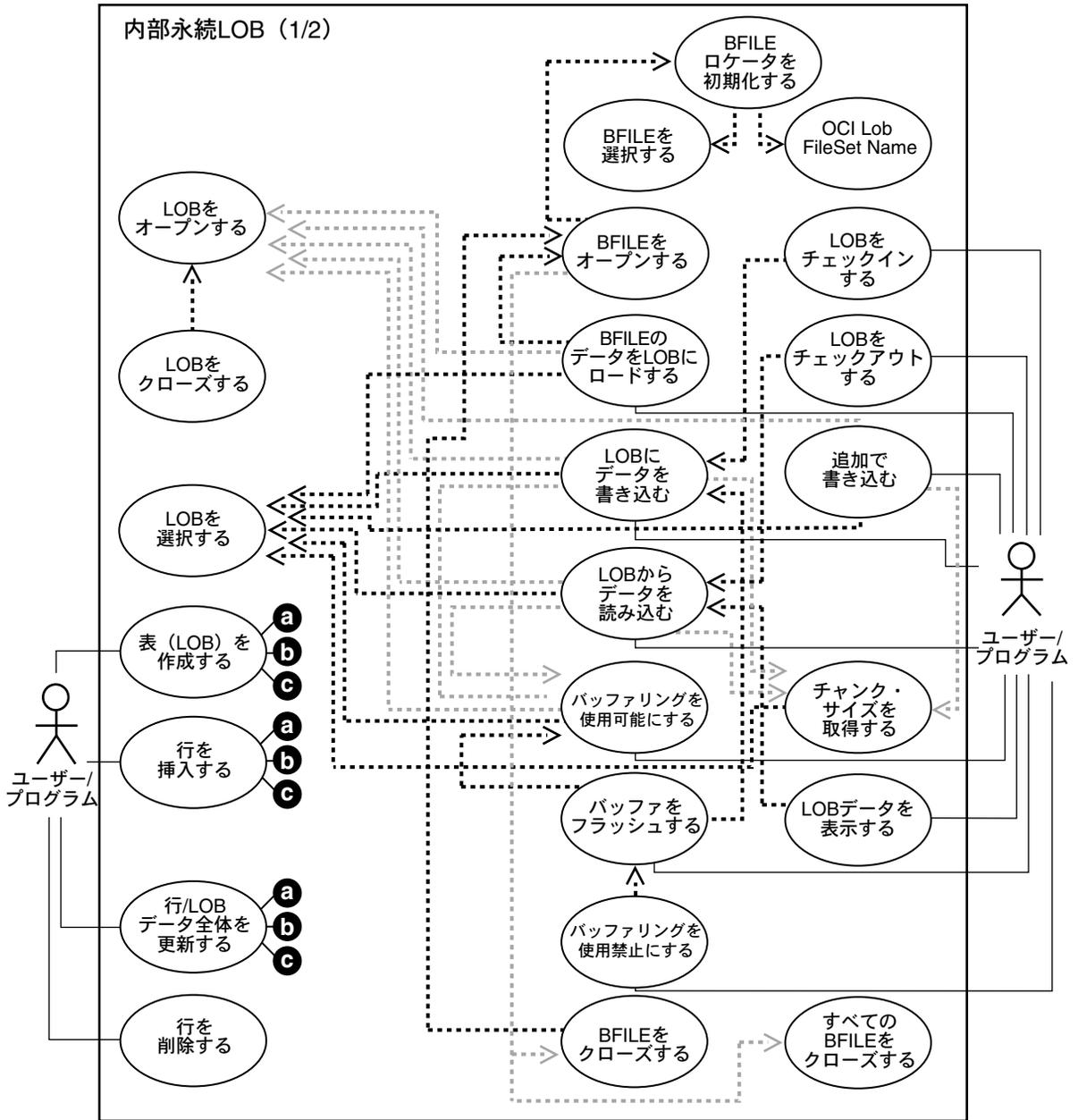
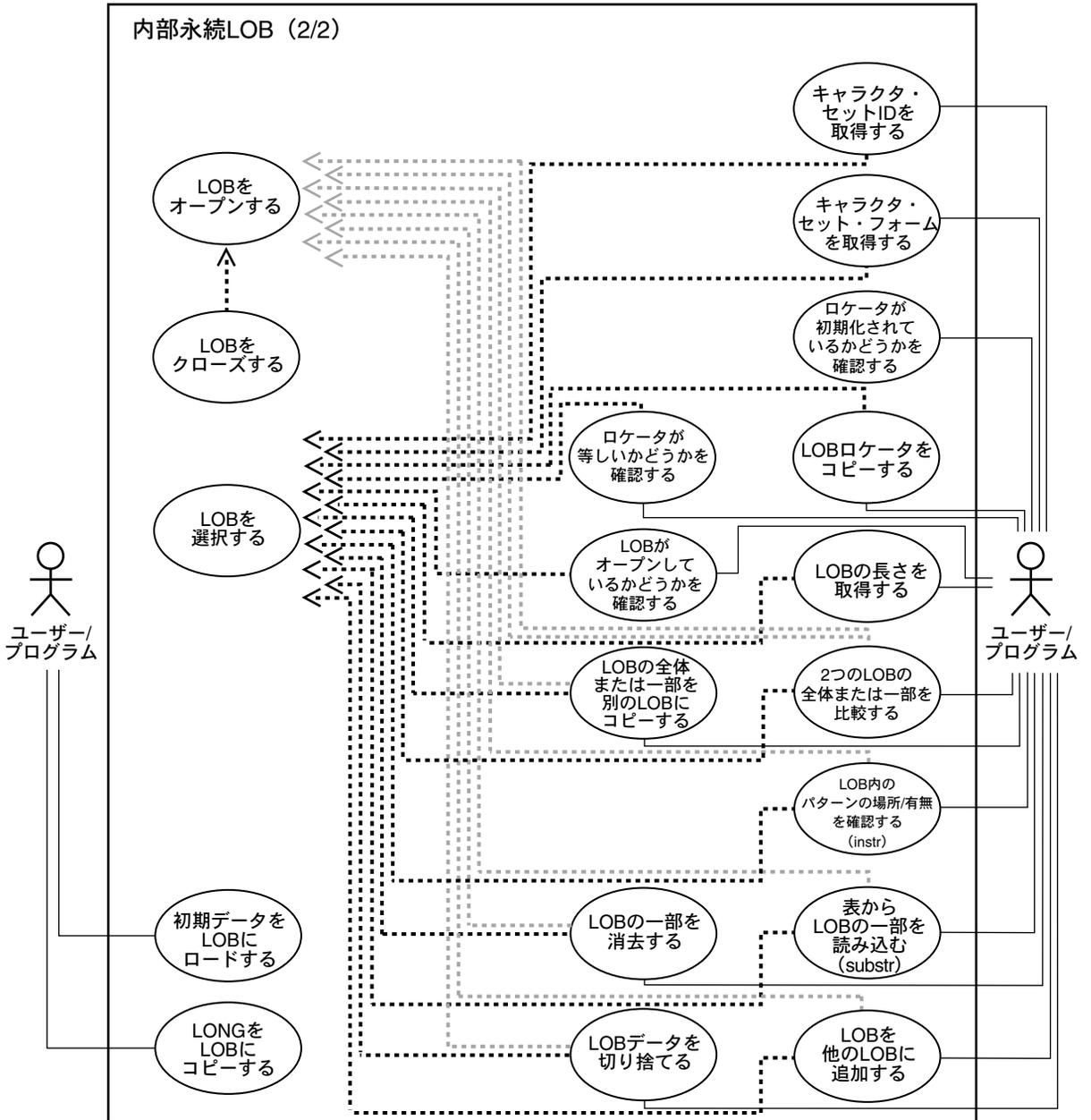
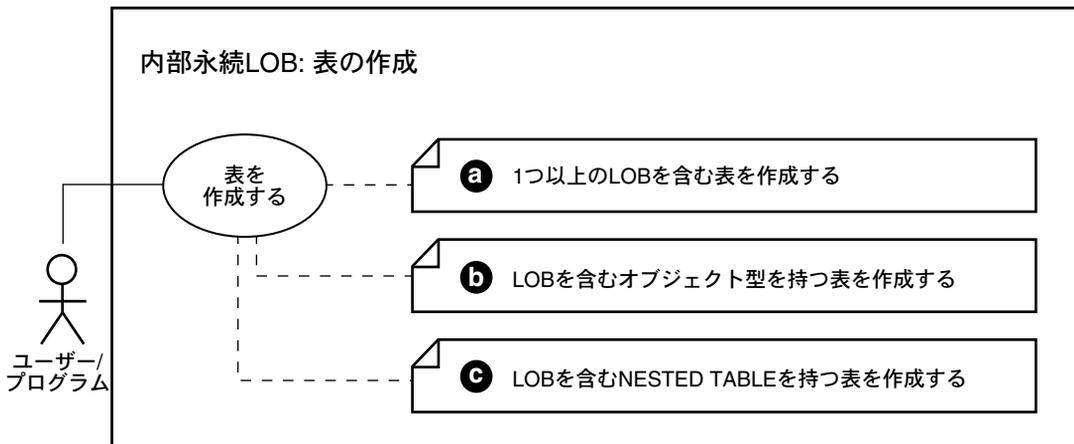


図 9-2 ユースケース・モデル図：内部永続LOB (2/2)



LOB を含む表を作成する 3 つの方法

図 9-3 ユースケース図 : LOB を含む表を作成する 3 つの方法



参照: 内部永続 LOB に関するすべての基本操作については、9-2 ページの「ユースケース・モデル:内部永続 LOB – 基本操作」を参照してください。

LOB を表に取り込むには、次の 3 つの方法があります。

- 表の列として取り込む方法 – 9-8 ページの「1つ以上の LOB 列を含む表の作成」を参照してください。
- オブジェクト型の属性として取り込む方法 – 9-13 ページの「LOB 属性を持つオブジェクト型を含む表の作成」を参照してください。
- NESTED TABLE 内に取り込む方法 – 9-18 ページの「LOB を含む NESTED TABLE の作成」を参照してください。

VARRAY を使用した 4 つ目の方法 – LOB への参照を含む VARRAY の作成方法については、5-28 ページの「LOB への参照を含む VARRAY の作成」を参照してください。

いずれの場合にも、SQL データ定義言語 (DDL) を使用して、表の LOB 列およびオブジェクト型の LOB 属性が定義されます。

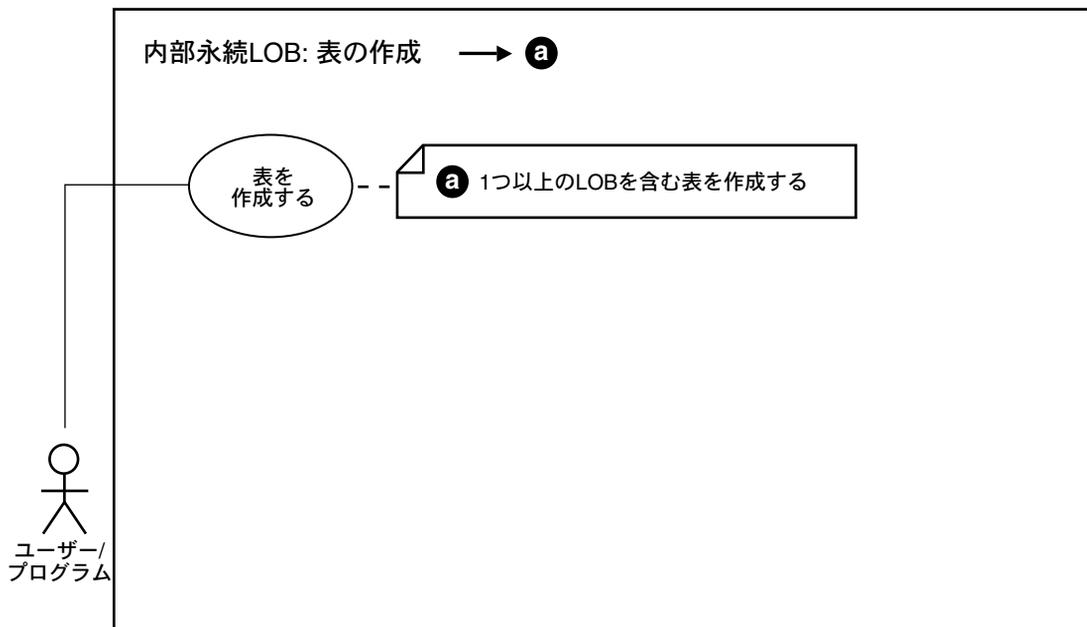
使用上の注意

LOB を含む表を作成するときは、次の章および項に説明されているガイドラインおよび例に従ってください。

- 第2章「基本コンポーネント」の「内部 LOB を NULL または空として初期化」
- 第4章「LOB の管理」
- 第7章「モデリングおよび設計」

1つ以上のLOB列を含む表の作成

図 9-4 ユースケース図：LOB列を含む表の作成



参照： 内部永続LOBに関するすべての基本操作については、9-2 ページの「ユースケース・モデル：内部永続LOB – 基本操作」を参照してください。

用途

1つ以上のLOB列を含む表を作成します。

使用上の注意

- `EMPTY_BLOB()` および `EMPTY_CLOB()` 関数を使用すると、LOB は初期化されますが、データは移入されません。空のLOBはNULLではありません。また、NULLのLOBは空ではありません。詳細は、9-23 ページの「[EMPTY_CLOB\(\) または EMPTY_BLOB\(\) を使用したLOB値の挿入](#)」を参照してください。

- 1 つ以上の LOB データ型の列を含む NESTED TABLE の作成の詳細は、9-18 ページの「LOB を含む NESTED TABLE の作成」を参照してください。
- 1 つ以上の LOB を含むオブジェクト列の作成の詳細は、9-13 ページの「LOB 属性を持つオブジェクト型を含む表の作成」を参照してください。

参照：

次のものを含む CREATE TABLE および ALTER TABLE 文で、LOB を使用する場合の構文については、『Oracle8i SQL リファレンス』を参照してください。

- BLOB、CLOB、NCLOB および BFILE 列
- EMPTY_BLOB および EMPTY_CLOB 関数
- 埋込みオブジェクトの LOB 属性および内部 LOB 列に対する LOB 記憶域句

構文

次のマニュアルの項を参照してください。

- SQL: 『Oracle8i SQL リファレンス』の第7章「SQL 文」の「CREATE TABLE」

使用例

次のアプリケーション例では、Multimedia_tab 表を中心としています。この表の列を構成する様々な型によって、クリップの作成に使用される何種類ものマルチメディア要素を 1 つにまとめることができます。

図 9-5 LOB 列を含む表を作成する例としての MULTIMEDIA_TAB

列名	データの種類								
MULTIMEDIA_TAB表									
CLIP_ID	STORY	FLSUB	PHOTO	FRAME	SOUND	VOICED_REF	INSEG_NTAB	MUSIC	MAP_OBJ
数値 NUMBER	テキスト CLOB	テキスト NCLOB	写真 BFILE	ビデオ BLOB	オーディオ BLOB	参照 VOICED_TYP	NESTED TABLE INSEG_TYP	オーディオ BFILE	オブジェクト型 MAP_TYP
PK									

キー

型

例

LOB 列を含む表の作成方法の例が SQL で示されています。

- [SQL: 1 つ以上の LOB 列を含む表の作成](#)

SQL: 1 つ以上の LOB 列を含む表の作成

次のようなデータ構造を設定しないと機能しない例もあります。

```
CONNECT system/manager;
DROP USER samp CASCADE;
DROP DIRECTORY AUDIO_DIR;
DROP DIRECTORY FRAME_DIR;
DROP DIRECTORY PHOTO_DIR;
DROP TYPE InSeg_typ force;
DROP TYPE InSeg_tab;
DROP TABLE InSeg_table;
CREATE USER samp identified by samp;
GRANT CONNECT, RESOURCE to samp;
CREATE DIRECTORY AUDIO_DIR AS '/tmp/';
CREATE DIRECTORY FRAME_DIR AS '/tmp/';
CREATE DIRECTORY PHOTO_DIR AS '/tmp/';
GRANT READ ON DIRECTORY AUDIO_DIR to samp;
GRANT READ ON DIRECTORY FRAME_DIR to samp;
GRANT READ ON DIRECTORY PHOTO_DIR to samp;
CONNECT samp/samp
CREATE TABLE a_table (blob_col BLOB);
```

```
CREATE TYPE Voiced_typ AS OBJECT (  
    Originator    VARCHAR2(30),  
    Script        CLOB,  
    Actor         VARCHAR2(30),  
    Take          NUMBER,  
    Recording     BFILE  
);  
  
CREATE TABLE VoiceoverLib_tab of Voiced_typ (  
    Script DEFAULT EMPTY_CLOB(),  
    CONSTRAINT TakeLib CHECK (Take IS NOT NULL),  
    Recording DEFAULT NULL  
);  
  
CREATE TYPE InSeg_typ AS OBJECT (  
    Segment       NUMBER,  
    Interview_Date DATE,  
    Interviewer   VARCHAR2(30),  
    Interviewee   VARCHAR2(30),  
    Recording     BFILE,  
    Transcript     CLOB  
);  
  
CREATE TYPE InSeg_tab AS TABLE of InSeg_typ;  
CREATE TYPE Map_typ AS OBJECT (  
    Region        VARCHAR2(30),  
    NW            NUMBER,  
    NE            NUMBER,  
    SW            NUMBER,  
    SE            NUMBER,  
    Drawing       BLOB,  
    Aerial        BFILE  
);  
  
CREATE TABLE Map_Libtab of Map_typ;  
CREATE TABLE Voiceover_tab of Voiced_typ (  
    Script DEFAULT EMPTY_CLOB(),  
    CONSTRAINT Take CHECK (Take IS NOT NULL),  
    Recording DEFAULT NULL  
);
```

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

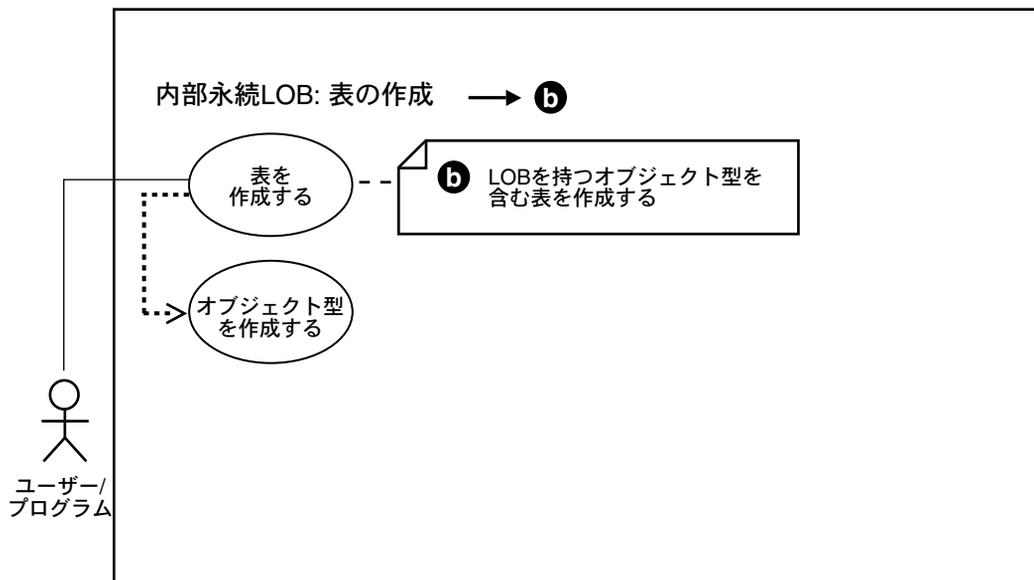
```
CREATE TABLE Multimedia_tab (  
    Clip_ID       NUMBER NOT NULL,  
    Story         CLOB default EMPTY_CLOB(),
```

1つ以上の LOB 列を含む表の作成

```
FLSub          NCLOB default EMPTY_CLOB(),
Photo          BFILE default NULL,
Frame          BLOB default EMPTY_BLOB(),
Sound          BLOB default EMPTY_BLOB(),
Voiced_ref     REF Voiced_typ,
InSeg_ntab     InSeg_tab,
Music          BFILE default NULL,
Map_obj        Map_typ
) NESTED TABLE InSeg_ntab STORE AS InSeg_nestedtab;
```

LOB 属性を持つオブジェクト型を含む表の作成

図 9-6 ユースケース図：LOB 属性を持つオブジェクト型を含む表の作成



参照： 内部永続 LOB に関するすべての基本的な操作については、9-2 ページの「ユースケース・モデル：内部永続 LOB – 基本操作」を参照してください。

用途

LOB 属性を持つオブジェクト型を含む表を作成します。

使用上の注意

ありません。

構文

次のマニュアルの項を参照してください。

- SQL: 『Oracle8i SQL リファレンス』の第7章「SQL 文」の「CREATE TABLE」

使用例

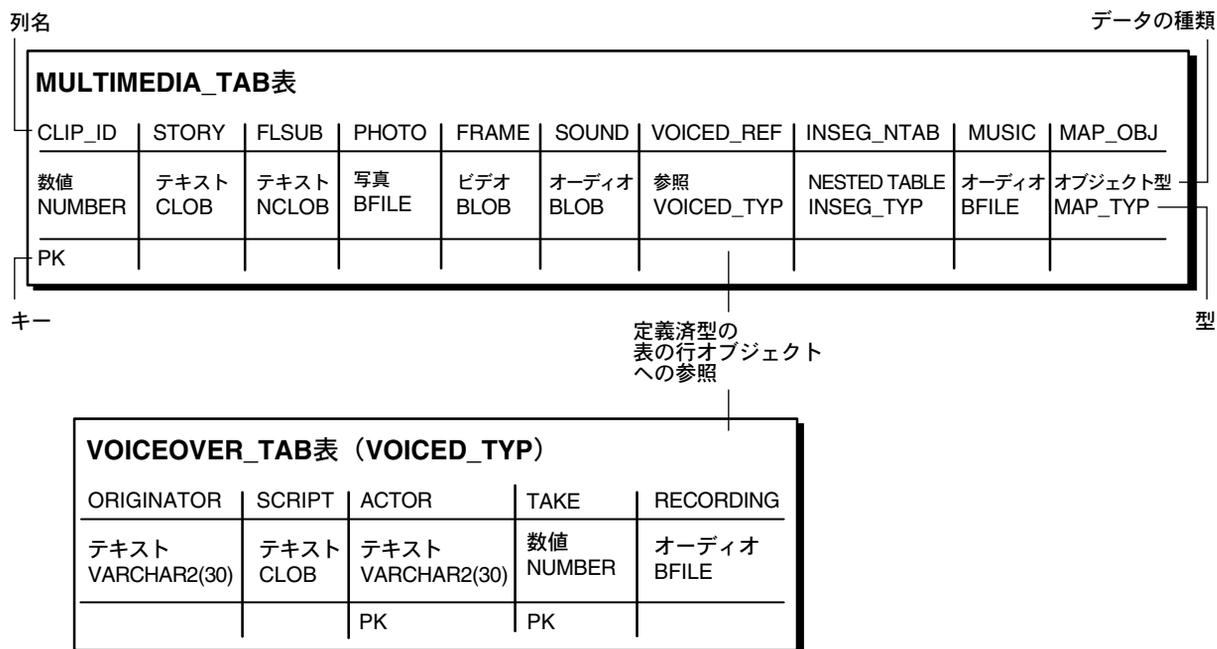
図に示されているように、LOB 属性を含むオブジェクト型を作成してから、そのオブジェクト型を使用する表を作成する必要があります。

アプリケーション例では、次の2つの方法でオブジェクト型にLOBが含まれています。

- **Voiced_typ データ型は、台本用に CLOB を、音声用に BFILE を使用します。**
Multimedia_tab 表には、Voiced_typ 型に基づく VoiceOver_tab 表の中の行オブジェクトを参照する Voiced_ref 列が含まれます。この型には、CLOB および BFILE の2種類のLOBが含まれます。CLOB は俳優が読む台本を格納し、BFILE は音声録音を保持します。
- **Map_obj 列は、BLOB を使用してマップを格納します。** Multimedia_tab 表は、Map_typ 型の列オブジェクトを含む Map_obj 列を含んでいます。この型は BLOB データ型を使用して、マップを描画形式で格納します。

参照： マルチメディア・アプリケーションおよび Multimedia_tab 表の詳細は、[第8章「サンプル・アプリケーション」](#)を参照してください。

図 9-7 LOB を含む型を作成する例としての VOICED_TYP



例

例が SQL で示されています。この例は、すべてのプログラム環境に適用されます。

- [SQL: LOB 属性を持つオブジェクト型を含む表の作成](#)

SQL: LOB 属性を持つオブジェクト型を含む表の作成

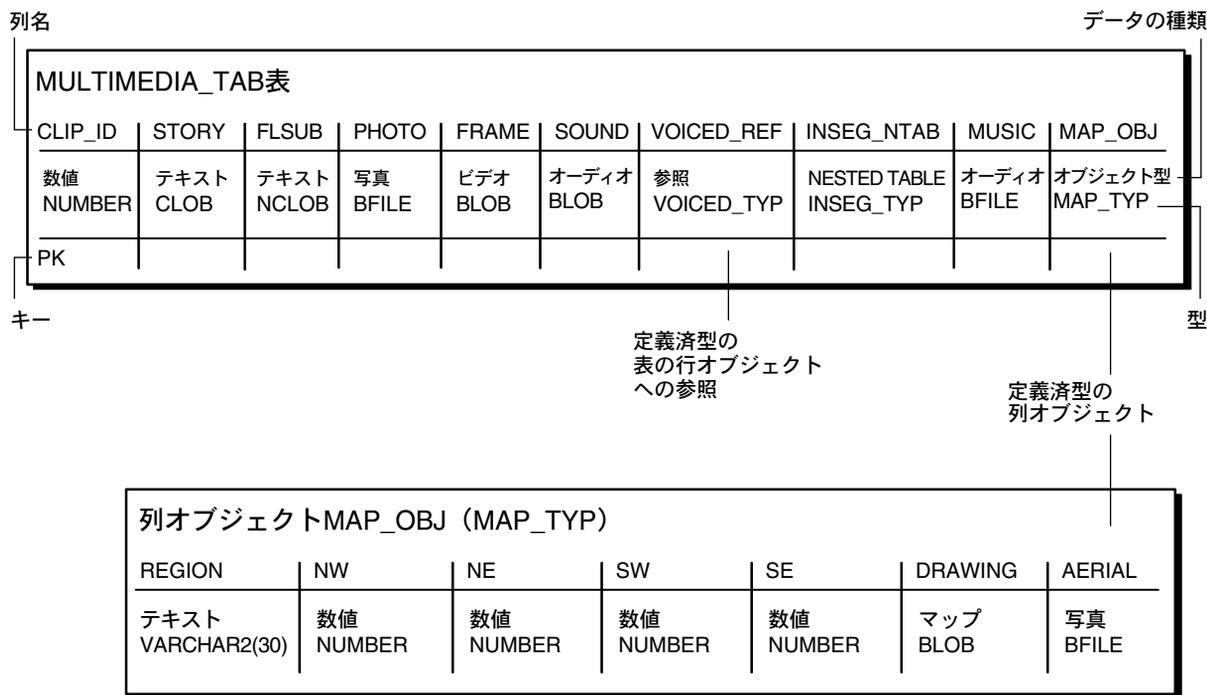
```
/* SQL DDL を使用して、型 Voiced_typ をナレーションの記録を含むことが  
できる表の基礎として作成します。*/
```

```
CREATE TYPE Voiced_typ AS OBJECT (
  Originator    VARCHAR2 (30) ,
  Script        CLOB,
  Actor         VARCHAR2 (30) ,
  Take          NUMBER,
  Recording     BFILE
);
```

```
/* SQL DDL を使用して、Voiceover_tab 表を作成します。*/
```

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

図 9-8 LOB を含む型を作成する例としての MAP_TYP



参照： 内部永続 LOB に関するすべての基本操作については、9-2 ページの「ユースケース・モデル:内部永続 LOB – 基本操作」を参照してください。

```
/* SQL DDLを使用して、型 Map_typ を列オブジェクトを含む表の基礎として作成します。*/  
CREATE TYPE Map_typ AS OBJECT (  
    Region          VARCHAR2(30),  
    NW              NUMBER,  
    NE              NUMBER,  
    SW              NUMBER,  
    SE              NUMBER,  
    Drawing         BLOB,  
    Aerial          BFILE  
);
```

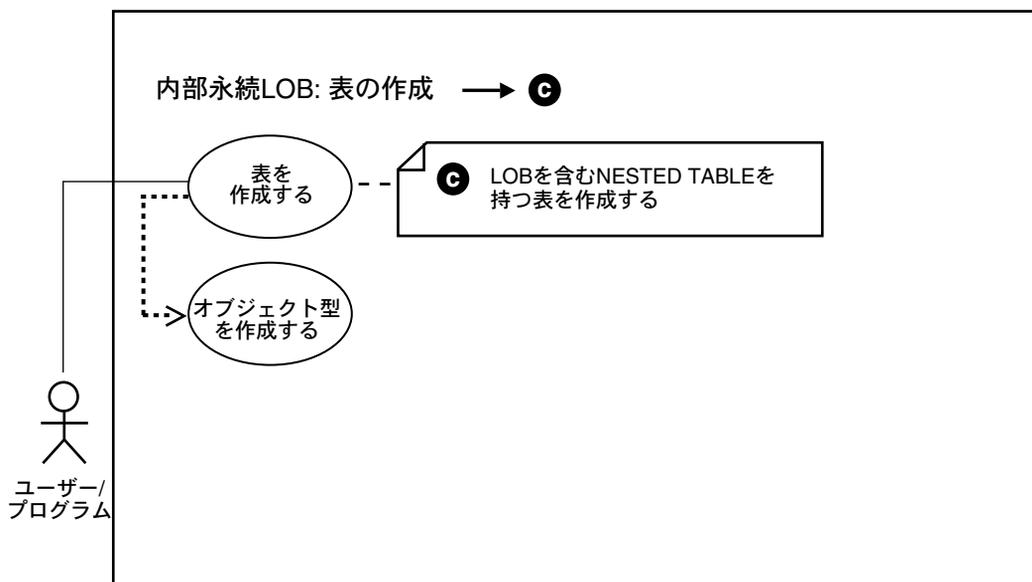
```
/* SQL DDLを使用して、サポート表 MapLib_tab をマップのアーカイブとして作成します。*/  
CREATE TABLE MapLib_tab of Map_typ;
```

参照： BLOB、CLOB および BFILE 属性を伴う DDL コマンド、CREATE TYPE および ALTER TYPE で、LOB を使用する場合の構文については、『Oracle8i SQL リファレンス』を参照してください。

注意： NCLOB はオブジェクト型の属性にはなりません。

LOB を含む NESTED TABLE の作成

図 9-9 ユースケース図 : LOB を含む NESTED TABLE の作成



参照： 内部永続 LOB に関するすべての基本操作については、9-2 ページの「ユースケース・モデル:内部永続 LOB – 基本操作」を参照してください。

用途

LOB を含む NESTED TABLE を作成します。

使用上の注意

ありません。

構文

次のマニュアルの項を参照してください。

- SQL: 『Oracle8i SQL リファレンス』の第7章「SQL 文」の「CREATE TABLE」

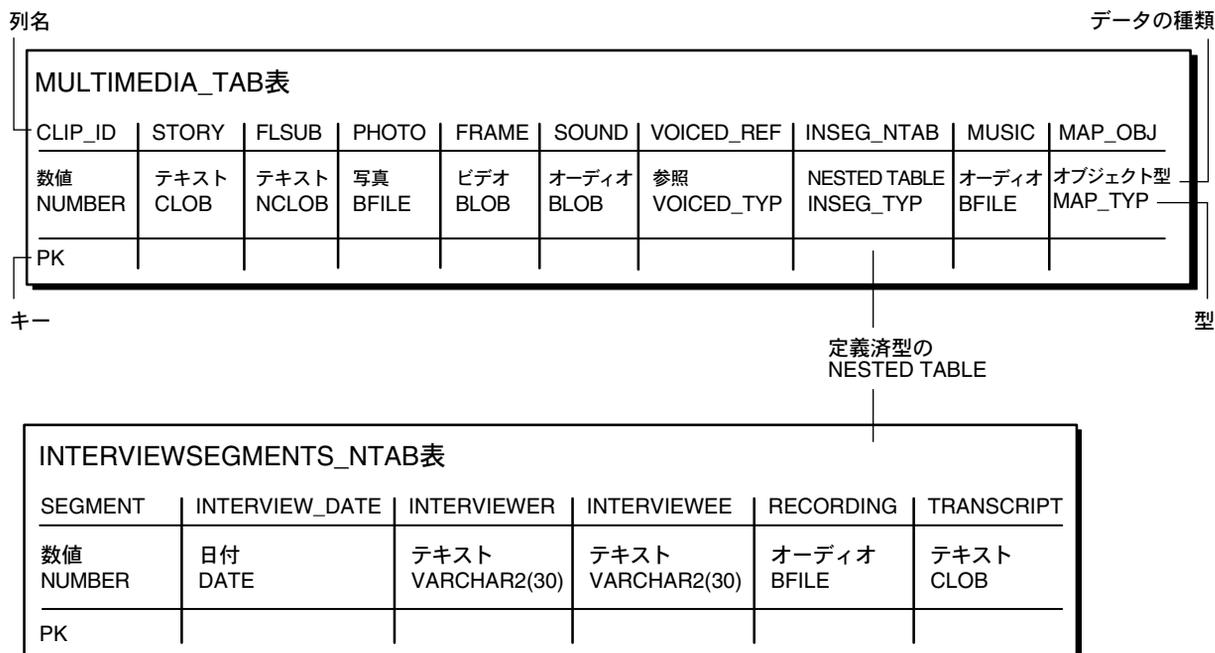
使用例

LOB 属性を含むオブジェクト型を作成してから、そのオブジェクト型に基づいた NESTED TABLE を作成します。この例の Multimedia_tab 表には、InSeg_typ 型を持つ NESTED TABLE である Inseg_ntab が含まれます。この型は、次の 2 種類の LOB データ型を使用しています。

- インタビューの音声録音用の BFILE
- 録音の台本を作成するユーザー用の CLOB

LOB 列を伴う表の作成方法は、前項で説明したとおりです（9-8 ページの「[1 つ以上の LOB 列を含む表の作成](#)」を参照）。ここでは、基礎となるオブジェクト型を作成する構文について説明します。

図 9-10 LOB を含む NESTED TABLE を作成する例としての INTERVIEWSEGMENTS_NTAB



例

例が SQL で示されています。この例は、すべてのプログラム環境に適用されます。

- [SQL: LOB を含む NESTED TABLE の作成](#)

SQL: LOB を含む NESTED TABLE の作成

```
/* 型 InSeg_typ を LOB を含む NESTED TABLE のベース型として作成します。*/  
DROP TYPE InSeg_typ force;  
DROP TYPE InSeg_tab;  
DROP TABLE InSeg_table;  
CREATE TYPE InSeg_typ AS OBJECT (  
    Segment          NUMBER,  
    Interview_Date   DATE,  
    Interviewer      VARCHAR2(30),  
    Interviewee      VARCHAR2(30),  
    Recording        BFILE,  
    Transcript       CLOB  
);
```

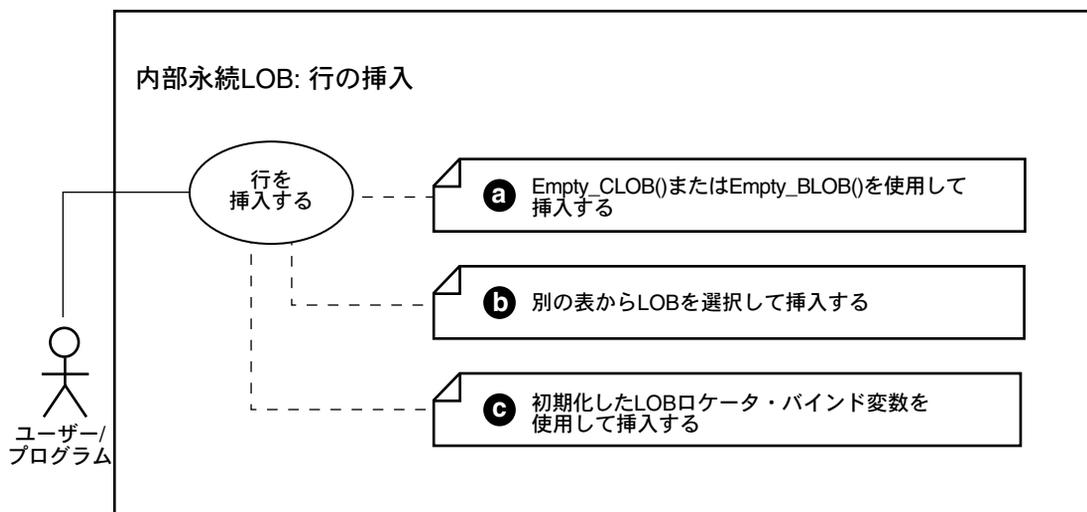
/* 型は作成されましたが、multi_media_tab にその型の NESTED TABLE を埋め込む必要があります。
そのため、次を実行します。*/

```
CREATE TYPE InSeg_tab AS TABLE of Inseg_typ;  
CREATE TABLE InSeg_table (  
    id number,  
    InSeg_ntab Inseg_tab)  
NESTED TABLE InSeg_ntab STORE AS InSeg_nestedtab;
```

NESTED TABLE の実際の埋込みは、その表を含む構造が定義されたときに行われます。この例では、Multimedia_tab が作成されるときに、NESTED TABLE 文によって行われます。

1つ以上のLOB値を行に挿入する3つの方法

図 9-11 LOB 値を行に挿入する3つの方法



参照： 内部永続LOBに関するすべての基本操作については、9-2 ページの「ユースケース・モデル:内部永続LOB – 基本操作」を参照してください。

LOB 値を行に挿入するには、次の3つの方法があります。

- ロケータを初期化することによって、LOB を行に挿入できます。9-23 ページの「EMPTY_CLOB() または EMPTY_BLOB() を使用した LOB 値の挿入」を参照してください。
- 別の表から行を選択することによって、LOB を行に挿入できます。9-27 ページの「別の表からの LOB の選択による行の挿入」を参照してください。
- 最初に初期化した LOB ロケータ・バインド変数を使用してから、LOB を行に挿入できます。9-29 ページの「初期化した LOB ロケータ・バインド変数を使用した行の挿入」を参照してください。

使用上の注意

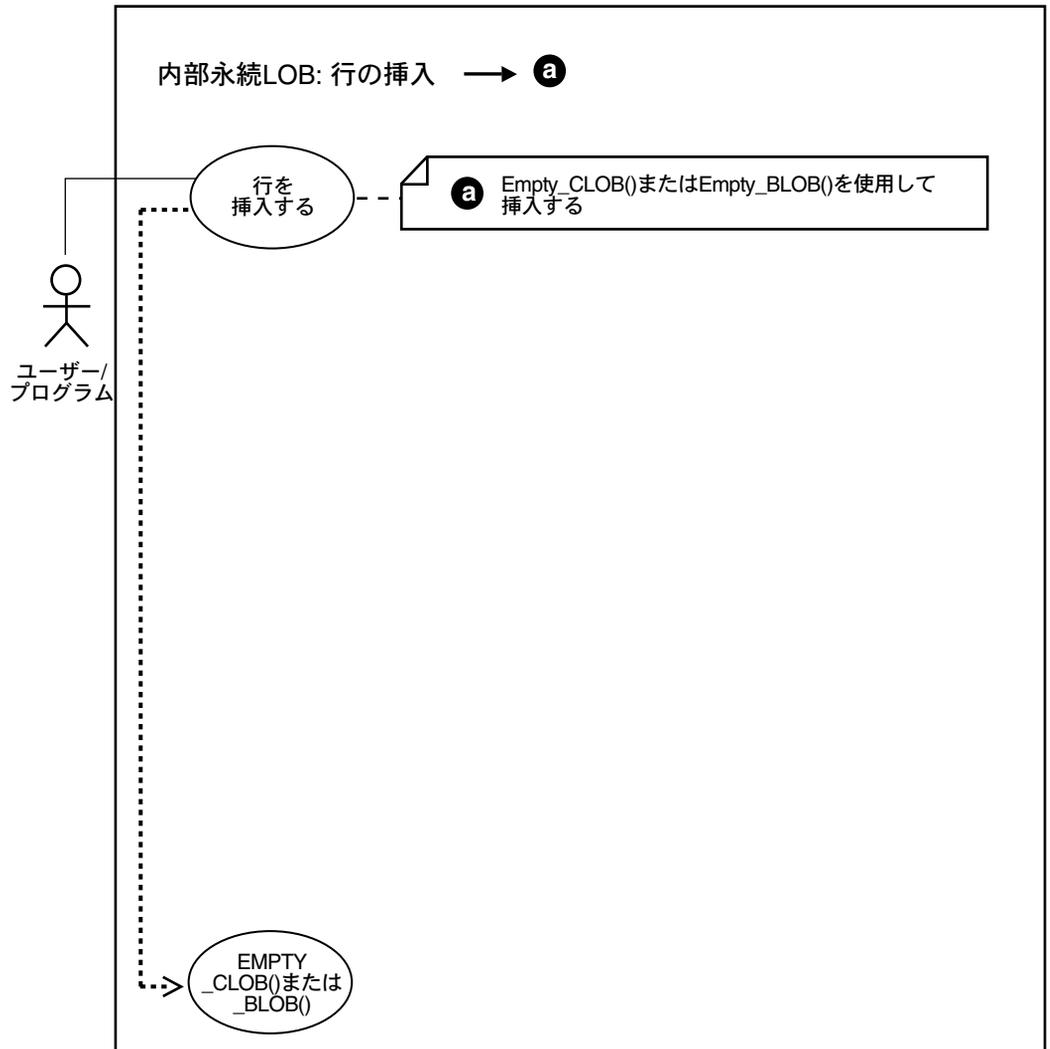
4,000 バイトを超えるバインドの場合

4,000 バイトを超えるバインドを含む場合の、LOB への挿入方法のガイドラインについては、[第 7 章「モデリングおよび設計」](#)の次の項を参照してください。

- 7-14 ページの「LOB の INSERT および UPDATE で現在可能な 4,000 バイトを超えるバインド」
- 7-14 ページの「4,000 バイトを超えるバインド (HEX から RAW または RAW から HEX への変換なし)」
- 7-16 ページの「例: PL/SQL - INSERT および UPDATE での 4,000 バイトを超えるバインドの使用」
- 7-17 ページの「例: PL/SQL - 4,000 バイトを超えるバインド (HEX から RAW / RAW から HEX への変換がサポートされていないため挿入は未サポート)」
- 7-18 ページの「例: PL/SQL - データに SQL 演算子が含まれる場合に 4,000 バイトを超えるバインドの結果を 4,000 バイトに制限」

EMPTY_CLOB() または EMPTY_BLOB() を使用した LOB 値の挿入

図 9-12 ユースケース図: EMPTY_CLOB() または EMPTY_BLOB() を使用した行の挿入



参照： 内部永続 LOB に関するすべての基本操作については、9-2 ページの「ユースケース・モデル:内部永続 LOB – 基本操作」を参照してください。

用途

EMPTY_CLOB() または EMPTY_BLOB() を使用して、LOB 値を挿入します。

使用上の注意

NULL 以外の LOB 列の作成

データを内部 LOB に書き込む前に、LOB 列を NULL 以外にしておきます。つまり、LOB 列には、空または移入された LOB 値を示すロケータを含める必要があります。EMPTY_BLOB() をデフォルトの述語に使用することによって、BLOB 列の値を初期化できます。同様に、EMPTY_CLOB() 関数を使用して、CLOB 列または NCLOB 列の値を初期化できます。

サイズが 4,000 バイト未満の文字または RAW 文字列を持つ LOB 列も初期化できます。次に例を示します。

```
INSERT INTO Multimedia_tab (clip_id, story)
VALUES (1, 'This is a One Line Story');
```

この初期化は CREATE TABLE で実行されます（「1 つ以上の LOB 列を含む表の作成」を参照）。また、この場合のように INSERT を使用しても実行できます。

構文

各プログラム環境で使用可能な機能のリストは、第 3 章「LOB プログラム環境」を参照してください。各プログラム環境における構文については、次のマニュアルの項を参照してください。

- SQL: 『Oracle8i SQL リファレンス』の第 7 章「SQL 文」の「INSERT」
- C (OCI) : 参照マニュアルはありません。
- COBOL (Pro*COBOL) : 参照マニュアルはありません。
- C/C++ (Pro*C/C++) : 参照マニュアルはありません。
- Visual Basic (OO4O) : 参照マニュアルはありません。

- Java (JDBC) : 『Oracle8i JDBC 開発者ガイドおよびリファレンス』の第7章「LOB と BFILE の操作」の「BLOB と CLOB の操作」の「BLOB または CLOB 列の作成と移入」、および『Oracle8i SQLJ 開発者ガイドおよびリファレンス』の第5章「型のサポート」の「Oracle の拡張型」の「BLOB、CLOB および BFILE のサポート」

使用例

参照: マルチメディア・アプリケーションおよび Multimedia_tab 表の詳細は、第8章「サンプル・アプリケーション」を参照してください。

例

次のプログラム環境の例が示されています。

- [SQL: EMPTY_CLOB\(\)/EMPTY_BLOB\(\) を使用した値の挿入](#) (9-25 ページ)
- C (OCI) : 今回のリリースでは例は記載されていません。
- C/C++ (Pro*C/C++) : 今回のリリースでは例は記載されていません。
- COBOL (Pro*COBOL) : 今回のリリースでは例は記載されていません。
- Visual Basic (OO4O) : 今回のリリースでは例は記載されていません。
- [Java \(JDBC\) : EMPTY_CLOB\(\)/EMPTY_BLOB\(\) を使用した値の挿入](#) (9-26 ページ)

SQL: EMPTY_CLOB()/EMPTY_BLOB() を使用した値の挿入

これらの関数は、Oracle8 SQL DML では特別関数として使用できますが、DBMS_LOB パッケージには含まれていません。

```

/* Multimedia_tab 表の新しい行では、
   列 STORY および FLSUB は EMPTY_CLOB() を使用して初期化され、
   列 FRAME および SOUND は EMPTY_BLOB() を使用して初期化され、
   NESTED TABLE の列 TRANSSCRIPT は EMPTY_CLOB() を使用して初期化され、
   列オブジェクトの列 DRAWING は EMPTY_BLOB() を使用して初期化されます。*/
INSERT INTO Multimedia_tab
VALUES (1, EMPTY_CLOB(), EMPTY_CLOB(), NULL, EMPTY_BLOB(), EMPTY_BLOB(),
NULL, InSeg_tab(InSeg_typ(1, NULL, 'Ted Koppell', 'Jimmy Carter', NULL,
EMPTY_CLOB()), NULL, Map_typ('Moon Mountain', 23, 34, 45, 56, EMPTY_BLOB(),
NULL));

/* Voiceover_tab 表の新しい行では、列 SCRIPT は EMPTY_CLOB() を使用して初期化されます。*/
INSERT INTO Voiceover_tab
VALUES ('Abraham Lincoln', EMPTY_CLOB(), 'James Earl Jones', 1, NULL);

```

Java (JDBC) : EMPTY_CLOB()/EMPTY_BLOB() を使用した値の挿入

```
Statement stmt = conn.createStatement() ;
    try {
stmt.execute ("insert into lobtable values (empty_blob());
    }
    catch{ ...}

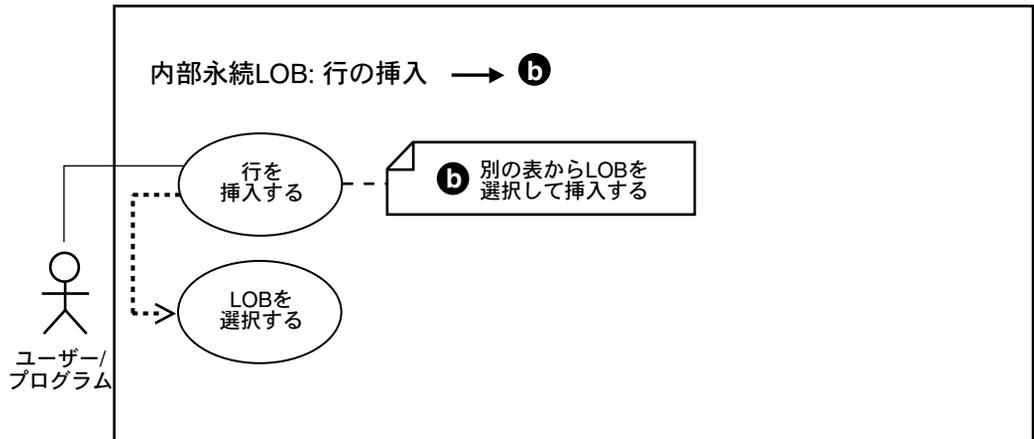
Util.trySQL, Util.doSQL below does the stmt.execute() ;

Util.trySQL (conn, "drop table lobtran_table");
Util.doSQL (conn, "create table lobtran_table (b1 blob, b2 blob, c1 clob,
                c2 clob, f1 bfile, f2 bfile)");

Util.doSQL (conn, "insert into lobtran_table values
                ('010101010101010101010101010101', empty_blob(),
                'onetwothreefour', empty_clob(),
                bfilename ('TEST_DIR', 'tkpjobLOB11.dat'),
                bfilename ('TEST_DIR', 'tkpjobLOB12.dat'))");
```

別の表からの LOB の選択による行の挿入

図 9-13 ユースケース図：別の表からの LOB の選択による行の挿入



参照： 内部永続 LOB に関するすべての基本操作については、9-2 ページの「ユースケース・モデル:内部永続 LOB – 基本操作」を参照してください。

用途

別の表から LOB を選択して LOB を含む列を挿入します。

使用上の注意

注意： BFILE には参照セマンティクスが適用されますが、内部 LOB 型の BLOB、CLOB および NCLOB では、コピー・セマンティクスが使用されます。BLOB、CLOB または NCLOB が、ある行から同じ表または別の表の行にコピーされる場合、LOB ロケータのみがコピーされるのではなく、実際の LOB 値がコピーされます。

たとえば、Voiceover_tab および VoiceoverLib_tab が同じスキーマを持つとすると、文によって、Voiceover_tab 表の中に新しい LOB ロケータが作成され、LOB データが、VoiceoverLib_tab から Voiceover_tab 表に挿入された新しい LOB ロケータが示す位置にコピーされます。

構文

次のマニュアルの項を参照してください。

- SQL: 『Oracle8i SQL リファレンス』の第7章「SQL 文」の「INSERT」

使用例

LOB に関してオブジェクト・リレーショナル技法を使用すると、型に関連する表の共通テンプレートとして定義できるというメリットがあります。たとえば、アーカイブ・データを格納する表と、これらのライブラリを使用する作業表の両方が共通の構造を持つ必要があります。

次のコード例は、ライブラリ表 `VoiceoverLib_tab` が、`Multimedia_tab` 表の `Voiced_ref` 列によって参照される `Voiceover_tab` と同じ型 (`Voiced_typ`) であるということに基づいています。この例では、値をライブラリ表の中に挿入し、次に同じデータを `SELECT` 操作を使用して `Multimedia_tab` に挿入しています。

参照： マルチメディア・アプリケーションおよび `Multimedia_tab` 表の詳細は、[第8章「サンプル・アプリケーション」](#)を参照してください。

例

例が SQL で示されています。この例は、すべてのプログラム環境に適用されます。

- [SQL: 別の表からの LOB の選択による行の挿入](#)

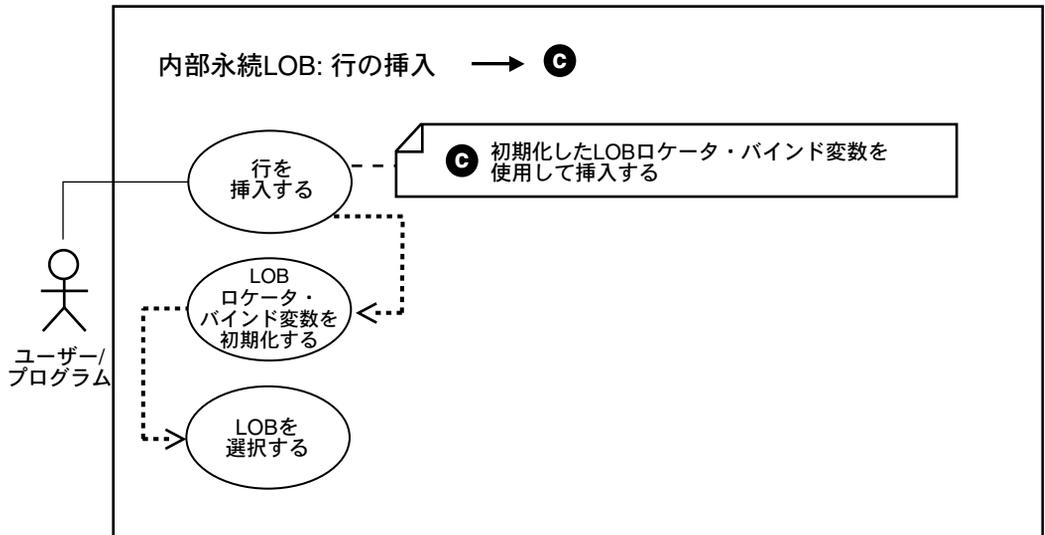
SQL: 別の表からの LOB の選択による行の挿入

```
/* アーカイブ表 VoiceoverLib_tab にレコードを格納します。 */
INSERT INTO VoiceoverLib_tab
  VALUES ('George Washington', EMPTY_CLOB(), 'Robert Redford', 1, NULL);

/* VoiceoverLib_tab から選択して、Voiceover_tab に値を挿入します。 */
INSERT INTO Voiceover_tab
  (SELECT * from VoiceoverLib_tab
   WHERE Take = 1);
```

初期化した LOB ロケータ・バインド変数を使用した行の挿入

図 9-14 ユースケース図：初期化した LOB ロケータ・バインド変数を使用した行の挿入



参照： 内部永続 LOB に関するすべての基本操作については、9-2 ページの「ユースケース・モデル：内部永続 LOB – 基本操作」を参照してください。

用途

初期化した LOB ロケータ・バインド変数を使用して、行を挿入します。

使用上の注意

4,000 バイトを超えるバインドの INSERT および UPDATE への使用に関する使用上の注意および例については、7-14 ページの「INSERT および UPDATE での 4,000 バイトを超えるバインド」を参照してください。

構文

各プログラム環境で使用可能な機能のリストは、第3章「LOB プログラム環境」を参照してください。各プログラム環境における構文については、次のマニュアルの項を参照してください。

- SQL: 『Oracle8i SQL リファレンス』の第7章「SQL 文」の「INSERT」
- C (OCI) : 『Oracle8i コール・インタフェース・プログラマーズ・ガイド』の第15章「OCI リレーショナル関数」の「LOB 関数」
- COBOL (Pro*COBOL) : 『Oracle8i Pro*COBOL プリコンパイラ・プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「INSERT (実行可能埋込み SQL)」
- C/C++ (Pro*C) : 『Oracle8i Pro*C/C++ プリコンパイラ・プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「INSERT (実行可能埋込み SQL)」
- Visual Basic (Oracle Objects for OLE オンライン・ヘルプ) : 「Help」の「Contents」タブから、「OO4O Automation Server」>「Objects」>「Oradynaset」を選択してください。
- Java (JDBC) : 『Oracle8i JDBC 開発者ガイドおよびリファレンス』の第7章「LOB と BFILE の操作」の「BLOB と CLOB の操作」の「BLOB または CLOB 列の作成と移入」、および『Oracle8i SQLJ 開発者ガイドおよびリファレンス』の第5章「型のサポート」の「Oracle の拡張型」の「BLOB、CLOB および BFILE のサポート」

使用例

次の例では、LOB ロケータ・バインド変数を使用して Multimedia_tab の1つの行の中にある Sound データを取得し、それを別の行に挿入します。

例

次のプログラム環境の例が示されています。

- PL/SQL (DBMS_LOB パッケージ) : 初期化した LOB ロケータ・バインド変数を使用した行の挿入 (9-31 ページ)
- C (OCI) : 初期化した LOB ロケータ・バインド変数を使用した行の挿入 (9-31 ページ)
- COBOL (Pro*COBOL) : 初期化した LOB ロケータ・バインド変数を使用した行の挿入 (9-33 ページ)
- C/C++ (Pro*C) : 初期化した LOB ロケータ・バインド変数を使用した行の挿入 (9-34 ページ)
- Visual Basic (OO4O) : 初期化した LOB ロケータ・バインド変数を使用した行の挿入 (9-35 ページ)

- [Java \(JDBC\) : 初期化した LOB ロケータ・バインド変数を使用した行の挿入 \(9-36 ページ\)](#)

PL/SQL (DBMS_LOB パッケージ) : 初期化した LOB ロケータ・バインド変数を使用した行の挿入

```
/* 例のプロシージャ insertUseBindVariable_proc は、DBMS_LOB パッケージの
一部ではないことに注意してください。*/
CREATE OR REPLACE PROCEDURE insertUseBindVariable_proc
  (Rownum IN NUMBER, Blob_loc IN BLOB) IS
BEGIN
  INSERT INTO Multimedia_tab (Clip_ID, Sound) VALUES (Rownum, Blob_loc);
END;

DECLARE
  Blob_loc BLOB;
BEGIN
  /* Clip_ID = 1 の行から LOB を選択します。
  LOB ロケータ・バインド変数を初期化します。*/
  SELECT Sound INTO Blob_loc
  FROM Multimedia_tab
  WHERE Clip_ID = 1;
  /* Clip_ID = 2 の行に挿入します。*/
  insertUseBindVariable_proc (2, Blob_loc);
  COMMIT;
END;
```

C (OCI) : 初期化した LOB ロケータ・バインド変数を使用した行の挿入

```
/* ロケータ変数にロケータを選択します。*/

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

  OCIDefine *defnp1;

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

  /* SQL 文を準備します。*/
```

```

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

/* 選択されている列を定義します。*/
checkerr (errhp, OCIDefineByPos(stmthp, &defnp1, errhp, (ub4) 1,
                                (dvoid *)&Lob_loc, (sb4)0,
                                (ub2)SQLT_BLOB, (dvoid *)0, (ub2 *)0, (ub2 *)0,
                                (ub4)OCI_DEFAULT));

/* SELECT を実行し、1 行フェッチします。*/
checkerr (errhp, OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                                (CONST OCISnapshot*) 0, (OCISnapshot*) 0,
                                (ub4) OCI_DEFAULT));

return (0);

}
/* バインド変数を使用して、表に選択されたロケータを挿入します。
   このファンクションによって、Multimedia_tab からロケータが選択され、
   同じ表の別の行に挿入されます。
*/
void insertUseBindVariable (envhp, errhp, svchp, stmthp)
OCIEnv      *envhp;
OCIError    *errhp;
OCISvcCtx   *svchp;
OCIStmt     *stmthp;
{
    int          clipid;
    OCILobLocator *Lob_loc;
    OCIBind     *bndhp2;
    OCIBind     *bndhp1;

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

/* ロケータ・リソースを割り当てます。*/
(void) OCIDescriptorAlloc((dvoid *) envhp,
                          (dvoid **) &Lob_loc, (ub4)OCI_DTYPE_LOB,
                          (size_t) 0, (dvoid **) 0);

/* マルチメディア表から LOB ロケータを選択します。*/
select_MultimediaLocator(Lob_loc, errhp, stmthp, svchp);

```

```

/* Clip_ID=2 の Multimedia_tab にロケータを挿入します。*/
clipid = 2;

/* SQL 文を準備します。*/
checkerr (errhp, OCIStmtPrepare(stmthp, errhp, insstmt, (ub4)
                                strlen((char *) insstmt),
                                (ub4) OCI_NTV_SYNTAX, (ub4)OCI_DEFAULT));

/* バインド位置をバインドします。*/
checkerr (errhp, OCIBindByPos(stmthp, &bndhp1, errhp, (ub4) 1,
                                (dvoid *) &clipid, (sb4) sizeof(clipid),
                                SQLT_INT, (dvoid *) 0, (ub2 *)0, (ub2 *)0,
                                (ub4) 0, (ub4 *) 0, (ub4) OCI_DEFAULT));
checkerr (errhp, OCIBindByPos(stmthp, &bndhp2, errhp, (ub4) 2,
                                (dvoid *) &Lob_loc, (sb4) 0, SQLT_BLOB,
                                (dvoid *) 0, (ub2 *)0, (ub2 *)0,
                                (ub4) 0, (ub4 *) 0, (ub4) OCI_DEFAULT));

/* SQL 文を実行します。*/
checkerr (errhp, OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                                (CONST OCISnapshot*) 0, (OCISnapshot*) 0,
                                (ub4) OCI_DEFAULT));

/* LOB リソースを解放します。*/
OCIDescriptorFree((dvoid *) Lob_loc, (ub4) OCI_DTYPE_LOB);
}

```

COBOL (Pro*COBOL) : 初期化した LOB ロケータ・バインド変数を使用した行の挿入

```

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

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

```

```
* BLOB ロケータを初期化します。
EXEC SQL ALLOCATE :BLOB1 END-EXEC.

* LOB を移入します。
EXEC SQL WHENEVER NOT FOUND GOTO END-OF-BLOB END-EXEC.
EXEC SQL
SELECT SOUND INTO :BLOB1
FROM MULTIMEDIA_TAB WHERE CLIP_ID = 1 END-EXEC.

* CLIP_ID=2 の値を挿入します。
EXEC SQL
INSERT INTO MULTIMEDIA_TAB (CLIP_ID, SOUND)
VALUES (2, :BLOB1) END-EXEC.

* ロケータが保持しているリソースを解放します。
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) : 初期化した LOB ロケータ・バインド変数を使用した行の挿入

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

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

```

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

    /* BLOB ロケータを初期化します。*/
    EXEC SQL ALLOCATE :Lob_loc;
    /* Clip_ID = 1 の行から LOB を選択します。*/
    EXEC SQL SELECT Sound INTO :Lob_loc
        FROM Multimedia_tab WHERE Clip_ID = 1;
    /* Clip_ID = 2 の行に挿入します。*/
    insertUseBindVariable_proc(2, Lob_loc);
    /* ロケータが保持しているリソースを解放します。*/
    EXEC SQL FREE :Lob_loc;
}

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

```

Visual Basic (0040) : 初期化した LOB ロケータ・バインド変数を使用した行の挿入

```

Dim OraDyn as OraDynaset, OraSound1 as OraBLOB, OraSoundClone as OraBLOB
Set OraDyn = OraDb.CreateDynaset(
    "SELECT * FROM Multimedia_tab ORDER BY clip_id", ORADYN_DEFAULT)
Set OraSound1 = OraDyn.Fields("Sound").Value
' 将来の参照用にクローンを作成します。
Set OraSoundClone = OraSound1

' 次の行に進みます。
OraDyn.MoveNext

```

、カレント行を更新し、LOB を OraSoundClone に設定します。

```
OraDyn.Edit  
Set OraSound1 = OraSoundClone  
OraDyn.Update
```

Java (JDBC) : 初期化した LOB ロケータ・バインド変数を使用した行の挿入

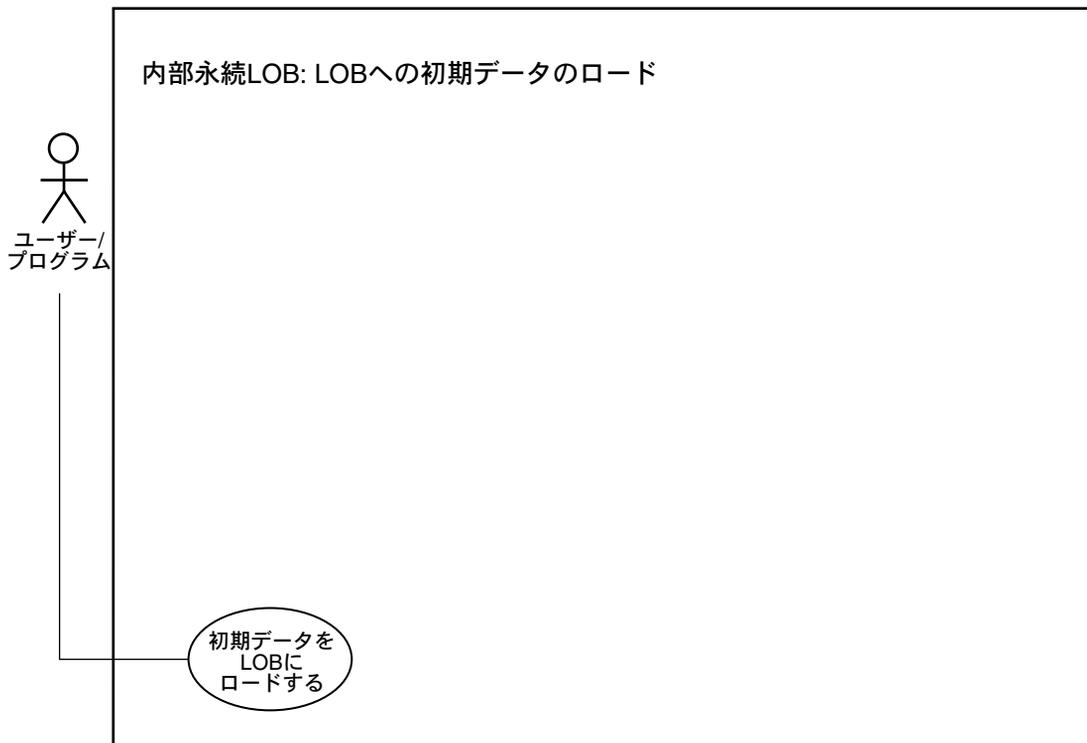
```
// コア JDBC クラス :  
import java.sql.DriverManager;  
import java.sql.Connection;  
import java.sql.Statement;  
import java.sql.PreparedStatement;  
import java.sql.ResultSet;  
import java.sql.SQLException;  
  
// Oracle 固有の JDBC クラス :  
import oracle.sql.*;  
import oracle.jdbc.driver.*;  
  
public class Ex2_31  
{  
    public static void main (String args [])  
        throws Exception  
    {  
        // Oracle JDBC Driver をロードします。  
        DriverManager.registerDriver (new oracle.jdbc.driver.OracleDriver ());  
        // データベースに接続します。  
        Connection conn =  
            DriverManager.getConnection ("jdbc:oracle:oci8:@", "samp", "samp");  
  
        // 自動コミットがオフの場合、高速になります。  
        conn.setAutoCommit (false);  
  
        // 文を作成します。  
        Statement stmt = conn.createStatement ();  
        try  
        {  
            ResultSet rset = stmt.executeQuery (  
                "SELECT sound FROM multimedia_tab WHERE clip_id = 1");  
            if (rset.next())  
            {  

```

```
// ResultSet からロケータを取り出します。
BLOB sound_blob = ((OracleResultSet)rset).getBLOB (1);
OraclePreparedStatement ops =
(OraclePreparedStatement) conn.prepareStatement(
    "INSERT INTO multimedia_tab (clip_id, sound) VALUES (2, ?)");
ops.setBlob(1, sound_blob);
ops.execute();
conn.commit();
conn.close();
    }
}
catch (SQLException e)
{
    e.printStackTrace();
}
}
```

内部 LOB (BLOB、CLOB、NCLOB) へのデータのロード

図 9-15 ユースケース図：内部 LOB への初期データのロード



参照： 内部永続 LOB に関するすべての基本操作については、9-2 ページの「ユースケース・モデル:内部永続 LOB – 基本操作」を参照してください。

用途

データを内部 LOB にロードします。

使用上の注意および例

SQL*Loader を使用したデータの内部 LOB へのロードに関する詳細およびヒントは、4-5 ページの「LOB ロード時の SQL*Loader の使用」を参照してください。

- 行内 LOB データのロード
 - 既定サイズ・フィールド内の行内 LOB データのロード
 - デリミタ付きフィールド内の行内 LOB データのロード
 - Length-Value Pair フィールド内の行内 LOB データのロード
- 行外 LOB データのロード
 - 1 ファイルにつき 1 つの LOB のロード
 - 既定サイズ・フィールド内の行外 LOB データのロード
 - デリミタ付きフィールド内の行外 LOB データのロード
 - Length-Value Pair フィールド内の行外 LOB データのロード

参照: 『Oracle8i ユーティリティ・ガイド』の第 II 部「SQL*Loader」を参照してください。

構文

前述の「使用上の注意および例」を参照してください。

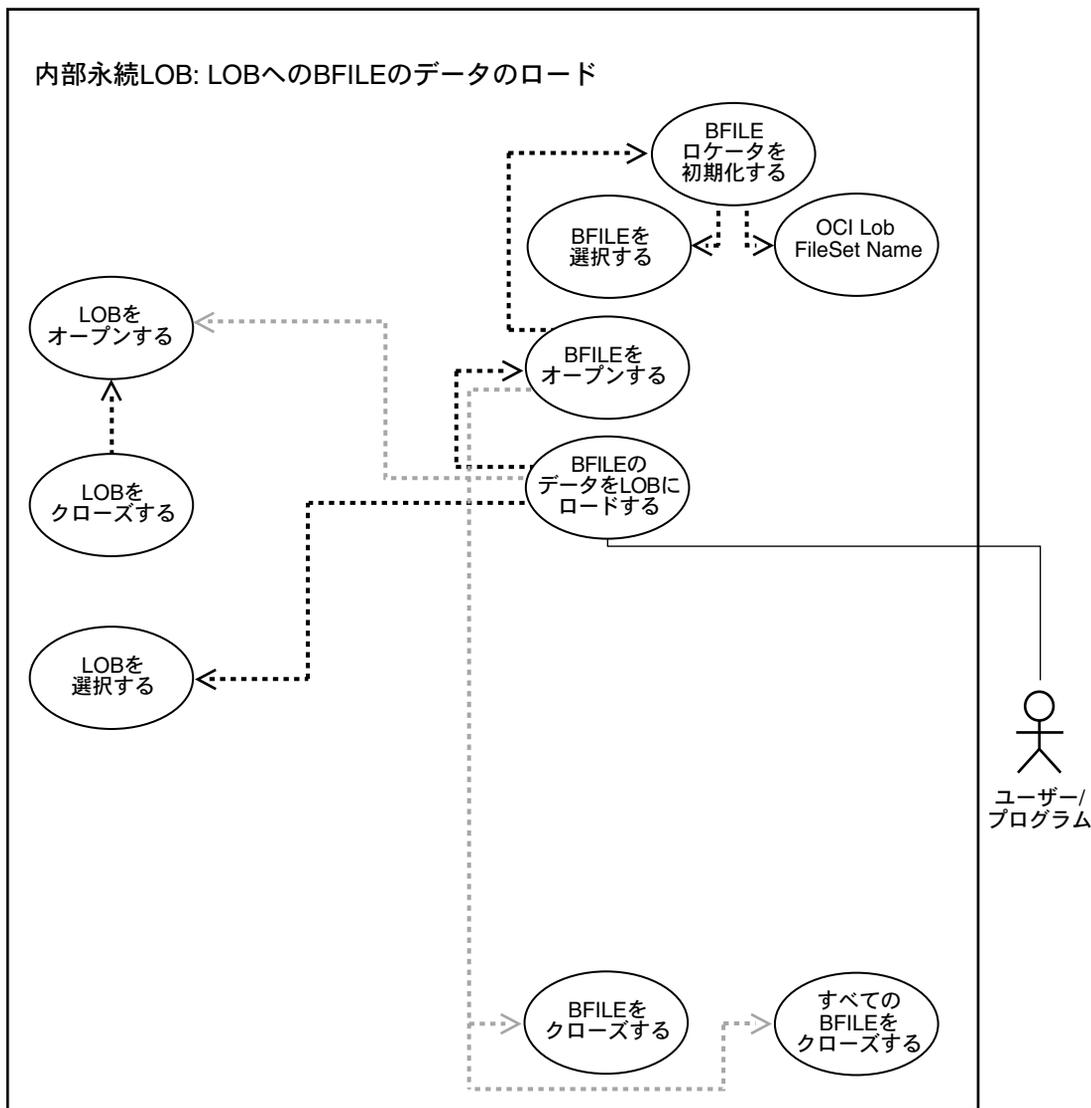
使用例

LOB は非常に大きくなる場合があるため、SQL*Loader が LOB データをメイン・データ・ファイル（データの残りの部分については行内）か 1 つ以上の 2 次データ・ファイルのどちらからでもロードできるようにしておく必要があります。

LOB データをメイン・データ・ファイルからロードするには、通常の SQL*Loader フォーマットを使用します。LOB データ・インスタンスは、サイズ・フィールド、デリミタ付きフィールドまたは Length-Value Pair フィールドで事前に決定しておくことができます。

LOB への BFILE データのロード

図 9-16 ユースケース図: LOB への BFILE データのロード



参照： 内部永続 LOB に関するすべての基本操作については、9-2 ページの「ユースケース・モデル:内部永続 LOB – 基本操作」を参照してください。

用途

BFILE のデータを LOB にロードします。

使用上の注意

BFILE データ上ではバイナリ・データからキャラクタ・セットへの変換が必要

OCI または OCI 機能にアクセスするプログラム環境を使用する場合、キャラクタ・セットの変換は 1 つのキャラクタ・セットから別のキャラクタ・セットに変換するとき、暗黙的に実行されます。ただし、バイナリ・データからキャラクタ・セットへの変換は実行されません。

LOADFROMFILE プロシージャを使用して CLOB または NCLOB に移入する場合は、BFILE のバイナリ・データを LOB に移入することになります。この場合、LOADFROMFILE を実行する前に、キャラクタ・セットの変換を BFILE データ上で実行しておく必要があります。

サイズを BFILE のサイズより小さく指定

- **DBMS_LOB.LOADFROMFILE:** サイズを BFILE のサイズより小さく指定する必要があります。
- **OCILobLoadFromFile:** サイズを BFILE のサイズより小さく指定する必要があります。

構文

各プログラム環境で使用可能な機能のリストは、第 3 章「LOB プログラム環境」を参照してください。各プログラム環境における構文については、次のマニュアルの項を参照してください。

- **PL/SQL (DBMS_LOB パッケージ) :** 『Oracle8i PL/SQL パッケージ・プロシージャ・リファレンス』の第 18 章「DBMS_LOB」の「サブプログラムの要約」の LOADFROMFILE プロシージャ
- **C (OCI) :** 『Oracle8i コール・インタフェース・プログラマーズ・ガイド』の第 15 章「OCI リレーショナル関数」の「LOB 関数」の OCILobLoadFromFile()

- COBOL (Pro*COBOL) : 『Oracle8i Pro*COBOL プリコンパイラ・プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB LOAD (実行可能埋込み SQL 拡張要素)」、「LOB OPEN (実行可能埋込み SQL 拡張要素)」、「LOB CLOSE (実行可能埋込み SQL 拡張要素)」
- C/C++ (Pro*C) : 『Oracle8i Pro*C/C++ プリコンパイラ・プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB LOAD (実行可能埋込み SQL 拡張要素)」
- Visual Basic (Oracle Objects for OLE オンラインオンライン・ヘルプ) : 「Help」の「Contents」タブから、「OO4O Automation Server」>「OBJECTS」>「OraBfile」>「METHODS」>「CopyFromBFILE」、および「OO4O Automation Server」>「OBJECTS」>「OraDynaset」、「OraDatabase」、「OraConnection」を選択してください。
- Java (JDBC) : 『Oracle8i JDBC 開発者ガイドおよびリファレンス』の第7章「LOB と BFILE の操作」の「BLOB と CLOB の操作」の「BLOB または CLOB 列の作成と移入」、および『Oracle8i SQLJ 開発者ガイドおよびリファレンス』の第5章「型のサポート」の「Oracle の拡張型」の「BLOB、CLOB および BFILE のサポート」

使用例

次の例は、オペレーティング・システムのソース・ファイル (Washington_audio) が LOB データを含み、ターゲットの LOB (Music) にロードされると想定しています。また、ディレクトリ・オブジェクト AUDIO_DIR がすでに存在し、ソース・ファイルの位置にマップされていると想定しています。

例

次のプログラム環境の例が示されています。

- PL/SQL (DBMS_LOB パッケージ) : LOB への BFILE データのロード (9-42 ページ)
- C (OCI) : LOB への BFILE データのロード (9-43 ページ)
- COBOL (Pro*COBOL) : LOB への BFILE データのロード (9-45 ページ)
- C/C++ (Pro*C) : LOB への BFILE データのロード (9-46 ページ)
- Visual Basic (OO4O) : LOB への BFILE データのロード (9-47 ページ)
- Java (JDBC) : LOB への BFILE データのロード (9-48 ページ)

PL/SQL (DBMS_LOB パッケージ) : LOB への BFILE データのロード

```
/* 例のプロシージャ loadLOBFromBFILE_proc は、DBMS_LOB パッケージの  
一部ではないことに注意してください。*/
```

```
CREATE OR REPLACE PROCEDURE loadLOBFromBFILE_proc IS
```

```

Dest_loc      BLOB;
Src_loc       BFILE := BFILENAME('FRAME_DIR', 'Washington_frame');
Amount        INTEGER := 4000;
BEGIN
  SELECT Frame INTO Dest_loc FROM Multimedia_tab
     WHERE Clip_ID = 3 FOR UPDATE;
  /* ソース BFILE のオープンは必須です。 */
  DBMS_LOB.OPEN(Src_loc, DBMS_LOB.LOB_READONLY);
  /* LOB のオープンはオプションです。 */
  DBMS_LOB.OPEN(Dest_loc, DBMS_LOB.LOB_READWRITE);
  DBMS_LOB.LOADFROMFILE(Dest_loc, Src_loc, Amount);
  /* LOB をオープンしている場合、その LOB をクローズする必要があります。 */
  DBMS_LOB.CLOSE(Dest_loc);
  DBMS_LOB.CLOSE(Src_loc);
  COMMIT;
END;

```

C (OCI) : LOB への BFILE データのロード

次の例は、Multimedia_tab から BLOB を選択し、それに BFILE のデータをロードする方法を示します。

```

sb4 select_lock_frame_locator_3(Lob_loc, errhp, svchp, stmthp)
OCILobLocator *Lob_loc;
OCIError      *errhp;
OCISvcCtx     *svchp;
OCISstmt      *stmthp;
{
  text *sqlstmt =
    (text *) "SELECT Frame FROM Multimedia_tab WHERE Clip_ID=3 FOR UPDATE";
  OCIDefine *defnpl;
  checkerr (errhp, OCISstmtPrepare(stmthp, errhp, sqlstmt,
                                   (ub4)strlen((char *)sqlstmt),
                                   (ub4) OCI_NTV_SYNTAX, (ub4) OCI_DEFAULT));
  checkerr (errhp, OCIDefineByPos(stmthp, &defnpl, errhp, (ub4) 1,
                                   (dvoid *)&Lob_loc, (sb4) 0,
                                   (ub2) SQLT_BLOB, (dvoid *) 0,
                                   (ub2 *) 0, (ub2 *) 0, (ub4) OCI_DEFAULT));
  /* SELECT を実行し、1 行フェッチします。 */
  checkerr(errhp, OCISstmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                                   (CONST OCISnapshot*) 0, (OCISnapshot*) 0,
                                   (ub4) OCI_DEFAULT));

  return 0;
}

```

```
void LoadLobDataFromBFile(envhp, errhp, svchp, stmthp)
OCIEnv    *envhp;
OCIError  *errhp;
OCISvcCtx *svchp;
OCIStmt   *stmthp;
{
    OCILobLocator *bfile;
    OCILobLocator *blob;
    ub4          amount= 4000;

    /* ソース (BFILE) および宛先 (BLOB) ロケータの記述子を割り当てます。*/
    OCIDescriptorAlloc((dvoid *)envhp, (dvoid **)&bfile,
                      (ub4)OCI_DTYPE_FILE, (size_t)0, (dvoid **)0);
    OCIDescriptorAlloc((dvoid *)envhp, (dvoid **)&blob,
                      (ub4)OCI_DTYPE_LOB, (size_t)0, (dvoid **)0);

    /* 更新用のフレーム・ロケータを選択します。*/
    printf (" select the frame locator...\n");
    select_lock_frame_locator_3(blob, errhp, svchp, stmthp);

    /* フレーム・ファイルのディレクトリのエイリアスおよびファイル名を設定します。*/
    printf (" set the file name in bfile\n");
    checkerr (errhp, OCILobFileSetName(envhp, errhp, &bfile, (text*)"FRAME_DIR",
                                       (ub2)strlen("FRAME_DIR"),
                                       (text*)"Washington_frame",
                                       (ub2)strlen("Washington_frame")));

    printf (" open the bfile\n");
    /* BFILE ロケータのオープンは必須です。*/
    checkerr (errhp, (OCILobOpen(svchp, errhp, bfile, OCI_LOB_READONLY)));

    printf(" open the lob\n");
    /* BLOB ロケータのオープンはオプションです。*/
    checkerr (errhp, (OCILobOpen(svchp, errhp, blob, OCI_LOB_READWRITE)));

    /* オーディオ・ファイル (BFILE) から BLOB ヘデータをロードします。*/
    printf (" load the LOB from File\n");
    checkerr (errhp, OCILobLoadFromFile(svchp, errhp, blob, bfile, (ub4)amount,
                                       (ub4)1, (ub4)1));

    /* LOB をオープンしている場合、その LOB をクローズする必要があります。*/
    checkerr (errhp, OCILobClose(svchp, errhp, bfile));
    checkerr (errhp, OCILobClose(svchp, errhp, blob));

    /* ロケータが保持しているリソースを解放します。*/
}
```

```

(void) OCIDescriptorFree((dvoid *) bfile, (ub4) OCI_DTYPE_FILE);
(void) OCIDescriptorFree((dvoid *) blob, (ub4) OCI_DTYPE_LOB);

return;
}

```

COBOL (Pro*COBOL) : LOB への BFILE データのロード

```

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.
* ロードする量を宣言します。ここでの値は任意に選択され
* ています。
01 LOB-AMT       PIC S9(9) COMP VALUE 10.
01 USERID        PIC X(11) VALUES "SAMP/SAMP".
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.

* BFILE ロケータの割当ておよび初期化を行います。
EXEC SQL ALLOCATE :BFILE1 END-EXEC.

* ディレクトリおよびファイル情報をセットアップします。
MOVE "AUDIO_DIR" TO DIR-ALIAS-ARR.
MOVE 9 TO DIR-ALIAS-LEN.
MOVE "washington_audio" TO FNAME-ARR.
MOVE 16 TO FNAME-LEN.

EXEC SQL
LOB FILE SET :BFILE1 DIRECTORY = :DIR-ALIAS,FILENAME = :FNAME
END-EXEC.

* 宛先 BLOB の割当ておよび初期化を行います。
EXEC SQL ALLOCATE :DEST END-EXEC.

```

```
EXEC SQL WHENEVER NOT FOUND GOTO END-OF-BLOB END-EXEC.
EXEC SQL SELECT SOUND INTO :DEST
      FROM MULTIMEDIA_TAB WHERE CLIP_ID = 3 FOR UPDATE END-EXEC.

* READ用にソース BFILE をオープンします。
EXEC SQL LOB OPEN :BFILE1 READ ONLY END-EXEC.

* READ/WRITE用に宛先 BLOB をオープンします。
EXEC SQL LOB OPEN :DEST READ WRITE END-EXEC.

* ソース BFILE から宛先 BLOB をロードします。
EXEC SQL LOB LOAD :LOB-AMT FROM FILE :BFILE1 INTO :DEST END-EXEC.

* ソースおよび宛先 LOB をクローズします。
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.
```

C/C++ (Pro*C) : LOB への BFILE データのロード

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

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

```

void loadLOBFromBFILE_proc()
{
    OCIBlobLocator *Dest_loc;
    OCIBFileLocator *Src_loc;
    char *Dir = "FRAME_DIR", *Name = "Washington_frame";
    int Amount = 4000;

    EXEC SQL WHENEVER SQLERROR DO Sample_Error();
    /* BFILE ロケータを初期化します。*/
    EXEC SQL ALLOCATE :Src_loc;
    EXEC SQL LOB FILE SET :Src_loc DIRECTORY = :Dir, FILENAME = :Name;
    /* BLOB ロケータを初期化します。*/
    EXEC SQL ALLOCATE :Dest_loc;
    EXEC SQL SELECT frame INTO :Dest_loc FROM Multimedia_tab
        WHERE Clip_ID = 3 FOR UPDATE;
    /* BFILE のオープンは必須です。*/
    EXEC SQL LOB OPEN :Src_loc READ ONLY;
    /* BLOB のオープンはオプションです。*/
    EXEC SQL LOB OPEN :Dest_loc READ WRITE;
    EXEC SQL LOB LOAD :Amount FROM FILE :Src_loc INTO :Dest_loc;
    /* LOB および BFILE がオープンされている場合、それらをクローズする必要があります。*/
    EXEC SQL LOB CLOSE :Dest_loc;
    EXEC SQL LOB CLOSE :Src_loc;
    /* ロケータが保持しているリソースを解放します。*/
    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 (0040) : LOB への BFILE データのロード

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

```

OraConnection.BeginTrans
Set OraDyn = OraDb.CreateDynaset(
    "SELECT * FROM Multimedia_tab ORDER BY clip_id", ORADYN_DEFAULT)
Set OraSound1 = OraDyn.Fields("Sound").Value

```

```
OraDb.Parameters.Add "id", 1,ORAPARAM_INPUT
OraDb.Parameters.Add "mybfile", Null,ORAPARAM_OUTPUT
OraDb.Parameters("mybfile").serverType = ORATYPE_BFILE

OraDb.ExecuteNonQuery ("begin GetBFile(:id, :mybfile); end;")

Set OraMyBFile = OraDb.Parameters("mybfile").Value
' 次の行に進みます。
OraDyn.MoveNext

OraDyn.Edit
' BFILE のデータで OraSound1 データを更新します。
OraSound1.CopyFromBFile OraMyBFile
OraDyn.Update

OraConnection.CommitTrans
```

Java (JDBC) : LOB への BFILE データのロード

```
// Java IO クラス :
import java.io.InputStream;
import java.io.OutputStream;

// コア JDBC クラス :
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 固有の JDBC クラス :
import oracle.sql.*;
import oracle.jdbc.driver.*;

public class Ex2_45
{
    public static void main (String args [])
        throws Exception
    {
        // Oracle JDBC Driver をロードします。
        DriverManager.registerDriver (new oracle.jdbc.driver.OracleDriver ());
        // データベースに接続します。
        Connection conn =
```

```
    DriverManager.getConnection ("jdbc:oracle:oci8:@", "samp", "samp");
    conn.setAutoCommit (false);

    // 文を作成します。
    Statement stmt = conn.createStatement ();

    try
    {
        BFILE src_lob = null;
        BLOB dest_lob = null;
        InputStream in = null;
        OutputStream out = null;
        byte buf[] = new byte[1000];
        ResultSet rset = null;

        rset = stmt.executeQuery (
            "SELECT BFILENAME('AUDIO_DIR', 'Washington_audio') FROM DUAL");
        if (rset.next())
        {
            src_lob = ((OracleResultSet)rset).getBFILE (1);
            src_lob.openFile();
            in = src_lob.getBinaryStream();
        }

        rset = stmt.executeQuery (
            "SELECT sound FROM multimedia_tab WHERE clip_id = 99 FOR UPDATE");
        if (rset.next())
        {
            dest_lob = ((OracleResultSet)rset).getBLOB (1);

            // dest_lob のアウトプット・ストリームをフェッチします。
            out = dest_lob.getBinaryOutputStream();
        }

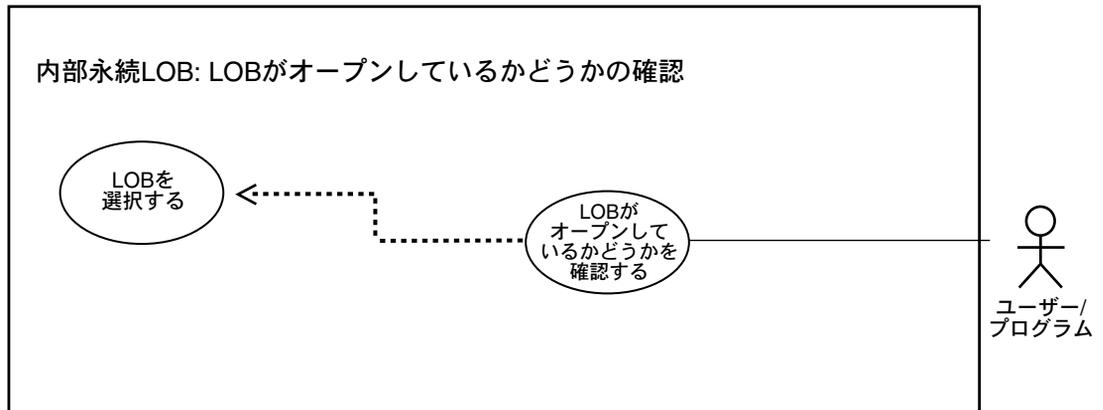
        int length = 0;
        int pos = 0;
        while ((in != null) && (out != null) && ((length = in.read(buf)) != -1))
        {
            System.out.println(
                "Pos = " + Integer.toString(pos) + ". Length = " +
                Integer.toString(length));
            pos += length;
            out.write(buf, pos, length);
        }
    }
}
```

```
// すべてのストリームおよびファイル・ハンドルをクローズします。
in.close();
out.flush();
out.close();
src_lob.closeFile();

// トランザクションをコミットします。
conn.commit();
conn.close();
}
catch (SQLException e)
{
    e.printStackTrace();
}
}
```

LOB がオープンしているかどうかの確認

図 9-17 ユースケース図：LOB がオープンしているかどうかの確認



用途

LOB がオープンしているかどうかを確認します。

使用上の注意

ありません。

構文

各プログラム環境で使用可能な機能のリストは、第3章「LOB プログラム環境」を参照してください。各プログラム環境における構文については、次のマニュアルの項を参照してください。

- PL/SQL (DBM_LOB パッケージ)：『Oracle8i PL/SQL パッケージ・プロシージャ・リファレンス』の第18章「DBMS_LOB」の「サブプログラムの要約」のOPEN プロシージャ、ISOPEN ファンクション
- C (OCI)：『Oracle8i コール・インタフェース・プログラマーズ・ガイド』の第15章「OCI リレーショナル関数」の「LOB 関数」のOCISeelfLobIsOpen()
- COBOL (Pro*COBOL)：『Oracle8i Pro*COBOL プリコンパイラ・プログラマーズ・ガイド』の付録F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB DESCRIBE (実行可能埋込み SQL 拡張要素)」

- C/C++ (Pro*C) : 『Oracle8i Pro*C/C++ プリコンパイラ・プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB DESCRIBE (実行可能埋込み SQL 拡張要素)」
- Visual Basic (OO4O) : 参照マニュアルはありません。
- Java (JDBC) : 『Oracle8i JDBC 開発者ガイドおよびリファレンス』の第7章「LOB と BFILE の操作」の「BLOB と CLOB の操作」の「BLOB または CLOB 列の作成と移入」、および『Oracle8i SQLJ 開発者ガイドおよびリファレンス』の第5章「型のサポート」の「Oracle の拡張型」の「BLOB、CLOB および BFILE のサポート」

使用例

次の例では、ビデオ・フレーム (Frame) をオープンし、LOB がオープンしているかどうかを確認します。

例

次のプログラム環境の例が示されています。

- [PL/SQL \(DBMS_LOB パッケージ\) : LOB がオープンしているかどうかの確認 \(9-52 ページ\)](#)
- [C \(OCI\) : LOB がオープンしているかどうかの確認 \(9-53 ページ\)](#)
- [COBOL \(Pro*COBOL\) : LOB がオープンしているかどうかの確認 \(9-54 ページ\)](#)
- [C/C++ \(Pro*C\) : LOB がオープンしているかどうかの確認 \(9-55 ページ\)](#)
- Visual Basic (OO4O) : 今回のリリースでは例は記載されていません。
- [Java \(JDBC\) : LOB がオープンしているかどうかの確認 \(9-56 ページ\)](#)

PL/SQL (DBMS_LOB パッケージ) : LOB がオープンしているかどうかの確認

/* 例のプロシージャ lobIsOpen_proc は、DBMS_LOB パッケージの一部ではないことに注意してください。*/

```
CREATE OR REPLACE PROCEDURE lobIsOpen_proc IS
  Lob_loc      BLOB;
  Retval       INTEGER;
BEGIN
  SELECT Frame INTO Lob_loc FROM Multimedia_tab where Clip_ID = 1;

  /* LOB のオープンはオプションです。*/
  DBMS_LOB.OPEN (Lob_loc , DBMS_LOB.LOB_READONLY);

  /* LOB がオープンしているかどうかを確認します。*/
  Retval := DBMS_LOB.ISOPEN(Lob_loc);
```

```

/* Retval の値は 1 で、LOB がオープンしていることを意味します。*/
END;

```

C (OCI) : LOB がオープンしているかどうかの確認

```

/* ロケータ変数にロケータを選択します。*/
sb4 select_frame_locator(Lob_loc, errhp, svchp, stmthp)
OCILobLocator *Lob_loc;
OCIError      *errhp;
OCISvcCtx     *svchp;
OCIStmt       *stmthp;
{
    text      *sqlstmt =
        (text *) "SELECT Frame FROM Multimedia_tab WHERE Clip_ID=1";
    OCIDefine *defnp1;
    checkerr (errhp, OCIStmtPrepare(stmthp, errhp, sqlstmt,
                                     (ub4)strlen((char *)sqlstmt),
                                     (ub4)OCI_NTV_SYNTAX, (ub4)OCI_DEFAULT));
    checkerr (errhp, OCIDefineByPos(stmthp, &defnp1, errhp, (ub4) 1,
                                     (dvoid *)&Lob_loc, (sb4) 0,
                                     (ub2) SQLT_BLOB, (dvoid *) 0,
                                     (ub2 *) 0, (ub2 *) 0, (ub4) OCI_DEFAULT));
    /* SELECT を実行し、1 行フェッチします。*/
    checkerr (errhp, OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                                     (CONST OCISnapshot*) 0, (OCISnapshot*) 0,
                                     (ub4) OCI_DEFAULT));

    return (0);
}

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

    /* ロケータ・リソースを割り当てます。*/
    (void) OCIDescriptorAlloc((dvoid *)envhp, (dvoid **)&Lob_loc,
                              (ub4)OCI_DTYPE_LOB, (size_t)0, (dvoid **)0);
    /* ロケータを選択します。*/
    (void)select_frame_locator(Lob_loc, errhp, svchp, stmthp);
}

```

```
/* LOB がオープンしているかどうかを確認します。*/
checkerr (errhp, OCILOBIsOpen(svchp, errhp, Lob_loc, &isOpen));

if (isOpen)
{
    printf(" Lob is Open\n");
    /* LOB がオープンされている場合の処理が行われます。*/
}
else
{
    printf(" Lob is not Open\n");
    /* LOB がオープンされていない場合の処理が行われます。*/
}
/* ロケータが保持しているリソースを解放します。*/
(void) OCIDescriptorFree((dvoid *) Lob_loc, (ub4) OCI_DTYPE_LOB);

return;
}
```

COBOL (Pro*COBOL) : LOB がオープンしているかどうかの確認

```
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 "SAMP/SAMP".
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.
```

```

* ターゲット BLOB の割当ておよび初期化を行います。
EXEC SQL ALLOCATE :BLOB1 END-EXEC.
EXEC SQL WHENEVER NOT FOUND GOTO END-OF-BLOB END-EXEC.
EXEC SQL SELECT FRAME INTO :BLOB1
FROM MULTIMEDIA_TAB WHERE CLIP_ID = 1 END-EXEC.

* LOB がオープンしているかどうかを確認します。
EXEC SQL
LOB DESCRIBE :BLOB1 GET ISOPEN INTO :ISOPN END-EXEC.

IF ISOPN = 1
*   < LOB OPEN の場合の処理が行われます >
    DISPLAY "The LOB is open"
ELSE
*   < LOB NOT OPEN の場合の処理が行われます >
    DISPLAY "The LOB is not open"
END-IF.

* BLOB が使用しているリソースを解放します。
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) : LOB がオープンしているかどうかの確認

```

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

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

```

```
    exit(1);
}

void seeIfLOBIsOpen()
{
    OCIBlobLocator *Lob_loc;
    int isOpen = 1;

    EXEC SQL WHENEVER SQLERROR DO Sample_Error();
    EXEC SQL ALLOCATE :Lob_loc;
    EXEC SQL SELECT Frame INTO :Lob_loc
        FROM Multimedia_tab WHERE Clip_ID = 1;
    /* LOB がオープンしているかどうかを確認します。*/
    EXEC SQL LOB DESCRIBE :Lob_loc GET ISOPEN INTO :isOpen;
    if (isOpen)
        printf("LOB is open\n");
    else
        printf("LOB is not open\n");
    /* この例ではLOB がオープンしていないことに注意してください。*/
    EXEC SQL FREE :Lob_loc;
}

void main()
{
    char *samp = "samp/samp";
    EXEC SQL CONNECT :samp;
    seeIfLOBIsOpen();
    EXEC SQL ROLLBACK WORK RELEASE;
}
```

Java (JDBC) : LOB がオープンしているかどうかの確認

```
// コア JDBC クラス :
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 固有の JDBC クラス :
import oracle.sql.*;
import oracle.jdbc.driver.*;

public class Ex2_48
{
    public Ex2_48 ()
    {
    }

    public static void main (String args [])
        throws Exception
    {
        // Oracle JDBC Driver をロードします。
        DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
        // データベースに接続します。
        Connection conn =
            DriverManager.getConnection ("jdbc:oracle:oci8:@", "samp", "samp");
        // 自動コミットがオフの場合、高速になります。
        conn.setAutoCommit (false);
        // 文を作成します。
        Statement stmt = conn.createStatement ();

        try
        {
            BLOB blob = null;
            ResultSet rset = stmt.executeQuery (
                "SELECT frame FROM multimedia_tab WHERE clip_id = 1");
            if (rset.next())
            {
                blob = ((OracleResultSet)rset).getBLOB (1);
            }

            OracleCallableStatement cstmt =
                (OracleCallableStatement) conn.prepareCall (
                    "BEGIN ? := DBMS_LOB.ISOPEN(?); END;");
            cstmt.registerOutParameter (1, Types.NUMERIC);
            cstmt.setBLOB(2, blob);
            cstmt.execute();
            int result = cstmt.getInt(1);
            System.out.println("The result is: " + Integer.toString(result));

            OracleCallableStatement cstmt2 = (OracleCallableStatement)
                conn.prepareCall (
                    "BEGIN DBMS_LOB.OPEN(?,DBMS_LOB.LOB_READONLY); END;");
            cstmt2.setBLOB(1, blob);
            cstmt2.execute();
        }
    }
}
```

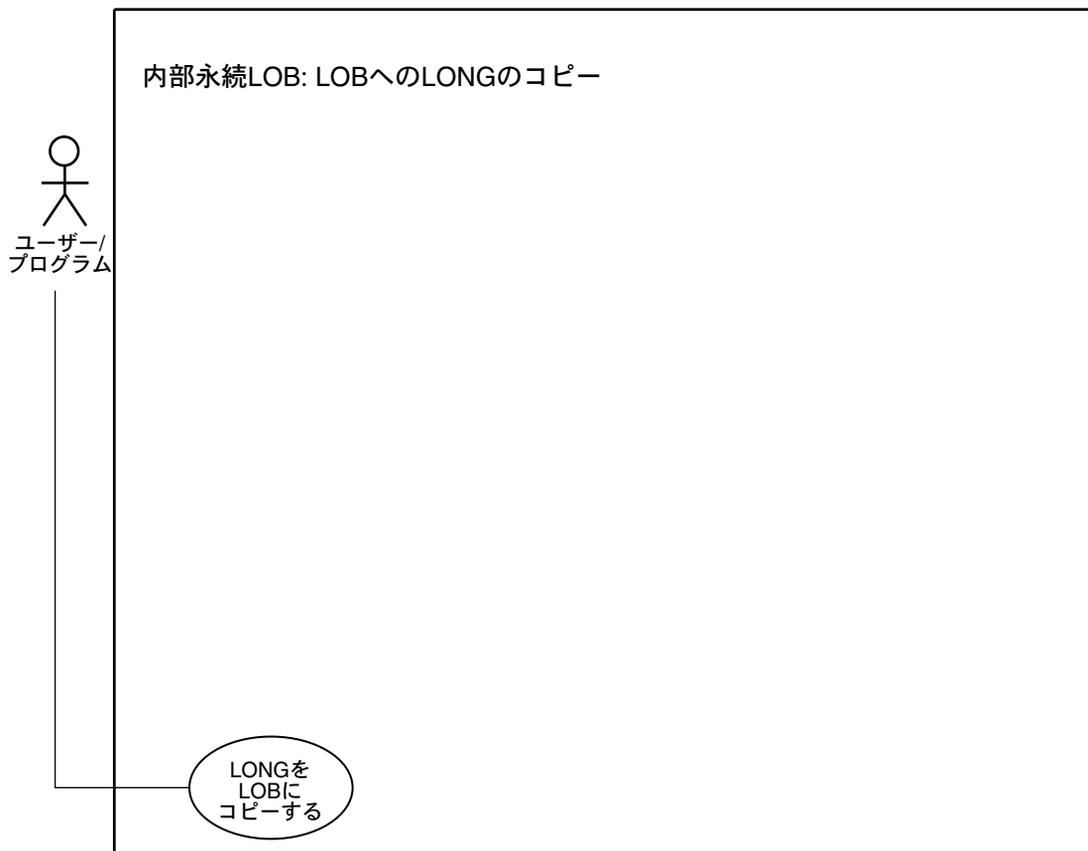
```
System.out.println("The LOB has been opened with a call to DBMS_LOB.OPEN()");

// 既存の cstmt ハンドルを使用して、ロケータの状態を再問合せします。
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();
}
}
```

LOB への LONG のコピー

図 9-18 ユースケース図：LOB への LONG のコピー



参照： 内部永続 LOB に関するすべての基本操作については、9-2 ページの「ユースケース・モデル：内部永続 LOB – 基本操作」を参照してください。

用途

LONG を LOB にコピーします。

使用上の注意

TO_LOB の使用には、次のような制限事項があることに注意してください。

- TO_LOB を使用してデータを LOB 列にコピーできますが、LOB 属性にはコピーできません。
- TO_LOB はどのリモート表にも使用できません。したがって、次のような文はすべて失敗します。

```
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;
```

- ターゲット表 (LOB 列を持つ表) が、BEFORE INSERT や INSTEAD OF INSERT のようなトリガーを持っている場合、:NEW.lob_col 変数はトリガー本体の中では参照できません。
- どの PL/SQL ブロックの中にも TO_LOB を配置できません。
- TO_LOB 関数を使用してデータを CLOB にコピーできますが、NCLOB にはコピーできません。これは、LONG データ型が CHAR データベース・キャラクタ・セットを持ち、同じ CHAR データベース・キャラクタ・セットを使用する CLOB にのみ変換できるためです。これに対して、NCLOB は NCHAR データベース・キャラクタ・セットを使用します。

構文

次のマニュアルの項を参照してください。

- SQL: 『Oracle8i SQL リファレンス』の第4章「関数」の「TO_LOB」

使用例

次のアーカイブ・ソース表 SoundsLib_tab が定義され、データが含まれているとします。

```
CREATE TABLE SoundsLib_tab  
(  
  Id          NUMBER,  
  Description  VARCHAR2(30),  
  SoundEffects LONG RAW  
);
```

次の例では、データをマルチメディア表の LONG RAW 列 (SoundEffects) から BLOB 列 (Sound) にコピーし、SQL 関数 TO_LOB を使用してこれを実行するとします。

例

例が SQL で示されています。この例は、すべてのプログラム環境に適用されます。

■ SQL: LONG の LOB へのコピー

SQL: LONG の LOB へのコピー

```
INSERT INTO Multimedia_tab (clip_id,sound) SELECT id, TO_LOB(SoundEffects)
FROM SoundsLib_tab WHERE id =1;
```

注意： 前述の操作を正常に行うには、次の手順を実行します

```
CREATE TABLE SoundsLib_tab (
    id          NUMBER,
    SoundEffects LONG RAW);
```

この機能は LONG を LOB に変換する、TO_LOB と呼ばれる演算子に基づいています。TO_LOB 演算子は、LONG 列のすべての行にあるデータを対応する LOB 列にコピーし、LONG データに対して LOB 機能を適用します。LONG 列に格納されるデータの型は、LOB に格納されるデータの型と一致する必要があります。たとえば、LONG RAW データは BLOB データにコピーする必要があり、LONG データは CLOB データにコピーする必要があります。

この操作を一回のみ実行し、データが正常にコピーされると、LONG 列を削除できます。ただし、LONG を表に格納するために必要だったすべての記憶域を再生できるわけではありません。不要な格納は行わないために、LONG データを新しい表か別の表の LOB にコピーすることをお勧めします。データが実際にコピーされたことを確認した後、元の表を削除できません。

この LONG から LOB への変換を簡単に行う方法として、LONG 列上の TO_LOB 演算子を SELECT 文の一部として使用し、CREATE TABLE... SELECT 文を使用する方法があります。また、INSERT... SELECT も使用できます。

次の方法では、LONG_TAB 表にある LONG_COL という名前の LONG 列を、LOB_TAB 表にある LOB_COL という名前の LOB 列にコピーします。これらの表には、表の中の各列の識別番号を含む ID 列があります。

次のステップを実行して、データを LONG 列から LOB 列にコピーします。

1. LONG 列を含む表と同じ定義で新しい表を作成します。ただし、LONG データ型のかわりに LOB データ型を使用します。

たとえば、次のように定義された表を持っていたとします。

```
CREATE TABLE Long_tab (  
    id          NUMBER,  
    long_col LONG);
```

次の SQL 文を使用して新しい表を作成します。

```
CREATE TABLE Lob_tab (  
    id          NUMBER,  
    blob_col BLOB);
```

注意： 新しい表を作成するときには、整合性制約、トリガー、権限付与、索引などの表のスキーマを保持していることを確認してください。TO_LOB 演算子はデータをコピーするのみです。表のスキーマは保持しません。

2. TO_LOB 演算子を使用して INSERT コマンドを発行し、データを LONG データ型の表から LOB データ型に挿入します。

たとえば、次のような SQL 文を発行します。

```
INSERT INTO Lob_tab  
    SELECT id,  
    TO_LOB(long_col)  
    FROM long_tab;
```

3. コピーが正常に終了したことを確認してから、LONG 列を含む表を削除します。

たとえば、次のような SQL コマンドを発行して LONG_TAB 表を削除します。

```
DROP TABLE Long_tab;
```

4. LONG データを含む表の名前を使用して、新しい表のシノニムを作成します。このシノニムはユーザーのデータベースおよびアプリケーションが正しく機能し続けていることを保証します。

たとえば、次のような SQL 文を発行します。

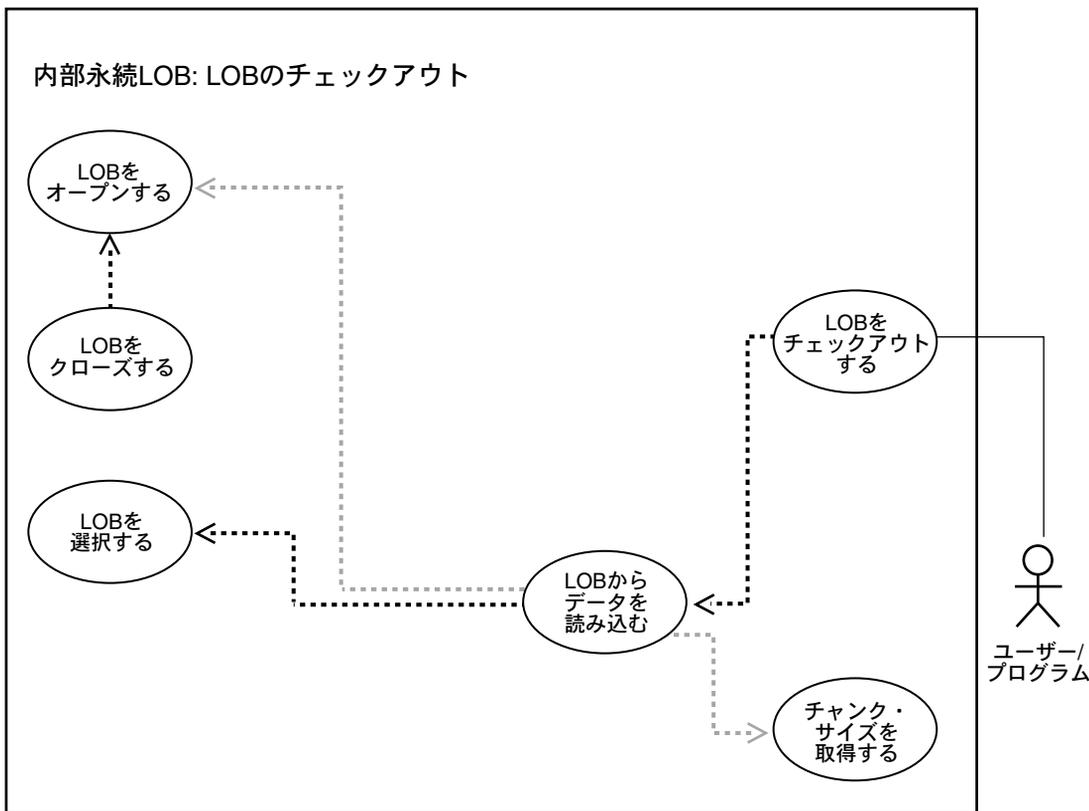
```
CREATE SYNONYM Long_tab FOR Lob_tab;
```

コピーが完了したら、その表を使用するすべてのアプリケーションを、LOB データを使用できるように変更する必要があります。

TO_LOB 演算子を使用して、データを LONG から CREATE TABLE...AS SELECT または INSERT...SELECT を使用した文の中の LOB にコピーします。INSERT...SELECT の場合は、UPDATE の前に、表を ALTER し、LOB 列を ADD しておく必要があります。UPDATE がエラーを戻す場合は (UNDO 用の領域がなかった場合)、WHERE 句を使用して LONG データを少しずつ LOB に移行できます。WHERE 句は LOB に対する機能を持ちませんが、LOB が NULL であるかどうかをテストできます。

LOB のチェックアウト

図 9-19 ユースケース図: LOB のチェックアウト



参照: 内部永続 LOB に関するすべての基本操作については、9-2 ページの「ユースケース・モデル:内部永続 LOB – 基本操作」を参照してください。

用途

LOB をチェックアウトします。

使用上の注意

ストリーム・メカニズム

大量の LOB データを最も効率よく読み込む方法は、ポーリングまたはコールバックを介してストリーム・メカニズムを使用可能にし、`OCILOBRead()` を使用することです。基礎となる読み込み操作を行うには、ストリーミングを伴う OCI または PRO*C インタフェースを使用します。`DBMS_LOB.READ` を使用するとパフォーマンスは最適化されません。

構文

各プログラム環境で使用可能な機能のリストは、第 3 章「LOB プログラム環境」を参照してください。各プログラム環境における構文については、次のマニュアルの項を参照してください。

- PL/SQL (DBMS_LOB パッケージ) : 『Oracle8i PL/SQL パッケージ・プロシージャ・リファレンス』の第 18 章「DBMS_LOB」の「サブプログラムの要約」の OPEN プロシージャ、READ プロシージャ、CLOSE プロシージャ
- C (OCI) : 『Oracle8i コール・インタフェース・プログラマーズ・ガイド』の第 15 章「OCI リレーショナル関数」の「LOB 関数」の `OCILOBOpen()`、`OCILOBRead()`、`OCILOBClose()`
- COBOL (Pro*COBOL) : 『Oracle8i Pro*COBOL プリコンパイラ・プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB READ (実行可能埋込み SQL 拡張要素)」
- C/C++ (Pro*C) : 『Oracle8i Pro*C/C++ プリコンパイラ・プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB OPEN (実行可能埋込み SQL 拡張要素)」、「LOB READ (実行可能埋込み SQL 拡張要素)」
- Visual Basic (Oracle Objects for OLE オンライン・ヘルプ) : 「Help」の「Contents」タブから、「OO4O Automation Server」>「OBJECTS」>「Orablob」>「METHODS」>「read」、「copytofile」を選択してください。
- Java (JDBC) : 『Oracle8i JDBC 開発者ガイドおよびリファレンス』の第 7 章「LOB と BFILE の操作」の「BLOB と CLOB の操作」の「BLOB または CLOB 列の作成と移入」、および『Oracle8i SQLJ 開発者ガイドおよびリファレンス』の第 5 章「型のサポート」の「Oracle の拡張型」の「BLOB、CLOB および BFILE のサポート」

使用例

チェックアウト / チェックイン操作の例としては、データベースから LOB のバージョンをクライアントへチェックアウトし、データベースにアクセスしないでクライアント上のデータを変更し、クライアント側でドキュメントに加えられた変更を一度にチェックインするような操作が考えられます。

次に、この使用例のチェックアウト部分を示します。このコードを使用すると、CLOB Transcript を NESTED TABLE である InSeg_ntab から読み込むことができます。この表には、ビュー間のセグメントが含まれ、クライアント側のテキスト・エディタで処理できます。使用例のチェックイン部分の詳細は、9-75 ページの「LOB のチェックイン」を参照してください。

例

次の例は、「LOB データの表示」に示されている例と似ています。次のプログラム環境の例が示されています。

- [PL/SQL \(DBMS_LOB パッケージ\) :LOB のチェックアウト \(9-66 ページ\)](#)
- [C \(OCI\) :LOB のチェックアウト \(9-67 ページ\)](#)
- [COBOL \(Pro*COBOL\) :LOB のチェックアウト \(9-69 ページ\)](#)
- [C/C++ \(Pro*C\) :LOB のチェックアウト \(9-71 ページ\)](#)
- [Visual Basic \(OO4O\) :LOB のチェックアウト \(9-72 ページ\)](#)
- [Java \(JDBC\) :LOB のチェックアウト \(9-73 ページ\)](#)

PL/SQL (DBMS_LOB パッケージ) :LOB のチェックアウト

```
/* 例のプロシージャ checkOutLOB_proc は、DBMS_LOB パッケージの
一部ではないことに注意してください。*/
CREATE OR REPLACE PROCEDURE checkOutLOB_proc IS
  Lob_loc      CLOB;
  Buffer        VARCHAR2 (32767);
  Amount       BINARY_INTEGER := 32767;
  Position     INTEGER := 2147483647;
BEGIN
  /* LOB を選択します。*/
  SELECT Intab.Transcript INTO Lob_loc
     FROM TABLE (SELECT Mtab.InSeg_ntab FROM Multimedia_tab Mtab
                  WHERE Mtab.Clip_ID = 1) Intab
         WHERE Intab.Segment = 1;
  /* LOB のオープンはオプションです。*/
  DBMS_LOB.OPEN (Lob_loc, DBMS_LOB.LOB_READONLY);
  LOOP
    DBMS_LOB.READ (Lob_loc, Amount, Position, Buffer);
    /* バッファを処理します。*/
    Position := Position + Amount;
  END LOOP;
  /* LOB をオープンしている場合、その LOB をクローズする必要があります。*/
  DBMS_LOB.CLOSE (Lob_loc);
EXCEPTION
```

```

WHEN NO_DATA_FOUND THEN
    DBMS_OUTPUT.PUT_LINE('End of data');
END;

```

C (OCI) : LOB のチェックアウト

/* この例では、標準のポーリング方法を使用して、BLOB の内容全体がピース単位でバッファに読み込まれます。このポーリング方法では、各バッファのピースが、BLOB 全体が読み込まれるまで、すべての READ 操作後に処理されます。*/

```

#define MAXBUFLLEN 32767
/* ロケータ変数にロケータを選択します。*/
sb4 select_transcript_locator(Lob_loc, errhp, stmthp, svchp)
OCILobLocator *Lob_loc;
OCIError      *errhp;
OCISvcCtx     *svchp;
OCIStmt       *stmthp;
{
    text *sqlstmt =
        (text *) "SELECT Intab.Transcript \
                FROM TABLE(SELECT Mtab.InSeg_ntab FROM Multimedia_tab Mtab \
                WHERE Mtab.Clip_ID = 1) Intab \
                WHERE Intab.Segment = 1";
    OCIDefine *defnp1;
    checkerr (errhp, OCIStmtPrepare(stmthp, errhp, sqlstmt,
                                    (ub4)strlen((char *)sqlstmt),
                                    (ub4) OCI_NTV_SYNTAX, (ub4) OCI_DEFAULT));
    checkerr (errhp, OCIDefineByPos(stmthp, &defnp1, errhp, (ub4) 1,
                                    (dvoid *)&Lob_loc, (sb4)0,
                                    (ub2) SQLT_CLOB, (dvoid *) 0,
                                    (ub2 *) 0, (ub2 *) 0, (ub4) OCI_DEFAULT));
    /* SELECT を実行し、1 行フェッチします。*/
    checkerr (errhp, OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                                    (CONST OCISnapshot*) 0, (OCISnapshot*) 0,
                                    (ub4) OCI_DEFAULT));

    return 0;
}

void checkoutLob(envhp, errhp, svchp, stmthp)
OCIEnv      *envhp;
OCIError    *errhp;
OCISvcCtx   *svchp;
OCIStmt     *stmthp;
{
    OCILobLocator *Lob_loc;
    ub4 amt;

```

```
ub4 offset;
sword retval;
boolean done;
ub1 bufp[MAXBUFLen];
ub4 buflen;

/* ロケータ記述子を割り当てます。*/
(void) OCIDescriptorAlloc((dvoid *)envhp, (dvoid **) &Lob_loc,
                          (ub4)OCI_DTYPE_LOB, (size_t) 0, (dvoid **) 0);
/* BLOB を選択します。*/
printf(" select the transcript locator...\n");
select_transcript_locator(Lob_loc, errhp, stmthp, svchp);

/* CLOB をオープンします。*/
printf (" open lob in checkOutLOB_proc\n");
checkerr (errhp, (OCILobOpen(svchp, errhp, Lob_loc, OCI_LOB_READONLY)));

/* amt = 0 (ゼロ) に設定すると、LOB の終わりまで読み込まれます。*/
amt = 0;
buflen = sizeof(bufp);

/* データをピースごとに処理します。*/
printf (" read lob in pieces\n");
offset = 1;
memset(bufp, '\0', MAXBUFLen);
done = FALSE;
while (!done)
{
    retval = OCILobRead(svchp, errhp, Lob_loc, &amt, offset, (dvoid *)bufp,
                       buflen, (dvoid *)0, (sb4 *)0, (dvoid *)0, (dvoid *)0, (ub4,
                       ub1) 0, (ub2) 0, (ub1) SQLCS_IMPLICIT);

    switch (retval)
    {
        case OCI_SUCCESS:
            /* 1つのピースか、または最後のピースのみです。*/
            /* bufp のデータを処理します。amt には、bufp に直前に読み込まれたデータの量が示され
            ます。これは、BLOB ではバイト、固定幅 CLOB では文字、および可変幅 CLOB ではバイト
            単位です。*/
            done = TRUE;
            break;
        case OCI_ERROR:
            checkerr (errhp, OCI_ERROR);
            done = TRUE;
            break;
    }
}
```

```

        case OCI_NEED_DATA:          /* 2つ以上のピースがあります。*/
/* bufp のデータを処理します。amt には、bufp に直前に読み込まれたデータの量が示され
   ます。これは、BLOB ではバイト、固定幅 CLOB では文字、および可変幅 CLOB ではバイト
   単位です。*/
        break;
        default:
            checkerr (errhp, retval);
done = TRUE;
        break;
    } /* while */
}
/* CLOB をオープンしている場合、その CLOB をクローズする必要があります。*/
printf (" close lob in checkOutLOB_proc\n");
checkerr (errhp, OCILobClose(svchp, errhp, Lob_loc));

/* ロケータが保持しているリソースを解放します。*/
(void) OCIDescriptorFree((dvoid *) Lob_loc, (ub4) OCI_DTYPE_LOB);

return;
}

```

COBOL (Pro*COBOL) : LOB のチェックアウト

```

IDENTIFICATION DIVISION.
PROGRAM-ID. CHECKOUT.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

01  USERID    PIC X(11) VALUES "SAMP/SAMP".
01  CLOB1     SQL-CLOB.
01  BUFFER    PIC X(5) VARYING.
01  AMT       PIC S9(9) COMP.
01  OFFSET    PIC S9(9) COMP VALUE 1.
01  D-BUFFER-LEN PIC 9.
01  D-AMT     PIC 9.
      EXEC SQL INCLUDE SQLCA END-EXEC.

PROCEDURE DIVISION.
READ-CLOB.
      EXEC SQL WHENEVER SQLERROR DO PERFORM SQL-ERROR END-EXEC.
      EXEC SQL
          CONNECT :USERID
      END-EXEC.

```

```
* CLOB ロケータの割当ておよび初期化を行います。
EXEC SQL ALLOCATE :CLOB1 END-EXEC.
EXEC SQL WHENEVER NOT FOUND GOTO END-OF-CLOB END-EXEC.
EXEC SQL
    SELECT STORY INTO :CLOB1 FROM MULTIMEDIA_TAB
    WHERE CLIP_ID = 2
END-EXEC.

* ポーリング読み込みを開始します。
MOVE 0 TO AMT.
* バッファに CLOB の最初のピースを読み込みます。
EXEC SQL
    LOB READ :AMT FROM :CLOB1 AT :OFFSET INTO :BUFFER
END-EXEC.
DISPLAY "Reading a CLOB ...".
DISPLAY " ".
MOVE BUFFER-LEN TO D-BUFFER-LEN.
DISPLAY "first read (", D-BUFFER-LEN, "): "
    BUFFER-ARR(1:BUFFER-LEN).

* CLOB の後続のピースを読み込みます。
READ-LOOP.
MOVE " " TO BUFFER-ARR.
EXEC SQL
    LOB READ :AMT FROM :CLOB1 INTO :BUFFER
END-EXEC.
MOVE BUFFER-LEN TO D-BUFFER-LEN.
DISPLAY "next read (", D-BUFFER-LEN, "): "
    BUFFER-ARR(1:BUFFER-LEN).

GO TO READ-LOOP.

* CLOB の最後のピースを読み込みます。
END-OF-CLOB.
EXEC SQL WHENEVER NOT FOUND CONTINUE END-EXEC.
EXEC SQL FREE :CLOB1 END-EXEC.
MOVE BUFFER-LEN TO D-BUFFER-LEN.
DISPLAY "last read (", D-BUFFER-LEN, "): "
    BUFFER-ARR(1:BUFFER-LEN).
EXEC SQL
    ROLLBACK WORK RELEASE
END-EXEC.
STOP RUN.

SQL-ERROR.
```

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

C/C++ (Pro*C) : LOB のチェックアウト

/* この例では、標準のポーリング方法を使用して、CLOB の内容全体がピース単位でバッファに READ されます。このポーリング方法では、各バッファのピースが、CLOB 全体が読み込まれるまで、すべての READ 操作後に処理されます。*/

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

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

#define BufferLength 256

void checkOutLOB_proc()
{
    OCIClobLocator *Lob_loc;
    int Amount;
    int Clip_ID, Segment;
    VARCHAR Buffer[BufferLength];

    EXEC SQL WHENEVER SQLERROR DO Sample_Error();
    EXEC SQL ALLOCATE :Lob_loc;

    /* ダイナミック SQL を使用して、LOB を取り出します。*/
    EXEC SQL PREPARE S FROM
        'SELECT Intab.Transcript \
```

```
FROM TABLE(SELECT Mtab.InSeg_ntab FROM Multimedia_tab Mtab \
WHERE Mtab.Clip_ID = :cid) Intab \
WHERE Intab.Segment = :seg';
EXEC SQL DECLARE C CURSOR FOR S;
Clip_ID = Segment = 1;
EXEC SQL OPEN C USING :Clip_ID, :Segment;
EXEC SQL FETCH C INTO :Lob_loc;
EXEC SQL CLOSE C;

/* LOB をオープンします。 */
EXEC SQL LOB OPEN :Lob_loc READ ONLY;
/* Setting Amount = 0 will initiate the polling method: */
Amount = 0;
/* バッファの最大サイズを設定します。 */
Buffer.len = BufferLength;
EXEC SQL WHENEVER NOT FOUND DO break;
while (TRUE)
{
    /* バッファに LOB のピースを読み込みます。 */
    EXEC SQL LOB READ :Amount FROM :Lob_loc INTO :Buffer;
    printf("Checkout %d characters\n", Buffer.len);
}
printf("Checkout %d characters\n", Amount);

/* LOB をオープンしている場合、その LOB をクローズする必要があります。 */
EXEC SQL LOB CLOSE :Lob_loc;
EXEC SQL FREE :Lob_loc;
}

void main()
{
    char *samp = "samp/samp";
    EXEC SQL CONNECT :samp;
    checkOutLOB_proc();
    EXEC SQL ROLLBACK WORK RELEASE;
}
```

Visual Basic (0040) : LOB のチェックアウト

'LOB の読み込みには、orablob.read または orablob.copytofile を使用する 2 つの方法があります。

'OraBlob.Read メカニズムの使用

```
Dim OraDyn as OraDynaset, OraSound as OraBlob, amount_read%, chunksize%, chunk
```

```
chunksize = 32767
set OraDyn = OraDb.CreateDynaset("select * from Multimedia_tab", ORADYN_DEFAULT)
set OraSound = OraDyn.Fields("Sound").Value
OraSound.PollingAmount = OraSound.Size 'Read entire BLOB contents
Do
    amount_read = OraSound.Read(chunk, chunksize) 'chunk returned is a variant of
type byte array
Loop Until OraSound.Status <> ORALOB_NEED_DATA

'OraBlob.CopyToFile メカニズムの使用
Set OraDyn = OraDb.CreateDynaset("select * from Multimedia_tab", ORADYN_DEFAULT)
Set OraSound = OraDyn.Fields("Sound").Value

'BLOB 全体の内容を読み込みます。
OraSound.CopyToFile "c:\mysound.aud"
```

Java (JDBC) : LOB のチェックアウト

```
import java.io.InputStream;
import java.io.OutputStream;

// コア JDBC クラス :
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 固有の JDBC クラス :
import oracle.sql.*;
import oracle.jdbc.driver.*;

public class Ex2_59
{
    static final int MAXBUFSIZE = 32767;
    public static void main (String args [])
        throws Exception
    {
        // Oracle JDBC Driver をロードします。
        DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
        // データベースに接続します。
        Connection conn =
```

```
    DriverManager.getConnection ("jdbc:oracle:oci8:@", "samp", "samp");
// 自動コミットがオフの場合、高速になります。
conn.setAutoCommit (false);
// 文を作成します。
Statement stmt = conn.createStatement ();
try
{
    CLOB src_lob = null;
    InputStream in = null;
    byte buf[] = new byte[MAXBUFSIZE];

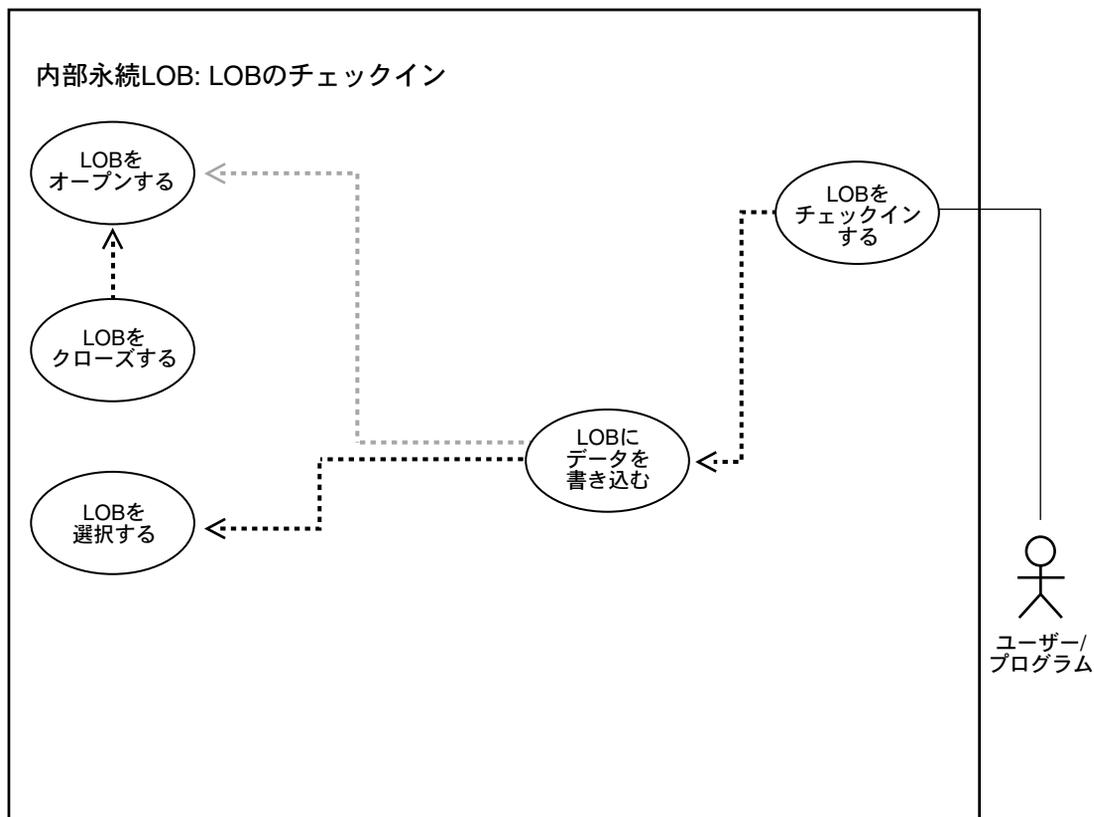
    ResultSet rset = stmt.executeQuery (
        "SELECT intab.transcript FROM TABLE( "
        + " SELECT mtab.inseg_ntab FROM multimedia_tab mtab "
        + " WHERE mtab.clip_id = 1) intab WHERE intab.segment = 1");
    if (rset.next())
    {
        src_lob = ((OracleResultSet)rset).getCLOB (1);
        in = src_lob.getAsciiStream();
    }

    int length = 0;
    int pos = 0;
    while ((in != null) && ((length = in.read(buf)) != -1))
    {
        pos += length;
        System.out.println(Integer.toString(pos));
        // バッファを処理します。
    }

    in.close();
    rset.close();
    stmt.close();
    conn.commit();
    conn.close();
}
catch (SQLException e)
{
    e.printStackTrace();
}
}
```

LOB のチェックイン

図 9-20 ユースケース図：LOB のチェックイン



参照： 内部永続LOBに関するすべての基本操作については、9-2 ページの「ユースケース・モデル：内部永続LOB – 基本操作」を参照してください。

用途

LOB をチェックインします。

使用上の注意

ストリーム・メカニズム

大量の LOB データを最も効率よく書き込む方法は、ポーリングまたはコールバックを介してストリーム・メカニズムを使用可能にし、OCILOBWrite() を使用することです。

構文

各プログラム環境で使用可能な機能のリストは、第3章「LOB プログラム環境」を参照してください。各プログラム環境における構文については、次のマニュアルの項を参照してください。

- PL/SQL (DBMS_LOB パッケージ) : 『Oracle8i PL/SQL パッケージ・プロシージャ・リファレンス』の第18章「DBMS_LOB」の「サブプログラムの要約」の OPEN プロシージャ、WRITE プロシージャ、CLOSE プロシージャ
- C (OCI) : 『Oracle8i コール・インタフェース・プログラマーズ・ガイド』の第15章「OCI リレーショナル関数」の「LOB 関数」の OCILOBOpen()、OCILOBWrite()、OCILOBClose()
- COBOL (Pro*COBOL) : 『Oracle8i Pro*COBOL プリコンパイラ・プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB WRITE (実行可能埋込み SQL 拡張要素)」
- C/C++ (Pro*C) : 『Oracle8i Pro*C/C++ プリコンパイラ・プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB WRITE (実行可能埋込み SQL 拡張要素)」
- Visual Basic (Oracle Objects for OLE オンライン・ヘルプ) : 「Help」の「Contents」タブから、「OO4O Automation Server」>「OBJECTS」>「Orablob」>「METHODS」>「write」、および「OBJECTS」>「Oradynaset」>「update」を選択してください。
- Java (JDBC) : 『Oracle8i JDBC 開発者ガイドおよびリファレンス』の第7章「LOB と BFILE の操作」の「BLOB と CLOB の操作」の「BLOB または CLOB 列の作成と移入」、および『Oracle8i SQLJ 開発者ガイドおよびリファレンス』の第5章「型のサポート」の「Oracle の拡張型」の「BLOB、CLOB および BFILE のサポート」

使用例

ここで説明する checkin 操作は、9-64 ページの「LOB のチェックアウト」の続きです。この場合、ビュー間のセグメントを含む NESTED TABLE である InSeg_ntab 中の CLOB Transcript 列にデータを書き戻す手順になります。このように、基本となる書き込み操作にストリーミングを使用するには、OCI または Pro*C インタフェースを使用します。DBMS_LOB.WRITE を使用するとパフォーマンスは最適化されません。

次の例は、様々なプログラム環境を使用した LOB のチェックイン方法を示しています。

例

次のプログラム環境の例が示されています。

- PL/SQL (DBMS_LOB パッケージ) : LOB のチェックイン (9-77 ページ)
- C (OCI) : LOB のチェックイン (9-78 ページ)
- COBOL (Pro*COBOL) : LOB のチェックイン (9-81 ページ)
- C/C++ (Pro*C) : LOB のチェックイン (9-83 ページ)
- Visual Basic (OO4O) : LOB のチェックイン (9-86 ページ)
- Java (JDBC) : LOB のチェックイン (9-87 ページ)

PL/SQL (DBMS_LOB パッケージ) : LOB のチェックイン

```
/* 例のプロシージャ checkInLOB_proc は、DBMS_LOB パッケージの
一部ではないことに注意してください。*/
CREATE OR REPLACE PROCEDURE checkInLOB_proc IS
  Lob_loc      CLOB;
  Buffer        VARCHAR2(32767);
  Amount       BINARY_INTEGER := 32767;
  Position     INTEGER := 2147483647;
  i            INTEGER;
BEGIN
  /* LOB を選択します。*/
  SELECT Intab.Transcript INTO Lob_loc
     FROM TABLE(SELECT Mtab.InSeg_ntab FROM Multimedia_tab Mtab
                  WHERE Clip_ID = 2) Intab
     WHERE Intab.Segment = 1
     FOR UPDATE;
  /* LOB のオープンはオプションです。*/
  DBMS_LOB.OPEN (Lob_loc, DBMS_LOB.LOB_READWRITE)
  FOR i IN 1..3 LOOP
    /* バッファに書き込むデータを充填します。*/
    /* データを書き込みます。*/
    DBMS_LOB.WRITE (Lob_loc, Amount, Position, Buffer);
    Position := Position + Amount;
  END LOOP;
  /* LOB をオープンしている場合、その LOB をクローズする必要があります。*/
  DBMS_LOB.CLOSE (Lob_loc);

EXCEPTION
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('Operation failed');
END;
```

C (OCI) : LOB のチェックイン

/* この例では、OCI が、単一のピースまたは複数のピースごとに任意の量のデータを内部 LOB に書き込む機能を提供する方法を示します。これには、標準のポーリングを使用するストリーミング・メカニズムが使用されます。静的に割り当てられたバッファを使用して、LOB に書き込まれているデータが保持されます。*/

```
#define MAXBUFLLEN 32767
/* ロケータ変数にロケータを選択します。*/
sb4 select_lock_transcript_locator(Lob_loc, errhp, stmthp, svchp)
OCILobLocator *Lob_loc;
OCIError      *errhp;
OCISvcCtx     *svchp;
OCIStmt       *stmthp;
{
    OCIDefine *defnp1;

    text *sqlstmt =
        (text *) "SELECT Intab.Transcript \
                FROM TABLE(SELECT Mtab.InSeg_ntab FROM Multimedia_tab Mtab \
                WHERE Mtab.Clip_ID = 2) Intab \
                WHERE Intab.Segment = 1 FOR UPDATE";

    checkerr (errhp, OCIStmtPrepare(stmthp, errhp, sqlstmt,
                                    (ub4)strlen((char *)sqlstmt),
                                    (ub4) OCI_NTV_SYNTAX, (ub4) OCI_DEFAULT));
    checkerr (errhp, OCIDefineByPos(stmthp, &defnp1, errhp, (ub4) 1,
                                    (dvoid *)&Lob_loc, (sb4) 0,
                                    (ub2) SQLT_CLOB, (dvoid *) 0,
                                    (ub2 *) 0, (ub2 *) 0, (ub4) OCI_DEFAULT));
    /* SELECT を実行し、1 行フェッチします。*/
    checkerr (errhp, OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                                    (CONST OCISnapshot*) 0, (OCISnapshot*) 0,
                                    (ub4) OCI_DEFAULT));

    return OCI_SUCCESS;
}

void checkinLob(envhp, errhp, svchp, stmthp)
OCIEnv      *envhp;
OCIError    *errhp;
OCISvcCtx   *svchp;
OCIStmt     *stmthp;
{
    OCIClobLocator *Lob_loc;
    ub4 Total = 2.5*MAXBUFLLEN;
    ub4 amtp;
```

```
ub4 offset;
ub4 remainder;
ub4 nbytes;
boolean last;
ub1 bufp[MAXBUFLEN];
sb4 err;

/* ロケータ記述子を割り当てます。*/
(void) OCIDescriptorAlloc((dvoid *)envhp, (dvoid **) &Lob_loc,
                          (ub4)OCI_DTYPE_LOB, (size_t) 0, (dvoid **) 0);
/* CLOB を選択します。*/
printf(" select the transcript locator...\n");
select_lock_transcript_locator(Lob_loc, errhp, stmthp, svchp);

/* CLOB をオープンします。*/
printf (" open the locator.\n");
checkerr (errhp, (OCILobOpen(svchp, errhp, Lob_loc, OCI_LOB_READWRITE)));

printf (" write the lob in pieces\n");
if (Total > MAXBUFLEN)
    nbytes = MAXBUFLEN; /* 標準のポーリングを介してストリーミングを使用します。*/
else
    nbytes = Total; /* 単一の書き込みのみが必要です。*/

/* バッファにnバイト相当のデータを充填します。*/
remainder = Total - nbytes;
/* Amount を 0 (ゼロ) に設定すると、OCI_LAST_PIECE を指定するまでデータが流されます。*/
amp = 0;
/* offset = <データの書き込みを開始する開始点> */
offset = 1;

if (0 == remainder)
{
    amp = nbytes;
    /* ここで、合計がMAXBUFLEN以下であるため、1ピースごと書き込みできます。*/
    checkerr (errhp, OCILobWrite (svchp, errhp, Lob_loc, amp,
                                offset, bufp, nbytes,
                                OCI_ONE_PIECE, (dvoid *) 0,
                                (sb4 *) (dvoid *, dvoid *, ub4 *, ub1 *) 0,
                                0, SQLCS_IMPLICIT));
}
else
{
    /* ここで、合計がMAXBUFLENより大きいため、標準のポーリングを介してストリーミングを使用します。*/
    /* 最初のピースを書き込みます。FIRST を指定すると、ポーリングが開始します。*/
    err = OCILobWrite (svchp, errhp, Lob_loc, &amp, offset, bufp, nbytes,
```

```
OCI_FIRST_PIECE, (dvoid *) 0,
    (sb4 (*) (dvoid *, dvoid *, ub4 *, ub1 *)) 0,
    0, SQLCS_IMPLICIT);
if (err != OCI_NEED_DATA)
    checkerr (errhp, err);
last = FALSE;

/* 次 (中間) および最後のピースを書き込みます。*/
do
{
    if (remainder > MAXBUFLLEN)
        nbytes = MAXBUFLLEN;    /* ピースが残っています。*/
    else
    {
        nbytes = remainder;    /* ここで、remainder は MAXBUFLLEN 以下です。*/
        last = TRUE;          /* これが最後のピースになります。*/
    }

    /* バッファに n バイト相当のデータを充填します。*/
    if (last)
    {
        /* LAST を指定すると、ポーリングが終了します。*/
        err = OCILobWrite (svchp, errhp, Lob_loc, &amp;
            offset, bufp, nbytes,
            OCI_LAST_PIECE, (dvoid *) 0,
            (sb4 (*) (dvoid*, dvoid*, ub4*, ub1 *)) 0,
            0, SQLCS_IMPLICIT);

        if (err != OCI_SUCCESS)
            checkerr (errhp, err);
    }
    else
    {
        err = OCILobWrite (svchp, errhp, Lob_loc, &amp;
            offset, bufp, nbytes,
            OCI_NEXT_PIECE, (dvoid *) 0,
            (sb4 (*) (dvoid*, dvoid*, ub4*, ub1 *)) 0,
            0, SQLCS_IMPLICIT);

        if (err != OCI_NEED_DATA)
            checkerr (errhp, err);
    }
    /* 残りの書込み量を判断します。*/
    remainder = remainder - nbytes;
} while (!last);
}

/* この時点では、(remainder == 0 (ゼロ)) です。*/
```

```

/* BLOB をオープンしている場合、その BLOB をクローズする必要があります。*/
checkerr (errhp, OCILobClose(svchp, errhp, Lob_loc));

/* ロケータが保持しているリソースを解放します。*/
(void) OCIDescriptorFree((dvoid *) Lob_loc, (ub4) OCI_DTYPE_LOB);

}

```

COBOL (Pro*COBOL) : LOB のチェックイン

```

IDENTIFICATION DIVISION.
PROGRAM-ID. CHECKIN.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT INFILE
        ASSIGN TO "datfile.dat"
        ORGANIZATION IS SEQUENTIAL.
DATA DIVISION.
FILE SECTION.

FD INFILE
    RECORD CONTAINS 80 CHARACTERS.
01 INREC          PIC X(80).

WORKING-STORAGE SECTION.
01 USERID        PIC X(11) VALUES "SAMP/SAMP".
01 CLOB1         SQL-CLOB.
01 BUFFER        PIC X(80) VARYING.
01 AMT           PIC S9(9) COMP VALUE 0.
01 OFFSET        PIC S9(9) COMP VALUE 1.
01 END-OF-FILE   PIC X(1) VALUES "N".
01 D-BUFFER-LEN  PIC 9.
01 D-AMT         PIC 9.
    EXEC SQL INCLUDE SQLCA END-EXEC.

PROCEDURE DIVISION.
WRITE-CLOB.
    EXEC SQL WHENEVER SQLERROR DO PERFORM SQL-ERROR END-EXEC.
    EXEC SQL CONNECT :USERID END-EXEC.

* CLOB ロケータの割当ておよび初期化を行います。
    EXEC SQL ALLOCATE :CLOB1 END-EXEC.
    EXEC SQL
        SELECT STORY INTO :CLOB1 FROM MULTIMEDIA_TAB
        WHERE CLIP_ID = 1 FOR UPDATE

```

```
END-EXEC.
```

- * 読み込み用の入力ファイルをオープンします。

```
OPEN INPUT INFILE.
```

- * レコード全体または最初のピースを書き込みます。
- * データ・ファイルを読み込み、BUFFER-ARR および BUFFER-LEN を移入します。
- * ファイル全体が読み込まれている場合、END-OF-FILE が "Y" に設定されます。

```
PERFORM READ-NEXT-RECORD.  
MOVE INREC TO BUFFER-ARR.  
MOVE 80 TO BUFFER-LEN.  
IF (END-OF-FILE = "Y")  
    MOVE 80 TO AMT  
    EXEC SQL  
        LOB WRITE ONE :AMT FROM :BUFFER  
        INTO :CLOB1 AT :OFFSET END-EXEC  
ELSE  
    DISPLAY "LOB WRITE FIRST"  
    DISPLAY BUFFER-ARR  
    MOVE 321 TO AMT  
    EXEC SQL  
        LOB WRITE FIRST :AMT FROM :BUFFER INTO :CLOB1  
    END-EXEC  
END-IF.
```

- * 入力データ・ファイルからの読み込みおよび CLOB への書き込みを続けます。

```
PERFORM READ-WRITE  
    UNTIL END-OF-FILE = "Y".  
PERFORM SIGN-OFF.  
STOP RUN.
```

```
READ-WRITE.
```

```
PERFORM READ-NEXT-RECORD.  
MOVE INREC TO BUFFER-ARR.  
DISPLAY "READ-WRITE".  
DISPLAY INREC.  
MOVE 80 TO BUFFER-LEN.  
IF END-OF-FILE = "Y"  
    DISPLAY "LOB WRITE LAST: ", BUFFER-ARR  
    MOVE 1 TO BUFFER-LEN  
    EXEC SQL  
        LOB WRITE LAST :AMT FROM :BUFFER INTO :CLOB1 END-EXEC  
ELSE  
    DISPLAY "LOB WRITE NEXT: ", BUFFER-ARR  
    MOVE 0 TO AMT  
    EXEC SQL  
        LOB WRITE NEXT :AMT FROM :BUFFER INTO :CLOB1 END-EXEC
```

```

END-IF.

READ-NEXT-RECORD.
MOVE SPACES TO INREC.
READ INFILE NEXT RECORD
AT END
MOVE "Y" TO END-OF-FILE.

SIGN-OFF.
CLOSE INFILE.
EXEC SQL FREE :CLOB1 END-EXEC.
EXEC SQL ROLLBACK WORK RELEASE END-EXEC.
STOP RUN.

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

```

C/C++ (Pro*C) : LOB のチェックイン

/* この例では、Pro*C/C++ が、単一のピースまたは複数のピースごとに任意の量のデータを内部 LOB に書き込む機能を提供する方法を示します。これには、標準のポーリングを使用するストリーミング・メカニズムが使用されます。静的なバッファを使用して、書き込まれているデータが保持されます。*/

```

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

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

#define BufferLength 512

```

```
void checkInLOB_proc(multiple) int multiple;
{
    OCIClobLocator *Lob_loc;
    VARCHAR Buffer[BufferLength];
    unsigned int Total;
    unsigned int Amount;
    unsigned int remainder, nbytes;
    boolean last;

    EXEC SQL WHENEVER SQLERROR DO Sample_Error();
    /* ロケータの割当ておよび初期化を行います。*/
    EXEC SQL ALLOCATE :Lob_loc;
    EXEC SQL SELECT Story INTO :Lob_loc
        FROM Multimedia_tab WHERE Clip_ID = 1 FOR UPDATE;
    /* LOB をオープンします。*/
    EXEC SQL LOB OPEN :Lob_loc READ WRITE;
    Total = Amount = (multiple * BufferLength);
    if (Total > BufferLength)
        nbytes = BufferLength; /* 標準のポーリングを介してストリーミングを使用します。*/
    else
        nbytes = Total; /* 単一の書き込みのみが必要です。*/
    /* バッファにnバイト相当のデータを充填します。*/
    memset((void *)Buffer.arr, 32, nbytes);
    Buffer.len = nbytes; /* 長さを設定します。*/
    remainder = Total - nbytes;
    if (0 == remainder)
    {
        /* ここで、合計がBufferLength以下であるため、1つのピースごと書き込みできます。*/
        EXEC SQL LOB WRITE ONE :Amount FROM :Buffer INTO :Lob_loc;
        printf("Write ONE Total of %d characters\n", Amount);
    }
    else
    {
        /* ここで、合計がBufferLengthより大きいため、標準のポーリングを介して
           ストリーミングを使用できます。FIRSTを指定すると、ポーリングが開始します。*/
        EXEC SQL LOB WRITE FIRST :Amount FROM :Buffer INTO :Lob_loc;
        printf("Write FIRST %d characters\n", Buffer.len);
        last = FALSE;
        /* 次 (中間) および最後のピースを書き込みます。*/
        do
        {
            if (remainder > BufferLength)
                nbytes = BufferLength; /* ピースが残っています。*/
            else
            {
                nbytes = remainder;
                last = TRUE; /* これが最後のピースになります。*/
            }
        }
    }
}
```

```
    }
    /* バッファにnバイト相当のデータを充填します。*/
    memset((void *)Buffer.arr, 32, nbytes);
    Buffer.len = nbytes;      /* 長さを設定します。*/
    if (last)
    {
        EXEC SQL WHENEVER SQLERROR DO Sample_Error();
        /* LASTを指定すると、ポーリングが終了します。*/
        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);
    }
    /* 残りの書込み量を判断します。*/
    remainder = remainder - nbytes;
} while (!last);
}

EXEC SQL WHENEVER SQLERROR DO Sample_Error();
/* この時点では、(Amount == 合計) であるため、合計量が書き込まれています。*/
/* LOBをクローズします。*/
EXEC SQL LOB CLOSE :Lob_loc;
EXEC SQL FREE :Lob_loc;
}

void main()
{
    char *samp = "samp/samp";
    EXEC SQL CONNECT :samp;
    checkInLOB_proc(1);
    EXEC SQL ROLLBACK WORK;
    checkInLOB_proc(4);
    EXEC SQL ROLLBACK WORK RELEASE;
}
```

Visual Basic (0040) : LOB のチェックイン

LOB の書込みには、`orablob.write` または `orablob.copyfromfile` を使用する 2 つの方法があります。

```
'OraBlob.Write メカニズムの使用
Dim MySession As OraSession
Dim OraDb As OraDatabase
Dim fnum As Integer
Dim OraDyn As OraDynaset, OraSound As OraBlob, amount_written%, chunksize%,
curchunk() As Byte

Set MySession = CreateObject("OracleInProcServer.XOraSession")
Set OraDb = MySession.OpenDatabase("exampledb", "samp/samp", 0&)
chunksize = 500
ReDim curchunk(chunksize)
Set OraDyn = OraDb.CreateDynaset("SELECT * FROM Multimedia_tab", ORADYN_DEFAULT)
Set OraSound = OraDyn.Fields("Sound").Value

fnum = FreeFile

Open "c:\tmp\washington_audio" For Binary As #fnum
OraSound.offset = 1
OraSound.pollingAmount = LOF(fnum)
remainder = LOF(fnum)

Dim piece As Byte
Get #fnum, , curchunk
OraDyn.Edit

piece = ORALOB_FIRST_PIECE
OraSound.Write curchunk, chunksize, ORALOB_FIRST_PIECE

While OraSound.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
    OraSound.Write curchunk, chunksize, piece
Wend

OraDyn.Update
```

```
'OraBlob.CopyFromFile メカニズムの使用
Set OraDyn = OraDb.CreateDynaset("select * from Multimedia_tab order by clip_id",
ORADYN_DEFAULT)
Set OraSound = OraDyn.Fields("Sound").Value
OraDyn.Edit
OraSound.CopyFromFile "c:\tmp\washington_audio"
OraDyn.Update
```

Java (JDBC) : LOB のチェックイン

```
import java.io.InputStream;
import java.io.OutputStream;

// コア JDBC クラス :
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 固有の JDBC クラス :
import oracle.sql.*;
import oracle.jdbc.driver.*;

public class Ex2_66
{
    static final int MAXBUFSIZE = 32767;

    public static void main (String args [])
        throws Exception
    {
        // Oracle JDBC Driver をロードします。
        DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

        // データベースに接続します。
        Connection conn =
            DriverManager.getConnection ("jdbc:oracle:oci8:@", "samp", "samp");

        // 自動コミットがオフの場合、高速になります。
        conn.setAutoCommit (false);

        // 文を作成します。
        Statement stmt = conn.createStatement ();
        try
```

```
{
    CLOB lob_loc = null;
    String buf = new String ("Some Text To Write");
    ResultSet rset = stmt.executeQuery (
        "SELECT story FROM multimedia_tab WHERE clip_id = 2 FOR UPDATE");
    if (rset.next())
    {
        lob_loc = ((OracleResultSet)rset).getCLOB (1);
    }

    long pos = 0;        // データが書き込まれる CLOB 内のオフセットです。
    long length = 0;    // 書き込むバッファのサイズです。

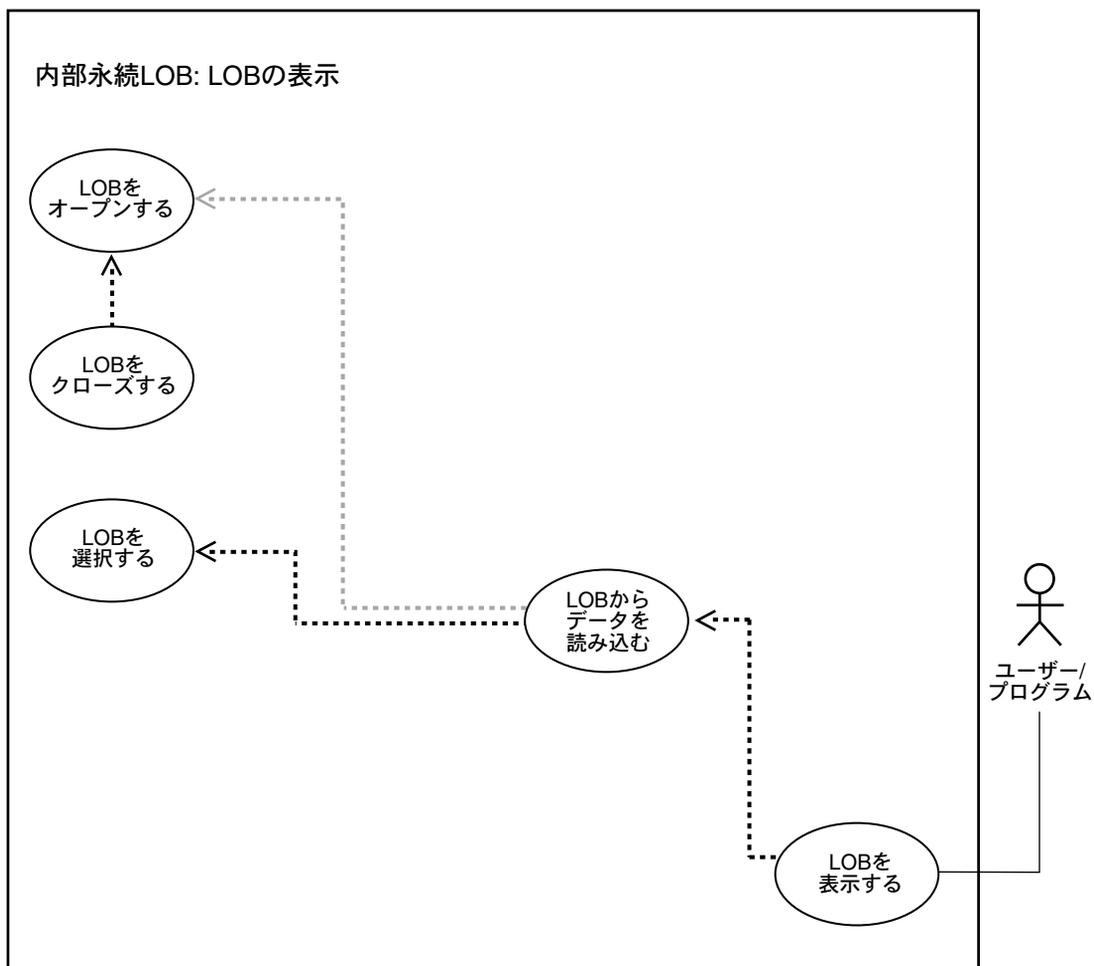
    // このループで、バッファが 3 回連続で書き込まれます。
    for (int i = 0; i < 3; i++)
    {
        pos = lob_loc.length();

        // 代替は lob_loc.putString(pos, buf) です。
        lob_loc.putString(pos, buf);

        // いくつかのデバッグ情報です。
        System.out.println(" putString(" + Long.toString(pos) + " buf);");
    }
    stmt.close();
    conn.commit();
    conn.close();
}
catch (SQLException e)
{
    e.printStackTrace();
}
}
```

LOB データの表示

図 9-21 ユースケース図 : LOB データの表示



参照: 内部永続LOBに関するすべての基本操作については、9-2ページの「ユースケース・モデル:内部永続LOB - 基本操作」を参照してください。

用途

LOB データを表示します。

使用上の注意

ストリーム・メカニズム

大量の LOB データを最も効率よく読み込む方法は、ストリーム・メカニズムを使用可能にして `OCILOBRead()` を使用することです。

構文

各プログラム環境で使用可能な機能のリストは、第 3 章「LOB プログラム環境」を参照してください。各プログラム環境における構文については、次のマニュアルの項を参照してください。

- PL/SQL (DBM_LOB パッケージ) : 『Oracle8i PL/SQL パッケージ・プロシージャ・リファレンス』の第 18 章「DBMS_LOB」の「サブプログラムの要約」の OPEN プロシージャ、READ プロシージャ、CLOSE プロシージャ
- C (OCI) : 『Oracle8i コール・インタフェース・プログラマーズ・ガイド』の第 15 章「OCI リレーショナル関数」の「LOB 関数」の `OCILOBOpen()`、`OCILOBRead()`、`OCILOBClose()`
- COBOL (Pro*COBOL) : 『Oracle8i Pro*COBOL プリコンパイラ・プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB READ (実行可能埋込み SQL 拡張要素)」
- C/C++ (Pro*C) : 『Oracle8i Pro*C/C++ プリコンパイラ・プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB READ (実行可能埋込み SQL 拡張要素)」
- Visual Basic (Oracle Objects for OLE オンライン・ヘルプ) : 「Help」の「Contents」タブから、「OO4O Automation Server」>「OBJECTS」>「OraClob」>「METHODS」>「read」、および「OBJECTS」>「OraClob」>「PROPERTIES」>「offset」を選択してください。
- Java (JDBC) : 『Oracle8i JDBC 開発者ガイドおよびリファレンス』の第 7 章「LOB と BFILE の操作」の「BLOB と CLOB の操作」の「BLOB または CLOB 列の作成と移入」、および『Oracle8i SQLJ 開発者ガイドおよびリファレンス』の第 5 章「型のサポート」の「Oracle の拡張型」の「BLOB、CLOB および BFILE のサポート」

使用例

LOB を表示する例として、この使用例では列オブジェクト `Map_obj` から、イメージ `Drawing` をクライアント側にストリーム読み込みし、データを表示します。

例

次のプログラム環境の例が示されています。

- PL/SQL (DBMS_LOB パッケージ) : LOB データの表示 (9-91 ページ)
- C (OCI) : LOB データの表示 (9-92 ページ)
- COBOL (Pro*COBOL) : LOB データの表示 (9-94 ページ)
- C/C++ (Pro*C) : LOB データの表示 (9-95 ページ)
- Visual Basic (OO4O) : LOB データの表示 (9-97 ページ)
- Java (JDBC) : LOB データの表示 (9-97 ページ)

PL/SQL (DBMS_LOB パッケージ) : LOB データの表示

```
/* 例のプロシージャ displayLOB_proc は、DBMS_LOB パッケージの
一部ではないことに注意してください。*/
CREATE OR REPLACE PROCEDURE displayLOB_proc IS
Lob_loc  BLOB;
Buffer   RAW(1024);
Amount   BINARY_INTEGER := 1024;
Position INTEGER := 1;
BEGIN
  /* LOB を選択します。*/
  SELECT m.Map_obj.Drawing INTO Lob_loc
  FROM Multimedia_tab m WHERE m.Clip_ID = 1;
  /* LOB のオープンはオプションです。*/
  DBMS_LOB.OPEN (Lob_loc, DBMS_LOB.LOB_READONLY);
  LOOP
    DBMS_LOB.READ (Lob_loc, Amount, Position, Buffer);
    /* バッファの内容を表示します。*/
    DBMS_OUTPUT.PUT_LINE(utl_raw.cast_to_varchar2(Buffer));
    Position := Position + Amount;
  END LOOP;
  /* LOB をオープンしている場合、その LOB をクローズする必要があります。*/
  DBMS_LOB.CLOSE (Lob_loc);
  EXCEPTION
    WHEN NO_DATA_FOUND THEN
      DBMS_OUTPUT.PUT_LINE('End of data');
END;
```

C (OCI) : LOB データの表示

/* この例では、標準のポーリング方法を使用して、BLOB の内容全体がピース単位でバッファに読み込まれます。このポーリング方法では、各バッファのピースが、BLOB 全体が読み込まれるまで、すべての READ 操作後に処理されます。*/

```
#define MAXBUFLLEN 32767
/* ロケータ変数にロケータを選択します。*/
sb4 select_mapobjectdrawing_locator(Lob_loc, errhp, svchp, stmthp)
OCILobLocator *Lob_loc;
OCIError      *errhp;
OCISvcCtx     *svchp;
OCIStmt       *stmthp;
{
    OCIDefine *defnp1;
    text      *sqlstmt =
        (text *) "SELECT m.Map_obj.Drawing \
                FROM Multimedia_tab m WHERE m.Clip_ID = 1";
    checkerr (errhp, OCIStmtPrepare(stmthp, errhp, sqlstmt,
                                    (ub4)strlen((char *)sqlstmt),
                                    (ub4) OCI_NTV_SYNTAX, (ub4) OCI_DEFAULT));
    checkerr (errhp, OCIDefineByPos(stmthp, &defnp1, errhp, (ub4) 1,
                                    (dvoid *)&Lob_loc, (sb4) 0,
                                    (ub2) SQLT_BLOB, (dvoid *) 0,
                                    (ub2 *) 0, (ub2 *) 0, (ub4) OCI_DEFAULT));

    /* SELECT を実行し、1 行フェッチします。*/
    checkerr (errhp, OCIStmtExecute (svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                                    (CONST OCISnapshot*) 0, (OCISnapshot*) 0,
                                    (ub4) OCI_DEFAULT));

    return 0;
}

void displayLob(envhp, errhp, svchp, stmthp)
OCIEnv      *envhp;
OCIError    *errhp;
OCISvcCtx   *svchp;
OCIStmt     *stmthp;
{
    OCIBlobLocator *Lob_loc;
    ub4 amt;
    ub4 offset;
    sword retval;
    boolean done;
    ub1 bufp[MAXBUFLLEN];
    ub4 buflen;
    OCILobLocator *Lob_Loc;
```

```
/* ソース (BFILE) および宛先 (BLOB) ロケータの記述子を割り当てます。*/
(void) OCIDescriptorAlloc((dvoid *) envhp,
                          (dvoid **) &Lob_loc, (ub4)OCI_DTYPE_LOB,
                          (size_t) 0, (dvoid **) 0);

/* BLOB を選択します。*/
printf(" select the mapobjectdrawing locator...\n");
select_mapobjectdrawing_locator(Lob_loc, errhp, svchp, stmthp);

/* BLOB をオープンします。*/
printf(" open the lob\n");
checkerr (errhp, (OCILobOpen(svchp, errhp, Lob_loc, OCI_LOB_READONLY)));

/* amt = 0 (ゼロ) に設定すると、LOB の終わりまで読み込まれます。*/
amt = 0;
buflen = sizeof(bufp);

/* データをピースごとに処理します。*/
printf(" Process the data in pieces\n");
offset = 1;
memset(bufp, '\0', MAXBUFLLEN);
done = FALSE;
while (!done)
{
    retval = OCILobRead(svchp, errhp, Lob_loc, &amt, offset, (dvoid *) bufp,
                       buflen, (dvoid *)0,
                       (sb4 *) (dvoid *, dvoid *, ub4, ub1)) 0,
                       (ub2) 0, (ub1) SQLCS_IMPLICIT);

    switch (retval)
    {
    case OCI_SUCCESS: /* 1 つのピースか、または最後のピースのみです。*/
        /* bufp のデータを処理します。amt には、bufp に直前に読み込まれたデータの量が示され
           ます。これは、BLOB ではバイト、固定幅 CLOB では文字、および可変幅 CLOB ではバイト
           単位です。*/
        done = TRUE;
        break;
    case OCI_ERROR:
        checkerr (errhp, retval);
        done = TRUE;
        break;
    case OCI_NEED_DATA: /* 2 つ以上のピースがあります。*/
        /* bufp のデータを処理します。amt には、bufp に直前に読み込まれたデータの量が示され
           ます。これは、BLOB ではバイト、固定幅 CLOB では文字、および可変幅 CLOB ではバイト
           単位です。*/
        /*
        break;
    default:
```

```
        checkerr (errhp, retval);
        done = TRUE;
        break;
    }
} /* while */

/* BLOB をオープンしている場合、BLOB をクローズする必要があります。*/
printf(" close the lob \n");
checkerr (errhp, OCILobClose(svchp, errhp, Lob_loc));

/* ロケータが保持しているリソースを解放します。*/
(void) OCIDescriptorFree((dvoid *) Lob_loc, (ub4) OCI_DTYPE_LOB);

}
```

COBOL (Pro*COBOL) : LOB データの表示

```
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.
    * BLOB ロケータの割当ておよび初期化を行います。
    EXEC SQL ALLOCATE :BLOB1 END-EXEC.
    EXEC SQL WHENEVER NOT FOUND GOTO END-OF-BLOB END-EXEC.
    EXEC SQL SELECT M.SOUND INTO :BLOB1
        FROM MULTIMEDIA_TAB M WHERE M.CLIP_ID = 1 END-EXEC.
    DISPLAY "Found column SOUND".
```

* ポーリング読み込みを開始します。

```

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) : LOB データの表示

/* この例では、標準のポーリング方法を使用して、BLOB の内容全体がピース単位でバッファに読み込まれます。このポーリング方法では、各バッファのピースが、BLOB 全体が読み込まれるまで、すべての READ 操作後に処理されます。*/

```

#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;
    /* このデータ型では、データ型を等価にする必要があります。*/
    EXEC SQL VAR Buffer IS VARRAW(BufferLength);

    EXEC SQL WHENEVER SQLERROR DO Sample_Error();
    EXEC SQL ALLOCATE :Lob_loc;
    /* BLOB を選択します。*/
    EXEC SQL SELECT m.Map_obj.Drawing INTO Lob_loc
        FROM Multimedia_tab m WHERE m.Clip_ID = 1;
    /* BLOB をオープンします。*/
    EXEC SQL LOB OPEN :Lob_loc READ ONLY;
    /* Amount = 0 (ゼロ) に設定すると、ポーリング方法が開始します。*/
    Amount = 0;
    /* バッファの最大サイズを設定します。*/
    Buffer.Length = BufferLength;
    EXEC SQL WHENEVER NOT FOUND DO break;
    while (TRUE)
    {
        /* バッファに1つのピースのBLOBを読み込みます。*/
        EXEC SQL LOB READ :Amount FROM :Lob_loc INTO :Buffer;
        /* (Buffer.Length == BufferLength) の量の Buffer.Data を処理します。*/
    }
    /* (Buffer.Length == Amount) の量の Buffer.Data を処理します。*/
    /* BLOB をオープンしている場合、そのBLOBをクローズする必要があります。*/
    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 (0040) : LOB データの表示

```
'OraClob.Read メカニズムの使用  
Dim MySession As OraSession  
Dim OraDb As OraDatabase  
  
Set MySession = CreateObject("OracleInProcServer.XOraSession")  
Set OraDb = MySession.OpenDatabase("exampleldb", "samp/samp", 0&)  
Dim OraDyn as OraDynaset, OraStory as OraClob, amount_read%, chunksize%, chunk  
chunksize = 32767  
Set OraDyn = OraDb.CreateDynaset("SELECT * FROM Multimedia_tab", ORADYN_DEFAULT)  
Set OraStory = OraDyn.Fields("Story").Value  
OraStory.PollingAmount = OraStory.Size 'Read entire CLOB contents  
Do  
    '戻されたチャンクは、可変の型バイト配列です。  
    amount_read = OraStory.Read(chunk, chunksize)  
    'Msgbox チャンク  
Loop Until OraStory.Status <> ORALOB_NEED_DATA
```

Java (JDBC) : LOB データの表示

```
// コア JDBC クラス :  
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 固有の JDBC クラス :  
import oracle.sql.*;  
import oracle.jdbc.driver.*;  
public class Ex2_72  
{  
    static final int MAXBUFSIZE = 32767;  
    public static void main (String args [])  
        throws Exception
```

```
{
// Oracle JDBC Driver をロードします。
DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

// データベースに接続します。
Connection conn =
    DriverManager.getConnection ("jdbc:oracle:oci8:@", "samp", "samp");

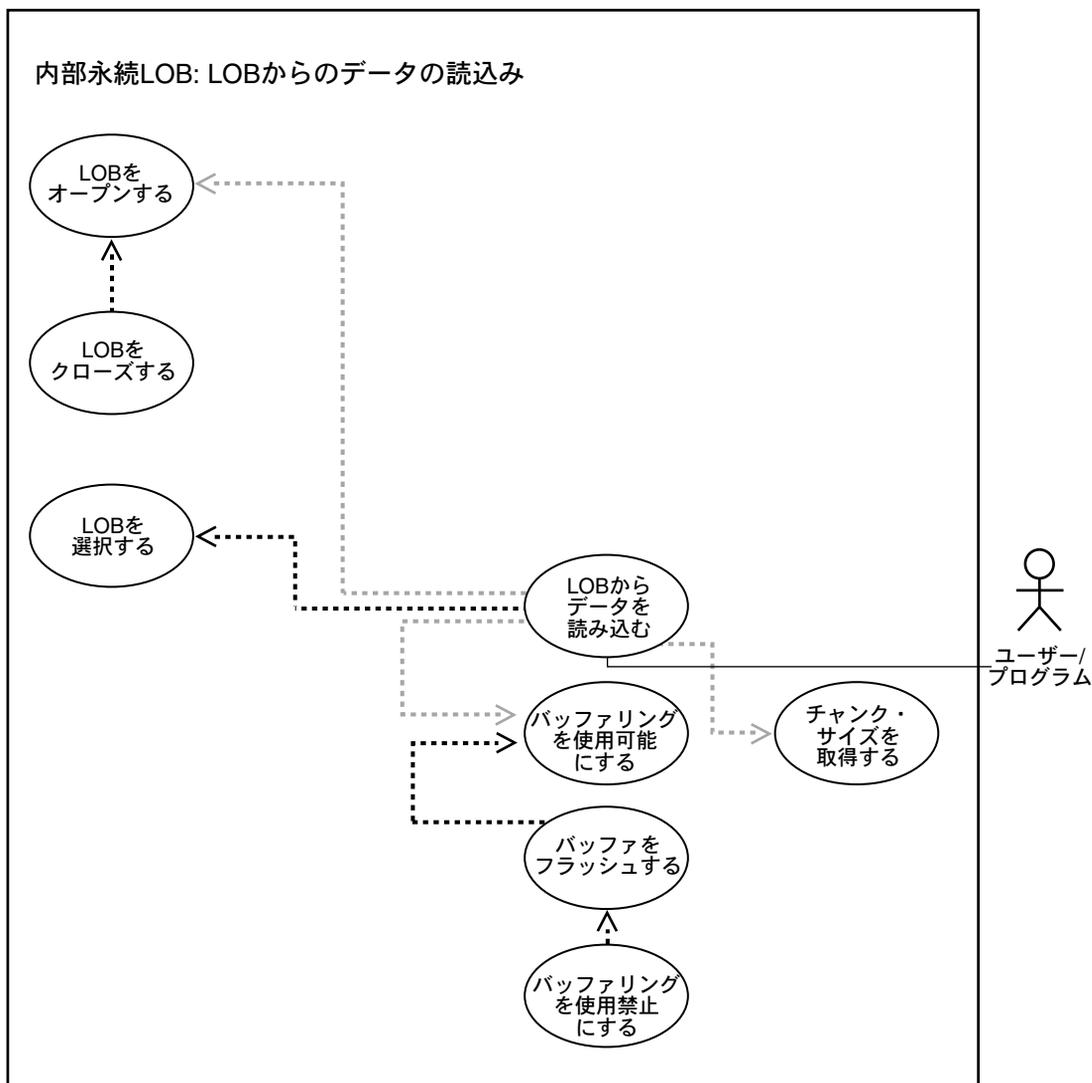
// 自動コミットがオフの場合、高速になります。
conn.setAutoCommit (false);

// 文を作成します。
Statement stmt = conn.createStatement ();
try
{
    BLOB lob_loc = null;
    InputStream in = null;
    byte buf[] = new byte[MAXBUFSIZE];
    int pos = 0;
    int length = 0;
    ResultSet rset = stmt.executeQuery (
        "SELECT m.map_obj.drawing FROM multimedia_tab m WHERE m.clip_id = 1");
    if (rset.next())
    {
        lob_loc = ((OracleResultSet)rset).getBLOB (1);
    }

// InputStream を介してこの LOB を読み込みます。
in = lob_loc.getBinaryStream();
while ((length = in.read(buf)) != -1)
{
    pos += length;
    System.out.println(Integer.toString(pos));
    // バッファの内容を処理します。
}
    in.close();
    stmt.close();
    conn.commit();
    conn.close();
}
catch (SQLException e)
{
    e.printStackTrace();
}
}
```

LOB からのデータの読み込み

図 9-22 ユースケース図：LOB からのデータの読み込み



参照： 内部永続 LOB に関するすべての基本操作については、9-2 ページの「ユースケース・モデル:内部永続 LOB - 基本操作」を参照してください。

用途

LOB からデータを読み込みます。

使用上の注意

ストリーム読み込み

大量の LOB データを最も効率よく読み込む方法は、ポーリングまたはコールバックを介してストリーム・メカニズムを使用可能にし、OCILOBRead() を使用することです。

LOB 値を読み込むとき、LOB の最後を超えて読み込んでもエラーにはなりません。開始オフセットおよび LOB のデータ量にかかわらず、通常 4GB -1 の入力を指定できます。読み込む量を決定するために、OCILOBGetLength() コールを繰り返し、LOB 値の長さを判断する必要はありません。

例

LOB の長さが 5,000 バイトで、オフセット 1,000 から開始して LOB 値全体を読み込むとします。また、LOB 値の現在の長さがわからないとします。すべてのパラメータの初期化を除いた OCI 読み込みコールを次に示します。

```
#define MAX_LOB_SIZE 4294967295
ub4 amount = MAX_LOB_SIZE;
ub4 offset = 1000;
OCILOBRead(svchp, errhp, lncp, &amount, offset, bufp, buflen, 0, 0, 0, 0)
```

注意：

- DBMS_LOB.READ では、読み込む量がデータのサイズより大きくてもかまいません。PL/SQL では、読み込む量がバッファ・サイズ以下である必要があります。また、バッファ・サイズは 32KB 以下である必要があります。
 - OCILOBRead では、量パラメータを 4GB -1 に指定できます。また、LOB の最後まで読み込みます。
-
-

- ポーリング・モードを使用するときは、バッファが一杯にならない場合があるため、各 `OCILOBRead()` コール後に量パラメータの値を調べて、バッファに何バイト読み込まれたかを確認してください。
- コールバックを使用する場合、コールバックへの入力である `len` パラメータにバッファに格納されたバイト数が示されます。バッファ全体にデータが格納されていない場合があるため、コールバック処理中に `len` パラメータを確認してください（『Oracle8i コール・インタフェース・プログラマーズ・ガイド』を参照）。

チャンク・サイズ

チャンクとは、1つまたは複数の Oracle ブロックです。LOB を含む表を作成する場合に、LOB 用のチャンク・サイズを指定できます。これは、LOB 値に対してアクセスまたは変更を行うときに、Oracle が使用するチャンク・サイズに対応します。チャンクの一部はシステム関連の情報を格納し、残りは LOB 値を格納します。`getchunksize` 関数は、LOB チャンク内で使用され、LOB 値を格納している領域の量を戻します。

このチャンク・サイズを複数回使用して `read` 要求を実行すると、パフォーマンスが向上します。これは、データをディスクから読み込むとき、Oracle データベースが使用しているサイズと同じ単位を使用することになるためです。アプリケーションに問題がなければ、同じ LOB チャンクに複数回の LOB 読み込みコールを発行するのではなく、全体のチャンクを取得するために十分なサイズでバッチ読み込みを行うとパフォーマンスが向上します。

構文

各プログラム環境で使用可能な機能のリストは、第3章「LOB プログラム環境」を参照してください。各プログラム環境における構文については、次のマニュアルの項を参照してください。

- PL/SQL (DBM_LOB パッケージ) : 『Oracle8i PL/SQL パッケージ・プロシージャ・リファレンス』の第18章「DBMS_LOB」の「サブプログラムの要約」の OPEN プロシージャ、GETCHUNKSIZE ファンクション、READ プロシージャ、CLOSE プロシージャ
- C (OCI) : 『Oracle8i コール・インタフェース・プログラマーズ・ガイド』の第15章「OCI リレーショナル関数」の「LOB 関数」の `OCILOBOpen()`、`OCILOBRead()`、`OCILOBClose()`
- COBOL (Pro*COBOL) : 『Oracle8i Pro*COBOL プリコンパイラ・プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB READ (実行可能埋込み SQL 拡張要素)」
- C/C++ (Pro*C) : 『Oracle8i Pro*C/C++ プリコンパイラ・プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB READ (実行可能埋込み SQL 拡張要素)」

- Visual Basic (Oracle Objects for OLE オンライン・ヘルプ) : 「Help」の「Contents」タブから、「OO4O Automation Server」>「OBJECTS」>「Oraclob」>「METHODS」>「read」を選択してください。
- Java (JDBC) : 『Oracle8i JDBC 開発者ガイドおよびリファレンス』の第7章「LOB と BFILE の操作」の「BLOB と CLOB の操作」の「BLOB または CLOB 列の作成と移入」、および『Oracle8i SQLJ 開発者ガイドおよびリファレンス』の第5章「型のサポート」の「Oracle の拡張型」の「BLOB、CLOB および BFILE のサポート」

使用例

次の例では、データを1つのビデオ・フレームから読み込みます。

例

次のプログラム環境の例が示されています。

- [PL/SQL \(DBMS_LOB パッケージ\) : LOB からのデータの読み込み \(9-102 ページ\)](#)
- [C \(OCI\) : LOB からのデータの読み込み \(9-103 ページ\)](#)
- [COBOL \(Pro*COBOL\) : LOB からのデータの読み込み \(9-105 ページ\)](#)
- [C \(OCI\) : LOB からのデータの読み込み \(9-103 ページ\)](#)
- [Visual Basic \(OO4O\) : LOB からのデータの読み込み \(9-108 ページ\)](#)
- [Java \(JDBC\) : LOB からのデータの読み込み \(9-108 ページ\)](#)

PL/SQL (DBMS_LOB パッケージ) : LOB からのデータの読み込み

/* 例のプロシージャ readLOB_proc は、DBMS_LOB パッケージの一部ではないことに注意してください。*/

```
CREATE OR REPLACE PROCEDURE readLOB_proc IS
  Lob_loc          BLOB;
  Buffer            RAW(32767);
  Amount           BINARY_INTEGER := 32767;
  Position         INTEGER := 1000;
  Chunksize       INTEGER;
BEGIN
  /* LOB を選択します。*/
  SELECT Frame INTO Lob_loc
  FROM Multimedia_tab
  WHERE Clip_ID = 1;
```

```

/* このLOB列のチャンクサイズを調べます。*/
Chunksize := DBMS_LOB.GETCHUNKSIZE(Lob_loc);
IF (Chunksize < 32767) THEN
    Amount := (32767 / Chunksize) * Chunksize;
END IF;
/* LOBのオープンはオプションです。*/
DBMS_LOB.OPEN (Lob_loc, DBMS_LOB.LOB_READONLY);
/* LOBからデータを読み込みます。*/
DBMS_LOB.READ (Lob_loc, Amount, Position, Buffer);
/* LOBをオープンしている場合、そのLOBをクローズする必要があります。*/
DBMS_LOB.CLOSE (Lob_loc);
END;

```

C (OCI) : LOB からのデータの読み込み

/* この例では、標準のポーリング方法を使用して、BLOBの内容全体がピース単位でバッファに読み込まれます。このポーリング方法では、各バッファのピースが、BLOB全体が読み込まれるまで、すべてのREAD操作後に処理されます。*/

```
#define MAXBUFLLEN 1000
```

```

/* ロケータ変数にロケータを選択します。*/
sb4 select_frame_locator(Lob_loc, errhp, svchp, stmthp)
OCILobLocator *Lob_loc;
OCIError      *errhp;
OCISvcCtx     *svchp;
OCIStmt      *stmthp;
{
    OCIDefine *defnp1;

    text *sqlstmt =
        (text *) "SELECT Frame \
                FROM Multimedia_tab m WHERE m.Clip_ID = 3";

    printf(" prepare statement in select_frame_locator\n");
    checkerr (errhp, OCIStmtPrepare(stmthp, errhp, sqlstmt,
                                    (ub4)strlen((char *) sqlstmt),
                                    (ub4) OCI_NTV_SYNTAX, (ub4) OCI_DEFAULT));
    printf(" OCIDefineByPos in select_frame_locator\n");
    checkerr (errhp, OCIDefineByPos(stmthp, &defnp1, errhp, (ub4) 1,
                                    (dvoid *)&Lob_loc, (sb4) 0,
                                    (ub2) SQLT_BLOB, (dvoid *) 0,
                                    (ub2 *) 0, (ub2 *) 0, (ub4) OCI_DEFAULT));
    /* SELECTを実行し、1行フェッチします。*/
    printf(" OCIStmtExecute in select_frame_locator\n");

```

```
        checkerr(errhp, OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                                     (CONST OCISnapshot*) 0, (OCISnapshot*) 0,
                                     (ub4) OCI_DEFAULT));
    }
    return 0;
}

void readLOB_proc(envhp, errhp, svchp, stmthp)
OCIEnv      *envhp;
OCIError    *errhp;
OCISvcCtx   *svchp;
OCIStmt     *stmthp;
{
    ub4 amt;
    ub4 offset;
    sword retval;
    ub1 bufp[MAXBUFLen];
    ub4 buflen;
    boolean done;

    OCILobLocator *Lob_loc;
    OCILobLocator *blob;

    /* ソース (BFILE) および宛先 (BLOB) ロケータの記述子を割り当てます。*/
    (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &Lob_loc,
                              (ub4)OCI_DTYPE_LOB, (size_t) 0, (dvoid **) 0);
    /* BLOB を選択します。*/
    printf(" call select_frame4read_locator\n");
    select_frame_locator(Lob_loc, errhp, svchp, stmthp);

    /* BLOB をオープンします。*/
    printf(" call OCILobOpen\n");
    checkerr (errhp, OCILobOpen(svchp, errhp, Lob_loc, OCI_LOB_READONLY));

    /* amt をバッファの長さに設定します。ここで amt は、BLOB を使用しているためバイト単位であることに注意してください。*/
    amt = 0;
    buflen = sizeof(bufp);

    /* データをピースごとに処理します。*/
    printf(" process the data in piece\n");
    offset = 1;
    memset(bufp, '\0', MAXBUFLen);
    done = FALSE;

    while (!done)
```

```

{
retval = OCILobRead(svchp, errhp, Lob_loc, &amt, offset, (dvoid *) bufp,
                  buflen, (dvoid *)0,
                  (sb4 *) (dvoid *, dvoid *, ub4, ub1)) 0,
                  (ub2) 0, (ub1) SQLCS_IMPLICIT);

switch (retval)
{
case OCI_SUCCESS:          /* amtp == bufp のため、1 ピースのみ */
/* bufp のデータを処理します。amt には、bufp に直前に読み込まれたデータの量が示され
ます。これは、BLOB ではバイト、固定幅 CLOB では文字、および可変幅 CLOB ではバイト
単位です。*/
printf("[%.*s]\n", buflen, bufp);
done = TRUE;
break;
case OCI_ERROR:
/* report_error();        ここでは、このファンクションは示しません。*/
done = TRUE;
break;
case OCI_NEED_DATA:
printf("[%.*s]\n", buflen, bufp);
break;
default:
(void) printf("Unexpected ERROR: OCILobRead() LOB.\n");
done = TRUE;
break;
}
}
/* BLOB をオープンしている場合、その BLOB をクローズする必要があります。*/
checkerr (errhp, OCILobClose(svchp, errhp, Lob_loc));

/* ロケータが保持しているリソースを解放します。*/
(void) OCIDescriptorFree((dvoid *) Lob_loc, (ub4) OCI_DTYPE_LOB);
}

```

COBOL (Pro*COBOL) : LOB からのデータの読み込み

```

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 "SAMP/SAMP".
      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.

* CLOB ロケータの割当ておよび初期化を行います。
      EXEC SQL ALLOCATE :BLOB1 END-EXEC.
      EXEC SQL WHENEVER NOT FOUND GOTO END-OF-BLOB END-EXEC.
      EXEC SQL
          SELECT FRAME INTO :BLOB1
          FROM MULTIMEDIA_TAB M WHERE M.CLIP_ID = 1 END-EXEC.
      EXEC SQL LOB OPEN :BLOB1 END-EXEC.

* 単一の読み込みを実行します。
      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++) : LOB からのデータの読み込み

```
#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;
    /* ここで Amount は BufferLength と等しいため、読み込みが 1 回のみ必要です。 */
    char Buffer[BufferLength];
    /* このデータ型では、データ型を等価にする必要があります。 */
    EXEC SQL VAR Buffer IS RAW(BufferLength);

    EXEC SQL WHENEVER SQLERROR DO Sample_Error();
    EXEC SQL ALLOCATE :Lob_loc;
    EXEC SQL SELECT Frame INTO :Lob_loc
        FROM Multimedia_tab WHERE Clip_ID = 1;
    /* BLOB をオープンします。 */
    EXEC SQL LOB OPEN :Lob_loc READ ONLY;
    EXEC SQL WHENEVER NOT FOUND CONTINUE;
    /* バッファに BLOB データを読み込みます。 */
    EXEC SQL LOB READ :Amount FROM :Lob_loc INTO :Buffer;
    printf("Read %d bytes\n", Amount);
    /* BLOB をクローズします。 */
    EXEC SQL LOB CLOSE :Lob_loc;
    EXEC SQL FREE :Lob_loc;
}

void main()
{
    char *samp = "samp/samp";
    EXEC SQL CONNECT :samp;
    readLOB_proc();
    EXEC SQL ROLLBACK WORK RELEASE;
}
```

Visual Basic (0040) : LOB からのデータの読み込み

```
'OraClob.Read メカニズムの使用
Dim MySession As OraSession
Dim OraDb As OraDatabase

Set MySession = CreateObject("OracleInProcServer.XOraSession")
Set OraDb = MySession.OpenDatabase("examplepdb", "samp/samp", 0&)
Dim OraDyn as OraDynaset, OraStory as OraClob, amount_read%, chunksize%, chunk

chunksize = 32767
Set OraDyn = OraDb.CreateDynaset("SELECT * FROM Multimedia_tab", ORADYN_DEFAULT)
Set OraStory = OraDyn.Fields("Story").Value
OraStory.pollingAmount = OraStory.Size
' CLOB 全体の内容を読み込みます。
Do
amount_read = OraStory.Read(chunk, chunksize)
' 戻されたチャンクは、可変の型バイト配列です。
Loop Until OraStory.Status <> ORALOB_NEED_DATA
```

Java (JDBC) : LOB からのデータの読み込み

```
// Java IO クラス :
import java.io.InputStream;
import java.io.OutputStream;

// コア JDBC クラス :
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 固有の JDBC クラス :
import oracle.sql.*;
import oracle.jdbc.driver.*;
public class Ex2_79
{
    static final int MAXBUFSIZE = 32767;
    public static void main (String args [])
        throws Exception
    {
        // Oracle JDBC Driver をロードします。
        DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
```

```
// データベースに接続します。
Connection conn =
    DriverManager.getConnection ("jdbc:oracle:oci8:@", "samp", "samp");

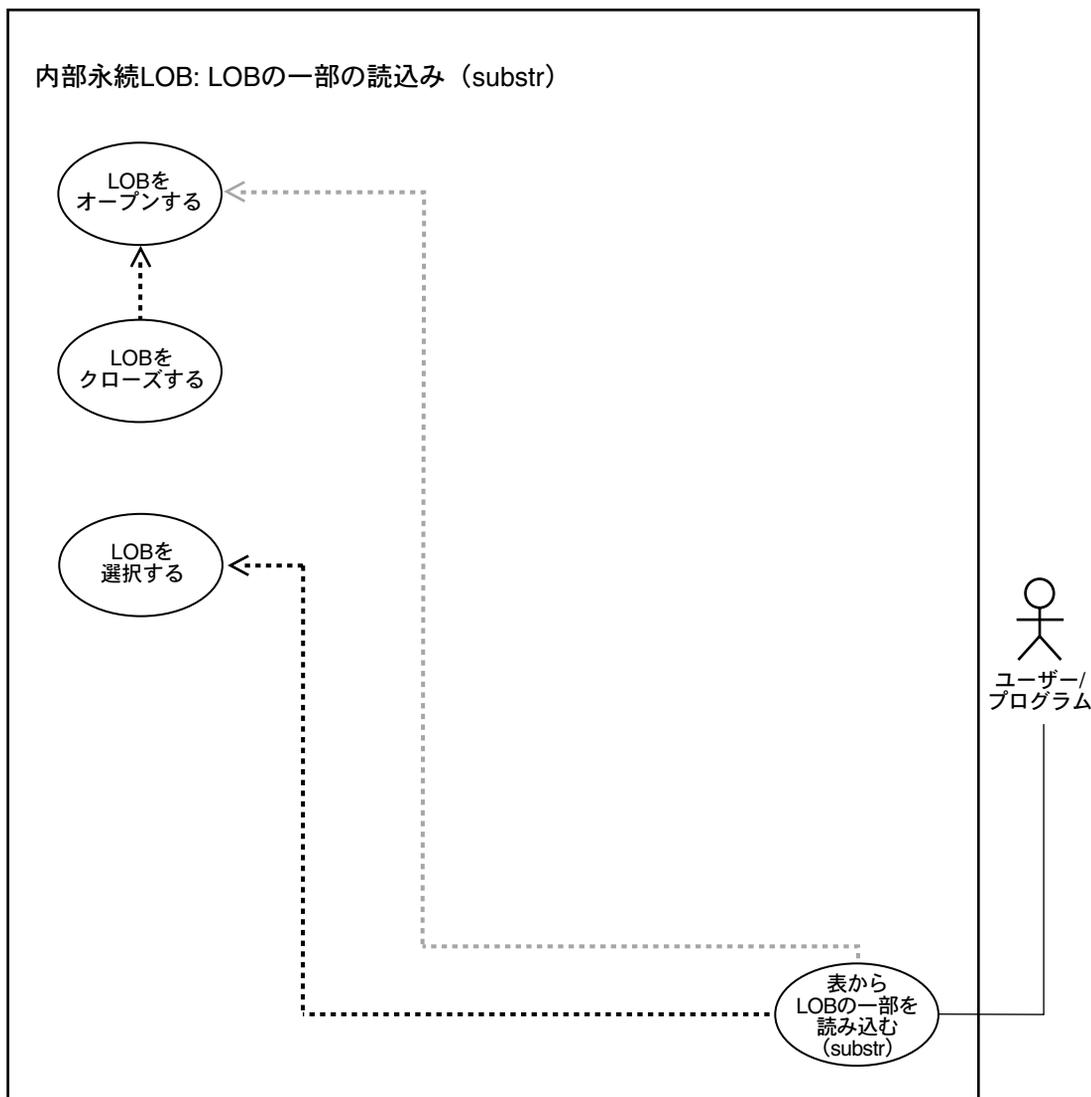
// 自動コミットがオフの場合、高速になります。
conn.setAutoCommit (false);

// 文を作成します。
Statement stmt = conn.createStatement ();
try
{
    BLOB lob_loc = null;
    byte buf[] = new byte [MAXBUFSIZE];
    ResultSet rset = stmt.executeQuery (
        "SELECT frame FROM multimedia_tab WHERE clip_id = 1");
    if (rset.next())
    {
        lob_loc = ((OracleResultSet)rset).getBLOB (1);
    }
    // MAXBUFSIZE は読み込むバイト数であり、1,000 は読み込みを開始するオフセットです。
    buf = lob_loc.getBytes(1000, MAXBUFSIZE);

    // バッファの内容を表示します。
    System.out.println(new String(buf));
    stmt.close();
    conn.commit();
    conn.close();
}
catch (SQLException e)
{
    e.printStackTrace();
}
}
```

LOB の一部の読み込み (substr)

図 9-23 ユースケース図 : LOB の一部の読み込み (substr)



参照： 内部永続 LOB に関するすべての基本操作については、9-2 ページの「[ユースケース・モデル:内部永続 LOB – 基本操作](#)」を参照してください。

用途

LOB の一部を読み込みます (substr)。

使用上の注意

ありません。

構文

各プログラム環境で使用可能な機能のリストは、[第 3 章「LOB プログラム環境](#)」を参照してください。各プログラム環境における構文については、次のマニュアルの項を参照してください。

- PL/SQL (DBM_LOB パッケージ) : 『Oracle8i PL/SQL パッケージ・プロシージャ・リファレンス』の第 18 章「DBMS_LOB」の「サブプログラムの要約」の SUBSTR ファンクション、OPEN プロシージャ、CLOSE プロシージャ
- C (OCI) : 参照マニュアルはありません。
- COBOL (Pro*COBOL) : 『Oracle8i Pro*COBOL プリコンパイラ・プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「ALLOCATE 実行可能埋込み SQL 拡張要素」、「LOB READ (実行可能埋込み SQL 拡張要素)」、「LOB CLOSE (実行可能埋込み SQL 拡張要素)」
- C/C++ (Pro*C/C++) : 『Oracle8i Pro*C/C++ プリコンパイラ・プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB READ (実行可能埋込み SQL 拡張要素)」、および『Oracle8i PL/SQL パッケージ・プロシージャ・リファレンス』の第 18 章「DBMS_LOB」の「サブプログラムの要約」の SUBSTR ファンクション
- Visual Basic (Oracle Objects for OLE オンライン・ヘルプ) : 「Help」の「Contents」タブから、「OO4O Automation Server」>「OBJECTS」>「Orablob」>「PROPERTIES」>「offset」、「chunksize」、および「OBJECTS」>「Oraclob」>「METHODS」>「read」を選択してください。
- Java (JDBC) : 『Oracle8i JDBC 開発者ガイドおよびリファレンス』の第 7 章「LOB と BFILE の操作」の「BLOB と CLOB の操作」の「BLOB または CLOB 列の作成と移入」、および『Oracle8i SQLJ 開発者ガイドおよびリファレンス』の第 5 章「型のサポート」の「Oracle の拡張型」の「BLOB、CLOB および BFILE のサポート」

使用例

次の例では、音響効果 Sound から一部を読み込みます。

例

次のプログラム環境の例が示されています。

- [PL/SQL \(DBMS_LOB パッケージ\) : LOB の一部の読み込み \(substr\) \(9-112 ページ\)](#)
- [C \(OCI\) : 今回のリリースでは例は記載されていません。](#)
- [COBOL \(Pro*COBOL\) : LOB の一部の読み込み \(substr\) \(9-113 ページ\)](#)
- [C/C++ \(Pro*C/C++\) : LOB の一部の読み込み \(substr\) \(9-114 ページ\)](#)
- [Visual Basic \(OO4O\) : LOB の一部の読み込み \(substr\) \(9-115 ページ\)](#)
- [Java \(JDBC\) : LOB の一部の読み込み \(substr\) \(9-116 ページ\)](#)

PL/SQL (DBMS_LOB パッケージ) : LOB の一部の読み込み (substr)

```
/* 例のプロシージャ substringLOB_proc は、DBMS_LOB パッケージの
一部ではないことに注意してください。 */
CREATE OR REPLACE PROCEDURE substringLOB_proc IS
    Lob_loc          BLOB;
    Amount           BINARY_INTEGER := 32767;
    Position         INTEGER := 1024;
    Buffer            RAW(32767);
BEGIN
    /* LOB を選択します。 */
    SELECT Sound INTO Lob_loc FROM Multimedia_tab
        WHERE Clip_ID = 1;
    /* LOB のオープンはおプションです。 */
    DBMS_LOB.OPEN (Lob_loc, DBMS_LOB.LOB_READONLY);
    Buffer := DBMS_LOB.SUBSTR(Lob_loc, Amount, Position);
    /* データを処理します。 */
    /* LOB をオープンしている場合、その LOB をクローズする必要があります。 */
    DBMS_LOB.CLOSE (Lob_loc);
END;

/* 次の SQL 文では、255 は読み込む量であり、1 は読み込みを開始するオフセットです。 */
SELECT DBMS_LOB.SUBSTR(Sound, 255, 1) FROM Multimedia_tab WHERE Clip_ID = 1;
```

COBOL (Pro*COBOL) : LOB の一部の読み込み (substr)

```

IDENTIFICATION DIVISION.
PROGRAM-ID. BLOB-SUBSTR.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

01 BLOB1          SQL-BLOB.
01 BUFFER2       PIC X(32767) VARYING.
01 AMT           PIC S9(9) COMP.
01 POS           PIC S9(9) COMP VALUE 1.
01 USERID       PIC X(11) VALUES "SAMP/SAMP".
EXEC SQL INCLUDE SQLCA END-EXEC.
EXEC SQL VAR BUFFER2 IS VARRAW(32767) END-EXEC.

PROCEDURE DIVISION.
BLOB-SUBSTR.
EXEC SQL WHENEVER SQLEERROR DO PERFORM SQL-ERROR END-EXEC.
EXEC SQL
CONNECT :USERID
END-EXEC.

* CLOB ロケータの割当ておよび初期化を行います。
EXEC SQL ALLOCATE :BLOB1 END-EXEC.

EXEC SQL WHENEVER NOT FOUND GOTO END-OF-BLOB END-EXEC.
EXEC SQL
SELECT FRAME INTO :BLOB1
FROM MULTIMEDIA_TAB M WHERE M.CLIP_ID = 1 END-EXEC.
DISPLAY "Selected the BLOB".

* READ ONLY 用の BLOB をオープンします。
EXEC SQL LOB OPEN :BLOB1 READ ONLY END-EXEC.

* PL/SQL を実行して、SUBSTR 機能を取得します。
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++) : LOB の一部の読み込み (substr)

/* Pro*C/C++ には、DEMS_LOB.SUBSTR() ファンクションに対する等価の埋込み SQL フォームがありません。ただし、Pro*C/C++ は、この例に示されているように、Pro*C/C++ プログラムに埋め込まれている無名 PL/SQL ブロックを使用して、PL/SQL と相互作用できます。*/

```
#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;
    /* このデータ型では、データ型を等価にする必要があります。*/
    EXEC SQL VAR Buffer IS VARRAW(BufferLength);

    EXEC SQL WHENEVER SQLERROR DO Sample_Error();
    EXEC SQL ALLOCATE :Lob_loc;
    EXEC SQL SELECT Sound INTO Lob_loc
```

```

        FROM Multimedia_tab WHERE Clip_ID = 1;
/* BLOB をオープンします。*/
EXEC SQL LOB OPEN :Lob_loc READ ONLY;
/* 無名 PL\SQL ブロックから SUBSTR() を起動します。*/
EXEC SQL EXECUTE
    BEGIN
        :Buffer := DBMS_LOB.SUBSTR(:Lob_loc, :Amount, :Position);
    END;
END-EXEC;
/* BLOB をクローズします。*/
EXEC SQL LOB CLOSE :Lob_loc;
/* データを処理します。*/
/* ロケータが使用しているリソースを解放します。*/
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 (0040) : LOB の一部の読み込み (substr)

```

'0040 で LOB (または BFILE) の一部を読み込むには、OraBlob.Offset および
'OraBlob.chunksizeNote プロパティを設定する必要があることに注意してください。
'OraClob.Read メカニズムの使用
Dim MySession As OraSession
Dim OraDb As OraDatabase
Dim OraDyn as OraDynaset, OraStory as OraClob, amount_read%, chunksize%, chunk

Set MySession = CreateObject("OracleInProcServer.XOraSession")
Set OraDb = MySession.OpenDatabase("exampledb", "samp/samp", 0&)

Set OraDyn = OraDb.CreateDynaset("SELECT * FROM Multimedia_tab", ORADYN_DEFAULT)
Set OraStory = OraDyn.Fields("Story").Value

'500 バイト以降の 100 バイトを読み込みます。
OraStory.Offset = 500
OraStory.PollingAmount = OraStory.Size 'Read entire CLOB contents
amount_read = OraStory.Read(chunk, 100)
'戻されたチャンクは、可変の型バイト配列です。

```

Java (JDBC) : LOB の一部の読み込み (substr)

```
import java.io.InputStream;
import java.io.OutputStream;

// コア JDBC クラス :
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 固有の JDBC クラス :
import oracle.sql.*;
import oracle.jdbc.driver.*;

public class Ex2_79
{

    static final int MAXBUFSIZE = 32767;
    public static void main (String args [])
        throws Exception
    {
        // Oracle JDBC Driver をロードします。
        DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

        // データベースに接続します。
        Connection conn =
            DriverManager.getConnection ("jdbc:oracle:oci8:@", "samp", "samp");

        // 自動コミットがオフの場合、高速になります。
        conn.setAutoCommit (false);

        // 文を作成します。
        Statement stmt = conn.createStatement ();
        try
        {
            BLOB lob_loc = null;
            byte buf[] = new byte[MAXBUFSIZE];

            ResultSet rset = stmt.executeQuery (
                "SELECT frame FROM multimedia_tab WHERE clip_id = 1");
            if (rset.next())
            {
```

```
        lob_loc = ((OracleResultSet)rset).getBLOB (1);
    }

    OracleCallableStatement cstmt = (OracleCallableStatement)
        conn.prepareStatement ("BEGIN DBMS_LOB.OPEN(?, "
            + "DBMS_LOB.LOB_READONLY); END;");
    cstmt.setBLOB(1, lob_loc);
    cstmt.execute();

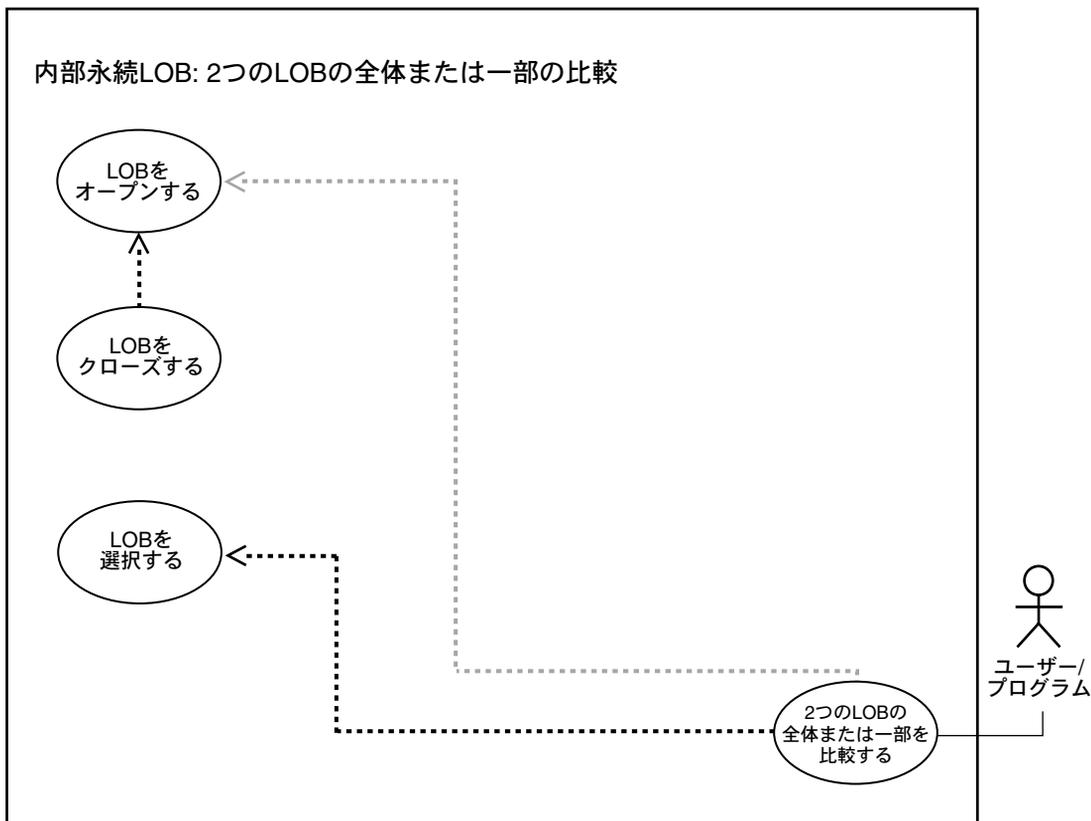
    // MAXBUFSIZE は読み込むバイト数であり、1,000 は読み込みを開始するオフセットです。
    buf = lob_loc.getBytes(1000, MAXBUFSIZE);
    // バッファの内容を表示します。

    cstmt = (OracleCallableStatement)
        conn.prepareStatement ("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();
}
}
```

2つのLOBの全体または一部の比較

図 9-24 ユースケース図：2つのLOBの全体または一部の比較



参照： 内部永続LOBに関するすべての基本操作については、9-2 ページの「ユースケース・モデル：内部永続LOB – 基本操作」を参照してください。

用途

2つのLOBの全部または一部を比較します。

使用上の注意

ありません。

構文

各プログラム環境で使用可能な機能のリストは、第3章「LOB プログラム環境」を参照してください。各プログラム環境における構文については、次のマニュアルの項を参照してください。

- PL/SQL (DBM_LOB パッケージ) : 『Oracle8i PL/SQL パッケージ・プロシージャ・リファレンス』の第18章「DBMS_LOB」の「サブプログラムの要約」の COMPARE ファンクション
- C (OCI) : 参照マニュアルはありません。
- COBOL (Pro*COBOL) : 『Oracle8i Pro*COBOL プリコンパイラ・プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「EXECUTE (実行可能埋込み SQL)」、および『Oracle8i PL/SQL パッケージ・プロシージャ・リファレンス』の第18章「DBMS_LOB」の「サブプログラムの要約」の COMPARE ファンクション
- C/C++ (Pro*C/C++) : 『Oracle8i Pro*C/C++ プリコンパイラ・プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB OPEN (実行可能埋込み SQL 拡張要素)」、「LOB CLOSE (実行可能埋込み SQL 拡張要素)」、および『Oracle8i PL/SQL パッケージ・プロシージャ・リファレンス』の第18章「DBMS_LOB」の「サブプログラムの要約」の COMPARE ファンクション
- Visual Basic (Oracle Objects for OLE オンライン・ヘルプ) : 「Help」の「Contents」タブから、「OO4O Automation Server」>「OBJECTS」>「Oradynaset」>「METHODS」>「movenext」を選択してください。
- Java (JDBC) : 『Oracle8i JDBC 開発者ガイドおよびリファレンス』の第7章「LOB と BFILE の操作」の「BLOB と CLOB の操作」の「BLOB または CLOB 列の作成と移入」、および『Oracle8i SQLJ 開発者ガイドおよびリファレンス』の第5章「型のサポート」の「Oracle の拡張型」の「BLOB、CLOB および BFILE のサポート」

使用例

次の例は、アーカイブ表 VideoframesLib_tab から2つのフレームを比較して異なっているかどうかをチェックし、比較の結果に応じて、フレームを Multimedia_tab に挿入します。

例

次のプログラム環境の例が示されています。

- [PL/SQL \(DBMS_LOB パッケージ\) : 2 つの LOB の全体または一部の比較 \(9-120 ページ\)](#)
- C (OCI) : 今回のリリースでは例は記載されていません。
- [COBOL \(Pro*COBOL\) : 2 つの LOB の全体または一部の比較 \(9-121 ページ\)](#)
- [C/C++ \(Pro*C/C++\) : 2 つの LOB の全体または一部の比較 \(9-122 ページ\)](#)
- [Visual Basic \(OO4O\) : 2 つの LOB の全体または一部の比較 \(9-123 ページ\)](#)
- [Java \(JDBC\) : 2 つの LOB の全体または一部の比較 \(9-124 ページ\)](#)

PL/SQL (DBMS_LOB パッケージ) : 2 つの LOB の全体または一部の比較

```
/* 例のプロシージャ compareTwoLOBs_proc は、DBMS_LOB パッケージの
一部ではないことに注意してください。 */
CREATE OR REPLACE PROCEDURE compareTwoLOBs_proc IS
  Lob_loc1          BLOB;
  Lob_loc2          BLOB;
  Amount           INTEGER := 32767;
  Retval           INTEGER;
BEGIN
  /* LOB を選択します。 */
  SELECT Frame INTO Lob_loc1 FROM Multimedia_tab
     WHERE Clip_ID = 1;
  SELECT Frame INTO Lob_loc2 FROM Multimedia_tab
     WHERE Clip_ID = 2;
  /* LOB のオープンはオプションです。 */
  DBMS_LOB.OPEN (Lob_loc1, DBMS_LOB.LOB_READONLY);
  DBMS_LOB.OPEN (Lob_loc2, DBMS_LOB.LOB_READONLY);
  /* 2 つのフレームを比較します。 */
  retval := DBMS_LOB.COMPARE(Lob_loc1, Lob_loc2, Amount, 1, 1);
  IF retval = 0 THEN
    DBMS_OUTPUT.PUT_LINE('Processing for equal frames');
  ELSE
    DBMS_OUTPUT.PUT_LINE('Processing for non-equal frames');
  END IF;
  /* LOB をオープンしている場合、その LOB をクローズする必要があります。 */
  DBMS_LOB.CLOSE (Lob_loc1);
  DBMS_LOB.CLOSE (Lob_loc2);
END;
```

COBOL (Pro*COBOL) : 2つのLOBの全体または一部の比較

```

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.
* BLOB ロケータの割当ておよび初期化を行います。
EXEC SQL ALLOCATE :BLOB1 END-EXEC.
EXEC SQL ALLOCATE :BLOB2 END-EXEC.
EXEC SQL WHENEVER NOT FOUND GOTO END-OF-BLOB END-EXEC.
EXEC SQL
SELECT FRAME INTO :BLOB1
FROM MULTIMEDIA_TAB M WHERE M.CLIP_ID = 1 END-EXEC.
EXEC SQL
SELECT FRAME INTO :BLOB2
FROM MULTIMEDIA_TAB M WHERE M.CLIP_ID = 2 END-EXEC.

* READ ONLY用のBLOBをオープンします。
EXEC SQL LOB OPEN :BLOB1 READ ONLY END-EXEC.
EXEC SQL LOB OPEN :BLOB2 READ ONLY END-EXEC.

* PL/SQLを実行して、COMPARE機能を取得します。
MOVE 4 TO AMT.
EXEC SQL EXECUTE
BEGIN
:RET := DBMS_LOB.COMPARE(:BLOB1, :BLOB2, :AMT, 1, 1); END; END-EXEC.

IF RET = 0

```

```
*          等価のLOBに対する論理がここに入ります。
          DISPLAY "BLOBs are equal"
ELSE
*          等価でないLOBに対する論理がここに入ります。
          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 SQLERRM.
EXEC SQL ROLLBACK WORK RELEASE END-EXEC.
STOP RUN.
```

C/C++ (Pro*C/C++) : 2つのLOBの全体または一部の比較

```
#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();
```

```

/* LOB ロケータを割り当てます。*/
EXEC SQL ALLOCATE :Lob_loc1;
EXEC SQL ALLOCATE :Lob_loc2;
/* LOB を選択します。*/
EXEC SQL SELECT Frame INTO :Lob_loc1
      FROM Multimedia_tab WHERE Clip_ID = 1;
EXEC SQL SELECT Frame INTO :Lob_loc2
      FROM Multimedia_tab WHERE Clip_ID = 2;
/* LOB のオープンはオプションです。*/
EXEC SQL LOB OPEN :Lob_loc1 READ ONLY;
EXEC SQL LOB OPEN :Lob_loc2 READ ONLY;
/* PL/SQL 内から DBMS_LOB.COMPARE () を使用して、2 つのフレームを比較します。*/
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");
/* LOB をオープンしている場合、その LOB をクローズする必要があります。*/
EXEC SQL LOB CLOSE :Lob_loc1;
EXEC SQL LOB CLOSE :Lob_loc2;
/* ロケータが保持しているリソースを解放します。*/
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 (0040) : 2 つの LOB の全体または一部の比較

```

Dim MySession As OraSession
Dim OraDb As OraDatabase

Set MySession = CreateObject("OracleInProcServer.XOraSession")
Set OraDb = MySession.OpenDatabase("examplepdb", "samp/samp", 0&)
Dim OraDyn as OraDynaset, OraSound1 as OraBLOB, OraSoundClone as OraBLOB

```

```
Set OraDyn = OraDb.CreateDynaset (
    "SELECT * FROM Multimedia_tab ORDER BY clip_id", ORADYN_DEFAULT)
Set OraSound1 = OraDyn.Fields("Sound").Value
' 将来の参照用にクローンを作成します。
Set OraSoundClone = OraSound1.Clone

' 次の行に進み、LOBを比較します。
OraDyn.MoveNext

MsgBox CBool(OraSound1.Compare(OraSoundClone, OraSoundClone.size, 1, 1))
```

Java (JDBC) : 2つのLOBの全体または一部の比較

```
import java.io.InputStream;
import java.io.OutputStream;

// コア JDBC クラス :
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 固有の JDBC クラス :
import oracle.sql.*;
import oracle.jdbc.driver.*;

public class Ex2_87
{
    static final int MAXBUFSIZE = 32767;
    public static void main (String args [])
        throws Exception
    {
        // Oracle JDBC Driver をロードします。
        DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

        // データベースに接続します。
        Connection conn =
            DriverManager.getConnection ("jdbc:oracle:oci8:@", "samp", "samp");

        // 自動コミットがオフの場合、高速になります。
        conn.setAutoCommit (false);
```

```
// 文を作成します。
Statement stmt = conn.createStatement ();
try
{
    BLOB lob_loc1 = null;
    BLOB lob_loc2 = null;
    ResultSet rset = stmt.executeQuery (
        "SELECT frame FROM multimedia_tab WHERE clip_id = 1");
    if (rset.next())
    {
        lob_loc1 = ((OracleResultSet)rset).getBLOB (1);
    }

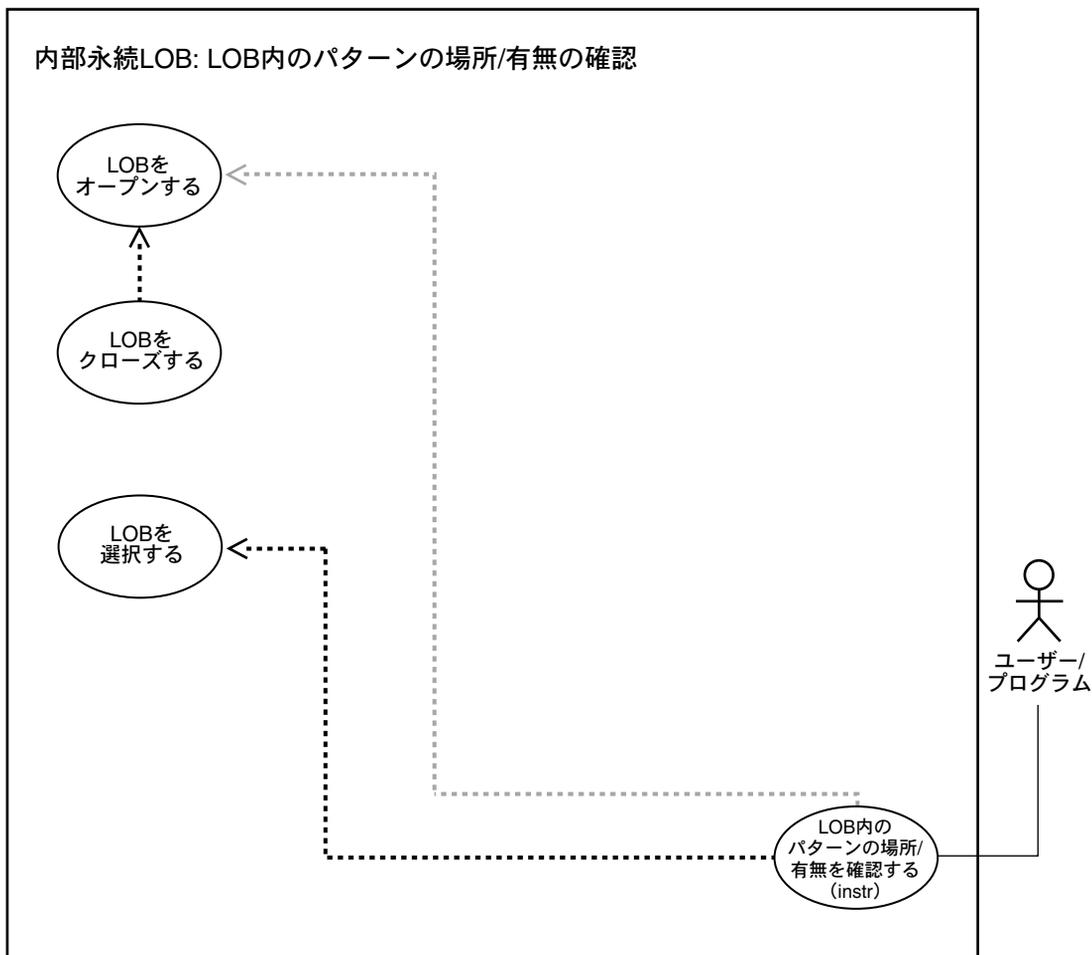
    rset = stmt.executeQuery (
        "SELECT frame FROM multimedia_tab WHERE clip_id = 99");
    if (rset.next())
    {
        lob_loc2 = ((OracleResultSet)rset).getBLOB (1);
    }

    if (lob_loc1.length() > lob_loc2.length())
        System.out.println ("Looking for LOB2 inside LOB1. result = "
            + Long.toString(lob_loc1.position(lob_loc2, 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();
}
}
```

LOB 内のパターンの有無の確認 (instr)

図 9-25 ユースケース図 : LOB 内のパターンの有無の確認 (instr)



参照: 内部永続 LOB に関するすべての基本操作については、9-2 ページの「ユースケース・モデル:内部永続 LOB - 基本操作」を参照してください。

用途

LOB 内のパターンの有無を確認します (instr)。

使用上の注意

ありません。

構文

各プログラム環境で使用可能な機能のリストは、第 3 章「LOB プログラム環境」を参照してください。各プログラム環境における構文については、次のマニュアルの項を参照してください。

- PL/SQL (DBM_LOB パッケージ) : 『Oracle8i PL/SQL パッケージ・プロシージャ・リファレンス』の第 18 章「DBMS_LOB」の「サブプログラムの要約」の INSTR ファンクション
- C (OCI) : 参照マニュアルはありません。
- COBOL (Pro*COBOL) : 『Oracle8i Pro*COBOL プリコンパイラ・プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB OPEN (実行可能埋込み SQL 拡張要素)」、「LOB CLOSE (実行可能埋込み SQL 拡張要素)」、および『Oracle8i PL/SQL パッケージ・プロシージャ・リファレンス』の第 18 章「DBMS_LOB」の「サブプログラムの要約」の INSTR ファンクション
- C/C++ (Pro*C/C++) : 『Oracle8i Pro*C/C++ プリコンパイラ・プログラマーズ・ガイド』付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB OPEN (実行可能埋込み SQL 拡張要素)」、「LOB CLOSE (実行可能埋込み SQL 拡張要素)」、および『Oracle8i PL/SQL パッケージ・プロシージャ・リファレンス』の第 18 章「DBMS_LOB」の「サブプログラムの要約」の INSTR ファンクション
- Visual Basic (OO4O) : 参照マニュアルはありません。
- Java (JDBC) : 『Oracle8i JDBC 開発者ガイドおよびリファレンス』の第 7 章「LOB と BFILE の操作」の「BLOB と CLOB の操作」の「BLOB または CLOB 列の作成と移入」、および『Oracle8i SQLJ 開発者ガイドおよびリファレンス』の第 5 章「型のサポート」の「Oracle の拡張型」の「BLOB、CLOB および BFILE のサポート」

使用例

次の例では、ストーリーボードのテキストに「children」という文字列が存在するかどうかを調べます。

例

次のプログラム環境の例が示されています。

- **PL/SQL (DBMS_LOB パッケージ) : LOB 内のパターンの有無の確認 (instr) (9-128 ページ)**
- **C (OCI) : 今回のリリースでは例は記載されていません。**
- **COBOL (Pro*COBOL) : LOB 内のパターンの有無の確認 (instr) (9-129 ページ)**
- **C/C++ (Pro*C/C++) : LOB 内のパターンの有無の確認 (instr) (9-130 ページ)**
- **Visual Basic (OO4O) : 今回のリリースでは例は記載されていません。**
- **Java (JDBC) : LOB 内のパターンの有無の確認 (instr) (9-131 ページ)**

PL/SQL (DBMS_LOB パッケージ) : LOB 内のパターンの有無の確認 (instr)

```
/* 例のプロシージャ instrstringLOB_proc は、DBMS_LOB パッケージの
一部ではないことに注意してください。*/
CREATE OR REPLACE PROCEDURE instrstringLOB_proc IS
  Lob_loc          CLOB;
  Pattern          VARCHAR2(30) := 'children';
  Position         INTEGER := 0;
  Offset           INTEGER := 1;
  Occurrence       INTEGER := 1;
BEGIN
  /* LOB を選択します。*/
  SELECT Story INTO Lob_loc
    FROM Multimedia_tab
   WHERE Clip_ID = 1;
  /* LOB のオープンはオプションです。*/
  DBMS_LOB.OPEN (Lob_loc, DBMS_LOB.LOB_READONLY);
  /* パターンを検索します。*/
  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;
  /* LOB をオープンしている場合、LOB をクローズする必要があります。*/
  DBMS_LOB.CLOSE (Lob_loc);
END;
```

COBOL (Pro*COBOL) : LOB 内のパターンの有無の確認 (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.

* CLOB ロケータの割当ておよび初期化を行います。
EXEC SQL ALLOCATE :CLOB1 END-EXEC.
EXEC SQL WHENEVER NOT FOUND GOTO END-OF-CLOB END-EXEC.
EXEC SQL SELECT STORY INTO :CLOB1
FROM MULTIMEDIA_TAB WHERE CLIP_ID = 1 END-EXEC.

* READ ONLY 用の CLOB をオープンします。
EXEC SQL LOB OPEN :CLOB1 READ ONLY END-EXEC.

* PL/SQL を実行して、INSTR 機能を取得します。
EXEC SQL EXECUTE
BEGIN
:POS := DBMS_LOB.INSTR(:CLOB1, :PATTERN, :OFFSET, :OCCURRENCE);
END; END-EXEC.

IF POS = 0
* パターンが見つからない場合に対する論理はここにあります。
DISPLAY "Pattern not found."
ELSE
* Pos には、パターンが見つかった位置が含まれます。
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++) : LOB 内のパターンの有無の確認 (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 Story INTO :Lob_loc
        FROM Multimedia_tab WHERE Clip_ID = 1;
    /* LOB のオープンはオプションです。*/
    EXEC SQL LOB OPEN :Lob_loc;
    /* PL/SQL ブロック内で BMS_LOB.INSTR() を使用して、パターンを検索します。*/
    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);
    /* LOB をオープンしている場合、その LOB をクローズする必要があります。 */
    EXEC SQL LOB CLOSE :Lob_loc;
    EXEC SQL FREE :Lob_loc;
}

void main()
{
    char *samp = "samp/samp";
    EXEC SQL CONNECT :samp;
    instringLOB_proc();
    EXEC SQL ROLLBACK WORK RELEASE;
}

```

Java (JDBC) : LOB 内のパターンの有無の確認 (instr)

```

import java.io.OutputStream;

// コア JDBC クラス :
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 固有の JDBC クラス :
import oracle.sql.*;
import oracle.jdbc.driver.*;

public class Ex2_91
{
    static final int MAXBUFSIZE = 32767;
    public static void main (String args [])
        throws Exception
    {
        // Oracle JDBC Driver をロードします。
        DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
    }
}

```

```
// データベースに接続します。
Connection conn =
    DriverManager.getConnection ("jdbc:oracle:oci8:@", "samp", "samp");

// 自動コミットがオフの場合、高速になります。
conn.setAutoCommit (false);

// 文を作成します。
Statement stmt = conn.createStatement ();
try
{
    final int offset = 1;        // 最初のバイトから検索を開始します。
    final int occurrence = 1;    // CLOB 内で最初に出現したパターンから開始します。

    CLOB lob_loc = null;
    String pattern = new String("Junk"); // CLOB 内で検索するパターンです。

    ResultSet rset = stmt.executeQuery (
        "SELECT story FROM multimedia_tab WHERE clip_id = 2");
    if (rset.next())
    {
        lob_loc = ((OracleResultSet)rset).getCLOB (1);
    }

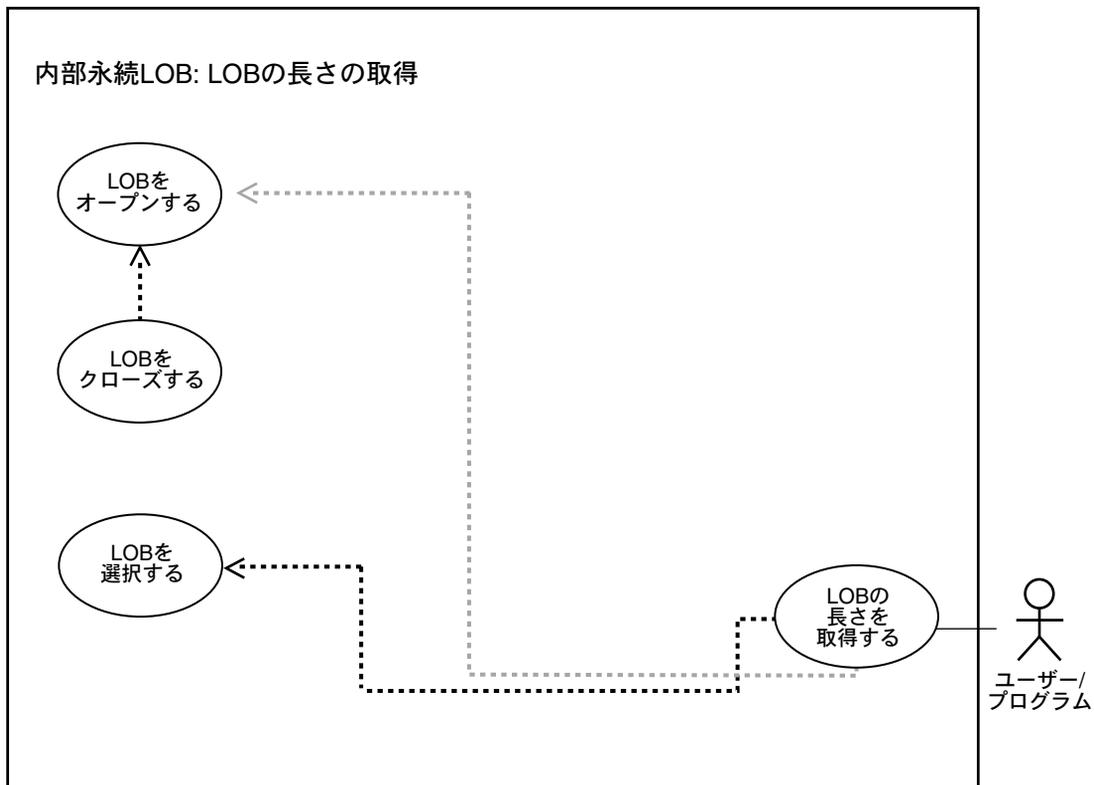
    // オフセット 1 から開始して、CLOB 内のパターン文字列の位置を検索します。
    long result = lob_loc.position(pattern, offset);
    System.out.println("Results of Pattern Comparison : " +
        Long.toString(result));

    stmt.close();
    conn.commit();
    conn.close();

}
catch (SQLException e)
{
    e.printStackTrace();
}
}
```

LOB の長さの取得

図 9-26 ユースケース図 : LOB の長さの取得



参照: 内部永続 LOB に関するすべての基本操作については、9-2 ページの「ユースケース・モデル:内部永続 LOB - 基本操作」を参照してください。

用途

LOB の長さを決定します。

使用上の注意

ありません。

構文

各プログラム環境で使用可能な機能のリストは、[第 3 章「LOB プログラム環境」](#)を参照してください。各プログラム環境における構文については、次のマニュアルの項を参照してください。

- PL/SQL (DBM_LOB パッケージ) : 『Oracle8i PL/SQL パッケージ・プロシージャ・リファレンス』の第 18 章「DBMS_LOB」の「サブプログラムの要約」の GETLENGTH ファンクション
- C (OCI) : 『Oracle8i コール・インタフェース・プログラマーズ・ガイド』の第 15 章「OCI リレーショナル関数」の「LOB 関数」の OCILobGetLength()
- COBOL (Pro*COBOL) : 『Oracle8i Pro*COBOL プリコンパイラ・プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB DESCRIBE (実行可能埋込み SQL 拡張要素)」
- C/C++ (Pro*C/C++) : 『Oracle8i Pro*C/C++ プリコンパイラ・プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB DESCRIBE (実行可能埋込み SQL 拡張要素)」
- Visual Basic (Oracle Objects for OLE オンライン・ヘルプ) : 「Help」の「Contents」タブから、「OO4O Automation Server」>「OBJECTS」>「Oradynaset」>「PROPERTIES」>「fields」を選択してください。
- Java (JDBC) : 『Oracle8i JDBC 開発者ガイドおよびリファレンス』の第 7 章「LOB と BFILE の操作」の「BLOB と CLOB の操作」の「BLOB または CLOB 列の作成と移入」、および『Oracle8i SQLJ 開発者ガイドおよびリファレンス』の第 5 章「型のサポート」の「Oracle の拡張型」の「BLOB、CLOB および BFILE のサポート」

使用例

次の例では、外国語のサブタイトル (FLSub) に関連して、LOB の長さを決定する方法を説明します。

例

次のプログラム環境の例が示されています。

- [PL/SQL \(DBMS_LOB パッケージ\) : LOB の長さの取得 \(9-135 ページ\)](#)
- [C \(OCI\) : LOB の長さの取得 \(9-135 ページ\)](#)
- [COBOL \(Pro*COBOL\) : LOB の長さの取得 \(9-137 ページ\)](#)

- C/C++ (Pro*C/C++) : LOB の長さの取得 (9-138 ページ)
- Visual Basic (OO4O) : LOB の長さの取得 (9-139 ページ)
- Java (JDBC) : LOB の長さの取得 (9-139 ページ)

PL/SQL (DBMS_LOB パッケージ) : LOB の長さの取得

```

/* 例のプロシージャ getLengthLOB_proc は、DBMS_LOB パッケージの
一部ではないことに注意してください。*/
CREATE OR REPLACE PROCEDURE getLengthLOB_proc IS
  Lob_loc      NCLOB;
  Length      INTEGER;
BEGIN
  /* LOB を選択します。*/
  SELECT FLSub INTO Lob_loc FROM Multimedia_tab
     WHERE Clip_ID = 2;
  /* LOB のオープンはオプションです。*/
  DBMS_LOB.OPEN (Lob_loc, DBMS_LOB.LOB_READONLY);
  /* LOB の長さを取得します。*/
  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;
  /* LOB をオープンしている場合、その LOB をクローズする必要があります。*/
  DBMS_LOB.CLOSE (Lob_loc);
END;

```

C (OCI) : LOB の長さの取得

```

/* ロケータ変数にロケータを選択します。*/
sb4 select_FLSub_locator(Lob_loc, errhp, svchp, stmthp)
OCILobLocator *Lob_loc;
OCIError      *errhp;
OCISvcCtx     *svchp;
OCIStmt       *stmthp;
{
  OCIDefine *defnp1;

  text *sqlstmt =
    (text *) "SELECT FLSub FROM Multimedia_tab WHERE Clip_ID = 2";

```

```
checkerr (errhp, OCIStmtPrepare(stmthp, errhp, sqlstmt,
                                (ub4)strlen((char *)sqlstmt),
                                (ub4)OCI_NTV_SYNTAX, (ub4)OCI_DEFAULT));
/* 選択されている列を定義します。*/
checkerr (errhp, OCIDefineByPos(stmthp, &defnp1, errhp, (ub4) 1,
                                (dvoid *)&Lob_loc, (sb4)0,
                                (ub2)SQLT_CLOB, (dvoid *)0, (ub2 *)0,
                                (ub2 *)0, (ub4)OCI_DEFAULT));
/* SELECT を実行し、1 行フェッチします。*/
checkerr (errhp, OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                                (CONST OCISnapshot*) 0, (OCISnapshot*) 0,
                                (ub4) OCI_DEFAULT));

return 0;
}

/* このファンクションによって、選択された LOB の長さが取得されます。*/
void getLengthLob(envhp, errhp, svchp, stmthp)
OCIEnv    *envhp;
OCIError  *errhp;
OCISvcCtx *svchp;
OCIStmt   *stmthp;
{
    ub4 length;
    OCILobLocator *Lob_loc;
    /* ロケータ・リソースを割り当てます。*/
    (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &Lob_loc,
                              (ub4)OCI_DTYPE_LOB, (size_t) 0, (dvoid **) 0);

    /* FLSub から LOB を選択します。*/
    printf(" select a FLSub locator\n");
    select_FLSub_locator(Lob_loc, errhp, svchp, stmthp);

    /* LOB のオープンはオプションです。*/
    printf(" Open the locator (optional)\n");
    checkerr (errhp, (OCILobOpen(svchp, errhp, Lob_loc, OCI_LOB_READONLY)));

    printf(" get the length of FLSub.\n");
    checkerr (errhp, OCILobGetLength(svchp, errhp, Lob_loc, &length));

    /* LOB が NULL または未定義の場合、長さは未定義です。*/
    fprintf(stderr, " Length of LOB is %d\n", length);

    /* LOB をオープンしている場合、その LOB をクローズする必要があります。*/
    checkerr (errhp, OCILobClose(svchp, errhp, Lob_loc));
}
```

```
/* ロケータが保持しているリソースを解放します。*/  
(void) OCIDescriptorFree((dvoid *) Lob_loc, (ub4) OCI_DTYPE_LOB);  
  
return;  
}
```

COBOL (Pro*COBOL) : LOB の長さの取得

```
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.  
  
* ターゲット CLOB の割当ておよび初期化を行います。  
   EXEC SQL ALLOCATE :CLOB1 END-EXEC.  
   EXEC SQL WHENEVER NOT FOUND GOTO END-OF-CLOB END-EXEC.  
   EXEC SQL  
     SELECT STORY INTO :CLOB1  
     FROM MULTIMEDIA_TAB WHERE CLIP_ID = 2 END-EXEC.  
  
* CLOB の長さを取得します。  
   EXEC SQL  
     LOB DESCRIBE :CLOB1 GET LENGTH INTO :LEN END-EXEC.  
   MOVE LEN TO D-LEN.  
   DISPLAY "The length is ", D-LEN.  
  
* CLOB が使用しているリソースを解放します。  
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++) : LOB の長さの取得

```
#include <oci.h>  
#include <stdio.h>  
#include <sqlca.h>  
  
void Sample_Error()  
{  
    EXEC SQL WHENEVER SQLERROR CONTINUE;  
    printf("%.*s\n", sqlca.sqlerrm.sqlerrml, sqlca.sqlerrm.sqlerrmc);  
    EXEC SQL ROLLBACK WORK RELEASE;  
    exit(1);  
}  
void getLengthLOB_proc()  
{  
    OCIClobLocator *Lob_loc;  
    unsigned int Length;  
  
    EXEC SQL WHENEVER SQLERROR DO Sample_Error();  
    EXEC SQL ALLOCATE :Lob_loc;  
    EXEC SQL SELECT Story INTO :Lob_loc  
        FROM Multimedia_tab WHERE Clip_ID = 1;  
    /* LOB のオープンはオプションです。 */  
    EXEC SQL LOB OPEN :Lob_loc READ ONLY;  
    /* 長さを取得します。 */  
    EXEC SQL LOB DESCRIBE :Lob_loc GET LENGTH INTO :Length;  
    /* LOB が NULL または初期化されていない場合、長さは未定義です。 */  
    printf("Length is %d characters\n", Length);  
    /* LOB をオープンしている場合、その LOB をクローズする必要があります。 */  
    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 (0040) : LOB の長さの取得

```
Dim MySession As OraSession
Dim OraDb As OraDatabase

Dim OraDyn As OraDynaset, OraSound1 As OraBlob, OraSoundClone As OraBlob

Set MySession = CreateObject("OracleInProcServer.XOraSession")
Set OraDb = MySession.OpenDatabase("exampledb", "samp/samp", 0&)
Set OraDyn = OraDb.CreateDynaset(
  "SELECT * FROM Multimedia_tab ORDER BY clip_id", ORADYN_DEFAULT)
Set OraSound1 = OraDyn.Fields("Sound").Value

'LOB のサイズを表示します。
MsgBox "Length of the lob is " & OraSound1.Size
```

Java (JDBC) : LOB の長さの取得

```
import java.io.OutputStream;

// コア JDBC クラス :
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 固有の JDBC クラス :
import oracle.sql.*;
import oracle.jdbc.driver.*;

public class Ex2_95
{
```

```
static final int MAXBUFSIZE = 32767;
public static void main (String args [])
    throws Exception
{
    // Oracle JDBC Driver をロードします。
    DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

    // データベースに接続します。
    Connection conn =
        DriverManager.getConnection ("jdbc:oracle:oci8:@", "samp", "samp");

    // 自動コミットがオフの場合、高速になります。
    conn.setAutoCommit (false);

    // 文を作成します。
    Statement stmt = conn.createStatement ();

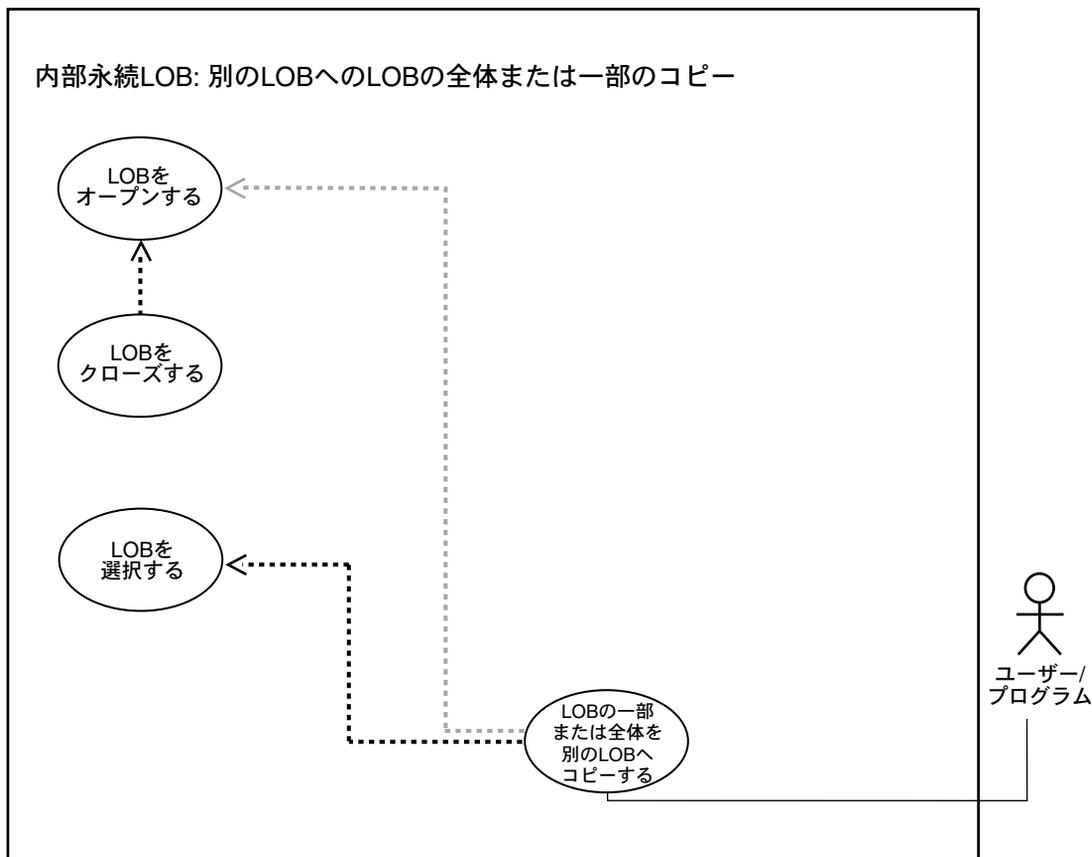
    try
    {
        CLOB lob_loc = null;
        ResultSet rset = stmt.executeQuery
            ("SELECT story FROM multimedia_tab WHERE clip_id = 2");
        if (rset.next())
        {
            lob_loc = ((OracleResultSet)rset).getCLOB (1);
        }

        System.out.println(
            "Length of this column is : " + Long.toString(lob_loc.length()));

        stmt.close();
        conn.commit();
        conn.close();
    }
    catch (SQLException e)
    {
        e.printStackTrace();
    }
}
```

別のLOBへのLOBの全体または一部のコピー

図 9-27 ユースケース図：別のLOBへのLOBの全体または一部のコピー



参照： 内部永続LOBに関するすべての基本操作については、9-2 ページの「ユースケース・モデル:内部永続LOB – 基本操作」を参照してください。

用途

LOBの全体または一部を、別のLOBにコピーします。

使用上の注意

更新前の行のロック

PL/SQL DBMS_LOB パッケージまたは OCI を使用して LOB の値を更新する前に、LOB を含む行をロックする必要があります。SQL INSERT 文および UPDATE 文は暗黙的に行をロックしますが、SQL プログラムおよび PL/SQL プログラムでは SQL SELECT FOR UPDATE 文を使用して、また OCI プログラムでは OCI pin または lock 関数を使用して明示的にロックします。

更新後のロケータの状態の詳細は、5-6 ページの「更新済ロケータを介した更新済 LOB」を参照してください。

構文

各プログラム環境で使用可能な機能のリストは、第 3 章「LOB プログラム環境」を参照してください。各プログラム環境における構文については、次のマニュアルの項を参照してください。

- PL/SQL (DBMS_LOB パッケージ) : 『Oracle8i PL/SQL パッケージ・プロシージャ・リファレンス』の第 18 章「DBMS_LOB」の「サブプログラムの要約」の COPY プロシージャ
- C (OCI) : 『Oracle8i コール・インタフェース・プログラマーズ・ガイド』の第 15 章「OCI リレーショナル関数」の「LOB 関数」の OCILobCopy()
- COBOL (Pro*COBOL) : 『Oracle8i Pro*COBOL プリコンパイラ・プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB COPY (実行可能埋込み SQL 拡張要素)」、および『Oracle8i PL/SQL パッケージ・プロシージャ・リファレンス』の第 18 章「DBMS_LOB」の「サブプログラムの要約」の COPY プロシージャ
- C/C++ (Pro*C/C++) : 『Oracle8i Pro*C/C++ プリコンパイラ・プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB COPY (実行可能埋込み SQL 拡張要素)」
- Visual Basic (Oracle Objects for OLE オンライン・ヘルプ) : 「Help」の「Contents」タブから、「OO4O Automation Server」>「OBJECTS」>「Oralob」>「METHODS」>「copy」、および「OBJECTS」>「Oradynaset」>「METHODS」>「movenext」、「edit」を選択してください。
- Java (JDBC) : 『Oracle8i JDBC 開発者ガイドおよびリファレンス』の第 7 章「LOB と BFILE の操作」の「BLOB と CLOB の操作」の「BLOB または CLOB 列の作成と移入」、および『Oracle8i SQLJ 開発者ガイドおよびリファレンス』の第 5 章「型のサポート」の「Oracle の拡張型」の「BLOB、CLOB および BFILE のサポート」

使用例

次の例では、1つのクリップから別のクリップへ Sound の一部をコピーします。

例

次のプログラム環境の例が示されています。

- [PL/SQL \(DBMS_LOB パッケージ\) : 別の LOB への LOB の全体または一部のコピー \(9-143 ページ\)](#)
- [C \(OCI\) : 別の LOB への LOB の全体または一部のコピー \(9-144 ページ\)](#)
- [COBOL \(Pro*COBOL\) : 別の LOB への LOB の全体または一部のコピー \(9-146 ページ\)](#)
- [C/C++ \(Pro*C/C++\) : 別の LOB への LOB の全体または一部のコピー \(9-148 ページ\)](#)
- [Visual Basic \(OO4O\) : 別の LOB への LOB の全体または一部のコピー \(9-149 ページ\)](#)
- [Java \(JDBC\) : 別の LOB への LOB の全体または一部のコピー \(9-149 ページ\)](#)

PL/SQL (DBMS_LOB パッケージ) : 別の LOB への LOB の全体または一部のコピー

```
/* 例のプロシージャ copyLOB_proc は、DBMS_LOB パッケージの
一部ではないことに注意してください。*/
CREATE OR REPLACE PROCEDURE copyLOB_proc IS
    Dest_loc    BLOB;
    Src_loc     BLOB;
    Amount      NUMBER;
    Dest_pos    NUMBER;
    Src_pos     NUMBER;
BEGIN
    SELECT Sound INTO Dest_loc FROM Multimedia_tab
        WHERE Clip_ID = 2 FOR UPDATE;
    /* LOB を選択します。*/
    SELECT Sound INTO Src_loc FROM Multimedia_tab
        WHERE Clip_ID = 1;
    /* LOB のオープンはオプションです。*/
    DBMS_LOB.OPEN(Dest_loc, DBMS_LOB.LOB_READWRITE);
    DBMS_LOB.OPEN(Src_loc, DBMS_LOB.LOB_READONLY);
    /* ソースの位置から宛先の位置へ LOB をコピーします。*/
    DBMS_LOB.COPY(Dest_loc, Src_loc, Amount, Dest_pos, Src_pos);
    /* LOB をオープンしている場合、その LOB をクローズする必要があります。*/
    DBMS_LOB.CLOSE(Dest_loc);
    DBMS_LOB.CLOSE(Src_loc);
    COMMIT;
EXCEPTION
```

```

        WHEN OTHERS THEN
            DBMS_OUTPUT.PUT_LINE('Operation failed');
    END;

```

C (OCI) : 別の LOB への LOB の全体または一部のコピー

```

/* ロケータを選択します。*/
sb4 select_lock_sound_locator_2(Lob_loc, dest_type, errhp, svchp, stmthp)
OCILobLocator *Lob_loc;
ub1 dest_type; /* 宛先ロケータかどうか */
OCIError *errhp;
OCISvcCtx *svchp;
OCISmt *stmthp;
{
    char sqlstmt[150];
    OCIDefine *defnp1;
    if (dest_type == TRUE)
    {
        strcpy (sqlstmt,
            (char *) "SELECT Sound FROM Multimedia_tab
                WHERE Clip_ID=2 FOR UPDATE");
        printf (" select destination sound locator\n");
    }
    else
    {
        strcpy(sqlstmt, (char *) "SELECT Sound FROM Multimedia_tab WHERE Clip_ID=1");
        printf (" select source sound locator\n");
    }
    checkerr (errhp, OCISmtPrepare(stmthp, errhp, (text *)sqlstmt,
        (ub4)strlen((char *)sqlstmt),
        (ub4) OCI_NTV_SYNTAX, (ub4) OCI_DEFAULT));
    checkerr (errhp, OCIDefineByPos(stmthp, &defnp1, errhp, (ub4) 1,
        (dvoid *)&Lob_loc, (sb4) 0,
        (ub2) SOLT_BLOB, (dvoid *) 0,
        (ub2 *) 0, (ub2 *) 0, (ub4) OCI_DEFAULT));
    /* SELECT を実行し、1 行フェッチします。*/
    checkerr (errhp, OCISmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
        (CONST OCISnapshot*) 0, (OCISnapshot*) 0,
        (ub4) OCI_DEFAULT));

    return 0;
}

/* このファンクションによって、ソース LOB の一部が宛先 LOB 内の指定された位置に
コピーされます。*/

```

```

void copyAllPartLob(envhp, errhp, svchp, stmthp)
OCIEnv      *envhp;
OCIError    *errhp;
OCISvcCtx   *svchp;
OCIStmt     *stmthp;
{
    OCIBlobLocator *Dest_loc, *Src_loc;
    int Amount = 1000;
    int Dest_pos = 100;
    int Src_pos = 1;

    /* <コピーする量 > */
    /* <コピー先の開始位置 > */
    /* <コピー元の開始位置 > */

    /* LOB ロケータを割り当てます。 */
    (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &Dest_loc,
                              (ub4)OCI_DTYPE_LOB, (size_t) 0, (dvoid **) 0);
    (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &Src_loc,
                              (ub4)OCI_DTYPE_LOB, (size_t) 0, (dvoid **) 0);

    /* LOB を選択します。 */
    printf(" select the destination and source locators\n");
    select_lock_sound_locator_2(Dest_loc, TRUE, errhp, svchp, stmthp);
    /* 宛先ロケータ */
    select_lock_sound_locator_2(Src_loc, FALSE, errhp, svchp, stmthp);
    /* ソース・ロケータ */

    /* LOB のオープンはオプションです。 */
    printf (" open the destination locator (optional)\n");
    checkerr (errhp, OCILobOpen(svchp, errhp, Dest_loc, OCI_LOB_READWRITE));
    printf (" open the source locator (optional)\n");
    checkerr (errhp, OCILobOpen(svchp, errhp, Src_loc, OCI_LOB_READONLY));

    printf (" copy the lob (amount) from the source to destination\n");
    checkerr (errhp, OCILobCopy(svchp, errhp, Dest_loc, Src_loc,
                               Amount, Dest_pos, Src_pos));

    /* LOB をオープンしている場合、そのLOB をクローズする必要があります。 */
    printf(" close the locators\n");
    checkerr (errhp, OCILobClose(svchp, errhp, Dest_loc));
    checkerr (errhp, OCILobClose(svchp, errhp, Src_loc));

    /* ロケータを保持しているリソースを解放します。 */
    (void) OCIDescriptorFree((dvoid *) Dest_loc, (ub4) OCI_DTYPE_LOB);
    (void) OCIDescriptorFree((dvoid *) Src_loc, (ub4) OCI_DTYPE_LOB);

    return;
}

```

COBOL (Pro*COBOL) : 別の LOB への LOB の全体または一部のコピー

```
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.
```

* コピーする量を定義します。

* この量は任意に選択されています。

```
01 AMT PIC S9(9) COMP VALUE 10.
```

* ソースおよび宛先の位置を定義します。

* これらの値は任意に選択されています。

```
01 SRC-POS PIC S9(9) COMP VALUE 1.
```

```
01 DEST-POS PIC S9(9) COMP VALUE 1.
```

* 戻り値は PL/SQL ファンクションからのものです。

```
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.
```

* BLOB ロケータの割当ておよび初期化を行います。

```
EXEC SQL ALLOCATE :DEST END-EXEC.
```

```
EXEC SQL ALLOCATE :SRC END-EXEC.
```

```
DISPLAY "Source and destination LOBs are open."
```

```
EXEC SQL WHENEVER NOT FOUND GOTO END-OF-BLOB END-EXEC.
```

```
EXEC SQL
```

```
SELECT SOUND INTO :SRC
```

```
FROM MULTIMEDIA_TAB M WHERE M.CLIP_ID = 1 END-EXEC.
```

```
DISPLAY "Source LOB populated."
```

```
EXEC SQL
```

```
SELECT SOUND INTO :DEST
```

```
FROM MULTIMEDIA_TAB M WHERE M.CLIP_ID = 2 FOR UPDATE END-EXEC.
```

```
DISPLAY "Destination LOB populated."
```

* 宛先 LOB を読み込み / 書込みでオープンし、ソース LOB を読み込み専用でオープンします。

```
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."
```

- * 必要な量をコピーします。

```
EXEC SQL  
    LOB COPY :AMT FROM :SRC AT :SRC-POS  
    TO :DEST AT :DEST-POS END-EXEC.  
DISPLAY "Src LOB copied to destination LOB."
```

- * PL/SQL を実行して COMPARE 機能を取得し、BLOB が
- * 同一であることを確認します。

```
EXEC SQL EXECUTE  
    BEGIN  
        :RET := DBMS_LOB.COMPARE(:SRC, :DEST, :AMT, 1, 1); END; END-EXEC.
```

```
IF RET = 0
```

- * 等価の BLOB に対する論理がここに入ります。

```
    DISPLAY "BLOBs are equal"
```

```
ELSE
```

- * 等価でない BLOB に対する論理がここに入ります。

```
    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++) : 別の LOB への LOB の全体または一部のコピー

```
#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();
    /* LOB ロケータを割り当てます。 */
    EXEC SQL ALLOCATE :Dest_loc;
    EXEC SQL ALLOCATE :Src_loc;
    /* LOB を選択します。 */
    EXEC SQL SELECT Sound INTO :Dest_loc
        FROM Multimedia_tab WHERE Clip_ID = 2 FOR UPDATE;
    EXEC SQL SELECT Sound INTO :Src_loc
        FROM Multimedia_tab WHERE Clip_ID = 1;
    /* LOB のオープンはオプションです。 */
    EXEC SQL LOB OPEN :Dest_loc READ WRITE;
    EXEC SQL LOB OPEN :Src_loc READ ONLY;
    /* ソース LOB のソース位置から宛先 LOB の宛先位置へ、指定した量をコピーします。 */
    EXEC SQL LOB COPY :Amount
        FROM :Src_loc AT :Src_pos TO :Dest_loc AT :Dest_pos;
    /* LOB をオープンしている場合、その LOB をクローズする必要があります。 */
    EXEC SQL LOB CLOSE :Dest_loc;
    EXEC SQL LOB CLOSE :Src_loc;
    /* ロケータが保持しているリソースを解放します。 */
    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 (OO40) : 別の LOB への LOB の全体または一部のコピー

```

Dim MySession As OraSession
Dim OraDb As OraDatabase
Dim OraDyn As OraDynaset, OraSound1 As OraBlob, OraSoundClone As OraBlob

Set MySession = CreateObject("OracleInProcServer.XOraSession")
Set OraDb = MySession.OpenDatabase("exampleldb", "samp/samp", 0&)
Set OraDyn = OraDb.CreateDynaset(
    "SELECT * FROM Multimedia_tab ORDER BY clip_id", ORADYN_DEFAULT)
Set OraSound1 = OraDyn.Fields("Sound").Value

Set OraSoundClone = OraSound1.Clone

' 次の行に進み、LOB をコピーします。

OraDyn.MoveNext

OraDyn.Edit
OraSound1.Copy OraSoundClone, OraSoundClone.Size, 1, 1
OraDyn.Update

```

Java (JDBC) : 別の LOB への LOB の全体または一部のコピー

```

import java.io.OutputStream;

// コア JDBC クラス :
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 固有の JDBC クラス :
import oracle.sql.*;
import oracle.jdbc.driver.*;

public class Ex2_100

```

```
{

public static void main (String args [])
    throws Exception
{
    // Oracle JDBC Driverをロードします。
    DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

    // データベースに接続します。
    Connection conn =
        DriverManager.getConnection ("jdbc:oracle:oci8:@", "samp", "samp");

    // 自動コミットがオフの場合、高速になります。
    conn.setAutoCommit (false);

    // 文を作成します。
    Statement stmt = conn.createStatement ();
    try
    {
        final int AMOUNT_TO_COPY = 2000;
        ResultSet rset = null;
        BLOB dest_loc = null;
        BLOB src_loc = null;
        InputStream in = null;
        OutputStream out = null;
        byte[] buf = new byte[AMOUNT_TO_COPY];
        rset = stmt.executeQuery (
            "SELECT sound FROM multimedia_tab WHERE clip_id = 1");
        if (rset.next())
        {
            src_loc = ((OracleResultSet)rset).getBLOB (1);
        }
        in = src_loc.getBinaryStream();

        rset = stmt.executeQuery (
            "SELECT sound FROM multimedia_tab WHERE clip_id = 2 FOR UPDATE");
        if (rset.next())
        {
            dest_loc = ((OracleResultSet)rset).getBLOB (1);
        }
        out = dest_loc.getBinaryOutputStream();

        // オフセット0 (ゼロ) から開始して、ストリームからバッファに
        // AMOUNT_TO_COPYバイト読み込みます。
        in.read(buf, 0, AMOUNT_TO_COPY);
    }
}
```

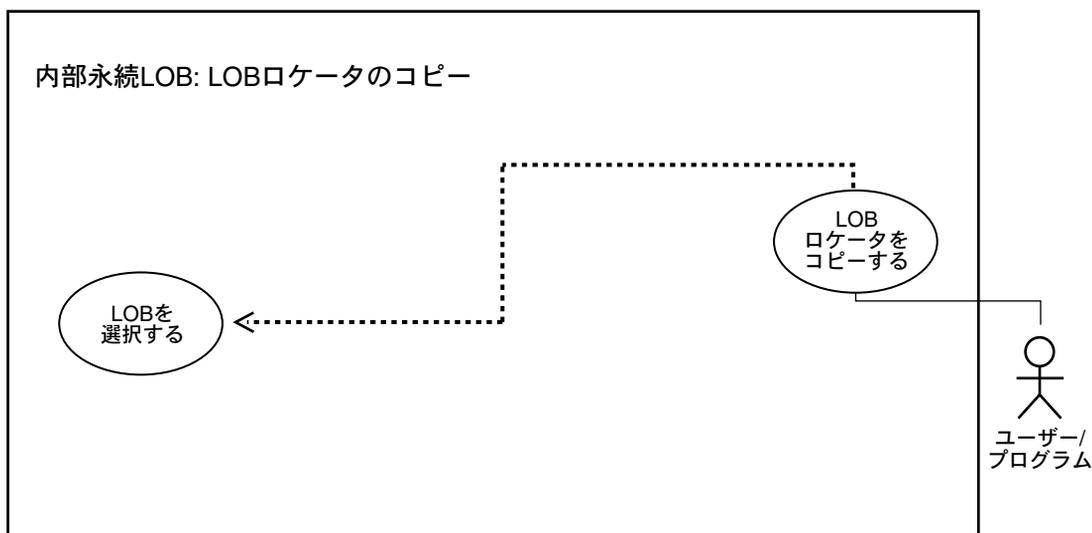
```
// オフセット 0 (ゼロ) から開始して、バッファからアウトプット・ストリームへ
// AMOUNT_TO_COPY バイト書き込みます。
out.write(buf, 0, AMOUNT_TO_COPY);

// すべてのストリームおよびハンドルをクローズします。
in.close();
out.flush();
out.close();
stmt.close();
conn.commit();
conn.close();

}
catch (SQLException e)
{
    e.printStackTrace();
}
}
```

LOB ロケータのコピー

図 9-28 ユースケース図 : LOB ロケータのコピー



参照: 内部永続 LOB に関するすべての基本操作については、9-2 ページの「ユースケース・モデル:内部永続 LOB - 基本操作」を参照してください。

用途

LOB ロケータをコピーします。

使用上の注意

ありません。

構文

各プログラム環境で使用可能な機能のリストは、第3章「LOB プログラム環境」を参照してください。各プログラム環境における構文については、次のマニュアルの項を参照してください。

- PL/SQL (DBM_LOB パッケージ) : 1つのLOB ロケータの別のLOB ロケータへの割当ての詳細は、5-2 ページの「[読取り一貫性のあるロケータ](#)」
- C (OCI) : 『Oracle8i コール・インタフェース・プログラマーズ・ガイド』の第15章「OCI リレーショナル関数」の「LOB 関数」の OCILobAssign()、OCILobIsEqual()
- COBOL (Pro*COBOL) : 『Oracle8i Pro*COBOL プリコンパイラ・プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「ALLOCATE (実行可能埋込み SQL 拡張要素)」、「LOB ASSIGN (実行可能埋込み SQL 拡張要素)」
- C/C++ (Pro*C/C++) : 『Oracle8i Pro*C/C++ プリコンパイラ・プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「SELECT (実行可能埋込み SQL)」、「LOB ASSIGN (実行可能埋込み SQL 拡張要素)」
- Visual Basic (Oracle Objects for OLE オンライン・ヘルプ) : 「Help」の「Contents」タブから、「OO4O Automation Server」>「OBJECTS」>「Orablob」>「METHODS」>「copy」を選択してください。
- Java (JDBC) : 『Oracle8i JDBC 開発者ガイドおよびリファレンス』の第7章「LOB と BFILE の操作」の「BLOB と CLOB の操作」の「BLOB または CLOB 列の作成と移入」、および『Oracle8i SQLJ 開発者ガイドおよびリファレンス』の第5章「型のサポート」の「Oracle の拡張型」の「BLOB、CLOB および BFILE のサポート」

使用例

次の例では、1つのロケータを、ビデオ・フレーム (Frame) を含む別のLOB にコピーする方法を説明します。様々なロケータが、同じデータや異なるデータ、また現在のデータや過去のデータを指す様子に注意してください。

例

次のプログラム環境の例が示されています。

- [PL/SQL \(DBMS_LOB パッケージ\) : LOB ロケータのコピー](#) (9-154 ページ)
- [C \(OCI\) : LOB ロケータのコピー](#) (9-154 ページ)
- [COBOL \(Pro*COBOL\) : LOB ロケータのコピー](#) (9-156 ページ)
- [C/C++ \(Pro*C/C++\) : LOB ロケータのコピー](#) (9-157 ページ)
- [Visual Basic \(OO4O\) : LOB ロケータのコピー](#) (9-158 ページ)
- [Java \(JDBC\) : LOB ロケータのコピー](#) (9-158 ページ)

PL/SQL (DBMS_LOB パッケージ) : LOB ロケータのコピー

注意： PL/SQL を使用して1つの LOB を別の LOB に割り当てるには、「:=」符号を使用する必要があります。これは、高度なトピックです。詳細は、5-2 ページの「[読取り一貫性のあるロケータ](#)」を参照してください。

```

/* 例のプロシージャ lobAssign_proc は、DBMS_LOB パッケージの
一部ではないことに注意してください。*/
CREATE OR REPLACE PROCEDURE lobAssign_proc IS
  Lob_loc1  blob;
  Lob_loc2  blob;
BEGIN
  SELECT Frame INTO Lob_loc1 FROM Multimedia_tab where Clip_ID = 1 FOR UPDATE;
  /* Lob_loc1 を Lob_loc2 に割り当て、この時点での LOB の値のコピーを保存します。 */
  Lob_loc2 := Lob_loc1;
  /* Lob_loc1 を介していくつかのデータを LOB に書き込む場合、Lob_loc2 は新しく書き込
まれたデータを参照しませんが、Lob_loc1 は新しいデータを参照します。*/
END;

```

C (OCI) : LOB ロケータのコピー

```

/* ロケータを選択します。*/
sb4 select_lock_frame_locator(Lob_loc, errhp, svchp, stmthp)
OCILobLocator *Lob_loc;
OCIError      *errhp;
OCISvcCtx     *svchp;
OCIStmt       *stmthp;
{
  text *sqlstmt =
    (text *)"SELECT Frame FROM Multimedia_tab WHERE Clip_ID=1 FOR UPDATE";
  OCIDefine *defnp1;
  checkerr (errhp, OCIStmtPrepare(stmthp, errhp, sqlstmt,
    (ub4)strlen((char *)sqlstmt),
    (ub4) OCI_NTV_SYNTAX, (ub4) OCI_DEFAULT));
  checkerr (errhp, OCIDefineByPos(stmthp, &defnp1, errhp, (ub4) 1,
    (dvoid *)&Lob_loc, (sb4) 0,
    (ub2) SQLT_BLOB, (dvoid *) 0,
    (ub2 *) 0, (ub2 *) 0, (ub4) OCI_DEFAULT));
  /* SELECT を実行し、1行フェッチします。*/
  checkerr (errhp, OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
    (CONST OCISnapshot*) 0, (OCISnapshot*) 0,

```

```

                                (ub4) OCI_DEFAULT));
    return (0);
}

void assignLob(envhp, errhp, svchp, stmthp)
OCIEnv      *envhp;
OCIError    *errhp;
OCISvcCtx   *svchp;
OCIStmt     *stmthp;
{
    OCILobLocator *dest_loc, *src_loc;
    boolean       isEqual;

    /* LOB ロケータを割り当てます。 */
    (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &dest_loc,
                              (ub4)OCI_DTYPE_LOB, (size_t) 0, (dvoid **) 0);
    (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &src_loc,
                              (ub4)OCI_DTYPE_LOB, (size_t) 0, (dvoid **) 0);

    /* LOB を選択します。 */
    printf (" select and lock a frame locator\n");
    select_lock_frame_locator(src_loc, errhp, svchp, stmthp); /* ソース・ロケータ */

    /* src_loc を dest_loc に割り当て、この時点での LOB の値のコピーを保存します。
     */
    printf(" assign the src locator to dest locator\n");
    checkerr (errhp, OCILobAssign(envhp, errhp, src_loc, &dest_loc));

    /* Lob_loc1 を介していくつかのデータを LOB に書き込む場合、Lob_loc2 は新しく書き込
     まれたデータを参照しませんが、Lob_loc1 は新しいデータを参照します。
     */

    /* OCI をコールして、2つのロケータが等しいかどうかを確認します。 */

    printf (" check if Lobs are Equal.\n");
    checkerr (errhp, OCILobIsEqual(envhp, src_loc, dest_loc, &isEqual));
    if (isEqual)
    {
        /* LOB ロケータは等しいです。 */
        printf(" Lob Locators are equal.\n");
    }
    else
    {
        /* LOB ロケータは等しくありません。 */
    }
}

```

```
    printf(" Lob Locators are NOT Equal.\n");
}

/* この例ではLOB ロケータは等しいことに注意してください。*/

/* ロケータが保持しているリソースを解放します。*/
(void) OCIDescriptorFree((dvoid *) dest_loc, (ub4) OCI_DTYPE_LOB);
(void) OCIDescriptorFree((dvoid *) src_loc, (ub4) OCI_DTYPE_LOB);
return;
}
```

COBOL (Pro*COBOL) : LOB ロケータのコピー

```
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.

* BLOB ロケータの割当ておよび初期化を行います。
    EXEC SQL ALLOCATE :DEST END-EXEC.
    EXEC SQL ALLOCATE :SRC END-EXEC.
    EXEC SQL WHENEVER NOT FOUND GOTO END-OF-BLOB END-EXEC.
    EXEC SQL
        SELECT FRAME INTO :SRC
        FROM MULTIMEDIA_TAB WHERE CLIP_ID = 2 FOR UPDATE
    END-EXEC.
    EXEC SQL LOB ASSIGN :SRC TO :DEST END-EXEC.

* ソースを介してデータをLOBに書き込む場合、宛先は新しく書き込まれた
* データを参照しません。
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++) : LOB ロケータのコピー

```
#include <oci.h>  
#include <stdio.h>  
#include <sqlca.h>  
  
void Sample_Error()  
{  
    EXEC SQL WHENEVER SQLERROR CONTINUE;  
    printf("%.*s\n", sqlca.sqlerrm.sqlerrml, sqlca.sqlerrm.sqlerrmc);  
    EXEC SQL ROLLBACK WORK RELEASE;  
    exit(1);  
}  
  
void lobAssign_proc()  
{  
    OCIBlobLocator *Lob_loc1, *Lob_loc2;  
  
    EXEC SQL WHENEVER SQLERROR DO Sample_Error();  
    EXEC SQL ALLOCATE :Lob_loc1;  
    EXEC SQL ALLOCATE :Lob_loc2;  
    EXEC SQL SELECT Frame INTO :Lob_loc1  
        FROM Multimedia_tab WHERE Clip_ID = 1 FOR UPDATE;  
    /* Lob_loc1 を Lob_loc2 に割り当て、この時点での LOB の値のコピーを保存します。*/  
    EXEC SQL LOB ASSIGN :Lob_loc1 TO :Lob_loc2;  
    /* Lob_loc1 を介していくつかのデータを LOB に書き込む場合、Lob_loc2 は新しく書き込  
        まれたデータを参照しませんが、Lob_loc1 は新しいデータを参照します。*/  
}
```

```
void main()
{
    char *samp = "samp/samp";
    EXEC SQL CONNECT :samp;
    lobAssign_proc();
    EXEC SQL ROLLBACK WORK RELEASE;
}
```

Visual Basic (0040) : LOB ロケータのコピー

```
Dim MySession As OraSession
Dim OraDb As OraDatabase
Dim OraDyn As OraDynaset, OraSound1 As OraBlob, OraSoundClone As OraBlob

Set MySession = CreateObject("OracleInProcServer.XOraSession")
Set OraDb = MySession.OpenDatabase("exampledb", "samp/samp", 0&)
Set OraDyn = OraDb.CreateDynaset(
    "SELECT * FROM Multimedia_tab ORDER BY clip_id ", ORADYN_DEFAULT)
Set OraSound1 = OraDyn.Fields("Sound").Value
Set OraSoundClone = OraSound1.Clone

OraDyn.MoveNext

'OraSound1 のオフセット 100 で、1,000 バイトの LOB 値 OraSoundClone を
'OraSound1 にコピーします。
OraDyn.Edit
OraSound1.Copy OraSoundClone, 1000, 100
OraDyn.Update
```

Java (JDBC) : LOB ロケータのコピー

```
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 固有の JDBC クラス :
import oracle.sql.*;
import oracle.jdbc.driver.*;
```

```
public class Ex2_104
{
public static void main (String args [])
    throws Exception
    {
    // Oracle JDBC Driver をロードします。
    DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

    // データベースに接続します。
    Connection conn =
    DriverManager.getConnection ("jdbc:oracle:oci8:@", "samp", "samp");

    // 自動コミットがオフの場合、高速になります。
    conn.setAutoCommit (false);

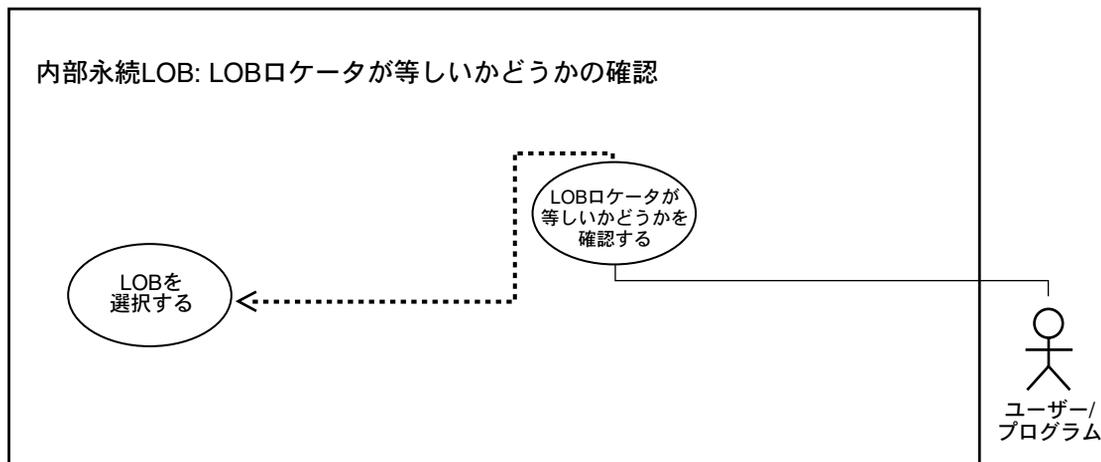
    // 文を作成します。
    Statement stmt = conn.createStatement ();
try
    {
    BLOB lob_loc1 = null;
    BLOB lob_loc2 = null;
    ResultSet rset = stmt.executeQuery (
        "SELECT frame FROM multimedia_tab WHERE clip_id = 1");
    if (rset.next())
    {
        lob_loc1 = ((OracleResultSet)rset).getBLOB (1);
    }

    // lob_loc1 を介してデータを LOB に書き込む場合、lob_loc2 は変更を参照しません。
    lob_loc2 = lob_loc1;
    stmt.close();
    conn.commit();
    conn.close();

    }
    catch (SQLException e)
    {
        e.printStackTrace();
    }
}
}
```

2つのLOBロケータが等しいかどうかの確認

図 9-29 ユースケース図：2つのLOBロケータが等しいかどうかの確認



参照： 内部永続LOBに関するすべての基本操作については、9-2 ページの「ユースケース・モデル: 内部永続LOB - 基本操作」を参照してください。

用途

2つのロケータが等しいかどうかを確認します。

使用上の注意

ありません。

構文

各プログラム環境で使用可能な機能のリストは、第3章「LOB プログラム環境」を参照してください。各プログラム環境における構文については、次のマニュアルの項を参照してください。

- PL/SQL (DBMS_LOB パッケージ) : 参照マニュアルはありません。
- C (OCI) : 『Oracle8i コール・インタフェース・プログラマーズ・ガイド』の第15章「OCIリレーショナル関数」の「LOB関数」の OCILobAssign(), OCILobIsEqual()
- COBOL (Pro*COBOL) : 参照マニュアルはありません。
- C/C++ (Pro*C/C++) : 『Oracle8i Pro*C/C++ プリコンパイラ・プログラマーズ・ガイド』の付録F「埋込みSQL文およびプリコンパイラ宣言文」の「LOB ASSIGN (実行可能埋込みSQL拡張要素)」
- Visual Basic (OO4O) : 参照マニュアルはありません。
- Java (JDBC) : 『Oracle8i JDBC 開発者ガイドおよびリファレンス』の第7章「LOBとBFILEの操作」の「BLOBとCLOBの操作」の「BLOBまたはCLOB列の作成と移入」、および『Oracle8i SQLJ 開発者ガイドおよびリファレンス』の第5章「型のサポート」の「Oracleの拡張型」の「BLOB、CLOBおよびBFILEのサポート」

使用例

2つのロケータが等しい場合、それらが同じバージョンのLOBデータを参照していることを意味します (5-2 ページの「読取り一貫性のあるロケータ」を参照)。次の例では、ロケータは同等です。ただし、ロケータがLOBデータの同じバージョンを参照していないと判断することが重要です。

例

次のプログラム環境の例が示されています。

- PL/SQL (DBMS_LOB) : 今回のリリースでは例は記載されていません。
- C (OCI) : 2つのLOBロケータが等しいかどうかの確認 (9-161 ページ)
- COBOL (Pro*COBOL) : 今回のリリースでは例は記載されていません。
- C/C++ (Pro*C/C++) : 2つのLOBロケータが等しいかどうかの確認 (9-163 ページ)
- Visual Basic (OO4O) : 今回のリリースでは例は記載されていません。
- Java (JDBC) : 2つのLOBロケータが等しいかどうかの確認 (9-164 ページ)

C (OCI) : 2つのLOBロケータが等しいかどうかの確認

```
/* ロケータを選択します。*/
sb4 select_lock_frame_locator(Lob_loc, errhnp, svchp, stmthp)
OCILobLocator *Lob_loc;
```

```
OCIError      *errhp;
OCISvcCtx     *svchp;
OCIStmt      *stmthp;
{
    text *sqlstmt =
        (text *)"SELECT Frame FROM Multimedia_tab WHERE Clip_ID=1 FOR UPDATE";
    OCIDefine *defnp1;
    checkerr (errhp, OCIStmtPrepare(stmthp, errhp, sqlstmt,
                                     (ub4)strlen((char *)sqlstmt),
                                     (ub4) OCI_NTV_SYNTAX, (ub4) OCI_DEFAULT));
    checkerr (errhp, OCIDefineByPos(stmthp, &defnp1, errhp, (ub4) 1,
                                     (dvoid *)&Lob_loc, (sb4) 0,
                                     (ub2) SQLT_BLOB, (dvoid *) 0,
                                     (ub2 *) 0, (ub2 *) 0, (ub4) OCI_DEFAULT));

    /* SELECT を実行し、1行フェッチします。*/
    checkerr(errhp, OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                                   (CONST OCISnapshot*) 0, (OCISnapshot*) 0,
                                   (ub4) OCI_DEFAULT));

    return (0);
}

void locatorIsEqual(envhp, errhp, svchp, stmthp)
OCIEnv      *envhp;
OCIError    *errhp;
OCISvcCtx   *svchp;
OCIStmt     *stmthp;
{
    OCILobLocator *dest_loc, *src_loc;
    boolean       isEqual;

    /* LOB ロケータを割り当てます。*/
    (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &dest_loc,
                              (ub4)OCI_DTYPE_LOB, (size_t) 0, (dvoid **) 0);
    (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &src_loc,
                              (ub4)OCI_DTYPE_LOB, (size_t) 0, (dvoid **) 0);

    /* LOB を選択します。*/
    printf (" select and lock a frame locator\n");
    select_lock_frame_locator(src_loc, errhp, svchp, stmthp); /* source locator */

    /* src_loc を dest_loc に割り当て、この時点でのLOBの値のコピーを保存します。*/
    printf(" assign the src locator to dest locator\n");
    checkerr (errhp, OCILobAssign(envhp, errhp, src_loc, &dest_loc));
}
```

```

/* Lob_loc1 を介していくつかのデータをLOBに書き込む場合、Lob_loc2 は新しく書き込まれたデータを参照しませんが、Lob_loc1 は新しいデータを参照します。*/

/* OCI をコールして、2つのロケータが等しいかどうかを確認します。*/

printf(" check if Lobs are Equal.\n");
checkerr( errhp, OCILobIsEqual( envhp, src_loc, dest_loc, &isEqual));

if (isEqual)
{
    /* LOB ロケータは等しいです。*/
    printf(" Lob Locators are equal.\n");
}
else
{
    /* LOB ロケータは等しくありません。*/
    printf(" Lob Locators are NOT Equal.\n");
}

/* この例では、LOB ロケータは等しいことに注意してください。*/

/* ロケータが保持しているリソースを解放します。*/
(void) OCIDescriptorFree((dvoid *) dest_loc, (ub4) OCI_DTYPE_LOB);
(void) OCIDescriptorFree((dvoid *) src_loc, (ub4) OCI_DTYPE_LOB);

return;
}

```

C/C++ (Pro*C/C++) : 2つのLOBロケータが等しいかどうかの確認

/* Pro*C/C++ では、2つのロケータが等しいかどうかをテストするメカニズムが提供されません。ただし、OCIを直接使用することによって、この例で示すように2つのロケータを比較し、これらが等しいかどうかを判断できます。*/

```

#include <sql2oci.h>
#include <stdio.h>
#include <sqlca.h>

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

```

```
        exit(1);
    }

void LobLocatorIsEqual_proc()
{
    OCIBlobLocator *Lob_loc1, *Lob_loc2;
    OCIEnv *oeh;
    boolean isEqual;
    EXEC SQL WHENEVER SQLERROR DO Sample_Error();
    EXEC SQL ALLOCATE :Lob_loc1;
    EXEC SQL ALLOCATE :Lob_loc2;
    EXEC SQL SELECT Frame INTO Lob_loc1
        FROM Multimedia_tab where Clip_ID = 1 FOR UPDATE;
    /* Lob_loc1 を Lob_loc2 に割り当て、この時点でのLOBの値のコピーを保存します。*/
    EXEC SQL LOB ASSIGN :Lob_loc1 TO :Lob_loc2;
    /* Lob_loc1 を介していくつかのデータをLOBに書き込む場合、Lob_loc2 は新しく書き込
       まれたデータを参照しませんが、Lob_loc1 は新しいデータを参照します。*/
    /* SQLLIB ルーチンを使用して、OCI環境ハンドルを取得します。*/
    (void) SQLEnvGet(SQL_SINGLE_RCTX, &oeh);
    /* OCI をコールして、2つのロケータが等しいかどうかを確認します。*/
    (void) OCILobIsEqual(oeh, Lob_loc1, Lob_loc2, &isEqual);
    if (isEqual)
        printf("The locators are equal\n");
    else
        printf("The locators are not equal\n");
    /* この例では、LOBロケータは等しいことに注意してください。*/
    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) : 2つのLOBロケータが等しいかどうかの確認

```
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固有のJDBCクラス :
import oracle.sql.*;
import oracle.jdbc.driver.*;

public class Ex2_108
{
    public static void main (String args [])
        throws Exception
    {
        // Oracle JDBC Driverをロードします。
        DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

        // データベースに接続します。
        Connection conn =
            DriverManager.getConnection ("jdbc:oracle:oci8:@", "samp", "samp");

        // 自動コミットがオフの場合、高速になります。
        conn.setAutoCommit (false);

        // 文を作成します。
        Statement stmt = conn.createStatement ();
        try
        {
            BLOB lob_loc1 = null;
            BLOB lob_loc2 = null;
            ResultSet rset = stmt.executeQuery (
                "SELECT sound FROM multimedia_tab WHERE clip_id = 2");
            if (rset.next())
            {
                lob_loc1 = ((OracleResultSet)rset).getBLOB (1);
            }

            // lob_loc1を介してデータをLOBに書き込む場合、lob_loc2は変更を参照しません。
            lob_loc2 = lob_loc1;

            // この例では、ロケータは等しいことに注意してください。
            if (lob_loc1.equals(lob_loc2))
            {
                // ロケータは等しいです。
            }
        }
    }
}
```

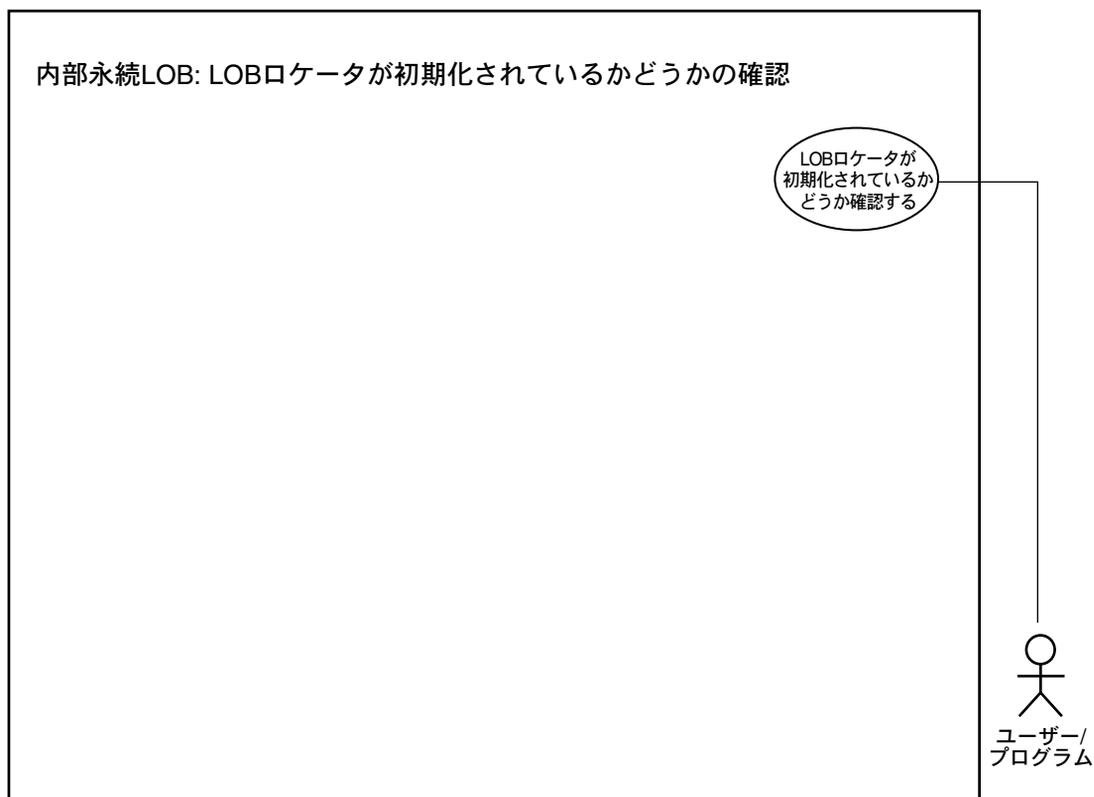
```
        System.out.println("Locators are equal");
    }
    else
    {
        // ロケータは異なります。
        System.out.println("Locators are NOT equal");
    }

    stmt.close();
    conn.commit();
    conn.close();

    }
    catch (SQLException e)
    {
        e.printStackTrace();
    }
}
}
```

LOB ロケータが初期化されているかどうかの確認

図 9-30 ユースケース図：LOB ロケータが初期化されているかどうかの確認



参照： 内部永続LOBに関するすべての基本操作については、9-2 ページの「ユースケース・モデル：内部永続LOB－基本操作」を参照してください。

用途

LOB ロケータが初期化されているかどうかを確認します。

使用上の注意

ありません。

構文

各プログラム環境で使用可能な機能のリストは、[第3章「LOB プログラム環境」](#)を参照してください。各プログラム環境における構文については、次のマニュアルの項を参照してください。

- PL/SQL (DBMS_LOB パッケージ) : 参照マニュアルはありません。
- C (OCI) : 『Oracle8i コール・インタフェース・プログラマーズ・ガイド』の第15章「OCI リレーショナル関数」の「LOB 関数」の OCILobLocatorIsInit()
- COBOL (Pro*COBOL) : 参照マニュアルはありません。
- C/C++ (Pro*C/C++) : 『Oracle8i Pro*C/C++ プリコンパイラ・プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびプリコンパイラ宣言文」、および『Oracle8i コール・インタフェース・プログラマーズ・ガイド』の第15章「OCI リレーショナル関数」の「LOB 関数」の OCILobLocatorIsInit()
- Visual Basic (OO4O) : 参照マニュアルはありません。
- Java (JDBC) : 参照マニュアルはありません。

使用例

この操作は、ロケータが初期化されているかどうかを判断します。次の例では、両方のロケータが初期化されています。

例

次のプログラム環境の例が示されています。

- PL/SQL (DBMS_LOB パッケージ) : 今回のリリースでは例は記載されていません。
- C (OCI) : [LOB ロケータが初期化されているかどうかの確認 \(9-169 ページ\)](#)
- COBOL (Pro*COBOL) : 今回のリリースでは例は記載されていません。
- C/C++ (Pro*C/C++) : [LOB ロケータが初期化されているかどうかの確認 \(9-170 ページ\)](#)
- Visual Basic (OO4O) : 今回のリリースでは例は記載されていません。
- Java (JDBC) : 今回のリリースでは例は記載されていません。

C (OCI) : LOB ロケータが初期化されているかどうかの確認

```

/* ロケータを選択します。*/
sb4 select_frame_locator(Lob_loc, errhp, svchp, stmthp)
OCILobLocator *Lob_loc;
OCIError      *errhp;
OCISvcCtx     *svchp;
OCIStmt       *stmthp;
{
    text      *sqlstmt =
        (text *) "SELECT Frame FROM Multimedia_tab WHERE Clip_ID=1";
    OCIDefine *defnp1;
    checkerr (errhp, OCIStmtPrepare(stmthp, errhp, sqlstmt,
                                    (ub4)strlen((char *)sqlstmt),
                                    (ub4)OCI_NTV_SYNTAX, (ub4)OCI_DEFAULT));
    checkerr (errhp, OCIDefineByPos(stmthp, &defnp1, errhp, (ub4) 1,
                                    (dvoid *)&Lob_loc, (sb4) 0,
                                    (ub2) SQLT_BLOB, (dvoid *) 0,
                                    (ub2 *) 0, (ub2 *) 0, (ub4) OCI_DEFAULT));

    /* SELECT を実行し、1行フェッチします。*/
    checkerr (errhp, OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                                    (CONST OCISnapshot*) 0, (OCISnapshot*) 0,
                                    (ub4) OCI_DEFAULT));

    return (0);
}

void isInitializedLob(envhp, errhp, svchp, stmthp)
OCIEnv      *envhp;
OCIError    *errhp;
OCISvcCtx   *svchp;
OCIStmt     *stmthp;
{
    OCILobLocator *Lob_loc1, *Lob_loc2;
    boolean      isInitialized;

    /* LOB ロケータを割り当てます。*/
    printf(" allocate locator 1 and 2\n");
    (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &Lob_loc1,
                              (ub4)OCI_DTYPE_LOB, (size_t) 0, (dvoid **) 0);
    (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &Lob_loc2,
                              (ub4)OCI_DTYPE_LOB, (size_t) 0, (dvoid **) 0);

    /* LOB を選択します。*/
    printf (" select a frame locator into locator 1\n");

```

```

select_frame_locator(Lob_loc1, errhp, svchp, stmthp);          /* ロケータ 1 */

/* ロケータ 1 が初期化されているかどうかを判断します。*/
checkerr(errhp, OCILobLocatorIsInit(envhp, errhp, Lob_loc1, &isInitialized));
/* IsInitialized の場合、TRUE が戻ります。*/
printf(" for Locator 1, isInitialized = %d\n", isInitialized);

/* ロケータ 2 が初期化されているかどうかを判断します。*/
checkerr(errhp, OCILobLocatorIsInit(envhp, errhp, Lob_loc2, &isInitialized));
/* IsInitialized の場合、FALSE が戻ります。*/
printf(" for Locator 2, isInitialized = %d\n", isInitialized);

/* ロケータが保持しているリソースを解放します。*/
(void) OCIDescriptorFree((dvoid *) Lob_loc1, (ub4) OCI_DTYPE_LOB);
(void) OCIDescriptorFree((dvoid *) Lob_loc2, (ub4) OCI_DTYPE_LOB);
return;
}

```

C/C++ (Pro*C/C++) : LOB ロケータが初期化されているかどうかの確認

/* Pro*C/C++ には、LOB ロケータが初期化されているかどうかを判断するための埋込み SQL 文のフォームがありません。Pro*C/C++ でのロケータは、EXEC SQL ALLOCATE 文を介して割り当てられた場合に初期化されます。ただし、次に示すように、埋込み SQL および OCI を使用して例を書くことができます。*/

```

#include <sql2oci.h>
#include <stdio.h>
#include <sqlca.h>

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

void LobLocatorIsInit_proc()
{
    OCIBlobLocator *Lob_loc;
    OCIEnv *oeh;
    OCIError *err;
    boolean isInitialized;

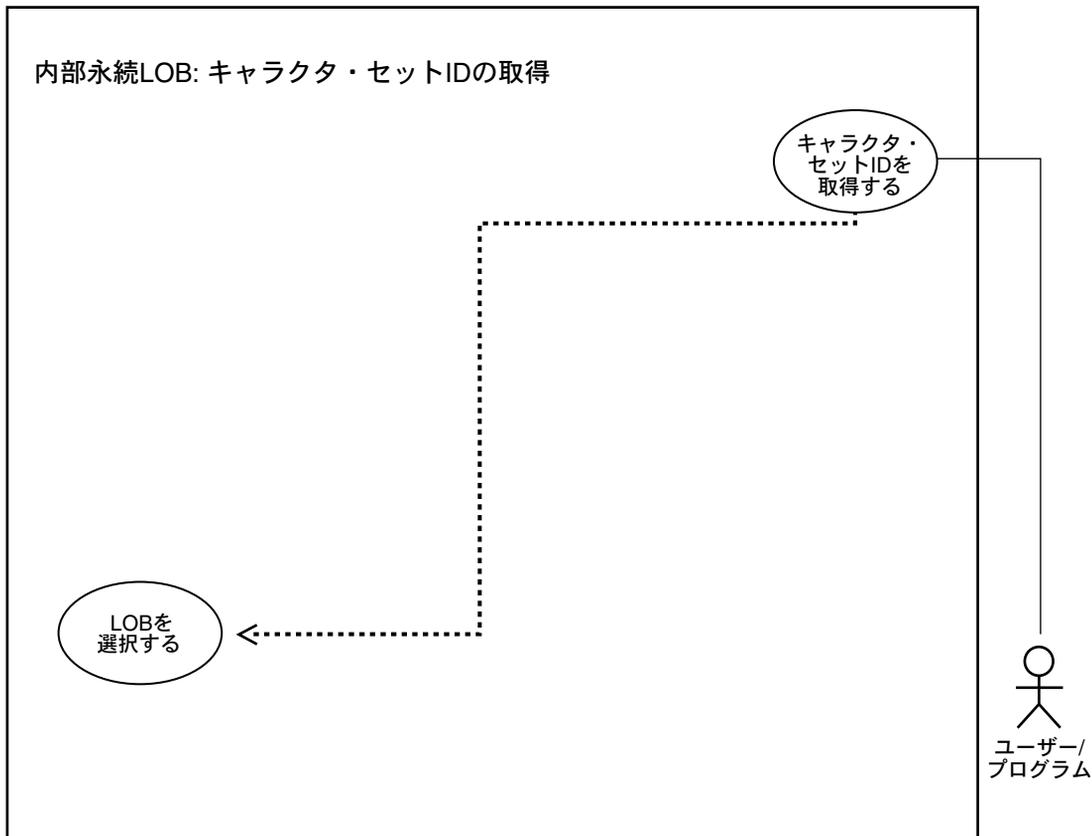
```

```
EXEC SQL WHENEVER SQLERROR DO Sample_Error();
EXEC SQL ALLOCATE :Lob_loc;
EXEC SQL SELECT Frame INTO Lob_loc
        FROM Multimedia_tab where Clip_ID = 1;
/* SQLLIB ルーチンを使用して、OCI 環境ハンドルを取得します。*/
(void) SQLEnvGet(SQL_SINGLE_RCTX, &oeh);
/* OCI エラー・ハンドルを割り当てます。*/
(void) OCIHandleAlloc((dvoid *)oeh, (dvoid **)&err,
        (ub4)OCI_HTYPE_ERROR, (ub4)0, (dvoid **)0);
/* OCI を使用して、ロケータが初期化されているかどうかを判断します。*/
(void) OCILobLocatorIsInit(oeh, err, Lob_loc, &isInitialized);
if (isInitialized)
    printf("The locator is initialized\n");
else
    printf("The locator is not initialized\n");
/* この例では、ロケータは初期化されていることに注意してください。*/
/* OCI エラー・ハンドルの割当てを解除します。*/
(void) OCIHandleFree(err, OCI_HTYPE_ERROR);
/* ロケータが保持しているリソースを解放します。*/
EXEC SQL FREE :Lob_loc;
}

void main()
{
    char *samp = "samp/samp";
    EXEC SQL CONNECT :samp;
    LobLocatorIsInit_proc();
    EXEC SQL ROLLBACK WORK RELEASE;
}
```

キャラクタ・セット ID の取得

図 9-31 ユースケース図: キャラクタ・セット ID の取得



用途

キャラクタ・セット ID を取得します。

使用上の注意

ありません。

構文

各プログラム環境で使用可能な機能のリストは、第3章「LOB プログラム環境」を参照してください。各プログラム環境における構文については、次のマニュアルの項を参照してください。

- PL/SQL (DBMS_LOB パッケージ) : 参照マニュアルはありません。
- C (OCI) : 『Oracle8i コール・インタフェース・プログラマーズ・ガイド』の第15章「OCI リレーショナル関数」の「LOB 関数」の OCILobCharSetId(), GetCsidLob()
- COBOL (Pro*COBOL) : 参照マニュアルはありません。
- C/C++ (Pro*C/C++) : 参照マニュアルはありません。
- Visual Basic (OO4O) : 参照マニュアルはありません。
- Java (JDBC) : 参照マニュアルはありません。

使用例

次の例では、外国語のサブタイトル (FLSub) のキャラクタ・セット ID を判断する方法を説明します。

例

この機能は OCI でのみ使用可能です。

- PL/SQL (DBMS_LOB パッケージ) : 今回のリリースでは例は記載されていません。
- **C (OCI) : キャラクタ・セット ID の取得** (9-173 ページ)
- COBOL (Pro*COBOL) : 今回のリリースでは例は記載されていません。
- C/C++ (Pro*C/C++) : 今回のリリースでは例は記載されていません。
- Visual Basic (OO4O) : 今回のリリースでは例は記載されていません。
- Java (JDBC) : 今回のリリースでは例は記載されていません。

C (OCI) : キャラクタ・セット ID の取得

```
/* このファンクションによって、有効な LOB ロケータがとられ、LOB のキャラクタ・セット ID
   が印刷されます。*/
```

```
/* ロケータを選択します。*/
```

```
sb4 select FLSub_locator(Lob_loc, errhp, svchp, stmthp)
OCILobLocator *Lob_loc;
```

```

OCIError      *errhp;
OCISvcCtx     *svchp;
OCIStmt      *stmthp;
{
    OCIDefine *defnp1;
    text *sqlstmt =
        (text *)"SELECT FLSub FROM Multimedia_tab WHERE Clip_ID = 2";
    checkerr (errhp, OCIStmtPrepare(stmthp, errhp, sqlstmt,
                                    (ub4)strlen((char *)sqlstmt),
                                    (ub4)OCI_NTV_SYNTAX, (ub4)OCI_DEFAULT));
    /* 選択されている列を定義します。*/
    checkerr (errhp, OCIDefineByPos(stmthp, &defnp1, errhp, (ub4) 1,
                                    (dvoid *)&Lob_loc, (sb4)0,
                                    (ub2)SQLT_CLOB, (dvoid *)0, (ub2 *)0,
                                    (ub2 *)0, (ub4)OCI_DEFAULT));
    /* SELECT を実行し、1 行フェッチします。*/
    checkerr (errhp, OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                                    (CONST OCISnapshot*) 0, (OCISnapshot*) 0,
                                    (ub4) OCI_DEFAULT));

    return 0;
}
sb4 getcsidLob (envhp, errhp, svchp, stmthp)
OCIEnv      *envhp;
OCIError    *errhp;
OCISvcCtx   *svchp;
OCIStmt     *stmthp;
{
    OCILobLocator *Lob_loc;
    ub2 charsetid =0 ;
    (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &Lob_loc,
                              (ub4)OCI_DTYPE_LOB, (size_t) 0, (dvoid **) 0);
    printf (" select a FLSub locator\n");
    select_FLSub_locator(Lob_loc, errhp, svchp, stmthp);
    printf (" get the character set id of FLSub_locator\n");

    /* LOB のキャラクタ・セット ID を取得します。*/
    checkerr (errhp, OCILobCharSetId(envhp, errhp, Lob_loc, &charsetid));
    printf(" character Set ID of FLSub is : %d\n", charsetid);

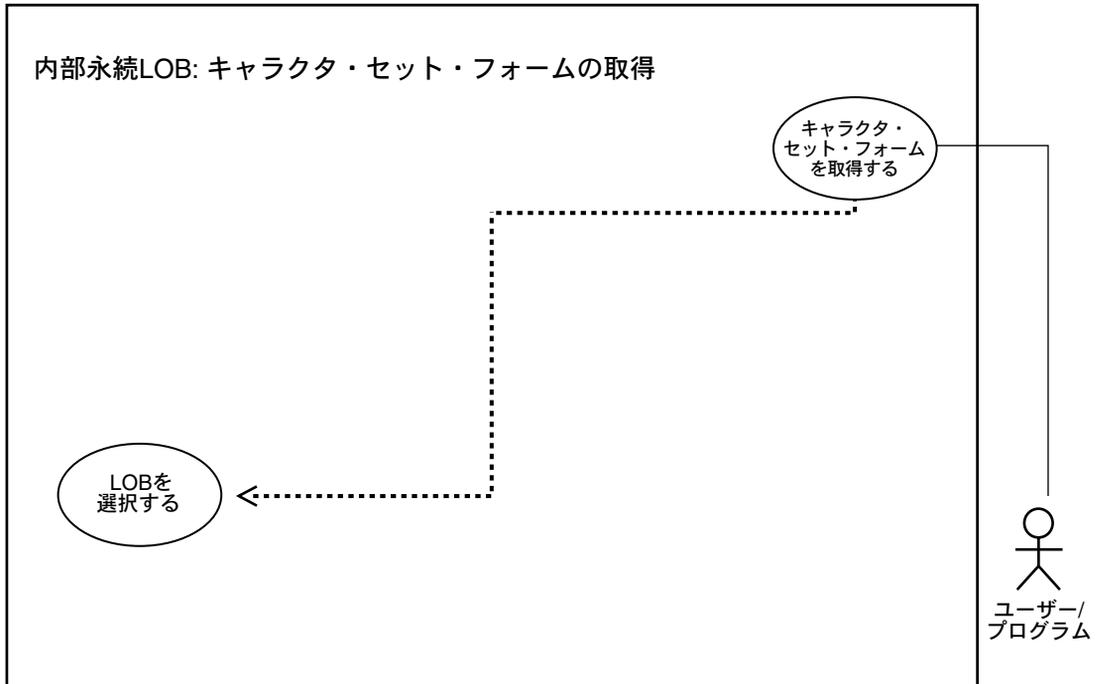
    /* ロケータが保持しているリソースを解放します。*/
    (void) OCIDescriptorFree((dvoid *) Lob_loc, (ub4) OCI_DTYPE_LOB);

    return;
}

```

キャラクタ・セット・フォームの取得

図 9-32 ユースケース図: キャラクタ・セット・フォームの取得



参照: 内部永続 LOB に関するすべての基本操作については、9-2 ページの「ユースケース・モデル: 内部永続 LOB - 基本操作」を参照してください。

用途

キャラクタ・セット・フォームを取得します。

使用上の注意

ありません。

構文

各プログラム環境で使用可能な機能のリストは、[第3章「LOB プログラム環境」](#)を参照してください。各プログラム環境における構文については、次のマニュアルの項を参照してください。

- PL/SQL (DBMS_LOB パッケージ) : 参照マニュアルはありません。
- C (OCI) : 『Oracle8i コール・インタフェース・プログラマーズ・ガイド』の第15章「OCI リレーショナル関数」の「LOB 関数」の OCILobCharSetForm()
- COBOL (Pro*COBOL) : 参照マニュアルはありません。
- C/C++ (Pro*C/C++) : 参照マニュアルはありません。
- Visual Basic (OO4O) : 参照マニュアルはありません。
- Java (JDBC) : 参照マニュアルはありません。

使用例

次の使用例では、外国語のサブタイトル (FLSub) のキャラクタ・セット・フォームを判断する方法を説明します。

例

この機能は OCI でのみ使用可能です。

- PL/SQL (DBMS_LOB パッケージ) : 今回のリリースでは例は記載されていません。
- [C \(OCI\) : キャラクタ・セット・フォームの取得 \(9-176 ページ\)](#)
- COBOL (Pro*COBOL) : 今回のリリースでは例は記載されていません。
- C/C++ (Pro*C/C++) : 今回のリリースでは例は記載されていません。
- Visual Basic (OO4O) : 今回のリリースでは例は記載されていません。
- Java (JDBC) : 今回のリリースでは例は記載されていません。

C (OCI) : キャラクタ・セット・フォームの取得

```

/* ロケータを選択します。*/
sb4 select_FLSub_locator(Lob_loc, errhp, svchp, stmthp)
OCILobLocator *Lob_loc;
OCIError      *errhp;
OCISvcCtx     *svchp;
OCIStmt       *stmthp;
{
    OCIDefine *defnp1;
    text *sqlstmt =

```

```

        (text *)"SELECT FLSub FROM Multimedia_tab WHERE Clip_ID = 2";
checkerr (errhp, OCIStmtPrepare(stmthp, errhp, sqlstmt,
                                (ub4)strlen((char *)sqlstmt),
                                (ub4)OCI_NTV_SYNTAX, (ub4)OCI_DEFAULT));

/* 選択されている列を定義します。*/
checkerr (errhp, OCIDefineByPos(stmthp, &defnp1, errhp, (ub4) 1,
                                (dvoid *)&Lob_loc, (sb4) 0,
                                (ub2)SQLT_CLOB, (dvoid *) 0, (ub2 *) 0,
                                (ub2 *) 0, (ub4)OCI_DEFAULT));

/* SELECT を実行し、1 行フェッチします。*/
checkerr (errhp, OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                                (CONST OCISnapshot*) 0, (OCISnapshot*) 0,
                                (ub4) OCI_DEFAULT));

return 0;
}

/* このファンクションによって、有効な LOB ロケータがとられ、LOB のキャラクタ・セット・フォーム
   が印刷されます。
   */
sb4 getcsformLob(envhp, errhp, svchp, stmthp)
OCIEnv      *envhp;
OCIError    *errhp;
OCISvcCtx   *svchp;
OCIStmt     *stmthp;
{
    OCILobLocator *Lob_loc;
    ub1 charset_form = 0 ;

    (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &Lob_loc,
                              (ub4)OCI_DTYPE_LOB, (size_t) 0, (dvoid **) 0);

    printf (" select a FLSub locator\n");
    select_FLSub_locator(Lob_loc, errhp, svchp, stmthp);
    printf (" get the character set form of FLSub\n");

    /* LOB のキャラクタ・セット ID を取得します。*/
    checkerr (errhp, OCILobCharSetForm(envhp, errhp, Lob_loc, &charset_form));
    printf(" character Set Form of FLSub is : %d\n", charset_form);

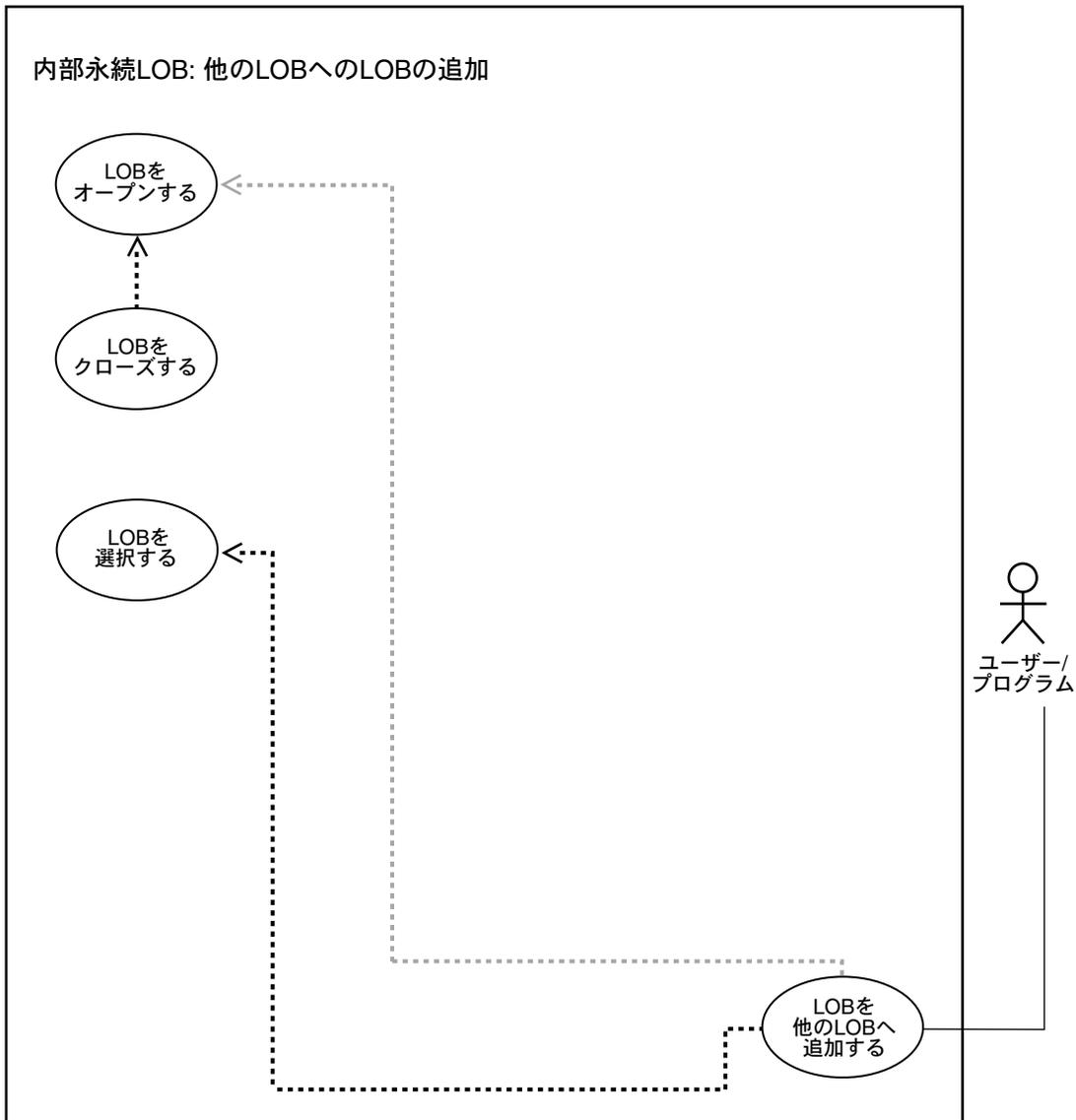
    /* ロケータが保持しているリソースを解放します。*/
    (void) OCIDescriptorFree((dvoid *) Lob_loc, (ub4) OCI_DTYPE_LOB);

    return;
}

```

他のLOBへのLOBの追加

図 9-33 ユースケース図：他のLOBへのLOBの追加



参照： 内部永続 LOB に関するすべての基本操作については、9-2 ページの「[ユースケース・モデル:内部永続 LOB – 基本操作](#)」を参照してください。

用途

LOB を他の LOB に追加します。

使用上の注意

更新前の行のロック

PL/SQL DBMS_LOB パッケージまたは OCI を使用して LOB 値を更新する前に、LOB を含む行をロックする必要があります。SQL INSERT 文および UPDATE 文は暗黙的に行をロックしますが、SQL プログラムおよび PL/SQL プログラムでは SQL SELECT FOR UPDATE 文を使用して、また OCI プログラムでは `oci pin` または `lock` 関数を使用して明示的にロックします。更新後のロケータの状態の詳細は、5-6 ページの「[更新済ロケータを介した更新済 LOB](#)」を参照してください。

構文

各プログラム環境で使用可能な機能のリストは、[第 3 章「LOB プログラム環境](#)」を参照してください。各プログラム環境における構文については、次のマニュアルの項を参照してください。

- PL/SQL (DBMS_LOB パッケージ) : 『Oracle8i PL/SQL パッケージ・プロシージャ・リファレンス』の第 18 章「DBMS_LOB」の「サブプログラムの要約」の APPEND プロシージャ
- C (OCI) : 『Oracle8i コール・インタフェース・プログラマーズ・ガイド』の第 15 章「OCI リレーショナル関数」の「LOB 関数」の OCILobAppend()
- COBOL (Pro*COBOL) : 『Oracle8i Pro*COBOL プリコンパイラ・プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB APPEND (実行可能埋込み SQL 拡張要素)」
- C/C++ (Pro*C/C++) : 『Oracle8i Pro*C/C++ プリコンパイラ・プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB APPEND (実行可能埋込み SQL 拡張要素)」
- Visual Basic (Oracle Objects for OLE オンライン・ヘルプ) : 「Help」の「Contents」タブから、「OO4O Automation Server」>「OBJECTS」>「Orablob」>「METHODS」>「append」を選択してください。

- Java (JDBC) : 『Oracle8i JDBC 開発者ガイドおよびリファレンス』の第7章「LOB と BFILE の操作」の「BLOB と CLOB の操作」の「BLOB または CLOB 列の作成と移入」、および『Oracle8i SQLJ 開発者ガイドおよびリファレンス』の第5章「型のサポート」の「Oracle の拡張型」の「BLOB、CLOB および BFILE のサポート」

使用例

次の例では、Sound の 1 つのセグメントを別のセグメントに追加します。波形の処理が可能なサウンド編集ツールの使用を想定しています。

例

次のプログラム環境の例が示されています。

- PL/SQL (DBMS_LOB パッケージ) : 他の LOB への LOB の追加 (9-180 ページ)
- C (OCI) : 他の LOB への LOB の追加 (9-181 ページ)
- COBOL (Pro*COBOL) : 他の LOB への LOB の追加 (9-183 ページ)
- C/C++ (Pro*C/C++) : 他の LOB への LOB の追加 (9-184 ページ)
- Visual Basic (OO4O) : 他の LOB への LOB の追加 (9-185 ページ)
- Java (JDBC) : 他の LOB への LOB の追加 (9-185 ページ)

PL/SQL (DBMS_LOB パッケージ) : 他の LOB への LOB の追加

```
/* 例のプロシージャ appendLOB_proc は、DBMS_LOB パッケージの
一部ではないことに注意してください。*/
CREATE OR REPLACE PROCEDURE appendLOB_proc IS
    Dest_loc      BLOB;
    Src_loc       BLOB;
BEGIN
    /* LOB を選択し、宛先 LOB ロケータを取得します。*/
    SELECT Sound INTO Dest_loc FROM Multimedia_tab
        WHERE Clip_ID = 2
        FOR UPDATE;
    /* LOB を選択し、ソース LOB ロケータを取得します。*/
    SELECT Sound INTO Src_loc FROM Multimedia_tab
        WHERE Clip_ID = 1;
    /* LOB のオープンはオプションです。*/
    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);
    /* LOB をオープンしている場合、その LOB をクローズする必要があります。*/
    DBMS_LOB.CLOSE (Dest_loc);
    DBMS_LOB.CLOSE (Src_loc);
```

```

COMMIT;

EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Operation failed');
END;

```

C (OCI) : 他の LOB への LOB の追加

```

/* このファンクションによって、ソース LOB が宛先 LOB の最後に追加されます。 */
/* ロケータを選択します。 */
sb4 select_lock_sound_locator_2(Lob_loc, dest_type, errhp, svchp, stmhpt)
OCILobLocator *Lob_loc;
ub1          dest_type;          /* 宛先ロケータかどうか */
OCIError     *errhp;
OCISvcCtx    *svchp;
OCISstmt     *stmhpt;
{
    char        sqlstmt[150];
    OCIDefine   *defnp1;
    if (dest_type == TRUE)
    {
        strcpy(sqlstmt,
            (char *)"SELECT Sound FROM Multimedia_tab WHERE Clip_ID=2 FOR UPDATE");
        printf (" select destination sound locator\n");
    }
    else
    {
        strcpy(sqlstmt, (char *)"SELECT Sound FROM Multimedia_tab WHERE Clip_ID=1");
        printf (" select source sound locator\n");
    }
    checkerr (errhp, OCISstmtPrepare(stmhpt, errhp, (text *)sqlstmt,
        (ub4)strlen((char *)sqlstmt),
        (ub4) OCI_NTV_SYNTAX, (ub4) OCI_DEFAULT));

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

    /* SELECT を実行し、1行フェッチします。 */
    checkerr(errhp, OCISstmtExecute(svchp, stmhpt, errhp, (ub4) 1, (ub4) 0,
        (CONST OCISnapshot*) 0, (OCISnapshot*) 0,
        (ub4) OCI_DEFAULT));

```

```
    return 0;
}
void appendLob(envhp, errhp, svchp, stmthp)
OCIEnv    *envhp;
OCIError  *errhp;
OCISvcCtx *svchp;
OCIStmt   *stmthp;
{
    OCILobLocator *Dest_loc, *Src_loc;

    /* LOB ロケータを割り当てます。 */
    (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &Dest_loc,
                              (ub4)OCI_DTYPE_LOB, (size_t) 0, (dvoid **) 0);
    (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &Src_loc,
                              (ub4)OCI_DTYPE_LOB, (size_t) 0, (dvoid **) 0);

    /* LOB を選択します。 */
    printf(" select source and destination Lobs\n");
    select_lock_sound_locator_2(Dest_loc, TRUE, errhp, svchp, stmthp);
    /* 宛先ロケータ */
    select_lock_sound_locator_2(Src_loc, FALSE, errhp, svchp, stmthp);
    /* ソース・ロケータ */

    /* LOB のオープンはオプションです。 */
    checkerr (errhp, OCILobOpen(svchp, errhp, Dest_loc, OCI_LOB_READWRITE));
    checkerr (errhp, OCILobOpen(svchp, errhp, Src_loc, OCI_LOB_READONLY));

    /* ソース LOB を宛先 LOB の最後に追加します。 */
    printf(" append the source Lob to the destination Lob\n");
    checkerr(errhp, OCILobAppend(svchp, errhp, Dest_loc, Src_loc));

    /* LOB をオープンしている場合、その LOB をクローズする必要があります。 */
    checkerr (errhp, OCILobClose(svchp, errhp, Dest_loc));
    checkerr (errhp, OCILobClose(svchp, errhp, Src_loc));

    /* ロケータが保持しているリソースを解放します。 */
    (void) OCIDescriptorFree((dvoid *) Dest_loc, (ub4) OCI_DTYPE_LOB);
    (void) OCIDescriptorFree((dvoid *) Src_loc, (ub4) OCI_DTYPE_LOB);

    return;
}
```

COBOL (Pro*COBOL) : 他のLOBへのLOBの追加

```
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.

* BLOB ロケータの割当ておよび初期化を行います。
      EXEC SQL ALLOCATE :DEST END-EXEC.
      EXEC SQL ALLOCATE :SRC END-EXEC.
      EXEC SQL WHENEVER NOT FOUND GOTO END-OF-BLOB END-EXEC.
      EXEC SQL SELECT SOUND INTO :DEST
          FROM MULTIMEDIA_TAB WHERE CLIP_ID = 2 FOR UPDATE END-EXEC.
      EXEC SQL SELECT SOUND INTO :SRC
          FROM MULTIMEDIA_TAB WHERE CLIP_ID = 1 END-EXEC.

* 宛先LOBを書込み / 読み込みでオープンし、ソースLOBを読み込み専用でオープンします。
      EXEC SQL LOB OPEN :DEST READ WRITE END-EXEC.
      EXEC SQL LOB OPEN :SRC READ ONLY END-EXEC.

* ソースLOBを宛先LOBに追加します。
      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++) : 他の LOB への LOB の追加

```
#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();

    /* ロケータを割り当てます。*/
    EXEC SQL ALLOCATE :Dest_loc;
    EXEC SQL ALLOCATE :Src_loc;

    /* 宛先ロケータを選択します。*/
    EXEC SQL SELECT Sound INTO :Dest_loc
        FROM Multimedia_tab WHERE Clip_ID = 2 FOR UPDATE;

    /* ソース・ロケータを選択します。*/
    EXEC SQL SELECT Sound INTO :Src_loc
        FROM Multimedia_tab WHERE Clip_ID = 1;

    /* LOB のオープンはオプションです。*/
    EXEC SQL LOB OPEN :Dest_loc READ WRITE;
    EXEC SQL LOB OPEN :Src_loc READ ONLY;

    /* ソース LOB を宛先 LOB の最後に追加します。*/
    EXEC SQL LOB APPEND :Src_loc TO :Dest_loc;
```

```

/* LOB をオープンしている場合、その LOB をクローズする必要があります。 */
EXEC SQL LOB CLOSE :Dest_loc;
EXEC SQL LOB CLOSE :Src_loc;

/* ロケータが保持しているリソースを解放します。 */
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 (OO40) : 他の LOB への LOB の追加

```

Dim MySession As OraSession
Dim OraDb As OraDatabase
Dim OraDyn As OraDynaset, OraSound1 As OraBlob, OraSoundClone As OraBlob

Set MySession = CreateObject("OracleInProcServer.XOraSession")
Set OraDb = MySession.OpenDatabase("exampledb", "samp/samp", 0&)
Set OraDyn = OraDb.CreateDynaset(
    "SELECT * FROM Multimedia_tab ORDER BY clip_id", ORADYN_DEFAULT)
Set OraSound1 = OraDyn.Fields("Sound").Value
Set OraSoundClone = OraSound1

OraDyn.MoveNext
OraDyn.Edit
OraSound1.Append OraSoundClone
OraDyn.Update

```

Java (JDBC) : 他の LOB への LOB の追加

```

import java.io.InputStream;
import java.io.OutputStream;

// コア JDBC クラス :
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 固有の JDBC クラス :
import oracle.sql.*;
import oracle.jdbc.driver.*;

public class Ex2_121
{
    static final int MAXBUFSIZE = 32767;
    public static void main (String args [])
        throws Exception
    {
        // Oracle JDBC Driver をロードします。
        DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

        // データベースに接続します。
        Connection conn =
            DriverManager.getConnection ("jdbc:oracle:oci8:@", "samp", "samp");

        // 自動コミットがオフの場合、高速になります。
        conn.setAutoCommit (false);

        // 文を作成します。
        Statement stmt = conn.createStatement ();
        try
        {
            ResultSet rset = null;
            BLOB dest_loc = null;
            BLOB src_loc = null;
            InputStream in = null;
            byte[] buf = new byte[MAXBUFSIZE];
            int length = 0;
            long pos = 0;
            rset = stmt.executeQuery (
                "SELECT sound FROM multimedia_tab WHERE clip_id = 2");
            if (rset.next())
            {
                src_loc = ((OracleResultSet)rset).getBLOB (1);
            }
            in = src_loc.getBinaryStream();
```

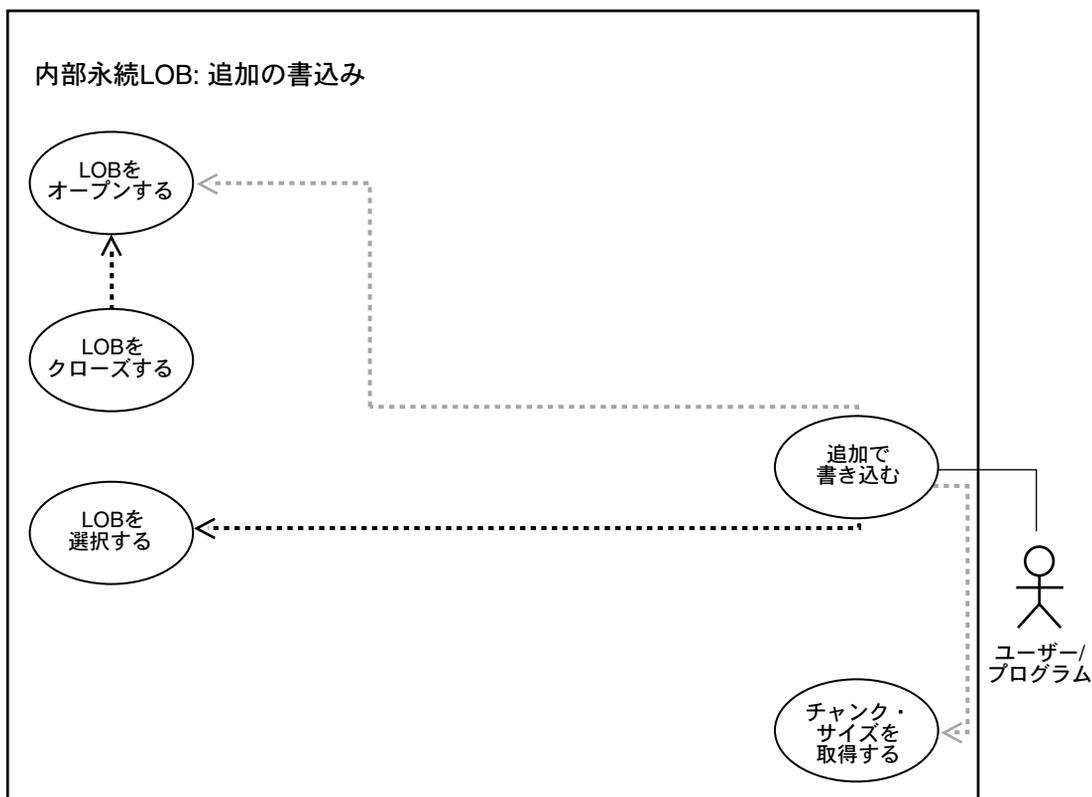
```
        rset = stmt.executeQuery (
            "SELECT sound FROM multimedia_tab WHERE clip_id = 1 FOR UPDATE");
        if (rset.next())
        {
            dest_loc = ((OracleResultSet)rset).getBLOB (1);
        }
        // LOB の最後に書き込み (つまり追加) を開始します。
        pos = dest_loc.length();
        while ((length = in.read(buf)) != -1)
        {
            // アウトプット LOB の位置 pos にバッファの内容を書き込みます。
            dest_loc.putBytes(pos, buf);
            pos += length;
        }

        // すべてのストリームおよびハンドルをクローズします。
        in.close();
        stmt.close();
        conn.commit();
        conn.close();

    }
    catch (SQLException e)
    {
        {
            e.printStackTrace();
        }
    }
}
```

LOB への追加の書込み

図 9-34 ユースケース図：LOB への追加の書込み



参照： 内部永続 LOB に関するすべての基本操作については、9-2 ページの「ユースケース・モデル：内部永続 LOB – 基本操作」を参照してください。

用途

LOB に追加で書き込みます。

使用上の注意

1 つずつまたはピース単位の書込み

writeappend 操作では、バッファが LOB の最後に書き込まれます。

OCI では、このコールを使用してバッファからピース単位で LOB に書き込むことができます。また、コールバックまたは標準のポーリング方法を使用してもピース単位に処理できます。

ピース単位の書込み：コールバックまたはポーリングの使用時期 ピース・パラメータの値が OCI_FIRST_PIECE の場合、データはコールバックまたはポーリングを介して提供される必要があります。

- コールバック関数が cbfp パラメータ内に定義されている場合、ピースがパイプに書き込まれてから、このコールバック関数を起動して次のピースを取得します。各ピースは bufp から書き込まれます。
- コールバック関数が定義されていないと、OCILobWriteAppend() は OCI_NEED_DATA エラー・コードを戻します。アプリケーションは OCILobWriteAppend() を再びコールし、さらに LOB のピースを書き込む必要があります。このモードでは、ピースのサイズおよび位置が異なると、コールごとにバッファ・ポインタや長さが異なる場合があります。ピース値 OCI_LAST_PIECE は、ピース単位の書込みを終了します。

更新前の行のロック

PL/SQL DBMS_LOB パッケージまたは OCI を使用して LOB の値を更新する前に、LOB を含む行をロックする必要があります。SQL INSERT 文および UPDATE 文は暗黙的に行をロックしますが、SQL プログラムおよび PL/SQL プログラムでは SQL SELECT FOR UPDATE 文を使用して、また OCI プログラムでは OCI pin または lock 関数を使用して明示的にロックします。

更新後のロケータの状態の詳細は、5-6 ページの「[更新済ロケータを介した更新済 LOB](#)」を参照してください。

構文

各プログラム環境で使用可能な機能のリストは、[第 3 章「LOB プログラム環境」](#)を参照してください。各プログラム環境における構文については、次のマニュアルの項を参照してください。

- PL/SQL (DBMS_LOB パッケージ) : 『Oracle8i PL/SQL パッケージ・プロシージャ・リファレンス』の第 18 章「DBMS_LOB」の「サブプログラムの要約」の WRITEAPPEND プロシージャ

- C (OCI) : 『Oracle8i コール・インタフェース・プログラマーズ・ガイド』の第 15 章「OCI リレーショナル関数」の「LOB 関数」の OCILobWriteAppend()
- COBOL (Pro*COBOL) : 『Oracle8i Pro*COBOL プリコンパイラ・プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB WRITE (実行可能埋込み SQL 拡張要素)」、「LOB APPEND (実行可能埋込み SQL 拡張要素)」
- C/C++ (Pro*C/C++) : 『Oracle8i Pro*C/C++ プリコンパイラ・プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB WRITE (実行可能埋込み SQL 拡張要素)」、「LOB APPEND (実行可能埋込み SQL 拡張要素)」
- Visual Basic (OO4O) : 参照マニュアルはありません。
- Java (JDBC) : 『Oracle8i JDBC 開発者ガイドおよびリファレンス』の第 7 章「LOB と BFILE の操作」の「BLOB と CLOB の操作」の「BLOB または CLOB 列の作成と移入」、および『Oracle8i SQLJ 開発者ガイドおよびリファレンス』の第 5 章「型のサポート」の「Oracle の拡張型」の「BLOB、CLOB および BFILE のサポート」

使用例

次の例では、ビデオ・フレーム (Frame) の終わりに書込みを行います。

例

次のプログラム環境の例が示されています。

- [PL/SQL \(DBMS_LOB パッケージ\) : LOB への追加の書込み \(9-190 ページ\)](#)
- [C \(OCI\) : LOB への追加の書込み \(9-191 ページ\)](#)
- [COBOL \(Pro*COBOL\) : LOB への追加の書込み \(9-192 ページ\)](#)
- [C/C++ \(Pro*C/C++\) : LOB への追加の書込み \(9-194 ページ\)](#)
- Visual Basic (OO4O) : 今回のリリースでは例は記載されていません。
- [Java \(JDBC\) : LOB への追加の書込み \(9-195 ページ\)](#)

PL/SQL (DBMS_LOB パッケージ) : LOB への追加の書込み

/* 例のプロシージャ lobWriteAppend_proc は、DBMS_LOB パッケージの一部ではないことに注意してください。*/

```
CREATE OR REPLACE PROCEDURE lobWriteAppend_proc IS
  Lob_loc      BLOB;
  Buffer        RAW(32767);
  Amount       Binary_integer := 32767;
BEGIN
  SELECT Frame INTO Lob_loc FROM Multimedia_tab where Clip_ID = 1 FOR UPDATE;
```

```

/* バッファにデータを充填します。*/
/* LOB のオープンはオプションです。*/
DBMS_LOB.OPEN (Lob_loc, DBMS_LOB.LOB_READWRITE);
/* バッファから LOB の最後へデータを追加します。*/
DBMS_LOB.WRITEAPPEND(Lob_loc, Amount, Buffer);
/* LOB をオープンしている場合、その LOB をクローズする必要があります。*/
DBMS_LOB.CLOSE(Lob_loc);
END;

```

C (OCI) : LOB への追加の書込み

```

/* ロケータ変数にロケータを選択します。*/
sb4 select_lock_frame_locator(Lob_loc, errhp, svchp, stmthp)
OCILobLocator *Lob_loc;
OCIError      *errhp;
OCISvcCtx     *svchp;
OCIStmt       *stmthp;
{
    text *sqlstmt =
        (text *)"SELECT Frame FROM Multimedia_tab WHERE Clip_ID=1 FOR UPDATE";
    OCIDefine *defnp1;
    checkerr (errhp, OCIStmtPrepare(stmthp, errhp, sqlstmt,
                                    (ub4)strlen((char *)sqlstmt),
                                    (ub4) OCI_NT_V_SYNTAX, (ub4) OCI_DEFAULT));
    checkerr (errhp, OCIDefineByPos(stmthp, &defnp1, errhp, (ub4) 1,
                                    (dvoid *)&Lob_loc, (sb4) 0,
                                    (ub2) SQLT_BLOB, (dvoid *) 0,
                                    (ub2 *) 0, (ub2 *) 0, (ub4) OCI_DEFAULT));

    /* SELECT を実行し、1 行フェッチします。*/
    checkerr(errhp, OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                                   (CONST OCI_Snapshot*) 0, (OCI_Snapshot*) 0,
                                   (ub4) OCI_DEFAULT));

    return (0);
}

#define MAXBUFLLEN 32767

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

```

```
{
    OCIBlobLocator *Lob_loc;
    ub4 amt;
    ub4 offset;
    sword retval;
    ub1 bufp[MAXBUFLEN];
    ub4 buflen;

    /* ロケータを割り当てます。*/
    (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &Lob_loc,
                              (ub4)OCI_DTYPE_LOB, (size_t) 0, (dvoid **) 0);

    /* BLOB を選択します。*/
    printf(" select and lock a frame locator\n");
    select_lock_frame_locator(Lob_loc, errhp, svchp, stnthp);

    /* BLOB をオープンします。*/
    checkerr (errhp, (OCILobOpen(svchp, errhp, Lob_loc, OCI_LOB_READWRITE)));

    /* amt をバッファの長さに設定します。ここで amt は、BLOB を使用しているためバイト単位であることに注意してください。*/
    amt = sizeof (bufp);
    buflen = sizeof (bufp);

    /* bufp にデータを充填します。*/
    /* LOB の最後にバッファのデータを書き込みます。*/
    printf(" write-append data to the frame Lob\n");
    checkerr (errhp, OCILobWriteAppend (svchp, errhp, Lob_loc, &amt,
                                         bufp, buflen,
                                         OCI_ONE_PIECE, (dvoid *)0,
                                         (sb4 *) (dvoid *, dvoid *, ub4 *, ub1 *))0,
                                         0, SQLCS_IMPLICIT));

    /* BLOB をオープンしている場合、その BLOB をクローズする必要があります。*/
    checkerr (errhp, OCILobClose(svchp, errhp, Lob_loc));
    /* ロケータが保持しているリソースを解放します。*/
    (void) OCIDescriptorFree((dvoid *) Lob_loc, (ub4) OCI_DTYPE_LOB);

    return;
}
```

COBOL (Pro*COBOL) : LOB への追加の書込み

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.

* BLOB ロケータの割当ておよび初期化を行います。
EXEC SQL ALLOCATE :BLOB1 END-EXEC.
EXEC SQL WHENEVER NOT FOUND GOTO END-OF-BLOB END-EXEC.
EXEC SQL SELECT FRAME INTO :BLOB1
FROM MULTIMEDIA_TAB WHERE CLIP_ID = 1 FOR UPDATE END-EXEC.

* ターゲット LOB をオープンします。
EXEC SQL LOB OPEN :BLOB1 READ WRITE END-EXEC.

* ここで AMT を移入します。
MOVE 5 TO AMT.
MOVE "2424242424" TO BUFFER.

* ソース LOB を宛先 LOB に追加します。
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++) : LOB への追加の書込み

```
#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 であるため、単一の書込みのみが必要です。*/
    char Buffer[BufferLength];
    /* このデータ型では、データ型を等価にする必要があります。*/
    EXEC SQL VAR Buffer IS RAW(BufferLength);
    EXEC SQL ALLOCATE :Lob_loc;
    EXEC SQL SELECT Frame INTO :Lob_loc
        FROM Multimedia_tab WHERE Clip_ID = 1 FOR UPDATE;
    /* LOB のオープンにはオプションです。*/
    EXEC SQL LOB OPEN :Lob_loc;
    memset((void *)Buffer, 1, BufferLength);
    /* LOB の最後にバッファのデータを書き込みます。*/
    EXEC SQL LOB WRITE APPEND :Amount FROM :Buffer INTO :Lob_loc;
    /* LOB をオープンしている場合、その LOB をクローズする必要があります。*/
    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) : LOB への追加の書込み

```
import java.io.OutputStream;

// コア JDBC クラス :
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 固有の JDBC クラス :
import oracle.sql.*;
import oracle.jdbc.driver.*;

public class Ex2_126
{
    static final int MAXBUFSIZE = 32767;
    public static void main (String args [])
        throws Exception
    {
        // Oracle JDBC Driver をロードします。
        DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

        // データベースに接続します。
        Connection conn =
            DriverManager.getConnection ("jdbc:oracle:oci8:@", "samp", "samp");

        // 自動コミットがオフの場合、高速になります。
        conn.setAutoCommit (false);

        // 文を作成します。
        Statement stmt = conn.createStatement ();
        try
        {
            BLOB dest_loc = null;
```

```
byte[] buf = new byte[MAXBUFSIZE];
long pos = 0;
ResultSet rset = stmt.executeQuery (
    "SELECT frame FROM multimedia_tab WHERE clip_id = 1 FOR UPDATE");
if (rset.next())
{
    dest_loc = ((OracleResultSet)rset).getBLOB (1);
}

// LOB の最後に書込み (つまり追加) を開始します。
pos = dest_loc.length();

// バッファに書き込む内容を充填します。
buf = (new String("Hello World")).getBytes();

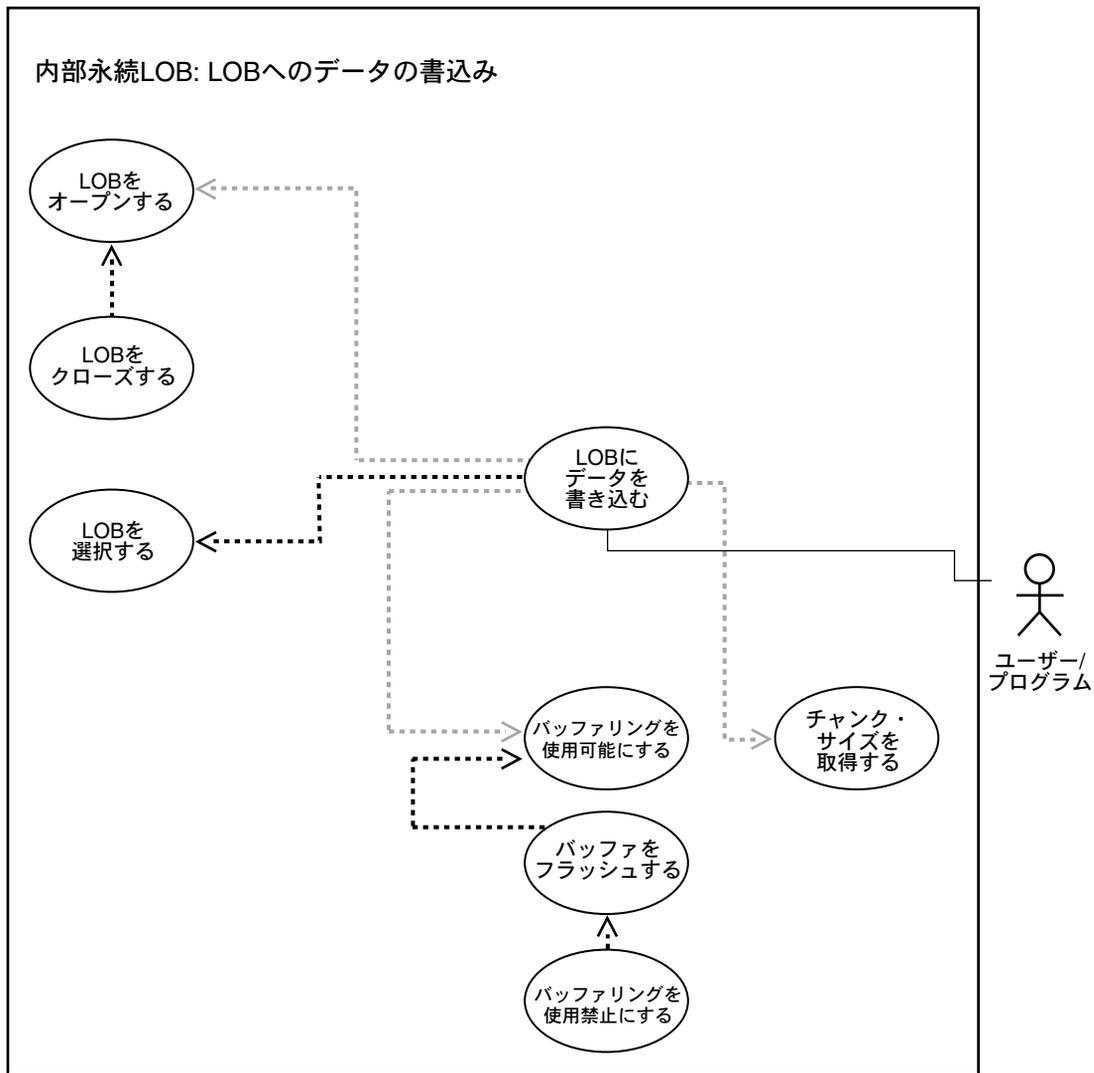
// アウトプット LOB の位置 pos にバッファの内容を書き込みます。
dest_loc.putBytes(pos, buf);

// すべてのストリームおよびハンドルをクローズします。
stmt.close();
conn.commit();
conn.close();

}
catch (SQLException e)
{
    e.printStackTrace();
}
}
```

LOB へのデータの書込み

図 9-35 ユースケース図：LOB へのデータの書込み



参照： 内部永続 LOB に関するすべての基本操作については、9-2 ページの「ユースケース・モデル:内部永続 LOB – 基本操作」を参照してください。

用途

データを LOB に書き込みます。

使用上の注意

ストリーム書き込み

大量の LOB データを最も効率よく書き込む方法は、ポーリングまたはコールバックを介してストリーム・メカニズムを使用可能にし、`OCILOBWrite()` を使用することです。LOB に書き込むデータの量がわかっている場合は、`OCILOBWrite()` のコール時にそのデータ量を指定します。これによって、ディスクに LOB データが連続的に書き込まれます。領域を効率的に利用できるのみでなく、LOB データの連続性により、その後の操作で読書きの速度が速くなります。

チャンク・サイズ

チャンクとは1つまたは複数の Oracle ブロックです。前述のとおり、LOB を含む表を作成するときに、LOB 用のチャンク・サイズを指定できます。これは、LOB 値に対してアクセス / 変更を行っているときに、Oracle が使用するチャンク・サイズに対応します。チャンクの一部はシステム関連の情報を格納し、残りは LOB 値を格納します。`getchunksize` 関数は、LOB チャンク内で使用され、LOB 値を格納している領域の量を戻します。

書き込みパフォーマンスを改善するために、チャンク・サイズの複数倍を指定する このチャンク・サイズの複数倍を指定して書き込み要求を実行すると、パフォーマンスが向上します。これは、LOB チャンクが書き込み要求ごとにバージョン化されるためです。書き込みすべてを1つのチャンクで行うと、余分のバージョン化は行われません。ご使用のアプリケーションに問題がなければ、同じサイズの LOB チャンクで複数回の LOB 書き込みコールを発行するのではなく、十分なサイズのチャンクにバッチ書き込みを行うと、パフォーマンスが向上します。

更新前の行のロック

PL/SQL DBMS_LOB パッケージまたは OCI を使用して LOB の値を更新する前に、LOB を含む行をロックする必要があります。SQL INSERT 文および UPDATE 文は暗黙的に行をロックしますが、SQL プログラムおよび PL/SQL プログラムでは SQL SELECT FOR UPDATE 文を使用して、また OCI プログラムでは `OCI pin` または `lock` 関数を使用して明示的にロックします。

更新後のロケータの状態の詳細は、5-6 ページの「[更新済ロケータを介した更新済 LOB](#)」を参照してください。

DBMS_LOB.WRITE() を使用した BLOB へのデータの書込み

16 進文字列を DBMS_LOB.WRITE() に渡して BLOB にデータを書き込む場合は、次のガイドラインに従ってください。

- 量パラメータは、バッファの長さパラメータ以下にする必要があります。
- バッファの長さは $((\text{量パラメータの値} \times 2) - 1)$ にする必要があります。このガイドラインが存在する理由は、パラメータ文字列の 2 つの文字が 1 つの 16 進文字とみなされる（また、16 進から RAW への暗黙的な変換が行われる）ためです。たとえば、文字列が 2 バイトごとに 1 つの RAW バイトへと変換されます。

次は正しい例です。

```
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;
```

この例の「an_amount」の値を次の値に置き換えると、エラー・メッセージ ORA-21560 が戻されます。

```
an_amount BINARY_INTEGER := 11;
```

または

```
an_amount BINARY_INTEGER := 19;
```

構文

各プログラム環境で使用可能な機能のリストは、[第 3 章「LOB プログラム環境」](#)を参照してください。各プログラム環境における構文については、次のマニュアルの項を参照してください。

- PL/SQL (DBM_LOB パッケージ) : 『Oracle8i PL/SQL パッケージ・プロシージャ・リファレンス』の第 18 章「DBMS_LOB」の「サブプログラムの要約」の WRITE プロシージャ
- C (OCI) : 『Oracle8i コール・インタフェース・プログラマーズ・ガイド』の第 15 章「OCI リレーショナル関数」の「LOB 関数」の OCILobWrite()
- COBOL (Pro*COBOL) : 『Oracle8i Pro*COBOL プリコンパイラ・プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB WRITE (実行可能埋込み SQL 拡張要素)」
- C/C++ (Pro*C/C++) : 『Oracle8i Pro*C/C++ プリコンパイラ・プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB WRITE (実行可能埋込み SQL 拡張要素)」
- Visual Basic (Oracle Objects for OLE オンライン・ヘルプ) : 「Help」の「Contents」タブから、「OO4O Automation Server」>「OBJECTS」>「Orablob」>「METHODS」>「write」、「copyfromfile」を選択してください。
- Java (JDBC) : 『Oracle8i JDBC 開発者ガイドおよびリファレンス』の第 7 章「LOB と BFILE の操作」の「BLOB と CLOB の操作」の「BLOB または CLOB 列の作成と移入」、および『Oracle8i SQLJ 開発者ガイドおよびリファレンス』の第 5 章「型のサポート」の「Oracle の拡張型」の「BLOB、CLOB および BFILE のサポート」

使用例

次の例では、LOB にデータを書き込むことによって STORY データ (クリップ用のストーリー・ボード) が更新できます。

例

次のプログラム環境の例が示されています。

- [PL/SQL \(DBMS_LOB パッケージ\) : LOB へのデータの書込み \(9-201 ページ\)](#)
- [C \(OCI\) : LOB へのデータの書込み \(9-202 ページ\)](#)
- [COBOL \(Pro*COBOL\) : LOB へのデータの書込み \(9-205 ページ\)](#)
- [C/C++ \(Pro*C/C++\) : LOB へのデータの書込み \(9-208 ページ\)](#)
- [Visual Basic \(OO4O\) : LOB へのデータの書込み \(9-210 ページ\)](#)
- [Java \(JDBC\) : LOB へのデータの書込み \(9-211 ページ\)](#)

PL/SQL (DBMS_LOB パッケージ) : LOB へのデータの書込み

```

/* 例のプロシージャ writeDataToLOB_proc は、DBMS_LOB パッケージの
一部ではないことに注意してください。*/
CREATE or REPLACE PROCEDURE writeDataToLOB_proc IS
  Lob_loc          CLOB;
  Buffer            VARCHAR2 (32767);
  Amount           BINARY_INTEGER := 32767;
  Position         INTEGER := 1;
  i                INTEGER;
BEGIN
  /* LOB を選択します。*/
  SELECT Story INTO Lob_loc
    FROM Multimedia_tab
   WHERE Clip_ID = 1
   FOR UPDATE;
  /* LOB のオープンはオプションです。*/
  DBMS_LOB.OPEN (Lob_loc, DBMS_LOB.LOB_READWRITE);
  /* LOB に書き込むデータをバッファに充填します。*/
  FOR i IN 1..3 LOOP
    DBMS_LOB.WRITE (Lob_loc, Amount, Position, Buffer);
    /* LOB に書き込む追加のデータをバッファに充填します。*/
    Position := Position + Amount;
  END LOOP;
  /* LOB をオープンしている場合、その LOB をクローズする必要があります。*/
  DBMS_LOB.CLOSE (Lob_loc);
END;

/* バッファのサイズおよび量が最初の例と異なる、2 番目の例を追加しています。*/
CREATE or REPLACE PROCEDURE writeDataToLOB_proc IS
  Lob_loc          CLOB;
  Buffer            VARCHAR2 (32767);
  Amount           BINARY_INTEGER := 32767;
  Position         INTEGER;
  i                INTEGER;
  Chunk_size       INTEGER;
BEGIN
  SELECT Story INTO Lob_loc
    FROM Multimedia_tab
   WHERE Clip_ID = 1
   FOR UPDATE;
  /* LOB のオープンはオプションです。*/
  DBMS_LOB.OPEN (Lob_loc, DBMS_LOB.LOB_READWRITE);
  Chunk_size := DBMS_LOB.GETCHUNKSIZE (Lob_loc);

```

```

/* LOB に書き込む 'Chunk_size' 相当データをバッファに充填します。LOB にデータ
   を書き込む場合、チャンク・サイズ（または複数のチャンク・サイズ）を使用します。
   チャンクの境界内で書込みを行い、DBMS_LOB.WRITE への単一のコール内で異なる
   チャンクをオーバーラップしていないことを確認してください。 */

Amount := Chunk_size;

/* 2 番目のチャンクの始まりから、データの書込みを開始します。 */
Position := Chunk_size + 1;
FOR i IN 1..3 LOOP
    DBMS_LOB.WRITE (Lob_loc, Amount, Position, Buffer);
    /* LOB に書き込む追加のデータ（サイズは Chunk_size）をバッファに充填
       します。 */
    Position := Position + Amount;
END LOOP;
/* LOB をオープンしている場合、その LOB をクローズする必要があります。 */
DBMS_LOB.CLOSE (Lob_loc);
END;

```

C (OCI) : LOB へのデータの書込み

/* この例では、単一のピースまたは複数のピースごと任意の量のデータを内部 LOB に書き込む、OCI の機能を示します。これには、標準のポーリングとともにストリーミングが使用されます。動的に割り当てられたバッファを使用して、LOB に書き込まれているデータが保持されます。 */

```

/* ロケータ変数にロケータを選択します。 */
sb4 select_lock_story_locator(Lob_loc, errhp, svchp, stmthp)
OCILobLocator *Lob_loc;
OCIError      *errhp;
OCISvcCtx     *svchp;
OCIStmt       *stmthp;
{
    text *sqlstmt =
        (text *) "SELECT Story FROM Multimedia_tab m \
                WHERE m.Clip_ID = 1 FOR UPDATE";
    OCIDefine *defnp1;
    checkerr (errhp, OCIStmtPrepare(stmthp, errhp, sqlstmt,
                                     (ub4) strlen((char *) sqlstmt),
                                     (ub4) OCI_NTV_SYNTAX, (ub4) OCI_DEFAULT));
    checkerr (errhp, OCIDefineByPos(stmthp, &defnp1, errhp, (ub4) 1,
                                     (dvoid *)&Lob_loc, (dvoid *) 0,
                                     (ub2) SQLT_CLOB, (dvoid *) 0,
                                     (ub2 *) 0, (ub2 *) 0, (ub4) OCI_DEFAULT));
    /* SELECT を実行し、1 行フェッチします。 */
    checkerr(errhp, OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,

```

```

(CONST OCISnapshot*) 0, (OCISnapshot*) 0,
(ub4) OCI_DEFAULT));

return (0);
}

#define MAXBUFLLEN 32767
void writeDataToLob(envhp, errhp, svchp, stmthp)
OCIEnv *envhp;
OCIError *errhp;
OCISvcCtx *svchp;
OCIStmt *stmthp;
{
    OCIClobLocator *Lob_loc;
    ub4 Total = 2.5*MAXBUFLLEN;
        /* <CLOB に書き込むデータの合計量 (バイト単位) > */
    unsigned int amt;
    unsigned int offset;
    unsigned int remainder, nbytes;
    boolean last;
    ub1 bufp[MAXBUFLLEN];
    sb4 err;

    /* ロケータ記述子を割り当てます。*/
    (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &Lob_loc,
        (ub4)OCI_DTYPE_LOB, (size_t) 0, (dvoid **) 0);

    /* LOB を選択します。*/
    printf (" select a story Lob\n");
    select_lock_story_locator(Lob_loc, errhp, svchp, stmthp);

    /* CLOB をオープンします。*/
    checkerr (errhp, (OCILobOpen(svchp, errhp, Lob_loc, OCI_LOB_READWRITE)));
    if (Total > MAXBUFLLEN)
        nbytes = MAXBUFLLEN; /* 標準のポーリングを介してストリーミングを使用します。*/
    else
        nbytes = Total; /* 単一の書込みのみ必要です。*/

    /* バッファに n バイト相当のデータを充填します。*/
    remainder = Total - nbytes;

    /* Amount を 0 (ゼロ) に設定すると、OCI_LAST_PIECE を指定するまでデータが流されます。*/
    amt = 0;
    offset = 1;

    printf(" write the Lob data in pieces\n");
    if (0 == remainder)
    {

```

```
amt = nbytes;
/* ここで、合計が MAXBUFLen 以下であるため、1つのピースごと書込みできます。*/
checkerr (errhp, OCILobWrite (svchp, errhp, Lob_loc, &amt,
                             offset, bufp, nbytes,
                             OCI_ONE_PIECE, (dvoid *)0,
                             (sb4 (*) (dvoid*,dvoid*,ub4*,ub1 *))0,
                             0, SQLCS_IMPLICIT));
}
else
{
/* ここで、合計が MAXBUFLen より大きいため、標準のポーリングを介してストリーミングを使用します。*/
/* 最初のピースを書き込みます。FIRST を指定すると、ポーリングが開始します。*/
err = OCILobWrite (svchp, errhp, Lob_loc, &amt,
                  offset, bufp, nbytes,
                  OCI_FIRST_PIECE, (dvoid *)0,
                  (sb4 (*) (dvoid*,dvoid*,ub4*,ub1 *))0,
                  0, SQLCS_IMPLICIT);
if (err != OCI_NEED_DATA)
    checkerr (errhp, err);
last = FALSE;
/* 次 (中間) および最後のピースを書き込みます。*/
do
{
    if (remainder > MAXBUFLen)
        nbytes = MAXBUFLen;          /* ピースが残っています。*/
    else
    {
        nbytes = remainder;          /* ここで、remainder は MAXBUFLen 以下です。*/
        last = TRUE;                 /* これが最後のピースになります。*/
    }

/* バッファに n バイト相当のデータを充填します。*/
if (last)
{
    /* LAST を指定すると、ポーリングが終了します。*/
    err = OCILobWrite (svchp, errhp, Lob_loc, &amt,
                      offset, bufp, nbytes,
                      OCI_LAST_PIECE, (dvoid *)0,
                      (sb4 (*) (dvoid*,dvoid*,ub4*,ub1 *))0,
                      0, SQLCS_IMPLICIT);
    if (err != OCI_SUCCESS)
        checkerr (errhp, err);
}
else
{
    err = OCILobWrite (svchp, errhp, Lob_loc, &amt,
```

```

        offset, bufp, nbytes,
        OCI_NEXT_PIECE, (dvoid *)0,
        (sb4 (*) (dvoid*, dvoid*, ub4*, ub1 *))0,
        0, SQLCS_IMPLICIT);
    if (err != OCI_NEED_DATA)
        checkerr (errhp, err);
}
/* 残りの書込み量を判断します。*/
remainder = remainder - nbytes;
} while (!last);
}

/* この時点では、(remainder == 0 (ゼロ)) です。*/

/* LOB をオープンしている場合、その LOB をクローズする必要があります。*/
checkerr (errhp, OCILobClose(svchp, errhp, Lob_loc));

/* ロケータが保持しているリソースを解放します。*/
(void) OCIDescriptorFree((dvoid *) Lob_loc, (ub4) OCI_DTYPE_LOB);

return;
}

```

COBOL (Pro*COBOL) : LOB へのデータの書込み

```

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.
```

- * 入力ファイルをオープンします。

```
OPEN INPUT INFILE.
```

- * CLOB ロケータの割当ておよび初期化を行います。

```
EXEC SQL ALLOCATE :CLOB1 END-EXEC.
```

```
EXEC SQL
```

```
SELECT STORY INTO :CLOB1 FROM MULTIMEDIA_TAB
```

```
WHERE CLIP_ID = 1 FOR UPDATE
```

```
END-EXEC.
```

- * レコード全体または最初のピースを書き込みます。
- * データ・ファイルを読み込み、BUFFER-ARR および BUFFER-LEN を移入します。
- * ファイル全体が読み込まれている場合、END-OF-FILE が "Y" に設定されます。

```
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.
```

- * 入力データ・ファイルからの読み込みおよび CLOB への書込みを継続します。

```
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++) : LOB へのデータの書込み

```
/* この例では、Pro*C/C++ が、単一のピースまたは複数のピースごとに任意の量のデータを
内部 LOB に書き込む機能を提供する方法を示します。これには、標準のポーリングを使用
するストリーミング・メカニズムが使用されます。動的に割り当てられたバッファを使用して、
LOB に書き込まれているデータが保持されます。*/
#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();
    /* ロケータの割当ておよび初期化を行います。*/
    EXEC SQL ALLOCATE :Lob_loc;
    EXEC SQL SELECT Story INTO Lob_loc
        FROM Multimedia_tab WHERE Clip_ID = 1 FOR UPDATE;
    /* CLOB をオープンします。*/
    EXEC SQL LOB OPEN :Lob_loc READ WRITE;
    Total = Amount = (multiple * BufferLength);
    if (Total > BufferLength)
        nbytes = BufferLength; /* 標準のポーリングを介して、ストリーミングを使用します。*/
    else
        nbytes = Total; /* 単一の書込みのみ必要です。*/
    /* バッファに n バイト相当のデータを充填します。*/
    memset((void *)Buffer.arr, 32, nbytes);
    Buffer.len = nbytes; /* 長さを設定します。*/
    remainder = Total - nbytes;
```

```
if (0 == remainder)
{
    /* ここで、合計が BufferLength 以下であるため、1 つのピースごと書き込みできます。*/
    EXEC SQL LOB WRITE ONE :Amount FROM :Buffer INTO :Lob_loc;
    printf("Write ONE Total of %d characters\n", Amount);
}
else
{
    /* ここで、合計が BufferLength より大きいため、標準のポーリングを介してストリーミングを使用します。*/
    /* 最初のピースを書き込みます。FIRST を指定すると、ポーリングが開始します。*/
    EXEC SQL LOB WRITE FIRST :Amount FROM :Buffer INTO :Lob_loc;
    printf("Write first %d characters\n", Buffer.len);
    last = FALSE;
    /* 次 (中間) および最後のピースを書き込みます。*/
    do
    {
        if (remainder > BufferLength)
            nbytes = BufferLength;          /* ピースが残っています。*/
        else
        {
            nbytes = remainder;          /* ここで、remainder は BufferLength 以下です。*/
            last = TRUE;                  /* これが最後のピースになります。*/
        }
        /* バッファに n バイト相当のデータを充填します。*/
        memset((void *)Buffer.arr, 32, nbytes);
        Buffer.len = nbytes;              /* 長さを設定します。*/
        if (last)
        {
            EXEC SQL WHENEVER SQLERROR DO Sample_Error();
            /* LAST を指定すると、ポーリングが終了します。*/
            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);
        }
        /* 残りの書き込み量を判断します。*/
        remainder = remainder - nbytes;
    } while (!last);
}
EXEC SQL WHENEVER SQLERROR DO Sample_Error();
/* この時点では、(Amount == 合計) であるため、合計量が書き込まれています。*/
```

```
/* CLOB をクローズします。*/  
EXEC SQL LOB CLOSE :Lob_loc;  
/* ロケータが保持しているリソースを解放します。*/  
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 (0040) : LOB へのデータの書込み

'LOB の書込みには、orablob.write または orablob.copyfromfile を使用する 2 つの方法があります。

'OraBlob.Write メカニズムの使用

```
Dim OraDyn As OraDynaset, OraSound As OraBlob, amount_written%, chunksize%,  
    curchunk() As Byte  
  
chunksize = 32767  
Set OraDyn = OraDb.CreateDynaset("SELECT * FROM Multimedia_tab", ORADYN_DEFAULT)  
Set OraSound = OraDyn.Fields("Sound").Value  
  
fnum = FreeFile  
Open "c:\tmp\washington_audio" For Binary As #fnum  
  
OraSound.offset = 1  
OraSound.pollingAmount = LOF(fnum)  
remainder = LOF(fnum)  
  
Dim piece As Byte  
Get #fnum, , curchunk  
  
OraDyn.Edit  
  
piece = ORALOB_FIRST_PIECE  
OraSound.Write curchunk, chunksize, ORALOB_FIRST_PIECE  
  
While OraSound.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
    OraSound.Write curchunk, chunksize, piece

Wend

OraDyn.Update

'OraBlob.CopyFromFile メカニズムの使用

Set OraDyn = OraDb.CreateDynaset("select * from Multimedia_tab", ORADYN_DEFAULT)
Set OraSound = OraDyn.Fields("Sound").Value

OraDyn.Edit
OraSound.CopyFromFile "c:\mysound.aud"
OraDyn.Update
```

Java (JDBC) : LOB へのデータの書込み

```
import java.io.OutputStream;

// コア JDBC クラス :
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 固有の JDBC クラス :
import oracle.sql.*;
import oracle.jdbc.driver.*;

public class Ex2_126
{
    static final int MAXBUFSIZE = 32767;
    public static void main (String args [])
        throws Exception
    {
```

```
// Oracle JDBC Driver をロードします。
DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

// データベースに接続します。
Connection conn =
DriverManager.getConnection ("jdbc:oracle:oci8:@", "samp", "samp");

// 自動コミットがオフの場合、高速になります。
conn.setAutoCommit (false);

// 文を作成します。
Statement stmt = conn.createStatement ();
try
{
    BLOB dest_loc = null;
    byte[] buf = new byte[MAXBUFSIZE];
    long pos = 0;
    ResultSet rset = stmt.executeQuery (
        "SELECT frame FROM multimedia_tab WHERE clip_id = 1 FOR UPDATE");
    if (rset.next())
    {
        dest_loc = ((OracleResultSet)rset).getBLOB (1);
    }

// LOB の最後に書込み (つまり追加) を開始します。
pos = dest_loc.length();

// 書き込む内容をバッファに充填します。
buf = (new String("Hello World")).getBytes();

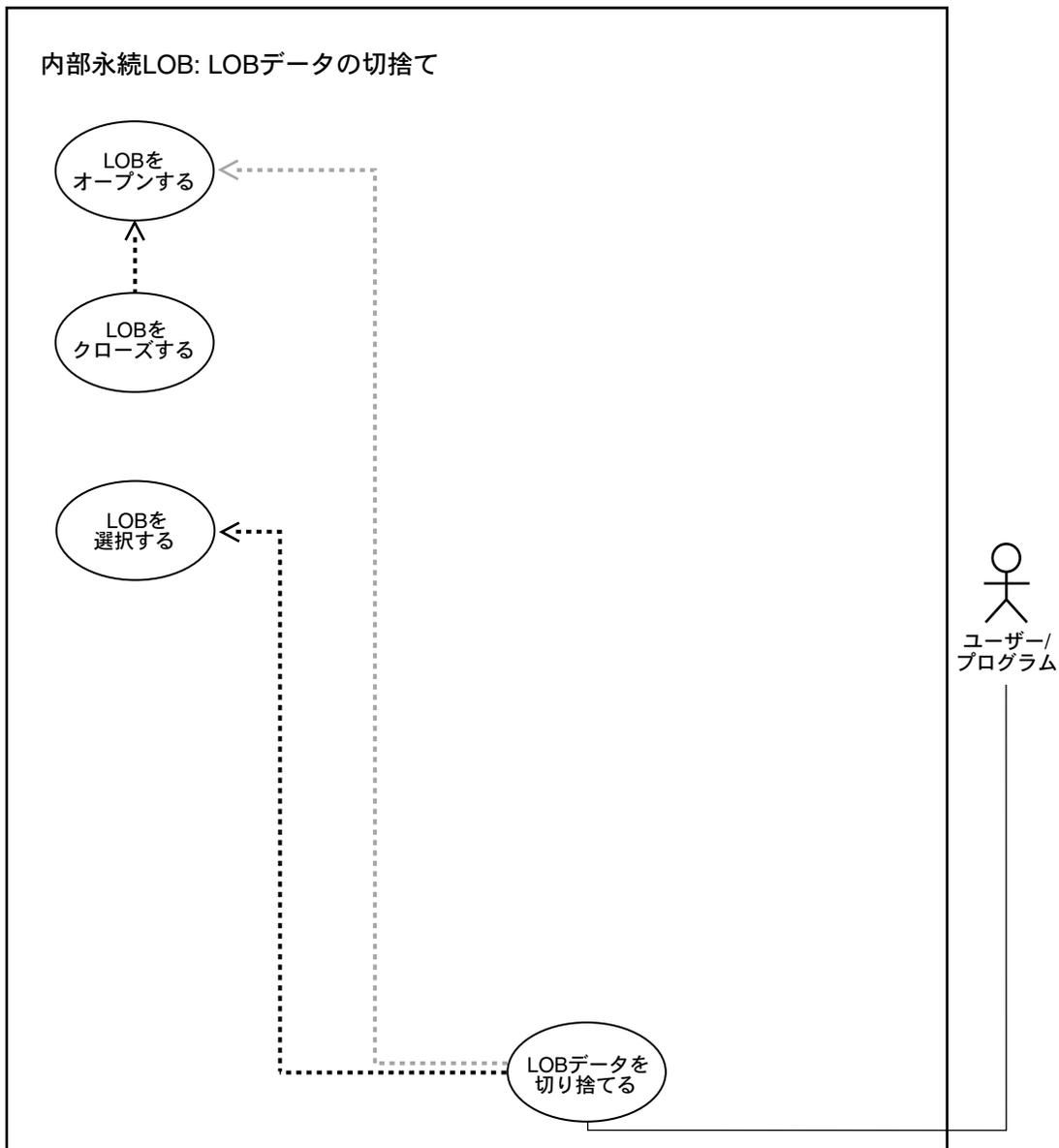
// アウトプット LOB の位置 pos にバッファの内容を書き込みます。
dest_loc.putBytes (pos, buf);

// すべてのストリームおよびハンドルをクローズします。
stmt.close();
conn.commit();
conn.close();

}
catch (SQLException e)
{
    e.printStackTrace();
}
}
```

LOB データの切捨て

図 9-36 ユースケース図 : LOB データの切捨て



参照： 内部永続 LOB に関するすべての基本操作については、9-2 ページの「ユースケース・モデル:内部永続 LOB – 基本操作」を参照してください。

用途

LOB データを切り捨てます。

使用上の注意

更新前の行のロック

PL/SQL DBMS_LOB パッケージまたは OCI を使用して LOB の値を更新する前に、LOB を含む行をロックする必要があります。SQL INSERT 文および UPDATE 文は暗黙的に行をロックしますが、SQL プログラムおよび PL/SQL プログラムでは SQL SELECT FOR UPDATE 文を使用して、また OCI プログラムでは OCI pin または lock 関数を使用して明示的にロックします。更新後のロケータの状態の詳細は、5-6 ページの「更新済ロケータを介した更新済 LOB」を参照してください。

構文

各プログラム環境で使用可能な機能のリストは、第 3 章「LOB プログラム環境」を参照してください。各プログラム環境における構文については、次のマニュアルの項を参照してください。

- PL/SQL (DBMS_LOB パッケージ) : 『Oracle8i PL/SQL パッケージ・プロシージャ・リファレンス』の第 18 章「DBMS_LOB」の「サブプログラムの要約」の TRIM プロシージャ
- C (OCI) : 『Oracle8i コール・インタフェース・プログラマーズ・ガイド』の第 15 章「OCI リレーショナル関数」の「LOB 関数」の OCILobTrim()
- COBOL (Pro*COBOL) : 『Oracle8i Pro*COBOL プリコンパイラ・プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB TRIM (実行可能埋込み SQL 拡張要素)」
- C/C++ (Pro*C/C++) : 『Oracle8i Pro*C/C++ プリコンパイラ・プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB TRIM (実行可能埋込み SQL 拡張要素)」
- Visual Basic (Oracle Objects for OLE オンライン・ヘルプ) : 「Help」の「Contents」タブから、「OO4O Automation Server」>「OBJECTS」>「Orablob」>「METHODS」>「trim」を選択してください。

- Java (JDBC) : 『Oracle8i JDBC 開発者ガイドおよびリファレンス』の第7章「LOB と BFILE の操作」の「BLOB と CLOB の操作」の「BLOB または CLOB 列の作成と移入」、および『Oracle8i SQLJ 開発者ガイドおよびリファレンス』の第5章「型のサポート」の「Oracle の拡張型」の「BLOB、CLOB および BFILE のサポート」

使用例

次の例では、Voiceover_tab 表の Script 列で参照されるテキスト (CLOB データ) にアクセスし、切り捨てます。

例

次のプログラム環境の例が示されています。

- PL/SQL (DBMS_LOB パッケージ) : LOB データの切捨て (9-215 ページ)
- C (OCI) : LOB データの切捨て (9-216 ページ)
- COBOL (Pro*COBOL) : LOB データの切捨て (9-217 ページ)
- C/C++ (Pro*C/C++) : LOB データの切捨て (9-218 ページ)
- Visual Basic (OO4O) : LOB データの切捨て (9-220 ページ)
- Java (JDBC) : LOB データの切捨て (9-221 ページ)

PL/SQL (DBMS_LOB パッケージ) : LOB データの切捨て

```

/* 例のプロシージャ trimLOB_proc は、DBMS_LOB パッケージの
一部ではないことに注意してください。*/
CREATE OR REPLACE PROCEDURE trimLOB_proc IS
  Lob_loc      CLOB;
BEGIN
  /* LOB を選択し、LOB ロケータを取得します。*/
  SELECT Mtab.Voiced_ref.Script INTO Lob_loc FROM Multimedia_tab Mtab
     WHERE Mtab.Clip_ID = 2
     FOR UPDATE;
  /* LOB のオープンはオプションです。*/
  DBMS_LOB.OPEN (Lob_loc, DBMS_LOB.LOB_READWRITE);
  /* LOB データを切り捨てます。*/
  DBMS_LOB.TRIM(Lob_loc,100);
  /* LOB をオープンしている場合、その LOB をクローズする必要があります。*/
  DBMS_LOB.CLOSE (Lob_loc);
COMMIT;
/* 例外処理です。*/

```

```

EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Operation failed');
END;
```

C (OCI) : LOB データの切捨て

```

/* ロケータ変数にロケータを選択します。*/
sb4 select_lock_voice_locator(Lob_loc, errhp, svchp, stmthp)
OCILobLocator *Lob_loc;
OCIError      *errhp;
OCISvcCtx     *svchp;
OCISstmt      *stmthp;
{
    text *sqlstmt =
        (text *) "SELECT Mtab.Voiced_ref.Script \
            FROM Multimedia_tab Mtab WHERE Mtab.Clip_ID = 2 FOR UPDATE";
    OCIDefine *defnpl;
    checkerr (errhp, OCISstmtPrepare(stmthp, errhp, sqlstmt,
        (ub4)strlen((char *)sqlstmt),
        (ub4) OCI_NTV_SYNTAX, (ub4) OCI_DEFAULT));
    checkerr (errhp, OCIDefineByPos(stmthp, &defnpl, errhp, (ub4) 1,
        (dvoid *)&Lob_loc, (sb4) 0,
        (ub2) SQLT_CLOB, (dvoid *) 0,
        (ub2 *) 0, (ub2 *) 0, (ub4) OCI_DEFAULT));

    /* SELECT を実行し、1行フェッチします。*/
    checkerr(errhp, OCISstmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
        (CONST OCISnapshot*) 0, (OCISnapshot*) 0,
        (ub4) OCI_DEFAULT));

    return 0;
}
void trimLob(envhp, errhp, svchp, stmthp)
OCIEnv      *envhp;
OCIError    *errhp;
OCISvcCtx   *svchp;
OCISstmt    *stmthp;
{
    OCILobLocator *Lob_loc;
    unsigned int trimLength;
    (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &Lob_loc,
        (ub4)OCI_DTYPE_LOB, (size_t) 0, (dvoid **) 0);

    /* LOB を選択します。*/
    printf( " select a voice LOB\n");
    select_lock_voice_locator(Lob_loc, errhp, svchp, stmthp);
```

```

/* CLOB をオープンします。*/
checkerr (errhp, (OCILobOpen(svchp, errhp, Lob_loc, OCI_LOB_READWRITE)));

/* LOB を新しい長さに切り捨てます。*/
trimLength = 100;                               /* <LOB の新しい切捨て長さ >*/

printf (" trim the lob to %d bytes\n", trimLength);
checkerr (errhp, OCILobTrim (svchp, errhp, Lob_loc, trimLength ));

/* CLOB をオープンしている場合、その CLOB をクローズする必要があります。*/
checkerr (errhp, OCILobClose(svchp, errhp, Lob_loc));

/* ロケータが保持しているリソースを解放します。*/
(void) OCIDescriptorFree((dvoid *) Lob_loc, (ub4) OCI_DTYPE_LOB);
}

```

COBOL (Pro*COBOL) : LOB データの切捨て

```

IDENTIFICATION DIVISION.
PROGRAM-ID. TRIM-CLOB.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

01 CLOB1          SQL-CLOB.
01 NEW-LEN        PIC S9(9) COMP.
* ソースおよび宛先の位置を定義します。
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.

* CLOB ロケータの割当ておよび初期化を行います。
                   EXEC SQL ALLOCATE :CLOB1 END-EXEC.
                   EXEC SQL WHENEVER NOT FOUND GOTO END-OF-CLOB END-EXEC.
                   EXEC SQL
                       SELECT MTAB.STORY INTO :CLOB1

```

```
FROM MULTIMEDIA_TAB MTAB
WHERE MTAB.CLIP_ID = 2 FOR UPDATE END-EXEC.
```

* CLOB をオープンします。

```
EXEC SQL LOB OPEN :CLOB1 READ WRITE END-EXEC.
```

* いくつかの値を NEW-LEN に移動させます。

```
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++) : LOB データの切捨て

/* 前述の「例」の項でセットアップされたデータ構造に加えて、次のような DML を使用する必要があります。

```
INSERT INTO multimedia_tab VALUES (2, 'The quick brown fox jumped over the lazy
dog', empty_clob(), NULL, empty_blob(), empty_blob(), NULL, NULL, NULL, NULL);
INSERT INTO voiceover_tab VALUES (voiced_typ('hello', (SELECT story FROM
multimedia_tab WHERE clip_id = 2), 'world', 1, NULL))
UPDATE multimedia_tab SET voiced_ref = (SELECT REF(r) FROM voiceover_tab r WHERE
r.take = 1) WHERE clip_id = 2
```

Then create this text file, pers_trim.typ, containing:

```
case=lower
```

```
type voiced_typ
```

次に、次のオブジェクト型変換コマンドを実行します。

```
ott intyp=pers_trim.typ outtyp=pers_trim_o.typ
```

```
hfile=pers_trim.h code=c user=samp/samp
```

```
*/
```

```
#include "pers_trim.h"
#include <stdio.h>
#include <sqlca.h>
void Sample_Error()
{
    EXEC SQL WHENEVER SQLERROR CONTINUE;
    printf("sqlcode = %ld\n", sqlca.sqlcode);
    printf("%.*s\n", sqlca.sqlerrm.sqlerrml, sqlca.sqlerrm.sqlerrmc);
    EXEC SQL ROLLBACK WORK RELEASE;
    exit(1);
}

void trimLOB_proc()
{
    voiced_typ_ref *vt_ref;
    voiced_typ *vt_typ;
    OCIClobLocator *Lob_loc;
    unsigned int Length, trimLength;

    EXEC SQL WHENEVER SQLERROR DO Sample_Error();
    EXEC SQL ALLOCATE :Lob_loc;
    EXEC SQL ALLOCATE :vt_ref;
    EXEC SQL ALLOCATE :vt_typ;

    /* 結合 SQL を使用して、REF を取り出します。 */
    EXEC SQL SELECT Mtab.Voiced_ref INTO :vt_ref
        FROM Multimedia_tab Mtab WHERE Mtab.Clip_ID = 2 FOR UPDATE;

    /* ナビゲーション・インタフェースを使用して、オブジェクトの参照を解除します。 */
    EXEC SQL OBJECT Deref :vt_ref INTO :vt_typ FOR UPDATE;
    Lob_loc = vt_typ->script;

    /* LOB のオープンはオプションです。 */
    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);

    /* LOB を新しい長さに切り捨てます。 */
    EXEC SQL LOB TRIM :Lob_loc TO :trimLength;

    /* LOB をオープンしている場合、その LOB をクローズする必要があります。 */
    EXEC SQL LOB CLOSE :Lob_loc;
```

```
/* オブジェクトに Modified (使用済) のマークを付けます。*/
EXEC SQL OBJECT UPDATE :vt_typ;

/* オブジェクト・キャッシュ内の LOB への変更をフラッシュします。*/
EXEC SQL OBJECT FLUSH :vt_typ;

/* 新しい (変更された) 長さを表示します。*/
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);

/* オブジェクトおよび LOB ロケータを解放します。*/
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 (0040) : LOB データの切捨て

```
Dim MySession As OraSession
Dim OraDb As OraDatabase
Dim OraDyn As OraDynaset, OraSound1 As OraBlob, OraSoundClone As OraBlob

Set MySession = CreateObject("OracleInProcServer.XOraSession")
Set OraDb = MySession.OpenDatabase("exampledb", "samp/samp", 0&)
Set OraDyn = OraDb.CreateDynaset(
    "SELECT * FROM Multimedia_tab ORDER BY clip_id", ORADYN_DEFAULT)
Set OraSound1 = OraDyn.Fields("Sound").Value

OraDyn.Edit
OraSound1.Trim 10
OraDyn.Update
```

Java (JDBC) : LOB データの切捨て

```
// Java IOクラス :
import java.io.InputStream;
import java.io.OutputStream;

// コア JDBC クラス :
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 固有の JDBC クラス :
import oracle.sql.*;
import oracle.jdbc.driver.*;

public class Ex2_141
{
    static final int MAXBUFSIZE = 32767;
    public static void main (String args [])
        throws Exception
    {
        // Oracle JDBC Driver をロードします。
        DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

        // データベースに接続します。
        Connection conn =
            DriverManager.getConnection ("jdbc:oracle:oci8:@", "samp", "samp");

        // 自動コミットがオフの場合、高速になります。
        conn.setAutoCommit (false);

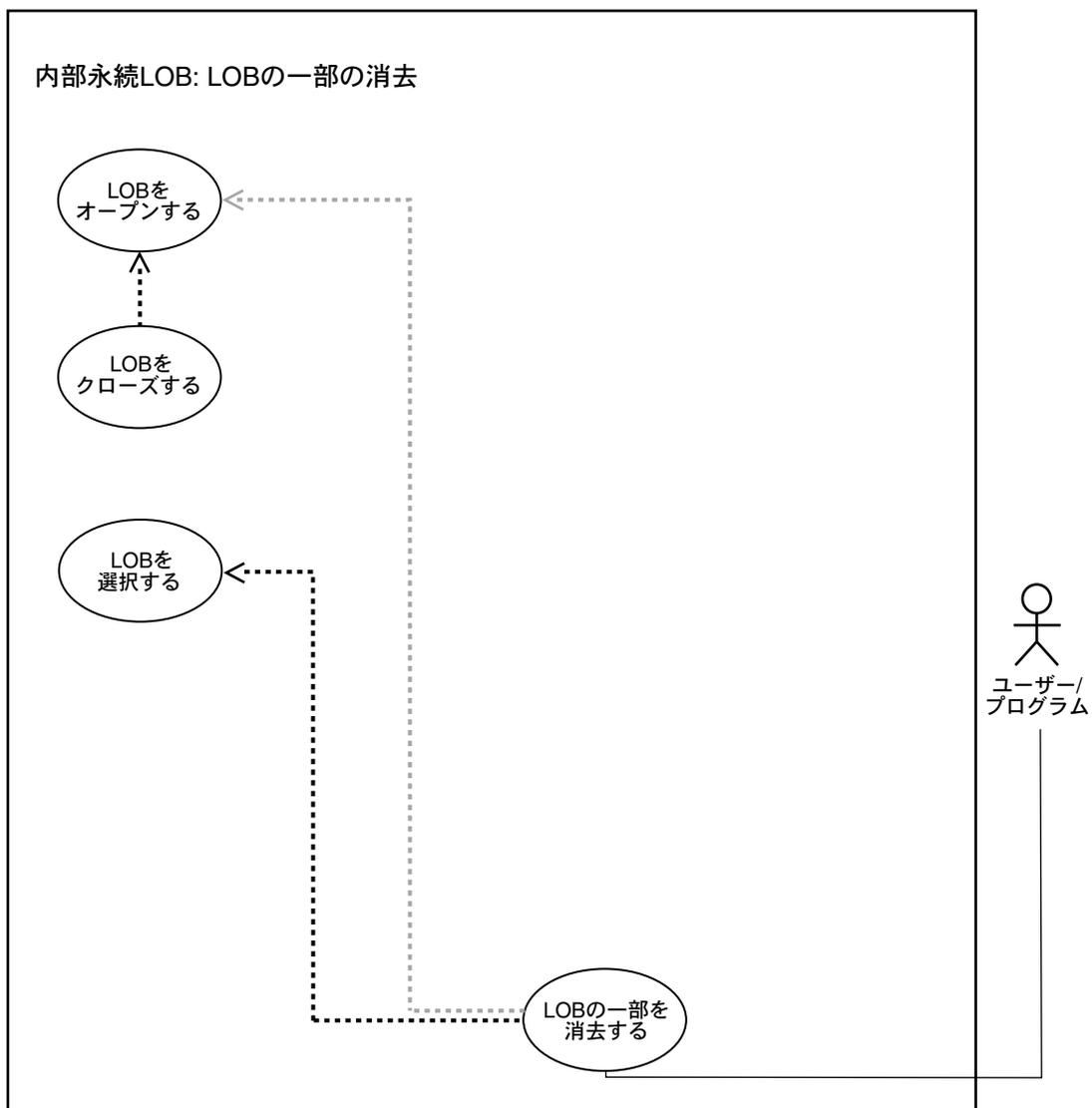
        // 文を作成します。
        Statement stmt = conn.createStatement ();
        try
        {
            CLOB lob_loc = null;

            ResultSet rset = stmt.executeQuery (
                "SELECT mtab.voiced_ref.script FROM multimedia_tab mtab
                WHERE mtab.clip_id = 2 FOR UPDATE");
            if (rset.next())
            {
                lob_loc = ((OracleResultSet)rset).getCLOB (1);
            }
        }
    }
}
```

```
    }  
  
    // READWRITE 用の LOB をオープンします。  
    OracleCallableStatement cstmt = (OracleCallableStatement)  
        conn.prepareCall ("BEGIN DBMS_LOB.OPEN(? ,DBMS_LOB.LOB_READWRITE);  
        END;");  
    cstmt.setCLOB(1, lob_loc);  
    cstmt.execute();  
  
    // LOB を 400 の長さに切り捨てます。  
    cstmt = (OracleCallableStatement)  
        conn.prepareCall ("BEGIN DBMS_LOB.TRIM(? , 400); END;");  
    cstmt.setCLOB(1, lob_loc);  
    cstmt.execute();  
  
    // LOB をクローズします。  
    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();  
    }  
    }  
}
```

LOB の一部の消去

図 9-37 ユースケース図：LOB の一部の消去



参照： 内部永続 LOB に関するすべての基本操作については、9-2 ページの「[ユースケース・モデル:内部永続 LOB – 基本操作](#)」を参照してください。

用途

LOB の一部を消去します。

使用上の注意

更新前の行のロック

PL/SQL DBMS_LOB パッケージまたは OCI を使用して LOB の値を更新する前に、LOB を含む行をロックする必要があります。SQL INSERT 文および UPDATE 文は暗黙的に行をロックしますが、SQL プログラムおよび PL/SQL プログラムでは SQL SELECT FOR UPDATE 文を使用して、また OCI プログラムでは OCI pin または lock 関数を使用して明示的にロックします。

更新後のロケータの状態の詳細は、5-6 ページの「[更新済ロケータを介した更新済 LOB](#)」を参照してください。

構文

各プログラム環境で使用可能な機能のリストは、[第 3 章「LOB プログラム環境](#)」を参照してください。各プログラム環境における構文については、次のマニュアルの項を参照してください。

- PL/SQL (DBMS_LOB パッケージ) : 『Oracle8i PL/SQL パッケージ・プロシージャ・リファレンス』の第 18 章「DBMS_LOB」の「サブプログラムの要約」の ERASE プロシージャ
- C (OCI) : 『Oracle8i コール・インタフェース・プログラマーズ・ガイド』の第 15 章「OCI リレーショナル関数」の「LOB 関数」の OCILobErase()
- COBOL (Pro*COBOL) : 『Oracle8i Pro*COBOL プリコンパイラ・プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB ERASE (実行可能埋込み SQL 拡張要素)」
- C/C++ (Pro*C/C++) : 『Oracle8i Pro*C/C++ プリコンパイラ・プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB ERASE (実行可能埋込み SQL 拡張要素)」
- Visual Basic (Oracle Objects for OLE オンライン・ヘルプ) : 「Help」の「Contents」タブから、「OO4O Automation Server」>「OBJECTS」>「Oralob」>「METHODS」>「erase」を選択してください。

- Java (JDBC) : 『Oracle8i JDBC 開発者ガイドおよびリファレンス』の第7章「LOB と BFILE の操作」の「BLOB と CLOB の操作」の「BLOB または CLOB 列の作成と移入」、および『Oracle8i SQLJ 開発者ガイドおよびリファレンス』の第5章「型のサポート」の「Oracle の拡張型」の「BLOB、CLOB および BFILE のサポート」

使用例

次の例では、サウンド (Sound) の一部を消去します。

例

次のプログラム環境の例が示されています。

- PL/SQL (DBMS_LOB パッケージ) : LOB の一部の消去 (9-225 ページ)
- C (OCI) : LOB の一部の消去 (9-226 ページ)
- COBOL (Pro*COBOL) : LOB の一部の消去 (9-227 ページ)
- C/C++ (Pro*C/C++) : LOB の一部の消去 (9-228 ページ)
- Visual Basic (OO4O) : LOB の一部の消去 (9-229 ページ)
- Java (JDBC) : LOB の一部の消去 (9-230 ページ)

PL/SQL (DBMS_LOB パッケージ) : LOB の一部の消去

```

/* 例のプロシージャ eraseLOB_proc は、DBMS_LOB パッケージの
一部ではないことに注意してください。*/
CREATE OR REPLACE PROCEDURE eraseLOB_proc IS
  Lob_loc      BLOB;
  Amount       INTEGER := 3000;
BEGIN
  /* LOB を選択し、LOB ロケータを取得します。*/
  SELECT Sound INTO lob_loc FROM Multimedia_tab
     WHERE Clip_ID = 1
     FOR UPDATE;
  /* LOB のオープンはおプションです。*/
  DBMS_LOB.OPEN (Lob_loc, DBMS_LOB.LOB_READWRITE);
  /* データを消去します。*/
  DBMS_LOB.ERASE (Lob_loc, Amount, 2000);
  /* LOB をオープンしている場合、その LOB をクローズする必要があります。*/
  DBMS_LOB.CLOSE (Lob_loc);
COMMIT;

```

```

/* 例外処理です。*/
EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Operation failed');
END;

```

C (OCI) : LOB の一部 of 消去

```

/* ロケータ変数にロケータを選択します。*/
sb4 select_lock_sound_locator(Lob_loc, errhp, svchp, stmthp)
OCILobLocator *Lob_loc;
OCIError      *errhp;
OCISvcCtx     *svchp;
OCIStmt       *stmthp;
{
    text *sqlstmt =
        (text *)"SELECT Sound FROM Multimedia_tab WHERE Clip_ID=1 FOR UPDATE";
    OCIDefine *defnp1;

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

    /* SELECT を実行し、1行フェッチします。*/
    checkerr(errhp, OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                                   (CONST OCISnapshot*) 0, (OCISnapshot*) 0,
                                   (ub4) OCI_DEFAULT));

    return 0;
}

void eraseLob(envhp, errhp, svchp, stmthp)
OCIEnv      *envhp;
OCIError    *errhp;
OCISvcCtx   *svchp;
OCIStmt     *stmthp;
{
    OCILobLocator *Lob_loc;
    ub4 amount = 3000;
    ub4 offset = 2000;

    OCILobLocator *Lob_Loc;

```

```

(void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &Lob_loc,
                          (ub4)OCI_DTYPE_LOB, (size_t) 0, (dvoid **) 0);

/* LOB を選択します。*/
printf( " select and lock a sound LOB\n");
select_lock_sound_locator(Lob_loc, errhp, svchp, stmthp);

/* BLOB をオープンします。*/
checkerr (errhp, (OCILobOpen(svchp, errhp, Lob_loc, OCI_LOB_READWRITE)));

/* 指定したオフセットから開始して、データを消去します。*/
printf(" erase %d bytes from the sound Lob\n", amount);
checkerr (errhp, OCILobErase (svchp, errhp, Lob_loc, &amount, offset ));

/* BLOB をオープンしている場合、その BLOB をクローズする必要があります。*/
checkerr (errhp, OCILobClose(svchp, errhp, Lob_loc));

/* ロケータが保持しているリソースを解放します。*/
(void) OCIDescriptorFree((dvoid *) Lob_loc, (ub4) OCI_DTYPE_LOB);

return;
}

```

COBOL (Pro*COBOL) : LOB の一部の消去

```

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.
* BLOB ロケータの割当ておよび初期化を行います。
          EXEC SQL ALLOCATE :BLOB1 END-EXEC.

```

```
EXEC SQL WHENEVER NOT FOUND GOTO END-OF-BLOB END-EXEC.  
EXEC SQL  
    SELECT SOUND INTO :BLOB1  
    FROM MULTIMEDIA_TAB MTAB  
    WHERE MTAB.CLIP_ID = 2 FOR UPDATE  
END-EXEC.
```

* BLOB をオープンします。

```
EXEC SQL LOB OPEN :BLOB1 READ WRITE END-EXEC.
```

* いくつかの値を AMT および OFFSET に移動させます。

```
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 SQLERRM.  
EXEC SQL ROLLBACK WORK RELEASE END-EXEC.  
STOP RUN.
```

C/C++ (Pro*C/C++) : LOB の一部の消去

```
#include <oci.h>  
#include <stdio.h>  
#include <sqlca.h>  
  
void Sample_Error()  
{  
    EXEC SQL WHENEVER SQLERROR CONTINUE;  
    printf("%.*s\n", sqlca.sqlerrm.sqlerrml, sqlca.sqlerrm.sqlerrmc);  
    EXEC SQL ROLLBACK WORK RELEASE;
```

```

    exit(1);
}

void eraseLob_proc()
{
    OCIBlobLocator *Lob_loc;
    int Amount = 5;
    int Offset = 5;

    EXEC SQL WHENEVER SQLERROR DO Sample_Error();
    EXEC SQL ALLOCATE :Lob_loc;
    EXEC SQL SELECT Sound INTO :Lob_loc
        FROM Multimedia_tab WHERE Clip_ID = 1 FOR UPDATE;
    /* LOB のオープンはオプションです。 */
    EXEC SQL LOB OPEN :Lob_loc READ WRITE;
    /* 指定したオフセットから開始して、データを消去します。 */
    EXEC SQL LOB ERASE :Amount FROM :Lob_loc AT :Offset;
    printf("Erased %d bytes\n", Amount);
    /* LOB をオープンしている場合、その LOB をクローズする必要があります。 */
    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 (0040) : LOB の一部の消去

```

Dim MySession As OraSession
Dim OraDb As OraDatabase
Dim OraDyn As OraDynaset, OraSound1 As OraBlob, OraSoundClone As OraBlob

Set MySession = CreateObject("OracleInProcServer.XOraSession")
Set OraDb = MySession.OpenDatabase("exampledb", "samp/samp", 0&)

Set OraDyn = OraDb.CreateDynaset("SELECT * FROM Multimedia_tab ORDER BY clip_id",
ORADYN_DEFAULT)
Set OraSound1 = OraDyn.Fields("Sound").Value

```

```
'100 バイト以降の 10 バイトを消去します。
OraDyn.Edit
OraSound1.Erase 10, 100
OraDyn.Update
```

Java (JDBC) : LOB の一部の消去

```
import java.io.InputStream;
import java.io.OutputStream;

// コア JDBC クラス :
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 固有の JDBC クラス :
import oracle.sql.*;
import oracle.jdbc.driver.*;

public class Ex2_145
{
    static final int MAXBUFSIZE = 32767;
    public static void main (String args [])
        throws Exception
    {
        // Oracle JDBC Driver をロードします。
        DriverManager.registerDriver (new oracle.jdbc.driver.OracleDriver ());

        // データベースに接続します。
        Connection conn =
            DriverManager.getConnection ("jdbc:oracle:oci8:@", "samp", "samp");

        // 自動コミットがオフの場合、高速になります。
        conn.setAutoCommit (false);

        // 文を作成します。
        Statement stmt = conn.createStatement ();
        try
        {
            BLOB lob_loc = null;
            int eraseAmount = 30;
            ResultSet rset = stmt.executeQuery (
```

```
        "SELECT sound FROM multimedia_tab WHERE clip_id = 2 FOR UPDATE");
    if (rset.next())
    {
        lob_loc = ((OracleResultSet)rset).getBLOB (1);
    }

    // READWRITE用のLOBをオープンします。
    OracleCallableStatement cstmt = (OracleCallableStatement)
        conn.prepareCall ("BEGIN DBMS_LOB.OPEN(?, "
            +"DBMS_LOB.LOB_READWRITE); END;");
    cstmt.setBLOB(1, lob_loc);
    cstmt.execute();

    // オフセット2,000から開始して、eraseAmountバイト消去します。
    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);

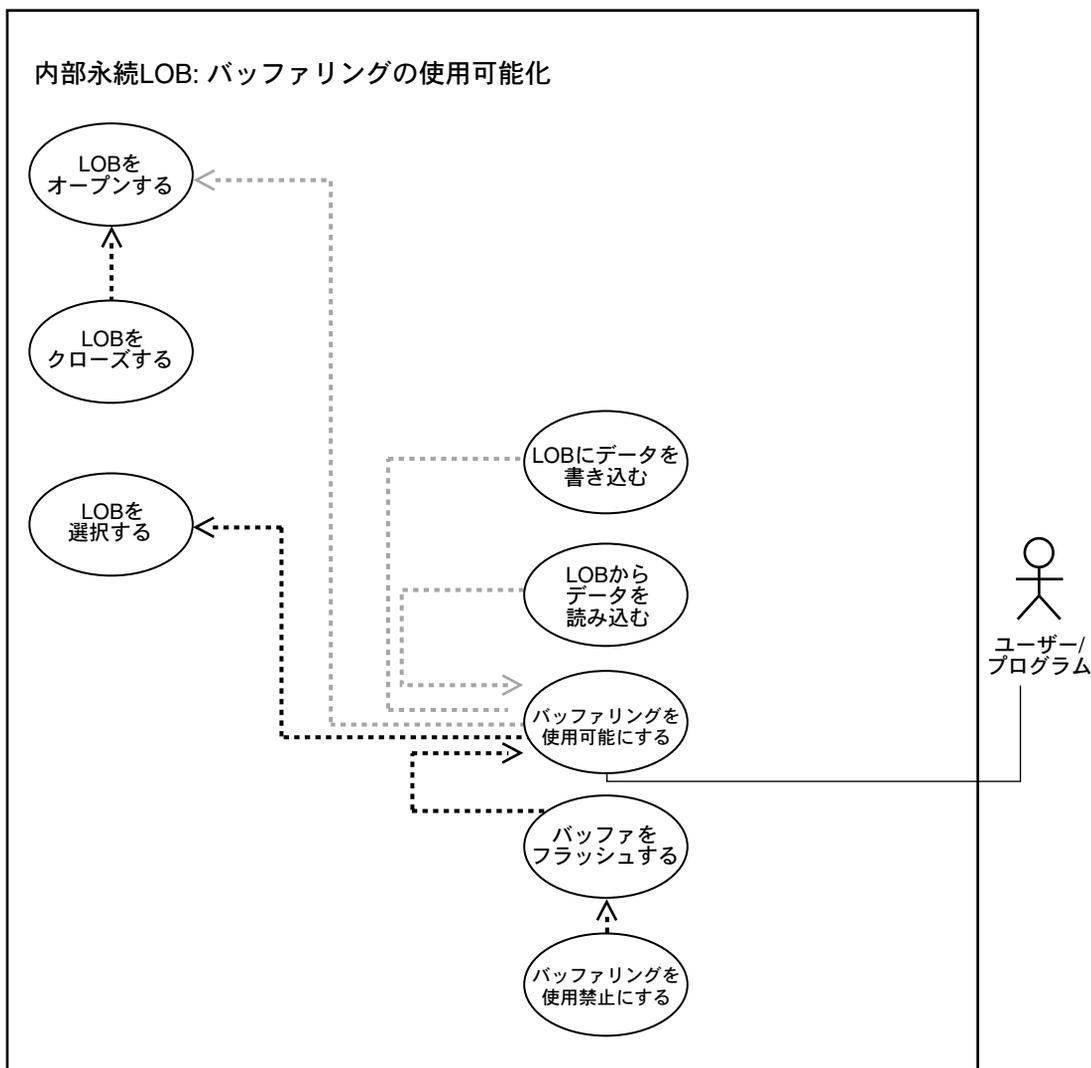
    // LOBをクローズします。
    cstmt = (OracleCallableStatement) conn.prepareCall (
        "BEGIN DBMS_LOB.CLOSE(?); END;");
    cstmt.setBLOB(1, lob_loc);
    cstmt.execute();

    conn.commit();
    stmt.close();
    cstmt.close();
    conn.commit();
    conn.close();

    }
    catch (SQLException e)
    {
        e.printStackTrace();
    }
}
}
```

LOB バッファリングの使用可能化

図 9-38 ユースケース図：LOB バッファリングの使用可能化



参照： 内部永続 LOB に関するすべての基本操作については、9-2 ページの「[ユースケース・モデル:内部永続 LOB – 基本操作](#)」を参照してください。

用途

LOB バッファリングを使用可能にします。

使用上の注意

バッファリングを使用可能にして、少量のデータの読み込みおよび書き込みを実行します。これらの作業を完了したら、他の LOB 操作を行う前にバッファリングを使用禁止にする必要があります。

注意：

- バッファをフラッシュし、変更を永続的にする必要があります。
 - バッファリングを使用可能にして、チェックインおよびチェックアウトで囲まれたストリーム読み込みおよび書き込みを実行しないでください。
-
-

LOB バッファリング・サブシステムの詳細は、5-19 ページの「[LOB バッファリング・サブシステム](#)」を参照してください。

構文

各プログラム環境で使用可能な機能のリストは、[第 3 章「LOB プログラム環境](#)」を参照してください。各プログラム環境における構文については、次のマニュアルの項を参照してください。

- PL/SQL (DBMS_LOB パッケージ) : 参照マニュアルはありません。
- C (OCI) : 『Oracle8i コール・インタフェース・プログラマーズ・ガイド』の第 15 章「OCI リレーショナル関数」の「LOB 関数」の OCIEnableLobBuffering()、OCIDisableLobBuffering()、OCIFlushBuffer()
- COBOL (Pro*COBOL) : 『Oracle8i Pro*COBOL プリコンパイラ・プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB ENABLE BUFFERING (実行可能埋込み SQL 拡張要素)」
- C/C++ (Pro*C/C++) : 『Oracle8i Pro*C/C++ プリコンパイラ・プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB ENABLE BUFFERING (実行可能埋込み SQL 拡張要素)」

- Visual Basic (Oracle Objects for OLE オンライン・ヘルプ) : 「Help」の「Contents」タブから、「OO4O Automation Server」>「OBJECTS」>「Oralob」>「METHODS」>「EnableBuffering」を選択してください。
- Java (JDBC) : 参照マニュアルはありません。

使用例

次の例は、ここに記述されている方法および関連する方法で開発された Sound についてのバッファリングの管理の一部です。

例

次のプログラム環境の例が示されています。

- PL/SQL (DBMS_LOB パッケージ) : 今回のリリースでは例は記載されていません。
- C (OCI) : [LOB バッファリングの使用可能化 \(9-234 ページ\)](#)
- COBOL (Pro*COBOL) : [LOB バッファリングの使用可能化 \(9-234 ページ\)](#)
- C/C++ (Pro*C/C++) : [LOB バッファリングの使用可能化 \(9-236 ページ\)](#)
- Visual Basic (OO4O) : [LOB バッファリングの使用可能化 \(9-234 ページ\)](#)
- Java (JDBC) : 今回のリリースでは例は記載されていません。

C (OCI) : LOB バッファリングの使用可能化

参照: 9-244 ページの「[LOB バッファリングの使用禁止化](#)」を参照してください。

COBOL (Pro*COBOL) : LOB バッファリングの使用可能化

```
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.

* BLOB ロケータの割当ておよび初期化を行います。
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.

* BLOB をオープンし、バッファリングを使用可能にします。
EXEC SQL LOB OPEN :BLOB1 READ WRITE END-EXEC.
EXEC SQL
    LOB ENABLE BUFFERING :BLOB1 END-EXEC.

* BLOB にいくつかのデータを書き込みます。
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.

* バッファリングされた書き込みをフラッシュします。
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++) : LOB バッファリングの使用可能化

```
#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;
    /* このデータ型では、データ型を等価にする必要があります。*/
    char Buffer[BufferLength];
    EXEC SQL VAR Buffer IS RAW(BufferLength);

    EXEC SQL WHENEVER SQLERROR DO Sample_Error();
    /* LOB の割当ておよび初期化を行います。*/
    EXEC SQL ALLOCATE :Lob_loc;
    EXEC SQL SELECT Sound INTO :Lob_loc
        FROM Multimedia_tab WHERE Clip_ID = 1 FOR UPDATE;

    /* LOB バッファリング・サブシステムを使用可能にします。*/
    EXEC SQL LOB ENABLE BUFFERING :Lob_loc;
    memset((void *)Buffer, 0, BufferLength);
    for (multiple = 0; multiple < 8; multiple++)
    {
        /* LOB にデータを書き込みます。*/
        EXEC SQL LOB WRITE ONE :Amount
            FROM :Buffer INTO :Lob_loc AT :Position;
        Position += BufferLength;
    }
    /* バッファの内容をフラッシュし、これらのリソースを解放します。*/
    EXEC SQL LOB FLUSH BUFFER :Lob_loc FREE;
    /* LOB バッファリング・サブシステムを使用禁止にします。*/
    EXEC SQL LOB DISABLE BUFFERING :Lob_loc;
}
```

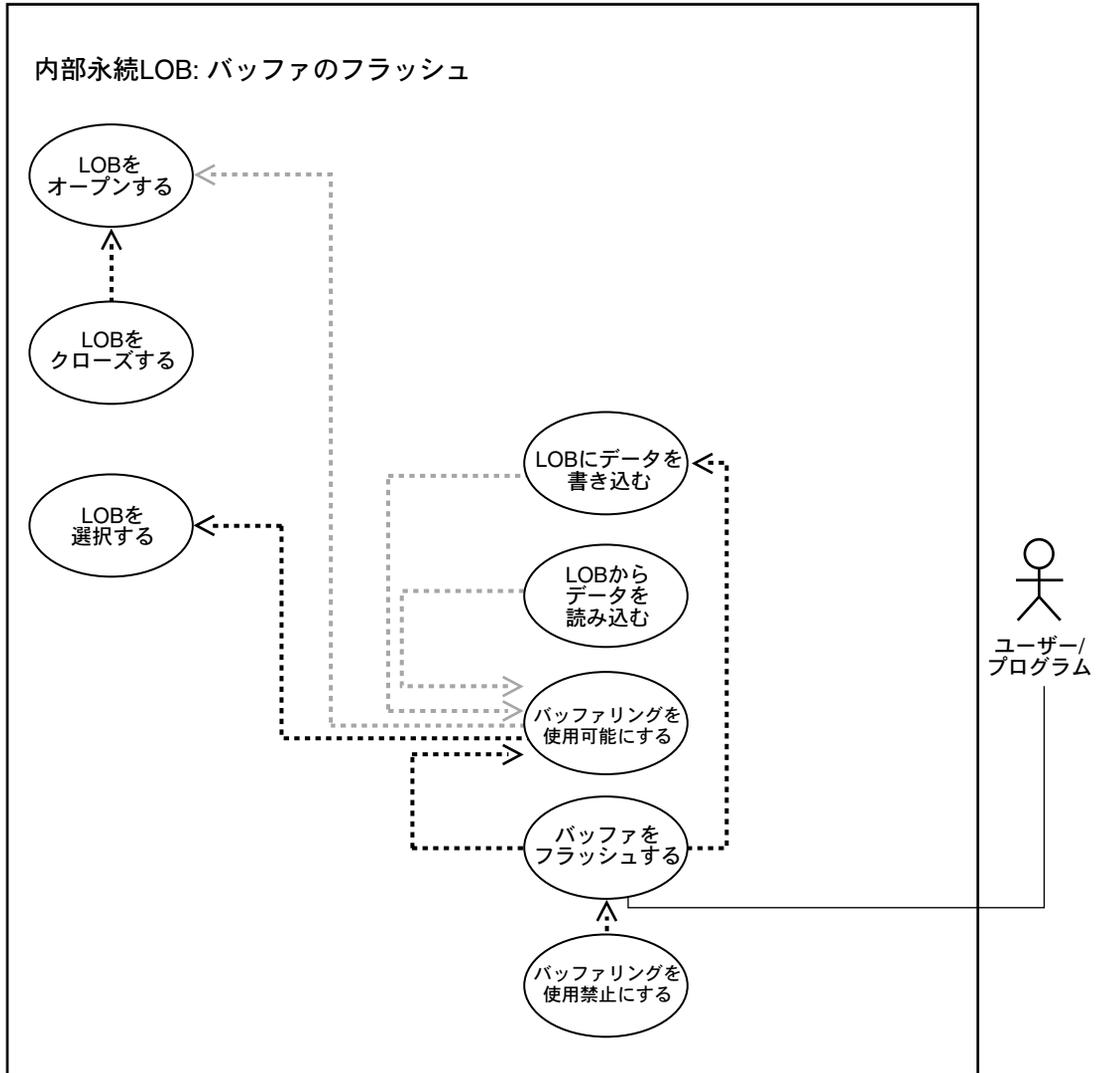
```
/* ロケータが保持しているリソースを解放します。*/  
EXEC SQL FREE :Lob_loc;  
}  
  
void main()  
{  
    char *samp = "samp/samp";  
    EXEC SQL CONNECT :samp;  
    enableBufferingLOB_proc();  
    EXEC SQL ROLLBACK WORK RELEASE;  
}
```

Visual Basic (0040) : LOB バッファリングの使用可能化

```
Dim MySession As OraSession  
Dim OraDb As OraDatabase  
Dim OraDyn As OraDynaset, OraSound1 As OraBlob, OraSoundClone As OraBlob  
  
Set MySession = CreateObject("OracleInProcServer.XOraSession")  
Set OraDb = MySession.OpenDatabase("exampledb", "samp/samp", 0&)  
Set OraDyn = OraDb.CreateDynaset(  
    "SELECT * FROM Multimedia_tab ORDER BY clip_id", ORADYN_DEFAULT)  
Set OraSound1 = OraDyn.Fields("Sound").Value  
' バッファリングを使用可能にします。  
OraSound1.EnableBuffering
```

バッファのフラッシュ

図 9-39 ユースケース図: バッファのフラッシュ



参照： 内部永続 LOB に関するすべての基本操作については、9-2 ページの「[ユースケース・モデル:内部永続 LOB – 基本操作](#)」を参照してください。

用途

LOB バッファをフラッシュします。

使用上の注意

バッファリングを使用可能にして、少量のデータの読み込みおよび書き込みを実行します。これらの作業を完了したら、他の LOB 操作を行う前にバッファリングを使用禁止にする必要があります。

注意：

- バッファをフラッシュし、変更を永続的にする必要があります。
 - バッファリングを使用可能にして、チェックインおよびチェックアウトで囲まれたストリーム読み込みおよび書き込みを実行しないでください。
-
-

LOB バッファリング・サブシステムの詳細は、5-19 ページの「[LOB バッファリング・サブシステム](#)」を参照してください。

構文

各プログラム環境で使用可能な機能のリストは、[第 3 章「LOB プログラム環境](#)」を参照してください。各プログラム環境における構文については、次のマニュアルの項を参照してください。

- PL/SQL (DBMS_LOB パッケージ) : 参照マニュアルはありません。
- C (OCI) : 『Oracle8i コール・インタフェース・プログラマーズ・ガイド』の第 15 章「OCI リレーショナル関数」の「LOB 関数」の OCIEnableLobBuffering()、OCIDisableLobBuffering()、OCIFlushBuffer()
- COBOL (Pro*COBOL) : 『Oracle8i Pro*COBOL プリコンパイラ・プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB FLUSH BUFFER (実行可能埋込み SQL 拡張要素)」
- C/C++ (Pro*C/C++) : 『Oracle8i Pro*C/C++ プリコンパイラ・プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB FLUSH BUFFER (実行可能埋込み SQL 拡張要素)」

- Visual Basic (Oracle Objects for OLE オンライン・ヘルプ) : 「Help」の「Contents」タブから、「OO4O Automation Server」>「OBJECTS」>「Orablob」>「METHODS」>「FlushBuffer」を選択してください。
- Java (JDBC) : 参照マニュアルはありません。

使用例

次の例は、ここに記述されている方法および関連する方法で開発された Sound についてのバッファリングの管理の一部です。

例

次のプログラム環境の例が示されています。

- PL/SQL (DBMS_LOB パッケージ) : 今回のリリースでは例は記載されていません。
- [C \(OCI\) : バッファのフラッシュ \(9-240 ページ\)](#)
- [COBOL \(Pro*COBOL\) : バッファのフラッシュ \(9-240 ページ\)](#)
- [C/C++ \(Pro*C/C++\) : バッファのフラッシュ \(9-242 ページ\)](#)
- Visual Basic (OO4O) : 今回のリリースでは例は記載されていません。
- Java (JDBC) : 今回のリリースでは例は記載されていません。

C (OCI) : バッファのフラッシュ

参照: 9-244 ページの「[LOB バッファリングの使用禁止化](#)」を参照してください。

COBOL (Pro*COBOL) : バッファのフラッシュ

```
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.
```

- * BLOB ロケータの割当ておよび初期化を行います。

```
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.
```

- * BLOB をオープンし、バッファリングを使用可能にします。

```
EXEC SQL LOB OPEN :BLOB1 READ WRITE END-EXEC.  
EXEC SQL LOB ENABLE BUFFERING :BLOB1 END-EXEC.
```

- * BLOB にいくつかのデータを書き込みます。

```
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.
```

- * バッファリングされた書き込みをフラッシュします。

```
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++) : バッファのフラッシュ

```
#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;  
  
    /* このデータ型では、データ型を等価にする必要があります。*/  
    char Buffer[BufferLength];  
    EXEC SQL VAR Buffer IS RAW(BufferLength);  
    EXEC SQL WHENEVER SQLERROR DO Sample_Error();  
  
    /* LOB の割当ておよび初期化を行います。*/  
    EXEC SQL ALLOCATE :Lob_loc;  
    EXEC SQL SELECT Sound INTO :Lob_loc  
        FROM Multimedia_tab WHERE Clip_ID = 1 FOR UPDATE;  
  
    /* LOB バッファリング・サブシステムを使用可能にします。*/  
    EXEC SQL LOB ENABLE BUFFERING :Lob_loc;  
    memset((void *)Buffer, 0, BufferLength);  
    for (multiple = 0; multiple < 8; multiple++)
```

```
{
    /* LOB にデータを書き込みます。 */
    EXEC SQL LOB WRITE ONE :Amount
        FROM :Buffer INTO :Lob_loc AT :Position;
    Position += BufferLength;
}
/* バッファの内容をフラッシュし、これらのリソースを解放します。 */
EXEC SQL LOB FLUSH BUFFER :Lob_loc FREE;
/* LOB バッファリング・サブシステムを使用禁止にします。 */
EXEC SQL LOB DISABLE BUFFERING :Lob_loc;
/* ロケータが保持しているリソースを解放します。 */
EXEC SQL FREE :Lob_loc;
}

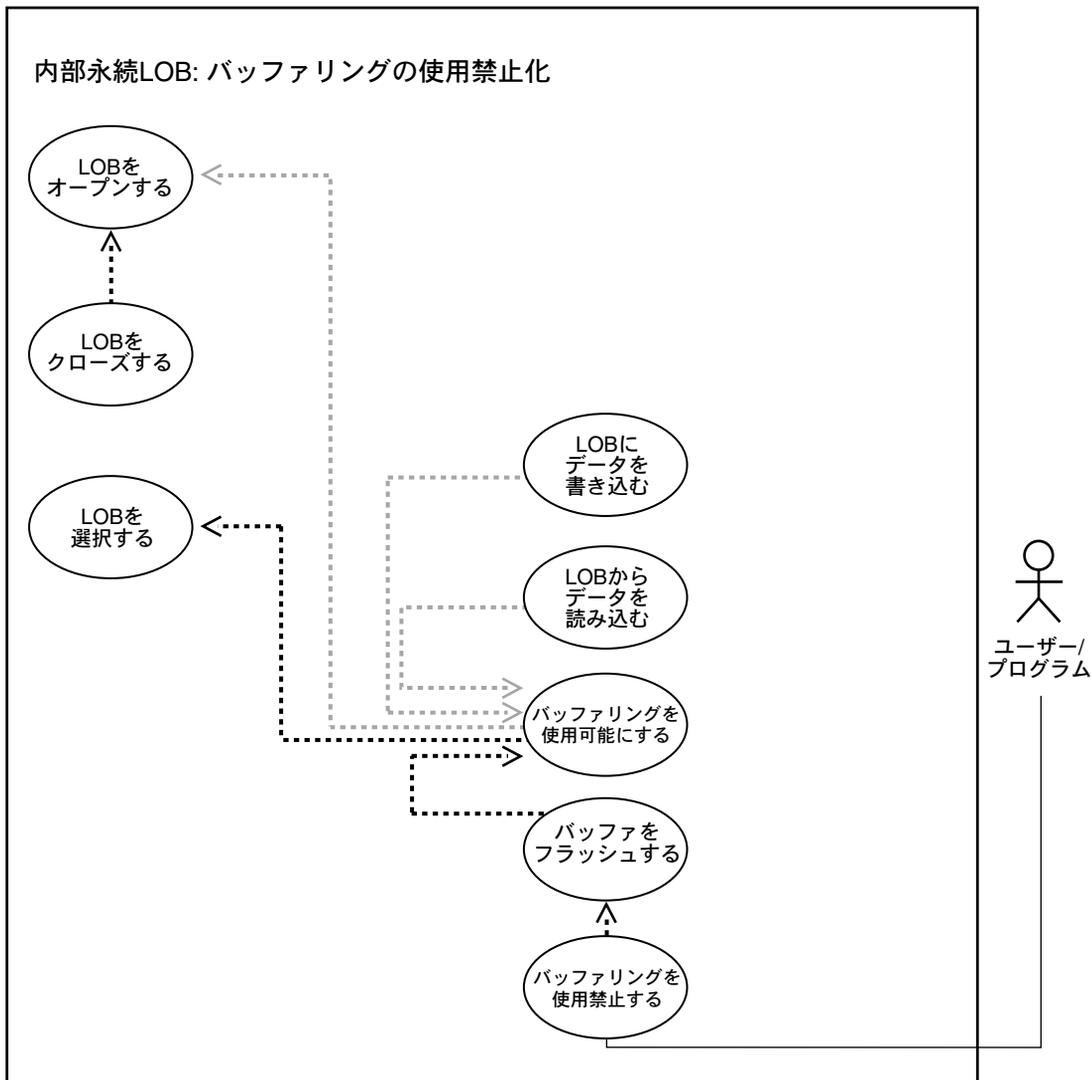
void main()
{
    char *samp = "samp/samp";
    EXEC SQL CONNECT :samp;
    flushBufferingLOB_proc();
    EXEC SQL ROLLBACK WORK RELEASE;
}
```

Visual Basic (OO40) : バッファのフラッシュ

注意： Visual Basic (OO40) は、次回のリリースで使用可能になります。

LOB バッファリングの使用禁止化

図 9-40 ユースケース図：LOB バッファリングの使用禁止化



参照： 内部永続 LOB に関するすべての基本操作については、9-2 ページの「[ユースケース・モデル:内部永続 LOB – 基本操作](#)」を参照してください。

用途

LOB バッファリングを使用禁止にします。

使用上の注意

バッファリングを使用可能にして、少量のデータの読み込みおよび書き込みを実行します。これらの作業を完了したら、他の LOB 操作を行う前にバッファリングを使用禁止にする必要があります。

注意：

- バッファをフラッシュし、変更を永続的にする必要があります。
 - バッファリングを使用可能にして、チェックインおよびチェックアウトで囲まれたストリーム読み込みおよび書き込みを実行しないでください。
-
-

LOB バッファリング・サブシステムの詳細は、5-19 ページの「[LOB バッファリング・サブシステム](#)」を参照してください。

構文

各プログラム環境で使用可能な機能のリストは、[第 3 章「LOB プログラム環境](#)」を参照してください。各プログラム環境における構文については、次のマニュアルの項を参照してください。

- PL/SQL (DBMS_LOB パッケージ) : 参照マニュアルはありません。
- C (OCI) : 『Oracle8i コール・インタフェース・プログラマーズ・ガイド』の第 15 章「OCI リレーショナル関数」の「LOB 関数」の OCIEnableLobBuffering()、OCIDisableLobBuffering()、OCIFlushBuffer()
- COBOL (Pro*COBOL) : 『Oracle8i Pro*COBOL プリコンパイラ・プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB DISABLE BUFFER (実行可能埋込み SQL 拡張要素)」
- C/C++ (Pro*C/C++) : 『Oracle8i Pro*C/C++ プリコンパイラ・プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB DISABLE BUFFER (実行可能埋込み SQL 拡張要素)」

- Visual Basic (Oracle Objects for OLE オンライン・ヘルプ) : 「Help」の「Contents」タブから、「OO4O Automation Server」>「OBJECTS」>「Oralob」>「METHODS」>「DisableBuffering」を選択してください。
- Java (JDBC) : 参照マニュアルはありません。

使用例

次の例は、ここに記述されている方法および関連する方法で開発された Sound についてのバッファリングの管理の一部です。

例

次のプログラム環境の例が示されています。

- PL/SQL (DBMS_LOB パッケージ) : 今回のリリースでは例は記載されていません。
- C (OCI) : [LOB バッファリングの使用禁止化 \(9-246 ページ\)](#)
- COBOL (Pro*COBOL) : [LOB バッファリングの使用禁止化 \(9-248 ページ\)](#)
- C/C++ (Pro*C/C++) : [LOB バッファリングの使用禁止化 \(9-250 ページ\)](#)
- Visual Basic (OO4O) : [LOB バッファリングの使用禁止化 \(9-251 ページ\)](#)
- Java (JDBC) : 今回のリリースでは例は記載されていません。

C (OCI) : LOB バッファリングの使用禁止化

```
/* ロケータ変数にロケータを選択します。*/
sb4 select_lock_sound_locator(Lob_loc, errhp, svchp, stmthp)
OCILobLocator *Lob_loc;
OCIError      *errhp;
OCISvcCtx     *svchp;
OCIStmt       *stmthp;
{
    text *sqlstmt =
        (text *)"SELECT Sound FROM Multimedia_tab WHERE Clip_ID=1 FOR UPDATE";
    OCIDefine *defnp1;
    checkerr (errhp, OCIStmtPrepare(stmthp, errhp, sqlstmt,
                                     (ub4)strlen((char *)sqlstmt),
                                     (ub4) OCI_NTV_SYNTAX, (ub4) OCI_DEFAULT));
    checkerr (errhp, OCIDefineByPos(stmthp, &defnp1, errhp, (ub4) 1,
                                     (dvoid *)&Lob_loc, (sb4) 0,
                                     (ub2) SQLT_BLOB, (dvoid *) 0,
                                     (ub2 *) 0, (ub2 *) 0, (ub4) OCI_DEFAULT));
```

```
/* SELECT を実行し、1 行フェッチします。*/
checkerr(errhp, OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                               (CONST OCISnapshot*) 0, (OCISnapshot*) 0,
                               (ub4) OCI_DEFAULT));

    return 0;
}
#define MAXBUFLLEN 32767
void lobBuffering (envhp, errhp, svchp, stmthp)
OCIEnv      *envhp;
OCIError    *errhp;
OCISvcCtx   *svchp;
OCIStmt     *stmthp;
{
    OCILobLocator *Lob_loc;
    ub4 amt;
    ub4 offset;
    sword retval;
    ub1 bufp[MAXBUFLLEN];
    ub4 buflen;

    /* ロケータ記述子を割り当てます。*/
    (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &Lob_loc,
                              (ub4)OCI_DTYPE_LOB, (size_t) 0, (dvoid **) 0);

    /* BLOB を選択します。*/
    printf (" select a sound Lob\n");
    select_lock_sound_locator(Lob_loc, errhp, svchp, stmthp);

    /* BLOB をオープンします。*/
    checkerr (errhp, (OCILobOpen(svchp, errhp, Lob_loc, OCI_LOB_READWRITE)));

    /* LOB バッファリングを使用可能にします。*/
    printf (" enable LOB buffering\n");
    checkerr (errhp, OCILobEnableBuffering(svchp, errhp, Lob_loc));

    printf (" write data to LOB\n");

    /* LOB にデータを書き込みます。*/
    amt = sizeof (bufp);
    buflen = sizeof (bufp);
    offset = 1;

    checkerr (errhp, OCILobWrite (svchp, errhp, Lob_loc, &amt,
                                  offset, bufp, buflen,
```

```

OCI_ONE_PIECE, (dvoid *)0,
(sb4 (*) (dvoid*,dvoid*,ub4*,ub1 *))0,
0, SQLCS_IMPLICIT));

/* バッファをフラッシュします。*/
printf(" flush the LOB buffers\n");
checkerr (errhp, OCILobFlushBuffer(svchp, errhp, Lob_loc,
                                   (ub4)OCI_LOB_BUFFER_FREE));

/* バッファリングを使用禁止にします。*/
printf (" disable LOB buffering\n");
checkerr (errhp, OCILobDisableBuffering(svchp, errhp, Lob_loc));

/* 後続のLOBの書込みではLOBバッファリング・サブシステムが使用されません。*/

/* BLOBをオープンしている場合、そのBLOBをクローズする必要があります。*/
checkerr (errhp, OCILobClose(svchp, errhp, Lob_loc));

/* ロケータが保持しているリソースを解放します。*/
(void) OCIDescriptorFree((dvoid *) Lob_loc, (ub4) OCI_DTYPE_LOB);

return;
}

```

COBOL (Pro*COBOL) : LOB バッファリングの使用禁止化

```

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.
```

- * BLOB ロケータの割当ておよび初期化を行います。

```
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.
```

- * BLOB をオープンし、バッファリングを使用可能にします。

```
EXEC SQL LOB OPEN :BLOB1 READ WRITE END-EXEC.
```

```
EXEC SQL
```

```
LOB ENABLE BUFFERING :BLOB1
```

```
END-EXEC.
```

- * BLOB にいくつかのデータを書き込みます。

```
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.
```

- * バッファリングされた書き込みをフラッシュします。

```
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++) : LOB バッファリングの使用禁止化

```
#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;
    /* このデータ型では、データ型を等価にする必要があります。*/
    char Buffer[BufferLength];
    EXEC SQL VAR Buffer IS RAW(BufferLength);
    EXEC SQL WHENEVER SQLERROR DO Sample_Error();

    /* LOB の割当ておよび初期化を行います。*/
    EXEC SQL ALLOCATE :Lob_loc;
    EXEC SQL SELECT Sound INTO :Lob_loc
        FROM Multimedia_tab WHERE Clip_ID = 1 FOR UPDATE;
    /* LOB バッファリング・サブシステムを使用可能にします。*/
    EXEC SQL LOB ENABLE BUFFERING :Lob_loc;
    memset((void *)Buffer, 0, BufferLength);
    for (multiple = 0; multiple < 7; multiple++)
    {
        /* LOB にデータを書き込みます。*/
        EXEC SQL LOB WRITE ONE :Amount
            FROM :Buffer INTO :Lob_loc AT :Position;
        Position += BufferLength;
    }
    /* バッファの内容をフラッシュし、これらのリソースを解放します。*/
    EXEC SQL LOB FLUSH BUFFER :Lob_loc FREE;
    /* LOB バッファリング・サブシステムを使用禁止にします。*/
    EXEC SQL LOB DISABLE BUFFERING :Lob_loc;
    /* バッファリングが使用禁止の場合のみ、WRITE APPEND 操作を行うことができます。*/
    EXEC SQL LOB WRITE APPEND ONE :Amount FROM :Buffer INTO :Lob_loc;
```

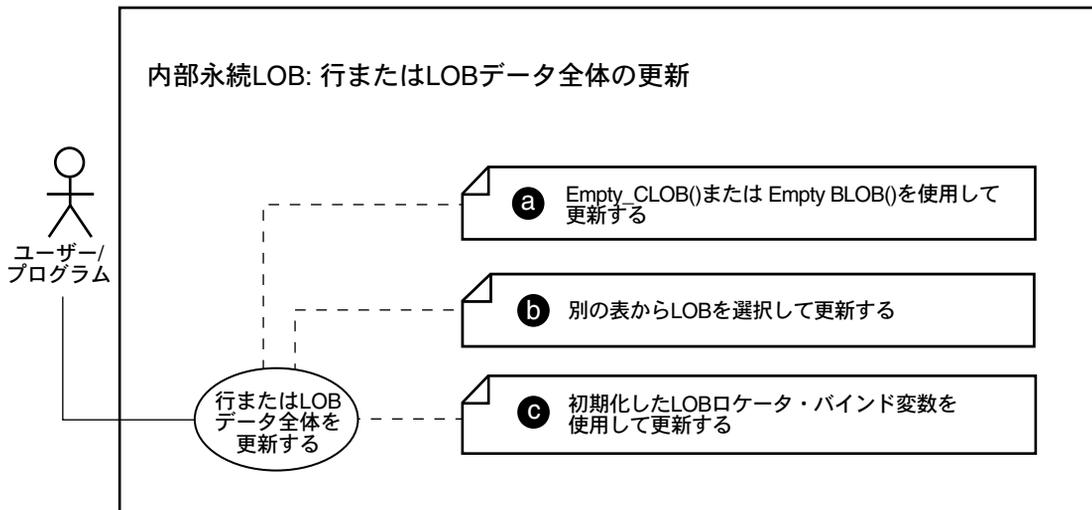
```
/* ロケータが保持しているリソースを解放します。*/  
EXEC SQL FREE :Lob_loc;  
}  
  
void main()  
{  
    char *samp = "samp/samp";  
    EXEC SQL CONNECT :samp;  
    disableBufferingLOB_proc();  
    EXEC SQL ROLLBACK WORK RELEASE;  
}
```

Visual Basic (0040) : LOB バッファリングの使用禁止化

```
Dim MySession As OraSession  
Dim OraDb As OraDatabase  
Dim OraDyn As OraDynaset, OraSound1 As OraBlob, OraSoundClone As OraBlob  
  
Set MySession = CreateObject("OracleInProcServer.XOraSession")  
Set OraDb = MySession.OpenDatabase("exampledb", "samp/samp", 0&)  
Set OraDyn = OraDb.CreateDynaset(  
    "SELECT * FROM Multimedia_tab ORDER BY clip_id", ORADYN_DEFAULT)  
  
Set OraSound1 = OraDyn.Fields("Sound").Value  
' バッファリングを使用禁止にします。  
OraSound1.DisableBuffering
```

LOB または LOB データ全体を更新する 3 つの方法

図 9-41 ユースケース図：LOB または LOB データ全体を更新する 3 つの方法



参照： 内部永続 LOB に関するすべての基本操作については、9-2 ページの「ユースケース・モデル:内部永続 LOB – 基本操作」を参照してください。

- a. [EMPTY_CLOB\(\) または EMPTY_BLOB\(\) を使用した LOB の更新](#) (9-255 ページ)
- b. [別の表からの LOB の選択による行の更新](#) (9-258 ページ)
- c. [初期化した LOB ロケータ・バインド変数を使用した更新](#) (9-260 ページ)

4,000 バイトを超えるバインドの場合

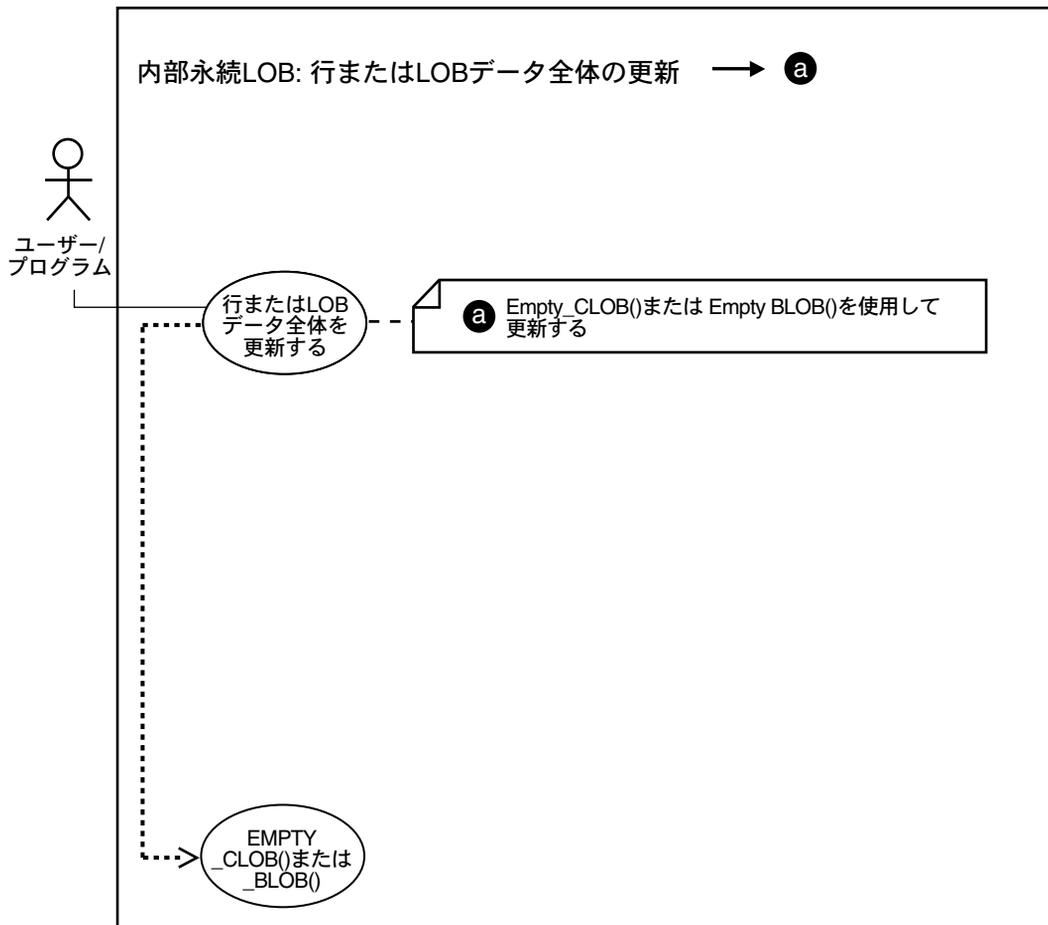
4,000 バイトを超えるバインドが伴う場合の LOB の更新方法については、第 7 章「モデリングおよび設計」の次の項を参照してください。

- 7-14 ページの「LOB の INSERT および UPDATE で現在可能な 4,000 バイトを超えるバインド」
- 7-14 ページの「4,000 バイトを超えるバインド (HEX から RAW または RAW から HEX への変換なし)」

- 7-16 ページの「例: PL/SQL - INSERT および UPDATE での 4,000 バイトを超えるバインドの使用」
- 7-17 ページの「例: PL/SQL - 4,000 バイトを超えるバインド (HEX から RAW / RAW から HEX への変換がサポートされていないため挿入は未サポート)」
- 7-18 ページの「例: PL/SQL - データに SQL 演算子が含まれる場合に 4,000 バイトを超えるバインドの結果を 4,000 バイトに制限」

EMPTY_CLOB() または EMPTY_BLOB() を使用した LOB の更新

図 9-42 ユースケース図 : EMPTY_CLOB() または EMPTY_BLOB() を使用した LOB の更新



参照: 内部永続 LOB に関するすべての基本操作については、9-2 ページの「ユースケース・モデル:内部永続 LOB – 基本操作」を参照してください。

用途

EMPTY_CLOB() または EMPTY_BLOB() を使用して、LOB を更新します。

使用上の注意

NULL 以外の LOB 列の作成

データを内部 LOB に書き込む前に、LOB 列を NULL 以外にしておきます。つまり、LOB 列には、空または移入された LOB 値を示すロケータを含める必要があります。EMPTY_BLOB() をデフォルトの述語に使用することによって、BLOB 列の値を初期化できます。同様に、EMPTY_CLOB() 関数を使用して、CLOB 列または NCLOB 列の値を初期化できます。

サイズが 4,000 バイト未満の文字列または RAW 文字列を含む LOB 列も初期化できます。次に例を示します。

```
UPDATE Multimedia_tab
   SET story = 'This is a One Line Story'
   WHERE clip_id = 2;
```

この初期化は CREATE TABLE でも実行可能です（「[1 つ以上の LOB 列を含む表の作成](#)」を参照）。また、この場合は INSERT を使用しても実行できます。

構文

次のマニュアルの項を参照してください。

- SQL: 『Oracle8i SQL リファレンス』の第7章「SQL 文」の「UPDATE」

使用例

次の例では、最初のクリップの様々なデータ型に対し、EMPTY_CLOB 操作で更新を行う一連の手続きを示します。

例

例が SQL で示されています。この例は、すべてのプログラム環境に適用されます。

- SQL: [EMPTY_CLOB\(\) または EMPTY_BLOB\(\) を使用した LOB の更新](#)

SQL: EMPTY_CLOB() または EMPTY_BLOB() を使用した LOB の更新

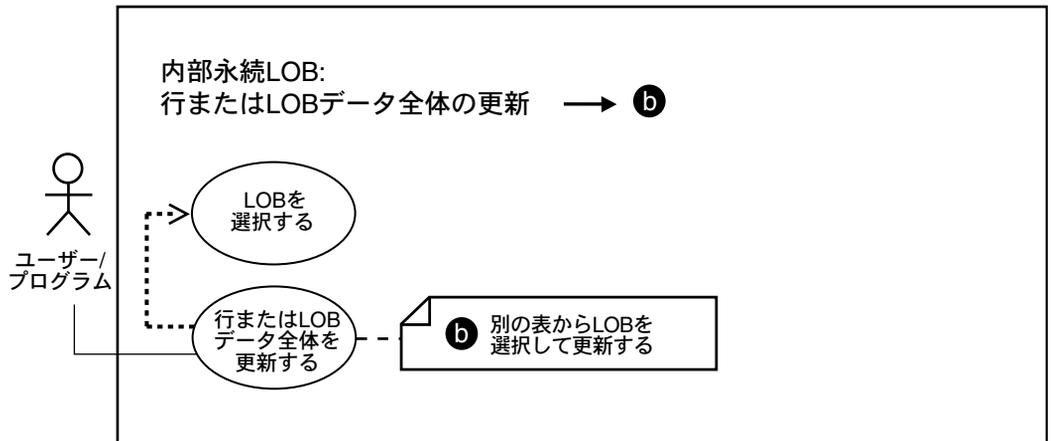
```
UPDATE Multimedia_tab SET Story = EMPTY_CLOB() WHERE Clip_ID = 1;
```

```
UPDATE Multimedia_tab SET FLSub = EMPTY_CLOB() WHERE Clip_ID = 1;
```

```
UPDATE multimedia_tab SET Sound = EMPTY_BLOB() WHERE Clip_ID = 1;
```

別の表からの LOB の選択による行の更新

図 9-43 ユースケース図：別の表からの LOB の選択による行の更新



参照： 内部永続 LOB に関するすべての基本操作については、9-2 ページの「ユースケース・モデル:内部永続 LOB – 基本操作」を参照してください。

用途

別の表から LOB を選択して LOB を更新します。

使用上の注意

ありません。

構文

次のマニュアルの項を参照してください。

- SQL: 『Oracle8i SQL リファレンス』の第7章「SQL文」の「UPDATE」

使用例

次の例では、参照を使用してアーカイブ記憶域（VoiceoverLib_tab）からのデータで台本を更新します。

例

例が SQL で示されています。この例は、すべてのプログラム環境に適用されます。

- [SQL: 別の表からの LOB の選択による行の更新](#)

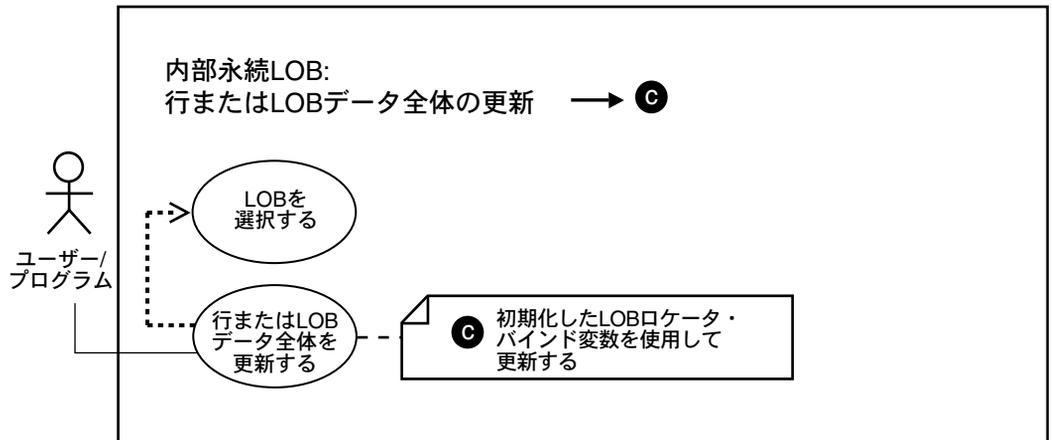
SQL: 別の表からの LOB の選択による行の更新

```
UPDATE Voiceover_tab SET (Originator, Script, Actor, Take, Recording) =  
    (SELECT * FROM VoiceoverLib_tab T2 WHERE T2.Take = 101);
```

```
UPDATE Multimedia_tab Mtab  
SET Voiced_ref =  
    (SELECT REF(Vref) FROM Voiceover_tab Vref  
    WHERE Vref.Actor = 'James Earl Jones' AND Vref.Take = 1)  
    WHERE Mtab.Clip_ID = 1;
```

初期化した LOB ロケータ・バインド変数を使用した更新

図 9-44 ユースケース図：初期化した LOB ロケータ・バインド変数を使用した更新



参照： 内部永続 LOB に関するすべての基本操作については、9-2 ページの「ユースケース・モデル：内部永続 LOB – 基本操作」を参照してください。

用途

初期化した LOB ロケータ・バインド変数を使用して更新します。

使用上の注意

ありません。

構文

各プログラム環境で使用可能な機能のリストは、第 3 章「LOB プログラム環境」を参照してください。各プログラム環境における構文については、次のマニュアルの項を参照してください。

- SQL: 『Oracle8i SQL リファレンス』の第 7 章「SQL 文」の「UPDATE」
- C (OCI) : 『Oracle8i コール・インタフェース・プログラマーズ・ガイド』の第 15 章「OCI リレーショナル関数」の「LOB 関数」

- COBOL (Pro*COBOL) : 『Oracle8i Pro*COBOL プリコンパイラ・プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「ALLOCATE (実行可能埋込み SQL 拡張要素)」
- C/C++ (Pro*C/C++) : 『Oracle8i Pro*C/C++ プリコンパイラ・プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびプリコンパイラ宣言文」
- Visual Basic (Oracle Objects for OLE オンライン・ヘルプ) : 「Help」の「Contents」タブから、「OO4O Automation Server」>「OBJECTS」>「Oradatabase」>「METHODS」>「ExecuteSQL」を選択してください。
- Java (JDBC) : 『Oracle8i JDBC 開発者ガイドおよびリファレンス』の第7章「LOB と BFILE の操作」の「BLOB と CLOB の操作」の「BLOB または CLOB 列の作成と移入」、および『Oracle8i SQLJ 開発者ガイドおよびリファレンス』の第5章「型のサポート」の「Oracle の拡張型」の「BLOB、CLOB および BFILE のサポート」

使用例

次の例では、ロケータ・バインド変数を使用して sound データを更新します。

例

次のプログラム環境の例が示されています。

- [SQL: 初期化した LOB ロケータ・バインド変数を使用した更新 \(9-260 ページ\)](#)
- [C \(OCI\) : 初期化した LOB ロケータ・バインド変数を使用した更新 \(9-261 ページ\)](#)
- [COBOL \(Pro*COBOL\) : 初期化した LOB ロケータ・バインド変数を使用した更新 \(9-262 ページ\)](#)
- [C/C++ \(Pro*C/C++\) : 初期化した LOB ロケータ・バインド変数を使用した更新 \(9-264 ページ\)](#)
- [Visual Basic \(OO4O\) : 初期化した LOB ロケータ・バインド変数を使用した更新 \(9-265 ページ\)](#)
- [Java \(JDBC\) : 初期化した LOB ロケータ・バインド変数を使用した更新 \(9-265 ページ\)](#)

SQL: 初期化した LOB ロケータ・バインド変数を使用した更新

/* 例のプロシージャ updateUseBindVariable_proc は、DEMS_LOB パッケージの一部ではないことに注意してください。*/

```
CREATE OR REPLACE PROCEDURE updateUseBindVariable_proc (lob_loc BLOB) IS
BEGIN
    UPDATE Multimedia_tab SET Sound = lob_loc WHERE Clip_ID = 2;
END;

DECLARE
```

```

Lob_loc      BLOB;
BEGIN
  /* LOB を選択します。*/
  SELECT Sound INTO Lob_loc
     FROM Multimedia_tab
     WHERE Clip_ID = 1;
  updateUseBindVariable_proc (Lob_loc);
  COMMIT;
END;

```

C (OCI) : 初期化した LOB ロケータ・バインド変数を使用した更新

```

/* ロケータ変数にロケータを選択します。*/
sb4 select_sound_locator(Lob_loc, errhp, svchp, stmthp)
OCILobLocator *Lob_loc;
OCIError      *errhp;
OCISvcCtx     *svchp;
OCISstmt      *stmthp;
{
  text *sqlstmt =
    (text *) "SELECT Sound FROM Multimedia_tab WHERE Clip_ID=2";
  OCIDefine *defnp1;
  checkerr (errhp, OCISstmtPrepare(stmthp, errhp, sqlstmt,
    (ub4) strlen((char *) sqlstmt),
    (ub4) OCI_NTV_SYNTAX, (ub4) OCI_DEFAULT));
  checkerr (errhp, OCIDefineByPos(stmthp, &defnp1, errhp, (ub4) 1,
    (dvoid *)&Lob_loc, (sb4) 0,
    (ub2) SQLT_BLOB, (dvoid *) 0,
    (ub2 *) 0, (ub2 *) 0, (ub4) OCI_DEFAULT));
  /* SELECT を実行し、1 行フェッチします。*/
  checkerr(errhp, OCISstmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
    (CONST OCISnapshot*) 0, (OCISnapshot*) 0,
    (ub4) OCI_DEFAULT));

  return 0;
}

/* 表の選択された行にある LOB を更新します。*/
void updateLobUsingBind (envhp, errhp, svchp, stmthp)
OCIEnv      *envhp;
OCIError    *errhp;
OCISvcCtx   *svchp;
OCISstmt    *stmthp;
{
  text *updstmt =
    (text *) "UPDATE Multimedia_tab SET Sound = :1 WHERE Clip_ID = 1";
  OCILobLocator *Lob_loc;

```

```
OCIBind      *bndhp1;

/* ロケータ・リソースを割り当てます。*/
(void) OCIDescriptorAlloc((dvoid *)envhp, (dvoid **)&Lob_loc,
                          (ub4)OCI_DTYPE_LOB, (size_t)0, (dvoid **)0);

/* ロケータを選択します。*/
printf(" select a sound locator\n");
(void)select_sound_locator(Lob_loc, errhp, svchp, stmthp);

/* SQL 文を準備します。*/
checkerr (errhp, OCISTmtPrepare(stmthp, errhp, updstmt, (ub4)
                               strlen((char *) updstmt),
                               (ub4) OCI_NTV_SYNTAX, (ub4)OCI_DEFAULT));

/* バインド位置をバインドします。*/
printf(" bind locator to bind position\n");

checkerr (errhp, OCIBindByPos(stmthp, &bndhp1, errhp, (ub4) 1,
                              (dvoid *) &Lob_loc, (sb4)0, SQLT_BLOB,
                              (dvoid *) 0, (ub2 *)0, (ub2 *)0,
                              (ub4) 0, (ub4 *) 0, (ub4) OCI_DEFAULT));

/* SQL 文を実行します。*/
printf ("update LOB column in another row using this locator\n");
checkerr (errhp, OCISTmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                               (CONST OCISnapshot*) 0, (OCISnapshot*) 0,
                               (ub4) OCI_DEFAULT));

return;
}
```

COBOL (Pro*COBOL) : 初期化した LOB ロケータ・バインド変数を使用した更新

```
IDENTIFICATION DIVISION.
PROGRAM-ID. UPDATE-BLOB.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

01 BLOB1          SQL-BLOB.
01 NEW-LEN       PIC S9(9) COMP.
01 AMT           PIC S9(9) COMP.
01 OFFSET       PIC S9(9) COMP.
```

```
* ソースおよび宛先の位置を定義します。
01 SRC-POS          PIC S9(9) COMP.
01 DEST-POS         PIC S9(9) COMP.
01 SRC-LOC          PIC S9(9) COMP.
01 DEST-LOC         PIC S9(9) COMP.
01 USERID          PIC X(11) VALUES "SAMP/SAMP".
                   EXEC SQL INCLUDE SQLCA END-EXEC.

PROCEDURE DIVISION.
UPDATE-BLOB.

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

* BLOB ロケータの割当ておよび初期化を行います。
                   EXEC SQL ALLOCATE :BLOB1 END-EXEC.
                   EXEC SQL WHENEVER NOT FOUND GOTO END-OF-BLOB END-EXEC.
                   EXEC SQL
                           SELECT SOUND INTO :BLOB1
                           FROM MULTIMEDIA_TAB
                           WHERE CLIP_ID = 1
                   END-EXEC.

                   EXEC SQL
                           UPDATE MULTIMEDIA_TAB
                           SET SOUND = :BLOB1 WHERE CLIP_ID = 2
                   END-EXEC.

END-OF-BLOB.
                   EXEC SQL WHENEVER NOT FOUND CONTINUE END-EXEC.
                   EXEC SQL FREE :BLOB1 END-EXEC.
                   EXEC SQL 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++) : 初期化した LOB ロケータ・バインド変数を使用した更新

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

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

void updateUseBindVariable_proc(Lob_loc)
    OCIBlobLocator *Lob_loc;
{
    EXEC SQL WHENEVER SQLERROR DO Sample_Error();
    EXEC SQL UPDATE Multimedia_tab SET Sound = :Lob_loc WHERE Clip_ID = 2;
}

void updateLOB_proc()
{
    OCIBlobLocator *Lob_loc;

    EXEC SQL ALLOCATE :Lob_loc;
    EXEC SQL SELECT Sound INTO :Lob_loc
        FROM Multimedia_tab WHERE Clip_ID = 1;
    updateUseBindVariable_proc(Lob_loc);
    EXEC SQL FREE :Lob_loc;
    EXEC SQL COMMIT WORK;
}

void main()
{
    char *samp = "samp/samp";
    EXEC SQL CONNECT :samp;
    updateLOB_proc();
    EXEC SQL ROLLBACK WORK RELEASE;
}
```

Visual Basic (0040) : 初期化した LOB ロケータ・バインド変数を使用した更新

```
Dim OraDyn As OraDynaset, OraSound as OraBlob

'clip_id = 1 の列を選択します。
Set OraDyn = OraDb.CreateDynaset("SELECT * FROM Multimedia_tab WHERE
    clip_id = 1", ORADYN_DEFAULT)

'フィールドから OraBlob オブジェクトを取得します。
Set OraSound = OraDyn.Fields("Sound").Value

'OraBlob オブジェクトに対するパラメータを作成します。
OraDb.Parameters.Add "SOUND", Null, ORAPARM_INPUT, ORATYPE_BLOB

'SOUND パラメータの値を OraSound に設定します。
OraDb.Parameters("SOUND").Value = OraSound

'clip_id = 2 の OraSound で、Multimedia_tab を更新します。
OraDb.ExecuteNonQuery("Update Multimedia_tab SET Sound = :SOUND
    WHERE Clip_id = 2")
```

Java (JDBC) : 初期化した LOB ロケータ・バインド変数を使用した更新

```
import java.sql.Connection;
import java.sql.Statement;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

// Oracle 固有の JDBC クラス :
import oracle.sql.*;
import oracle.jdbc.driver.*;

public class Ex2_163
{
    public static void main (String args [])
        throws Exception
    {
        // Oracle JDBC Driver をロードします。
        DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

        // データベースに接続します。
        Connection conn =
            DriverManager.getConnection ("jdbc:oracle:oci8:@", "samp", "samp");
```

```
// 自動コミットがオフの場合、高速になります。
conn.setAutoCommit (false);

// 文を作成します。
Statement stmt = conn.createStatement ();
try
{
    ResultSet rset = stmt.executeQuery (
        "SELECT sound FROM multimedia_tab WHERE clip_id = 1");
    if (rset.next ())
    {
        // ResultSet から LOB ロケータを取り出します。
        BLOB sound_blob = ((OracleResultSet)rset).getBLOB (1);

        OraclePreparedStatement ops =
            (OraclePreparedStatement) conn.prepareStatement(
                "UPDATE multimedia_tab SET SOUND = ? WHERE clip_id = 2");
        ops.setBlob(1, sound_blob);
        ops.execute();
        rset.close();
        stmt.close();
    }
    conn.commit();
    conn.close();
}
catch (SQLException e)
{
    e.printStackTrace();
}
}
```

LOB を含む表の行の削除

図 9-45 ユースケース図：LOB を含む表の行の削除



参照： 内部永続 LOB に関するすべての基本操作については、9-2 ページの「ユースケース・モデル:内部永続 LOB – 基本操作」を参照してください。

用途

LOB を含む表の行を削除します。

使用上の注意

次のコマンドの 1 つを使用して、内部 LOB 列または属性を含む行を削除します。

- SQL DML: DELETE
- 効率的に削除する SQL DDL: DROP TABLE、TRUNCATE TABLE、DROP TABLESPACE

いずれの場合も、LOB ロケータのみでなく LOB 値も削除することになります。

注意： ただし、読取り一貫性の機能によって、LOB ロケータを戻した文 (SELECT など) の実行時の古い LOB 値にはアクセス可能です。これは、高度なトピックです。詳細は、5-2 ページの「[読取り一貫性のあるロケータ](#)」を参照してください。

個別の行に対する個別の LOB ロケータ

LOB 列を含む表の 2 つの個別の列には、LOB 値が同じであるかないかにかかわらず、それぞれ個別の LOB ロケータおよび個別の LOB 値のコピーがあります。行の削除は、LOB が別の行からコピーされたものであっても、コピー元の行データまたは LOB ロケータに影響しません。

構文

次のマニュアルの項を参照してください。

- SQL: 『Oracle8i SQL リファレンス』の第7章「SQL 文」の「DELETE」、「DROP TABLE」、「TRUNCATE」

使用例

次の例では、クリップ 10 に対応付けられているすべてのデータを削除します。

例

例が SQL で示されています。この例は、すべてのプログラム環境に適用されます。

- [SQL: LOB の削除](#)

SQL: LOB の削除

```
DELETE FROM Multimedia_tab WHERE Clip_ID = 10;
```

```
DROP TABLE Multimedia_tab;
```

```
TRUNCATE TABLE Multimedia_tab;
```

10

テンポラリ LOB

ユースケース・モデル

この章では、テンポラリ LOB に対する操作（テンポラリ LOB がオープンしているかどうかの確認など）を、ユースケースの点から説明します。表 10-1「ユースケース・モデル概要：内部テンポラリ LOB」に、すべてのユースケースを示します。

ユースケース・モデルの図形概要

「ユースケース・モデル図：内部テンポラリ LOB (1/2)」および「ユースケース・モデル図：内部テンポラリ LOB (2/2)」に、ユースケースおよびその相互関係を示します。

個々のユースケース

内部永続 LOB の各ユースケースは、次のように説明されています。

- **ユースケース図**：ユースケースを表す図（図の見方については、「はじめに」の「図の解釈方法」を参照）
- **用途**：LOB に関するこのユースケースの用途
- **使用上の注意**：LOB 操作の実装に有効なガイドライン
- **構文**：様々なプログラム環境における、ユースケースに対する LOB 関連アクティビティの基礎となる構文
- **使用例**：ユースケース実装の例を仮想マルチメディア・アプリケーションの点から表現した使用例。第 8 章「サンプル・アプリケーション」を参照してください。
- **例**：ユースケースの実装に使用できる各プログラム環境での例。これらは、第 8 章「サンプル・アプリケーション」で説明する `Multimedia_tab` 表を基にしています。

ユースケース・モデル：内部テンポラリ LOB

表 10-1 「ユースケース・モデル概要：内部テンポラリ LOB」の「+」は、特定のユースケースで、プログラム環境の例が提供されているものを示します（詳細および関連マニュアルは、第 3 章「LOB プログラム環境」を参照）。

この表では、プログラム環境を次の略称で表しています。

- P – DBMS_LOB パッケージを使用した PL/SQL
- O – OCI (Oracle Call Interface) を使用した C
- B – Pro*COBOL プリコンパイラを使用した COBOL
- C – Pro*C/C++ プリコンパイラを使用した C/C++
- V – OO4O (Oracle Objects for OLE) を使用した Visual Basic
- J – JDBC (Java Database Connectivity) を使用した Java
- S – SQL

表 10-1 ユースケース・モデル概要：内部テンポラリ LOB

ユースケース	プログラム環境の例					
	P	O	B	C	V	J
10-14 ページの「テンポラリ LOB の作成」	+	+	+	+		
10-22 ページの「LOB がテンポラリ LOB であるかどうかの確認」	+	+	+	+		
10-28 ページの「テンポラリ LOB の解放」	+	+	+	+		
10-33 ページの「テンポラリ LOB への BFILE データのロード」	+	+	+	+		
10-40 ページの「テンポラリ LOB がオープンしているかどうかの確認」	+	+	+	+		
10-46 ページの「テンポラリ LOB データの表示」	+	+	+	+		
10-56 ページの「テンポラリ LOB からのデータの読込み」	+	+	+	+		
10-67 ページの「テンポラリ LOB の一部の読込み (substr)」	+		+	+		
10-74 ページの「2 つの (テンポラリ) LOB の全体または一部の比較」	+		+	+		
10-81 ページの「テンポラリ LOB 内のパターンの有無の確認 (instr)」	+		+	+		
10-88 ページの「テンポラリ LOB の長さの取得」	+	+	+	+		
10-97 ページの「(テンポラリ) LOB の全体または一部の他へのコピー」	+	+	+	+		
10-107 ページの「テンポラリ LOB の LOB ロケータのコピー」	+	+	+	+		

ユースケース	プログラム環境の例						
	P	O	B	C	V	J	
10-116 ページの「テンポラリ LOB ロケータが他と等しいかどうかの確認」		+		+			
10-121 ページの「テンポラリ LOB の LOB ロケータが初期化されているかどうかの確認」		+		+			
10-125 ページの「テンポラリ LOB のキャラクタ・セット ID の取得」		+					
10-128 ページの「テンポラリ LOB のキャラクタ・セット・フォームの取得」		+					
10-131 ページの「(テンポラリ) LOB の他への追加」	+	+	+	+			
10-140 ページの「テンポラリ LOB への追加での書込み」	+	+	+	+			
10-148 ページの「テンポラリ LOB へのデータの書込み」	+	+	+	+			
10-158 ページの「テンポラリ LOB データの切捨て」	+	+	+	+			
10-166 ページの「テンポラリ LOB の一部の消去」	+	+	+	+			
10-175 ページの「テンポラリ LOB に対する LOB バッファリングの使用可能化」		+	+	+			
10-182 ページの「テンポラリ LOB に対するバッファのフラッシュ」		+	+	+			
10-188 ページの「テンポラリ LOB に対する LOB バッファリングの使用禁止化」		+	+	+			

図 10-1 ユースケース・モデル図：内部テンポラリ LOB (1/2)

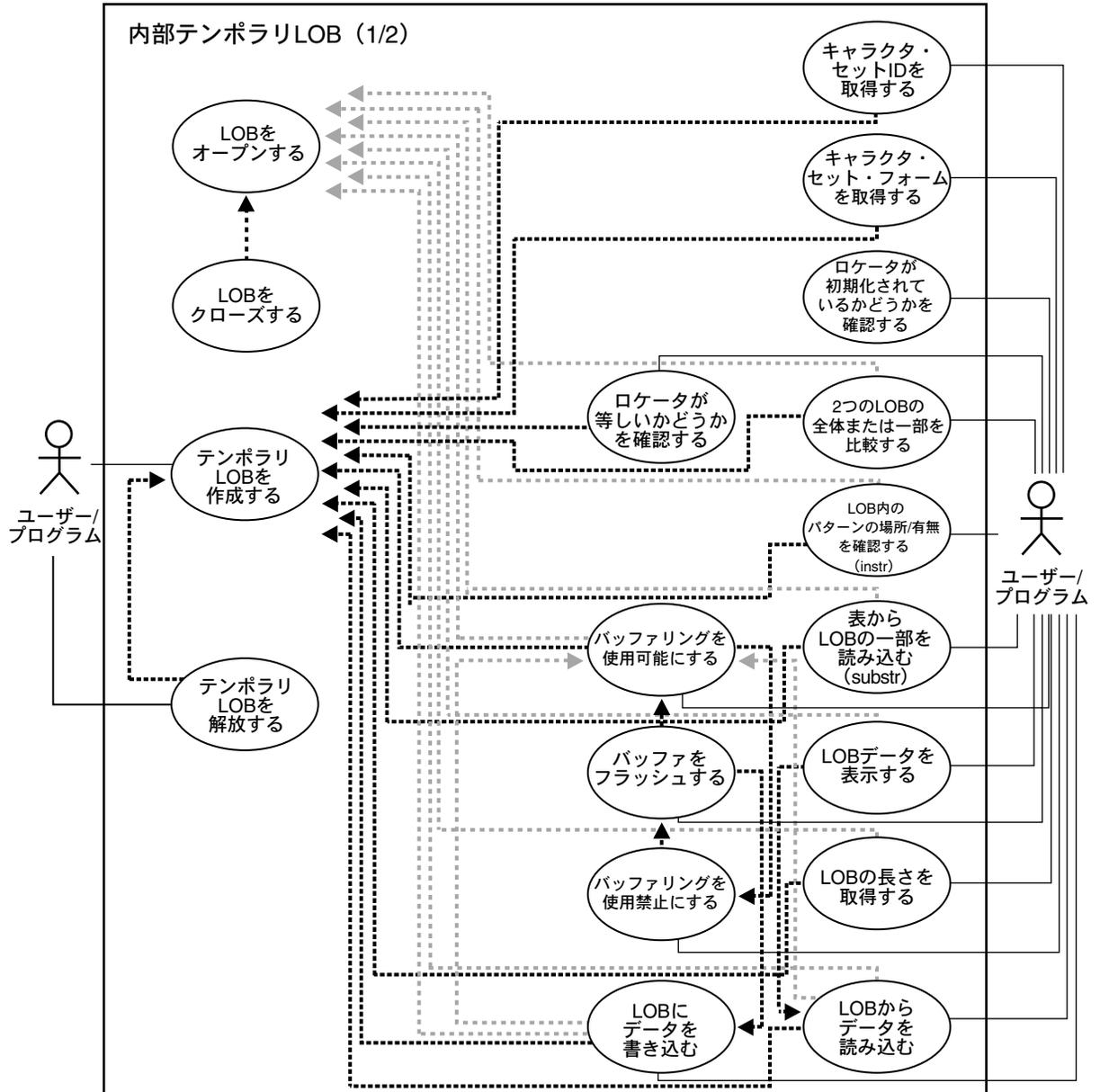
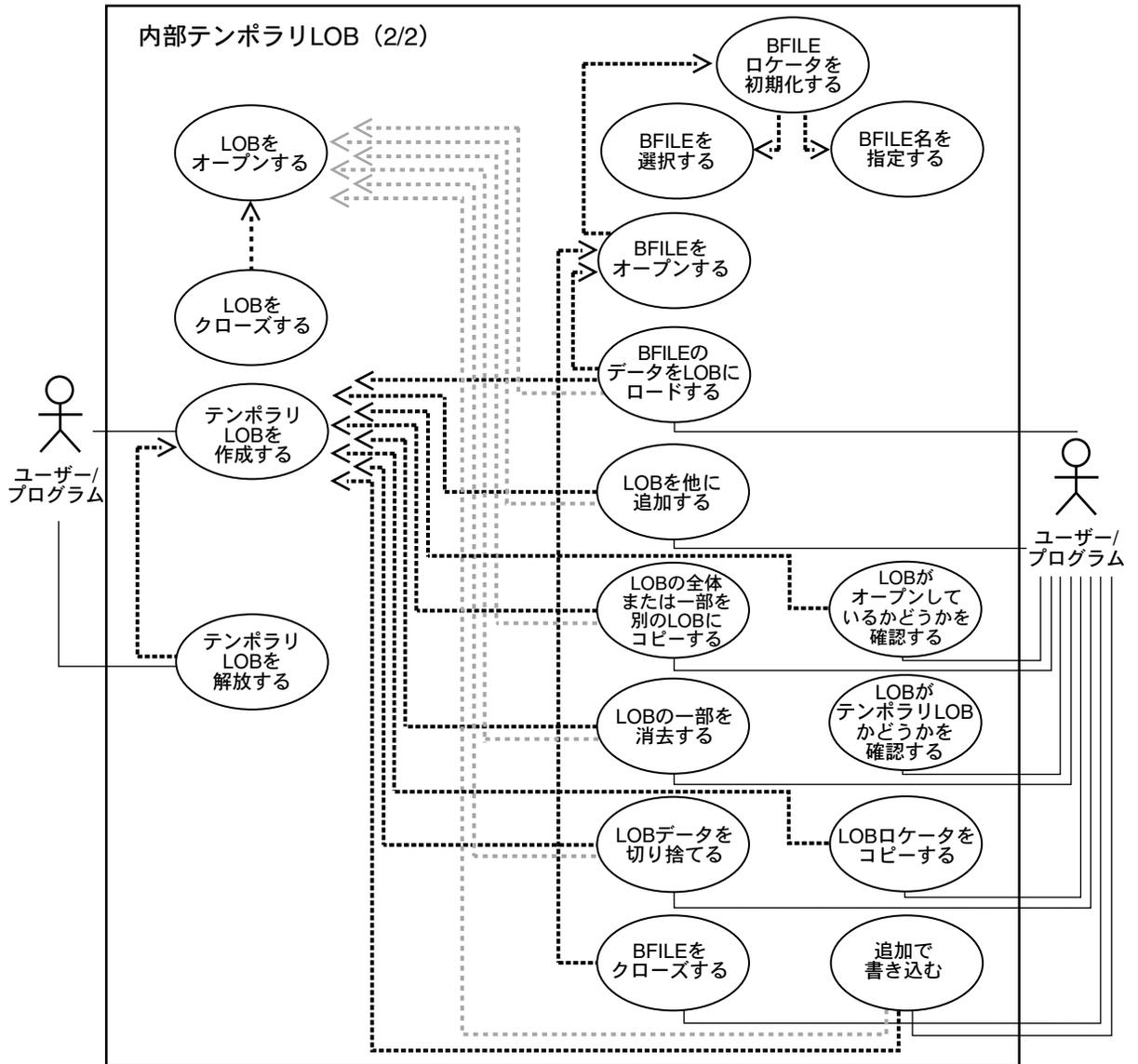


図 10-2 ユースケース・モデル図：内部テンポラリLOB (2/2)



プログラム環境

注意： リリース 8.1 では、テンポラリ LOB に対する Visual Basic または Java はサポートされていません。

Oracle8i では、次のプログラム環境またはインタフェースでのテンポラリ LOB の定義、作成、削除、アクセスおよび更新がサポートされます。

- DBMS_LOB パッケージを使用した PL/SQL
- Pro*C プリコンパイラを使用した C/C++
- Pro*COBOL プリコンパイラを使用した COBOL
- OCI を使用した C

ロケータ

前述のインタフェースは、テンポラリ LOB に対しても、永続 LOB に対する場合と同じようにロケータを介して操作できます。テンポラリ LOB はどの表にも含まれないため、テンポラリ LOB に対して SQL DML を操作することはできません。DBMS_LOB パッケージ、OCI またはその他のプログラム・インタフェースを使用して操作する必要があります。

IN 値として使用できるテンポラリ LOB ロケータ

テンポラリ LOB に対する SQL は、IN 値として使用できるテンポラリ LOB のロケータで、ロケータを介してアクセスされる値を使用することによってサポートされます。具体的には、次のように使用できます。

- INSERT、UPDATE、DELETE または SELECT の WHERE 句の値

次に例を示します。

```
SELECT pattern FROM composite_image WHERE temp_lob_pattern_id =  
somepattern_match_function(lobvalue);
```

- SELECT INTO... 文の変数

次に例を示します。

```
SELECT PermanentLob INTO TemporaryLob_loc FROM Demo_tab WHERE Column1 := 1;
```

注意： テンポラリ LOB を指す LOB ロケータ内に永続 LOB を選択すると、ロケータが永続 LOB を指すようになります。永続 LOB をコピーしてテンポラリ LOB に置くことはありません。

テンポラリ LOB および内部永続 LOB に使用できる関数

テンポラリ LOB のユースケース・モデル図を図 10-1「ユースケース・モデル図：内部テンポラリ LOB (1/2)」および図 10-2「ユースケース・モデル図：内部テンポラリ LOB (2/2)」と比較してください。次の関数が内部永続 LOB およびテンポラリ LOB に使用できることがわかります。

- DBMS_LOB パッケージ PL/SQL プロシージャ (COMPARE、INSTR、SUBSTR)
- DBMS_LOB パッケージ PL/SQL プロシージャおよび対応する OCI 関数 (Append、Copy、Erase、Getlength、Loadfromfile、Read、Trim、Write、WriteAppend)。
- OCI 関数 (OCILobAssign、OCILobLocatorIsInit など)。

さらに、ISTEMPORARY 関数を使用して、LOB のロケータが一時的であるかどうかを判断できます。

注意： テンポラリ LOB では、トランザクションおよび読取一貫性をサポートしていません。

一時表領域へのテンポラリ LOB データの格納

テンポラリ LOB は、他のデータのようにデータベース内に永続的に格納されるわけではありません。このデータは一時表領域には格納されますが、どの表にも格納されません。これは、どの表にも属さない内部テンポラリ LOB (BLOB、CLOB、NCLOB) をサーバーに作成できますが、その LOB を格納できないことを意味します。

テンポラリ LOB は表スキーマに対応付けられていないため、テンポラリ LOB については「行内」および「行外」という言葉は意味を持ちません。

注意： すべてのテンポラリ LOB はサーバーに格納されます。クライアント側ではテンポラリ LOB はサポートされていません。

テンポラリ LOB の存続期間

テンポラリ LOB のデフォルト存続期間は、1 つのセッションとなります。

テンポラリ LOB の作成インタフェースに、テンポラリ LOB の存続期間のデフォルト有効範囲を指定できるパラメータがあります。デフォルトでは、すべてのテンポラリ LOB は、LOB が作成されたセッションの終了で削除されます。プロセスが予期せずに終了した場合またはデータベースがクラッシュした場合は、すべてのテンポラリ LOB が削除されます。

OCI による論理バケットへのテンポラリ LOB のグループ化

OCI ユーザーは、テンポラリ LOB を論理バケットにグループ化できます。

OCIDuration は、テンポラリ LOB に対する格納期間を表します。ユーザーが特定の存続期間を指定しない場合は、テンポラリ LOB が置かれる各セッションごとのデフォルトの存続期間が設定されます。デフォルトの存続期間は、ユーザーのセッションが終了するときに終わります。また、OCIDurationEnd 操作を実行して、OCIDuration 内のすべての内容を解放することもできます。

メモリー操作

LOB バッファリングおよび CACHE、NOCACHE、CACHE READS

イメージのモーフィングや、LOB データの形式を別の形式に変更するなど、LOB に対して変換操作を行う場合、さらにその後これをデータベースに戻す場合に、テンポラリ LOB は特に有効です。

これらの変換操作には、LOB バッファリングを使用できます。各テンポラリ LOB に対して CACHE、NOCACHE または CACHE READS を指定でき、さらに必要なくなったときにテンポラリ LOB を個別に解放することができます。

一時表領域

一時表領域を使用して、テンポラリ LOB データが格納されます。データ記憶域リソースは、ユーザーの一時表領域アクセスの制御を介して DBA によって制御されますが、別の一時表領域の作成によっても制御されます。

テンポラリ LOB 領域の明示的な解放による再使用

テンポラリ LOB の数の増加に伴い、メモリー使用量は徐々に増加します。テンポラリ LOB を明示的に解放することによって、セッション内のテンポラリ LOB 領域を再使用できます。

- セッションが完了するとき: 1 つ以上のテンポラリ LOB を明示的に解放しても、その空間が再び通常どおりに割り当てられるように一時表領域に返されるわけではありません。その空間は、そのセッション内で再使用可能な状態で保たれます。
- セッションが終了するとき: プロセスが予期せずに終了した場合またはデータベースがクラッシュした場合は、テンポラリ LOB が削除されるとともに、テンポラリ LOB 領域が解放されます。どのような場合でも、ユーザーのセッションが終了したときに、領域は一般的な再使用のために一時表領域に戻されます。

テンポラリ LOB ロケータ内への永続 LOB の選択

前述のように、次の文を実行すると、

```
SELECT permanent_lob INTO temporary_lob_locator FROM y_blah WHERE x_blah
```

temporary_lob_locator は、permanent_lob のロケータによって上書きされます。temporary_lob_locator は、表に格納された LOB を指すようになります。

注意: temporary_lob のロケータを他の変数に保存しない限り、SELECT INTO 操作を実行する前に temporary_lob_locator が最初に指していた LOB を追跡できなくなります。

この場合、テンポラリ LOB は暗黙的に解放されなくなります。領域を無駄にしたくない場合は、永続 LOB ロケータで上書きする前にテンポラリ LOB を明示的に解放します。

CR およびロールバックはテンポラリ LOB でサポートされていないため、エラーが発生した場合は、テンポラリ LOB を解放してから再開する必要があります。

ロケータおよびセマンティクス

ユーザーがテンポラリ LOB インスタンスを作成すると、エンジンが LOB データへのロケータを作成して戻します。テンポラリ LOB は、永続 LOB ロケータでサポートされていない操作はどれもサポートしませんが、テンポラリ LOB ロケータには固有の機能があります。

テンポラリ LOB 固有の機能

次の機能は、テンポラリ LOB に固有のもので、

- **永続 LOB ロケータによるテンポラリ LOB ロケータの上書き**

たとえば、次の問合せを実行すると、

```
SELECT permanent_lob INTO temporary_lob_locator FROM y_blah
WHERE x_blah = a_number;
```

temporary_lob_locator は、permanent_lob のロケータによって上書きされます。これは、上書きされたテンポラリ LOB を指す temporary_lob のロケータのコピーを保持しない限り、このテンポラリ LOB にアクセスするためのロケータを失うことを意味します。

- **複数ロケータを同じテンポラリ LOB に割り当てる場合のパフォーマンスへの影響**

永続 LOB との一貫性を保つため、また LOB の ANSI 規格に準拠するために、テンポラリ LOB は値セマンティクスを遵守します。CR、UNDO およびバージョンはテンポラリ LOB に対して生成されないため、複数のロケータを同じテンポラリ LOB に割り当てた場合にパフォーマンスが影響を受けることがあります。これは、意味的にはテンポラリ LOB のコピーをそれぞれのロケータが持つためです。ユーザーが OCILobAssign または PL/SQL の同等の代入を行うたびに、データベースはテンポラリ LOB のコピーを作成します（ただし、パフォーマンス向上のためにこれが後で行われることもあります）。

各ロケータはそれ自身の LOB 値を指します。1 つのロケータがテンポラリ LOB の作成に使用され、OCILobAssign を使用してそのテンポラリ LOB に別の LOB ロケータを割り当てる場合、データベースは元のテンポラリ LOB をコピーして 2 つ目のロケータが元のテンポラリ LOB ではなくコピーを指すようにします。

- **テンポラリ LOB に対する複数ロケータの使用の回避**

複数のユーザーが同じ LOB を変更するには、同じロケータを使用して行う必要があります。テンポラリ LOB は値セマンティクスを使用していますが、OCI 内のロケータへのポインタを使用し、必要に応じて、同じテンポラリ LOB ロケータを指すロケータへの複数のポインタを用意することで、疑似参照セマンティクスを適用できます。PL/SQL では、モジュール間でテンポラリ LOB ロケータを参照によって渡すことによって、同じ効果を得ることができます。これによって、テンポラリ LOB に対して複数ロケータの使用を回避でき、またこのようなモジュールがテンポラリ LOB のローカル・コピーを作成することを阻止できます。

ユーザーがコピーを作成してしまう状況、またサーバーへの余分なラウンドトリップが 1 回以上発生する状況の例を 2 つあげます。

* テンポラリ LOB の他のテンポラリ LOB への割当て

```
DECLARE
  Va BLOB;
  Vb BLOB;
BEGIN
  DBMS_LOB.CREATETEMPORARY (Vb, TRUE);
  DBMS_LOB.CREATETEMPORARY (Va, TRUE);
  Va := Vb;
END;
```

これによって、Oracle が Vb のコピーを作成し、ロケータ Va がこれを指すようにします。Va が参照していたテンポラリ LOB も解放します。

* コレクションから他のコレクションへの割当て

テンポラリ LOB がコレクションの要素であり、コレクションを他のコレクションに割り当てた場合、更新されたテンポラリ LOB ロケータに対するコピーのオーバーヘッドおよび解放のオーバーヘッドが発生することがあります。テンポラリ LOB を属性として含むオブジェクト型を、他の同様のオブジェクト型に割り当て、相互に割り当てられたテンポラリ LOB ロケータがある場合にも、これらのオーバーヘッドが発生します。これは、オブジェクト型がテンポラリ LOB ロケータを指している LOB を属性として持つためです。

参照：

コレクションの詳細は、次のマニュアルを参照してください。

- 『Oracle8i 概要』
- 『Oracle8i アプリケーション開発者ガイド 基礎編』

コレクションまたは複合オブジェクトについて、このような割当ておよびコピー操作を伴うアプリケーションで前述のオーバーヘッドを回避する場合は、内部永続 LOB を使用することをお勧めします。正確にいうと、次のとおりです。

* コレクションまたは複合オブジェクトの代入やコピーを行う場合は、コレクションまたは複合オブジェクトの中にテンポラリ LOB を使用しないでください。

* SELECT 文では、LOB 値を INTO 句でテンポラリ LOB ロケータに代入しないでください。

- ユーザー定義の存続期間内のテンポラリ LOB の解放：ロケータの再割当てによるオーバーヘッドの発生

- **OCI:** 存続期間内のテンポラリ LOB を持ち、その存続期間で `OCIDurationEnd` をコールし、その後にそのテンポラリ LOB のロケータを他の LOB に再度割り当てた場合、オーバーヘッドが発生します。

以前に `OCIDurationEnd` をコールしたかどうかに関係なく、Oracle はロケータが指しているテンポラリ LOB を解放しようとしています。また、そのようなロケータを使用してテンポラリ LOB にアクセスすると、エラーが発生します。ユーザーが一度 `OCIDurationEnd` を発行すると、解放しようとしている LOB を参照しているロケータがまだある場合でも、その存続期間内のすべてのテンポラリ LOB が解放されます。

データベースがオブジェクト・オプションを使用している場合、`OCIDurationBegin` コールを使用して、ユーザー定義の `OCIDurations` を作成できます。ユーザーは、`OCIDurationEnd` をコールして `OCIDuration` を終了できます。存続期間内で存在しているすべてのテンポラリ LOB が解放されます。

- **PL/SQL:** PL/SQL では、ユーザー定義の存続期間は公開されていません。ただし、ユーザーは、事前に定義された存続期間パラメータ `DBMS_LOB.SESSION` または `DBMS_LOB.CALL` を使用してセッション有効範囲またはコール有効範囲のいずれかを指定できます。

テンポラリ LOB におけるセキュリティ上の問題

セキュリティは LOB ロケータを介して提供されています。

- テンポラリ LOB を作成したユーザーのみがアクセスできます。
- ロケータはユーザー・セッションから別のユーザー・セッションに渡されるように設計されていません。ロケータをセッションから別のセッションに渡すことができたとしても、次のようになります。
 - 元のセッションからは、新しいセッション内のテンポラリ LOB にアクセスできません。
 - 新しい（現行の）セッションから、ロケータが移行した元のセッション内のテンポラリ LOB にアクセスできません。
- テンポラリ LOB データの参照は、各ユーザーのセッション内に限定されます。他のセッションからのロケータを他のユーザーが使用しても、同じ `lobid` を持つ自分自身のセッション内の LOB にしかアクセスできません。アプリケーションのユーザーはこのようなことを行うべきではありませんが、行った場合にも他のユーザーのデータには影響を与えることはありません。

NOCOPY 制限事項

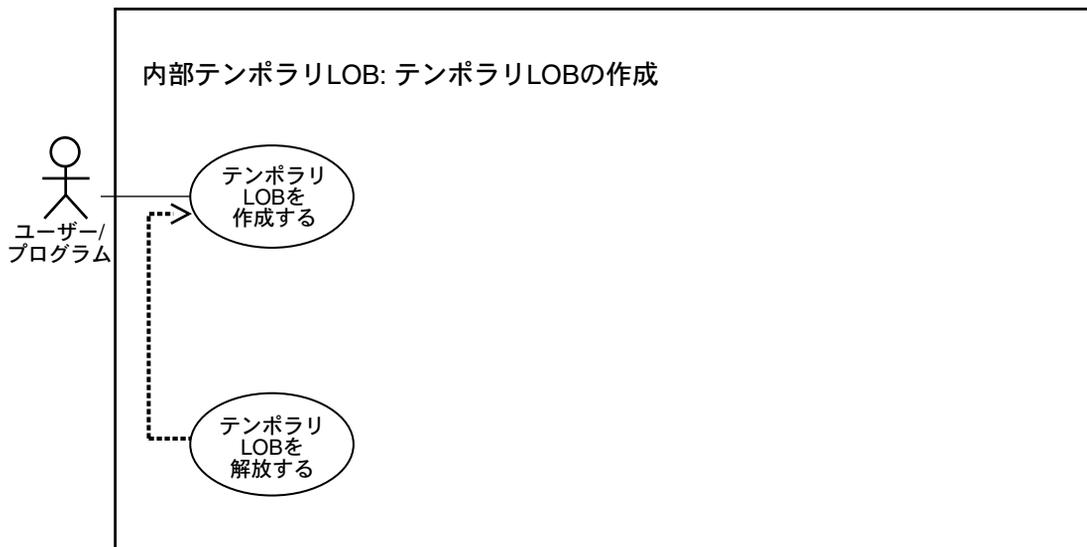
NOCOPY の使用に関するガイドライン、制限事項およびヒントについては、『Oracle8i PL/SQL ユーザーズ・ガイドおよびリファレンス』の第7章「サブプログラム」の「NOCOPY コンパイラ・ヒントの使用」を参照してください。

テンポラリ LOB の管理

Oracle はテンポラリ LOB をセッションごとに追跡しており、`v$temporary_lob` という `v$` ビューを提供します。アプリケーションは、どのユーザーがテンポラリ LOB を所有しているかをセッションから判断できます。この表は、監視のため、およびテンポラリ LOB が使用している一時領域の緊急クリーン・アップのガイドとして、DBA が使用できます。

テンポラリ LOB の作成

図 10-3 ユースケース図 : テンポラリ LOB の作成



参照： 内部テンポラリ LOB に関するすべての基本操作については、10-2 ページの「ユースケース・モデル概要 : 内部テンポラリ LOB」を参照してください。

用途

テンポラリ LOB を作成します。

使用上の注意

テンポラリ LOB は、作成された時点では空です。

テンポラリ LOB は、永続 LOB でサポートされている `EMPTY_BLOB()` 関数または `EMPTY_CLOB()` 関数をサポートしません。`EMPTY_BLOB()` 関数は、LOB が初期化されてはいても何のデータも移入されていないことを意味します。

構文

各プログラム環境で使用可能な機能のリストは、第3章「LOB プログラム環境」を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- PL/SQL (DBMS_LOB) : 『Oracle8i PL/SQL パッケージ・プロシージャ・リファレンス』の第18章「DBMS_LOB」の「サブプログラムの要約」の CREATETEMPORARY プロシージャ、FREETEMPORARY プロシージャ
- OCI (OCI) : 『Oracle8i コール・インタフェース・プログラマーズ・ガイド』の第15章「OCI リレーショナル関数」の「LOB 関数」の OCILobCreateTemporary()
- COBOL (Pro*COBOL) : 『Oracle8i Pro*COBOL プリコンパイラ・プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB DESCRIBE (実行可能埋込み SQL 拡張要素)」、「LOB COPY (実行可能埋込み SQL 拡張要素)」
- C/C++ (Pro*C/C++) : 『Oracle8i Pro*C/C++ プリコンパイラ・プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB DESCRIBE (実行可能埋込み SQL 拡張要素)」、「LOB COPY (実行可能埋込み SQL 拡張要素)」
- Visual Basic (OO4O) : 参照マニュアルはありません。
- Java (JDBC) : 参照マニュアルはありません。

使用例

この例では、Multimedia_tab 表から1つのビデオ・フレームが読み込まれます。その後、テンポラリ LOB が作成され、これを使用してビデオ・イメージが MPEG 形式から JPEG 形式に変換されます。作成されたテンポラリ LOB は CACHE を介して読み込まれ、明示的に解放されなかった場合は、ユーザーのセッションが終了した時点で自動的にクリーン・アップされます。

例

次のプログラム環境の例が示されています。

- [PL/SQL \(DBMS_LOB パッケージ\) : テンポラリ LOB の作成 \(10-16 ページ\)](#)
- [C \(OCI\) : テンポラリ LOB の作成 \(10-16 ページ\)](#)
- [COBOL \(Pro*COBOL\) : テンポラリ LOB の作成 \(10-18 ページ\)](#)
- [C/C++ \(Pro*C/C++\) : テンポラリ LOB の作成 \(10-20 ページ\)](#)
- Visual Basic (OO4O) : 今回のリリースでは例は記載されていません。
- Java (JDBC) : 今回のリリースでは例は記載されていません。

PL/SQL (DBMS_LOB パッケージ) : テンポラリ LOB の作成

注意: 次のようなデータ構造を設定しないと機能しない場合もあります。

```
CREATE TABLE long_raw_tab (id number, long_raw_col long raw);
INSERT INTO long_raw_tab VALUES (1,HEXTORAW('7D'));
INSERT INTO multimedia_tab (clip_id,frame) SELECT
    id,TO_LOB(long_raw_col) FROM long_raw_tab;
```

注意: DBMS_LOB.CREATETEMPORARY プロシージャは、オプションの存続期間パラメータを使用します。PL/SQL では、この存続期間パラメータは LOB データの存続期間についてのヒントとしてのみ使用されます。PL/SQL は、ヒントを考慮して、LOB データの存続期間を内部的に計算します。LOB データの存続期間を指定する必要はありません。

```
DECLARE
    Dest_loc      BLOB;
    Src_loc       BLOB;
    Amount        INTEGER := 4000;
BEGIN
    SELECT Frame INTO Src_loc FROM Multimedia_tab WHERE Clip_ID = 1;
    /* テンポラリ LOB を作成します。 */
    DBMS_LOB.CREATETEMPORARY (Dest_loc,TRUE);
    /* Src_loc からテンポラリ LOB にフレーム全体をコピーします。 */
    DBMS_LOB.COPY (Dest_loc,Src_loc,DBMS_LOB.GETLENGTH (Src_loc),1,1);
    DBMS_LOB.FREETEMPORARY (Dest_loc);
END;
```

C (OCI) : テンポラリ LOB の作成

/* このファンクションは、Multimedia_tab 表から 1 つのビデオ・フレームを読み込みます。その後、テンポラリ LOB が作成されるため、このテンポラリ LOB を使用して、ビデオ・イメージを MPEG 形式から JPEG 形式に変換できます。作成されたテンポラリ LOB は CACHE を介して読み込まれ、すぐに明示的に解放されなかった場合は、ユーザーのセッションが終了した時点で自動的にクリーン・アップされます。このファンクションは、成功した場合は 0 (ゼロ) を返し、失敗した場合は -1 を返します。 */

```
sb4 select_and_createtemp (OCILobLocator *lob_loc,
    OCIError *errhp,
```

```

OCISvcCtx   *svchp,
OCISstmt    *stmthp,
OCIEnv      *envhnp)
{
    OCIDefine   *defnp1;
    OCIBind     *bndhp;
    text       *sqlstmt;
    int rowind =1;
    ub4 loblen = 0;
    OCILobLocator *tblob;
    printf ("in select_and_createtemp \n");
    if(OCIDescriptorAlloc((dvoid*)envhnp, (dvoid **)&tblob,
                          (ub4)OCI_DTYPE_LOB, (size_t)0, (dvoid**)0))
    {
        printf("failed in OCIDescriptor Alloc in select_and_createtemp \n");
        return -1;
    }

    /* Clip_ID =1 で、任意に選択します。*/
    sqlstmt = (text *)
        "SELECT Frame FROM Multimedia_tab WHERE Clip_ID = 1 FOR UPDATE";

    if (OCISstmtPrepare(stmthp, errhp, sqlstmt,
                       (ub4) strlen((char *)sqlstmt),
                       (ub4) OCI_NTV_SYNTAX, (ub4) OCI_DEFAULT))
    {
        (void) printf("FAILED: OCISstmtPrepare() sqlstmt\n");
        return -1;
    }

    /* BLOB に対して定義します。*/
    if (OCIDefineByPos(stmthp,
                      &defnp1, errhp, (ub4) 1, (dvoid *) &lob_loc, (sb4)0,
                      (ub2) SQLT_BLOB, (dvoid *) 0, (ub2 *) 0,
                      (ub2 *) 0, (ub4) OCI_DEFAULT))
    {
        (void) printf("FAILED: Select locator: OCIDefineByPos()\n");
        return -1;
    }

    /* SELECT を実行し、1 行フェッチします。*/
    if (OCISstmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                       (CONST OCISnapshot*) 0, (OCISnapshot*) 0,
                       (ub4) OCI_DEFAULT))
    {
        (void) printf("FAILED: OCISstmtExecute() sqlstmt\n");
    }
}

```

```
        return -1;
    }
    if (OCILobCreateTemporary(svchp,
                              errhp, tblob, (ub2)0, SQLCS_IMPLICIT,
                              OCI_TEMP_BLOB, OCI_ATTR_NOCACHE,
                              OCI_DURATION_SESSION))
    {
        (void) printf("FAILED: CreateTemporary() \n");
        return -1;
    }

    if (OCILobGetLength(svchp, errhp, lob_loc, &lloblen) != 0)
    {
        printf("OCILobGetLength FAILED\n");
        return -1;
    }
    if (OCILobCopy(svchp, errhp, tblob, lob_loc, (ub4)lloblen, (ub4) 1, (ub4) 1))
    {
        printf("OCILobCopy FAILED \n");
    }
    if (OCILobFreeTemporary(svchp, errhp, tblob))
    {
        printf("FAILED: OCILobFreeTemporary call \n");
        return -1;
    }

    return 0;
}
```

COBOL (Pro*COBOL) : テンポラリ LOB の作成

```
IDENTIFICATION DIVISION.
PROGRAM-ID. CREATE-TEMPORARY.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

01  USERID    PIC X(11) VALUES "SAMP/SAMP".
01  BLOB1     SQL-BLOB.
01  TEMP-BLOB SQL-BLOB.
01  LEN       PIC S9(9) COMP.
01  D-LEN     PIC 9(9).
01  ORASLNRD  PIC 9(4).
```

```
EXEC SQL INCLUDE SQLCA END-EXEC.  
EXEC ORACLE OPTION (ORACA=YES) END-EXEC.  
EXEC SQL INCLUDE ORACA END-EXEC.  
PROCEDURE DIVISION.  
CREATE-TEMPORARY.  
EXEC SQL WHENEVER SQLERROR DO PERFORM SQL-ERROR END-EXEC.  
EXEC SQL  
CONNECT :USERID  
END-EXEC.
```

- * CLOB ロケータの割当ておよび初期化を行います。

```
EXEC SQL ALLOCATE :BLOB1 END-EXEC.  
EXEC SQL ALLOCATE :TEMP-BLOB END-EXEC.  
EXEC SQL  
LOB CREATE TEMPORARY :TEMP-BLOB  
END-EXEC.
```

```
EXEC SQL WHENEVER NOT FOUND GOTO END-OF-BLOB END-EXEC.  
EXEC ORACLE OPTION (SELECT_ERROR=NO) END-EXEC.  
EXEC SQL  
SELECT FRAME INTO :BLOB1  
FROM MULTIMEDIA_TAB  
WHERE CLIP_ID = 1  
END-EXEC.
```

- * 永続 BLOB の長さを取得します。

```
EXEC SQL  
LOB DESCRIBE :BLOB1  
GET LENGTH INTO :LEN  
END-EXEC.
```

- * 長さ全体を永続からテンポラリにコピーします。

```
EXEC SQL  
LOB COPY :LEN FROM :BLOB1 TO :TEMP-BLOB  
END-EXEC.
```

- * テンポラリ LOB を解放します。

```
EXEC SQL  
LOB FREE TEMPORARY :TEMP-BLOB  
END-EXEC.
```

```
END-OF-BLOB.  
EXEC SQL WHENEVER NOT FOUND CONTINUE END-EXEC.  
EXEC SQL FREE :BLOB1 END-EXEC.  
EXEC SQL FREE :TEMP-BLOB END-EXEC.
```

```
STOP RUN.

SQL-ERROR.
EXEC SQL WHENEVER SQLERROR CONTINUE END-EXEC.
MOVE ORASLNR TO ORASLNRD.
DISPLAY " ".
DISPLAY "ORACLE ERROR DETECTED ON LINE ", ORASLNRD, ":".
DISPLAY " ".
DISPLAY SQLERRMC.
EXEC SQL ROLLBACK WORK RELEASE END-EXEC.
STOP RUN.
```

C/C++ (Pro*C/C++) : テンポラリ LOB の作成

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

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

void createTempLOB_proc()
{
    OCIBlobLocator *Lob_loc, *Temp_loc;
    int Amount;

    EXEC SQL WHENEVER SQLERROR DO Sample_Error();
    /* LOB ロケータを割り当てます。 */
    EXEC SQL ALLOCATE :Lob_loc;
    EXEC SQL ALLOCATE :Temp_loc;

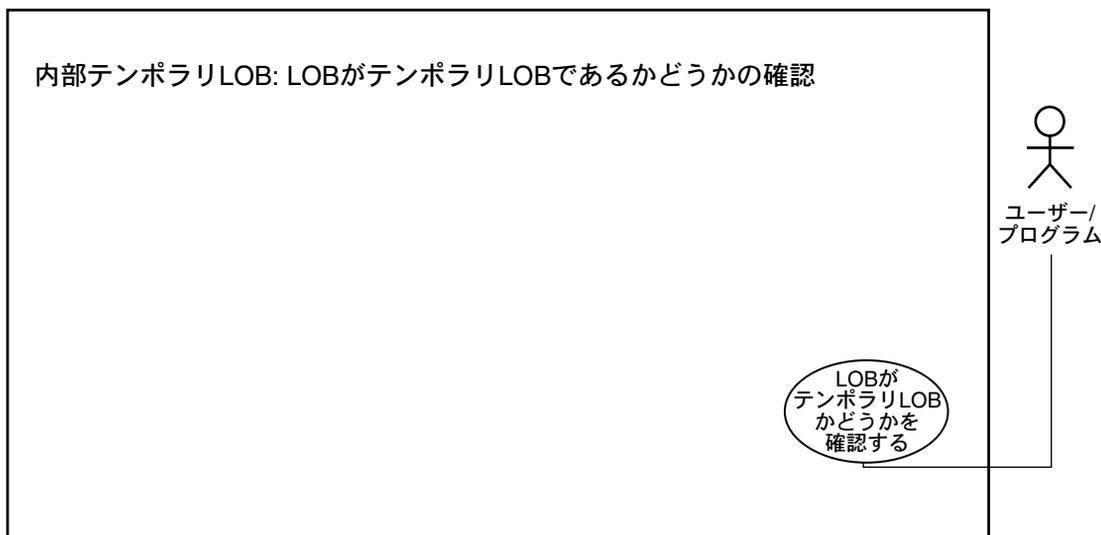
    /* テンポラリ LOB を作成します。 */
    EXEC SQL LOB CREATE TEMPORARY :Temp_loc;
    EXEC SQL SELECT Frame INTO :Lob_loc FROM Multimedia_tab WHERE Clip_ID = 1;

    /* ソース LOB の完全な長さをテンポラリ LOB にコピーします。 */
    EXEC SQL LOB DESCRIBE :Lob_loc GET LENGTH INTO :Amount;
    EXEC SQL LOB COPY :Amount FROM :Lob_loc TO :Temp_loc;
```

```
/* テンポラリ LOB を解放します。*/  
EXEC SQL LOB FREE TEMPORARY :Temp_loc;  
  
/* ロケータが保持しているリソースを解放します。*/  
EXEC SQL FREE :Lob_loc;  
EXEC SQL FREE :Temp_loc;  
}  
  
void main()  
{  
    char *samp = "samp/samp";  
    EXEC SQL CONNECT :samp;  
    createTempLOB_proc();  
    EXEC SQL ROLLBACK WORK RELEASE;  
}
```

LOB がテンポラリ LOB であるかどうかの確認

図 10-4 ユースケース図：LOB がテンポラリ LOB であるかどうかの確認



参照： 内部テンポラリ LOB に関するすべての基本操作については、10-2 ページの「ユースケース・モデル：内部テンポラリ LOB」を参照してください。

用途

LOB がテンポラリ LOB であるかどうかを確認します。

使用上の注意

ありません。

構文

各プログラム環境で使用可能な機能のリストは、第 3 章「LOB プログラム環境」を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- PL/SQL (DBMS_LOB) : 『Oracle8i PL/SQL パッケージ・プロシージャ・リファレンス』の第 18 章「DBMS_LOB」の「サブプログラムの要約」の ISTEMPORARY ファンクション、FREETEMPORARY プロシージャ
- C (OCI) : 『Oracle8i コール・インタフェース・プログラマーズ・ガイド』の第 15 章「OCI リレーショナル関数」の「LOB 関数」の OCILobIsTemporary()
- COBOL (Pro*COBOL) : 『Oracle8i Pro*COBOL プリコンパイラ・プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB DESCRIBE (実行可能埋込み SQL 拡張要素)」、および『Oracle8i PL/SQL パッケージ・プロシージャ・リファレンス』の第 18 章「DBMS_LOB」の「サブプログラムの要約」の ISTEMPORARY ファンクション、FREETEMPORARY プロシージャ
- C/C++ (Pro*C/C++) : 『Oracle8i Pro*C/C++ プリコンパイラ・プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB DESCRIBE (実行可能埋込み SQL 拡張要素)」および『Oracle8i PL/SQL パッケージ・プロシージャ・リファレンス』の第 18 章「DBMS_LOB」の「サブプログラムの要約」の ISTEMPORARY ファンクション、FREETEMPORARY プロシージャ
- Visual Basic (OO4O) : 参照マニュアルはありません。
- Java (JDBC) : 参照マニュアルはありません。

使用例

次の例は、ロケータがテンポラリ LOB に対応付けられているかどうかを問い合わせます。

例

次のプログラム環境の例が示されています。

- [PL/SQL \(DBMS_LOB パッケージ\) : LOB がテンポラリ LOB であるかどうかの確認 \(10-23 ページ\)](#)
- [C \(OCI\) : LOB がテンポラリ LOB であるかどうかの確認 \(10-24 ページ\)](#)
- [COBOL \(Pro*COBOL\) : LOB がテンポラリ LOB であるかどうかの確認 \(10-25 ページ\)](#)
- [C/C++ \(Pro*C/C++\) : LOB がテンポラリ LOB であるかどうかの確認 \(10-26 ページ\)](#)
- Visual Basic (OO4O) : 今回のリリースでは例は記載されていません。
- Java (JDBC) : 今回のリリースでは例は記載されていません。

PL/SQL (DBMS_LOB パッケージ) : LOB がテンポラリ LOB であるかどうかの確認

/* 以下の例も、テンポラリ LOB を解放する例です。まず、LOB ロケータがテンポラリ LOB を指すかどうかを確認し、その後解放します。そうでない場合、エラーが発生します。*/

```
CREATE or REPLACE PROCEDURE freeTempLob_proc(Lob_loc IN OUT BLOB) IS
BEGIN
  /* 渡されたテンポラリ LOB ロケータを解放します。*/
  /* まず、ロケータがテンポラリ LOB を指していることを確認します。*/
  IF DBMS_LOB.ISTEMPORARY(Lob_loc) = 1 THEN
    /* テンポラリ LOB ロケータを解放します。*/
    DBMS_LOB.FREETEMPORARY(Lob_loc);
    DBMS_OUTPUT.PUT_LINE(' temporary LOB was freed!');
  ELSE
    /* エラーが出力されます。*/
    DBMS_OUTPUT.PUT_LINE(
      'Locator passed in was not a temporary LOB locator!');
  END IF;
END;
```

C (OCI) : LOB がテンポラリ LOB であるかどうかの確認

/* このファンクションもテンポラリ LOB を解放します。ファンクションは、ロケータを引数として取り、ロケータがテンポラリ LOB ロケータであるかどうかを確認します。テンポラリ LOB である場合、ファンクションはテンポラリ LOB を解放します。そうでない場合は、ロケータがテンポラリ LOB ロケータではなかった旨のメッセージが出力されます。このファンクションは、成功した場合は 0 (ゼロ) を返し、失敗した場合は -1 を返します。*/

```
sb4 check_and_free_temp(OCILobLocator *tblob,
                        OCLError      *errhp,
                        OCISvcCtx     *svchp,
                        OCISstmt      *stmthp,
                        OCIEnv        *envhp)
{
  boolean is_temp;
  is_temp = FALSE;

  if (OCILobIsTemporary(envhp, errhp, tblob, &is_temp))
  {
    printf ("FAILED: OCILobIsTemporary call\n");
    return -1;
  }
  if(is_temp)
  {
    if(OCILobFreeTemporary(svchp, errhp, tblob))
```

```
{
    printf ("FAILED: OCILobFreeTemporary call\n");
    return -1;

}else
{
    printf("Temporary LOB freed\n");
}
}else
{
    printf("locator is not a temporary LOB locator\n");
}
return 0;
}
```

COBOL (Pro*COBOL) : LOB がテンポラリ LOB であるかどうかの確認

```
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.
```

```
* BLOB ロケータの割当ておよび初期化を行います。
    EXEC SQL ALLOCATE :TEMP-BLOB END-EXEC.
    EXEC SQL
        LOB CREATE TEMPORARY :TEMP-BLOB
    END-EXEC.
```

```
* LOB がテンポラリであるかどうかをチェックします。
EXEC SQL
    LOB DESCRIBE :TEMP-BLOB
    GET ISTEMPORARY INTO :IS-TEMP
END-EXEC.

IF IS-TEMP = 1
*   テンポラリ LOB の論理がここに入ります。
    DISPLAY "LOB is temporary."
ELSE
*   永続 LOB の論理がここに入ります。
    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++) : LOB がテンポラリ LOB であるかどうかの確認

```
#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;

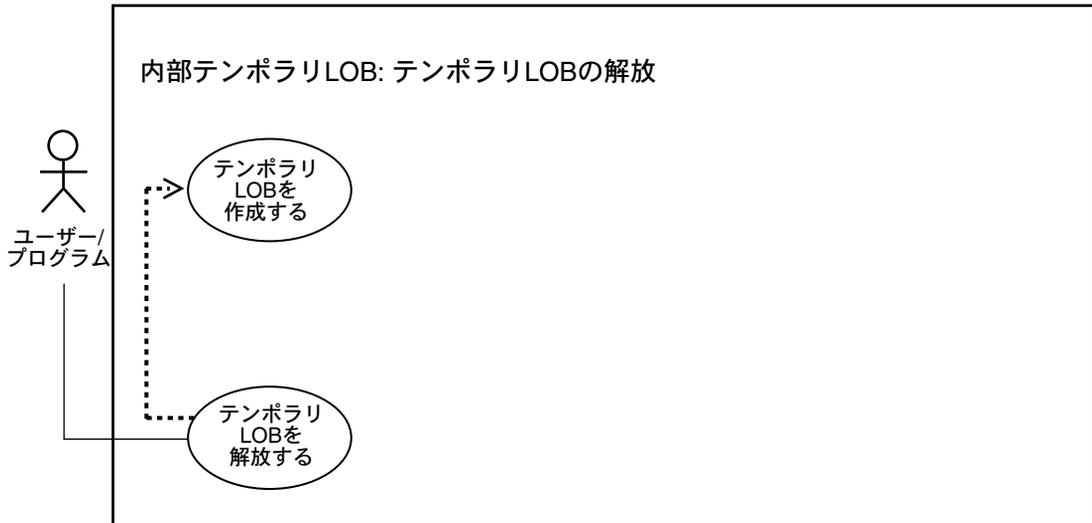
EXEC SQL WHENEVER SQLERROR DO Sample_Error();
/* テンポラリ LOB の割当ておよび作成を行います。*/
EXEC SQL ALLOCATE :Temp_loc;
EXEC SQL LOB CREATE TEMPORARY :Temp_loc;
/* ロケータがテンポラリ LOB ロケータであるかどうかを判断します。*/
EXEC SQL LOB DESCRIBE :Temp_loc GET ISTEMPORARY INTO :isTemporary;

/* この例では、isTemporary は 1 (TRUE) である必要があることに注意してください。*/
if (isTemporary)
    printf("Locator is a Temporary LOB locator\n");
/* テンポラリ LOB を解放します。*/
EXEC SQL LOB FREE TEMPORARY :Temp_loc;
/* ロケータが保持しているリソースを解放します。*/
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;
}
```

テンポラリ LOB の解放

図 10-5 ユースケース図：テンポラリ LOB の解放



参照： 内部テンポラリ LOB に関するすべての基本操作については、10-2 ページの「ユースケース・モデル: 内部テンポラリ LOB」を参照してください。

用途

テンポラリ LOB を解放します。

使用上の注意

テンポラリ LOB インスタンスは、たとえば OCI または DBMS_LOB パッケージで、適切な `FREETEMPORARY`、`OCIDurationEnd` または `OCILOBFreeTemporary` 文を使用してのみ破棄できます。

テンポラリ LOB を永続的なものにするには、OCI または `DBMS_LOB copy()` コマンドを明示的に使用して、テンポラリ LOB を永続的な LOB にコピーする必要があります。

構文

各プログラム環境で使用可能な機能のリストは、[第 3 章「LOB プログラム環境」](#)を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- PL/SQL (DBMS_LOB) : 『Oracle8i PL/SQL パッケージ・プロシージャ・リファレンス』の第 18 章「DBMS_LOB」の「サブプログラムの要約」の FREETEMPORARY プロシージャ
- C (OCI) : 『Oracle8i コール・インタフェース・プログラマーズ・ガイド』の第 15 章「OCI リレーショナル関数」の「LOB 関数」の OCILobFreeTemporary()
- COBOL (Pro*COBOL) : 『Oracle8i Pro*COBOL プリコンパイラ・プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB FREE TEMPORARY (実行可能埋込み SQL 拡張要素)」
- C/C++ (Pro*C/C++) : 『Oracle8i Pro*C/C++ プリコンパイラ・プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB FREE TEMPORARY (実行可能埋込み SQL 拡張要素)」
- Visual Basic (OO4O) : 参照マニュアルはありません。
- Java (JDBC) : 参照マニュアルはありません。

使用例

ありません。

例

次のプログラム環境の例が示されています。

- [PL/SQL \(DBMS_LOB パッケージ\) : テンポラリ LOB の解放 \(10-29 ページ\)](#)
- [C \(OCI\) : テンポラリ LOB の解放 \(10-30 ページ\)](#)
- [COBOL \(Pro*COBOL\) : テンポラリ LOB の解放 \(10-31 ページ\)](#)
- [C/C++ \(Pro*C/C++\) : テンポラリ LOB の解放 \(10-32 ページ\)](#)
- Visual Basic (OO4O) : 今回のリリースでは例は記載されていません。
- Java (JDBC) : 今回のリリースでは例は記載されていません。

PL/SQL (DBMS_LOB パッケージ) : テンポラリ LOB の解放

```

DECLARE
    Dest_loc      BLOB;
    Src_loc       BFILE := BFILENAME('AUDIO_DIR', 'Washington_audio');
    Amount        INTEGER := 4000;
BEGIN
    DBMS_LOB.CREATETEMPORARY (Dest_loc,TRUE);
    /* BFILE のオープンは必須です。 */
    DBMS_LOB.OPEN (Src_loc, DBMS_LOB.LOB_READONLY);
    /* LOB のオープンはオプションです。 */
    DBMS_LOB.OPEN (Dest_loc,DBMS_LOB.LOB_READWRITE);
    DBMS_LOB.LOADFROMFILE (Dest_loc, Src_loc, Amount);
    /* LOB をオープンしている場合、その LOB をクローズする必要があります。 */
    DBMS_LOB.CLOSE (Src_loc);
    DBMS_LOB.CLOSE (Dest_loc);
    /* テンポラリ LOB を解放します。 */
    DBMS_LOB.FREETEMPORARY (Dest_loc);
END;

```

C (OCI) : テンポラリ LOB の解放

/* このファンクションは、テンポラリ LOB を作成し、その後解放します。
 このファンクションは、成功した場合は 0 (ゼロ) を返し、失敗した場合は -1 を返します。 */

```

sb4 freeTempLob (OCIError      *errhp,
                OCISvcCtx     *svchp,
                OCISstmt      *stmthp,
                OCIEnv        *envhp)
{
    OCILobLocator *tblob;
    checkerr (errhp,OCIDescriptorAlloc ((dvoid*)envhp, (dvoid **)&tblob,
                                        (ub4)OCI_DTYPE_LOB, (size_t)0,
                                        (dvoid**)0));
    if (OCILobCreateTemporary (svchp, errhp, tblob, (ub2)0, SQLCS_IMPLICIT,
                               OCI_TEMP_BLOB, OCI_ATTR_NOCACHE,
                               OCI_DURATION_SESSION))
    {
        (void) printf ("FAILED:CreateTemporary():freeTempLob\n");
        return -1;
    }

    if (OCILobFreeTemporary (svchp, errhp, tblob))
    {
        printf ("FAILED: OCILobFreeTemporary call in freeTempLob\n");
        return -1;
    }
}

```

```
}else
{
printf("Temporary LOB freed in freeTempLob\n");
}
return 0;
}
```

COBOL (Pro*COBOL) : テンポラリ LOB の解放

```
IDENTIFICATION DIVISION.
PROGRAM-ID. FREE-TEMPORARY.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
```

```
01 USERID PIC X(11) VALUES "SAMP/SAMP".
```

```
01 TEMP-BLOB SQL-BLOB.
01 IS-TEMP PIC S9(9) COMP.
01 ORASLNRD PIC 9(4).
EXEC SQL INCLUDE SQLCA END-EXEC.
EXEC ORACLE OPTION (ORACA=YES) END-EXEC.
EXEC SQL INCLUDE ORACA END-EXEC.
```

```
PROCEDURE DIVISION.
FREE-TEMPORARY.
```

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

- * BLOB ロケータの割当ておよび初期化を行います。

```
EXEC SQL ALLOCATE :TEMP-BLOB END-EXEC.
EXEC SQL
LOB CREATE TEMPORARY :TEMP-BLOB
END-EXEC.
```
- * テンポラリ LOB に何らかの操作を行います。
- * テンポラリ LOB を解放します。

```
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++) : テンポラリ LOB の解放

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

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

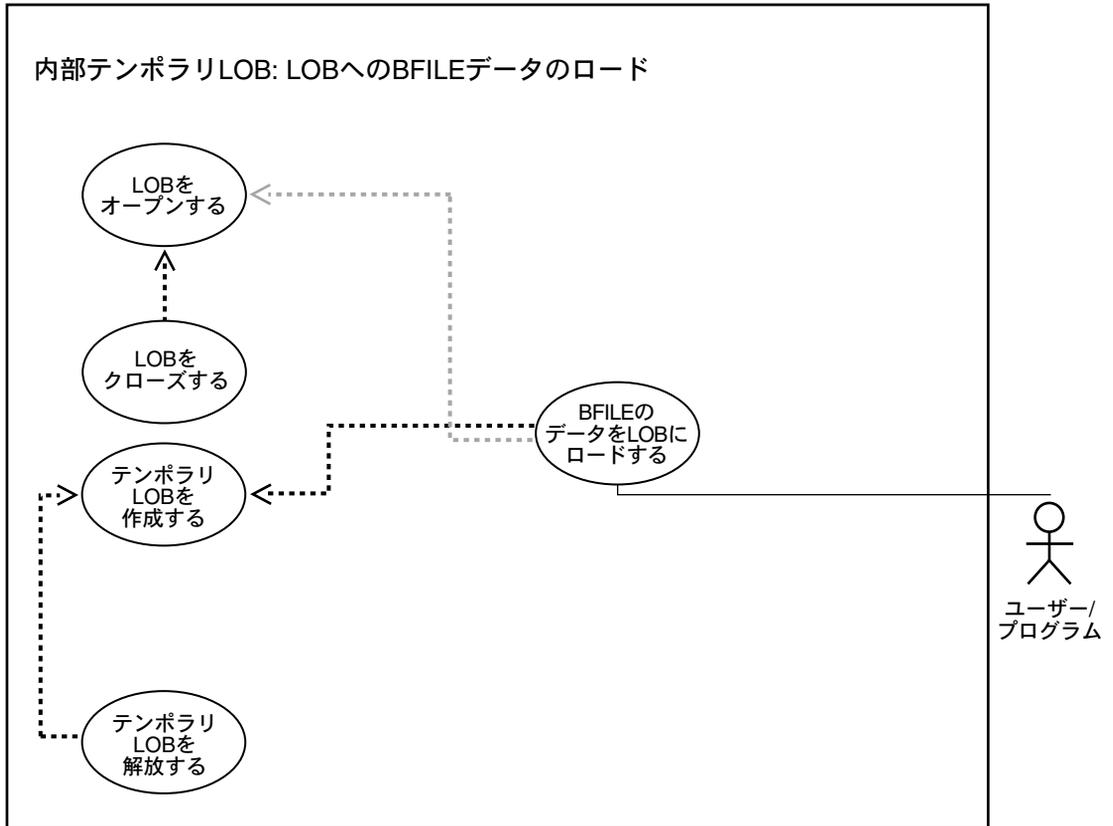
void freeTempLob_proc()
{
    OCIBlobLocator *Temp_loc;

    EXEC SQL WHENEVER SQLERROR DO Sample_Error();
    EXEC SQL ALLOCATE :Temp_loc;
    EXEC SQL LOB CREATE TEMPORARY :Temp_loc;
    /* テンポラリ LOB に何らかの操作を行います。 */
    EXEC SQL LOB FREE TEMPORARY :Temp_loc;
    EXEC SQL FREE :Temp_loc;
}

void main()
{
    char *samp = "samp/samp";
    EXEC SQL CONNECT :samp;
    freeTempLob_proc();
    EXEC SQL ROLLBACK WORK RELEASE;
}
```

テンポラリ LOB への BFILE データのロード

図 10-6 ユースケース図: テンポラリ LOB への BFILE データのロード



参照: 内部テンポラリ LOB に関するすべての基本操作については、10-2 ページの「ユースケース・モデル: 内部テンポラリ LOB」を参照してください。

用途

BFILE のデータをテンポラリ LOB にロードします。

使用上の注意

OCI または OCI 機能にアクセスするプログラム環境を使用する場合、キャラクタ・セットの変換は 1 つのキャラクタ・セットから別のキャラクタ・セットに変換するとき、暗黙的に実行されます。ただし、バイナリ・データからキャラクタ・セットへの場合は、暗黙の変換は実行されません。loadfromfile 操作を使用して CLOB または NCLOB に入れる場合は、BFILE のバイナリ・データを LOB に入れることとなります。この場合、loadfromfile を実行する前に、BFILE のデータに対しキャラクタ・セットの変換を実行しておく必要があります。

構文

各プログラム環境で使用可能な機能のリストは、第 3 章「LOB プログラム環境」を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- PL/SQL (DBMS_LOB) : 『Oracle8i PL/SQL パッケージ・プロシージャ・リファレンス』の第 18 章「DBMS_LOB」の「サブプログラムの要約」の CREATETEMPORARY プロシージャ、FREETEMPORARY プロシージャ
- C (OCI) : 『Oracle8i コール・インタフェース・プログラマーズ・ガイド』の第 15 章「OCI リレーショナル関数」の「LOB 関数」の OCILobLoadFromFile()
- COBOL (Pro*COBOL) : 『Oracle8i Pro*COBOL プリコンパイラ・プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB LOAD (実行可能埋込み SQL 拡張要素)」
- C/C++ (Pro*C/C++) : 『Oracle8i Pro*C/C++ プリコンパイラ・プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB LOAD (実行可能埋込み SQL 拡張要素)」
- Visual Basic (OO4O) : 参照マニュアルはありません。
- Java (JDBC) : 参照マニュアルはありません。

使用例

この例のプロシージャは、ターゲット LOB にロードする LOB データを含むオペレーティング・システムのソース・ディレクトリ (AUDIO_DIR) が存在することを想定しています。

例

次のプログラム環境の例が示されています。

- [PL/SQL \(DBMS_LOB パッケージ\) : テンポラリ LOB への BFILE データのロード \(10-34 ページ\)](#)
- [C \(OCI\) : テンポラリ LOB への BFILE データのロード \(10-35 ページ\)](#)

- COBOL (Pro*COBOL) : テンポラリ LOB への BFILE データのロード (10-37 ページ)
- C/C++ (Pro*C/C++) : テンポラリ LOB への BFILE データのロード (10-38 ページ)
- Visual Basic (OO4O) : 今回のリリースでは例は記載されていません。
- Java (JDBC) : 今回のリリースでは例は記載されていません。

PL/SQL (DBMS_LOB パッケージ) : テンポラリ LOB への BFILE データのロード

/* 例のプロシージャ freeTempLob_proc は、DBMS_LOB パッケージの一部ではないことに注意してください。*/

```
CREATE or REPLACE PROCEDURE freeTempLob_proc(Lob_loc IN OUT BLOB) IS
BEGIN
    DBMS_LOB.CREATETEMPORARY(Lob_loc,TRUE);
    /* テンポラリ LOB ロケータを使用し、その後解放します。*/
    /* テンポラリ LOB ロケータを解放します。*/
    DBMS_LOB.FREETEMPORARY(Lob_loc);
    DBMS_OUTPUT.PUT_LINE('Temporary LOB was freed');
END;
```

C (OCI) : テンポラリ LOB への BFILE データのロード

/* ここでは、テンポラリ LOB を作成する方法および BFILE の内容をテンポラリ LOB にロードする方法を示します。*/

```
sb4 load_temp(OCIError *errhp,
              OCISvcCtx *svchp,
              OCIStmt *stmthp,
              OCIEnv *envhp)
{
    OCILobLocator *bfile;
    int amount =100;
    OCILobLocator *tblob;

    printf("in load_temp\n");
    if(OCIDescriptorAlloc((dvoid*)envhp, (dvoid **)&tblob,
                          (ub4)OCI_DTYPE_LOB, (size_t)0, (dvoid**)0))
    {
        printf("OCIDescriptorAlloc failed in load_temp\n");
        return -1;
    }
    if(OCIDescriptorAlloc((dvoid*)envhp, (dvoid **)&bfile,
                          (ub4)OCI_DTYPE_FILE, (size_t)0, (dvoid**)0))
```

```

{
    printf("OCIDescriptorAlloc failed in load_temp\n");
    return -1;
}

/* テンポラリ LOB を作成します。*/
if (OCILobCreateTemporary(svchp, errhp, tblob, (ub2)0,
                          SQLCS_IMPLICIT, OCI_TEMP_BLOB,
                          OCI_ATTR_NOCACHE, OCI_DURATION_SESSION))
{
    (void) printf("FAILED: CreateTemporary() \n");
    return -1;
}

if (OCILobFileSetName(envhp, errhp, &bfile, (text *)"AUDIO_DIR",
                      (ub2)strlen("AUDIO_DIR"), (text *)"Washington_audio",
                      (ub2)strlen("Washington_audio")))
{
    printf("OCILobFileSetName FAILED in load_temp\n");
    return -1;
}

/* BFILE のオープンは必須です。*/
if (OCILobFileOpen(svchp, errhp, (OCILobLocator *) bfile, OCI_LOB_READONLY))
{
    printf("OCILobFileOpen FAILED for the bfile load_temp \n");
    return -1;
}

/* LOB のオープンはオプションです。*/
if (OCILobOpen(svchp, errhp, (OCILobLocator *) tblob, OCI_LOB_READWRITE))
{
    printf("OCILobOpen FAILED for temp LOB \n");
    return -1;
}

if (OCILobLoadFromFile(svchp,
                      errhp,
                      tblob,
                      (OCILobLocator*)bfile,
                      (ub4)amount,
                      (ub4)1, (ub4)1))
{
    printf("OCILobLoadFromFile FAILED\n");
    return -1;
}

```

```

/* LOB をクローズします。*/
if (OCILobFileClose(svchp, errhp, (OCILobLocator *) bfile))
{
    printf( "OCILobClose FAILED for bfile \n");
    return -1;
}

checkerr(errhp, (OCILobClose(svchp, errhp, (OCILobLocator *) tblob)));

/* 使い終わったテンポラリ LOB を解放します。*/
if (OCILobFreeTemporary(svchp, errhp, tblob))
{
    printf("OCILobFreeTemporary FAILED \n");
    return -1;
}
}
}

```

COBOL (Pro*COBOL) : テンポラリ LOB への BFILE データのロード

```

IDENTIFICATION DIVISION.
PROGRAM-ID. TEMP-LOB-ISOPEN.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

01 USERID    PIC X(11) VALUES "SAMP/SAMP".
01 TEMP-BLOB    SQL-BLOB.
01 SRC-BFILE    SQL-BFILE.
01 DIR-ALIAS    PIC X(30) VARYING.
01 FNAME        PIC X(20) VARYING.
01 DIR-IND      PIC S9(4) COMP.
01 FNAME-IND    PIC S9(4) COMP.
01 AMT          PIC S9(9) COMP.
01 IS-OPEN     PIC S9(9) COMP.
01 ORASLNDRD   PIC 9(4) .

EXEC SQL INCLUDE SQLCA END-EXEC.
EXEC ORACLE OPTION (ORACA=YES) END-EXEC.
EXEC SQL INCLUDE ORACA END-EXEC.

PROCEDURE DIVISION.
TEMP-LOB-ISOPEN.
EXEC SQL WHENEVER SQLERROR DO PERFORM SQL-ERROR END-EXEC.
EXEC SQL
    CONNECT :USERID
END-EXEC.

```

```
* BLOB ロケータの割当ておよび初期化を行います。
EXEC SQL ALLOCATE :TEMP-BLOB END-EXEC.
EXEC SQL
      LOB CREATE TEMPORARY :TEMP-BLOB
END-EXEC.

* テンポラリ LOB をオープンします。
EXEC SQL LOB OPEN :TEMP-BLOB READ ONLY END-EXEC.
EXEC SQL
      LOB DESCRIBE :TEMP-BLOB GET ISOPEN INTO :IS-OPEN
END-EXEC.

IF IS-OPEN = 1
*   オープンしているテンポラリ LOB の論理がここに入ります。
      DISPLAY "Temporary LOB is OPEN."
ELSE
*   クローズしたテンポラリ LOB の論理がここに入ります。
      DISPLAY "Temporary LOB is CLOSED."
END-IF.
EXEC SQL
      ROLLBACK WORK RELEASE
END-EXEC.
STOP RUN.

SQL-ERROR.
EXEC SQL
      WHENEVER SQLERROR CONTINUE
END-EXEC.
MOVE ORASLNR TO ORASLNRD.
DISPLAY " ".
DISPLAY "ORACLE ERROR DETECTED ON LINE ", ORASLNRD, ":".
DISPLAY " ".
DISPLAY SQLERRMC.
EXEC SQL
      ROLLBACK WORK RELEASE
END-EXEC.
STOP RUN.
```

C/C++ (Pro*C/C++) : テンポラリ LOB への BFILE データのロード

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

void Sample_Error()
```

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

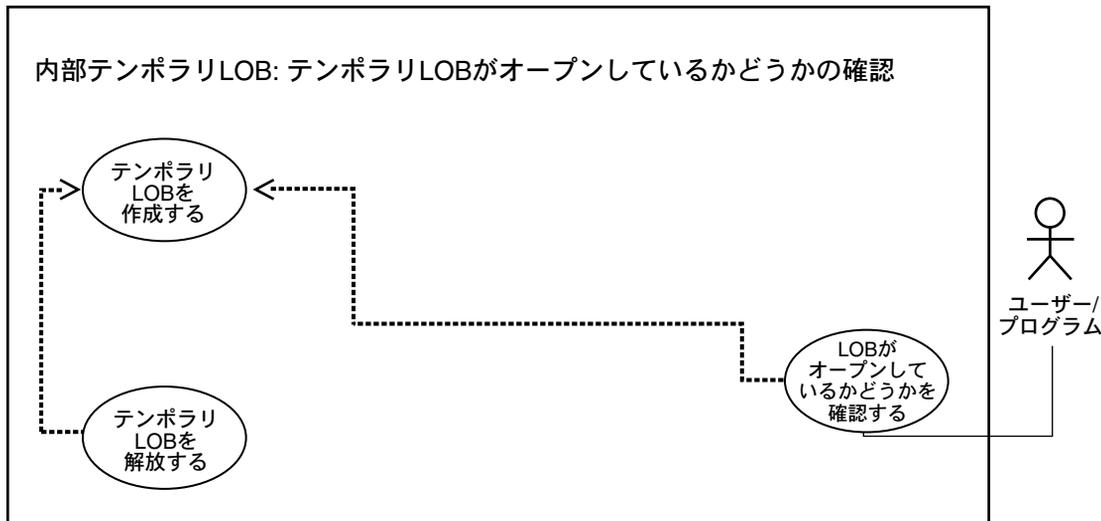
void loadTempLobFromBFILE_proc()
{
OCIBlobLocator *Temp_loc;
OCIBFileLocator *Lob_loc;
char *Dir = "AUDIO_DIR", *Name = "Washington_audio";
int Amount = 4096;

EXEC SQL WHENEVER SQLERROR DO Sample_Error();
/* テンポラリ LOB の割当ておよび作成を行います。*/
EXEC SQL ALLOCATE :Temp_loc;
EXEC SQL LOB CREATE TEMPORARY :Temp_loc;
/* BFILE ロケータの割当ておよび初期化を行います。*/
EXEC SQL ALLOCATE :Lob_loc;
EXEC SQL LOB FILE SET :Lob_loc DIRECTORY = :Dir, FILENAME = :Name;
/* BFILE のオープンはオプションです。*/
/* LOB のオープンはオプションです。*/
EXEC SQL LOB OPEN :Lob_loc READ ONLY;
EXEC SQL LOB OPEN :Temp_loc READ WRITE;
/* BFILE からテンポラリ LOB にデータをロードします。*/
EXEC SQL LOB LOAD :Amount FROM FILE :Lob_loc INTO :Temp_loc;
/* LOB をオープンしている場合、その LOB をクローズする必要があります。*/
EXEC SQL LOB CLOSE :Temp_loc;
EXEC SQL LOB CLOSE :Lob_loc;
/* テンポラリ LOB を解放します。*/
EXEC SQL LOB FREE TEMPORARY :Temp_loc;
/* ロケータが保持しているリソースを解放します。*/
EXEC SQL FREE :Temp_loc;
EXEC SQL FREE :Lob_loc;
}

void main()
{
char *samp = "samp/samp";
EXEC SQL CONNECT :samp;
loadTempLobFromBFILE_proc();
EXEC SQL ROLLBACK WORK RELEASE;
}
```

テンポラリ LOB がオープンしているかどうかの確認

図 10-7 ユースケース図：テンポラリ LOB がオープンしているかどうかの確認



参照： 内部テンポラリ LOB に関するすべての基本操作については、10-2 ページの「ユースケース・モデル: 内部テンポラリ LOB」を参照してください。

用途

テンポラリ LOB がオープンしているかどうかを確認します。

使用上の注意

ありません。

構文

各プログラム環境で使用可能な機能のリストは、第 3 章「LOB プログラム環境」を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- PL/SQL (DBMS_LOB) : 『Oracle8i PL/SQL パッケージ・プロシージャ・リファレンス』の第 18 章「DBMS_LOB」の「サブプログラムの要約」の ISOPEN ファンクション

- C (OCI) : 『Oracle8i コール・インタフェース・プログラマーズ・ガイド』の第15章「OCI リレーショナル関数」の「LOB 関数」の OCILOBIsOpen()
- COBOL (Pro*COBOL) : 『Oracle8i Pro*COBOL プリコンパイラ・プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB DESCRIBE (実行可能埋込み SQL 拡張要素)」、および『Oracle8i PL/SQL パッケージ・プロシージャ・リファレンス』の第18章「DBMS_LOB」の「サブプログラムの要約」の ISOPEN ファンクション
- C/C++ (Pro*C/C++) : 『Oracle8i Pro*C/C++ プリコンパイラ・プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB DESCRIBE (実行可能埋込み SQL 拡張要素)」、および『Oracle8i PL/SQL パッケージ・プロシージャ・リファレンス』の第18章「DBMS_LOB」の「サブプログラムの要約」の ISOPEN ファンクション
- Visual Basic (OO4O) : 参照マニュアルはありません。
- Java (JDBC) : 参照マニュアルはありません。

使用例

この例では、ロケータを入力として取り、テンポラリ LOB を作成してこれをオープンし、さらにこの LOB がオープンしているかどうかをテストします。

例

次のプログラム環境の例が示されています。

- [PL/SQL: テンポラリ LOB がオープンしているかどうかの確認 \(10-41 ページ\)](#)
- [C \(OCI\) : テンポラリ LOB がオープンしているかどうかの確認 \(10-42 ページ\)](#)
- [COBOL \(Pro*COBOL\) : テンポラリ LOB がオープンしているかどうかの確認 \(10-43 ページ\)](#)
- [C/C++ \(Pro*C/C++\) : テンポラリ LOB がオープンしているかどうかの確認 \(10-44 ページ\)](#)
- Visual Basic (OO4O) : 今回のリリースでは例は記載されていません。
- Java (JDBC) : 今回のリリースでは例は記載されていません。

PL/SQL: テンポラリ LOB がオープンしているかどうかの確認

```
/* 例のプロシージャ seeTempLOBIsOpen_proc は、DBMS_LOB パッケージの
一部ではないことに注意してください。このプロシージャは、ロケータを入力と
して取り、テンポラリ LOB を作成してこれをオープンし、さらにこの LOB がオープ
ンしているかどうかをテストします。*/
```

```
CREATE OR REPLACE PROCEDURE seeTempLOBIsOpen_proc(lob_loc IN OUT BLOB,
```

```
Retval OUT INTEGER) IS
BEGIN
  /* テンポラリ LOB を作成します。 */
  DBMS_LOB.CREATETEMPORARY (Lob_loc, TRUE);
  /* LOB オープンしているかどうかを確認します。 */
  Retval := DBMS_LOB.ISOPEN (Lob_loc);
  /* LOB がオープンしている場合、Retval の値は 1 になります。 */
  /* テンポラリ LOB を解放します。 */
  DBMS_LOB.FREETEMPORARY (Lob_loc);
END;
```

C (OCI) : テンポラリ LOB がオープンしているかどうかの確認

/* このファンクションは、ロケータを取り、ファンクションが成功した場合に 0 (ゼロ) を戻します。ファンクションは、"Temporary LOB is open" または "Temporary LOB is closed" というメッセージを出力します。ロケータがテンポラリ LOB を実際に指しているかどうかをチェックしませんが、オープンまたはクローズ・テストのどちらかは動作します。このファンクションは、成功した場合は 0 (ゼロ) を戻し、失敗した場合は -1 を戻します。 */

```
sb4 seeTempLOBIsOpen (OCILobLocator *lob_loc,
                     OCIError      *errhp,
                     OCISvcCtx     *svchp,
                     OCISstmt      *stmthp,
                     OCIEnv        *envhp)
{
  boolean is_open = FALSE;

  printf("in seeTempLOBIsOpen \n");

  if (OCILobCreateTemporary (svchp,
                             errhp,
                             lob_loc,
                             (ub2) 0,
                             SQLCS_IMPLICIT,
                             OCI_TEMP_BLOB,
                             OCI_ATTR_NOCACHE,
                             OCI_DURATION_SESSION))
  {
    (void) printf("FAILED: CreateTemporary() \n");
    return -1;
  }

  if (OCILobIsOpen (svchp, errhp, lob_loc, &is_open))
  {
    printf("OCILobIsOpen FAILED\n");
  }
}
```

```

        return -1;
    }
    if(is_open)
    {
        printf("Temporary LOB is open\n");
    }
    else
    {
        printf("Temporary LOB is closed\n");
    }
}

if(OCILobFreeTemporary(svchp, errhp, lob_loc))
{
    printf("OCILobFreeTemporary FAILED \n");
    return -1;
}
return 0;
}

```

COBOL (Pro*COBOL) : テンポラリ LOB がオープンしているかどうかの確認

```

IDENTIFICATION DIVISION.
PROGRAM-ID. TEMP-LOB-ISOPEN.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

01  USERID    PIC X(11) VALUES "SAMP/SAMP".
01  TEMP-BLOB SQL-BLOB.
01  SRC-BFILE SQL-BFILE.
01  DIR-ALIAS PIC X(30) VARYING.
01  FNAME     PIC X(20) VARYING.
01  DIR-IND   PIC S9(4) COMP.
01  FNAME-IND PIC S9(4) COMP.
01  AMT       PIC S9(9) COMP.
01  IS-OPEN   PIC S9(9) COMP.
01  ORASLNRD  PIC 9(4) .

EXEC SQL INCLUDE SQLCA END-EXEC.
EXEC ORACLE OPTION (ORACA=YES) END-EXEC.
EXEC SQL INCLUDE ORACA END-EXEC.

PROCEDURE DIVISION.
TEMP-LOB-ISOPEN.
EXEC SQL WHENEVER SQLERROR DO PERFORM SQL-ERROR END-EXEC.
EXEC SQL

```

テンポラリ LOB がオープンしているかどうかの確認

```
CONNECT :USERID
END-EXEC.

* BLOB ロケータの割当ておよび初期化を行います。
EXEC SQL ALLOCATE :TEMP-BLOB END-EXEC.
EXEC SQL
    LOB CREATE TEMPORARY :TEMP-BLOB
END-EXEC.

* テンポラリ LOB をオープンします。
EXEC SQL LOB OPEN :TEMP-BLOB READ ONLY END-EXEC.
EXEC SQL
    LOB DESCRIBE :TEMP-BLOB GET ISOPEN INTO :IS-OPEN
END-EXEC.

IF IS-OPEN = 1
*   オープンしているテンポラリ LOB の論理がここに入ります。
    DISPLAY "Temporary LOB is OPEN."
ELSE
*   クローズしたテンポラリ LOB の論理がここに入ります。
    DISPLAY "Temporary LOB is CLOSED."
END-IF.
EXEC SQL
    ROLLBACK WORK RELEASE
END-EXEC.
STOP RUN.

SQL-ERROR.
EXEC SQL
    WHENEVER SQLERROR CONTINUE
END-EXEC.
MOVE ORASLNR TO ORASLNRD.
DISPLAY " ".
DISPLAY "ORACLE ERROR DETECTED ON LINE ", ORASLNRD, ": ".
DISPLAY " ".
DISPLAY SQLERRMC.
EXEC SQL
    ROLLBACK WORK RELEASE
END-EXEC.
STOP RUN.
```

C/C++ (Pro*C/C++) : テンポラリ LOB がオープンしているかどうかの確認

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

```
void Sample_Error()
{
    EXEC SQL WHENEVER SQLERROR CONTINUE;
    printf("%.s\n", sqlca.sqlterm.sqlterm1, sqlca.sqlterm.sqltermc);
    EXEC SQL ROLLBACK WORK RELEASE;
    exit(1);
}

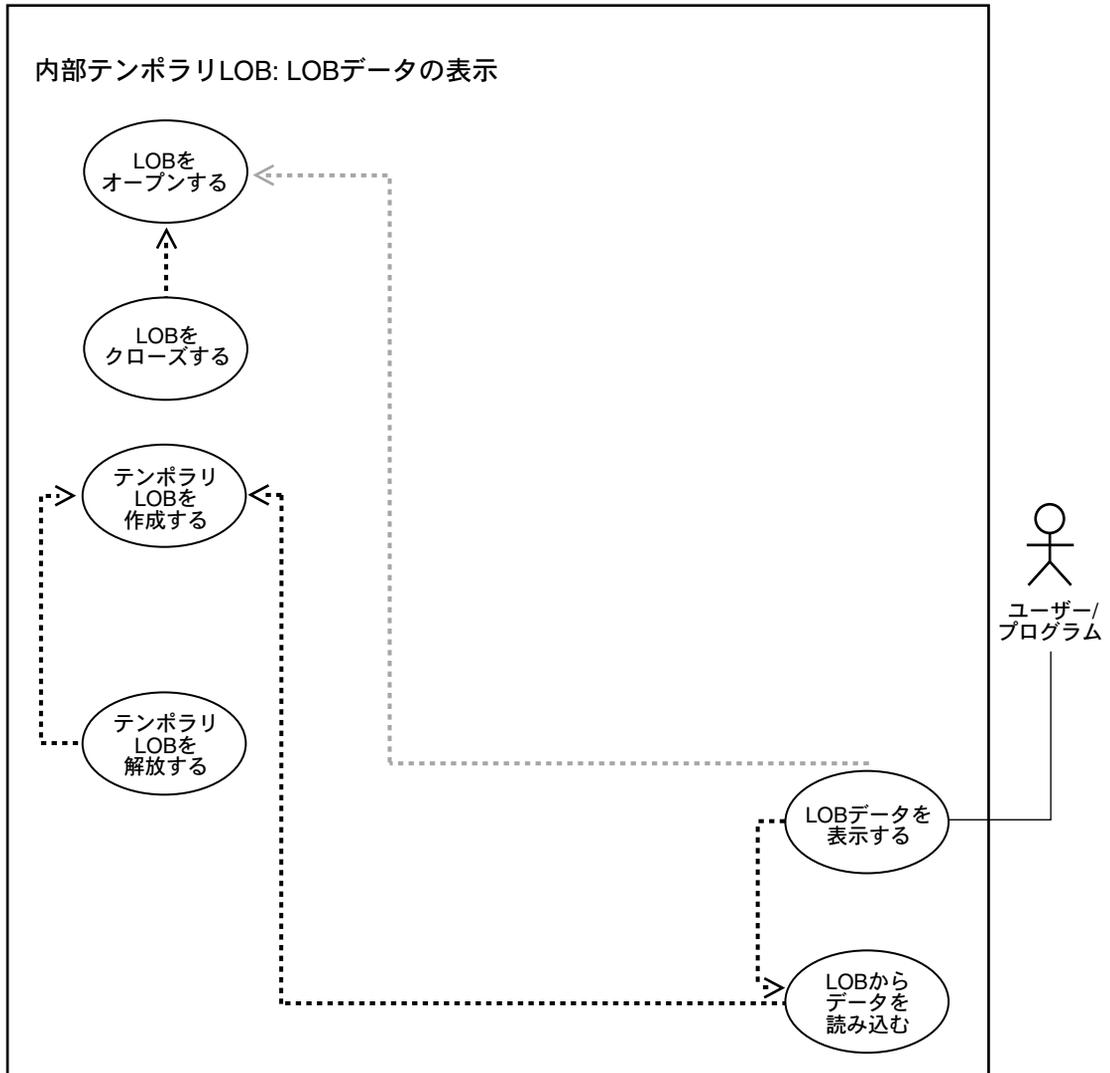
void tempLobIsOpen_proc()
{
    OCIBlobLocator *Temp_loc;
    int isOpen = 0;

    EXEC SQL WHENEVER SQLERROR DO Sample_Error();
    /* テンポラリ LOB の割当ておよび作成を行います。*/
    EXEC SQL ALLOCATE :Temp_loc;
    EXEC SQL LOB CREATE TEMPORARY :Temp_loc;
    /* テンポラリ LOB をオープンします。*/
    EXEC SQL LOB OPEN :Temp_loc READ ONLY;
    /* LOB がオープンしているかどうかを判断します。*/
    EXEC SQL LOB DESCRIBE :Temp_loc GET ISOPEN INTO :isOpen;
    if (isOpen)
        printf("Temporary LOB is open\n");
    else
        printf("Temporary LOB is not open\n");
    /* この例では、LOB がオープンしているため、isOpen == 1 (TRUE) であることに
       注意してください。*/
    /* LOB をクローズします。*/
    EXEC SQL LOB CLOSE :Temp_loc;
    /* テンポラリ LOB を解放します。*/
    EXEC SQL LOB FREE TEMPORARY :Temp_loc;
    /* ロケータが保持しているリソースを解放します。*/
    EXEC SQL FREE :Temp_loc;
}

void main()
{
    char *samp = "samp/samp";
    EXEC SQL CONNECT :samp;
    tempLobIsOpen_proc();
    EXEC SQL ROLLBACK WORK RELEASE;
}
```

テンポラリ LOB データの表示

図 10-8 ユースケース図：テンポラリ LOB データの表示



用途

テンポラリ LOB データを表示します。

使用上の注意

ありません。

構文

各プログラム環境で使用可能な機能のリストは、第3章「LOB プログラム環境」を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- PL/SQL (DBMS_LOB) : 『Oracle8i PL/SQL パッケージ・プロシージャ・リファレンス』の第18章「DBMS_LOB」の「サブプログラムの要約」の OPEN プロシージャ、LOADFROMFILE プロシージャ、READ プロシージャ、および第29章「DBMS_OUTPUT」の「サブプログラムの要約」の PUT および PUT_LINE プロシージャ
- C (OCI) : 『Oracle8i コール・インタフェース・プログラマーズ・ガイド』の第15章「OCI リレーショナル関数」の「LOB 関数」の OCILobRead()
- COBOL (Pro*COBOL) : 『Oracle8i Pro*COBOL プリコンパイラ・プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB READ (実行可能埋込み SQL 拡張要素)」
- C/C++ (Pro*C/C++) : 『Oracle8i Pro*C/C++ プリコンパイラ・プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB READ (実行可能埋込み SQL 拡張要素)」
- Visual Basic (OO4O) : 参照マニュアルはありません。
- Java (JDBC) : 参照マニュアルはありません。

使用例

LOB を表示する例として、データを表示するために、列オブジェクト Map_obj からクライアント側へのイメージ Drawing のストリーム読み込みを行います。

例

次のプログラム環境の例が示されています。

- [PL/SQL \(DBMS_LOB パッケージ\) : テンポラリ LOB データの表示 \(10-48 ページ\)](#)
- [C \(OCI\) : テンポラリ LOB データの表示 \(10-49 ページ\)](#)
- [COBOL \(Pro*COBOL\) : テンポラリ LOB データの表示 \(10-52 ページ\)](#)

- C/C++ (Pro*C/C++) : テンポラリ LOB データの表示 (10-54 ページ)
- Visual Basic (OO4O) : 今回のリリースでは例は記載されていません。
- Java (JDBC) : 今回のリリースでは例は記載されていません。

PL/SQL (DBMS_LOB パッケージ) : テンポラリ LOB データの表示

/* 次のファンクションは、Washington_audio ファイルへのアクセス、テンポラリ LOB の作成、ファイルからのデータのロード、およびそのデータの読み戻しおよび表示を行います。*/

```
DECLARE
    Dest_loc      BLOB;
    Src_loc       BFILE := BFILENAME('AUDIO_DIR', 'Washington_audio');
    Amount        INTEGER := 128;
    Bbuf          RAW(128);
    Position      INTEGER := 1;
BEGIN
    DBMS_LOB.CREATETEMPORARY (Dest_loc, TRUE);
    /* FILE のオープンは必須です。*/
    DBMS_LOB.OPEN (Src_loc, DBMS_LOB.LOB_READONLY);
    /* LOB のオープンはオプションです。*/
    DBMS_LOB.OPEN (Dest_loc, DBMS_LOB.LOB_READWRITE);
    DBMS_LOB.LOADFROMFILE (Dest_loc, Src_loc, Amount);

    LOOP
        DBMS_LOB.READ (Dest_loc, Amount, Position, Bbuf);
        /* バッファの内容を表示します。*/
        DBMS_OUTPUT.PUT_LINE ('Result : ' || utl_raw.cast_to_varchar2 (Bbuf));
        Position := Position + Amount;
    END LOOP;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE ('End of data loaded into temp LOB');

    DBMS_LOB.CLOSE (Dest_loc);
    DBMS_LOB.FREETEMPORARY (Dest_loc);
    /* 後でファイルをクローズしない限り、ファイルをクローズする必要があります。*/
    DBMS_LOB.CLOSE (Src_loc);
END;
```

C (OCI) : テンポラリ LOB データの表示

/* 次のファンクションは、Washington_audio ファイルへのアクセス、テンポラリ LOB の作成、ファイルからのデータのロード、およびそのデータの読み戻しおよび表示を行います。読み込みは、ストリーミングの方法で行われます。このファンクションは、指定されたファイルがディレクトリ別名が "AUDIO_DIR" のディレクトリに保持されていると想定します。また、ファイルの長さが 14,000 バイト（読み込みおよびロード量）以上であるとも想定します。これらの量は、この例での任意の値です。このファンクションでは、fprintf() を使用してファイルの内容が表示されます。これは、テキスト・データに適していますが、バイナリ・データの方法に変更する必要がある場合があります。オーディオ・データに対しては、たとえば、オーディオ・ファンクションをコールできます。このファンクションは、成功した場合は 0（ゼロ）を返し、失敗した場合は -1 を返します。*/

```
#define MAXBUFLLEN 32767
sb4 display_file_to_lob( OCIError      *errhp,
                        OCISvcCtx     *svchp,
                        OCIStmt       *stmthp,
                        OCIEnv        *envhp)
{
    int rowind;
    char *binfile;
    OCILobLocator *tblob;
    OCILobLocator *bfile;

    ub4 amount = 14000;
    ub4 offset = 0;
    ub4 loblent = 0;
    ub4 amtp = 0;
    sword retval;
    ub4 piece = 1;
    ub4 remainder = 0;
    ub1 bufp[MAXBUFLLEN];
    sb4 return_code = 0;

    (void) printf("\n==> Testing loading files into lobes and displaying them\n\n");

    if(OCIDescriptorAlloc((dvoid *)envhp, (dvoid **) &tblob,
                          (ub4) OCI_DTYPE_LOB,
                          (size_t) 0, (dvoid **) 0))
    {
        printf("OCIDescriptor Alloc FAILED in print_length\n");
        return -1;
    }

    if(OCIDescriptorAlloc((dvoid *)envhp, (dvoid **) &bfile,
                          (ub4) OCI_DTYPE_FILE,
                          (size_t) 0, (dvoid **) 0))
```

```

{
    printf("OCIDescriptor Alloc FAILED in print_length\n");
    return -1;
}

/* テンポラリ LOB を作成します。*/
if(OCILobCreateTemporary(svchp, errhp, tblob, (ub2)0, SQLCS_IMPLICIT,
    OCI_TEMP_BLOB, OCI_ATTR_NOCACHE,
    OCI_DURATION_SESSION))
{
    (void) printf("FAILED: CreateTemporary() \n");
    return -1;
}

if(OCILobFileSetName(envhp, errhp, &bfile, (text*)"AUDIO_DIR",
    (ub2)strlen("AUDIO_DIR"), (text*)"Washington_audio",
    (ub2)strlen("Washington_audio")))
{
    printf("OCILobFileSetName FAILED\n");
    return_code = -1;
}

/* BFILE をオープンします。*/
if(OCILobFileOpen(svchp, errhp, (OCILobLocator *) bfile, OCI_FILE_READONLY))
{
    printf("OCILobFileOpen FAILED \n");
    return_code = -1;
}

if(OCILobLoadFromFile(svchp, errhp, tblob, (OCILobLocator*) bfile, (ub4) amount,
    (ub4)1, (ub4)1))
{
    printf("OCILobLoadFromFile FAILED\n");
    return_code = -1;
}

offset = 1;
memset(bufp, '\0', MAXBUFLen);

retval = OCILobRead(svchp, errhp, tblob, &amp;tp, offset,
    (dvoid *) bufp, (amount < MAXBUFLen ? amount : MAXBUFLen),
    (dvoid *) 0, (sb4) (*)(dvoid *, dvoid *, ub4, ub1) 0,
    (ub2) 0, (ub1) SQLCS_IMPLICIT);

printf("1st piece read from file is %s\n", bufp);

```

```

switch (retval)
{
  case OCI_SUCCESS:          /* 1つのピースのみです。*/
    (void) printf("stream read piece # %d \n", ++piece);
    (void) printf("piece read was %s\n",bufp);
    break;
  case OCI_ERROR:
    /* report_error(); ここでは示されない関数です。*/
    break;
  case OCI_NEED_DATA:       /* 2つ以上のピースがあります。*/
    remainder = amount;
    printf("remainder is %d \n",remainder);
    do
    {
      memset(bufp, '\0', MAXBUFLen);
      amtp = 0;
      remainder -= MAXBUFLen;
      printf("remainder is %d \n",remainder);
      retval = OCILobRead(svchp, errhp, tblob, &amtp, offset,
                          (dvoid *) bufp, (ub4) MAXBUFLen, (dvoid *) 0,
                          (sb4 *) (dvoid *, dvoid *, ub4, ub1) 0,
                          (ub2) 0, (ub1) SQLCS_IMPLICIT);

      /* 戻された読み量は、FIRST、NEXT ピースに対して未定義です。*/
      (void) fprintf(stderr,"stream read %d th piece, amtp = %d\n",
                    ++piece, amtp);
      (void) fprintf(stderr,"piece of length read was %d\n",
                    strlen((const char*)bufp));
      (void) printf("piece read was %s\n",bufp);
    } while (retval == OCI_NEED_DATA);
    break;
  default:
    (void) printf("Unexpected ERROR: OCILobRead() LOB.\n");
    break;
}

/* オーディオ・ファイルをクローズします。*/
if (OCILobFileClose(svchp, errhp, (OCILobLocator *) bfile))
{
  printf("OCILobFileClose FAILED\n");
  return_code = -1;
}
/* 使い終わったテンポラリ LOB をクリーン・アップします。*/
if (check_and_free_temp(tblob, errhp, svchp,stmthp, envhp))

```

```
{
    printf("check and free failed in load test\n");
    return_code = -1;
}
return return_code;
}
```

COBOL (Pro*COBOL) : テンポラリ LOB データの表示

```
IDENTIFICATION DIVISION.
PROGRAM-ID. ONE-READ-BLOB.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
01  USERID          PIC X(9)  VALUES "SAMP/SAMP".
01  TEMP-BLOB       SQL-BLOB.
01  SRC-BFILE       SQL-BFILE.
01  DIR-ALIAS       PIC X(30) VARYING.
01  FNAME           PIC X(20) VARYING.
01  BUFFER2         PIC X(32767) VARYING.
01  AMT             PIC S9(9)  COMP.
01  OFFSET          PIC S9(9)  COMP VALUE 1.
01  ORASLNRD        PIC 9(4).
01  ISTEMP          PIC S9(9)  COMP.

EXEC SQL INCLUDE SQLCA END-EXEC.
EXEC ORACLE OPTION (ORACA=YES) END-EXEC.
EXEC SQL INCLUDE ORACA END-EXEC.
EXEC SQL VAR BUFFER2 IS LONG RAW(32767) END-EXEC.
PROCEDURE DIVISION.
ONE-READ-BLOB.
EXEC SQL WHENEVER SQLERROR DO PERFORM SQL-ERROR END-EXEC.
EXEC SQL
CONNECT :USERID
END-EXEC.

* BLOB ロケータの割当ておよび初期化を行います。
EXEC SQL ALLOCATE :SRC-BFILE END-EXEC.
EXEC SQL ALLOCATE :TEMP-BLOB END-EXEC.
EXEC SQL LOB CREATE TEMPORARY :TEMP-BLOB END-EXEC.
EXEC SQL WHENEVER NOT FOUND GOTO END-OF-BLOB END-EXEC.

* ディレクトリおよびファイル情報を設定します。
MOVE "AUDIO_DIR" TO DIR-ALIAS-ARR.
```

```
MOVE 9 TO DIR-ALIAS-LEN.  
MOVE "Washington_audio" TO FNAME-ARR.  
MOVE 16 TO FNAME-LEN.
```

```
EXEC SQL  
  LOB FILE SET :SRC-BFILE DIRECTORY = :DIR-ALIAS,  
  FILENAME = :FNAME  
END-EXEC.
```

```
EXEC SQL  
  LOB DESCRIBE :SRC-BFILE GET LENGTH INTO :AMT  
END-EXEC.
```

* ソース BFILE および宛先テンポラリ BLOB をオープンします。

```
EXEC SQL LOB OPEN :SRC-BFILE READ ONLY END-EXEC.  
EXEC SQL LOB OPEN :TEMP-BLOB READ WRITE END-EXEC.
```

```
EXEC SQL  
  LOB LOAD :AMT FROM FILE :SRC-BFILE INTO :TEMP-BLOB  
END-EXEC.
```

* 単一の読み込みを行います。

```
EXEC SQL  
  LOB READ :AMT FROM :TEMP-BLOB INTO :BUFFER2  
END-EXEC.
```

```
DISPLAY "Read ", BUFFER2, " from TEMP-BLOB".
```

```
END-OF-BLOB.  
EXEC SQL LOB CLOSE :TEMP-BLOB END-EXEC.  
EXEC SQL LOB CLOSE :SRC-BFILE END-EXEC.  
EXEC SQL LOB FREE TEMPORARY :TEMP-BLOB END-EXEC.  
EXEC SQL FREE :TEMP-BLOB END-EXEC.  
EXEC SQL FREE :SRC-BFILE END-EXEC.  
STOP RUN.
```

```
SQL-ERROR.  
EXEC SQL  
  WHENEVER SQLERROR CONTINUE  
END-EXEC.  
MOVE ORASLNR TO ORASLNRD.  
DISPLAY " ".  
DISPLAY "ORACLE ERROR DETECTED ON LINE ", ORASLNRD, ":".  
DISPLAY " ".  
DISPLAY SQLERRMC.
```

```
EXEC SQL
    ROLLBACK WORK RELEASE
END-EXEC.
STOP RUN.
```

C/C++ (Pro*C/C++) : テンポラリ LOB データの表示

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

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

#define BufferLength 1024

void displayTempLOB_proc()
{
    OCIBlobLocator *Temp_loc;
    OCIFileLocator *Lob_loc;
    char *Dir = "PHOTO_DIR", *Name = "Lincoln_photo";
    int Amount;
    struct {
        unsigned short Length;
        char Data[BufferLength];
    } Buffer;
    int Position = 1;
    /* このデータ型では、データ型を等しくする必要があります。*/
    EXEC SQL VAR Buffer IS VARRAW(BufferLength);

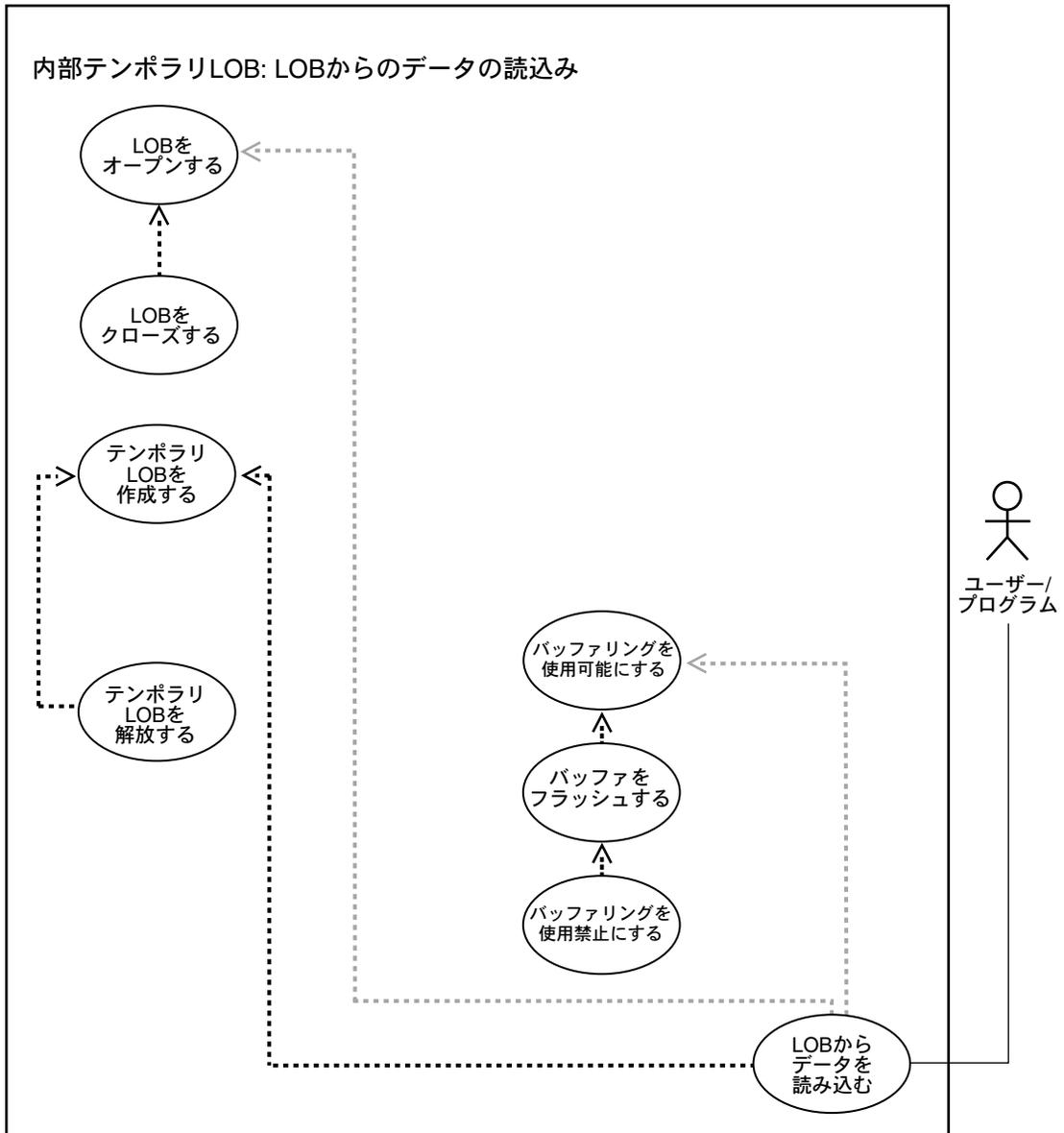
    EXEC SQL WHENEVER SQLERROR DO Sample_Error();
    /* LOB ロケータの割当ておよび初期化を行います。*/
    EXEC SQL ALLOCATE :Lob_loc;
    EXEC SQL ALLOCATE :Temp_loc;
    EXEC SQL LOB CREATE TEMPORARY :Temp_loc;
    EXEC SQL LOB FILE SET :Lob_loc DIRECTORY = :Dir, FILENAME = :Name;
    /* LOB のオープンはおプションです。*/
    EXEC SQL LOB OPEN :Lob_loc READ ONLY;
    EXEC SQL LOB OPEN :Temp_loc READ WRITE;
```

```
/* BFILEからテンポラリ LOBに、指定した量をロードします。*/
EXEC SQL LOB DESCRIBE :Lob_loc GET LENGTH INTO :Amount;
EXEC SQL LOB LOAD :Amount FROM FILE :Lob_loc AT :Position INTO :Temp_loc;
/* Amount = 0 (ゼロ) に設定すると、ポーリング方法が開始します。*/
Amount = 0;
/* バッファの最大サイズを設定します。*/
Buffer.Length = BufferLength;
EXEC SQL WHENEVER NOT FOUND DO break;
while (TRUE)
{
    /* バッファに1つのピースの BLOB を読み込みます。*/
    EXEC SQL LOB READ :Amount FROM :Temp_loc INTO :Buffer;
    printf("Display %d bytes\n", Buffer.Length);
}
printf("Display %d bytes\n", Amount);
/* LOB をオープンしている場合、その LOB をクローズする必要があります。*/
EXEC SQL LOB CLOSE :Lob_loc;
EXEC SQL LOB CLOSE :Temp_loc;
/* テンポラリ LOB を解放します。*/
EXEC SQL LOB FREE TEMPORARY :Temp_loc;
/* ロケータが保持しているリソースを解放します。*/
EXEC SQL FREE :Temp_loc;
}

void main()
{
    char *samp = "samp/samp";
    EXEC SQL CONNECT :samp;
    displayTempLOB_proc();
    EXEC SQL ROLLBACK WORK RELEASE;
}
```

テンポラリ LOB からのデータの読み込み

図 10-9 ユースケース図：テンポラリ LOB からのデータの読み込み



用途

テンポラリ LOB からデータを読み込みます。

使用上の注意

ストリーム読み込み

大量の LOB データを最も効率よく読み込む方法は、OCILOBRead() を使ったポーリングやコールバックによるストリーミングです。

LOB 値を読み込むとき、LOB の最後を超えて読み込んでもエラーにはなりません。開始オフセットと LOB のデータ量にかかわらず、通常 4GB の入力を指定できます。読み込む量を決定するために、OCILOBGetLength() コールでサーバーへのラウンドトリップを行い、LOB 値の長さを判断する必要はありません。

たとえば、LOB の長さが 5,000 バイトで、オフセット 1,000 から開始して LOB 値全体を読み込むとします。また、LOB 値の現在の長さがわからないとします。この場合の OCI 読み込みコールを示します（初期化パラメータは省略してあります）。

```
#define MAX_LOB_SIZE 4294967295
ub4 amount = MAX_LOB_SIZE;
ub4 offset = 1000;
OCILOBRead(svchp, errhp, locp, &amount, offset, bufp, buf1, 0, 0, 0, 0)
```

ポーリング・モードを使用するときは、バッファが一杯にならない場合があるため、各 OCILOBRead() コール後に量パラメータの値を調べて、バッファに何バイト読み込まれたかを確認してください。

コールバックを使用する場合、コールバックへの入力である len パラメータにバッファに格納されたバイト数が示されます。バッファ全体にデータが格納されていない場合があるため、コールバック処理中に len パラメータを確認してください（『Oracle8i コール・インタフェース・プログラマーズ・ガイド』を参照）。

構文

各プログラム環境で使用可能な機能のリストは、第 3 章「LOB プログラム環境」を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- PL/SQL (DBMS_LOB) : 『Oracle8i PL/SQL パッケージ・プロシージャ・リファレンス』の第 18 章「DBMS_LOB」の「サブプログラムの要約」の READ プロシージャ

- C (OCI) : 『Oracle8i コール・インタフェース・プログラマーズ・ガイド』の第 15 章「OCI リレーショナル関数」の「LOB 関数」の OCILobRead()
- COBOL (Pro*COBOL) : 『Oracle8i Pro*COBOL プリコンパイラ・プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB READ (実行可能埋込み SQL 拡張要素)」
- C/C++ (Pro*C/C++) : 『Oracle8i Pro*C/C++ プリコンパイラ・プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB READ (実行可能埋込み SQL 拡張要素)」
- Visual Basic (OO4O) : 参照マニュアルはありません。
- Java (JDBC) : 参照マニュアルはありません。

使用例

この例では、データを 1 つのビデオ・フレームから読み込みます。

例

次のプログラム環境の例が示されています。

- [PL/SQL \(DBMS_LOB パッケージ\) : テンポラリ LOB からのデータの読み込み](#) (10-58 ページ)
- [C \(OCI\) : テンポラリ LOB からのデータの読み込み](#) (10-59 ページ)
- [COBOL \(Pro*COBOL\) : テンポラリ LOB からのデータの読み込み](#) (10-62 ページ)
- [C/C++ \(Pro*C/C++\) : テンポラリ LOB からのデータの読み込み](#) (10-64 ページ)
- Visual Basic (OO4O) : 今回のリリースでは例は記載されていません。
- Java (JDBC) : 今回のリリースでは例は記載されていません。

PL/SQL (DBMS_LOB パッケージ) : テンポラリ LOB からのデータの読み込み

/* PL/SQL はストリーミング読み込みをサポートしないことに注意してください。oci の例では、ストリーミング読み込みを示します。*/

```
DECLARE
  Dest_loc      BLOB;
  Src_loc       BFILE := BFILENAME('AUDIO_DIR', 'Washington_audio');
  Amount        INTEGER := 4000;
  Bbuf          RAW(32767);
  Position      INTEGER :=1;
BEGIN
  DBMS_LOB.CREATETEMPORARY(Dest_loc, TRUE);
  /* FILE のオープンは必須です。*/
  DBMS_LOB.OPEN(Src_loc, DBMS_LOB.LOB_READONLY);
```

```

/* LOB のオープンはオプションです。 */
DBMS_LOB.LOADFROMFILE(Dest_loc, Src_loc, Amount);
DBMS_LOB.READ (Dest_loc, Amount, Position, Ebuf);
/* LOB をオープンしている場合、その LOB をクローズする必要があります。 */
DBMS_LOB.CLOSE(Src_loc);

```

C (OCI) : テンポラリ LOB からのデータの読み

/* これは、テンポラリ LOB からのデータの読みおよび表示に示したものと同一例です。このファンクションは Washinton_audio ファイルを取り、そのファイルを入力としての BFILE としてオープンし、テンポラリ LOB にそのファイル・データをロードし、さらにその後テンポラリ LOB から一度に 500 バイト以下のデータを読み込みます。5000 バイトは、この例で任意に選択されたバッファの最大長です。このファンクションは、成功した場合は 0 (ゼロ) を戻し、失敗した場合は -1 を戻します。 */

```

#define MAXBUFLLEN 32767

sb4 test_file_to_lob (OCILobLocator *lob_loc,
                    OCIError      *errhp,
                    OCISvcCtx     *svchp,
                    OCISstmt     *stmthp,
                    OCIEnv       *envhp)
{
    int rowind;
    OCILobLocator *bfile;

    ub4 amount = 14000;
    ub4 offset = 0;
    ub4 loblen = 0;
    ub4 amtp = 0;
    sword retval;
    ub4 piece = 1;
    ub4 remainder=0;
    ub1 bufp[MAXBUFLLEN];

    (void) printf(
        "\n====> Testing loading files into lobes and displaying them\n\n");

    if (OCIDescriptorAlloc((dvoid **) &bfile,
                          (ub4)OCI_DTYPE_LOB, (size_t)0,
                          (dvoid**)0))

/* テンポラリ LOB を作成します。 */

```

```

if(OCILobCreateTemporary(svchp, errhp, lob_loc, (ub2)0, SQLCS_IMPLICIT,
                        OCI_TEMP_BLOB, OCI_ATTR_NOCACHE,
                        OCI_DURATION_SESSION))
{
    (void) printf("FAILED: CreateTemporary() \n");
    return -1;
}
if(OCILobFileSetName(envhp, errhp, &bfile, (text*)"AUDIO_DIR",
                    (ub2)strlen("AUDIO_DIR"),
                    (text*)"Washington_audio",
                    (ub2)strlen("Washington_audio")))
{
    printf("OCILobFileSetName FAILED\n");
    return -1;
}
if (OCILobFileOpen(svchp, errhp, (OCILobLocator *) bfile, OCI_FILE_READONLY))
{
    printf("OCILobFileOpen FAILED \n");
    return -1;
}
if(OCILobLoadFromFile(svchp, errhp, lob_loc, (OCILobLocator*)bfile, (ub4)amount,
                    (ub4)1, (ub4)1))
{
    printf("OCILobLoadFromFile FAILED\n");
    return -1;
}

offset = 1;
memset(bufp, '\0', MAXBUFLen);

retval = OCILobRead(svchp, errhp, lob_loc, &amtp, offset, (dvoid *) bufp,
                    (amount < MAXBUFLen ? amount : MAXBUFLen), (dvoid *)0,
                    (sb4 *) (dvoid *, dvoid *, ub4, ub1)) 0,
                    (ub2) 0, (ub1) SQLCS_IMPLICIT);
fprintf(stderr, "1st piece read from file is %s\n", bufp);

switch (retval)
{
case OCI_SUCCESS:
    /* 1つのピースのみです。*/
    (void) printf("stream read piece # %d \n", ++piece);
    (void) printf("piece read was %s\n", bufp);
    break;
case OCI_ERROR:
    /* report_error(); ここでは示されない関数です。*/
    break;
}

```

```

case OCI_NEED_DATA:          /* 2 つ以上のピースがあります。*/
    remainder = amount;
    fprintf(stderr, "remainder is %d \n", remainder);
    do
    {
        memset(bufp, '\0', MAXBUFLEN);
        amtp = 0;
        remainder -= MAXBUFLEN;
        fprintf(stderr, "remainder is %d \n", remainder);

        retval = OCILobRead(svchp, errhp, lob_loc, &amtp, offset,
                            (dvoid *) bufp, (ub4) MAXBUFLEN, (dvoid *) 0,
                            (sb4 *) (dvoid *, dvoid *, ub4, ub1)) 0,
                            (ub2) 0, (ub1) SQLCS_IMPLICIT);

        /* 戻された読み量は、FIRST、NEXT ピースに対して未定義です。*/
        (void) fprintf(stderr, "stream read %d th piece, amtp = %d\n",
                        ++piece, amtp);
        (void) fprintf(stderr,
                        "piece of length read was %d\n", strlen((const char *)bufp));
        (void) fprintf(stderr, "piece read was %s\n", bufp);
    } while (retval == OCI_NEED_DATA);
    break;
default:
    (void) printf("Unexpected ERROR: OCILobRead() LOB.\n");
    break;
}

/* オーディオ・ファイルをクローズします。*/
if (OCILobFileClose(svchp, errhp, (OCILobLocator *) bfile))
{
    printf("OCILobFileClose FAILED\n");
    return -1;
}
if (OCIDescriptorFree ((dvoid*) lob_loc, (ub4) OCI_DTYPE_LOB))
{
    printf("failed in OCIDescriptor Free\n");
    return -1;
}

/* 使い終わったテンポラリ LOB をクリーン・アップします。*/
if (check_and_free_temp(lob_loc, errhp, svchp, stmthp, envhp))
{
    printf("check and free failed in load test\n");
    return -1;
}

```

```
    }
    return 0;
}

sb4 check_and_free_temp(OCILOBLocator *tblob,
                       OCIError      *errhp,
                       OCISvcCtx     *svchp,
                       OCISstmt     *stnthp,
                       OCIEnv       *envhp)
{
    boolean is_temp;
    is_temp = FALSE;
    if (OCILOBIsTemporary (envhp,errhp, tblob, &is_temp))
    {
        printf ("FAILED: OciLobIsTemporary call \n");
    }
    if(is_temp)
    {
        if (OCILOBFreeTemporary (svchp, errhp,tblob))
        {
            printf ("FAILED: OCILOBFreeTemporary call \n");
            return -1;
        } else
        {
            printf ("Temporary LOB freed\n");
        }
    }
}else
{
    printf ("locator is not a temporary LOB locator\n");
}
return 0;
}
```

COBOL (Pro*COBOL) : テンポラリ LOB からのデータの読み込み

```
IDENTIFICATION DIVISION.
PROGRAM-ID. ONE-READ-BLOB.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
01  USERID          PIC X(9)  VALUES "SAMP/SAMP".
01  TEMP-BLOB      SQL-BLOB.
01  SRC-BFILE      SQL-BFILE.
01  DIR-ALIAS      PIC X(30)  VARYING.
```

```

01 FNAME          PIC X(20) VARYING.
01 BUFFER2       PIC X(32767) VARYING.
01 AMT           PIC S9(9) COMP.
01 OFFSET        PIC S9(9) COMP VALUE 1.
01 ORASLNRD      PIC 9(4).
01 ISTEMP        PIC S9(9) COMP.

EXEC SQL INCLUDE SQLCA END-EXEC.
EXEC ORACLE OPTION (ORACA=YES) END-EXEC.
EXEC SQL INCLUDE ORACA END-EXEC.
EXEC SQL VAR BUFFER2 IS LONG RAW(32767) END-EXEC.
PROCEDURE DIVISION.
ONE-READ-BLOB.
EXEC SQL WHENEVER SQLERROR DO PERFORM SQL-ERROR END-EXEC.
EXEC SQL
CONNECT :USERID
END-EXEC.

* BLOB ロケータの割当ておよび初期化を行います。
EXEC SQL ALLOCATE :SRC-BFILE END-EXEC.
EXEC SQL ALLOCATE :TEMP-BLOB END-EXEC.
EXEC SQL LOB CREATE TEMPORARY :TEMP-BLOB END-EXEC.
EXEC SQL WHENEVER NOT FOUND GOTO END-OF-BLOB END-EXEC.

* ディレクトリおよびファイル情報を設定します。
MOVE "AUDIO_DIR" TO DIR-ALIAS-ARR.
MOVE 9 TO DIR-ALIAS-LEN.
MOVE "Washington_audio" TO FNAME-ARR.
MOVE 16 TO FNAME-LEN.

EXEC SQL
LOB FILE SET :SRC-BFILE DIRECTORY = :DIR-ALIAS,
FILENAME = :FNAME
END-EXEC.

EXEC SQL
LOB DESCRIBE :SRC-BFILE GET LENGTH INTO :AMT
END-EXEC.

* ソース BFILE および宛先テンポラリ BLOB をオープンします。
EXEC SQL LOB OPEN :SRC-BFILE READ ONLY END-EXEC.
EXEC SQL LOB OPEN :TEMP-BLOB READ WRITE END-EXEC.

EXEC SQL
LOB LOAD :AMT FROM FILE :SRC-BFILE INTO :TEMP-BLOB

```

```
END-EXEC.
```

* 単一の読み込みを行います。

```
EXEC SQL
    LOB READ :AMT FROM :TEMP-BLOB INTO :BUFFER2
END-EXEC.
```

```
DISPLAY "Read ", BUFFER2, " from TEMP-BLOB".
```

```
END-OF-BLOB.
```

```
EXEC SQL LOB CLOSE :TEMP-BLOB END-EXEC.
EXEC SQL LOB CLOSE :SRC-BFILE END-EXEC.
EXEC SQL LOB FREE TEMPORARY :TEMP-BLOB END-EXEC.
EXEC SQL FREE :TEMP-BLOB END-EXEC.
EXEC SQL FREE :SRC-BFILE END-EXEC.
STOP RUN.
```

```
SQL-ERROR.
```

```
EXEC SQL
    WHENEVER SQLERROR CONTINUE
END-EXEC.
MOVE ORASLNR TO ORASLNRD.
DISPLAY " ".
DISPLAY "ORACLE ERROR DETECTED ON LINE ", ORASLNRD, ":".
DISPLAY " ".
DISPLAY SQLERRMC.
EXEC SQL
    ROLLBACK WORK RELEASE
END-EXEC.
STOP RUN.
```

C/C++ (Pro*C/C++) : テンポラリ LOB からのデータの読み込み

```
/* テンポラリ LOB からデータを読み込みます。 */
#include <oci.h>
#include <stdio.h>
#include <sqlca.h>

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

```
    exit(1);
}

#define BufferLength 1024

void readTempLOB_proc()
{
    OCIBlobLocator *Temp_loc;
    OCIBFileLocator *Lob_loc;
    char *Dir = "AUDIO_DIR", *Name = "Washington_audio";
    int Length, Amount;
    struct {
        unsigned short Length;
        char Data[BufferLength];
    } Buffer;

    /* このデータ型では、データ型を等しくする必要があります。*/
    EXEC SQL VAR Buffer IS VARRAW(BufferLength);
    EXEC SQL WHENEVER SQLERROR DO Sample_Error();

    /* BFILE ロケータの割当ておよび初期化を行います。*/
    EXEC SQL ALLOCATE :Lob_loc;
    EXEC SQL LOB FILE SET :Lob_loc DIRECTORY = :Dir, FILENAME = :Name;

    /* BFILE の長さを決定します。*/
    EXEC SQL LOB DESCRIBE :Lob_loc GET LENGTH INTO :Length;

    /* テンポラリ LOB の割当ておよび作成を行います。*/
    EXEC SQL ALLOCATE :Temp_loc;
    EXEC SQL LOB CREATE TEMPORARY :Temp_loc;

    /* 読み込み用の BFILE をオープンします。*/
    EXEC SQL LOB OPEN :Lob_loc READ ONLY;

    /* テンポラリ LOB に BFILE をロードします。*/
    Amount = Length;
    EXEC SQL LOB LOAD :Amount FROM FILE :Lob_loc INTO :Temp_loc;

    /* BFILE をクローズします。*/
    EXEC SQL LOB CLOSE :Lob_loc;
    Buffer.Length = BufferLength;
    EXEC SQL WHENEVER NOT FOUND DO break;
    while (TRUE)
    {
        /* バッファに 1 つのピースのテンポラリ LOB を読み込みます。*/
```

```
        EXEC SQL LOB READ :Amount FROM :Temp_loc INTO :Buffer;
        printf("Read %d bytes\n", Buffer.Length);
    }
    printf("Read %d bytes\n", Amount);

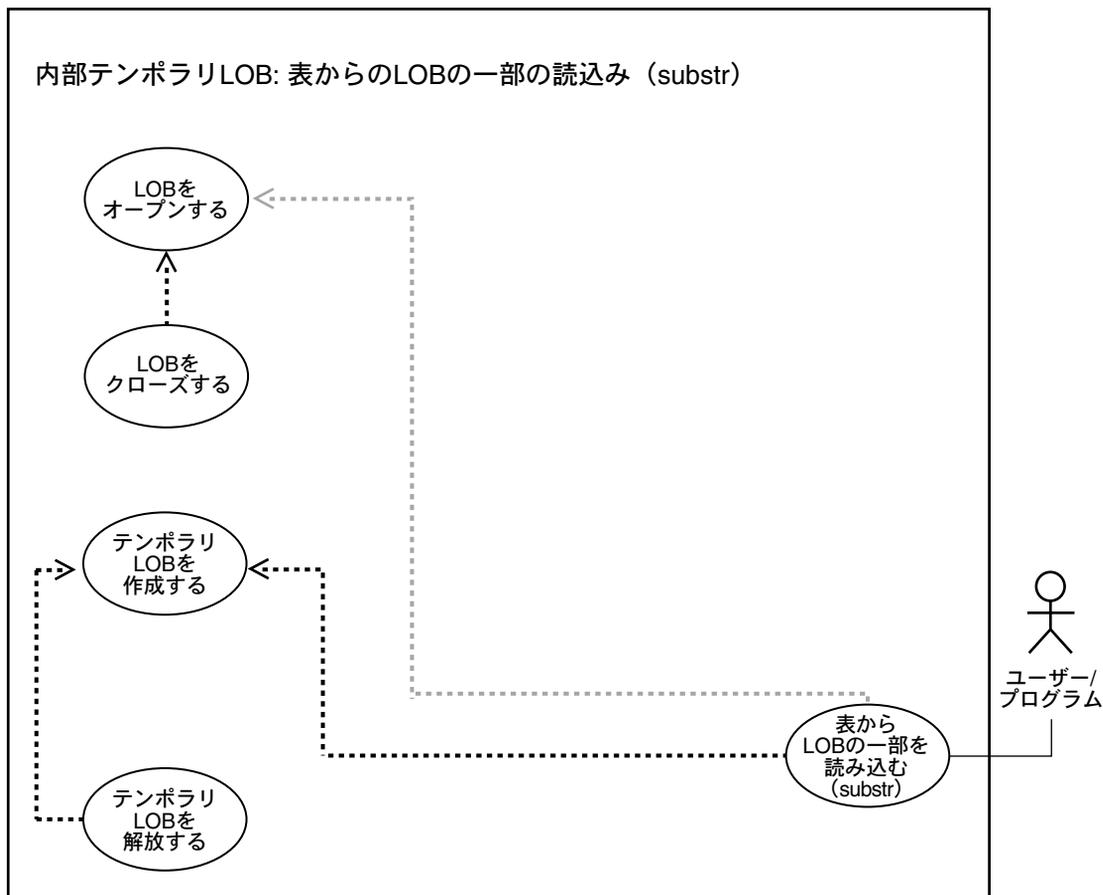
    /* テンポラリ LOB を解放します。*/
    EXEC SQL LOB FREE TEMPORARY :Temp_loc;

    /* ロケータが保持しているリソースを解放します。*/
    EXEC SQL FREE :Temp_loc;
    EXEC SQL FREE :Lob_loc;
}

void main()
{
    char *samp = "samp/samp";
    EXEC SQL CONNECT :samp;
    readTempLOB_proc();
    EXEC SQL ROLLBACK WORK RELEASE;
}
```

テンポラリ LOB の一部の読み込み (substr)

図 10-10 ユースケース図：表からのテンポラリ LOB の一部の読み込み (substr)



参照： 内部テンポラリ LOB に関するすべての基本操作については、10-2 ページの「ユースケース・モデル：内部テンポラリ LOB」を参照してください。

用途

テンポラリ LOB の一部を読み込みます (substr)。

使用上の注意

ありません。

構文

各プログラム環境で使用可能な機能のリストは、第 3 章「LOB プログラム環境」を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- PL/SQL (DBMS_LOB) : 『Oracle8i PL/SQL パッケージ・プロシージャ・リファレンス』の第 18 章「DBMS_LOB」の「サブプログラムの要約」の SUBSTR ファンクション
- C (OCI) : 参照マニュアルはありません。
- COBOL (Pro*COBOL) : 『Oracle8i Pro*COBOL プリコンパイラ・プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB READ (実行可能埋込み SQL 拡張要素)」
- C/C++ (Pro*C/C++) : 『Oracle8i Pro*C/C++ プリコンパイラ・プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB LOAD (実行可能埋込み SQL 拡張要素)」、および『Oracle8i PL/SQL パッケージ・プロシージャ・リファレンス』の第 18 章「DBMS_LOB」の「サブプログラムの要約」の SUBSTR ファンクション
- Visual Basic (OO4O) : 参照マニュアルはありません。
- Java (JDBC) : 参照マニュアルはありません。

使用例

この例では、音響効果 sound から一部を読み込むための操作を示します。

例

次のプログラム環境の例が示されています。

- PL/SQL (DBMS_LOB パッケージ) : テンポラリ LOB の一部の読み込み (substr) (10-68 ページ)
- C (OCI) : 今回のリリースでは例は記載されていません。
- COBOL (Pro*COBOL) : テンポラリ LOB の一部の読み込み (substr) (10-69 ページ)
- C/C++ (Pro*C/C++) : テンポラリ LOB の一部の読み込み (substr) (10-71 ページ)

- Visual Basic (OO4O) : 今回のリリースでは例は記載されていません。
- Java (JDBC) : 今回のリリースでは例は記載されていません。

PL/SQL (DBMS_LOB パッケージ) : テンポラリ LOB の一部の読み込み (substr)

```

/* 例のプロシージャ substrTempLOB_proc は、DBMS_LOB パッケージの
一部ではないことに注意してください。 */
/* この例では、AUDIO_DIR 別名を持つディレクトリに、ユーザーが 'Washington_audio'
ファイルを持っていると想定します。 */
CREATE or REPLACE PROCEDURE substrTempLOB_proc IS
  Dest_loc      BLOB;
  Src_loc       BFILE := BFILENAME('AUDIO_DIR', 'Washington_audio');
  Amount        INTEGER := 32767;
  Bbuf          RAW(32767);
  Position      INTEGER :=128;
BEGIN
  DBMS_LOB.CREATETEMPORARY(Dest_loc,TRUE);
  /* FILE のオープンは必須です。 */
  DBMS_LOB.OPEN(Src_loc, DBMS_LOB.LOB_READONLY);
  /* LOB のオープンはオプションです。 */
  DBMS_LOB.OPEN(Dest_loc,DBMS_LOB.LOB_READWRITE);
  DBMS_LOB.LOADFROMFILE(Dest_loc, Src_loc, Amount);
  Bbuf := DBMS_LOB.SUBSTR(Dest_loc, Amount, Position);
  /* LOB をオープンしている場合、その LOB をクローズする必要があります。 */
  DBMS_LOB.CLOSE(Src_loc);
  DBMS_LOB.CLOSE(Dest_loc);
END;
```

COBOL (Pro*COBOL) : テンポラリ LOB の一部の読み込み (substr)

```

IDENTIFICATION DIVISION.
PROGRAM-ID. ONE-READ-BLOB.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

01  USERID          PIC X(9)  VALUES "SAMP/SAMP".
01  TEMP-BLOB       SQL-BLOB.
01  SRC-BFILE       SQL-BFILE.
01  DIR-ALIAS       PIC X(30) VARYING.
01  FNAME           PIC X(20) VARYING.
01  BUFFER2         PIC X(32767) VARYING.
01  AMT             PIC S9(9)  COMP.
```

```
01 OFFSET          PIC S9(9) COMP VALUE 1.
01 ORASLNDR        PIC 9(4).
01 ISTEMP          PIC S9(9) COMP.

EXEC SQL INCLUDE SQLCA END-EXEC.
EXEC ORACLE OPTION (ORACA=YES) END-EXEC.
EXEC SQL INCLUDE ORACA END-EXEC.
EXEC SQL VAR BUFFER2 IS LONG RAW(32767) END-EXEC.
```

```
PROCEDURE DIVISION.
ONE-READ-BLOB.
```

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

- * BLOB ロケータの割当ておよび初期化を行います。

```
EXEC SQL ALLOCATE :SRC-BFILE END-EXEC.
EXEC SQL ALLOCATE :TEMP-BLOB END-EXEC.
EXEC SQL LOB CREATE TEMPORARY :TEMP-BLOB END-EXEC.
EXEC SQL WHENEVER NOT FOUND GOTO END-OF-BLOB END-EXEC.
```

- * ディレクトリおよびファイル情報を設定します。

```
MOVE "AUDIO_DIR" TO DIR-ALIAS-ARR.
MOVE 9 TO DIR-ALIAS-LEN.
MOVE "Washington_audio" TO FNAME-ARR.
MOVE 16 TO FNAME-LEN.

EXEC SQL
LOB FILE SET :SRC-BFILE DIRECTORY = :DIR-ALIAS,
FILENAME = :FNAME
END-EXEC.
```

```
EXEC SQL
LOB DESCRIBE :SRC-BFILE GET LENGTH INTO :AMT
END-EXEC.
```

- * ソース BFILE および宛先テンポラリ BLOB をオープンします。

```
EXEC SQL LOB OPEN :SRC-BFILE READ ONLY END-EXEC.
EXEC SQL LOB OPEN :TEMP-BLOB READ WRITE END-EXEC.

EXEC SQL
LOB LOAD :AMT FROM FILE :SRC-BFILE INTO :TEMP-BLOB
END-EXEC.
```

* 単一の読み込みを行います。

```
EXEC SQL
  LOB READ :AMT FROM :TEMP-BLOB INTO :BUFFER2
END-EXEC.
```

```
DISPLAY "Read ", BUFFER2, " from TEMP-BLOB".
```

```
END-OF-BLOB.
EXEC SQL LOB CLOSE :TEMP-BLOB END-EXEC.
EXEC SQL LOB CLOSE :SRC-BFILE END-EXEC.
EXEC SQL LOB FREE TEMPORARY :TEMP-BLOB END-EXEC.
EXEC SQL FREE :TEMP-BLOB END-EXEC.
EXEC SQL FREE :SRC-BFILE END-EXEC.
STOP RUN.
```

```
SQL-ERROR.
EXEC SQL
  WHENEVER SQLERROR CONTINUE
END-EXEC.
MOVE ORASLNR TO ORASLNRD.
DISPLAY " ".
DISPLAY "ORACLE ERROR DETECTED ON LINE ", ORASLNRD, ":".
DISPLAY " ".
DISPLAY SQLERRMC.
EXEC SQL
  ROLLBACK WORK RELEASE
END-EXEC.
STOP RUN.
```

C/C++ (Pro*C/C++) : テンポラリ LOB の一部の読み込み (substr)

/* Pro*C/C++ には、DEMS_LOB.SUBSTR() ファンクションに対する同等の埋込み SQL フォームがありません。ただし、Pro*C/C++ は、この例に示されているように、Pro*C/C++ プログラムに埋め込まれている無名 PL/SQL ブロックを使用して、PL/SQL と相互作用できます。*/

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

void Sample_Error()
{
  EXEC SQL WHENEVER SQLERROR CONTINUE;
```

```

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

#define BufferLength 4096

void substringTempLOB_proc()
{
    OCIBlobLocator *Temp_loc;
    OCIBFileLocator *Lob_loc;
    char *Dir = "AUDIO_DIR", *Name = "Washington_audio";
    int Position = 1024;
    unsigned int Length;
    int Amount = BufferLength;
    struct {
        unsigned short Length;
        char Data[BufferLength];
    } Buffer;
    /* このデータ型では、データ型を等しくする必要があります。 */
    EXEC SQL VAR Buffer IS VARRAW(BufferLength);

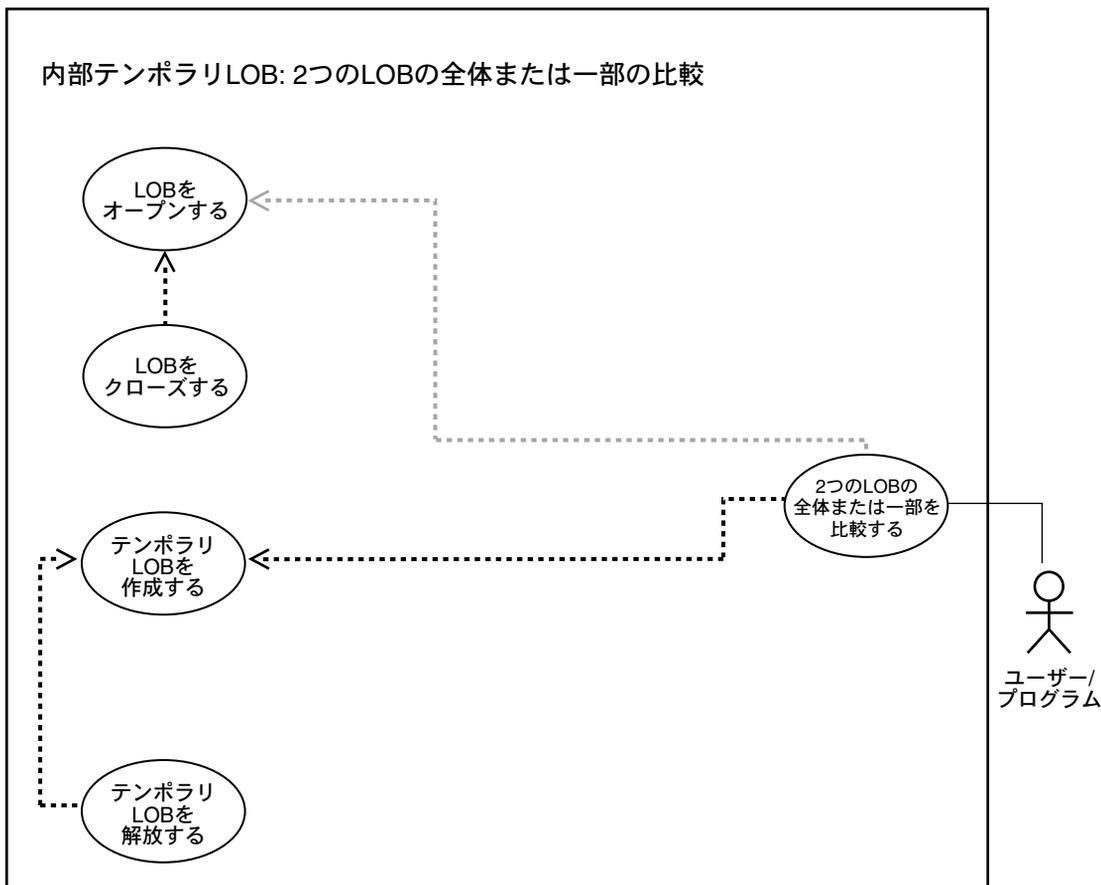
    EXEC SQL WHENEVER SQLERROR DO Sample_Error();
    /* テンポラリ LOB の割当ておよび作成を行います。 */
    EXEC SQL ALLOCATE :Temp_loc;
    EXEC SQL LOB CREATE TEMPORARY :Temp_loc;
    /* BFILE ロケータの割当ておよび初期化を行います。 */
    EXEC SQL ALLOCATE :Lob_loc;
    EXEC SQL LOB FILE SET :Lob_loc DIRECTORY = :Dir, FILENAME = :Name;
    /* LOB をオープンします。 */
    EXEC SQL LOB OPEN :Lob_loc READ ONLY;
    EXEC SQL LOB OPEN :Temp_loc READ WRITE;
    /* BFILE の長さを決定し、それをテンポラリ LOB にロードします。 */
    EXEC SQL LOB DESCRIBE :Lob_loc GET LENGTH INTO :Length;
    EXEC SQL LOB LOAD :Length FROM FILE :Lob_loc INTO :Temp_loc;
    /* PL/SQL ブロック内で、テンポラリ LOB に SUBSTR() を起動します。 */
    EXEC SQL EXECUTE
        BEGIN
            :Buffer := DBMS_LOB.SUBSTR(:Temp_loc, :Amount, :Position);
        END;
    END-EXEC;
    /* バッファ内のデータを処理します。 */
    /* LOB をオープンしている場合、その LOB をクローズする必要があります。 */
    EXEC SQL LOB CLOSE :Lob_loc;
    EXEC SQL LOB CLOSE :Temp_loc;

```

```
/* テンポラリ LOB を解放します。*/  
EXEC SQL LOB FREE TEMPORARY :Temp_loc;  
/* ロケータが使用しているリソースを解放します。*/  
EXEC SQL FREE :Lob_loc;  
EXEC SQL FREE :Temp_loc;  
}  
  
void main()  
{  
    char *samp = "samp/samp";  
    EXEC SQL CONNECT :samp;  
    substringTempLOB_proc();  
    EXEC SQL ROLLBACK WORK RELEASE;  
}
```

2つの（テンポラリ）LOBの全体または一部の比較

図 10-11 ユースケース図：2つのテンポラリLOBの全体または一部の比較



参照： 内部テンポラリLOBに関するすべての基本操作については、10-2ページの「ユースケース・モデル：内部テンポラリLOB」を参照してください。

用途

2つのテンポラリLOBの全体または一部を比較します。

使用上の注意

ありません。

構文

各プログラム環境で使用可能な機能のリストは、第3章「LOBプログラム環境」を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- PL/SQL (DBMS_LOB) : 『Oracle8i PL/SQL パッケージ・プロシージャ・リファレンス』の第18章「DBMS_LOB」の「サブプログラムの要約」の COMPARE ファンクション
- C (OCI) : 参照マニュアルはありません。
- COBOL (Pro*COBOL) : 『Oracle8i Pro*COBOL プリコンパイラ・プログラマーズ・ガイド』の付録F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB COPY (実行可能埋込み SQL 拡張要素)」、および『Oracle8i PL/SQL パッケージ・プロシージャ・リファレンス』の第18章「DBMS_LOB」の「サブプログラムの要約」の COMPARE ファンクション
- C/C++ (Pro*C/C++) : 『Oracle8i Pro*C/C++ プリコンパイラ・プログラマーズ・ガイド』の付録F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB COPY (実行可能埋込み SQL 拡張要素)」、および『Oracle8i PL/SQL パッケージ・プロシージャ・リファレンス』の第18章「DBMS_LOB」の「サブプログラムの要約」の COMPARE ファンクション
- Visual Basic (OO4O) : 参照マニュアルはありません。
- Java (JDBC) : 参照マニュアルはありません。

使用例

次の例は、アーカイブ表 VideoframesLib_tab の2つのフレームを比較して、異なっているかどうかを確認します。比較の結果に応じて、フレームを Multimedia_tab に挿入します。

例

次のプログラム環境の例が示されています。

- PL/SQL (DBMS_LOB パッケージ) : 2つの（テンポラリ）LOBの全体または一部の比較 (10-75 ページ)
- C (OCI) : 今回のリリースでは例は記載されていません。
- COBOL (Pro*COBOL) : 2つの（テンポラリ）LOBの全体または一部の比較 (10-77 ページ)

- C/C++ (Pro*C/C++) : 2つの（テンポラリ）LOBの全体または一部の比較（10-79 ページ）
- Visual Basic (OO4O) : 今回のリリースでは例は記載されていません。
- Java (JDBC) : 今回のリリースでは例は記載されていません。

PL/SQL (DBMS_LOB パッケージ) : 2つの（テンポラリ）LOBの全体または一部の比較

/* 例のプロシージャ compareTwoTemporPersistLOBs_proc は、DBMS_LOB パッケージの一部ではないことに注意してください。*/

```
CREATE OR REPLACE PROCEDURE compareTwoTmpPerLOBs_proc IS
  Lob_loc1 BLOB;
  Lob_loc2 BLOB;
  Temp_loc BLOB;
  Amount   INTEGER := 32767;
  Retval   INTEGER;
BEGIN
  /* LOB を選択します。*/
  SELECT Frame INTO Lob_loc1 FROM Multimedia_tab
     WHERE Clip_ID = 1;
  SELECT Frame INTO Lob_loc2 FROM Multimedia_tab
     WHERE Clip_ID = 2;
  /* フレームを比較する前に、テンポラリ LOB にフレームをコピーし、別の形式に変換します。*/
  DBMS_LOB.CREATETEMPORARY(Temp_loc, TRUE);
  DBMS_LOB.OPEN(Temp_loc, DBMS_LOB.LOB_READWRITE);
  DBMS_LOB.OPEN(Lob_loc1, DBMS_LOB.LOB_READONLY);
  DBMS_LOB.OPEN(Lob_loc2, DBMS_LOB.LOB_READONLY);
  /* テンポラリ LOB に永続 LOB をコピーします。*/
  DBMS_LOB.COPY(Temp_loc, Lob_loc2, DBMS_LOB.GETLENGTH(Lob_loc2), 1, 1);
  /* テンポラリ LOB を比較する前に、テンポラリ LOB に次のような変換ファンクションを
     実行します。*/
  /* ...some_conversion_format_function(Temp_loc); */
  retval := DBMS_LOB.COMPARE(Lob_loc1, Temp_loc, Amount, 1, 1);
  IF retval = 0 THEN
    DBMS_OUTPUT.PUT_LINE('Processing for equal frames');
  ELSE
    DBMS_OUTPUT.PUT_LINE('Processing for non-equal frames');
  END IF;
  DBMS_LOB.CLOSE(Temp_loc);
  DBMS_LOB.CLOSE(Lob_loc1);
  DBMS_LOB.CLOSE(Lob_loc2);
  /* 使い終わったテンポラリ LOB を解放します。*/
  DBMS_LOB.FREETEMPORARY(Temp_loc);
END;
```

COBOL（Pro*COBOL）：2つの（テンポラリ）LOBの全体または一部の比較

```

IDENTIFICATION DIVISION.
PROGRAM-ID. BLOB-COMPARE.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

01 USERID    PIC X(11) VALUES "SAMP/SAMP".
01 BLOB1     SQL-BLOB.
01 BLOB2     SQL-BLOB.
01 TEMP-BLOB SQL-BLOB.
01 RET       PIC S9(9) COMP.
01 AMT       PIC S9(9) COMP VALUE 5.
01 ORASLNRD  PIC 9(4).

EXEC SQL INCLUDE SQLCA END-EXEC.
EXEC ORACLE OPTION (ORACA=YES) END-EXEC.
EXEC SQL INCLUDE ORACA END-EXEC.

PROCEDURE DIVISION.
BLOB-COMPARE.

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

* BLOB ロケータの割当ておよび初期化を行います。
EXEC SQL ALLOCATE :BLOB1 END-EXEC.
EXEC SQL ALLOCATE :BLOB2 END-EXEC.
EXEC SQL WHENEVER NOT FOUND GOTO END-OF-BLOB END-EXEC.
EXEC SQL
    SELECT FRAME INTO :BLOB1
    FROM MULTIMEDIA_TAB M WHERE M.CLIP_ID = 1
END-EXEC.

EXEC SQL
    SELECT FRAME INTO :BLOB2
    FROM MULTIMEDIA_TAB M WHERE M.CLIP_ID = 2
END-EXEC.

* テンポラリLOBの割当ておよび作成を行います。
EXEC SQL ALLOCATE :TEMP-BLOB END-EXEC.
EXEC SQL
    LOB CREATE TEMPORARY :TEMP-BLOB

```

2つの（テンポラリ）LOBの全体または一部の比較

```
END-EXEC.

* BLOB を READ ONLY でオープンし、テンポラリ LOB を READ/WRITE でオープンします。
EXEC SQL LOB OPEN :BLOB1 READ ONLY END-EXEC.
EXEC SQL LOB OPEN :BLOB2 READ ONLY END-EXEC.
EXEC SQL LOB OPEN :TEMP-BLOB READ WRITE END-EXEC.

* BLOB2 からテンポラリ BLOB にデータをコピーします。
EXEC SQL
    LOB COPY :AMT FROM :BLOB2 TO :TEMP-BLOB
END-EXEC.

* PL/SQL を実行して、COMPARE 機能を使用します。
MOVE 5 TO AMT.
EXEC SQL EXECUTE
    BEGIN
        :RET := DBMS_LOB.COMPARE(:BLOB1, :TEMP-BLOB, :AMT, 1, 1);
    END;
END-EXEC.

IF RET = 0
*     同等の BLOB の論理がここに入ります。
    DISPLAY "BLOBs are equal"
ELSE
*     同等ではない BLOB の論理がここに入ります。
    DISPLAY "BLOBs are not equal"
END-IF.

EXEC SQL LOB CLOSE :BLOB1 END-EXEC.
EXEC SQL LOB CLOSE :BLOB2 END-EXEC.
EXEC SQL LOB CLOSE :TEMP-BLOB END-EXEC.
EXEC SQL LOB FREE TEMPORARY :TEMP-BLOB END-EXEC.

EXEC SQL FREE :TEMP-BLOB END-EXEC.

END-OF-BLOB.

EXEC SQL WHENEVER NOT FOUND CONTINUE END-EXEC.
EXEC SQL FREE :BLOB1 END-EXEC.
EXEC SQL FREE :BLOB2 END-EXEC.
STOP RUN.

SQL-ERROR.
EXEC SQL
    WHENEVER SQLERROR CONTINUE
END-EXEC.
```

```

MOVE ORASLNR TO ORASLNRD.
DISPLAY " ".
DISPLAY "ORACLE ERROR DETECTED ON LINE ", ORASLNRD, ":".
DISPLAY " ".
DISPLAY SQLERRMC.
EXEC SQL
    ROLLBACK WORK RELEASE
END-EXEC.
STOP RUN.

```

C/C++（Pro*C/C++）：2つの（テンポラリ）LOBの全体または一部の比較

```

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

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

void compareTwoTempOrPersistLOBs_proc()
{
    OCIBlobLocator *Lob_loc1, *Lob_loc2, *Temp_loc;
    int Amount = 128;
    int Retval;

    EXEC SQL WHENEVER SQLERROR DO Sample_Error();
    /* LOB ロケータを割り当てます。*/
    EXEC SQL ALLOCATE :Lob_loc1;
    EXEC SQL ALLOCATE :Lob_loc2;
    /* LOB を選択します。*/
    EXEC SQL SELECT Frame INTO :Lob_loc1
        FROM Multimedia_tab WHERE Clip_ID = 1;
    EXEC SQL SELECT Frame INTO :Lob_loc2
        FROM Multimedia_tab WHERE Clip_ID = 2;
    /* テンポラリ LOB の割当ておよび作成を行います。*/
    EXEC SQL ALLOCATE :Temp_loc;
    EXEC SQL LOB CREATE TEMPORARY :Temp_loc;

    /* LOB のオープンはオプションです。*/

```

2つの（テンポラリ）LOBの全体または一部の比較

```
EXEC SQL LOB OPEN :Lob_loc1 READ ONLY;
EXEC SQL LOB OPEN :Lob_loc2 READ ONLY;
EXEC SQL LOB OPEN :Temp_loc READ WRITE;
/* テンポラリLOBに永続LOBをコピーします。*/
EXEC SQL LOB COPY :Amount FROM :Lob_loc2 TO :Temp_loc;

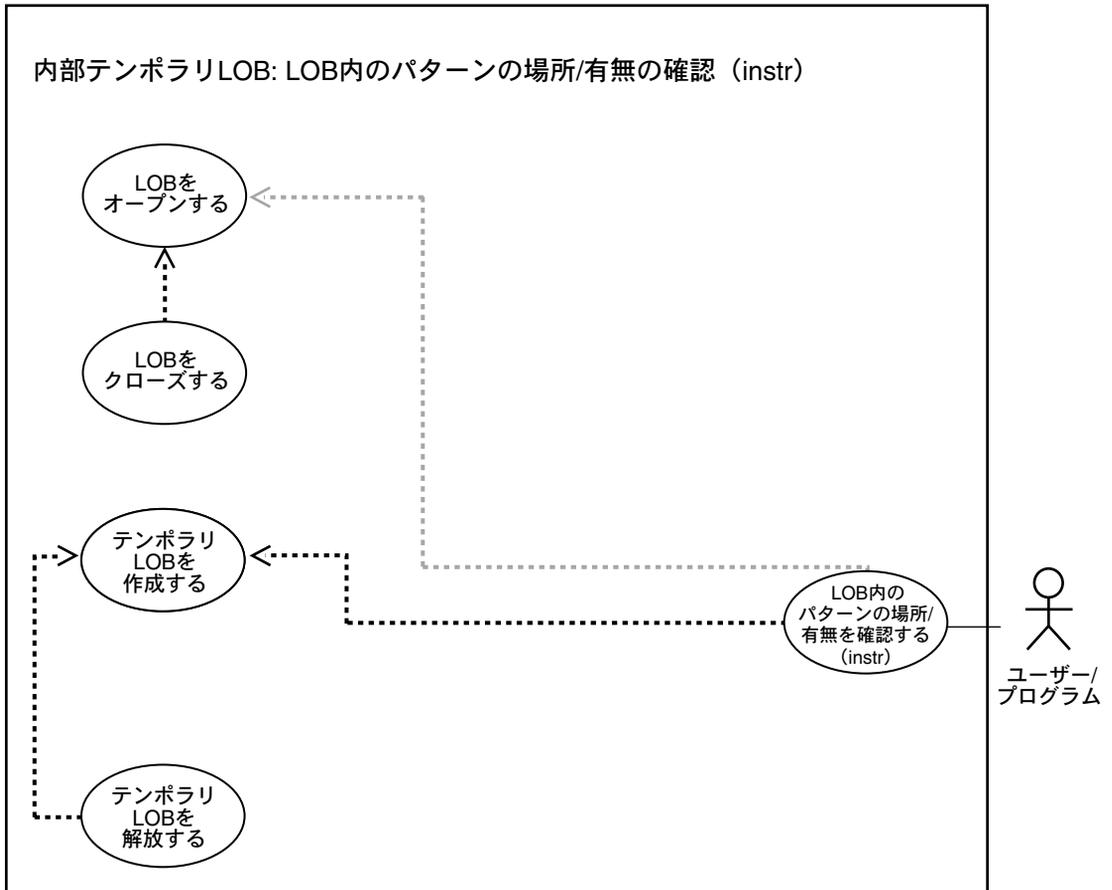
/* PL/SQL内からDBMS_LOB.COMPARE()を使用して、2つのフレームを比較します。*/
EXEC SQL EXECUTE
    BEGIN
        :RetVal := DBMS_LOB.COMPARE(:Lob_loc1, :Temp_loc, :Amount, 1, 1);
    END;
END-EXEC;
if (0 == Retval)
    printf("Frames are equal\n");
else
    printf("Frames are not equal\n");
/* LOBをオープンしている場合、そのLOBをクローズする必要があります。*/
EXEC SQL LOB CLOSE :Lob_loc1;
EXEC SQL LOB CLOSE :Lob_loc2;
EXEC SQL LOB CLOSE :Temp_loc;
/* テンポラリLOBを解放します。*/
EXEC SQL LOB FREE TEMPORARY :Temp_loc;

/* ロケータが保持しているリソースを解放します。*/
EXEC SQL FREE :Lob_loc1;
EXEC SQL FREE :Lob_loc2;
EXEC SQL FREE :Temp_loc;
}

void main()
{
    char *samp = "samp/samp";
    EXEC SQL CONNECT :samp;
    compareTwoTempOrPersistLOBs_proc();
    EXEC SQL ROLLBACK WORK RELEASE;
}
```

テンポラリ LOB 内のパターンの有無の確認 (instr)

図 10-12 ユースケース図：テンポラリ LOB 内のパターンの有無の確認 (instr)



参照： 内部テンポラリ LOB に関するすべての基本操作については、10-2 ページの「ユースケース・モデル: 内部テンポラリ LOB」を参照してください。

用途

テンポラリ LOB 内のパターンの有無を確認します (instr)。

使用上の注意

ありません。

構文

各プログラム環境で使用可能な機能のリストは、第 3 章「LOB プログラム環境」を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- PL/SQL (DBMS_LOB) : 『Oracle8i PL/SQL パッケージ・プロシージャ・リファレンス』の第 18 章「DBMS_LOB」の「サブプログラムの要約」の INSTR ファンクション
- C (OCI) : 参照マニュアルはありません。
- COBOL (Pro*COBOL) : 『Oracle8i Pro*COBOL プリコンパイラ・プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB COPY (実行可能埋込み SQL 拡張要素)」、および『Oracle8i PL/SQL パッケージ・プロシージャ・リファレンス』の第 18 章「DBMS_LOB」の「サブプログラムの要約」の INSTR ファンクション
- C/C++ (Pro*C/C++) : 『Oracle8i Pro*C/C++ プリコンパイラ・プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB COPY (実行可能埋込み SQL 拡張要素)」、および『Oracle8i PL/SQL パッケージ・プロシージャ・リファレンス』の第 18 章「DBMS_LOB」の「サブプログラムの要約」の INSTR ファンクション
- Visual Basic (OO4O) : 参照マニュアルはありません。
- Java (JDBC) : 参照マニュアルはありません。

使用例

この例では、ストーリーボード・テキストを調べ、「children」という文字列が存在するかどうかをチェックします。

例

次のプログラム環境の例が示されています。

- [PL/SQL \(DBMS_LOB パッケージ\) : テンポラリ LOB 内のパターンの有無の確認 \(instr\) \(10-82 ページ\)](#)
- C (OCI) : 今回のリリースでは例は記載されていません。
- [COBOL \(Pro*COBOL\) : テンポラリ LOB 内のパターンの有無の確認 \(instr\) \(10-84 ページ\)](#)

- C/C++ (Pro*C/C++) : テンポラリ LOB 内のパターンの有無の確認 (instr) (10-86 ページ)
- Visual Basic (OO4O) : 今回のリリースでは例は記載されていません。
- Java (JDBC) : 今回のリリースでは例は記載されていません。

PL/SQL (DBMS_LOB パッケージ) : テンポラリ LOB 内のパターンの有無の確認 (instr)

/* 例のプロシージャ instrTempLOB_proc は、DBMS_LOB パッケージの一部ではないことに注意してください。*/

```
CREATE OR REPLACE PROCEDURE instrTempLOB_proc IS
  Lob_loc      CLOB;
  Temp_clob    CLOB;
  Pattern      VARCHAR2(30) := 'children';   Position      INTEGER := 0;
  Offset       INTEGER := 1;
  Occurrence   INTEGER := 1;
BEGIN
  /* テンポラリ LOB を作成し、それに CLOB をコピーします。*/
  DBMS_LOB.CREATETEMPORARY(Temp_clob,TRUE);
  SELECT Story INTO Lob_loc
     FROM Multimedia_tab
     WHERE Clip_ID = 1;

  DBMS_LOB.OPEN(Temp_clob,DBMS_LOB.LOB_READWRITE);
  DBMS_LOB.OPEN(Lob_loc,DBMS_LOB.LOB_READONLY);
  /* テンポラリ CLOB に CLOB をコピーします。*/
  DBMS_LOB.COPY(Temp_clob,Lob_loc,DBMS_LOB.GETLENGTH(Lob_loc),1,1);
  /* テンポラリ CLOB 内のパターンを検索します。*/
  Position := DBMS_LOB.INSTR(Temp_clob, Pattern, Offset, Occurrence);
  IF Position = 0 THEN
    DBMS_OUTPUT.PUT_LINE('Pattern not found');
  ELSE
    DBMS_OUTPUT.PUT_LINE('The pattern occurs at ' || position);
  END IF;
  DBMS_LOB.CLOSE(Lob_loc);
  DBMS_LOB.CLOSE(Temp_clob);
  /* テンポラリ LOB を解放します。*/
  DBMS_LOB.FREETEMPORARY(Temp_clob);
END;
```

COBOL (Pro*COBOL) : テンポラリ LOB 内のパターンの有無の確認 (instr)

```
IDENTIFICATION DIVISION.
PROGRAM-ID. CLOB-INSTR.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

01  USERID    PIC X(11) VALUES "SAMP/SAMP".
01  CLOB1     SQL-CLOB.
01  TEMP-CLOB SQL-CLOB.
01  PATTERN   PIC X(8) VALUE "children".
01  BUFFER2   PIC X(32767) VARYING.
01  OFFSET    PIC S9(9) COMP VALUE 1.
01  OCCURRENCE PIC S9(9) COMP VALUE 1.
01  LEN       PIC S9(9) COMP.
01  POS       PIC S9(9) COMP.
01  ORASLNRD  PIC 9(4).

EXEC SQL INCLUDE SQLCA END-EXEC.
EXEC ORACLE OPTION (ORACA=YES) END-EXEC.
EXEC SQL INCLUDE ORACA END-EXEC.
EXEC SQL VAR BUFFER2 IS LONG RAW(32767) END-EXEC.

PROCEDURE DIVISION.
CLOB-INSTR.

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

* CLOB ロケータの割当ておよび初期化を行います。
EXEC SQL ALLOCATE :CLOB1 END-EXEC.

EXEC SQL WHENEVER NOT FOUND GOTO END-OF-CLOB END-EXEC.

EXEC SQL WHENEVER NOT FOUND GOTO END-OF-CLOB END-EXEC.
EXEC ORACLE OPTION (SELECT_ERROR=NO) END-EXEC.
EXEC SQL
SELECT STORY INTO :CLOB1
FROM MULTIMEDIA_TAB WHERE CLIP_ID = 1
END-EXEC.
EXEC SQL ALLOCATE :TEMP-CLOB END-EXEC.
EXEC SQL
LOB CREATE TEMPORARY :TEMP-CLOB
END-EXEC.
```

```

* CLOB を READ ONLY でオープンします。
EXEC SQL LOB OPEN :CLOB1 READ ONLY END-EXEC.

* LOB DESCRIBE を使用して、CLOB1 の長さを取得します。
EXEC SQL
    LOB DESCRIBE :CLOB1 GET LENGTH INTO :LEN
END-EXEC.
EXEC SQL
    LOB COPY :LEN FROM :CLOB1 TO :TEMP-CLOB
END-EXEC.

* PL/SQL を実行して、INSTR 機能を取得します。
EXEC SQL EXECUTE
    BEGIN
        :POS := DBMS_LOB.INSTR(:TEMP-CLOB, :PATTERN,
                               :OFFSET, :OCCURRENCE);

    END;
END-EXEC.

IF POS = 0
*     パターンが見つからない場合の論理はここにあります。
    DISPLAY "Pattern was not found"
ELSE
*     Pos には、パターンが見つかった位置が含まれます。
    DISPLAY "Pattern was found"
END-IF.

* LOB をクローズおよび解放します。
EXEC SQL LOB CLOSE :CLOB1 END-EXEC.
EXEC SQL FREE :TEMP-CLOB END-EXEC.
EXEC SQL
    LOB FREE TEMPORARY :TEMP-CLOB
END-EXEC.
EXEC SQL FREE :TEMP-CLOB END-EXEC.

END-OF-CLOB.

EXEC SQL WHENEVER NOT FOUND CONTINUE END-EXEC.
EXEC SQL FREE :CLOB1 END-EXEC.
STOP RUN.

SQL-ERROR.
EXEC SQL
    WHENEVER SQLEERROR CONTINUE
END-EXEC.
MOVE ORASLNR TO ORASLNRD.
DISPLAY " ".

```

```
        DISPLAY "ORACLE ERROR DETECTED ON LINE ", ORASLNRD, ":".
        DISPLAY " ".
        DISPLAY SQLERRMC.
        EXEC SQL
            ROLLBACK WORK RELEASE
        END-EXEC.
        STOP RUN.
```

C/C++ (Pro*C/C++) : テンポラリ LOB 内のパターンの有無の確認 (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 instrstringTempLOB_proc()
{
    OCIClobLocator *Lob_loc, *Temp_loc;
    char *Pattern = "The End";
    unsigned int Length;
    int Position = 0;
    int Offset = 1;
    int Occurrence = 1;

    EXEC SQL WHENEVER SQLERROR DO Sample_Error();
    /* 永続 LOB の割当ておよび初期化を行います。*/
    EXEC SQL ALLOCATE :Lob_loc;
    EXEC SQL SELECT Story INTO :Lob_loc
        FROM Multimedia_tab WHERE Clip_ID = 1;
    /* テンポラリ LOB の割当ておよび作成を行います。*/
    EXEC SQL ALLOCATE :Temp_loc;
    EXEC SQL LOB CREATE TEMPORARY :Temp_loc;
    /* LOB のオープンはおプションです。*/
    EXEC SQL LOB OPEN :Lob_loc READ ONLY;
    EXEC SQL LOB OPEN :Temp_loc READ WRITE;
    /* 永続 LOB の長さを決定します。*/
    EXEC SQL LOB DESCRIBE :Lob_loc GET LENGTH into :Length;
    /* テンポラリ LOB に永続 LOB をコピーします。*/
```

```

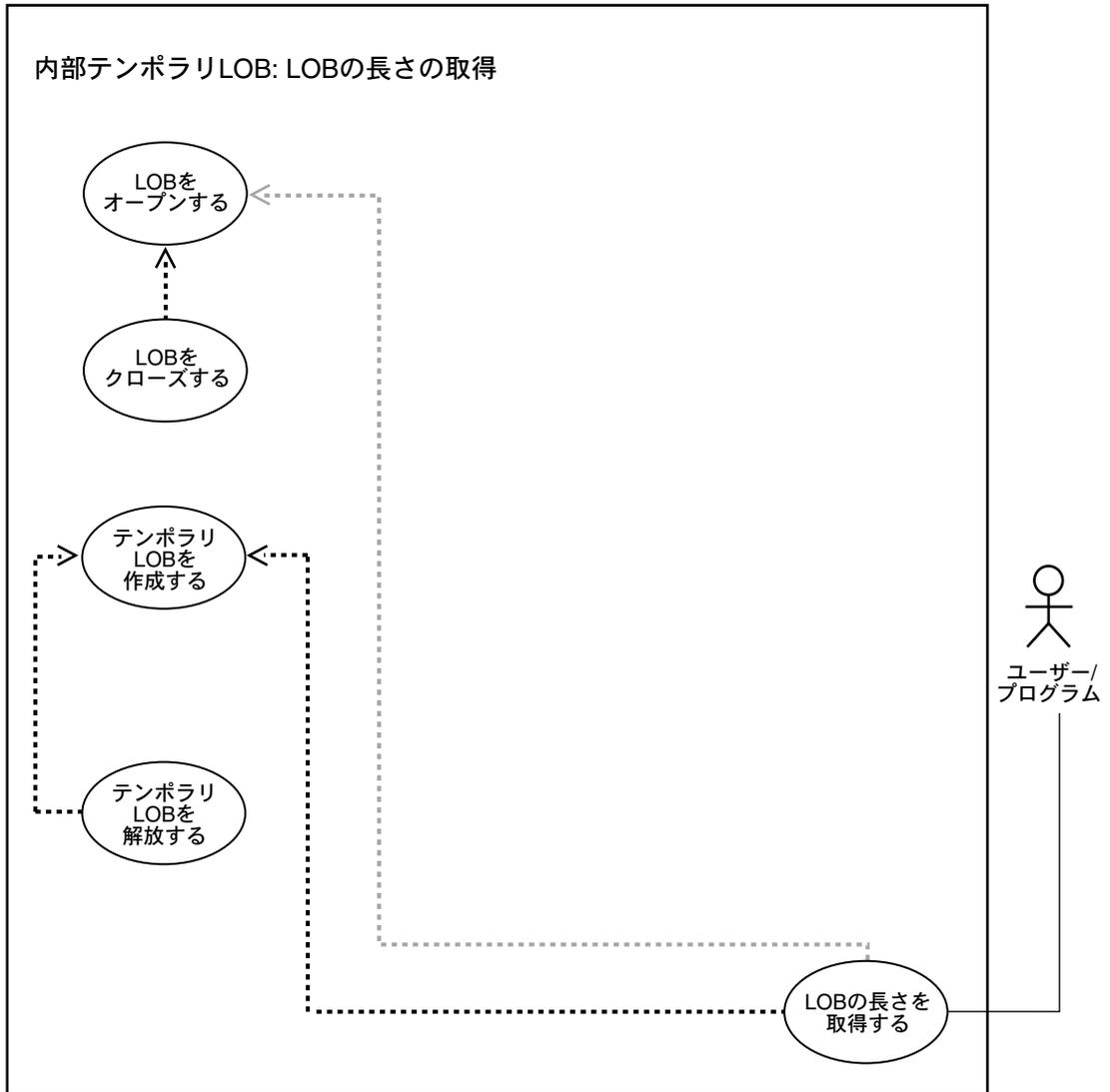
EXEC SQL LOB COPY :Length FROM :Lob_loc TO :Temp_loc;
/* PL/SQL ブロック内で DBMS_LOB.INSTR() を使用して、パターンを検索します。*/
EXEC SQL EXECUTE
    BEGIN
        :Position :=
            DBMS_LOB.INSTR(:Temp_loc, :Pattern, :Offset, :Occurrence);
    END;
END-EXEC;
if (0 == Position)
    printf("Pattern not found\n");
else
    printf("The pattern occurs at %d\n", Position);
/* LOB をオープンしている場合、その LOB をクローズする必要があります。*/
EXEC SQL LOB CLOSE :Lob_loc;
EXEC SQL LOB CLOSE :Temp_loc;
/* テンポラリ LOB を解放します。*/
EXEC SQL LOB FREE TEMPORARY :Temp_loc;
/* ロケータが保持しているリソースを解放します。*/
EXEC SQL FREE :Lob_loc;
EXEC SQL FREE :Temp_loc;
}

void main()
{
    char *samp = "samp/samp";
    EXEC SQL CONNECT :samp;
    instrstringTempLOB_proc();
    EXEC SQL ROLLBACK WORK RELEASE;
}

```

テンポラリ LOB の長さの取得

図 10-13 ユースケース図：テンポラリ LOB の長さの取得



参照： 内部テンポラリ LOB に関するすべての基本操作については、10-2 ページの「ユースケース・モデル: 内部テンポラリ LOB」を参照してください。

用途

テンポラリ LOB の長さを取得します。

使用上の注意

ありません。

構文

各プログラム環境で使用可能な機能のリストは、第 3 章「LOB プログラム環境」を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- PL/SQL (DBMS_LOB) : 『Oracle8i PL/SQL パッケージ・プロシージャ・リファレンス』の第 18 章「DBMS_LOB」の「サブプログラムの要約」の「GETLENGTH」
- C (OCI) : 『Oracle8i コール・インタフェース・プログラマーズ・ガイド』の第 15 章「OCI リレーショナル関数」の「LOB 関数」の OCILobGetLength()
- COBOL (Pro*COBOL) : 『Oracle8i Pro*COBOL プリコンパイラ・プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB DESCRIBE (実行可能埋込み SQL 拡張要素)」、および『Oracle8i PL/SQL パッケージ・プロシージャ・リファレンス』の第 18 章「DBMS_LOB」の「サブプログラムの要約」の「GETLENGTH」
- C/C++ (Pro*C/C++) : 『Oracle8i Pro*C/C++ プリコンパイラ・プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB DESCRIBE (実行可能埋込み SQL 拡張要素)」、および『Oracle8i PL/SQL パッケージ・プロシージャ・リファレンス』の第 18 章「DBMS_LOB」の「サブプログラムの要約」の「GETLENGTH」
- Visual Basic (OO4O) : 参照マニュアルはありません。
- Java (JDBC) : 参照マニュアルはありません。

使用例

次の例では、インタビューの長さを取得して、4GB の制限を超えていないかどうかを調べます。

例

次のプログラム環境の例が示されています。

- PL/SQL (DBMS_LOB パッケージ) : テンポラリ LOB の長さの取得 (10-90 ページ)
- C (OCI) : テンポラリ LOB の長さの取得 (10-91 ページ)
- COBOL (Pro*COBOL) : テンポラリ LOB の長さの取得 (10-93 ページ)
- C/C++ (Pro*C/C++) : テンポラリ LOB の長さの取得 (10-95 ページ)
- Visual Basic (OO4O) : 今回のリリースでは例は記載されていません。
- Java (JDBC) : 今回のリリースでは例は記載されていません。

PL/SQL (DBMS_LOB パッケージ) : テンポラリ LOB の長さの取得

/* 例のプロシージャ getLengthTempCLOB_proc は、DBMS_LOB パッケージの一部ではないことに注意してください。 */

```
CREATE OR REPLACE PROCEDURE getLengthTempCLOB_proc IS
    Length      INTEGER;
    tlob        CLOB;
    bufc        VARCHAR2(8);
    Amount      NUMBER;
    pos         NUMBER;
    Src_loc     BFILE := BFILENAME('AUDIO_DIR', 'Washington_audio');
BEGIN
    DBMS_LOB.CREATETEMPORARY(tlob,TRUE);
    /* LOB のオープンはおプションです。 */
    DBMS_LOB.OPEN(tlob,DBMS_LOB.LOB_READWRITE);
    /* ファイルのオープンは必須です。 */
    DBMS_LOB.OPEN(Src_loc, DBMS_LOB.LOB_READONLY);
    Amount := 32767;
    DBMS_LOB.LOADFROMFILE(tlob, Src_loc, Amount);
    /* LOB の長さを取得します。 */
    length := DBMS_LOB.GETLENGTH(tlob);
    IF length = 0 THEN
        DBMS_OUTPUT.PUT_LINE('LOB is empty. ');
    ELSE
        DBMS_OUTPUT.PUT_LINE('The length is ' || length);
    END IF;
    /* オープンされたすべての LOB をクローズする必要があります。 */
    DBMS_LOB.CLOSE(tlob);
    DBMS_LOB.CLOSE(Src_loc);
    /* 使い終わったテンポラリ LOB を解放します。 */
    DBMS_LOB.FREETEMPORARY(tlob);
END;
```

C (OCI) : テンポラリ LOB の長さの取得

/* このファンクションはテンポラリ LOB ロケータを引数の量として取り、対応する LOB の長さを出力します。このファンクションは、成功した場合は 0 (ゼロ) を戻し、失敗した場合は -1 を戻します。*/

```

sb4 print_length( OCIError      *errhp,
                  OCISvcCtx    *svchp,
                  OCIStmt     *stmthp,
                  OCIEnv       *envhp)
{
    ub4 length=0;
    ub4 amount = 4;
    ub4 pos = 1;
    OCILobLocator *bfile;
    OCILobLocator *tblob;
    sb4 return_code = 0;

    printf("in print_length\n");
    if(OCIDescriptorAlloc((dvoid *)envhp, (dvoid **) &tblob,
                          (ub4) OCI_DTYPE_LOB,
                          (size_t) 0, (dvoid **) 0))
    {
        printf("OCIDescriptor Alloc FAILED in print_length\n");
        return -1;
    }

    if(OCIDescriptorAlloc((dvoid *)envhp, (dvoid **) &bfile,
                          (ub4) OCI_DTYPE_FILE,
                          (size_t) 0, (dvoid **) 0))
    {
        printf("OCIDescriptor Alloc FAILED in print_length\n");
        return -1;
    }

    if(OCILobFileSetName(envhp, errhp, &bfile, (text *)"AUDIO_DIR",
                          (ub2)strlen("AUDIO_DIR"),
                          (text *)"Washington_audio",
                          (ub2)strlen("Washington_audio")))
    {
        printf("OCILobFileSetName FAILED\n");
        return_code = -1;
    }
    checkerr(errhp, (OCILobFileOpen(svchp, errhp,

```

```

                                (OCILobLocator *) bfile,
                                OCI_LOB_READONLY));
/* テンポラリ BLOB を作成します。 */
if (OCILobCreateTemporary(svchp, errhp, tblob, (ub2)0, SQLCS_IMPLICIT,
                          OCI_TEMP_BLOB, OCI_ATTR_NOCACHE,
                          OCI_DURATION_SESSION))
{
    (void) printf("FAILED: CreateTemporary() \n");
    return_code = -1 ;
}

if (OCILobOpen(svchp, errhp, (OCILobLocator *) tblob, OCI_LOB_READWRITE))
{
    (void) printf("FAILED: Open Temporary \n");
    return_code = -1;
}

if (OCILobLoadFromFile(svchp, errhp, tblob, (OCILobLocator*)bfile,
                      (ub4)amount, (ub4)1, (ub4)1))
{
    (void) printf("FAILED: Open Temporary \n");
    return_code = -1;
}

if (OCILobGetLength(svchp, errhp, tblob, &length))
{
    printf ("FAILED: OCILobGetLength in print_length\n");
    return_code = -1;
}

/* BFILE およびテンポラリ LOB をクローズします。 */
checkerr(errhp, OCILobFileClose(svchp, errhp, (OCILobLocator *) bfile));

checkerr(errhp, OCILobClose(svchp, errhp, (OCILobLocator *) tblob));

/* 使い終わったテンポラリ LOB を解放します。 */
if (OCILobFreeTemporary(svchp, errhp, tblob))
{
    printf("OCILobFreeTemporary FAILED \n");
    return_code = -1;
}
fprintf(stderr, "Length of LOB is %d\n", length);
return return_code;
}

```

COBOL (Pro*COBOL) : テンポラリ LOB の長さの取得

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

01  USERID    PIC X(11) VALUES "SAMP/SAMP".
01  TEMP-BLOB      SQL-BLOB.
01  SRC-BFILE     SQL-BFILE.
01  DIR-ALIAS     PIC X(30) VARYING.
01  FNAME        PIC X(20) VARYING.
01  DIR-IND       PIC S9(4) COMP.
01  FNAME-IND     PIC S9(4) COMP.
01  AMT           PIC S9(9) COMP VALUE 10.
01  LEN          PIC S9(9) COMP.
01  LEN-D         PIC 9(4).
01  ORASLNRD     PIC 9(4).

EXEC SQL INCLUDE SQLCA END-EXEC.
EXEC ORACLE OPTION (ORACA=YES) END-EXEC.
EXEC SQL INCLUDE ORACA END-EXEC.

PROCEDURE DIVISION.
TEMP-LOB-LENGTH.
EXEC SQL WHENEVER SQLERROR DO PERFORM SQL-ERROR END-EXEC.
EXEC SQL
CONNECT :USERID
END-EXEC.

* BFILE および BLOB ロケータの割当ておよび初期化を行います。
EXEC SQL ALLOCATE :TEMP-BLOB END-EXEC.
EXEC SQL ALLOCATE :SRC-BFILE END-EXEC.
EXEC SQL
LOB CREATE TEMPORARY :TEMP-BLOB
END-EXEC.

* ディレクトリおよびファイル情報を設定します。
MOVE "AUDIO_DIR" TO DIR-ALIAS-ARR.
MOVE 9 TO DIR-ALIAS-LEN.
MOVE "Washington_audio" TO FNAME-ARR.
MOVE 16 TO FNAME-LEN.

EXEC SQL
LOB FILE SET :SRC-BFILE DIRECTORY = :DIR-ALIAS,
```

テンポラリ LOB の長さの取得

```
FILENAME = :FNAME  
END-EXEC.
```

- * ソース BFILE および宛先テンポラリ BLOB をオープンします。

```
EXEC SQL LOB OPEN :TEMP-BLOB READ WRITE END-EXEC.  
EXEC SQL LOB OPEN :SRC-BFILE READ ONLY END-EXEC.  
EXEC SQL  
    LOB LOAD :AMT FROM FILE :SRC-BFILE INTO :TEMP-BLOB  
END-EXEC.
```

- * テンポラリ LOB の長さを取得します。

```
EXEC SQL  
    LOB DESCRIBE :TEMP-BLOB GET LENGTH INTO :LEN  
END-EXEC.  
MOVE LEN TO LEN-D.  
DISPLAY "Length of TEMPORARY LOB is ", LEN-D.
```

- * LOB をクローズします。

```
EXEC SQL LOB CLOSE :SRC-BFILE END-EXEC.  
EXEC SQL LOB CLOSE :TEMP-BLOB END-EXEC.
```

- * テンポラリ LOB を解放します。

```
EXEC SQL  
    LOB FREE TEMPORARY :TEMP-BLOB  
END-EXEC.
```

- * LOB ロケータを解放します。

```
EXEC SQL FREE :TEMP-BLOB END-EXEC.  
EXEC SQL FREE :SRC-BFILE END-EXEC.  
STOP RUN.
```

```
SQL-ERROR.
```

```
EXEC SQL WHENEVER SQLERROR CONTINUE END-EXEC.  
MOVE ORASLNR TO ORASLNRD.  
DISPLAY " ".  
DISPLAY "ORACLE ERROR DETECTED ON LINE ", ORASLNRD, ":".  
DISPLAY " ".  
DISPLAY SQLERRMC.  
EXEC SQL ROLLBACK WORK RELEASE END-EXEC.  
STOP RUN.
```

C/C++ (Pro*C/C++) : テンポラリ LOB の長さの取得

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

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

void getLengthTempLOB_proc()
{
    OCIBlobLocator *Temp_loc;
    OCIFileLocator *Lob_loc;
    char *Dir = "AUDIO_DIR", *Name = "Washington_audio";
    int Length, Amount;

    EXEC SQL WHENEVER SQLERROR DO Sample_Error();

    /* テンポラリ LOB の割当ておよび作成を行います。 */
    EXEC SQL ALLOCATE :Temp_loc;
    EXEC SQL LOB CREATE TEMPORARY :Temp_loc;

    /* BFILE ロケータの割当ておよび初期化を行います。 */
    EXEC SQL ALLOCATE :Lob_loc;
    EXEC SQL LOB FILE SET :Lob_loc DIRECTORY = :Dir, FILENAME = :Name;

    /* LOB のオープンはオプションです。 */
    EXEC SQL LOB OPEN :Lob_loc READ ONLY;
    EXEC SQL LOB OPEN :Temp_loc READ WRITE;

    /* BFILE からテンポラリ LOB に、指定した量をロードします。 */
    Amount = 4096;
    EXEC SQL LOB LOAD :Amount FROM FILE :Lob_loc INTO :Temp_loc;

    /* テンポラリ LOB の長さを取得します。 */
    EXEC SQL LOB DESCRIBE :Temp_loc GET LENGTH INTO :Length;

    /* この例では、Length == Amount == 4096 であることに注意してください。 */
    printf("Length is %d bytes\n", Length);

    /* LOB をオープンしている場合、その LOB をクローズする必要があります。 */
    EXEC SQL LOB CLOSE :Lob_loc;
}
```

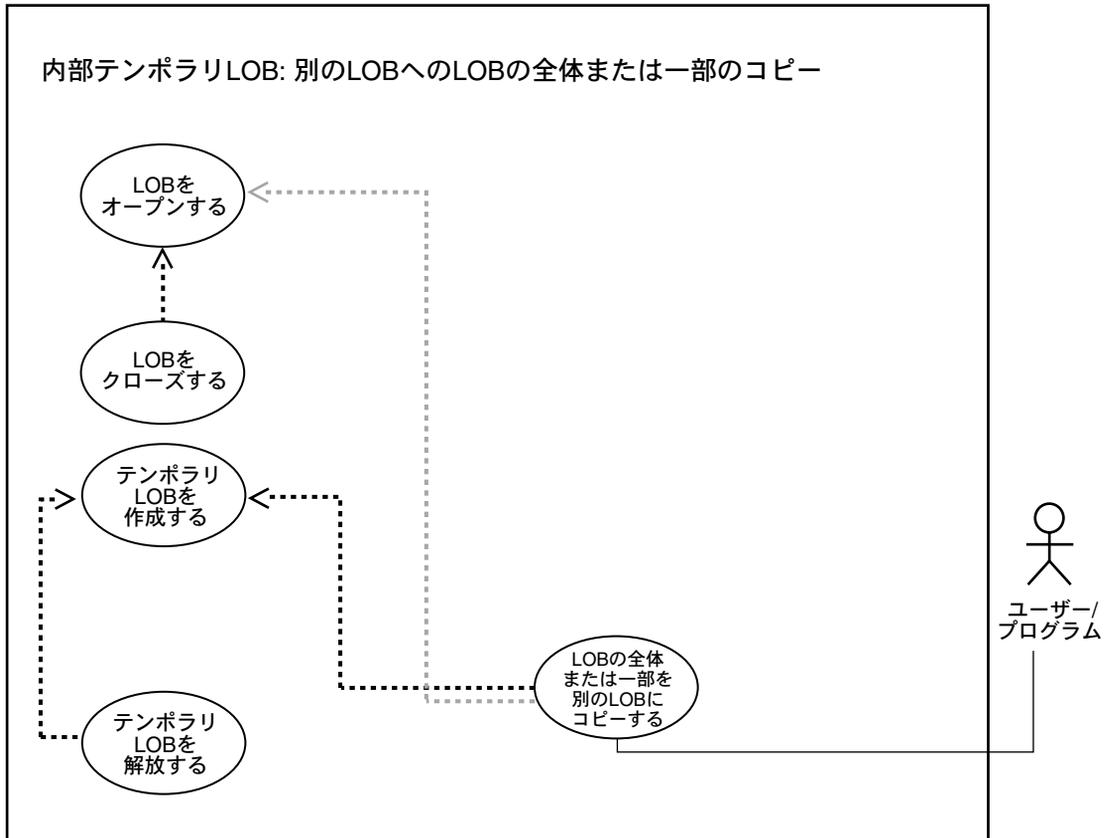
```
EXEC SQL LOB CLOSE :Temp_loc;

/* テンポラリ LOB を解放します。*/
EXEC SQL LOB FREE TEMPORARY :Temp_loc;
/* ロケータが保持しているリソースを解放します。*/
EXEC SQL FREE :Lob_loc;
EXEC SQL FREE :Temp_loc;
}

void main()
{
    char *samp = "samp/samp";
    EXEC SQL CONNECT :samp;
    getLengthTempLOB_proc();
    EXEC SQL ROLLBACK WORK RELEASE;
}
```

(テンポラリ) LOB の全体または一部の他へのコピー

図 10-14 ユースケース図：(テンポラリ) LOB の全体または一部の他へのコピー



参照： 内部テンポラリ LOB に関するすべての基本操作については、10-2 ページの「ユースケース・モデル: 内部テンポラリ LOB」を参照してください。

用途

テンポラリ LOB の全体または一部を他へコピーします。

使用上の注意

ありません。

構文

各プログラム環境で使用可能な機能のリストは、第3章「LOB プログラム環境」を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- PL/SQL (DBMS_LOB) : 『Oracle8i PL/SQL パッケージ・プロシージャ・リファレンス』の第18章「DBMS_LOB」の「サブプログラムの要約」の COPY プロシージャ
- C (OCI) : 『Oracle8i コール・インタフェース・プログラマーズ・ガイド』の第15章「OCI リレーショナル関数」の「LOB 関数」の OCILobCopy()
- COBOL (Pro*COBOL) : 『Oracle8i Pro*COBOL プリコンパイラ・プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB COPY (実行可能埋込み SQL 拡張要素)」
- C/C++ (Pro*C/C++) : 『Oracle8i Pro*C/C++ プリコンパイラ・プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB COPY (実行可能埋込み SQL 拡張要素)」
- Visual Basic (OO4O) : 参照マニュアルはありません。
- Java (JDBC) : 参照マニュアルはありません。

使用例

次の表を想定しています。

```
CREATE TABLE VoiceoverLib_tab of VOICED_TYP;
```

この VoiceoverLib_tab は、Multimedia_tab 表の Voiced_ref 列が参照している Voiceover_tab と同じ型です。

```
INSERT INTO Voiceover_tab  
  (SELECT * FROM VoiceoverLib_tab Vtab1  
   WHERE T2.Take = 101);
```

この文によって、Voiceover_tab 表内に新しい LOB ロケータが作成され、Voiceover_tab 表に挿入された新しいロケータによって参照される位置に vtab1 からの LOB データがコピーされます。

例

次のプログラム環境の例が示されています。

- PL/SQL (DBMS_LOB パッケージ) : (テンポラリ) LOB の全体または一部の他へのコピー (10-98 ページ)
- C (OCI) : (テンポラリ) LOB の全体または一部の他へのコピー (10-100 ページ)
- COBOL (Pro*COBOL) : (テンポラリ) LOB の全体または一部の他へのコピー (10-103 ページ)
- C/C++ (Pro*C/C++) : (テンポラリ) LOB の全体または一部の他へのコピー (10-105 ページ)
- Visual Basic (OO4O) : 今回のリリースでは例は記載されていません。
- Java (JDBC) : 今回のリリースでは例は記載されていません。

PL/SQL (DBMS_LOB パッケージ) : (テンポラリ) LOB の全体または一部の他へのコピー

/* 例のプロシージャ copyTempLOB_proc は、DBMS_LOB パッケージの一部ではないことに注意してください。*/

```
CREATE OR REPLACE PROCEDURE copyTempLOB_proc IS
  Dest_pos      NUMBER;
  Src_pos       NUMBER;
  Dest_loc      BLOB;
  Dest_loc2     BLOB;
  Src_loc       BFILE := BFILENAME('AUDIO_DIR', 'Washington_audio');
  Amount        INTEGER := 32767;
BEGIN
  DBMS_LOB.CREATETEMPORARY(Dest_loc2, TRUE);
  DBMS_LOB.CREATETEMPORARY(Dest_loc, TRUE);
  /* FILE のオープンは必須です。*/
  DBMS_LOB.OPEN(Src_loc, DBMS_LOB.LOB_READONLY);
  /* テンポラリ LOB のオープンはオプションです。*/
  DBMS_LOB.OPEN(Dest_loc, DBMS_LOB.LOB_READWRITE);
  DBMS_LOB.OPEN(Dest_loc2, DBMS_LOB.LOB_READWRITE);

  DBMS_LOB.LOADFROMFILE(Dest_loc, Src_loc, Amount);
  /* Dest_pos を、ターゲット・テンポラリ LOB への書き込みを開始する必要がある
  位置に設定します。*/
  /* ソースの位置から宛先の位置に LOB をコピーします。*/
  /* Amount をコピーする量に設定します。*/
  Amount := 328;
  Dest_pos := 1000;
  Src_pos := 1000;
  /* Src_pos を、tcllob_src からのデータのコピーを開始する必要がある位置に設定
  します。*/
  DBMS_LOB.COPY(Dest_loc2, Dest_loc, Amount, Dest_pos, Src_pos);
```

```
COMMIT;
EXCEPTION
WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('Operation failed');
    DBMS_LOB.CLOSE(Dest_loc);
    DBMS_LOB.CLOSE(Dest_loc2);
    DBMS_LOB.CLOSE(Src_loc);
    DBMS_LOB.FREETEMPORARY(Dest_loc);
    DBMS_LOB.FREETEMPORARY(Dest_loc2);
END;
```

C (OCI) : (テンポラリ) LOB の全体または一部の他へのコピー

/* このファンクションは、1つのテンポラリ LOB から別のテンポラリ LOB に 4,000 バイトをコピーします。オフセット 1 から開始してソース LOB を読み込み、オフセット 2 で宛先に書き込みます。このファンクションは、成功した場合は 0 (ゼロ) を返し、失敗した場合は -1 を返します。*/

```
sb4 copy_temp_lobs (OCIError      *errhp,
                   OCISvcCtx     *svchp,
                   OCIStmt      *stmthp,
                   OCIEnv       *envhp)
{
    OCIDefine *defnp1;
    OCILobLocator *tblob;
    OCILobLocator *tblob2;
    OCILobLocator *bfile;
    int rowind =1;
    ub4 amount=4000;
    ub4 src_offset=1;
    ub4 dest_offset=2;
    sb4 return_code = 0;

    printf("in copy_temp_lobs \n");

    if(OCIDescriptorAlloc((dvoid*)envhp, (dvoid **)&tblob,
                        (ub4)OCI_DTYPE_LOB, (size_t)0, (dvoid**)0))
    {
        printf("OCIDescriptorAlloc failed in copy_temp_lobs\n");
        return -1;
    }

    if(OCIDescriptorAlloc((dvoid*)envhp, (dvoid **)&bfile,
                        (ub4)OCI_DTYPE_FILE, (size_t)0, (dvoid**)0))
```

```
{
    printf("OCIDescriptorAlloc failed in copy_temp_lob\n");
    return -1;
}

if(OCILobCreateTemporary(svchp, errhp, tblob, (ub2)0, SQLCS_IMPLICIT,
                        OCI_TEMP_BLOB, OCI_ATTR_NOCACHE,
                        OCI_DURATION_SESSION))
{
    (void) printf("FAILED: CreateTemporary() \n");
    return -1;
}

if(OCIDescriptorAlloc((dvoid*)envhp, (dvoid **)&tblob2,
                    (ub4)OCI_DTYPE_LOB, (size_t)0, (dvoid**)0))
{
    printf("OCIDescriptorAlloc failed in copy_temp_lob\n");
    return_code = -1;
}

if(OCILobCreateTemporary(svchp, errhp, tblob2, (ub2)0, SQLCS_IMPLICIT,
                        OCI_TEMP_BLOB, OCI_ATTR_NOCACHE,
                        OCI_DURATION_SESSION))
{
    (void) printf("FAILED: CreateTemporary() \n");
    return_code = -1;
}

if(OCILobFileSetName(envhp, errhp, &bfile, (text *)"AUDIO_DIR",
                    (ub2)strlen("AUDIO_DIR"),
                    (text *)"Washington_audio",
                    (ub2)strlen("Washington_audio")))
{
    printf("OCILobFileSetName FAILED\n");
    return_code = -1;
}

if(OCILobFileOpen(svchp, errhp, (OCILobLocator *) bfile, OCI_LOB_READONLY))
{
    printf("OCILobFileOpen FAILED for the bfile\n");
    return_code = -1;
}

if (OCILobOpen(svchp, errhp, (OCILobLocator *) tblob, OCI_LOB_READWRITE))
{
```

```
        printf( "OCILobOpen FAILED for temp LOB \n");
        return_code = -1;
    }
    if (OCILobOpen(svchp, errhp, (OCILobLocator *) tblob2, OCI_LOB_READWRITE ))
    {
        printf( "OCILobOpen FAILED for temp LOB \n");
        return_code = -1;
    }

    if(OCILobLoadFromFile(svchp, errhp, tblob, (OCILobLocator*)bfile,
                          (ub4)amount, (ub4)1,(ub4)1))
    {
        printf( "OCILobLoadFromFile FAILED\n");
        return_code = -1;
    }

    if (OCILobCopy(svchp, errhp, tblob2, tblob, amount, dest_offset,
                  src_offset))
    {
        printf ("FAILED: OCILobCopy in copy_temp_lobs\n");
        return -1;
    }
    /* LOB をクローズします。 */

    if (OCILobFileClose(svchp, errhp, (OCILobLocator *) bfile))
    {
        printf( "OCILobFileClose FAILED for bfile \n");
        return_code = -1;
    }
    if (OCILobClose(svchp, errhp, (OCILobLocator *) tblob))
    {
        printf( "OCILobClose FAILED for temporary LOB \n");
        return_code = -1;
    }
    if (OCILobClose(svchp, errhp, (OCILobLocator *) tblob2))
    {
        printf( "OCILobClose FAILED for temporary LOB \n");
        return_code = -1;
    }
    /* 使い終わったテンポラリ LOB を解放します。 */
    if(OCILobFreeTemporary(svchp, errhp, tblob))
    {
        printf("OCILobFreeTemporary FAILED \n");
        return_code = -1;
    }
}
```

```

if (OCILobFreeTemporary(svchp, errhp, tblob2))
{
    printf("OCILobFreeTemporary FAILED \n");
    return_code = -1;
}
return return_code;
}

```

COBOL (Pro*COBOL) : (テンポラリ) LOB の全体または一部の他へのコピー

```

IDENTIFICATION DIVISION.
PROGRAM-ID. TEMP-BLOB-COPY.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

```

```

01 USERID    PIC X(11) VALUES "SAMP/SAMP".
01 TEMP-DEST SQL-BLOB.
01 TEMP-SRC  SQL-BLOB.
01 SRC-BFILE SQL-BFILE.
01 DIR-ALIAS PIC X(30) VARYING.
01 FNAME     PIC X(30) VARYING.
01 AMT       PIC S9(9) COMP.

```

* ソースおよび宛先の位置を定義します。

```

01 SRC-POS    PIC S9(9) COMP VALUE 1.
01 DEST-POS   PIC S9(9) COMP VALUE 1.
01 ORASLNRD   PIC 9(4).

```

```

EXEC SQL INCLUDE SQLCA END-EXEC.
EXEC ORACLE OPTION (ORACA=YES) END-EXEC.
EXEC SQL INCLUDE ORACA END-EXEC.

```

PROCEDURE DIVISION.

TEMP-BLOB-COPY.

```

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

```

* BLOB ロケータの割当ておよび初期化を行います。

```

EXEC SQL ALLOCATE :TEMP-DEST END-EXEC.
EXEC SQL ALLOCATE :TEMP-SRC END-EXEC.
EXEC SQL ALLOCATE :SRC-BFILE END-EXEC.
EXEC SQL
    LOB CREATE TEMPORARY :TEMP-DEST
END-EXEC.

```

```
EXEC SQL
    LOB CREATE TEMPORARY :TEMP-SRC
END-EXEC.
```

- * ディレクトリおよびファイル情報を設定します。

```
MOVE "AUDIO_DIR" TO DIR-ALIAS-ARR.
MOVE 9 TO DIR-ALIAS-LEN.
MOVE "Washington_audio" TO FNAME-ARR.
MOVE 16 TO FNAME-LEN.
```

```
EXEC SQL
    LOB FILE SET :SRC-BFILE DIRECTORY = :DIR-ALIAS,
    FILENAME = :FNAME
END-EXEC.
```

- * ソース BFILE および宛先テンポラリ BLOB をオープンします。

```
EXEC SQL LOB OPEN :TEMP-SRC READ WRITE END-EXEC.
EXEC SQL LOB OPEN :TEMP-DEST READ WRITE END-EXEC.
EXEC SQL LOB OPEN :SRC-BFILE READ ONLY END-EXEC.
```

- * コピーする量を AMT に移動させます。

```
EXEC SQL
    LOB LOAD :AMT FROM FILE :SRC-BFILE INTO :TEMP-SRC
END-EXEC.
```

- * BFILE からテンポラリ LOB にデータをコピーします。

```
EXEC SQL
    LOB COPY :AMT FROM :TEMP-SRC AT :SRC-POS
    TO :TEMP-DEST AT :DEST-POS
END-EXEC.
```

```
EXEC SQL LOB CLOSE :TEMP-SRC END-EXEC.
EXEC SQL LOB CLOSE :TEMP-DEST END-EXEC.
EXEC SQL LOB CLOSE :SRC-BFILE END-EXEC.
EXEC SQL
```

```
    LOB FREE TEMPORARY :TEMP-SRC
END-EXEC.
```

```
EXEC SQL
    LOB FREE TEMPORARY :TEMP-DEST
END-EXEC.
```

```
EXEC SQL FREE :TEMP-SRC END-EXEC.
EXEC SQL FREE :TEMP-DEST END-EXEC.
```

```
EXEC SQL FREE :SRC-BFILE END-EXEC.  
STOP RUN.  
  
SQL-ERROR.  
EXEC SQL  
    WHENEVER SQLERROR CONTINUE  
END-EXEC.  
MOVE ORASLNR TO ORASLNRD.  
DISPLAY " ".  
DISPLAY "ORACLE ERROR DETECTED ON LINE ", ORASLNRD, ":".  
DISPLAY " ".  
DISPLAY SQLERRMC.  
EXEC SQL ROLLBACK WORK RELEASE END-EXEC.  
STOP RUN.
```

C/C++ (Pro*C/C++) : (テンポラリ) LOB の全体または一部の他へのコピー

```
#include <oci.h>  
#include <stdio.h>  
#include <sqlca.h>  
  
void Sample_Error()  
{  
    EXEC SQL WHENEVER SQLERROR CONTINUE;  
    printf("%.s\n", sqlca.sqlerrm.sqlerrml, sqlca.sqlerrm.sqlerrmc);  
    EXEC SQL ROLLBACK WORK RELEASE;  
    exit(1);  
}  
  
void copyTempLOB_proc()  
{  
    OCIBlobLocator *Temp_loc1, *Temp_loc2;  
    OCIBFileLocator *Lob_loc;  
    char *Dir = "AUDIO_DIR", *Name = "Washington_audio";  
    int Amount;  
  
    EXEC SQL WHENEVER SQLERROR DO Sample_Error();  
    /* テンポラリ LOB の割当ておよび作成を行います。*/  
    EXEC SQL ALLOCATE :Temp_loc1;  
    EXEC SQL ALLOCATE :Temp_loc2;  
    EXEC SQL LOB CREATE TEMPORARY :Temp_loc1;  
    EXEC SQL LOB CREATE TEMPORARY :Temp_loc2;  
    /* BFILE ロケータの割当ておよび初期化を行います。*/  
    EXEC SQL ALLOCATE :Lob_loc;
```

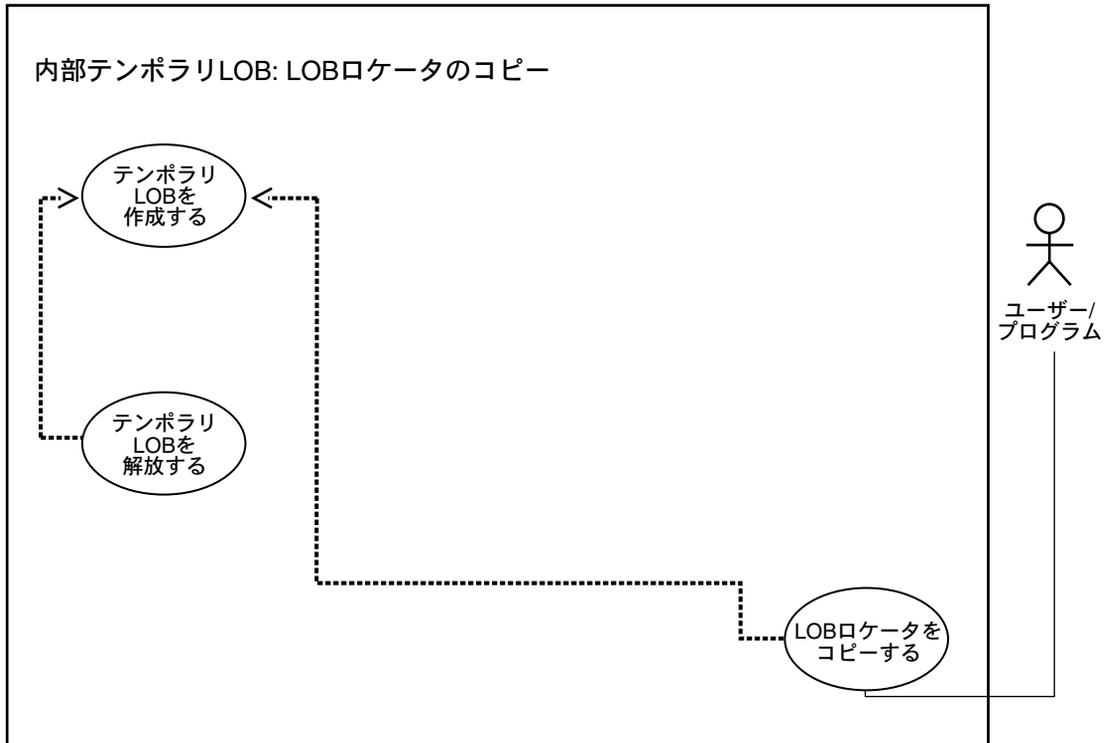
(テンポラリ) LOB の全体または一部の他へのコピー

```
EXEC SQL LOB FILE SET :Lob_loc DIRECTORY = :Dir, FILENAME = :Name;
/* LOB のオープンはおプションです。*/
EXEC SQL LOB OPEN :Lob_loc READ ONLY;
EXEC SQL LOB OPEN :Temp_loc1 READ WRITE;
EXEC SQL LOB OPEN :Temp_loc2 READ WRITE;
/* BFILE からテンポラリ LOB の 1 つに、指定した量をロードします。*/
Amount = 4096;
EXEC SQL LOB LOAD :Amount FROM FILE :Lob_loc INTO :Temp_loc1;
/* テンポラリ LOB から別のテンポラリ LOB に、指定した量をコピーします。*/
EXEC SQL LOB COPY :Amount FROM :Temp_loc1 TO :Temp_loc2;
/* LOB をオープンしている場合、その LOB をクローズする必要があります。*/
EXEC SQL LOB CLOSE :Temp_loc1;
EXEC SQL LOB CLOSE :Temp_loc2;
EXEC SQL LOB CLOSE :Lob_loc;
/* テンポラリ LOB を解放します。*/
EXEC SQL LOB FREE TEMPORARY :Temp_loc1;
EXEC SQL LOB FREE TEMPORARY :Temp_loc2;
/* ロケータが保持しているリソースを解放します。*/
EXEC SQL FREE :Temp_loc1;
EXEC SQL FREE :Temp_loc2;
EXEC SQL FREE :Lob_loc;
}

void main()
{
    char *samp = "samp/samp";
    EXEC SQL CONNECT :samp;
    copyTempLOB_proc();
    EXEC SQL ROLLBACK WORK RELEASE;
}
```

テンポラリ LOB の LOB ロケータのコピー

図 10-15 ユースケース図：テンポラリ LOB の LOB ロケータのコピー



参照： 内部テンポラリ LOB に関するすべての基本操作については、10-2 ページの「ユースケース・モデル：内部テンポラリ LOB」を参照してください。

用途

テンポラリ LOB の LOB ロケータをコピーします。

使用上の注意

ありません。

構文

各プログラム環境における構文については、次のマニュアルを参照してください。

- PL/SQL (DBMS_LOB) : 『Oracle8i PL/SQL パッケージ・プロシージャ・リファレンス』の第 18 章「DBMS_LOB」の「サブプログラムの要約」の CREATETEMPORARY プロシージャ、LOADFROMFILE プロシージャ、FREETEMPORARY プロシージャ
- C (OCI) : 『Oracle8i コール・インタフェース・プログラマーズ・ガイド』の第 15 章「OCI リレーショナル関数」の「LOB 関数」の OCILobLocatorIsInit()
- COBOL (Pro*COBOL) : 『Oracle8i Pro*COBOL プリコンパイラ・プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB ASSIGN (実行可能埋込み SQL 拡張要素)」
- C/C++ (Pro*C/C++) : 『Oracle8i Pro*C/C++ プリコンパイラ・プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB ASSIGN (実行可能埋込み SQL 拡張要素)」
- Visual Basic (OO4O) : 参照マニュアルはありません。
- Java (JDBC) : 参照マニュアルはありません。

使用例

次の例では、テンポラリ LOB ロケータを別のテンポラリ LOB へコピーします。

例

次のプログラム環境の例が示されています。

- PL/SQL (DBMS_LOB パッケージ) : テンポラリ LOB の LOB ロケータのコピー (10-108 ページ)
- C (OCI) : テンポラリ LOB の LOB ロケータのコピー (10-110 ページ)
- COBOL (Pro*COBOL) : テンポラリ LOB の LOB ロケータのコピー (10-112 ページ)
- C/C++ (Pro*C/C++) : テンポラリ LOB の LOB ロケータのコピー (10-114 ページ)
- Visual Basic (OO4O) : 今回のリリースでは例は記載されていません。
- Java (JDBC) : 今回のリリースでは例は記載されていません。

PL/SQL (DBMS_LOB パッケージ) : テンポラリ LOB の LOB ロケータのコピー

注意: PL/SQL で 1 つの LOB を別の LOB に代入するには、「=」符号を使用します。詳細は、5-2 ページの「[読取り一貫性のあるロケータ](#)」を参照してください。

```
/* 例のプロシージャ copyTempLOBLocator_proc は、DBMS_LOB パッケージの
一部ではないことに注意してください。*/

CREATE OR REPLACE PROCEDURE copyTempLOBLocator_proc (
    Lob_loc1 IN OUT CLOB, Lob_loc2 IN OUT CLOB) IS

    bufp      VARCHAR2(4);
    Amount    NUMBER := 32767;
    Src_loc   BFILE := BFILENAME('AUDIO_DIR', 'Washington_audio');

BEGIN
    DBMS_LOB.CREATETEMPORARY(Lob_loc1,TRUE);
    DBMS_LOB.CREATETEMPORARY(Lob_loc2,TRUE);
    /* 最初のテンポラリ LOB にいくつかのデータを移入します。*/
    /* ファイルのオープンは必須です。*/
    DBMS_LOB.OPEN(Src_loc,DBMS_LOB.LOB_READONLY);
    /* LOB のオープンはオプションです。*/
    DBMS_LOB.OPEN(Lob_loc1,DBMS_LOB.LOB_READWRITE);
    DBMS_LOB.OPEN(Lob_loc2,DBMS_LOB.LOB_READWRITE);
    DBMS_LOB.LOADFROMFILE(Lob_loc1,Src_loc,Amount);

    /* Lob_loc1 を Lob_loc2 に割り当て、この時点での Lob_loc1 が参照するテンポラ
    リ LOB の値のコピーを作成します。*/
    Lob_loc2 := Lob_loc1;

    /* Lob_loc1 を介していくつかのデータを LOB に書き込む場合、Lob_loc2 は新しく書き込
    まれたデータを参照しませんが、Lob_loc1 は新しいデータを参照します。*/
    /* LOB をオープンしている場合、その LOB をクローズする必要があります。*/
    DBMS_LOB.CLOSE (Src_loc);
    DBMS_LOB.CLOSE (Lob_loc1);
    DBMS_LOB.CLOSE (Lob_loc2);
    DBMS_LOB.FREETEMPORARY(Lob_loc1);
    DBMS_LOB.FREETEMPORARY(Lob_loc2);

END;
```

C (OCI) : テンポラリ LOB の LOB ロケータのコピー

/* このファンクションは、2つのテンポラリ LOB を作成します。一方を移入して、そのロケータをもう一方のテンポラリ LOB ロケータにコピーします。*/

```

sb4 copy_locators( OCIError      *errhp,
                  OCISvcCtx    *svchp,
                  OCIEnv      *envhp)
{
    sb4 return_code = 0;
    OCILobLocator *tblob;
    OCILobLocator *tblob2;
    OCILobLocator *bfile;
    ub4 amount = 4000;

    checkerr(errhp, OCIDescriptorAlloc((dvoid *)envhp, (dvoid **) &tblob,
                                       (ub4) OCI_DTYPE_LOB,
                                       (size_t) 0, (dvoid **) 0));

    checkerr(errhp, OCIDescriptorAlloc((dvoid *)envhp, (dvoid **) &tblob2,
                                       (ub4) OCI_DTYPE_LOB,
                                       (size_t) 0, (dvoid **) 0));

    checkerr(errhp, OCIDescriptorAlloc((dvoid *)envhp, (dvoid **) &bfile,
                                       (ub4) OCI_DTYPE_FILE,
                                       (size_t) 0, (dvoid **) 0));

    if(OCILobFileSetName(envhp, errhp, &bfile, (text *)"AUDIO_DIR",
                        (ub2)strlen("AUDIO_DIR"),
                        (text *)"Washington_audio",
                        (ub2)strlen("Washington_audio")))
    {
        printf("OCILobFileSetName FAILED in load_temp\n");
        return -1;
    }

    if (OCILobFileOpen(svchp, errhp, (OCILobLocator *) bfile, OCI_FILE_READONLY))
    {
        printf("OCILobFileOpen FAILED for the bfile load_temp \n");
        return -1;
    }

    if(OCILobCreateTemporary(svchp, errhp, tblob, (ub2)0, SQLCS_IMPLICIT,
                            OCI_TEMP_BLOB, OCI_ATTR_NOCACHE,
                            OCI_DURATION_SESSION))
    {

```

```
(void) printf("FAILED: CreateTemporary() \n");
return -1;
}

if(OCILobCreateTemporary(svchp, errhp, tblob2, (ub2)0, SQLCS_IMPLICIT,
                        OCI_TEMP_BLOB, OCI_ATTR_NOCACHE,
                        OCI_DURATION_SESSION))
{
(void) printf("FAILED: CreateTemporary() \n");
return -1;
}

if (OCILobOpen(svchp, errhp, (OCILobLocator *) tblob, OCI_LOB_READWRITE))
{
printf( "OCILobOpen FAILED for temp LOB \n");
return -1;
}

if (OCILobOpen(svchp, errhp, (OCILobLocator *) tblob2, OCI_LOB_READWRITE))
{
printf( "OCILobOpen FAILED for temp LOB \n");
return -1;
}

if(OCILobLoadFromFile(svchp, errhp, tblob, (OCILobLocator*)bfile,
                    (ub4)amount, (ub4)1, (ub4)1))
{
printf("OCILobLoadFromFile failed \n");
return_code = -1;
}

if(OCILobLocatorAssign(svchp, errhp, (CONST OCILobLocator *)tblob, &tblob2))
{
printf("OCILobLocatorAssign failed \n");
return_code = -1;
}

/* LOB をクローズします。 */
if (OCILobFileClose(svchp, errhp, (OCILobLocator *) bfile))
{
printf( "OCILobClose FAILED for bfile \n");
return -1;
}

checkerr(errhp, (OCILobClose(svchp, errhp, (OCILobLocator *) tblob)));
checkerr(errhp, (OCILobClose(svchp, errhp, (OCILobLocator *) tblob2)));
```

```
/* 使い終わったテンポラリ LOB を解放します。*/
if(OCILobFreeTemporary(svchp, errhp, tblob))
{
    printf("OCILobFreeTemporary FAILED \n");
    return -1;
}

if(OCILobFreeTemporary(svchp, errhp, tblob2))
{
    printf("OCILobFreeTemporary FAILED \n");
    return -1;
}
}
```

COBOL (Pro*COBOL) : テンポラリ LOB の LOB ロケータのコピー

```
IDENTIFICATION DIVISION.
PROGRAM-ID. TEMP-BLOB-COPY-LOCATOR.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

01  USERID    PIC X(11) VALUES "SAMP/SAMP".

01  TEMP-DEST    SQL-BLOB.
01  TEMP-SRC     SQL-BLOB.
01  SRC-BFILE    SQL-BFILE.
01  DIR-ALIAS    PIC X(30) VARYING.
01  FNAME        PIC X(30) VARYING.
01  AMT          PIC S9(9) COMP.

01  ORASLNRD     PIC 9(4).

EXEC SQL INCLUDE SQLCA END-EXEC.
EXEC ORACLE OPTION (ORACA=YES) END-EXEC.
EXEC SQL INCLUDE ORACA END-EXEC.

PROCEDURE DIVISION.
TEMP-BLOB-COPY-LOCATOR.

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

- * BLOB ロケータの割当ておよび初期化を行います。


```
EXEC SQL ALLOCATE :TEMP-DEST END-EXEC.
EXEC SQL ALLOCATE :TEMP-SRC END-EXEC.
EXEC SQL ALLOCATE :SRC-BFILE END-EXEC.
EXEC SQL
    LOB CREATE TEMPORARY :TEMP-DEST
END-EXEC.
EXEC SQL
    LOB CREATE TEMPORARY :TEMP-SRC
END-EXEC.
```
- * ディレクトリおよびファイル情報を設定します。


```
MOVE "AUDIO_DIR" TO DIR-ALIAS-ARR.
MOVE 9 TO DIR-ALIAS-LEN.
MOVE "Washington_audio" TO FNAME-ARR.
MOVE 16 TO FNAME-LEN.

EXEC SQL
    LOB FILE SET :SRC-BFILE DIRECTORY = :DIR-ALIAS,
    FILENAME = :FNAME
END-EXEC.
```
- * ソース BFILE および宛先テンポラリ BLOB をオープンします。


```
EXEC SQL LOB OPEN :TEMP-SRC READ WRITE END-EXEC.
EXEC SQL LOB OPEN :TEMP-DEST READ WRITE END-EXEC.
EXEC SQL LOB OPEN :SRC-BFILE READ ONLY END-EXEC.
```
- * コピーする量を AMT に移動させます。


```
MOVE 5 TO AMT.
EXEC SQL
    LOB LOAD :AMT FROM FILE :SRC-BFILE INTO :TEMP-SRC
END-EXEC.
```
- * 宛先 BLOB ロケータにソース BLOB ロケータを割り当てます。


```
EXEC SQL
    LOB ASSIGN :TEMP-SRC TO :TEMP-DEST
END-EXEC.

EXEC SQL LOB CLOSE :TEMP-SRC END-EXEC.
EXEC SQL LOB CLOSE :TEMP-DEST END-EXEC.
EXEC SQL LOB CLOSE :SRC-BFILE END-EXEC.
EXEC SQL
    LOB FREE TEMPORARY :TEMP-SRC
END-EXEC.
EXEC SQL
    LOB FREE TEMPORARY :TEMP-DEST
```

```
END-EXEC.
EXEC SQL FREE :TEMP-SRC END-EXEC.
EXEC SQL FREE :TEMP-DEST END-EXEC.
EXEC SQL FREE :SRC-BFILE END-EXEC.
STOP RUN.

SQL-ERROR.
EXEC SQL
    WHENEVER SQLERROR CONTINUE
END-EXEC.
MOVE ORASLNR TO ORASLNRD.
DISPLAY " ".
DISPLAY "ORACLE ERROR DETECTED ON LINE ", ORASLNRD, ":".
DISPLAY " ".
DISPLAY SQLERRMC.
EXEC SQL ROLLBACK WORK RELEASE END-EXEC.
STOP RUN.
```

C/C++ (Pro*C/C++) : テンポラリ LOB の LOB ロケータのコピー

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

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

void copyTempLobLocator_proc()
{
    OCIBlobLocator *Temp_loc1, *Temp_loc2;
    OCIBFileLocator *Lob_loc;
    char *Dir = "AUDIO_DIR", *Name = "Washington_audio";
    int Amount = 4096;

    EXEC SQL WHENEVER SQLERROR DO Sample_Error();

    /* テンポラリ LOB の割当ておよび作成を行います。*/
    EXEC SQL ALLOCATE :Temp_loc1;
    EXEC SQL ALLOCATE :Temp_loc2;
    EXEC SQL LOB CREATE TEMPORARY :Temp_loc1;
    EXEC SQL LOB CREATE TEMPORARY :Temp_loc2;
```

```
/* BFILE ロケータの割当ておよび初期化を行います。*/
EXEC SQL ALLOCATE :Lob_loc;
EXEC SQL LOB FILE SET :Lob_loc DIRECTORY = :Dir, FILENAME = :Name;

/* LOB のオープンはオプションです。*/
EXEC SQL LOB OPEN :Lob_loc READ ONLY;
EXEC SQL LOB OPEN :Temp_loc1 READ WRITE;
EXEC SQL LOB OPEN :Temp_loc2 READ WRITE;

/* BFILE からテンポラリ LOB に、指定した量をロードします。*/
EXEC SQL LOB LOAD :Amount FROM FILE :Lob_loc INTO :Temp_loc1;
/* Temp_loc1 を Temp_loc2 に割当て、この時点での Temp_loc1 が参照するテンポラ
   リ LOB の値のコピーを作成します。*/
EXEC SQL LOB ASSIGN :Temp_loc1 TO :Temp_loc2;

/* LOB をオープンしている場合、その LOB をクローズする必要があります。*/
EXEC SQL LOB CLOSE :Lob_loc;
EXEC SQL LOB CLOSE :Temp_loc1;
EXEC SQL LOB CLOSE :Temp_loc2;

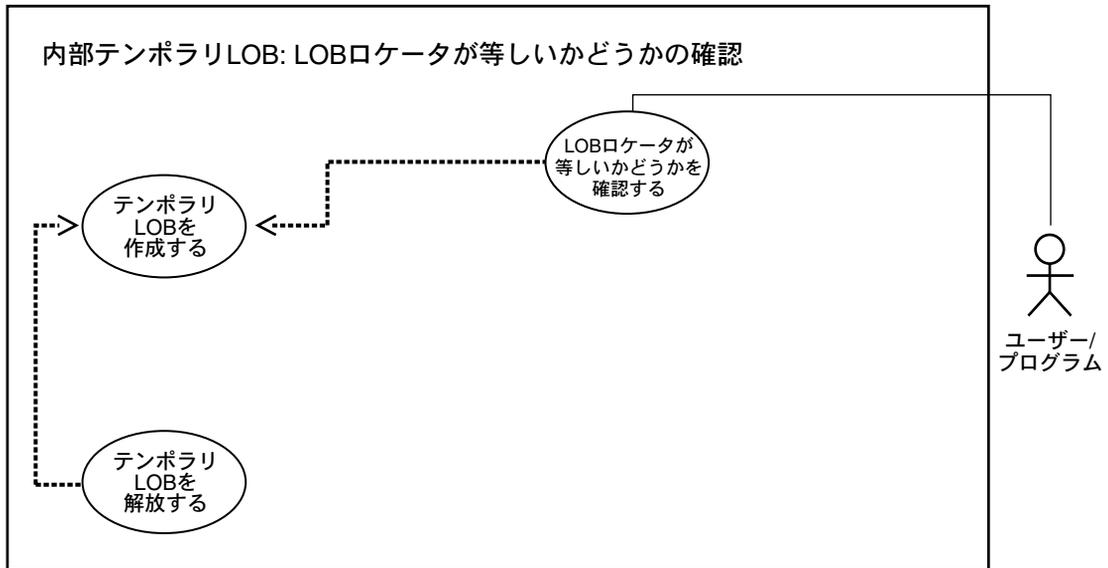
/* テンポラリ LOB を解放します。*/
EXEC SQL LOB FREE TEMPORARY :Temp_loc1;
EXEC SQL LOB FREE TEMPORARY :Temp_loc2;

/* ロケータが保持しているリソースを解放します。*/
EXEC SQL FREE :Lob_loc;
EXEC SQL FREE :Temp_loc1;
EXEC SQL FREE :Temp_loc2;
}

void main()
{
    char *samp = "samp/samp";
    EXEC SQL CONNECT :samp;
    copyTempLobLocator_proc();
    EXEC SQL ROLLBACK WORK RELEASE;
}
```

テンポラリ LOB ロケータが他と等しいかどうかの確認

図 10-16 ユースケース図：（テンポラリ）LOB ロケータが他と等しいかどうかの確認



参照： 内部テンポラリ LOB に関するすべての基本操作については、10-2 ページの「ユースケース・モデル: 内部テンポラリ LOB」を参照してください。

用途

テンポラリ LOB の LOB ロケータが他のテンポラリ LOB ロケータと等しいかどうかを確認します。

使用上の注意

2つのロケータが等しい場合、それらが同じバージョンの LOB データを参照していることを意味します (5-2 ページの「読取り一貫性のあるロケータ」を参照)。

構文

各プログラム環境で使用可能な機能のリストは、第3章「LOB プログラム環境」を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- PL/SQL (DBMS_LOB) : 参照マニュアルはありません。
- C (OCI) : 『Oracle8i コール・インタフェース・プログラマーズ・ガイド』の第15章「OCI リレーショナル関数」の「LOB 関数」の OCILobIsEqual()
- COBOL (Pro*COBOL) : 参照マニュアルはありません。
- C/C++ (Pro*C/C++) : 『Oracle8i Pro*C/C++ プリコンパイラ・プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB ASSIGN (実行可能埋込み SQL 拡張要素)」、および『Oracle8i コール・インタフェース・プログラマーズ・ガイド』の第15章「OCI リレーショナル関数」の「LOB 関数」の OCILobIsEqual()
- Visual Basic (OO4O) : 参照マニュアルはありません。
- Java (JDBC) : 参照マニュアルはありません。

使用例

ありません。

例

次のプログラム環境の例が示されています。

- PL/SQL (DBMS_LOB パッケージ) : 今回のリリースでは例は記載されていません。
- C (OCI) : テンポラリ LOB の LOB ロケータが他と等しいかどうかの確認 (10-117 ページ)
- COBOL (Pro*Cobol) : 今回のリリースでは例は記載されていません。
- C/C++ (Pro*C/C++) : テンポラリ LOB の LOB ロケータが他と等しいかどうかの確認 (10-119 ページ)
- Visual Basic (OO4O) : 今回のリリースでは例は記載されていません。
- Java (JDBC) : 今回のリリースでは例は記載されていません。

C (OCI) : テンポラリ LOB の LOB ロケータが他と等しいかどうかの確認

```

sb4 ck_isequal (OCIError      *errhp,
                OCISvcCtx    *svchp,
                OCIStmt      *stmthp,
                OCIEnv       *envhp)
{
    OCILobLocator *loc1;
    OCILobLocator *loc2;
    boolean is_equal;
    is_equal= FALSE;
    OCILobDescriptor ????
    if(OCILobCreateTemporary(svchp, errhp, loc1, (ub2)0, SQLCS_IMPLICIT,
                             OCI_TEMP_BLOB, OCI_ATTR_NOCACHE,
                             OCI_DURATION_SESSION))
    {
        (void) printf("FAILED: CreateTemporary() \n");
        return -1;
    }
    if(OCILobCreateTemporary(svchp, errhp, loc2, (ub2)0, SQLCS_IMPLICIT,
                             OCI_TEMP_BLOB, OCI_ATTR_NOCACHE,
                             OCI_DURATION_SESSION))
    {
        (void) printf("FAILED: CreateTemporary() \n");
        return -1;
    }

    if (OCILobIsEqual (envhp, loc1, loc2, &is_equal))
    {
        printf ("FAILED: OCILobLocatorIsEqual call\n");
        return -1;
    }
    if(is_equal)
    {
        fprintf (stderr, "LOB loators are equal \n");
        return -1;
    }
    }else
    {
        fprintf(stderr, "LOB locators are not equal \n");
    }
    if(OCILobFreeTemporary(svchp, errhp, loc1))
    {
        printf("FAILED: OCILobFreeTemporary for temp LOB #1\n");
        return -1;
    }
    if(OCILobFreeTemporary(svchp, errhp, loc2))

```

```

    {
        printf("FAILED: OCILobFreeTemporary for temp LOB #2\n");
        return -1;
    }
    OCILobDescriptor free????
    return 0;
}

```

C/C++ (Pro*C/C++) : テンポラリ LOB の LOB ロケータが他と等しいかどうかの確認

```

#include <sql2oci.h>
#include <stdio.h>
#include <sqlca.h>

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

void seeTempLobLocatorsAreEqual_proc()
{
    OCIBlobLocator *Temp_loc1, *Temp_loc2;
    OCIBFileLocator *Lob_loc;
    char *Dir = "AUDIO_DIR", *Name = "Washington_audio";
    int Amount = 4096;
    OCIEnv *oeh;
    int isEqual = 0;

    EXEC SQL WHENEVER SQLERROR DO Sample_Error();
    /* テンポラリ LOB の割当ておよび作成を行います。 */
    EXEC SQL ALLOCATE :Temp_loc1;
    EXEC SQL ALLOCATE :Temp_loc2;
    EXEC SQL LOB CREATE TEMPORARY :Temp_loc1;
    EXEC SQL LOB CREATE TEMPORARY :Temp_loc2;
    /* BFILE ロケータの割当ておよび初期化を行います。 */
    EXEC SQL ALLOCATE :Lob_loc;
    EXEC SQL LOB FILE SET :Lob_loc DIRECTORY = :Dir, FILENAME = :Name;
    /* LOB のオープンはオプションです。 */
    EXEC SQL LOB OPEN :Lob_loc READ ONLY;
    EXEC SQL LOB OPEN :Temp_loc1 READ WRITE;
}

```

テンポラリ LOB ロケータが他と等しいかどうかの確認

```
EXEC SQL LOB OPEN :Temp_loc2 READ WRITE;

/* BFILE からテンポラリ LOB の 1 つに、指定した量をロードします。*/
EXEC SQL LOB LOAD :Amount FROM FILE :Lob_loc INTO :Temp_loc1;
/* OCI 環境ハンドルを取り出します。*/
(void) SQLEnvGet(SQL_SINGLE_RCTX, &oeh);

/* 埋込み SQL を使用して、Temp_loc1 を Temp_loc2 に割り当てます。*/
EXEC SQL LOB ASSIGN :Temp_loc1 TO :Temp_loc2;

/* テンポラリ LOB が等しいかどうかを判断します。*/
(void) OCILobIsEqual(oeh, Temp_loc1, Temp_loc2, &isEqual);

/* この場合、isEqual は 0 (ゼロ) (FALSE) である必要があります。*/
printf("Locators %s equal\n", isEqual ? "are" : "are not");

/* c ポインタ割当てを使用して、Temp_loc1 を Temp_loc2 に割り当てます。*/
Temp_loc2 = Temp_loc1;

/* LOB が等しいかどうかを判断し直します。*/
(void) OCILobIsEqual(oeh, Temp_loc1, Temp_loc2, &isEqual);

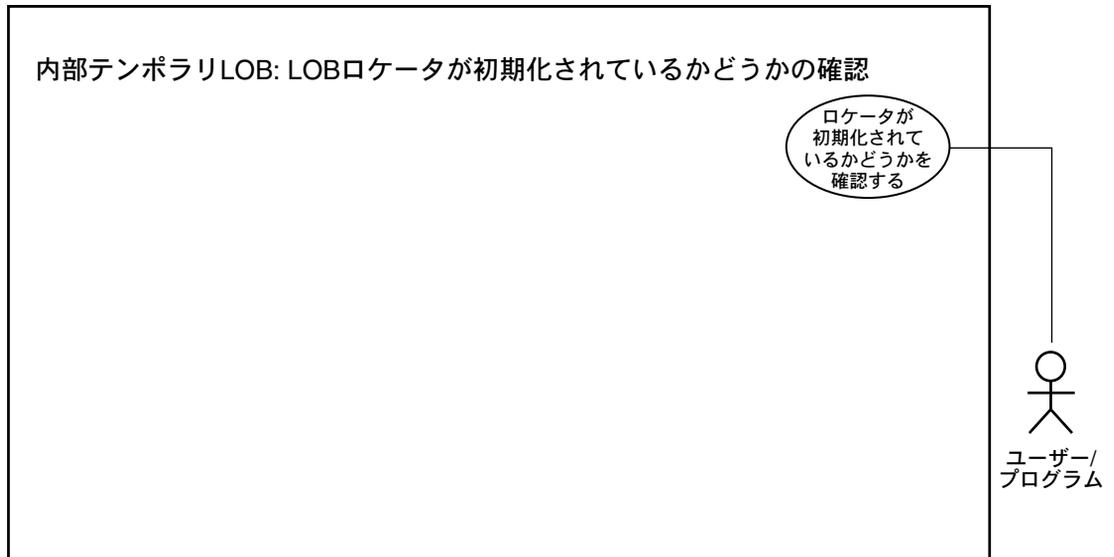
/* この場合、isEqual の値は 1 (TRUE) である必要があります。*/
printf("Locators %s equal\n", isEqual ? "are" : "are not");

/* LOB をオープンしている場合、その LOB をクローズする必要があります。*/
EXEC SQL LOB CLOSE :Lob_loc;
/* Temp_loc1 と Temp_loc2 は等しいため、一方をクローズまたは解放すると、
   暗黙的にもう一方にも同じ操作が行われます。*/
EXEC SQL LOB CLOSE :Temp_loc1;
EXEC SQL LOB FREE TEMPORARY :Temp_loc1;
/* ロケータが保持しているリソースを解放します。*/
EXEC SQL FREE :Lob_loc;
EXEC SQL FREE :Temp_loc1;
}

void main()
{
    char *samp = "samp/samp";
    EXEC SQL CONNECT :samp;
    seeTempLobLocatorsAreEqual_proc();
    EXEC SQL ROLLBACK WORK RELEASE;
}
```

テンポラリ LOB の LOB ロケータが初期化されているかどうかの確認

図 10-17 ユースケース図：テンポラリ LOB の LOB ロケータが初期化されているかどうかの確認



参照： 内部テンポラリ LOB に関するすべての基本操作については、10-2 ページの「ユースケース・モデル: 内部テンポラリ LOB」を参照してください。

用途

テンポラリ LOB の LOB ロケータが初期化されているかどうかを確認します。

使用上の注意

ありません。

構文

各プログラム環境で使用可能な機能のリストは、第3章「LOB プログラム環境」を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- PL/SQL (DBMS_LOB) : 参照マニュアルはありません。
- C (OCI) : 『Oracle8i コール・インタフェース・プログラマーズ・ガイド』の第15章「OCI リレーショナル関数」の「LOB 関数」の OCILobLocatorIsInit()
- COBOL (Pro*COBOL) : 参照マニュアルはありません。
- C/C++ (Pro*C/C++) : 『Oracle8i Pro*C/C++ プリコンパイラ・プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB CREATE TEMPORARY (実行可能埋込み SQL 拡張要素)」、および『Oracle8i コール・インタフェース・プログラマーズ・ガイド』の第15章「OCI リレーショナル関数」の「LOB 関数」の OCILobLocatorIsInit()
- Visual Basic (OO4O) : 参照マニュアルはありません。
- Java (JDBC) : 参照マニュアルはありません。

使用例

この関数は、LOB ロケータを取得し、初期化されているかどうかをチェックします。初期化されている場合は、「LOB is initialized」というメッセージが出力されます。初期化されていない場合は、「LOB is not initialized」と出力されます。

例

次のプログラム環境の例が示されています。

- PL/SQL (DBMS_LOB パッケージ) : 今回のリリースでは例は記載されていません。
- C (OCI) : テンポラリ LOB の LOB ロケータが初期化されているかどうかの確認 (10-122 ページ)
- COBOL (Pro*Cobol) : 今回のリリースでは例は記載されていません。
- C/C++ (Pro*C/C++) : テンポラリ LOB の LOB ロケータが初期化されているかどうかの確認 (10-123 ページ)
- Visual Basic (OO4O) : 今回のリリースでは例は記載されていません。
- Java (JDBC) : 今回のリリースでは例は記載されていません。

C (OCI) : テンポラリ LOB の LOB ロケータが初期化されているかどうかの確認

```
/* このファンクションは、LOB ロケータを取り、LOB ロケータが初期化されているかどうかを  
   チェックします。初期化されている場合、「LOB is initialized」というメッセージが  
   出力されます。初期化されていない場合は、「LOB is not initialized」というメッセージ  
   が出力されます。このファンクションは、成功した場合は0 (ゼロ) を戻し、失敗した場合は  
   -1 を戻します。*/
```

```
sb4 ck_isinit (OCIlobLocator *lob_loc,  
              OCIError      *errhp,  
              OCISvcCtx     *svchp,  
              OCISstmt      *stmthp,  
              OCIEnv        *envhp)  
{  
    boolean is_init;  
    is_init= FALSE;  
    if (OCIlobLocatorIsInit(envhp, errhp, lob_loc, &is_init))  
    {  
        printf ("FAILED: OCIlobLocatorIsInit call\n");  
        return -1;  
    }  
    if(is_init)  
    {  
        printf ("LOB is initialized\n");  
    }else  
    {  
        printf("LOB is not initialized\n");  
    }  
    return 0;  
}
```

C/C++ (Pro*C/C++) : テンポラリ LOB の LOB ロケータが初期化されているかどうかの確認

```
#include <sql2oci.h>  
#include <stdio.h>  
#include <sqlca.h>  
  
void Sample_Error()  
{  
    EXEC SQL WHENEVER SQLERROR CONTINUE;  
    printf("%.*s\n", sqlca.sqlerrm.sqlerrml, sqlca.sqlerrm.sqlerrmc);  
    EXEC SQL ROLLBACK WORK RELEASE;  
    exit(1);  
}
```

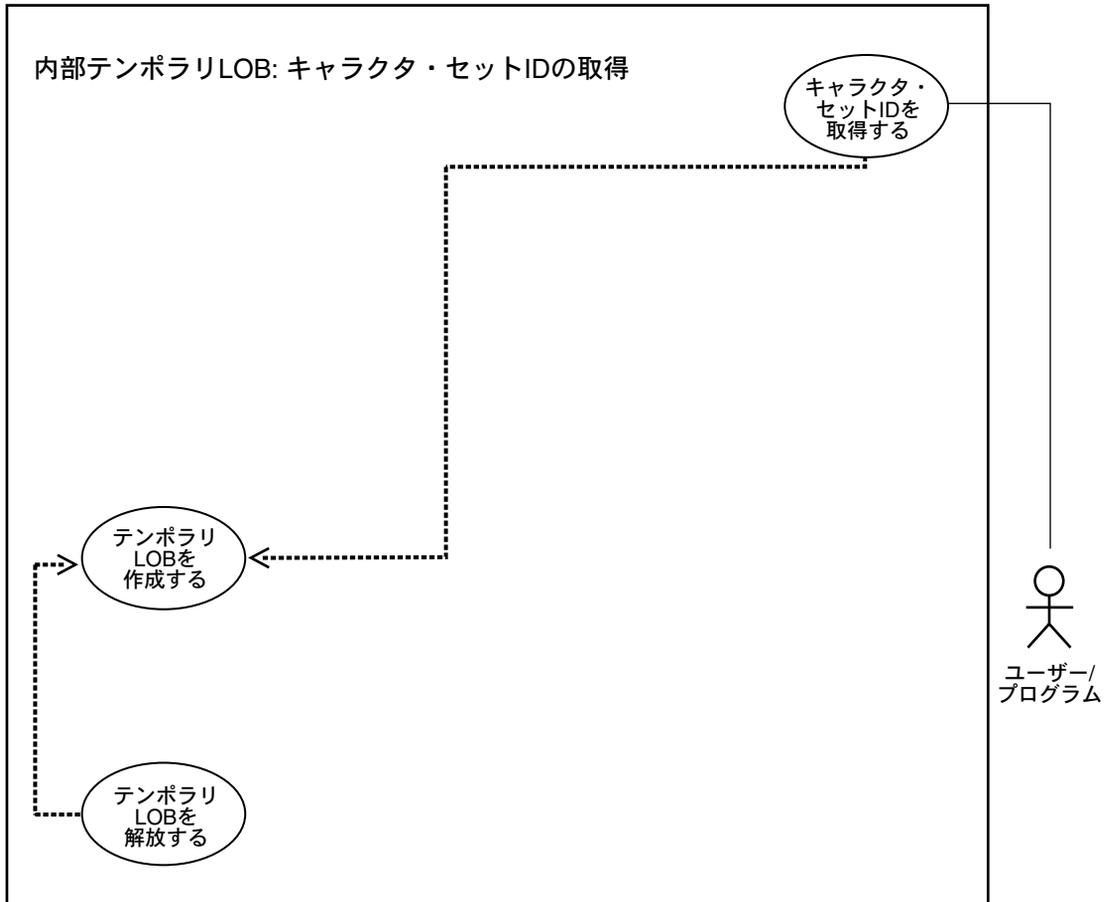
```
}
void tempLobLocatorIsInit_proc()
{
    OCIBlobLocator *Temp_loc;
    OCIEnv *oeh;
    OCIError *err;
    boolean isInitialized = 0;

    EXEC SQL WHENEVER SQLERROR DO Sample_Error();
    EXEC SQL ALLOCATE :Temp_loc;
    EXEC SQL LOB CREATE TEMPORARY :Temp_loc;
    /* SQLLIB ルーチンを使用して、OCI 環境ハンドルを取得します。*/
    (void) SQLEnvGet(SQL_SINGLE_RCTX, &oeh);
    /* OCI エラー・ハンドルを割り当てます。*/
    (void) OCIHandleAlloc((dvoid *)oeh, (dvoid **)&err,
                          (ub4)OCI_HTYPE_ERROR, (ub4)0, (dvoid **)0);
    /* OCI を使用して、ロケータが初期化されているかどうかを判断します。*/
    (void) OCILobLocatorIsInit(oeh, err, Temp_loc, &isInitialized);
    if (isInitialized)
        printf("Locator is initialized\n");
    else
        printf("Locator is not initialized\n");
    /* この例では、ロケータは初期化されていることに注意してください。*/
    /* OCI エラー・ハンドルの割当てを解除します。*/
    (void) OCIHandleFree(err, OCI_HTYPE_ERROR);
    /* テンポラリ LOB を解放します。*/
    EXEC SQL LOB FREE TEMPORARY :Temp_loc;
    /* ロケータが保持しているリソースを解放します。*/
    EXEC SQL FREE :Temp_loc;
}

void main()
{
    char *samp = "samp/samp";
    EXEC SQL CONNECT :samp;
    tempLobLocatorIsInit_proc();
    EXEC SQL ROLLBACK WORK RELEASE;
}
```

テンポラリ LOB のキャラクタ・セット ID の取得

図 10-18 ユースケース図：テンポラリ LOB のキャラクタ・セット ID の取得



参照： 内部テンポラリ LOB に関するすべての基本操作については、10-2 ページの「ユースケース・モデル：内部テンポラリ LOB」を参照してください。

用途

テンポラリ LOB のキャラクタ・セット ID を取得します。

使用上の注意

ありません。

構文

各プログラム環境で使用可能な機能のリストは、[第3章「LOB プログラム環境」](#)を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- PL/SQL (DBMS_LOB) : 参照マニュアルはありません。
- C (OCI) : 『Oracle8i コール・インタフェース・プログラマーズ・ガイド』の第15章「OCI リレーショナル関数」の「LOB 関数」の OCILobCharSetId()
- COBOL (Pro*COBOL) : 参照マニュアルはありません。
- C/C++ (Pro*C/C++) : 参照マニュアルはありません。
- Visual Basic (OO4O) : 参照マニュアルはありません。
- Java (JDBC) : 参照マニュアルはありません。

使用例

この関数は、LOB ロケータを取得し、LOB のキャラクタ・セット ID を出力します。

例

次のプログラム環境の例が示されています。

- PL/SQL (DBMS_LOB パッケージ) : 今回のリリースでは例は記載されていません。
- [C \(OCI\) : テンポラリ LOB のキャラクタ・セット ID の取得 \(10-126 ページ\)](#)
- COBOL (Pro*Cobol) : 今回のリリースでは例は記載されていません。
- C/C++ (Pro*C/C++) : 今回のリリースでは例は記載されていません。
- Visual Basic (OO4O) : 今回のリリースでは例は記載されていません。
- Java (JDBC) : 今回のリリースでは例は記載されていません。

C (OCI) : テンポラリ LOB のキャラクタ・セット ID の取得

/* このファンクションは、LOB ロケータを取り、LOB のキャラクタ・セット ID を出力します。
このファンクションは、成功した場合は 0 (ゼロ) を戻し、失敗した場合は -1 を戻します。*/

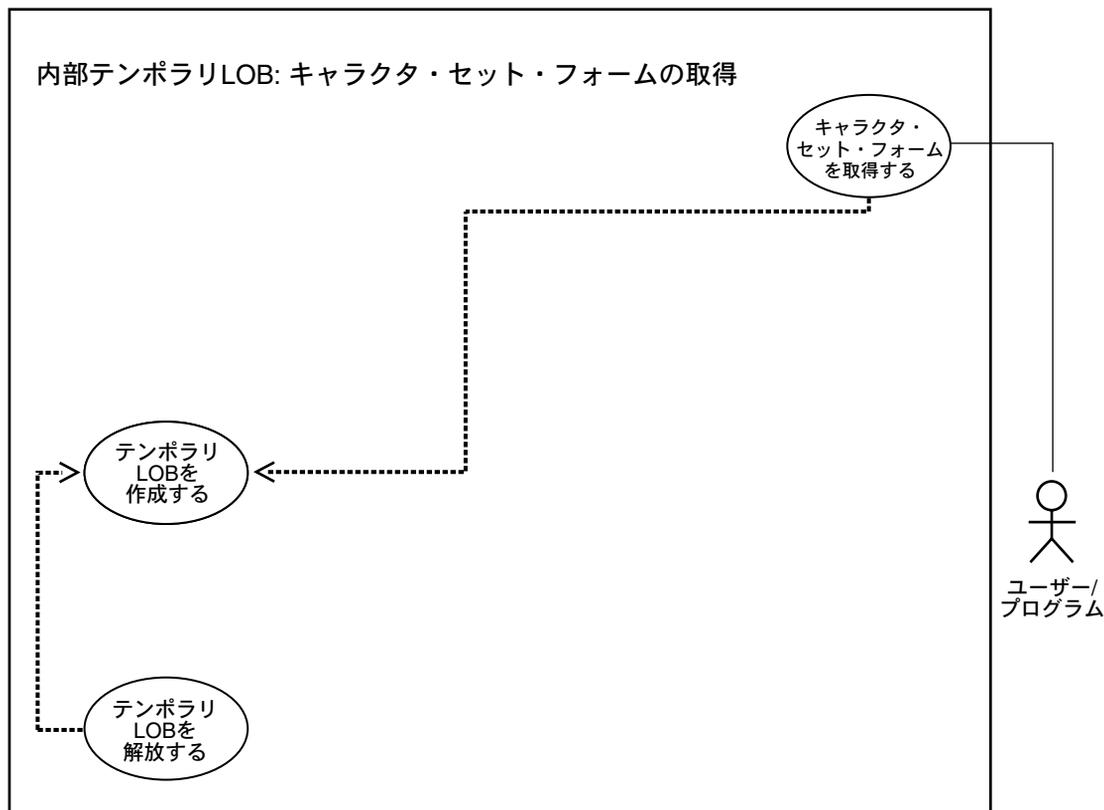
```
sb4 get_charsetid (OCIlobLocator *lob_loc,
                  OCIError      *errhp,
                  OCISvcCtx     *svchp,
                  OCIStmt      *stmthp,
                  OCIEnv       *envhp)
{
    ub2 charsetid=199;
    if(OCIlobCreateTemporary(svchp, errhp, lob_loc, (ub2)0, SQLCS_IMPLICIT,
                             OCI_TEMP_CLOB, OCI_ATTR_NOCACHE,
                             OCI_DURATION_SESSION))
    {
        (void) printf("FAILED: CreateTemporary() \n");
        return -1;
    }

    if (OCIlobCharSetId(envhp, errhp, lob_loc, &charsetid))
    {
        printf ("FAILED: OCIlobCharSetId call\n");
        return -1;
    }
    fprintf (stderr, "LOB charsetid is %d\n", charsetid);
    if (OCIlobFreeTemporary(svchp, errhp, lob_loc))
    {
        printf("FAILED: OCIlobFreeTemporary \n");
        return -1;
    }

    return 0;
}
```

テンポラリ LOB のキャラクタ・セット・フォームの取得

図 10-19 ユースケース図：テンポラリ LOB のキャラクタ・セット・フォームの取得



参照： 内部テンポラリ LOB に関するすべての基本操作については、10-2 ページの「ユースケース・モデル：内部テンポラリ LOB」を参照してください。

用途

テンポラリ LOB のキャラクタ・セット・フォームを取得します。

使用上の注意

ありません。

構文

各プログラム環境で使用可能な機能のリストは、[第3章「LOB プログラム環境」](#)を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- PL/SQL (DBMS_LOB) : 参照マニュアルはありません。
- C (OCI) : 『Oracle8i コール・インタフェース・プログラマーズ・ガイド』の第15章「OCI リレーショナル関数」の「LOB 関数」の OCILobCharSetForm()
- COBOL (Pro*COBOL) : 参照マニュアルはありません。
- C/C++ (Pro*C/C++) : 参照マニュアルはありません。
- Visual Basic (OO4O) : 参照マニュアルはありません。
- Java (JDBC) : 参照マニュアルはありません。

使用例

この関数は、LOB ロケータを取得し、LOB のキャラクタ・セット・フォームを出力します。

例

次のプログラム環境の例が示されています。

- PL/SQL (DBMS_LOB パッケージ) : 今回のリリースでは例は記載されていません。
- [C \(OCI\) : テンポラリ LOB のキャラクタ・セット・フォームの取得 \(10-129 ページ\)](#)
- COBOL (Pro*Cobol) : 今回のリリースでは例は記載されていません。
- C/C++ (Pro*C/C++) : 今回のリリースでは例は記載されていません。
- Visual Basic (OO4O) : 今回のリリースでは例は記載されていません。
- Java (JDBC) : 今回のリリースでは例は記載されていません。

C (OCI) : テンポラリ LOB のキャラクタ・セット・フォームの取得

/* このファンクションは、LOB ロケータを取り、LOB のキャラクタ・セット・フォームを出力します。
このファンクションは、成功した場合は 0 (ゼロ) を戻し、失敗した場合は -1 を戻します。*/

```

sb4 get_charsetform (OCIlobLocator *lob_loc,
                    OCIError      *errhp,
                    OCISvcCtx     *svchp,
                    OCISstmt     *stmthp,
                    OCIEnv        *envhp)
{
    ub1 charsetform = 0;
    if (OCIlobCreateTemporary(svchp, errhp, lob_loc, (ub2)0,
                              SQLCS_IMPLICIT, OCI_TEMP_CLOB, OCI_ATTR_NOCACHE,
                              OCI_DURATION_SESSION))
    {
        (void) printf("FAILED: CreateTemporary() \n");
        return -1;
    }

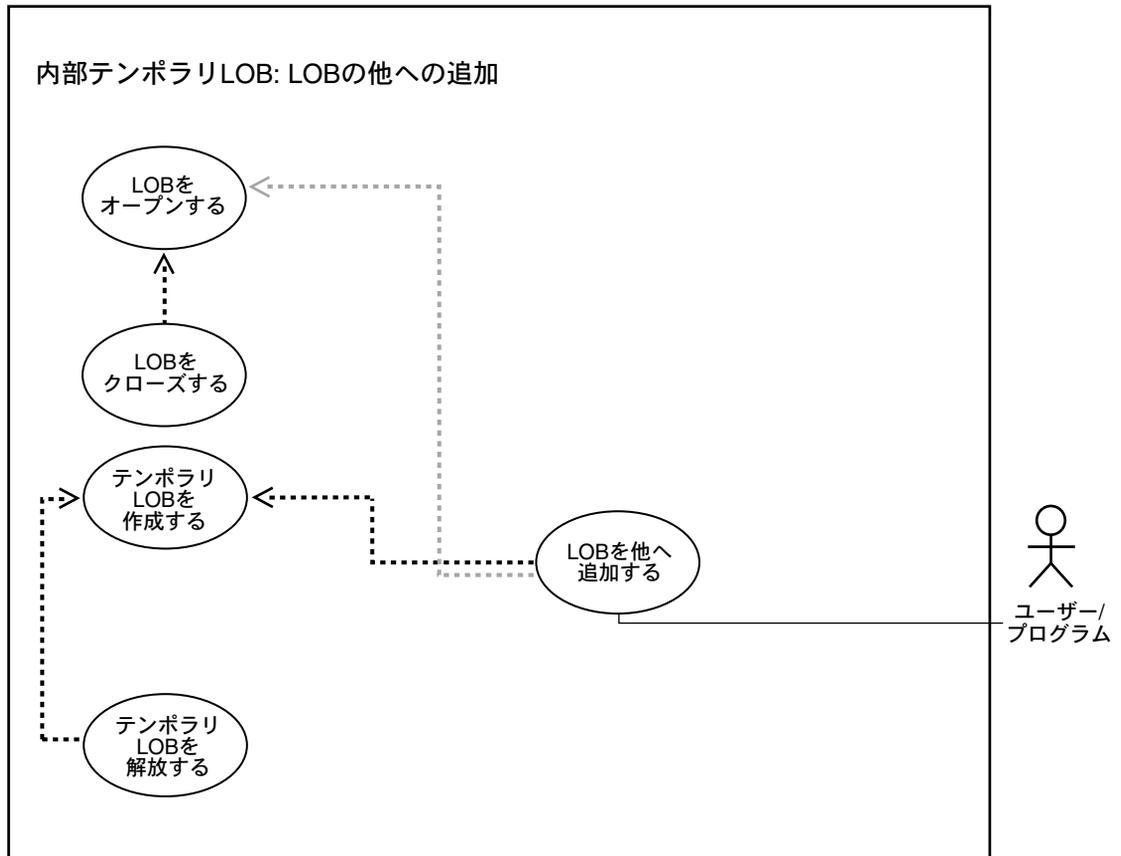
    if (OCIlobCharSetForm(envhp, errhp, lob_loc, &charsetform))
    {
        printf ("FAILED: OCIlobCharSetForm call\n");
        return -1;
    }
    fprintf (stderr, "LOB charsetform is %d\n", charsetform);

    if (OCIlobFreeTemporary (svchp, errhp, lob_loc))
    {
        printf("FAILED: OCIlobFreeTemporary \n");
        return -1;
    }
    return 0;
}

```

(テンポラリ) LOB の他への追加

図 10-20 ユースケース図：(テンポラリ) LOB の他への追加



参照： 内部テンポラリ LOB に関するすべての基本操作については、10-2 ページの「ユースケース・モデル: 内部テンポラリ LOB」を参照してください。

用途

(テンポラリ) LOB を他へ追加します。

使用上の注意

ありません。

構文

各プログラム環境で使用可能な機能のリストは、[第3章「LOB プログラム環境」](#)を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- PL/SQL (DBMS_LOB) : 『Oracle8i PL/SQL パッケージ・プロシージャ・リファレンス』の第18章「DBMS_LOB」の「サブプログラムの要約」の APPEND ファンクション
- C (OCI) : 『Oracle8i コール・インタフェース・プログラマーズ・ガイド』の第15章「OCI リレーショナル関数」の「LOB 関数」の OCILobAppend()
- COBOL (Pro*COBOL) : 『Oracle8i Pro*COBOL プリコンパイラ・プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB APPEND (実行可能埋込み SQL 拡張要素)」
- C/C++ (Pro*C/C++) : 『Oracle8i Pro*C/C++ プリコンパイラ・プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB APPEND (実行可能埋込み SQL 拡張要素)」
- Visual Basic (OO4O) : 参照マニュアルはありません。
- Java (JDBC) : 参照マニュアルはありません。

使用例

この例は、サウンドの 1 セグメントを他へ追加する作業を行います。波形の処理が可能なサウンド編集ツールを使用します。

例

次のプログラム環境の例が示されています。

- [PL/SQL \(DBMS_LOB パッケージ\) : \(テンポラリ\) LOB の他への追加 \(10-132 ページ\)](#)
- [C \(OCI\) : \(テンポラリ\) LOB の他への追加 \(10-133 ページ\)](#)
- [COBOL \(Pro*COBOL\) : \(テンポラリ\) LOB の他への追加 \(10-136 ページ\)](#)
- [C/C++ \(Pro*C/C++\) : \(テンポラリ\) LOB の他への追加 \(10-138 ページ\)](#)
- Visual Basic (OO4O) : 今回のリリースでは例は記載されていません。
- Java (JDBC) : 今回のリリースでは例は記載されていません。

PL/SQL (DBMS_LOB パッケージ) : (テンポラリ) LOB の他への追加

```

/* 例のプロシージャ appendTempLOB_proc は、DBMS_LOB パッケージの
一部ではないことに注意してください。*/
CREATE OR REPLACE PROCEDURE appendTempLOB_proc IS
    Dest_loc2 CLOB;
    Dest_loc  CLOB;
    Amount    NUMBER;
    Src_loc   BFILE := BFILENAME('AUDIO_DIR', 'Washington_audio');
BEGIN
    DBMS_LOB.CREATETEMPORARY(Dest_loc, TRUE);
    DBMS_LOB.CREATETEMPORARY(Dest_loc2, TRUE);
    DBMS_LOB.OPEN(Dest_loc, DBMS_LOB.LOB_READWRITE);
    DBMS_LOB.OPEN(Dest_loc2, DBMS_LOB.LOB_READWRITE);
    DBMS_LOB.OPEN(Src_loc, DBMS_LOB.LOB_READWRITE);
    Amount := 32767;
    DBMS_LOB.LOADFROMFILE(Dest_loc, Src_loc, Amount);
    DBMS_LOB.LOADFROMFILE(Dest_loc2, Src_loc, Amount);
    DBMS_LOB.APPEND(Dest_loc, Dest_loc2);
    /* テンポラリ LOB をクローズし、その後解放します。*/
    DBMS_LOB.CLOSE(Dest_loc);
    DBMS_LOB.CLOSE(Dest_loc2);
    DBMS_LOB.CLOSE(Src_loc);
    DBMS_LOB.FREETEMPORARY(Dest_loc);
    DBMS_LOB.FREETEMPORARY(Dest_loc2);
END;

```

C (OCI) : (テンポラリ) LOB の他への追加

/* このファンクションは、2つのテンポラリ LOB ロケータを取り、2番目の LOB を 1番目の LOB に追加します。このファンクションは、成功した場合は 0 (ゼロ) を戻し、失敗した場合は -1 を戻します。*/

```

sb4 append_temp_lobs (OCIError      *errhp,
                    OCISvcCtx      *svchp,
                    OCIStmt        *stmthp,
                    OCIEnv         *envhp)
{
    OCILobLocator *tblob;
    OCILobLocator *tblob2;
    OCILobLocator *bfile;
    ub4 amt = 4000;
    sb4 return_code = 0;

    printf("in append \n");
    if(OCIDescriptorAlloc((dvoid *)envhp, (dvoid **) &tblob,

```

```

        (ub4) OCI_DTYPE_LOB,
        (size_t) 0, (dvoid **) 0))
    {
        printf("OCIDescriptor Alloc FAILED in print_length\n");
        return -1;
    }
if(OCIDescriptorAlloc((dvoid *)envhp, (dvoid **) &tblob2,
    (ub4) OCI_DTYPE_LOB,
    (size_t) 0, (dvoid **) 0))
    {
        printf("OCIDescriptor Alloc FAILED in print_length\n");
        return -1;
    }

if(OCIDescriptorAlloc((dvoid *)envhp, (dvoid **) &bfile,
    (ub4) OCI_DTYPE_FILE,
    (size_t) 0, (dvoid **) 0))
    {
        printf("OCIDescriptor Alloc FAILED in print_length\n");
        return -1;
    }

/* BFILE が Washington_audio ファイルを指すように設定します。*/
if(OCILobFileSetName(envhp, errhp, &bfile, (text *)"AUDIO_DIR",
    (ub2)strlen("AUDIO_DIR"),
    (text *)"Washington_audio",
    (ub2)strlen("Washington_audio")))
    {
        printf("OCILobFileSetName FAILED\n");
        return -1;
    }

if (OCILobFileOpen(svchp, errhp, (OCILobLocator *) bfile, OCI_LOB_READONLY))
    {
        printf("OCILobFileOpen FAILED for the bfile\n");
        return_code = -1;
    }

if(OCILobCreateTemporary(svchp, errhp, tblob, (ub2)0, SQLCS_IMPLICIT,
    OCI_TEMP_CLOB, OCI_ATTR_NOCACHE,
    OCI_DURATION_SESSION))
    {
        (void) printf("FAILED: CreateTemporary() \n");
        return_code = -1;
    }

if(OCILobCreateTemporary(svchp, errhp, tblob2, (ub2)0, SQLCS_IMPLICIT,
```

```

OCI_TEMP_CLOB, OCI_ATTR_NOCACHE,
OCI_DURATION_SESSION)
{
    (void) printf("FAILED: CreateTemporary() \n");
    return_code = -1;
}

/* LOB をオープンします。 */
if (OCILobOpen(svchp, errhp, (OCILobLocator *) tblob, OCI_LOB_READWRITE))
{
    printf( "OCILobOpen FAILED for temp LOB tblob \n");
    return_code = -1;
}

if (OCILobOpen(svchp, errhp, (OCILobLocator *) tblob2, OCI_LOB_READWRITE))
{
    printf( "OCILobOpen FAILED for temp LOB, tblob2 \n");
    return_code = -1;
}

/* ソース・テンポラリ LOB にいくつかのデータを移入します。 */

if(OCILobLoadFromFile(svchp, errhp, tblob, (OCILobLocator*)bfile,
                      (ub4)amt, (ub4)1, (ub4)1))
{
    printf( "OCILobLoadFromFile FAILED\n");
    return_code = -1;
}

/* ソース LOB を宛先テンポラリ LOB に追加します。 */
if (OCILobAppend(svchp, errhp, tblob2, tblob))
{
    printf( "FAILED: OCILobAppend in append_temp_lobs\n");
    return_code = -1;
}
else
{
    printf("Append succeeded\n");
}

if (OCILobFreeTemporary(svchp, errhp, tblob))
{
    printf("FAILED: OCILobFreeTemporary \n");
    return_code = -1;
}
if (OCILobFreeTemporary(svchp, errhp, tblob2))
{
    printf("FAILED: OCILobFreeTemporary\n");
}

```

```
        return_code = -1;
    }
    return return_code;
}
```

COBOL (Pro*COBOL) : (テンポラリ) LOB の他への追加

```
IDENTIFICATION DIVISION.
PROGRAM-ID. APPEND-TEMP-BLOB.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
```

* ユーザー名およびパスワードを定義します。

```
01  USERID    PIC X(11) VALUES "SAMP/SAMP".
```

* テンポラリ LOB およびソース BFILE を定義します。

```
01  TEMP-BLOB1    SQL-BLOB.
01  TEMP-BLOB2    SQL-BLOB.
01  SRC-BFILE     SQL-BFILE.
01  AMT           PIC S9(9) COMP.
01  DIR-ALIAS     PIC X(30) VARYING.
01  FNAME        PIC X(30) VARYING.
```

* BFILE 内のソース位置を定義します。

```
01  SRC-POS      PIC S9(9) COMP.
```

* エラーの場合の行番号を定義します。

```
01  ORASLNRD     PIC 9(4).
```

```
EXEC SQL INCLUDE SQLCA END-EXEC.
EXEC ORACLE OPTION (ORACA=YES) END-EXEC.
EXEC SQL INCLUDE ORACA END-EXEC.
```

```
PROCEDURE DIVISION.
```

```
APPEND-TEMP-BLOB.
```

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

* BLOB ロケータの割当ておよび初期化を行います。

```
EXEC SQL ALLOCATE :TEMP-BLOB1 END-EXEC.
EXEC SQL ALLOCATE :TEMP-BLOB2 END-EXEC.
EXEC SQL ALLOCATE :SRC-BFILE END-EXEC.
EXEC SQL
    LOB CREATE TEMPORARY :TEMP-BLOB1
```

```
END-EXEC.  
EXEC SQL  
    LOB CREATE TEMPORARY :TEMP-BLOB2  
END-EXEC.
```

- * ディレクトリおよびファイル情報を設定します。

```
MOVE "AUDIO_DIR" TO DIR-ALIAS-ARR.  
MOVE 9 TO DIR-ALIAS-LEN.  
MOVE "Washington_audio" TO FNAME-ARR.  
MOVE 16 TO FNAME-LEN.
```

```
EXEC SQL  
    LOB FILE SET :SRC-BFILE DIRECTORY = :DIR-ALIAS,  
    FILENAME = :FNAME  
END-EXEC.
```

- * ソース BFILE および宛先テンポラリ BLOB をオープンします。

```
EXEC SQL LOB OPEN :TEMP-BLOB2 READ WRITE END-EXEC.  
EXEC SQL LOB OPEN :TEMP-BLOB1 READ WRITE END-EXEC.  
EXEC SQL LOB OPEN :SRC-BFILE READ ONLY END-EXEC.  
DISPLAY "LOBs opened."
```

- * コピーする量を AMT に移動させます。

```
MOVE 5 TO AMT.  
MOVE 1 TO SRC-POS.  
EXEC SQL  
    LOB LOAD :AMT FROM FILE :SRC-BFILE  
    AT :SRC-POS INTO :TEMP-BLOB1  
END-EXEC.
```

```
ADD 1 TO AMT GIVING SRC-POS.
```

```
EXEC SQL  
    LOB LOAD :AMT FROM FILE :SRC-BFILE  
    AT :SRC-POS INTO :TEMP-BLOB2  
END-EXEC.  
DISPLAY "Temporary LOBs loaded".
```

```
EXEC SQL  
    LOB APPEND :TEMP-BLOB2 TO :TEMP-BLOB1  
END-EXEC.  
DISPLAY "LOB APPEND complete."
```

```
EXEC SQL  
    LOB FREE TEMPORARY :TEMP-BLOB1  
END-EXEC.  
EXEC SQL  
    LOB FREE TEMPORARY :TEMP-BLOB2
```

```
END-EXEC.  
EXEC SQL FREE :TEMP-BLOB1 END-EXEC.  
EXEC SQL FREE :TEMP-BLOB2 END-EXEC.  
EXEC SQL FREE :SRC-BFILE END-EXEC.  
STOP RUN.  
  
SQL-ERROR.  
EXEC SQL  
    WHENEVER SQLERROR CONTINUE  
END-EXEC.  
MOVE ORASLNR TO ORASLNRD  
DISPLAY " ".  
DISPLAY "ORACLE ERROR DETECTED ON LINE ", ORASLNRD, ":".  
DISPLAY " ".  
DISPLAY SQLERRMC.  
EXEC SQL ROLLBACK WORK RELEASE END-EXEC.  
STOP RUN.
```

C/C++ (Pro*C/C++) : (テンポラリ) LOB の他への追加

```
#include <oci.h>  
#include <stdio.h>  
#include <sqlca.h>  
  
void Sample_Error()  
{  
    EXEC SQL WHENEVER SQLERROR CONTINUE;  
    printf("%.*s\n", sqlca.sqlerrm.sqlerrml, sqlca.sqlerrm.sqlerrmc);  
    EXEC SQL ROLLBACK WORK RELEASE;  
    exit(1);  
}  
  
void appendTempLOB_proc()  
{  
    OCIBlobLocator *Temp_loc1, *Temp_loc2;  
    OCIBFileLocator *Lob_loc;  
    char *Dir = "AUDIO_DIR", *Name = "Washington_audio";  
    int Amount = 2048;  
    int Position = 1;  
  
    EXEC SQL WHENEVER SQLERROR DO Sample_Error();  
    /* テンポラリ LOB の割当ておよび作成を行います。 */  
    EXEC SQL ALLOCATE :Temp_loc1;  
    EXEC SQL ALLOCATE :Temp_loc2;  
    EXEC SQL LOB CREATE TEMPORARY :Temp_loc1;  
    EXEC SQL LOB CREATE TEMPORARY :Temp_loc2;
```

```
/* BFILE ロケータの割当ておよび初期化を行います。*/
EXEC SQL ALLOCATE :Lob_loc;
EXEC SQL LOB FILE SET :Lob_loc DIRECTORY = :Dir, FILENAME = :Name;

/* LOB のオープンはオプションです。*/
EXEC SQL LOB OPEN :Lob_loc READ ONLY;
EXEC SQL LOB OPEN :Temp_loc1 READ WRITE;
EXEC SQL LOB OPEN :Temp_loc2 READ WRITE;

/* BFILE から 1 番目のテンポラリ LOB に、指定した量をロードします。*/
EXEC SQL LOB LOAD :Amount FROM FILE :Lob_loc AT :Position INTO :Temp_loc1;

/* 同じ BFILE から次のロードの位置を設定します。*/
Position = Amount + 1;

/* BFILE から 2 番目のテンポラリ LOB に、2 番目の量をロードします。*/
EXEC SQL LOB LOAD :Amount FROM FILE :Lob_loc AT :Position INTO :Temp_loc2;

/* 1 番目のテンポラリ LOB の終わりに、2 番目のテンポラリ LOB を追加します。*/
EXEC SQL LOB APPEND :Temp_loc2 TO :Temp_loc1;

/* LOB をオープンしている場合、その LOB をクローズする必要があります。*/
EXEC SQL LOB CLOSE :Lob_loc;
EXEC SQL LOB CLOSE :Temp_loc1;
EXEC SQL LOB CLOSE :Temp_loc2;

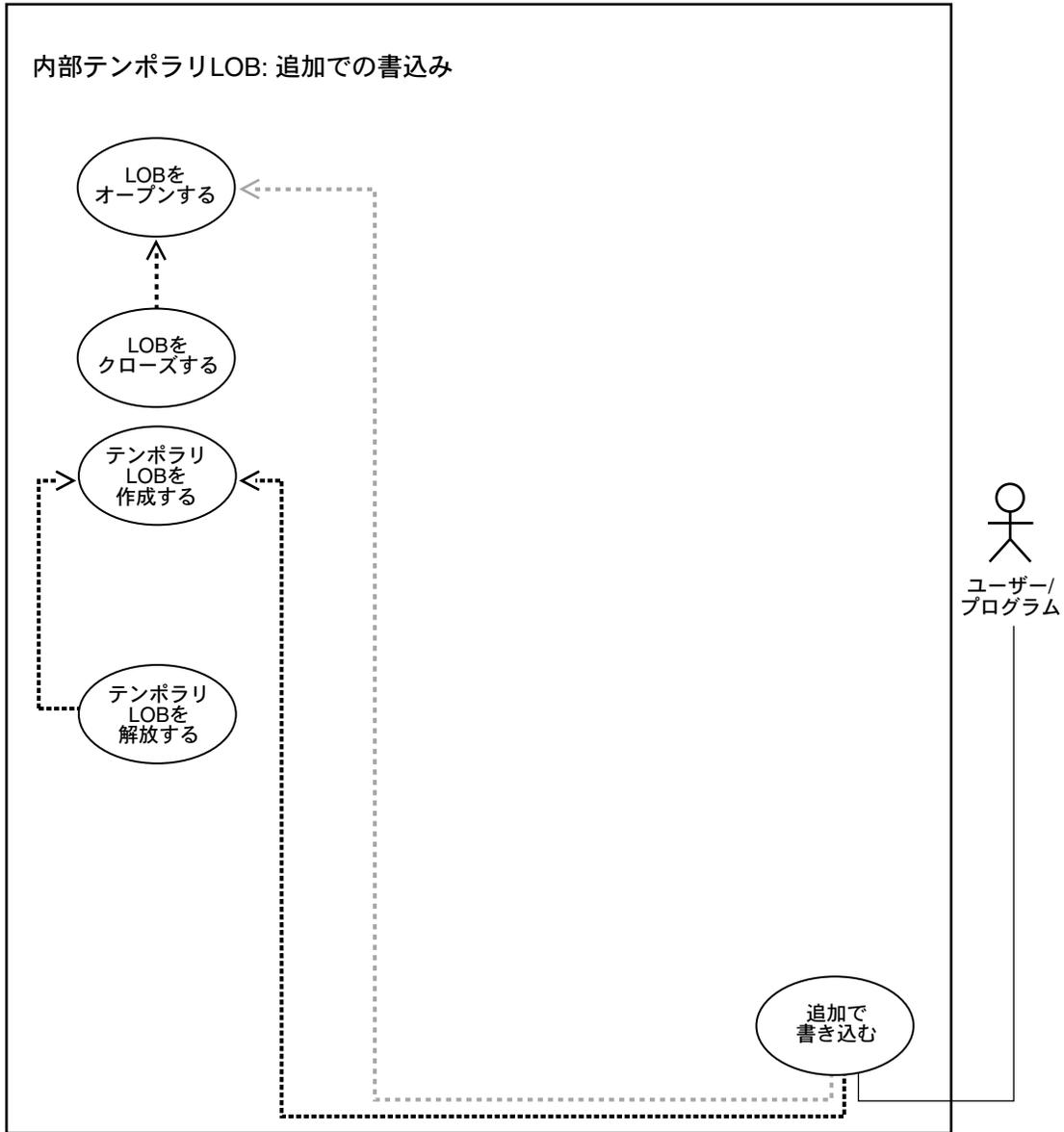
/* テンポラリ LOB を解放します。*/
EXEC SQL LOB FREE TEMPORARY :Temp_loc1;
EXEC SQL LOB FREE TEMPORARY :Temp_loc2;

/* ロケータが保持しているリソースを解放します。*/
EXEC SQL FREE :Lob_loc;
EXEC SQL FREE :Temp_loc1;
EXEC SQL FREE :Temp_loc2;
}

void main()
{
    char *samp = "samp/samp";
    EXEC SQL CONNECT :samp;
    appendTempLOB_proc();
    EXEC SQL ROLLBACK WORK RELEASE;
}
```

テンポラリ LOB への追加での書込み

図 10-21 ユースケース図：テンポラリ LOB への追加での書込み



参照： 内部テンポラリ LOB に関するすべての基本操作については、10-2 ページの「ユースケース・モデル: 内部テンポラリ LOB」を参照してください。

用途

テンポラリ LOB に追加で書き込みます。

使用上の注意

ありません。

構文

各プログラム環境で使用可能な機能のリストは、第 3 章「LOB プログラム環境」を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- PL/SQL (DBMS_LOB) : 『Oracle8i PL/SQL パッケージ・プロシージャ・リファレンス』の第 18 章「DBMS_LOB」の「サブプログラムの要約」の WRITEAPPEND プロシージャ
- C (OCI) : 『Oracle8i コール・インタフェース・プログラマーズ・ガイド』の第 15 章「OCI リレーショナル関数」の「LOB 関数」の OCILobWriteAppend()
- COBOL (Pro*COBOL) : 『Oracle8i Pro*COBOL プリコンパイラ・プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB WRITE APPEND (実行可能埋込み SQL 拡張要素)」
- C/C++ (Pro*C/C++) : 『Oracle8i Pro*C/C++ プリコンパイラ・プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB WRITE APPEND (実行可能埋込み SQL 拡張要素)」
- Visual Basic (OO4O) : 参照マニュアルはありません。
- Java (JDBC) : 参照マニュアルはありません。

使用例

次の例では、Washington_audio ファイルから、32767 バイトのデータを読み込み、テンポラリ LOB に追加します。

例

次のプログラム環境の例が示されています。

- PL/SQL (DBMS_LOB パッケージ) : テンポラリ LOB への追加での書込み (10-141 ページ)
- C (OCI) : テンポラリ LOB への追加での書込み (10-143 ページ)
- COBOL (Pro*COBOL) : テンポラリ LOB への追加での書込み (10-144 ページ)
- C/C++ (Pro*C/C++) : テンポラリ LOB への追加での書込み (10-146 ページ)
- Visual Basic (OO4O) : 今回のリリースでは例は記載されていません。
- Java (JDBC) : 今回のリリースでは例は記載されていません。

PL/SQL (DBMS_LOB パッケージ) : テンポラリ LOB への追加での書込み

/* 例のプロシージャ writeAppendTempLOB_proc は、DBMS_LOB パッケージの一部ではないことに注意してください。この例のプロシージャでは、Washington_audio ファイルからテンポラリ LOB に 32,767 バイトのデータが読み込まれ、そのデータがバッファに追加されます。*/

```
CREATE OR REPLACE PROCEDURE writeAppendTempLOB_proc IS
  Lob_loc      BLOB;
  Buffer        RAW(32767);
  Src_loc      BFILE := BFILENAME('AUDIO_DIR', 'Washington_audio');
  Amount       Binary_integer := 32767;
BEGIN
  DBMS_LOB.CREATETEMPORARY(Lob_loc, TRUE);
  /* テンポラリ LOB のオープンはオプションです。*/
  DBMS_LOB.OPEN(Lob_loc, DBMS_LOB.LOB_READWRITE);
  /* FILE のオープンは必須です。*/
  DBMS_LOB.OPEN(Src_loc, DBMS_LOB.LOB_READONLY);
  /* バッファにデータを充填します。*/
  DBMS_LOB.LOADFROMFILE (Lob_loc, Src_loc, Amount);

  /* バッファから LOB の終わりにデータを追加します。*/
  DBMS_LOB.WRITEAPPEND(Lob_loc, Amount, Buffer);
  DBMS_LOB.CLOSE(Src_loc);
  DBMS_LOB.CLOSE(Lob_loc);
  DBMS_LOB.FREETEMPORARY(Lob_loc);
END;
```

C (OCI) : テンポラリ LOB への追加での書込み

```

#define MAXBUFLLEN 32767
sb4 write_append_temp_lobs (OCIError      *errhp,
                             OCISvcCtx    *svchp,
                             OCIStmt      *stmthp,
                             OCIEnv       *envhp)
{
    OCIClobLocator *tclob;
    unsigned int Total = 40000;
    unsigned int amtp;
    unsigned int nbytes;
    ub1 bufp[MAXBUFLLEN];

    /* ロケータ記述子を割り当てます。*/
    (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &tclob ,
                              (ub4)OCI_DTYPE_LOB, (size_t) 0, (dvoid **) 0);

    if(OCILobCreateTemporary(svchp, errhp, tclob, (ub2)0, SQLCS_IMPLICIT,
                              OCI_TEMP_CLOB, OCI_ATTR_NOCACHE, OCI_DURATION_SESSION))
    {
        (void) printf("FAILED: CreateTemporary() \n");
        return -1;
    }

    /* CLOB をオープンします。*/
    printf("calling open \n");
    checkerr (errhp, (OCILobOpen(svchp, errhp, tclob, OCI_LOB_READWRITE)));

    nbytes = MAXBUFLLEN; /* 標準のポーリングを介してストリーミングを使用します。*/

    /* バッファにnバイト相当のデータを充填します。*/
    memset (bufp, 'a', 32767);

    amtp = sizeof(bufp);
    /* Amount を 0 (ゼロ) に設定すると、OCI_LAST_PIECEを指定するまでデータが流されます。*/

    printf("calling write append \n");
    checkerr (errhp, OCILobWriteAppend (svchp, errhp, tclob, &amtp,
                                        bufp, nbytes, OCI_ONE_PIECE, (dvoid *)0,
                                        (sb4 (*) (dvoid*,dvoid*,ub4*,ub1 *))0,
                                        0, SQLCS_IMPLICIT));

    printf("calling close \n");
    /* LOB をオープンしている場合、その LOB をクローズする必要があります。*/
    checkerr (errhp, OCILobClose(svchp, errhp, tclob));
}

```

```
/* テンポラリ LOB を解放します。*/  
printf("calling free\n");  
checkerr(errhnp, OCILobFreeTemporary(svchp, errhnp, tclob));  
  
/* ロケータが保持しているリソースを解放します。*/  
(void) OCIDescriptorFree((dvoid *) tclob, (ub4) OCI_DTYPE_LOB);  
}
```

COBOL (Pro*COBOL) : テンポラリ LOB への追加での書込み

```
IDENTIFICATION DIVISION.  
PROGRAM-ID. WRITE-APPEND-TEMP.  
ENVIRONMENT DIVISION.  
DATA DIVISION.  
WORKING-STORAGE SECTION.  
  
01 USERID PIC X(11) VALUES "SAMP/SAMP".  
01 TEMP-BLOB SQL-BLOB.  
01 SRC-BFILE SQL-BFILE.  
01 BUFFER PIC X(2048).  
01 DIR-ALIAS PIC X(30) VARYING.  
01 FNAME PIC X(20) VARYING.  
01 DIR-IND PIC S9(4) COMP.  
01 FNAME-IND PIC S9(4) COMP.  
01 AMT PIC S9(9) COMP VALUE 10.  
EXEC SQL VAR BUFFER IS RAW(2048) END-EXEC.  
01 ORASLNDRD PIC 9(4).  
  
EXEC SQL INCLUDE SQLCA END-EXEC.  
EXEC ORACLE OPTION (ORACA=YES) END-EXEC.  
EXEC SQL INCLUDE ORACA END-EXEC.  
  
PROCEDURE DIVISION.  
WRITE-APPEND-TEMP.  
  
EXEC SQL WHENEVER SQLERROR DO PERFORM SQL-ERROR END-EXEC.  
EXEC SQL  
CONNECT :USERID  
END-EXEC.  
  
* BFILE および BLOB ロケータの割当ておよび初期化を行います。  
EXEC SQL ALLOCATE :TEMP-BLOB END-EXEC.  
EXEC SQL ALLOCATE :SRC-BFILE END-EXEC.  
EXEC SQL  
LOB CREATE TEMPORARY :TEMP-BLOB  
END-EXEC.
```

```
* ディレクトリおよびファイル情報を設定します。
MOVE "AUDIO_DIR" TO DIR-ALIAS-ARR.
MOVE 9 TO DIR-ALIAS-LEN.
MOVE "Washington_audio" TO FNAME-ARR.
MOVE 16 TO FNAME-LEN.

EXEC SQL
  LOB FILE SET :SRC-BFILE DIRECTORY = :DIR-ALIAS,
  FILENAME = :FNAME
END-EXEC.

* ソース BFILE および宛先テンポラリ BLOB をオープンします。
EXEC SQL LOB OPEN :TEMP-BLOB READ WRITE END-EXEC.
EXEC SQL LOB OPEN :SRC-BFILE READ ONLY END-EXEC.
EXEC SQL
  LOB LOAD :AMT FROM FILE :SRC-BFILE INTO :TEMP-BLOB
END-EXEC.

MOVE "262626" TO BUFFER.
MOVE 3 TO AMT.
* BUFFER のデータを TEMP-BLOB に追加します。
EXEC SQL
  LOB WRITE APPEND :AMT FROM :BUFFER INTO :TEMP-BLOB
END-EXEC.
* LOB をクローズします。
EXEC SQL LOB CLOSE :SRC-BFILE END-EXEC.
EXEC SQL LOB CLOSE :TEMP-BLOB END-EXEC.

* テンポラリ LOB を解放します。
EXEC SQL
  LOB FREE TEMPORARY :TEMP-BLOB
END-EXEC.

* LOB ロケータを解放します。
EXEC SQL FREE :TEMP-BLOB END-EXEC.
EXEC SQL FREE :SRC-BFILE END-EXEC.
STOP RUN.

SQL-ERROR.
EXEC SQL
  WHENEVER SQLERROR CONTINUE
END-EXEC.
MOVE ORASLNR TO ORASLNRD.
DISPLAY " ".
DISPLAY "ORACLE ERROR DETECTED ON LINE ", ORASLNRD, ":".
DISPLAY " ".
```

```
DISPLAY SQLERRM.  
EXEC SQL  
    ROLLBACK WORK RELEASE  
END-EXEC.  
STOP RUN.
```

C/C++ (Pro*C/C++) : テンポラリ LOB への追加での書込み

```
#include <oci.h>  
#include <stdio.h>  
#include <sqlca.h>  
  
void Sample_Error()  
{  
    EXEC SQL WHENEVER SQLERROR CONTINUE;  
    printf("%.*s\n", sqlca.sqlerrm.sqlerrml, sqlca.sqlerrm.sqlerrmc);  
    EXEC SQL ROLLBACK WORK RELEASE;  
    exit(1);  
}  
  
#define BufferLength 256  
  
void writeAppendTempLOB_proc()  
{  
    OCIBlobLocator *Temp_loc;  
    OCIBFileLocator *Lob_loc;  
    char *Dir = "AUDIO_DIR", *Name = "Washington_audio";  
    int Amount;  
    struct {  
        unsigned short Length;  
        char Data[BufferLength];  
    } Buffer;  
    EXEC SQL VAR Buffer IS VARRAW(BufferLength);  
    EXEC SQL WHENEVER SQLERROR DO Sample_Error();  
  
    /* テンポラリ LOB の割当ておよび作成を行います。*/  
    EXEC SQL ALLOCATE :Temp_loc;  
    EXEC SQL LOB CREATE TEMPORARY :Temp_loc;  
  
    /* BFILE ロケータの割当ておよび初期化を行います。*/  
    EXEC SQL ALLOCATE :Lob_loc;  
    EXEC SQL LOB FILE SET :Lob_loc DIRECTORY = :Dir, FILENAME = :Name;  
  
    /* LOB のオープンはオプションです。*/
```

```
EXEC SQL LOB OPEN :Lob_loc READ ONLY;
EXEC SQL LOB OPEN :Temp_loc READ WRITE;

/* BFILE からテンポラリ LOB に、指定した量をロードします。*/
Amount = 2048;
EXEC SQL LOB LOAD :Amount FROM FILE :Lob_loc INTO :Temp_loc;
strcpy((char *)Buffer.Data, "afafafafafaf");
Buffer.Length = 6;

/* バッファの内容をテンポラリ LOB の終わりに書き込みます。*/
Amount = Buffer.Length;
EXEC SQL LOB WRITE APPEND :Amount FROM :Buffer INTO :Temp_loc;

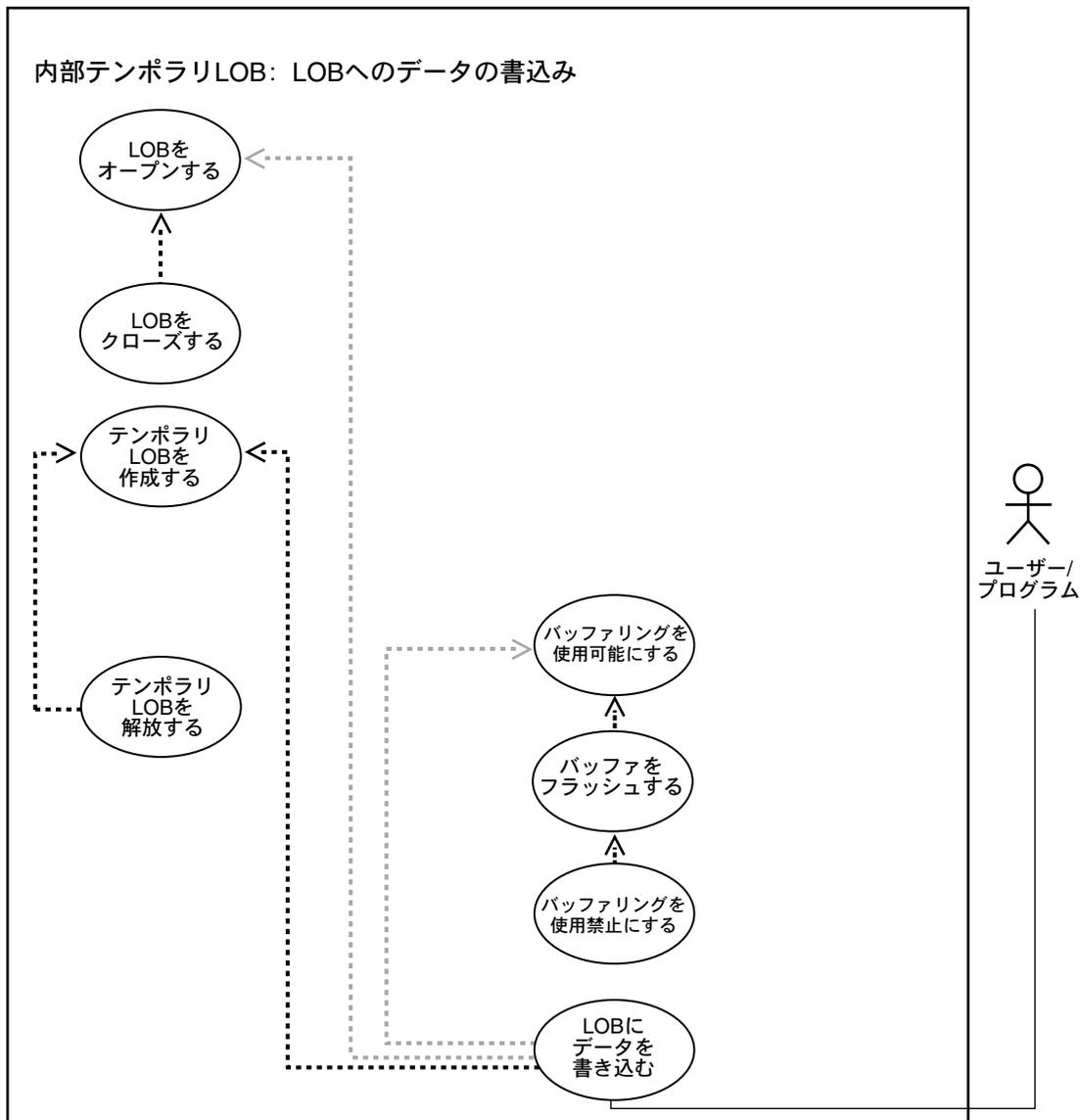
/* LOB をオープンしている場合、その LOB をクローズする必要があります。*/
EXEC SQL LOB CLOSE :Lob_loc;
EXEC SQL LOB CLOSE :Temp_loc;

/* テンポラリ LOB を解放します。*/
EXEC SQL LOB FREE TEMPORARY :Temp_loc;
/* ロケータが保持しているリソースを解放します。*/
EXEC SQL FREE :Lob_loc;
EXEC SQL FREE :Temp_loc;
}

void main()
{
    char *samp = "samp/samp";
    EXEC SQL CONNECT :samp;
    writeAppendTempLOB_proc();
    EXEC SQL ROLLBACK WORK RELEASE;
}
```

テンポラリ LOB へのデータの書込み

図 10-22 ユースケース図：テンポラリ LOB へのデータの書込み



参照： 内部テンポラリ LOB に関するすべての基本操作については、10-2 ページの「ユースケース・モデル: 内部テンポラリ LOB」を参照してください。

用途

データをテンポラリ LOB に書き込みます。

使用上の注意

ストリーム書込み

大量の LOB データを最も効率よく書き込む方法は、OCILOBWrite() を使ったポーリングやコールバックによるストリーミングです。LOB に書き込むデータの量がわかっている場合は、OCILOBWrite() のコール時にそのデータ量を指定します。これによって、ディスクに LOB データが連続的に書き込まれます。領域を効率的に使用できるのみでなく、LOB データの連続性により、その後の操作で読み書きの速度が速くなります。

DBMS_LOB.WRITE() を使用したデータのテンポラリ BLOB への書込み

16 進文字列を DBMS_LOB.WRITE() に渡して BLOB にデータを書き込む場合は、次のガイドラインに従ってください。

- amount パラメータは、バッファの length パラメータ以下にする必要があります。
- バッファの length は ((amount × 2) - 1) にする必要があります。このガイドラインが存在する理由は、文字列の 2 つの文字が 1 つの 16 進文字とみなされる（また、16 進からローへの暗黙的な変換が行われる）ためです。すなわち、文字列が 2 バイトごとに 1 つのロー・バイトへ変換されます。

次は正しい例です。

```
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;
```

前の例の「an_amount」の値を次の値に置き換えると、エラー・メッセージ ORA-21560 が戻されます。

```
an_amount BINARY_INTEGER := 11;
```

または

```
an_amount BINARY_INTEGER := 19;
```

構文

各プログラム環境で使用可能な機能のリストは、[第 3 章「LOB プログラム環境」](#)を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- PL/SQL (DBMS_LOB) : 『Oracle8i PL/SQL パッケージ・プロシージャ・リファレンス』の第 18 章「DBMS_LOB」の「サブプログラムの要約」の WRITE プロシージャ
- C (OCI) : 『Oracle8i コール・インタフェース・プログラマーズ・ガイド』の第 15 章「OCI リレーショナル関数」の「LOB 関数」の OCILobWrite()
- COBOL (Pro*COBOL) : 『Oracle8i Pro*COBOL プリコンパイラ・プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB WRITE (実行可能埋込み SQL 拡張要素)」
- C/C++ (Pro*C/C++) : 『Oracle8i Pro*C/C++ プリコンパイラ・プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB WRITE (実行可能埋込み SQL 拡張要素)」
- Visual Basic (OO4O) : 参照マニュアルはありません。
- Java (JDBC) : 参照マニュアルはありません。

使用例

次の例では、LOB にデータを書き込むことによって STORY データ（クリップ用のストーリーボード）が更新できます。

例

次のプログラム環境の例が示されています。

- [PL/SQL \(DBMS_LOB パッケージ\) : テンポラリ LOB へのデータの書込み \(10-150 ページ\)](#)
- [C \(OCI\) : テンポラリ LOB へのデータの書込み \(10-151 ページ\)](#)
- [COBOL \(Pro*COBOL\) : テンポラリ LOB へのデータの書込み \(10-154 ページ\)](#)
- [C/C++ \(Pro*C/C++\) : テンポラリ LOB へのデータの書込み \(10-155 ページ\)](#)

- Visual Basic (OO4O) : 今回のリリースでは例は記載されていません。
- Java (JDBC) : 今回のリリースでは例は記載されていません。

PL/SQL (DBMS_LOB パッケージ) : テンポラリ LOB へのデータの書込み

```

/* 例のプロシージャ writeDataToTempLOB_proc は、DBMS_LOB パッケージの
一部ではないことに注意してください。*/
CREATE or REPLACE PROCEDURE writeDataToTempLOB_proc IS
  Lob_loc          CLOB;
  Buffer            VARCHAR2(26);
  Amount           BINARY_INTEGER := 26;
  Position         INTEGER := 1;
  i                INTEGER;
BEGIN
  DBMS_LOB.CREATETEMPORARY(Lob_loc,TRUE);
  /* LOB のオープンはオプションです。*/
  DBMS_LOB.OPEN (Lob_loc, DBMS_LOB.LOB_READWRITE);
  /* LOB に書き込むデータをバッファに充填します。*/
  Buffer := 'abcdefghijklmnopqrstuvwxyz';

  FOR i IN 1..3 LOOP
    DBMS_LOB.WRITE (Lob_loc, Amount, Position, Buffer);
    /* LOB に書き込む追加のデータをバッファに充填します。*/
    Position := Position + Amount;
  END LOOP;
  /* LOB をオープンしている場合、その LOB をクローズする必要があります。*/
  DBMS_LOB.CLOSE (Lob_loc);
  DBMS_LOB.FREETEMPORARY(Lob_loc);
END;

```

C (OCI) : テンポラリ LOB へのデータの書込み

```

/* この例では、ポーリングを使用したストリーミング書込みを示します。*/
#define MAXBUFLLEN 32767
sb4 write_temp_lobs (OCIError          *errhp,
                    OCISvcCtx         *svchp,
                    OCIStmt           *stmthp,
                    OCIEnv            *envhp)
{
  OCIClobLocator *tclob;
  unsigned int Total = 40000;
  unsigned int amtp;
  unsigned int offset;
  unsigned int remainder, nbytes;
  boolean last;

```

```

ub1 bufp [MAXBUFLLEN];
sb4      err;

/* ロケータ記述子を割り当てます。*/
(void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &tclob ,
                          (ub4)OCI_DTYPE_LOB, (size_t) 0, (dvoid **) 0);

if (OCILobCreateTemporary(svchp,
                          errhp,
                          tclob,
                          (ub2)0,
                          SQLCS_IMPLICIT,
                          OCI_TEMP_CLOB,
                          OCI_ATTR_NOCACHE,
                          OCI_DURATION_SESSION))
{
    (void) printf("FAILED: CreateTemporary() \n");
    return -1;
}

/* CLOB をオープンします。*/
checkerr (errhp, (OCILobOpen(svchp, errhp, tclob, OCI_LOB_READWRITE)));

if (Total > MAXBUFLLEN)
    nbytes = MAXBUFLLEN; /* 標準のポーリングを介してストリーミングを使用します。*/
else
    nbytes = Total;      /* 単一の書込みのみ必要です。*/

/* バッファに n バイト相当のデータを充填します。*/
memset (bufp, 'a', 32767);

remainder = Total - nbytes;
amtp = 0;
offset = 1;
/* Amount を 0 (ゼロ) に設定すると、OCI_LAST_PIECE を指定するまでデータが流されます。*/

if (0 == remainder)
{
    amtp = nbytes;
    /* ここで、合計が MAXBUFLLEN 以下であるため、1 つのピースごと書込みできます。*/
    checkerr (errhp, OCILobWrite (svchp, errhp, tclob, &amtp,
                                  offset, bufp, nbytes,
                                  OCI_ONE_PIECE, (dvoid *)0,
                                  (sb4 (*) (dvoid*,dvoid*,ub4*,ub1 *))0,
                                  0, SQLCS_IMPLICIT));
}
else

```

```
{
/* ここで、合計がMAXBUFLenより大きい場合、標準のポーリングを介してストリーミングを使用します。*/
/* 最初のピースを書き込みます。FIRSTを指定すると、ポーリングが開始します。*/
err = OCILobWrite (svchp, errhp, tclob, &amp;tp,
                  offset, bufp, nbytes,
                  OCI_FIRST_PIECE, (dvoid *)0,
                  (sb4 (*) (dvoid*,dvoid*,ub4*,ub1 *))0,
                  0, SQLCS_IMPLICIT);

if (err != OCI_NEED_DATA)
    checkerr (errhp, err);

last = FALSE;
/* 次 (中間) および最後のピースを書き込みます。*/
do
{
    if (remainder > MAXBUFLen)
        nbytes = MAXBUFLen;          /* ピースが残っています。*/
    else
    {
        nbytes = remainder;          /* ここで、remainderはMAXBUFLen以下です。*/
        last = TRUE;                 /* これが最後のピースになります。*/
    }

    /* バッファにnバイト相当のデータを充填します。*/

    if (last)
    {
        /* LASTを指定すると、ポーリングが終了します。*/
        err = OCILobWrite (svchp, errhp, tclob, &amp;tp,
                          offset, bufp, nbytes,
                          OCI_LAST_PIECE, (dvoid *)0,
                          (sb4 (*) (dvoid*,dvoid*,ub4*,ub1 *))0,
                          0, SQLCS_IMPLICIT);

        if (err != 0)
            checkerr (errhp, err);
    }
    else
    {
        err = OCILobWrite (svchp, errhp, tclob, &amp;tp,
                          offset, bufp, nbytes,
                          OCI_NEXT_PIECE, (dvoid *)0,
                          (sb4 (*) (dvoid*,dvoid*,ub4*,ub1 *))0,
                          0, SQLCS_IMPLICIT);
    }
}
```

```
        if (err != OCI_NEED_DATA)
            checkerr (errhp, err);

    }
    /* 残りの書込み量を判断します。*/
    remainder = remainder - nbytes;
    } while (!last);
}
/* この時点では、(remainder == 0 (ゼロ)) です。*/

/* LOB をオープンしている場合、その LOB をクローズする必要があります。*/
checkerr (errhp, OCILobClose(svchp, errhp, tclob));

/* テンポラリ LOB を解放します。*/
checkerr(errhp, OCILobFreeTemporary(svchp, errhp, tclob));

/* ロケータが保持しているリソースを解放します。*/
(void) OCIDescriptorFree((dvoid *) tclob, (ub4) OCI_DTYPE_LOB);
}
```

COBOL (Pro*COBOL) : テンポラリ LOB へのデータの書込み

```
IDENTIFICATION DIVISION.
PROGRAM-ID. WRITE-TEMP.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

01  USERID    PIC X(11) VALUES "SAMP/SAMP".
01  TEMP-CLOB SQL-CLOB.
01  BUFFER    PIC X(20) VARYING.
01  DIR-ALIAS PIC X(30) VARYING.
01  FNAME     PIC X(20) VARYING.
01  DIR-IND   PIC S9(4) COMP.
01  FNAME-IND PIC S9(4) COMP.
01  AMT       PIC S9(9) COMP VALUE 10.
01  ORASLNRD  PIC 9(4) .

EXEC SQL INCLUDE SQLCA END-EXEC.
EXEC ORACLE OPTION (ORACA=YES) END-EXEC.
EXEC SQL INCLUDE ORACA END-EXEC.

PROCEDURE DIVISION.
WRITE-TEMP.
```

```
EXEC SQL WHENEVER SQLERROR DO PERFORM SQL-ERROR END-EXEC.  
EXEC SQL  
    CONNECT :USERID  
END-EXEC.  
  
* BFILE および BLOB ロケータの割当ておよび初期化を行います。  
EXEC SQL ALLOCATE :TEMP-CLOB END-EXEC.  
EXEC SQL  
    LOB CREATE TEMPORARY :TEMP-CLOB  
END-EXEC.  
EXEC SQL LOB OPEN :TEMP-CLOB READ WRITE END-EXEC.  
  
MOVE "ABCDE12345ABCDE12345" TO BUFFER-ARR.  
MOVE 20 TO BUFFER-LEN.  
MOVE 20 TO AMT.  
* BUFFER 内のデータを TEMP-CLOB に追加します。  
EXEC SQL LOB WRITE :AMT FROM :BUFFER INTO :TEMP-CLOB END-EXEC.  
  
* LOB をクローズします。  
EXEC SQL LOB CLOSE :TEMP-CLOB END-EXEC.  
  
* テンポラリ LOB を解放します。  
EXEC SQL LOB FREE TEMPORARY :TEMP-CLOB END-EXEC.  
  
* LOB ロケータを解放します。  
EXEC SQL FREE :TEMP-CLOB END-EXEC.  
STOP RUN.  
  
SQL-ERROR.  
EXEC SQL WHENEVER SQLERROR CONTINUE END-EXEC.  
MOVE ORASLNR TO ORASLNDR.  
DISPLAY " ".  
DISPLAY "ORACLE ERROR DETECTED ON LINE ", ORASLNDR, " :".  
DISPLAY " ".  
DISPLAY SQLERRMC.  
EXEC SQL ROLLBACK WORK RELEASE END-EXEC.  
STOP RUN.
```

C/C++ (Pro*C/C++) : テンポラリ LOB へのデータの書込み

```
#include <oci.h>  
#include <stdio.h>  
#include <string.h>  
#include <sqlca.h>  
  
void Sample_Error()
```

```

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

#define BufferLength 1024

void writeDataToTempLOB_proc(multiple) int multiple;
{
    OCIClobLocator *Temp_loc;
    varchar Buffer[BufferLength];
    unsigned int Total;
    unsigned int Amount;
    unsigned int remainder, nbytes;
    boolean last;

    EXEC SQL WHENEVER SQLERROR DO Sample_Error();
    /* テンポラリ LOB の割当ておよび初期化を行います。 */
    EXEC SQL ALLOCATE :Temp_loc;
    EXEC SQL LOB CREATE TEMPORARY :Temp_loc;
    /* テンポラリ LOB をオープンします。 */
    EXEC SQL LOB OPEN :Temp_loc READ WRITE;
    Total = Amount = (multiple * BufferLength);
    if (Total > BufferLength)
        nbytes = BufferLength; /* 標準のポーリングを介してストリーミングを使用します。 */
    else
        nbytes = Total; /* 単一の書込みのみ必要です。 */
    /* バッファに n バイト相当のデータを充填します。 */
    memset((void *)Buffer.arr, 32, nbytes);
    Buffer.len = nbytes; /* 長さを設定します。 */
    remainder = Total - nbytes;
    if (0 == remainder)
    {
        /* ここで、合計が BufferLength 以下であるため、1 つのピースごと書込みできます。 */
        EXEC SQL LOB WRITE ONE :Amount FROM :Buffer INTO :Temp_loc;
        printf("Write ONE Total of %d characters\n", Amount);
    }
    else
    {
        /* ここで、合計が BufferLength より大きいいため、標準のポーリングを介してストリーミングを使用します。 */
        /* 最初のピースを書き込みます。FIRST を指定すると、ポーリングが開始します。 */
        EXEC SQL LOB WRITE FIRST :Amount FROM :Buffer INTO :Temp_loc;
        printf("Write FIRST %d characters\n", Buffer.len);
        last = FALSE;
    }
}

```

```
/* 次 (中間) および最後のピースを書き込みます。*/
do
{
    if (remainder > BufferLength)
        nbytes = BufferLength;          /* ピースが残っています。*/
    else
    {
        nbytes = remainder;            /* ここで、remainder は BufferLength 以下です。*/
        last = TRUE;                   /* これが最後のピースになります。*/
    }
    /* バッファに n バイト相当のデータを充填します。*/
    memset((void *)Buffer.arr, 32, nbytes);
    Buffer.len = nbytes;                /* 長さを設定します。*/
    if (last)
    {
        EXEC SQL WHENEVER SQLERROR DO Sample_Error();
        /* LAST を指定すると、ポーリングが終了します。*/
        EXEC SQL LOB WRITE LAST :Amount FROM :Buffer INTO :Temp_loc;
        printf("Write LAST Total of %d characters\n", Amount);
    }
    else
    {
        EXEC SQL WHENEVER SQLERROR DO break;
        EXEC SQL LOB WRITE NEXT :Amount FROM :Buffer INTO :Temp_loc;
        printf("Write NEXT %d characters\n", Buffer.len);
    }
    /* 残りの書き込み量を判断します。*/
    remainder = remainder - nbytes;
} while (!last);

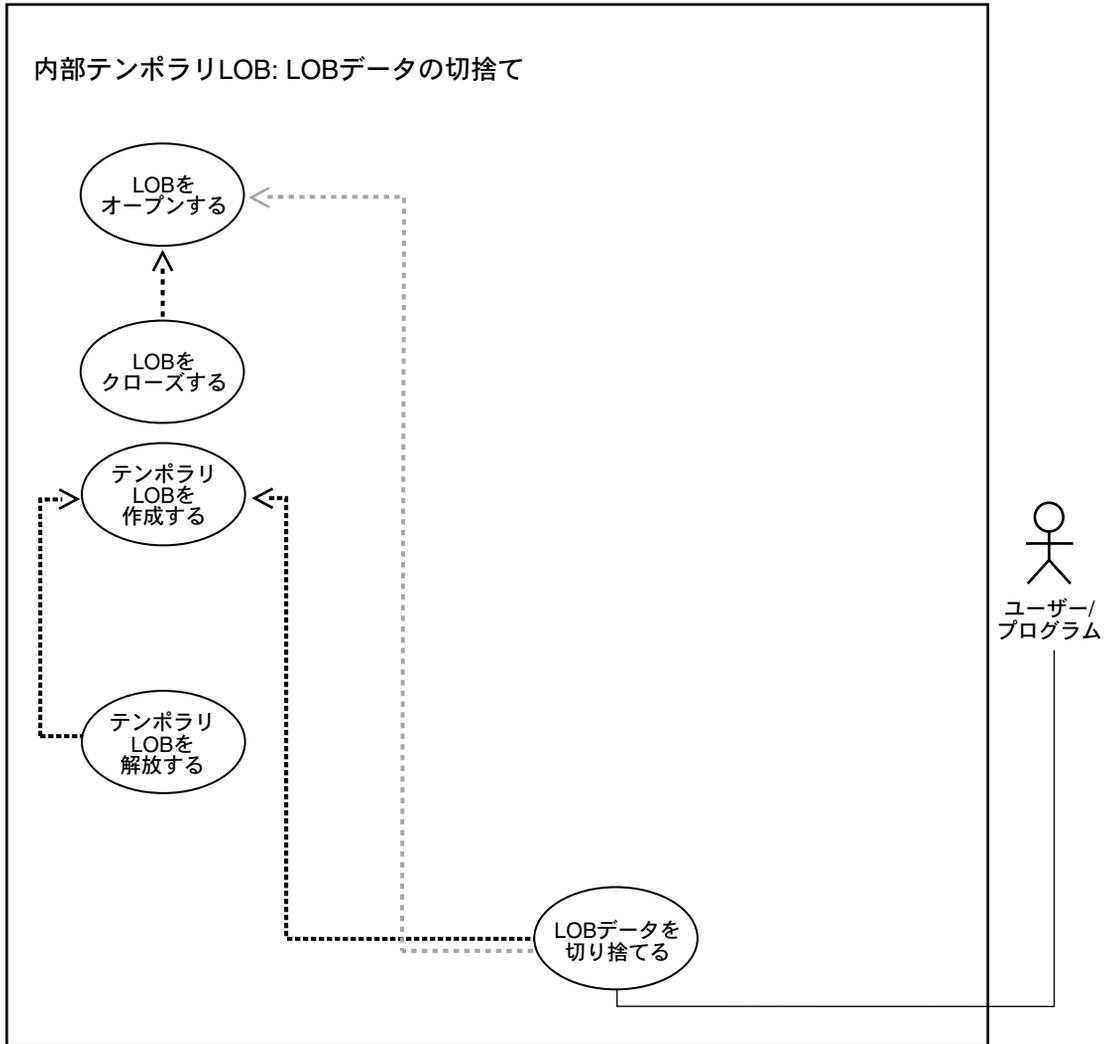
}

EXEC SQL WHENEVER SQLERROR DO Sample_Error();
/* この時点では、(Amount == 合計) であるため、合計量が書き込まれています。*/
/* テンポラリ LOB をクローズします。*/
EXEC SQL LOB CLOSE :Temp_loc;
/* テンポラリ LOB を解放します。*/
EXEC SQL LOB FREE TEMPORARY :Temp_loc;
/* ロケータが保持しているリソースを解放します。*/
EXEC SQL FREE :Temp_loc;
}

void main()
{
    char *samp = "samp/samp";
    EXEC SQL CONNECT :samp;
    writeDataToTempLOB_proc(1);          /* 1 つのピースを書き込みます。*/
    writeDataToTempLOB_proc(4);        /* ポーリングを使用して、複数のピースを書き込みます。*/
    EXEC SQL ROLLBACK WORK RELEASE;
}
}
```

テンポラリ LOB データの切捨て

図 10-23 ユースケース図：テンポラリ LOB データの切捨て



参照： 内部テンポラリ LOB に関するすべての基本操作については、10-2 ページの「ユースケース・モデル：内部テンポラリ LOB」を参照してください。

用途

テンポラリ LOB データを切り捨てます。

使用上の注意

ありません。

構文

各プログラム環境で使用可能な機能のリストは、[第 3 章「LOB プログラム環境」](#)を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- PL/SQL (DBMS_LOB) : 『Oracle8i PL/SQL パッケージ・プロシージャ・リファレンス』の第 18 章「DBMS_LOB」の「サブプログラムの要約」の TRIM プロシージャ
- C (OCI) : 『Oracle8i コール・インタフェース・プログラマーズ・ガイド』の第 15 章「OCI リレーショナル関数」の「LOB 関数」の OCILobTrim()
- COBOL (Pro*COBOL) : 『Oracle8i Pro*COBOL プリコンパイラ・プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB TRIM (実行可能埋込み SQL 拡張要素)」
- C/C++ (Pro*C/C++) : 『Oracle8i Pro*C/C++ プリコンパイラ・プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB TRIM (実行可能埋込み SQL 拡張要素)」
- Visual Basic (OO4O) : 参照マニュアルはありません。
- Java (JDBC) : 参照マニュアルはありません。

使用例

次の例では、Voiceover_tab 表の Script 列で参照されるテキスト (CLOB データ) にアクセスし、切り捨てます。

例

次のプログラム環境の例が示されています。

- [PL/SQL \(DBMS_LOB パッケージ\) : テンポラリ LOB データの切捨て \(10-159 ページ\)](#)
- [C \(OCI\) : テンポラリ LOB データの切捨て \(10-160 ページ\)](#)
- [COBOL \(Pro*COBOL\) : テンポラリ LOB データの切捨て \(10-162 ページ\)](#)
- [C/C++ \(Pro*C/C++\) : テンポラリ LOB データの切捨て \(10-164 ページ\)](#)
- Visual Basic (OO4O) : 今回のリリースでは例は記載されていません。
- Java (JDBC) : 今回のリリースでは例は記載されていません。

PL/SQL (DBMS_LOB パッケージ) : テンポラリ LOB データの切捨て

```

/* 例のプロシージャ trimTempLOB_proc は、DBMS_LOB パッケージの
一部ではないことに注意してください。*/
CREATE OR REPLACE PROCEDURE trimTempLOB_proc IS
  Lob_loc      CLOB;
  Amount       number;
  Src_loc      BFILE := BFILENAME('AUDIO_DIR', 'Washington_audio');
  TrimAmount   number := 100;
BEGIN
  /* テンポラリ LOB を作成します。*/
  DBMS_LOB.CREATETEMPORARY (Lob_loc, TRUE);
  /* LOB のオープンはオプションです。*/
  DBMS_LOB.OPEN (Lob_loc, DBMS_LOB.LOB_READWRITE);
  /* ファイルのオープンは必須です。*/
  DBMS_LOB.OPEN (Src_loc, DBMS_LOB.LOB_READONLY);
  /* テンポラリ LOB にいくつかのデータを移入します。*/
  Amount := 32767;
  DBMS_LOB.LOADFROMFILE (Lob_loc, Src_loc, Amount);
  DBMS_LOB.TRIM (Lob_loc, TrimAmount);
  /* LOB をオープンしている場合、その LOB をクローズする必要があります。*/
  DBMS_LOB.CLOSE (Lob_loc);
  DBMS_LOB.CLOSE (Src_loc);
  DBMS_LOB.FREETEMPORARY (Lob_loc);
COMMIT;
EXCEPTION
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('Operation failed');
END;

```

C (OCI) : テンポラリ LOB データの切捨て

```

sb4 trim_temp_lobs (  OCIError      *errhp,
                     OCISvcCtx    *svchp,
                     OCISstmt     *stmthp,
                     OCIEnv       *envhp)
{
  OCILobLocator *tblob;
  OCILobLocator *bfile;
  ub4 amt = 4000;
  ub4 trim_size = 2;
  sb4 return_code = 0;

  printf("in trim\n");
  if(OCIDescriptorAlloc((dvoid *)envhp, (dvoid **) &tblob,
                       (ub4) OCI_DTYPE_LOB,

```

```
        (size_t) 0, (dvoid **) 0))
    {
        printf("OCIDescriptor Alloc FAILED in trim\n");
        return -1;
    }

    if(OCIDescriptorAlloc((dvoid *)envhp, (dvoid **) &bfile,
        (ub4) OCI_DTYPE_FILE,
        (size_t) 0, (dvoid **) 0))
    {
        printf("OCIDescriptor Alloc FAILED in trim\n");
        return -1;
    }

    /* BFILE が Washington_audio ファイルを指すように設定します。 */
    if(OCILobFileName(envhp, errhp, &bfile, (text *)"AUDIO_DIR",
        (ub2)strlen("AUDIO_DIR"),
        (text *)"Washington_audio",
        (ub2)strlen("Washington_audio")))
    {
        printf("OCILobFileName FAILED\n");
        return -1;
    }

    if (OCILobFileOpen(svchp, errhp, (OCILobLocator *) bfile, OCI_LOB_READONLY))
    {
        printf("OCILobFileOpen FAILED for the bfile\n");
        return_code = -1;
    }

    if(OCILobCreateTemporary(svchp, errhp, tblob, (ub2)0, SQLCS_IMPLICIT,
        OCI_TEMP_BLOB, OCI_ATTR_NOCACHE,
        OCI_DURATION_SESSION))
    {
        (void) printf("FAILED: CreateTemporary() \n");
        return -1;
    }

    if (OCILobOpen(svchp, errhp, (OCILobLocator *) tblob, OCI_LOB_READWRITE))
    {
        printf("OCILobOpen FAILED for temp LOB \n");
        return_code = -1;
    }

    /* テンポラリ LOB に 4,000 バイトのデータを移入します。 */
    if(OCILobLoadFromFile(svchp, errhp, tblob, (OCILobLocator*)bfile,
        (ub4)amt, (ub4)1, (ub4)1))
```

```
{
    printf( "OCILobLoadFromFile FAILED\n");
    return_code = -1;
}

if (OCILobTrim(svchp, errhp, (OCILobLocator *) tblob, trim_size))
{
    printf( "OCILobTrim FAILED for temp LOB \n");
    return_code = -1;
} else
{
    printf( "OCILobTrim succeeded for temp LOB \n");
}

if (OCILobClose(svchp, errhp, (OCILobLocator *) bfile))
{
    printf( "OCILobClose FAILED for bfile \n");
    return_code = -1;
}

if (OCILobClose(svchp, errhp, (OCILobLocator *) tblob))
{
    printf( "OCILobClose FAILED for temporary LOB \n");
    return_code = -1;
}
/* 使い終わったテンポラリ LOB を解放します。*/
if (OCILobFreeTemporary(svchp, errhp, tblob))
{
    printf( "OCILobFreeTemporary FAILED \n");
    return_code = -1;
}
return return_code;
}
```

COBOL (Pro*COBOL) : テンポラリ LOB データの切捨て

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

01  USERID    PIC X(11) VALUES "SAMP/SAMP".
01  TEMP-BLOB SQL-BLOB.
01  SRC-BFILE SQL-BFILE.
01  DIR-ALIAS PIC X(30) VARYING.
```

```

01 FNAME          PIC X(20) VARYING.
01 DIR-IND        PIC S9(4) COMP.
01 FNAME-IND      PIC S9(4) COMP.
01 AMT            PIC S9(9) COMP VALUE 10.
01 ORASLNRD       PIC 9(4).

```

```

EXEC SQL INCLUDE SQLCA END-EXEC.
EXEC ORACLE OPTION (ORACA=YES) END-EXEC.
EXEC SQL INCLUDE ORACA END-EXEC.

```

PROCEDURE DIVISION.

TEMP-LOB-TRIM.

```

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

```

* BFILE および BLOB ロケータの割当ておよび初期化を行います。

```

EXEC SQL ALLOCATE :TEMP-BLOB END-EXEC.
EXEC SQL ALLOCATE :SRC-BFILE END-EXEC.
EXEC SQL
    LOB CREATE TEMPORARY :TEMP-BLOB
END-EXEC.

```

* ディレクトリおよびファイル情報を設定します。

```

MOVE "AUDIO_DIR" TO DIR-ALIAS-ARR.
MOVE 9 TO DIR-ALIAS-LEN.
MOVE "Washington_audio" TO FNAME-ARR.
MOVE 16 TO FNAME-LEN.

EXEC SQL
    LOB FILE SET :SRC-BFILE DIRECTORY = :DIR-ALIAS,
    FILENAME = :FNAME
END-EXEC.

```

* ソース BFILE および宛先テンポラリ BLOB をオープンします。

```

EXEC SQL LOB OPEN :TEMP-BLOB READ WRITE END-EXEC.
EXEC SQL LOB OPEN :SRC-BFILE READ ONLY END-EXEC.
EXEC SQL
    LOB LOAD :AMT FROM FILE :SRC-BFILE INTO :TEMP-BLOB
END-EXEC.

```

* データの後ろ半分を切り捨てます。

```

MOVE 5 TO AMT.
EXEC SQL LOB TRIM :TEMP-BLOB TO :AMT END-EXEC.

```

```
* LOB をクローズします。
EXEC SQL LOB CLOSE :SRC-BFILE END-EXEC.
EXEC SQL LOB CLOSE :TEMP-BLOB END-EXEC.

* テンポラリ LOB を解放します。
EXEC SQL LOB FREE TEMPORARY :TEMP-BLOB END-EXEC.

* LOB ロケータを解放します。
EXEC SQL FREE :TEMP-BLOB END-EXEC.
EXEC SQL FREE :SRC-BFILE END-EXEC.
STOP RUN.

SQL-ERROR.
EXEC SQL WHENEVER SQLERROR CONTINUE END-EXEC.
MOVE ORASLNR TO ORASLNRD.
DISPLAY " ".
DISPLAY "ORACLE ERROR DETECTED ON LINE ", ORASLNRD, ":".
DISPLAY " ".
DISPLAY SQLERRMC.
EXEC SQL ROLLBACK WORK RELEASE END-EXEC.
STOP RUN.
```

C/C++ (Pro*C/C++) : テンポラリ LOB データの切捨て

```
void trimTempLOB_proc()
#include <oci.h>
#include <stdio.h>
#include <sqlca.h>

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

void trimTempLOB_proc()
{
OCIBlobLocator *Temp_loc;
OCIBFileLocator *Lob_loc;
char *Dir = "AUDIO_DIR", *Name = "Washington_audio";
int Amount = 4096;
int trimLength;
```

```
/* テンポラリ LOB の割当ておよび作成を行います。*/
EXEC SQL ALLOCATE :Temp_loc;
EXEC SQL LOB CREATE TEMPORARY :Temp_loc;

/* BFILE ロケータの割当ておよび初期化を行います。*/
EXEC SQL ALLOCATE :Lob_loc;
EXEC SQL LOB FILE SET :Lob_loc DIRECTORY = :Dir, FILENAME = :Name;

/* LOB のオープンはオプションです。*/
EXEC SQL LOB OPEN :Lob_loc READ ONLY;
EXEC SQL LOB OPEN :Temp_loc READ WRITE;

/* BFILE からテンポラリ LOB に、指定した量をロードします。*/
EXEC SQL LOB LOAD :Amount FROM FILE :Lob_loc INTO :Temp_loc;

/* テンポラリ LOB の新しい長さを設定します。*/
trimLength = (int) (Amount / 2);

/* テンポラリ LOB を新しい長さに切り捨てます。*/
EXEC SQL LOB TRIM :Temp_loc TO :trimLength;

/* LOB をオープンしている場合、その LOB をクローズする必要があります。*/
EXEC SQL LOB CLOSE :Lob_loc;
EXEC SQL LOB CLOSE :Temp_loc;

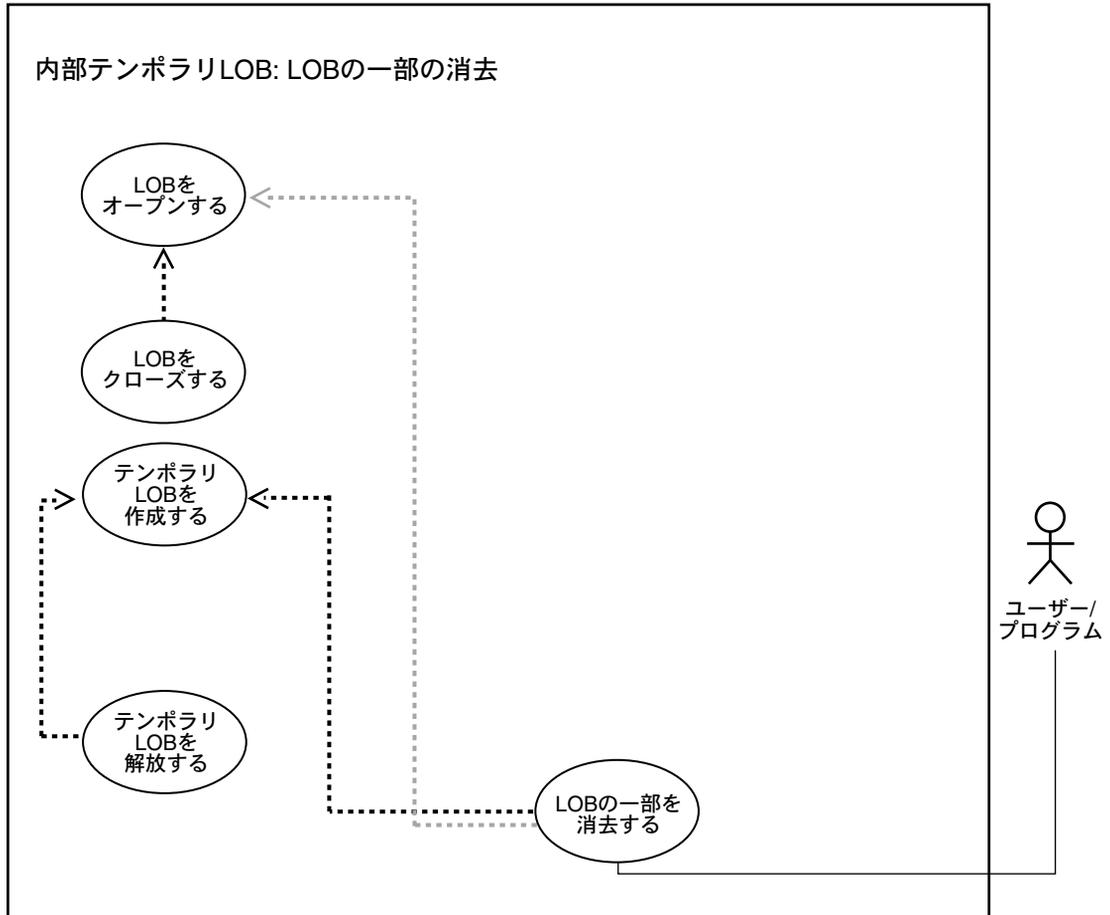
/* テンポラリ LOB を解放します。*/
EXEC SQL LOB FREE TEMPORARY :Temp_loc;

/* ロケータが保持しているリソースを解放します。*/
EXEC SQL FREE :Lob_loc;
EXEC SQL FREE :Temp_loc;
}

void main()
{
    char *samp = "samp/samp";
    EXEC SQL CONNECT :samp;
    trimTempLOB_proc();
    EXEC SQL ROLLBACK WORK RELEASE;
}
```

テンポラリ LOB の一部の消去

図 10-24 ユースケース図：テンポラリ LOB の一部の消去



参照： 内部テンポラリ LOB に関するすべての基本操作については、10-2 ページの「[ユースケース・モデル: 内部テンポラリ LOB](#)」を参照してください。

用途

テンポラリ LOB の一部を消去します。

使用上の注意

ありません。

構文

各プログラム環境で使用可能な機能のリストは、第3章「LOB プログラム環境」を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- PL/SQL (DBMS_LOB) : 『Oracle8i PL/SQL パッケージ・プロシージャ・リファレンス』の第18章「DBMS_LOB」の「サブプログラムの要約」の ERASE プロシージャ
- C (OCI) : 『Oracle8i コール・インタフェース・プログラマーズ・ガイド』の第15章「OCI リレーショナル関数」の「LOB 関数」の OCILobErase()
- COBOL (Pro*COBOL) : 『Oracle8i Pro*COBOL プリコンパイラ・プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB ERASE (実行可能埋込み SQL 拡張要素)」
- C/C++ (Pro*C/C++) : 『Oracle8i Pro*C/C++ プリコンパイラ・プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB ERASE (実行可能埋込み SQL 拡張要素)」
- Visual Basic (OO4O) : 参照マニュアルはありません。
- Java (JDBC) : 参照マニュアルはありません。

使用例

ありません。

例

次のプログラム環境の例が示されています。

- PL/SQL (DBMS_LOB パッケージ) : テンポラリ LOB の一部の消去 (10-167 ページ)
- C (OCI) : テンポラリ LOB の一部の消去 (10-168 ページ)
- COBOL (Pro*COBOL) : テンポラリ LOB の一部の消去 (10-171 ページ)
- C/C++ (Pro*C/C++) : テンポラリ LOB の一部の消去 (10-173 ページ)
- Visual Basic (OO4O) : 今回のリリースでは例は記載されていません。
- Java (JDBC) : 今回のリリースでは例は記載されていません。

PL/SQL (DBMS_LOB パッケージ) : テンポラリ LOB の一部の消去

```

/* 例のプロシージャ eraseTempLOB_proc は、DBMS_LOB パッケージの
一部ではないことに注意してください。*/
CREATE OR REPLACE PROCEDURE trimTempLOB_proc IS
  Lob_loc      CLOB;
  amt          number;
  Src_loc      BFILE := BFILENAME('AUDIO_DIR', 'Washington_audio');
  Amount      INTEGER := 32767;
BEGIN

  /* テンポラリ LOB を作成します。*/
  DBMS_LOB.CREATETEMPORARY(Lob_loc,TRUE);

  /* LOB のオープンはオプションです。*/
  DBMS_LOB.OPEN (Lob_loc, DBMS_LOB.LOB_READWRITE);

  /* テンポラリ LOB にいくつかのデータを移入します。*/
  Amount := 32767;
  DBMS_LOB.LOADFROMFILE(Lob_loc, Src_loc, Amount);
  /* LOB データを消去します。*/
  amt := 3000;
  DBMS_LOB.ERASE(Lob_loc, amt, 2);

  /* LOB をオープンしている場合、その LOB をクローズする必要があります。*/
  DBMS_LOB.CLOSE (Lob_loc);
  DBMS_LOB.CLOSE(Src_loc);
  DBMS_LOB.FREETEMPORARY(Lob_loc);
COMMIT;
EXCEPTION
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('Operation failed');
END;

```

C (OCI) : テンポラリ LOB の一部の消去

```

/* オフセット100 で、テンポラリ LOB 内の 2 バイトを消去します。*/
sb4 erase_temp_lobs ( OCIError      *errhp,
                     OCISvcCtx    *svchp,
                     OCIStmt      *stmthp,
                     OCIEnv       *envhp)
{
  OCILobLocator *tblob;
  OCILobLocator *bfile;

```

```
ub4 amt = 4000;
ub4 erase_size = 2;
ub4 erase_offset = 100;
sb4 return_code = 0;

printf("in erase\n");
if(OCIDescriptorAlloc((dvoid *)envhp, (dvoid **) &tblob,
                      (ub4) OCI_DTYPE_LOB,
                      (size_t) 0, (dvoid **) 0))
{
    printf("OCIDescriptor Alloc FAILED \n");
    return -1;
}

if(OCIDescriptorAlloc((dvoid *)envhp, (dvoid **) &bfile,
                      (ub4) OCI_DTYPE_FILE,
                      (size_t) 0, (dvoid **) 0))
{
    printf("OCIDescriptor Alloc FAILED \n");
    return -1;
}

/* BFILEがWashington_audio ファイルを指すように設定します。*/
if(OCILobFileNameSet(envhp, errhp, &bfile,
                    (text *)"AUDIO_DIR",
                    (ub2)strlen("AUDIO_DIR"),
                    (text *)"Washington_audio",
                    (ub2)strlen("Washington_audio")))
{
    printf("OCILobFileNameSet FAILED\n");
    return -1;
}

if (OCILobFileOpen(svchp, errhp, (OCILobLocator *) bfile, OCI_LOB_READONLY))
{
    printf( "OCILobFileOpen FAILED for the bfile\n");
    return_code = -1;
}

if(OCILobCreateTemporary(svchp, errhp, tblob, (ub2)0, SQLCS_IMPLICIT,
                        OCI_TEMP_BLOB, OCI_ATTR_NOCACHE, OCI_DURATION_SESSION))
{
    (void) printf("FAILED: CreateTemporary() \n");
    return -1;
}
```

```
if (OCILobOpen(svchp, errhp, (OCILobLocator *) tblob, OCI_LOB_READWRITE))
{
    printf( "OCILobOpen FAILED for temp LOB \n");
    return_code = -1;
}

/* テンポラリ LOB に 4,000 バイトのデータを移入します。 */
if (OCILobLoadFromFile(svchp,
                      errhp,
                      tblob,
                      (OCILobLocator*)bfile,
                      (ub4)amt,
                      (ub4)1, (ub4)1))
{
    printf( "OCILobLoadFromFile FAILED\n");
    return_code = -1;
}

if (OCILobErase(svchp, errhp, (OCILobLocator *) tblob, &erase_size,
                erase_offset))
{
    printf( "OCILobErase FAILED for temp LOB \n");
    return_code = -1;
} else
{
    printf( "OCILobErase succeeded for temp LOB \n");
}

if (OCILobClose(svchp, errhp, (OCILobLocator *) bfile))
{
    printf( "OCILobClose FAILED for bfile \n");
    return_code = -1;
}

if (OCILobClose(svchp, errhp, (OCILobLocator *) tblob))
{
    printf( "OCILobClose FAILED for temporary LOB \n");
    return_code = -1;
}

/* 使い終わったテンポラリ LOB を解放します。 */
if (OCILobFreeTemporary(svchp, errhp, tblob))
{
    printf("OCILobFreeTemporary FAILED \n");
}
```

```
    return_code = -1;
  }
  return return_code;
}
```

COBOL (Pro*COBOL) : テンポラリ LOB の一部の消去

```
IDENTIFICATION DIVISION.
PROGRAM-ID. TEMP-BLOB-ERASE.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

01  USERID    PIC X(11) VALUES "SAMP/SAMP".
01  TEMP-BLOB SQL-BLOB.
01  SRC-BFILE SQL-BFILE.
01  DIR-ALIAS PIC X(30) VARYING.
01  FNAME     PIC X(20) VARYING.
01  DIR-IND   PIC S9(4) COMP.
01  FNAME-IND PIC S9(4) COMP.
01  AMT       PIC S9(9) COMP VALUE 10.
01  POS       PIC S9(9) COMP VALUE 1.
01  ORASLNRD  PIC 9(4).

EXEC SQL INCLUDE SQLCA END-EXEC.
EXEC ORACLE OPTION (ORACA=YES) END-EXEC.
EXEC SQL INCLUDE ORACA END-EXEC.
PROCEDURE DIVISION.
TEMP-BLOB-ERASE.
EXEC SQL WHENEVER SQLERROR DO PERFORM SQL-ERROR END-EXEC.
EXEC SQL
CONNECT :USERID
END-EXEC.

* BLOB ロケータの割当ておよび初期化を行います。
EXEC SQL ALLOCATE :TEMP-BLOB END-EXEC.
EXEC SQL ALLOCATE :SRC-BFILE END-EXEC.
EXEC SQL LOB CREATE TEMPORARY :TEMP-BLOB END-EXEC.

* ディレクトリおよびファイル情報を設定します。
MOVE "AUDIO_DIR" TO DIR-ALIAS-ARR.
MOVE 9 TO DIR-ALIAS-LEN.
MOVE "Washington_audio" TO FNAME-ARR.
```

```
MOVE 16 TO FNAME-LEN.
```

```
EXEC SQL
  LOB FILE SET :SRC-BFILE DIRECTORY = :DIR-ALIAS,
  FILENAME = :FNAME
END-EXEC.
```

- * ソース BFILE および宛先テンポラリ BLOB をオープンします。

```
EXEC SQL LOB OPEN :TEMP-BLOB READ WRITE END-EXEC.
EXEC SQL LOB OPEN :SRC-BFILE READ ONLY END-EXEC.
EXEC SQL
  LOB LOAD :AMT FROM FILE :SRC-BFILE INTO :TEMP-BLOB
END-EXEC.
```

- * いくつかの LOB データを消去します。

```
EXEC SQL
  LOB ERASE :AMT FROM :TEMP-BLOB AT :POS
END-EXEC.
```

- * LOB をクローズします。

```
EXEC SQL LOB CLOSE :SRC-BFILE END-EXEC.
EXEC SQL LOB CLOSE :TEMP-BLOB END-EXEC.
```

- * テンポラリ LOB を解放します。

```
EXEC SQL
  LOB FREE TEMPORARY :TEMP-BLOB
END-EXEC.
```

- * LOB ロケータを解放します。

```
EXEC SQL FREE :TEMP-BLOB END-EXEC.
EXEC SQL FREE :SRC-BFILE END-EXEC.
STOP RUN.
```

```
SQL-ERROR.
```

```
EXEC SQL
  WHENEVER SQLERROR CONTINUE
END-EXEC.
MOVE ORASLNR TO ORASLNRD.
DISPLAY " ".
DISPLAY "ORACLE ERROR DETECTED ON LINE ", ORASLNRD, ":".
DISPLAY " ".
DISPLAY SQLERRMC.
EXEC SQL ROLLBACK WORK RELEASE END-EXEC.
STOP RUN.
```

C/C++ (Pro*C/C++) : テンポラリ LOB の一部の消去

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

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

void eraseTempLOB_proc()
{
    OCIBlobLocator *Temp_loc;
    OCIFileLocator *Lob_loc;
    char *Dir = "AUDIO_DIR", *Name = "Washington_audio";
    int Amount;
    int Position = 1024;

    EXEC SQL WHENEVER SQLERROR DO Sample_Error();

    /* テンポラリ LOB の割当ておよび作成を行います。 */
    EXEC SQL ALLOCATE :Temp_loc;
    EXEC SQL LOB CREATE TEMPORARY :Temp_loc;

    /* BFILE ロケータの割当ておよび初期化を行います。 */
    EXEC SQL ALLOCATE :Lob_loc;
    EXEC SQL LOB FILE SET :Lob_loc DIRECTORY = :Dir, FILENAME = :Name;

    /* LOB のオープンはオプションです。 */
    EXEC SQL LOB OPEN :Lob_loc READ ONLY;
    EXEC SQL LOB OPEN :Temp_loc READ WRITE;

    /* BFILE からテンポラリ LOB に、指定した量をロードします。 */
    Amount = 4096;
    EXEC SQL LOB LOAD :Amount FROM FILE :Lob_loc INTO :Temp_loc;

    /* 任意の位置で、テンポラリ LOB から指定した量を消去します。 */
    Amount = 2048;
    EXEC SQL LOB ERASE :Amount FROM :Temp_loc AT :Position;
```

```
/* LOB をオープンしている場合、その LOB をクローズする必要があります。*/
EXEC SQL LOB CLOSE :Lob_loc;
EXEC SQL LOB CLOSE :Temp_loc;

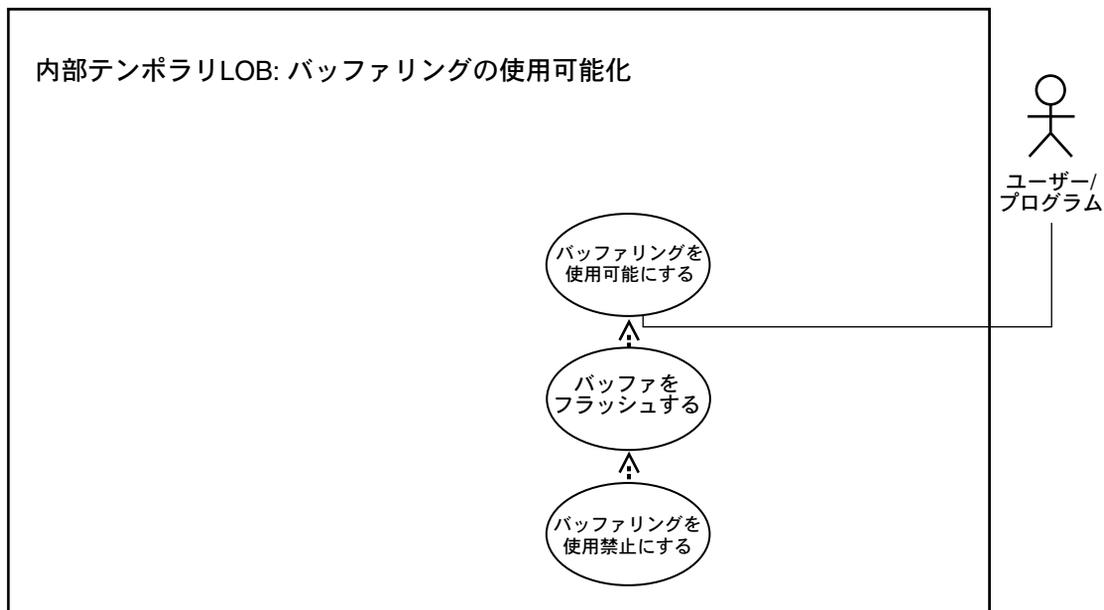
/* テンポラリ LOB を解放します。*/
EXEC SQL LOB FREE TEMPORARY :Temp_loc;

/* ロケータが保持しているリソースを解放します。*/
EXEC SQL FREE :Lob_loc;
EXEC SQL FREE :Temp_loc;
}

void main()
{
    char *samp = "samp/samp";
    EXEC SQL CONNECT :samp;
    eraseTempLOB_proc();
    EXEC SQL ROLLBACK WORK RELEASE;
}
```

テンポラリ LOB に対する LOB バッファリングの使用可能化

図 10-25 ユースケース図：テンポラリ LOB に対する LOB バッファリングの使用可能化



参照： 内部テンポラリ LOB に関するすべての基本操作については、10-2 ページの「ユースケース・モデル：内部テンポラリ LOB」を参照してください。

用途

テンポラリ LOB に対する LOB バッファリングを使用可能にします。

使用上の注意

一連の少量の読み書きを実行する場合は、バッファリングを使用可能にします。これらの作業を完了したら、他の LOB 操作を行う前にバッファリングを使用禁止にする必要があります。

注意： チェックインおよびチェックアウトを伴うストリーム読み込みおよび書込みを実行する際は、バッファリングを使用可能にしないでください。

構文

各プログラム環境で使用可能な機能のリストは、[第 3 章「LOB プログラム環境」](#)を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- PL/SQL (DBMS_LOB) : 参照マニュアルはありません。
- C (OCI) : 『Oracle8i コール・インタフェース・プログラマーズ・ガイド』の第 15 章「OCI リレーショナル関数」の「LOB 関数」の OCIEnableLobBuffering()、OCIDisableLobBuffering()、OCILobFlushBuffer()
- COBOL (Pro*COBOL) : 『Oracle8i Pro*COBOL プリコンパイラ・プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB ENABLE BUFFERING (実行可能埋込み SQL 拡張要素)」
- C/C++ (Pro*C/C++) : 『Oracle8i Pro*C/C++ プリコンパイラ・プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB ENABLE BUFFERING (実行可能埋込み SQL 拡張要素)」
- Visual Basic (OO4O) : 参照マニュアルはありません。
- Java (JDBC) : 参照マニュアルはありません。

使用例

ありません。

例

次のプログラム環境の例が示されています。

- PL/SQL (DBMS_LOB パッケージ) : 今回のリリースでは例は記載されていません。
- C (OCI) : [テンポラリ LOB に対する LOB バッファリングの使用可能化 \(10-177 ページ\)](#)
- COBOL (Pro*COBOL) : [テンポラリ LOB に対する LOB バッファリングの使用可能化 \(10-178 ページ\)](#)
- C/C++ (Pro*C/C++) : [テンポラリ LOB に対する LOB バッファリングの使用可能化 \(10-180 ページ\)](#)
- Visual Basic (OO4O) : 今回のリリースでは例は記載されていません。

- Java (JDBC) : 今回のリリースでは例は記載されていません。

C (OCI) : テンポラリ LOB に対する LOB バッファリングの使用可能化

```

#define MAXBUFLLEN 32767
sb4 lobBuffering (envhp, errhp, svchp, stmthp)
OCIEnv      *envhp;
OCIError    *errhp;
OCISvcCtx   *svchp;
OCIStmt     *stmthp;
{
    OCILobLocator *tblob;
    ub4 amt;
    ub4 offset;
    sword retval;
    ub1 bufp[MAXBUFLLEN];
    ub4 buflen;

    /* LOB ロケータの記述子を割り当てます。*/
    (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &tblob,
                              (ub4)OCI_DTYPE_LOB, (size_t) 0, (dvoid **) 0);

    /* BLOB を選択します。*/
    printf (" create a temporary Lob\n");
    /* テンポラリ LOB を作成します。*/
    if (OCILobCreateTemporary(svchp, errhp, tblob, (ub2)0, SQLCS_IMPLICIT,
                              OCI_TEMP_BLOB,
                              OCI_ATTR_NOCACHE,
                              OCI_DURATION_SESSION))
    {
        (void) printf("FAILED: CreateTemporary() \n");
        return -1;
    }

    /* BLOB をオープンします。*/
    if (OCILobOpen(svchp, errhp, (OCILobLocator *) tblob, OCI_LOB_READWRITE))
    {
        printf ("OCILobOpen FAILED for temp LOB \n");
        return -1;
    }

    /* LOB バッファリングを使用可能にします。*/
    printf (" enable LOB buffering\n");
    checkerr (errhp, OCILobEnableBuffering(svchp, errhp, tblob));

    printf (" write data to LOB\n");

```

```
/* LOB にデータを書き込みます。*/
amt      = sizeof (bufp);
bufllen  = sizeof (bufp);
offset   = 1;
checkerr (errhp, OCILobWrite (svchp, errhp, tblob, &amt,
                              offset, bufp, bufllen,
                              OCI_ONE_PIECE, (dvoid *)0,
                              (sb4 (*) (dvoid*, dvoid*, ub4*, ub1 *))0,
                              0, SQLCS_IMPLICIT));

/* バッファをフラッシュします。*/
printf(" flush the LOB buffers\n");
checkerr (errhp, OCILobFlushBuffer (svchp, errhp, tblob,
                                     (ub4)OCI_LOB_BUFFER_FREE));

/* バッファリングを使用禁止にします。*/
printf (" disable LOB buffering\n");
checkerr (errhp, OCILobDisableBuffering (svchp, errhp, tblob));

/* 後続の LOB 書込みでは、LOB バッファリング・サブシステムが使用されません。*/

/* BLOB をオープンしている場合、その BLOB をクローズする必要があります。*/
checkerr (errhp, OCILobClose (svchp, errhp, tblob));

/* 使い終わったテンポラリ LOB を解放します。*/
if (OCILobFreeTemporary (svchp, errhp, tblob))
{
    printf ("OCILobFreeTemporary FAILED \n");
    return -1;
}

/* ロケータが保持しているリソースを解放します。*/
(void) OCIDescriptorFree ((dvoid *) tblob, (ub4) OCI_DTYPE_LOB);

return;
}
```

COBOL (Pro*COBOL) : テンポラリ LOB に対する LOB バッファリングの使用可能化

```
IDENTIFICATION DIVISION.
PROGRAM-ID. TEMP-LOB-BUFFERING.
ENVIRONMENT DIVISION.
DATA DIVISION.
```

WORKING-STORAGE SECTION.

```
01 USERID    PIC X(11) VALUES "SAMP/SAMP".
01 TEMP-BLOB          SQL-BLOB.
01 BUFFER           PIC X(80).
01 AMT              PIC S9(9) COMP VALUE 10.
01 ORASLNRD        PIC 9(4).
```

```
EXEC SQL VAR BUFFER IS RAW(80) END-EXEC.
EXEC SQL INCLUDE SQLCA END-EXEC.
EXEC ORACLE OPTION (ORACA=YES) END-EXEC.
EXEC SQL INCLUDE ORACA END-EXEC.
```

PROCEDURE DIVISION.

TEMP-LOB-BUFFERING.

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

* CLOB ロケータの割当ておよび初期化を行います。

```
EXEC SQL ALLOCATE :TEMP-BLOB END-EXEC.
EXEC SQL
    LOB CREATE TEMPORARY :TEMP-BLOB
END-EXEC.
```

* テンポラリ LOB に対するバッファリングを使用可能にします。

```
EXEC SQL LOB ENABLE BUFFERING :TEMP-BLOB END-EXEC.
```

* テンポラリ LOB にいくつかのデータを書き込みます。

```
MOVE '2525252626262525' TO BUFFER.
EXEC SQL
    LOB WRITE ONE :AMT FROM :BUFFER
    INTO :TEMP-BLOB END-EXEC
```

* バッファリングされた書き込みをフラッシュします。

```
EXEC SQL
    LOB FLUSH BUFFER :TEMP-BLOB FREE END-EXEC.
```

* テンポラリ LOB に対するバッファリングを使用禁止にします。

```
EXEC SQL
    LOB DISABLE BUFFERING :TEMP-BLOB
END-EXEC.
EXEC SQL
```

```
        LOB FREE TEMPORARY :TEMP-BLOB
    END-EXEC.
EXEC SQL FREE :TEMP-BLOB END-EXEC.
STOP RUN.

SQL-ERROR.
EXEC SQL WHENEVER SQLERROR CONTINUE END-EXEC.
MOVE ORASLNR TO ORASLNRD.
DISPLAY " ".
DISPLAY "ORACLE ERROR DETECTED ON LINE ", ORASLNRD, ":".
DISPLAY " ".
DISPLAY SQLERRMC.
EXEC SQL ROLLBACK WORK RELEASE END-EXEC.
STOP RUN.
```

C/C++ (Pro*C/C++) : テンポラリ LOB に対する LOB バッファリングの使用可能化

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

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

#define BufferLength 1024

void enableBufferingTempLOB_proc()
{
    OCIClobLocator *Temp_loc;
    varchar Buffer[BufferLength];
    int Amount = BufferLength;
    int multiple, Length = 0, Position = 1;

    EXEC SQL WHENEVER SQLERROR DO Sample_Error();
    /* テンポラリ LOB の割当ておよび作成を行います。*/
    EXEC SQL ALLOCATE :Temp_loc;
    EXEC SQL LOB CREATE TEMPORARY :Temp_loc;
```

```

/* LOB バッファリング・サブシステムを使用可能にします。*/
EXEC SQL LOB ENABLE BUFFERING :Temp_loc;
memset((void *)Buffer.arr, 42, BufferLength);
Buffer.len = BufferLength;
for (multiple = 0; multiple < 8; multiple++)
{
    /* テンポラリ LOB にデータを書き込みます。*/
    EXEC SQL LOB WRITE ONE :Amount
        FROM :Buffer INTO :Temp_loc AT :Position;
    Position += BufferLength;
}

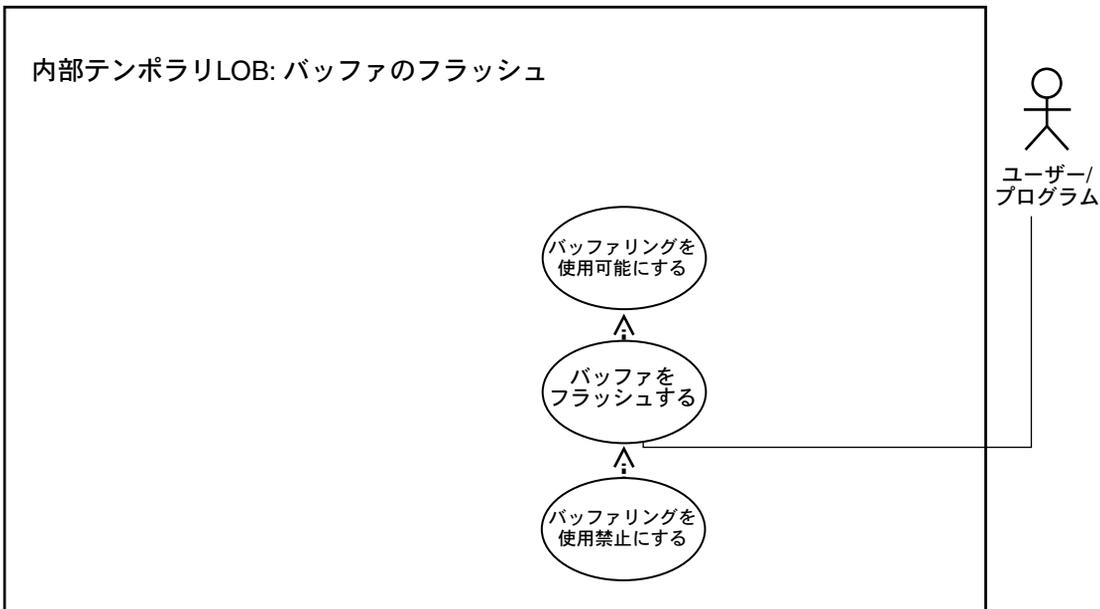
/* バッファの内容をフラッシュし、それらのリソースを解放します。*/
EXEC SQL LOB FLUSH BUFFER :Temp_loc FREE;
/* LOB バッファリング・サブシステムを使用禁止にします。*/
EXEC SQL LOB DISABLE BUFFERING :Temp_loc;
EXEC SQL LOB DESCRIBE :Temp_loc GET LENGTH INTO :Length;
printf("Wrote %d characters using the Buffering Subsystem\n", Length);

/* テンポラリ LOB を解放します。*/
EXEC SQL LOB FREE TEMPORARY :Temp_loc;
/* ロケータが保持しているリソースを解放します。*/
EXEC SQL FREE :Temp_loc;
}
void main()
{
    char *samp = "samp/samp";
    EXEC SQL CONNECT :samp;
    enableBufferingTempLOB_proc();
    EXEC SQL ROLLBACK WORK RELEASE;
}

```

テンポラリ LOB に対するバッファのフラッシュ

図 10-26 ユースケース図：テンポラリ LOB に対するバッファのフラッシュ



参照： 内部テンポラリ LOB に関するすべての基本操作については、10-2 ページの「ユースケース・モデル: 内部テンポラリ LOB」を参照してください。

用途

テンポラリ LOB に対してバッファをフラッシュします。

使用上の注意

ありません。

構文

各プログラム環境で使用可能な機能のリストは、第 3 章「LOB プログラム環境」を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- PL/SQL (DBMS_LOB) : 参照マニュアルはありません。
- C (OCI) : 『Oracle8i コール・インタフェース・プログラマーズ・ガイド』の第 15 章「OCI リレーショナル関数」の「LOB 関数」の OCILobFlushBuffer()
- COBOL (Pro*COBOL) : 『Oracle8i Pro*COBOL プリコンパイラ・プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB FLUSH BUFFER (実行可能埋込み SQL 拡張要素)」
- C/C++ (Pro*C/C++) : 『Oracle8i Pro*C/C++ プリコンパイラ・プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB FLUSH BUFFER (実行可能埋込み SQL 拡張要素)」
- Visual Basic (OO4O) : 参照マニュアルはありません。
- Java (JDBC) : 参照マニュアルはありません。

使用例

ありません。

例

次のプログラム環境の例が示されています。

- PL/SQL (DBMS_LOB パッケージ) : 今回のリリースでは例は記載されていません。
- C (OCI) : [テンポラリ LOB に対するバッファのフラッシュ \(10-82 ページ\)](#)
- COBOL (Pro*COBOL) : [テンポラリ LOB 内のパターンの有無の確認 \(instr\) \(10-84 ページ\)](#)
- C/C++ (Pro*C/C++) : [テンポラリ LOB 内のパターンの有無の確認 \(instr\) \(10-86 ページ\)](#)
- Visual Basic (OO4O) : 今回のリリースでは例は記載されていません。
- Java (JDBC) : 今回のリリースでは例は記載されていません。

C (OCI) : テンポラリ LOB に対するバッファのフラッシュ

```

sb4 lobBuffering (envhp, errhp, svchp, stmthp)
OCIEnv      *envhp;
OCIError    *errhp;
OCISvcCtx   *svchp;
OCIStmt     *stmthp;
{
    OCILobLocator *tblob;
    ub4 amt;
    ub4 offset;

```

```

sword retval;
ub1 bufp[MAXBUFLen];
ub4 buflen;

/* LOB ロケータの記述子を割り当てます。*/
(void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &tblob,
                          (ub4)OCI_DTYPE_LOB, (size_t) 0, (dvoid **) 0);

/* BLOB を選択します。*/
printf (" create a temporary Lob\n");
/* テンポラリ LOB を作成します。*/
if(OCILobCreateTemporary(svchp, errhp, tblob, (ub2)0,
                          SQLCS_IMPLICIT, OCI_TEMP_BLOB,
                          OCI_ATTR_NOCACHE, OCI_DURATION_SESSION))
{
    (void) printf("FAILED: CreateTemporary() \n");
    return -1;
}

/* BLOB をオープンします。*/
if (OCILobOpen(svchp, errhp, (OCILobLocator *) tblob, OCI_LOB_READWRITE))
{
    printf( "OCILobOpen FAILED for temp lob \n");
    return -1;
}

/* LOB バッファリングを使用可能にします。*/
printf (" enable LOB buffering\n");
checkerr (errhp, OCILobEnableBuffering(svchp, errhp, tblob));

printf (" write data to LOB\n");

/* LOB にデータを書き込みます。*/
amt    = sizeof(bufp);
buflen = sizeof(bufp);
offset = 1;
checkerr (errhp, OCILobWrite (svchp, errhp, tblob, &amt,
                              offset, bufp, buflen,
                              OCI_ONE_PIECE, (dvoid *)0,
                              (sb4 *) (dvoid*,dvoid*,ub4*,ub1 *)0,
                              0, SQLCS_IMPLICIT));

/* バッファをフラッシュします。*/
printf(" flush the LOB buffers\n");
checkerr (errhp, OCILobFlushBuffer(svchp, errhp, tblob,
                                  (ub4)OCI_LOB_BUFFER_FREE));

/* バッファリングを使用禁止にします。*/
printf (" disable LOB buffering\n");

```

```

checkerr (errhp, OCILobDisableBuffering(svchp, errhp, tblob));

/* 後続の LOB 書き込みでは、LOB バッファリング・サブシステムが使用されません。*/

/* BLOB をオープンしている場合、その BLOB をクローズする必要があります。*/
checkerr (errhp, OCILobClose(svchp, errhp, tblob));

/* 使い終わったテンポラリ LOB を解放します。*/
if (OCILobFreeTemporary(svchp, errhp, tblob))
{
    printf("OCILobFreeTemporary FAILED \n");
    return -1;
}

/* ロケータが保持しているリソースを解放します。*/
(void) OCIDescriptorFree((dvoid *) tblob, (ub4) OCI_DTYPE_LOB);

return;
}

```

COBOL (Pro*COBOL) : テンポラリ LOB に対するバッファのフラッシュ

```

IDENTIFICATION DIVISION.
PROGRAM-ID. FREE-TEMPORARY.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

01  USERID    PIC X(11) VALUES "SAMP/SAMP".

01  TEMP-BLOB    SQL-BLOB.
01  IS-TEMP     PIC S9(9) COMP.
01  ORASLNRD    PIC 9(4).

EXEC SQL INCLUDE SQLCA END-EXEC.
EXEC ORACLE OPTION (ORACA=YES) END-EXEC.
EXEC SQL INCLUDE ORACA END-EXEC.

PROCEDURE DIVISION.
FREE-TEMPORARY.

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

```

```
* BLOB ロケータの割当ておよび初期化を行います。
EXEC SQL ALLOCATE :TEMP-BLOB END-EXEC.
EXEC SQL
      LOB CREATE TEMPORARY :TEMP-BLOB
END-EXEC.

* テンポラリ LOB に何らかの操作を行います。

* テンポラリ LOB を解放します。
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++) : テンポラリ LOB に対するバッファのフラッシュ

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

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

#define BufferLength 1024

void flushBufferingTempLOB_proc()
{
```

```
OCIClobLocator *Temp_loc;
varchar Buffer[BufferLength];
int Amount = BufferLength;
int multiple, Length = 0, Position = 1;

EXEC SQL WHENEVER SQLERROR DO Sample_Error();
/* テンポラリ LOB の割当ておよび作成を行います。*/
EXEC SQL ALLOCATE :Temp_loc;
EXEC SQL LOB CREATE TEMPORARY :Temp_loc;
/* LOB バッファリング・サブシステムを使用可能にします。*/
EXEC SQL LOB ENABLE BUFFERING :Temp_loc;
memset((void *)Buffer.arr, 42, BufferLength);
Buffer.len = BufferLength;
for (multiple = 0; multiple < 8; multiple++)
{
    /* テンポラリ LOB にデータを書き込みます。*/
    EXEC SQL LOB WRITE ONE :Amount
        FROM :Buffer INTO :Temp_loc AT :Position;
        Position += BufferLength;
}
/* バッファの内容をフラッシュし、それらのリソースを解放します。*/
EXEC SQL LOB FLUSH BUFFER :Temp_loc FREE;

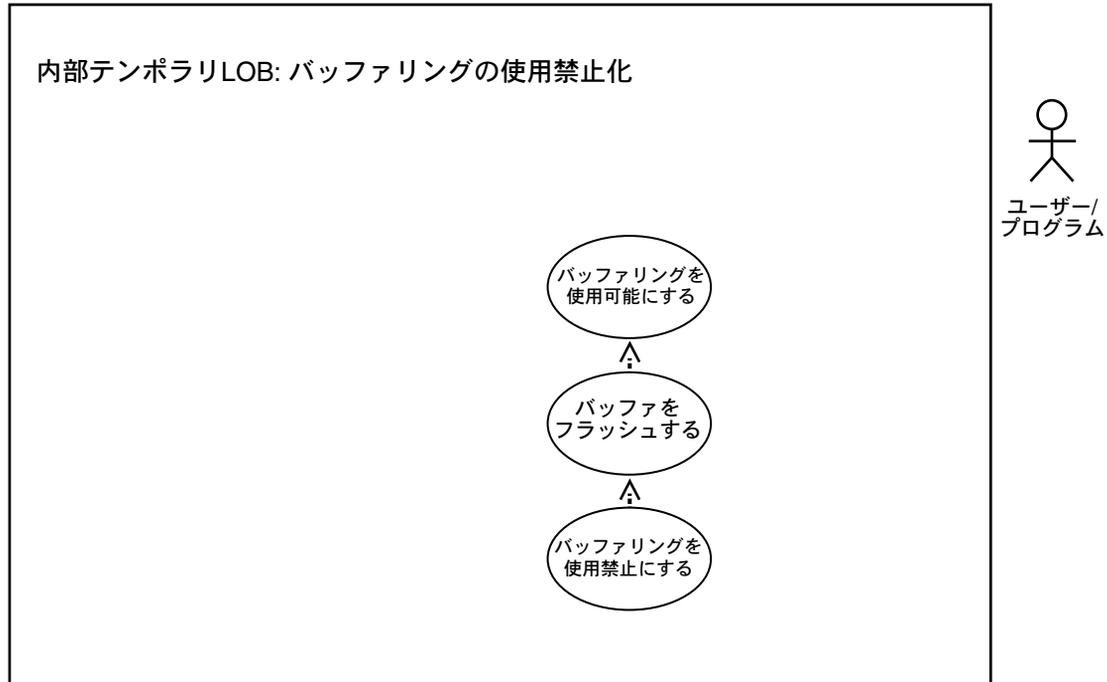
/* LOB バッファリング・サブシステムを使用禁止にします。*/
EXEC SQL LOB DISABLE BUFFERING :Temp_loc;
EXEC SQL LOB DESCRIBE :Temp_loc GET LENGTH INTO :Length;
printf("Wrote %d characters using the Buffering Subsystem\n", Length);

/* テンポラリ LOB を解放します。*/
EXEC SQL LOB FREE TEMPORARY :Temp_loc;
/* ロケータが保持しているリソースを解放します。*/
EXEC SQL FREE :Temp_loc;
}

void main()
{
    char *samp = "samp/samp";
    EXEC SQL CONNECT :samp;
    flushBufferingTempLOB_proc();
    EXEC SQL ROLLBACK WORK RELEASE;
}
```

テンポラリ LOB に対する LOB バッファリングの使用禁止化

図 10-27 ユースケース図: LOB バッファリングの使用禁止化



参照: 内部テンポラリ LOB に関するすべての基本操作については、10-2 ページの「ユースケース・モデル: 内部テンポラリ LOB」を参照してください。

用途

テンポラリ LOB バッファリングを使用禁止にします。

使用上の注意

一連の少量の読み書きを実行する場合は、バッファリングを使用可能にします。これらの作業を完了したら、他の LOB 操作を行う前にバッファリングを使用禁止にする必要があります。

注意： チェックインおよびチェックアウトを伴うストリーム読みおよび書き込みを実行する際は、バッファリングを使用可能にしないでください。

構文

各プログラム環境で使用可能な機能のリストは、[第3章「LOB プログラム環境」](#)を参照してください。各プログラム環境における構文については、次のマニュアルを参照してください。

- PL/SQL (DBMS_LOB) : 参照マニュアルはありません。
- C (OCI) : 『Oracle8i コール・インタフェース・プログラマーズ・ガイド』の第15章「OCI リレーショナル関数」の「LOB 関数」の OCILobDisableBuffering()
- COBOL (Pro*COBOL) : 『Oracle8i Pro*COBOL プリコンパイラ・プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB DISABLE BUFFERING (実行可能埋込み SQL 拡張要素)」
- C/C++ (Pro*C/C++) : 『Oracle8i Pro*C/C++ プリコンパイラ・プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB DISABLE BUFFERING (実行可能埋込み SQL 拡張要素)」
- Visual Basic (OO4O) : 参照マニュアルはありません。
- Java (JDBC) : 参照マニュアルはありません。

使用例

ありません。

例

次のプログラム環境の例が示されています。

- PL/SQL (DBMS_LOB パッケージ) : 今回のリリースでは例は記載されていません。
- C (OCI) : [テンポラリ LOB に対する LOB バッファリングの使用禁止化](#) (10-189 ページ)
- COBOL (Pro*COBOL) : [テンポラリ LOB に対する LOB バッファリングの使用禁止化](#) (10-191 ページ)
- C/C++ (Pro*C/C++) : [テンポラリ LOB に対する LOB バッファリングの使用禁止化](#) (10-193 ページ)
- Visual Basic (OO4O) : 今回のリリースでは例は記載されていません。
- Java (JDBC) : 今回のリリースでは例は記載されていません。

C (OCI) : テンポラリ LOB に対する LOB バッファリングの使用禁止化

```

sb4 lobBuffering (envhp, errhp, svchp, stmthp)
OCIEnv    *envhp;
OCIError  *errhp;
OCISvcCtx *svchp;
OCIStmt   *stmthp;
{
    OCILobLocator *tblob;
    ub4 amt;
    ub4 offset;
    sword retval;
    ub1 bufp[MAXBUFLLEN];
    ub4 buflen;

    /* LOB ロケータの記述子を割り当てます。*/
    (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &tblob,
                              (ub4)OCI_DTYPE_LOB, (size_t) 0, (dvoid **) 0);

    /* BLOB を選択します。*/
    printf (" create a temporary Lob\n");
    /* テンポラリ LOB を作成します。*/
    if(OCILobCreateTemporary(svchp,errhp, tblob, (ub2)0, SQLCS_IMPLICIT,
                             OCI_TEMP_BLOB,
                             OCI_ATTR_NOCACHE,
                             OCI_DURATION_SESSION))
    {
        (void) printf("FAILED: CreateTemporary() \n");
        return -1;
    }

    /* BLOB をオープンします。*/
    if (OCILobOpen(svchp, errhp, (OCILobLocator *) tblob, OCI_LOB_READWRITE))
    {
        printf( "OCILobOpen FAILED for temp LOB \n");
        return -1;
    }

    /* LOB バッファリングを使用可能にします。*/
    printf (" enable LOB buffering\n");
    checkerr (errhp, OCILobEnableBuffering(svchp, errhp, tblob));

    printf (" write data to LOB\n");

```

```

/* LOB にデータを書き込みます。 */
amt      = sizeof(bufp);
buflen  = sizeof(bufp);
offset   = 1;
checkerr (errhp, OCILobWrite (svchp, errhp, tblob, &amt,
                              offset, bufp, buflen,
                              OCI_ONE_PIECE, (dvoid *)0,
                              (sb4 (*) (dvoid*,dvoid*,ub4*,ub1 *))0,
                              0, SQLCS_IMPLICIT));

/* バッファをフラッシュします。 */
printf(" flush the LOB buffers\n");
checkerr (errhp, OCILobFlushBuffer(svchp, errhp, tblob,
                                   (ub4)OCI_LOB_BUFFER_FREE));

/* バッファリングを使用禁止にします。 */
printf(" disable LOB buffering\n");
checkerr (errhp, OCILobDisableBuffering(svchp, errhp, tblob));

/* 後続の LOB 書込みでは、LOB バッファリング・サブシステムが使用されません。 */

/* BLOB をオープンしている場合、その BLOB をクローズする必要があります。 */
checkerr (errhp, OCILobClose(svchp, errhp, tblob));

/* 使い終わったテンポラリ LOB を解放します。 */
if (OCILobFreeTemporary(svchp, errhp, tblob))
{
    printf("OCILobFreeTemporary FAILED \n");
    return -1;
}

/* ロケータが保持しているリソースを解放します。 */
(void) OCIDescriptorFree((dvoid *) tblob, (ub4) OCI_DTYPE_LOB);

return;
}

```

COBOL (Pro*COBOL) : テンポラリ LOB に対する LOB バッファリングの使用禁止化

```

IDENTIFICATION DIVISION.
PROGRAM-ID. TEMP-LOB-BUFFERING.
ENVIRONMENT DIVISION.
DATA DIVISION.

```

テンポラリ LOB に対する LOB バッファリングの使用禁止化

WORKING-STORAGE SECTION.

```
01  USERID    PIC X(11) VALUES "SAMP/SAMP".
01  TEMP-BLOB          SQL-BLOB.
01  BUFFER          PIC X(80) .
01  AMT            PIC S9(9) COMP VALUE 10.
01  ORASLNDR          PIC 9(4) .
```

```
EXEC SQL VAR BUFFER IS RAW(80) END-EXEC.
EXEC SQL INCLUDE SQLCA END-EXEC.
EXEC ORACLE OPTION (ORACA=YES) END-EXEC.
EXEC SQL INCLUDE ORACA END-EXEC.
```

PROCEDURE DIVISION.

TEMP-LOB-BUFFERING.

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

* CLOB ロケータの割当ておよび初期化を行います。

```
EXEC SQL ALLOCATE :TEMP-BLOB END-EXEC.
EXEC SQL
      LOB CREATE TEMPORARY :TEMP-BLOB
END-EXEC.
```

* テンポラリ LOB のバッファリングを使用可能にします。

```
EXEC SQL
      LOB ENABLE BUFFERING :TEMP-BLOB
END-EXEC.
```

* テンポラリ LOB にいくつかのデータを書き込みます。

```
MOVE '252525262626252525' TO BUFFER.
EXEC SQL
      LOB WRITE ONE :AMT FROM :BUFFER
      INTO :TEMP-BLOB
END-EXEC
```

* バッファリングされた書き込みをフラッシュします。

```
EXEC SQL
      LOB FLUSH BUFFER :TEMP-BLOB FREE
END-EXEC.
```

* テンポラリ LOB のバッファリングを使用禁止にします。

```
EXEC SQL
    LOB DISABLE BUFFERING :TEMP-BLOB
END-EXEC.
EXEC SQL
    LOB FREE TEMPORARY :TEMP-BLOB
END-EXEC.

EXEC SQL FREE :TEMP-BLOB END-EXEC.
STOP RUN.
```

```
SQL-ERROR.
EXEC SQL WHENEVER SQLERROR CONTINUE END-EXEC.
MOVE ORASLNR TO ORASLNRD.
DISPLAY " ".
DISPLAY "ORACLE ERROR DETECTED ON LINE ", ORASLNRD, ":".
DISPLAY " ".
DISPLAY SQLERRMC.
EXEC SQL ROLLBACK WORK RELEASE END-EXEC.
STOP RUN.
```

C/C++ (Pro*C/C++) : テンポラリ LOB に対する LOB バッファリングの使用禁止化

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

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

#define BufferLength 1024

void disableBufferingTempLOB_proc()
{
    OCIClobLocator *Temp_loc;
    varchar Buffer[BufferLength];
    int Amount = BufferLength;
    int multiple, Length = 0, Position = 1;
```

テンポラリ LOB に対する LOB バッファリングの使用禁止化

```
EXEC SQL WHENEVER SQLERROR DO Sample_Error();

/* テンポラリ LOB の割当ておよび作成を行います。*/
EXEC SQL ALLOCATE :Temp_loc;
EXEC SQL LOB CREATE TEMPORARY :Temp_loc;

/* LOB バッファリング・サブシステムを使用可能にします。*/
EXEC SQL LOB ENABLE BUFFERING :Temp_loc;
memset((void *)Buffer.arr, 42, BufferLength);
Buffer.len = BufferLength;
for (multiple = 0; multiple < 7; multiple++)
{
    /* テンポラリ LOB にデータを書き込みます。*/
    EXEC SQL LOB WRITE ONE :Amount
        FROM :Buffer INTO :Temp_loc AT :Position;
    Position += BufferLength;
}

/* バッファの内容をフラッシュし、それらのリソースを解放します。*/
EXEC SQL LOB FLUSH BUFFER :Temp_loc FREE;

/* LOB バッファリング・サブシステムを使用禁止にします。*/
EXEC SQL LOB DISABLE BUFFERING :Temp_loc;

/* バッファリングが使用禁止の場合のみ、追加の書き込みを行うことができます。*/
EXEC SQL LOB WRITE APPEND ONE :Amount FROM :Buffer INTO :Temp_loc;
EXEC SQL LOB DESCRIBE :Temp_loc GET LENGTH INTO :Length;

printf("Wrote a total of %d characters\n", Length);

/* テンポラリ LOB を解放します。*/
EXEC SQL LOB FREE TEMPORARY :Temp_loc;

/* ロケータが保持しているリソースを解放します。*/
EXEC SQL FREE :Temp_loc;
}

void main()
{
    char *samp = "samp/samp";
    EXEC SQL CONNECT :samp;
    disableBufferingTempLOB_proc();
    EXEC SQL ROLLBACK WORK RELEASE;
}
```

外部 LOB (BFILE)

ユースケース・モデル

この章では、外部 LOB に対する操作 (BFILE からのデータの読み込みなど) を、ユースケースの点から説明します。表 11-1 「ユースケース・モデル:外部 LOB (BFILE)」に、すべてのユースケースを示します。

ユースケース・モデルの図形概要

「ユースケース・モデル図:外部 LOB (BFILE)」に、ユースケースおよびその相互関係を示します。

個々のユースケース

外部 LOB (BFILE) の各ユースケースは、次のように説明されています。

- **ユースケース図**: ユースケースを表す図 (図の見方については、「はじめに」の「図の解釈方法」を参照)
- **用途**: LOB に関するこのユースケースの用途
- **使用上の注意**: LOB 操作の実装に有効なガイドライン
- **構文**: 様々なプログラム環境における、ユースケースに対する LOB 関連操作の基礎となる構文
- **使用例**: 仮想マルチメディア・アプリケーションのユースケースの実装の例を表した使用例。第 8 章「サンプル・アプリケーション」を参照してください。
- **例**: 各プログラム環境でのユースケースの例。これらは、第 8 章「サンプル・アプリケーション」で説明するマルチメディア・アプリケーションおよび Multimedia_tab 表を基にしています。

ユースケース・モデル: 外部 LOB (BFILE)

表 11-1 「ユースケース・モデル: 外部 LOB (BFILE)」の「+」は、特定のユースケースで、プログラム環境の例が提供されているものを示します（詳細および関連マニュアルは、第 3 章「LOB プログラム環境」を参照）。

この表では、プログラム環境を次の略称で表しています。

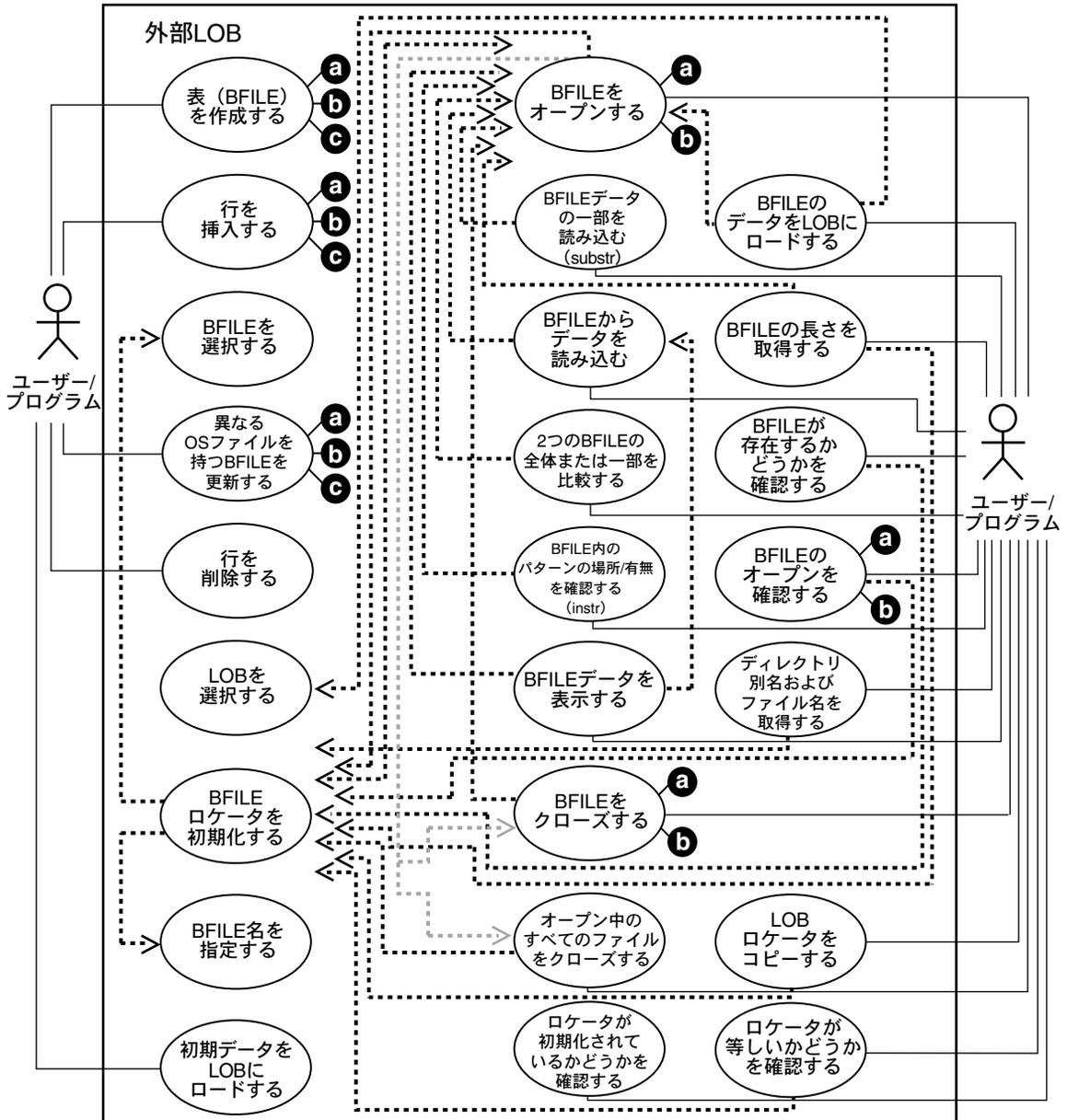
- P – DBMS_LOB パッケージを使用した PL/SQL
- O – OCI (Oracle Call Interface) を使用した C
- B – Pro*COBOL プリコンパイラを使用した COBOL
- C – Pro*C/C++ プリコンパイラを使用した C/C++
- V – OO4O (Oracle Objects for OLE) を使用した Visual Basic
- J – JDBC (Java Database Connectivity) を使用した Java
- S – SQL

表 11-1 ユースケース・モデル: 外部 LOB (BFILE)

ユースケースおよびページ	プログラム環境の例					
	P	O	B	C	V	J
11-14 ページの「BFILE を含む表を作成する 3 つの方法」						
11-15 ページの「1 つ以上の BFILE 列を含む表の作成」	S	S	S	S	S	S
11-18 ページの「BFILE 属性を持つオブジェクト型の表の作成」	S	S	S	S	S	S
11-21 ページの「BFILE を含む NESTED TABLE を持つ表の作成」	S	S	S	S	S	S
11-23 ページの「BFILE を含む行を挿入する 3 つの方法」						
11-24 ページの「BFILENAME() を使用した行の挿入」	S	+	+	+	+	+
11-32 ページの「別の表からの BFILE の選択による BFILE 行の挿入」	S	S	S	S	S	S
11-34 ページの「初期化した BFILE ロケータを使用した BFILE を含む行の挿入」	+	+	+	+	+	+
11-42 ページの「外部 LOB (BFILE) へのデータのロード」	S	S	S	S	S	S
11-46 ページの「LOB への BFILE データのロード」	+	+	+	+	+	+

ユースケースおよびページ	プログラム環境の例					
	P	O	B	C	V	J
11-55 ページの「BFILE をオープンする 2 つの方法」						
11-57 ページの「FILEOPEN を使用した BFILE のオープン」	+	+				+
11-62 ページの「OPEN を使用した BFILE のオープン」	+	+	+	+	+	+
11-69 ページの「BFILE のオープンを確認する 2 つの方法」						
11-71 ページの「FILEISOPEN を使用した BFILE のオープンの確認」	+	+				+
11-77 ページの「ISOPEN を使用した BFILE のオープンの確認」	+	+	+	+	+	+
11-86 ページの「BFILE データの表示」	+	+	+	+	+	+
11-97 ページの「BFILE からのデータの読み込み」	+	+	+	+	+	+
11-107 ページの「BFILE データの一部の読み込み (substr)」	+		+	+	+	+
11-114 ページの「2 つの BFILE の全体または一部の比較」	+		+	+	+	+
11-123 ページの「BFILE 内のパターンの有無の確認 (instr)」	+		+	+		+
11-131 ページの「BFILE が存在するかどうかの確認」	+	+	+	+	+	+
11-140 ページの「BFILE の長さの取得」	+	+	+	+	+	+
11-149 ページの「BFILE 用の LOB ロケータのコピー」	+	+	+	+		+
11-156 ページの「BFILE の LOB ロケータが初期化されているかどうかの確認」		+		+		
11-161 ページの「BFILE の LOB ロケータが他と等しいかどうかの確認」		+		+		+
11-168 ページの「ディレクトリ別名およびファイル名の取得」	+	+	+	+	+	+
11-177 ページの「BFILE を含む行を更新する 3 つの方法」						
11-178 ページの「BFILENAME() を使用した BFILE の更新」	S	S	S	S	S	S
11-181 ページの「別の表からの BFILE の選択による BFILE の更新」	S	S	S	S	S	S
11-183 ページの「初期化した BFILE ロケータを使用した BFILE の更新」	+	+	+	+	+	+
11-191 ページの「BFILE をクローズする 2 つの方法」						
11-193 ページの「FILECLOSE を使用した BFILE のクローズ」	+	+			+	+
11-198 ページの「CLOSE を使用した BFILE のクローズ」	+	+	+	+	+	+
11-205 ページの「オープン中のすべての BFILE のクローズ」	+	+	+	+	+	+
11-214 ページの「BFILE を含む表の行の削除」	S	S	S	S	S	S

図 11-1 ユースケース・モデル図：外部LOB (BFILE)



外部 LOB (BFILE) へのアクセス

外部 LOB (BFILE) にアクセスするには、次のインタフェースの1つを使用します。

- Pro*C/C++ や Pro*COBOL などのプリコンパイラ
- OCI (Oracle Call Interface)
- BMS_LOB パッケージを介した PL/SQL
- JDBC
- Oracle Objects for OLE (OO4O)

参照： 外部 LOB (BFILE) へのアクセスに使用する 6 つのインタフェースおよびその機能の詳細は、[第 3 章「LOB プログラム環境」](#)を参照してください。

ディレクトリ・オブジェクト

DIRECTORY オブジェクトによって、Oracle Server における BFILE のアクセスおよび使用の管理が容易になります (『Oracle8i SQL リファレンス』の第 7 章「SQL 文」の「CREATE DIRECTORY」を参照)。DIRECTORY は、アクセスするファイルが置かれているサーバーのファイル・システムの物理ディレクトリの論理的な名前を指定します。サーバーのファイル・システム内にあるファイルへは、DIRECTORY オブジェクトに必要なアクセス権限が付与されている場合に限り、アクセスできます。

BFILE ロケータの初期化

DIRECTORY オブジェクトによって、ファイルの位置を柔軟に扱えるようになり、アプリケーション内の物理ファイルの絶対パス名をハードコード化しなくても済みます。DIRECTORY 別名は、BFILE ロケータの初期化の際に、BFILENAME () 関数 (SQL および PL/SQL) または OCILobFileSetName () (OCI) で使用されます。

注意： Oracle は、指定したディレクトリおよびパス名が実際に存在するかどうかは検証しません。ご使用のオペレーティング・システムで有効なディレクトリを指定するよう注意してください。オペレーティング・システムがパス名の大 / 小文字を区別している場合は、必ず正しい形式でディレクトリを指定してください。最後のスラッシュを指定する必要はありません (たとえば、`/tmp/`ではなく、`/tmp`でも構いません)。

データベース・レコードへのオペレーティング・システム・ファイルの対応付け

オペレーティング・システム (OS) ・ファイル を BFILE に対応付けるために、まずオペレーティング・システム・ファイルへのフルパス名の別名である DIRECTORY オブジェクトを作成する必要があります。

既存のオペレーティング・システム・ファイルを特定の表の関連するデータベース・レコードに対応付けるには、Oracle の SQL DML (データ操作言語) を使用します。次に例を示します。

- BFILE の列がサーバーのファイル・システム内の既存ファイルを参照するように BFILE を初期化するには、INSERT を使用します。
- BFILE の参照ターゲットを変更するには、UPDATE を使用します。
- BFILENAME () 関数を使用して、BFILE を NULL に初期化してから、後でオペレーティング・システム・ファイルを参照するように更新します。
- OCI ユーザーの場合、OCILobFileSetName () を使用して、INSERT 文の VALUES 句で使用される BFILE ロケータ変数を初期化することもできます。

例

次の文では、ファイル Image1.gif および image2.gif をそれぞれ key_value が 21 および 22 のレコードに対応付けます。「IMG」は、Image1.dif および image2.dif が格納される物理ディレクトリを示す DIRECTORY オブジェクトです。

注意： 次のようなデータ構造を設定しないと機能しない場合もあります。

```
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'));
```

次の UPDATE 文は、key_value が 22 の行の場合にターゲット・ファイルを image3.gif に変更します。

```
UPDATE Lob_table SET f_lob = BFILENAME('IMG', 'image3.gif')  
WHERE Key_value = 22;
```

BFILENAME() および値の初期化

BFILENAME() は、BFILE 列が外部ファイルを参照するように BFILE を初期化するために使用する組み込み関数です。

SQL DML を使用して物理ファイルがレコードに一度対応付けられると、BFILE に対するその後の読み込み操作は、PL/SQL の DBMS_LOB パッケージおよび OCI を使用して実行できます。ただし、これらのファイルは、BFILE を介してアクセスされる場合は読み込み専用であるため、BFILE による更新または削除はできません。

BFILE は参照セマンティクスであるため、同一レコード内または異なるレコード内に、同じファイルを参照する複数の BFILE 列を持つことができます。たとえば、次の UPDATE 文は、lob_table 内で key_value が 21 の行を持つ BFILE が、key_value が 22 の行と同じファイルを参照するように設定します。

```
UPDATE lob_table
  SET f_lob = (SELECT f_lob FROM lob_table WHERE key_value = 22)
  WHERE key_value = 21;
```

初期化という点から BFILENAME() を考えてみてください。この関数は、次の値を初期化できます。

- BFILE 列
- PL/SQL モジュール内で宣言される BFILE (自動) 変数

メリット: これには次のようなメリットがあります。

- 特定の BFILE が一時的にしか必要でなく、操作中のモジュール内に限られている場合に、データベース内の列に対応付けなくても、BFILE 関連の API を変数に対して使用できます。
- サーバー側の表に BFILE 列を作成し、この列の値を初期化し、その後 SELECT で取り出す必要がないため、サーバーへのラウンドトリップを抑えることができます。

詳細は、DBMS_LOB.LOADFROMFILE の例を参照してください (11-46 ページの「[LOB への BFILE データのロード](#)」を参照)。

OCI で BFILENAME() に相当するものは OCILobFileSetName() であり、同様に使用できます。

ディレクトリ名の指定

DIRECTORY オブジェクトの命名規則は、表および索引の命名規則と同じです。つまり、通常の識別子は大文字で解釈されますが、デリミタ付き識別子そのまま解釈されます。たとえば、次のような文があるとします。

```
CREATE DIRECTORY scott_dir AS '/usr/home/scott';
```

この文によって、名前が「SCOTT_DIR」（大文字）のディレクトリ・オブジェクトが作成されます。ただし、次の文のように、DIRECTORY 名にデリミタ付き識別子が使用されるとします。

```
CREATE DIRECTORY "Mary_Dir" AS '/usr/home/mary';
```

この文では、DIRECTORY オブジェクトの名前は「Mary_Dir」になります。BFILENAME() をコールするときは、「SCOTT_DIR」および「Mary_Dir」を使用します。次に例を示します。

```
BFILENAME('SCOTT_DIR', 'afile')  
BFILENAME('Mary_Dir', 'afile')
```

Windows プラットフォーム上の場合

たとえば、Windows NT 上では、ディレクトリ名の大文字 / 小文字が区別されません。したがって、次の 2 つの文は同じディレクトリを表します。

```
CREATE DIRECTORY "big_cap_dir" AS "g:¥data¥source";
```

```
CREATE DIRECTORY "small_cap_dir" AS "G:¥DATA¥SOURCE";
```

BFILE セキュリティ

この項では、BFILE セキュリティ・モデルおよびそれに対応付けられた SQL 文について説明します。BFILE セキュリティに関係する主な SQL 文は、次のとおりです。

- SQL DDL: DIRECTORY オブジェクトの CREATE、REPLACE および ALTER
- SQL DML: DIRECTORY オブジェクトに対する READ システム、オブジェクト権限の GRANT および REVOKE

所有権および権限

DIRECTORY オブジェクトは、システム所有のオブジェクトです。システム所有のオブジェクトの詳細は、『Oracle8i SQL リファレンス』を参照してください。Oracle8i では、次の 2 つの新しいシステム権限をサポートしており、これらは DBA のみに付与されます。

- CREATE ANY DIRECTORY: ディレクトリ・オブジェクトの作成または変更用
- DROP ANY DIRECTORY: ディレクトリ・オブジェクトの削除用

ディレクトリ・オブジェクトに対する読取り権限

DIRECTORY オブジェクトに対する READ 権限によって、そのディレクトリ下に置かれたファイルを読み込むことができます。DIRECTORY オブジェクトの作成者には、READ 権限が自動的に付与されます。

GRANT オプション付きの READ 権限を付与されている場合は、その権限を他のユーザーやロールに付与して、自分の権限ドメインに追加することもできます。

注意： READ 権限は、個々のファイルではなく DIRECTORY オブジェクトのみに定義されます。したがって、同じディレクトリ内のファイルに異なる権限を割り当てることはできません。

DIRECTORY オブジェクトが表す物理ディレクトリには、Oracle Server プロセスに対して適切なオペレーティング・システム権限（この場合は読込み）がない場合もあります。

DBA には、次のことを確認する責任があります。

- 物理ディレクトリの存在
- Oracle Server プロセスに対して、ファイル、ディレクトリおよびディレクトリへのパスの読取り権限が使用可能になっていること
- データベース・ユーザーによってファイル・アクセスが行われている間、ディレクトリおよび読取り権限が使用可能な状態に保たれていること

ここでの権限とは、単に Oracle Server がそのディレクトリ内のファイルを読むことができるということです。これらの権限は、実際のファイル操作時に、PL/SQL の DBMS_LOB パッケージおよび OCI API によってチェックおよび施行されます。

警告： CREATE ANY DIRECTORY 権限および DROP ANY DIRECTORY 権限によって、サーバーのファイル・システムがすべてのデータベース・ユーザーに公開される可能性があるため、DBA は、セキュリティ違反を防止するため、一般のデータベース・ユーザーに対するこれらの権限の付与は慎重に行う必要があります。

BFILE セキュリティ用の SQL DDL

ディレクトリ・オブジェクトを作成、置換および削除する次の SQL DDL 文の詳細は、『Oracle8i SQL リファレンス』を参照してください。

- CREATE DIRECTORY
- DROP DIRECTORY

BFILE セキュリティ用の SQL DML

BFILE に対してセキュリティを提供する次の SQL DML 文の詳細は、『Oracle8i SQL リファレンス』を参照してください。

- GRANT (システム権限)
- GRANT (オブジェクト権限)
- REVOKE (システム権限)
- REVOKE (オブジェクト権限)
- AUDIT (新規文)
- AUDIT (スキーマ・オブジェクト)

ディレクトリのカタログ・ビュー

カタログ・ビューは、DIRECTORY オブジェクト用に提供され、これによってユーザーは、オブジェクトの名前、および対応するパスおよび権限を参照できます。サポートされるビューは次のとおりです。

- ALL_DIRECTORIES (OWNER, DIRECTORY_NAME, DIRECTORY_PATH)

このビューは、ユーザーがアクセスできるすべてのディレクトリが記述されます。

- DBA_DIRECTORIES (OWNER, DIRECTORY_NAME, DIRECTORY_PATH)

このビューによって、データベース全体について指定されたすべてのディレクトリが記述されます。

DIRECTORY 使用のガイドライン

DIRECTORY 機能の主な目的は、サーバーのファイル・システム内の大きなファイルへのアクセスを DBA が管理するうえで、単純で柔軟性があり、耐侵入性のある、安全性の高いメカニズムを使用可能にすることです。ただし、この目的を実現するには、DBA が DIRECTORY オブジェクトの使用時に次のガイドラインに従うことが非常に重要です。

- **DIRECTORY は、データ・ファイルのディレクトリなどにマップしないでください。**
DIRECTORY は、Oracle データ・ファイル、制御ファイル、ログ・ファイルおよびその他のシステム・ファイルを含む物理ディレクトリにマップしないでください。これらのファイルを（不意にまたはなんらかの理由で）変更すると、データベースまたはサーバーのオペレーティング・システムが破損する場合があります。
- **DBA のみがシステム権限を持つ必要があります。**
CREATE ANY DIRECTORY などのシステム権限（DBA にあらかじめ付与される）を他のユーザーに付与する場合は、慎重に行ってください。多くの場合、データベース管理者のみが、これらの権限を持ちます。
- **DIRECTORY オブジェクト権限を付与するときは、十分な注意が必要です。**
DIRECTORY オブジェクトへの権限を他のユーザーに付与する場合は、注意が必要です。同様に、権限をユーザーに付与するときは、WITH GRANT OPTION 句の使用にも注意が必要です。
- **データベース運用中に、DIRECTORY オブジェクトを削除または置換しないでください。**
データベース運用中に、DIRECTORY オブジェクトを安易に削除または置換しないでください。DIRECTORY オブジェクトを削除または置換した場合、このディレクトリ・オブジェクトに対応付けられているすべてのファイルでのすべてのセッションの操作が失敗します。さらに、これらのファイルを正常にクローズする前に、DROP コマンドまたは REPLACE コマンドを実行した場合は、プログラムにおけるこれらのファイルへの参照が失われ、これらのファイルに対応付けられているシステム・リソースも、セッションを停止するまで解放されません。

PL/SQL ユーザーに残される唯一の手段は、たとえば DBMS_LOB.FILECLOSEALL() をコールするプログラム・ブロックを実行しファイル操作を再開するか、またはセッションを完全に終了するかのいずれかです。したがって、これらのコマンドは慎重に使用し、できればメンテナンスのダウン時間中に使用します。
- **ユーザーの DIRECTORY オブジェクト権限を取り消すときには、十分な注意が必要です。**
REVOKE 文を使用してユーザーの DIRECTORY オブジェクト権限を取り消した場合、ユーザーのセッションに依存するファイルに対する後続の操作は失敗します。ユーザーは、権限を再取得してファイルをクローズするか、またはセッションで FILECLOSEALL() を実行し、ファイル操作を再開する必要があります。

一般的に、ファイル・アクセスの管理に DIRECTORY オブジェクトを使用することは、オペレーティング・システム・レベルでのシステム管理作業の延長です。なんらかの計画を立てることによって、Oracle プロセス用の READ 権限を持つ適切なディレクトリへファイルを論理的に構成できます。

これらの物理ファイルにマップする READ 権限とともに DIRECTORY オブジェクトを作成し、特定のデータベース・ユーザーにこれらのディレクトリへのアクセス権限を付与できます。

マルチスレッド・サーバー (MTS) ・モードの BFILE

Oracle8i では、マルチスレッド・サーバー (MTS) ・モードでの BFILE のセッション移行がサポートされません。これは、オープンされた BFILE に対する操作を、MTS サーバーへのコール終了後も継続できることを意味します。

BFILE 操作が含まれるセッションは、1 台の共有サーバーに限定され、サーバー間での移行はできません。この制約は、次のリリースで削除されます。

外部 LOB (BFILE) ロケータ

BFILE では、値はサーバー側のオペレーティング・システム・ファイル (データベースの外部) に格納されます。そのファイルを参照する BFILE ロケータは、行内に格納されます。

BFILE 表内の 2 つの行が同じファイルを参照するとき DBMS_LOB.FILEOPEN() で使用される BFILE ロケータ変数 (たとえば L1) が、別のロケータ変数 (たとえば L2) に割り当てられた場合、L1 および L2 はどちらも同じファイルを参照します。BFILE 列を持つ表の 2 つの行は、同じファイルを参照することも、2 つの異なるファイルを参照することもできます。このことは、開発者が慎重であればメリットになりますが、そうでなければデメリットになる場合があります。

BFILE ロケータ変数 BFILE ロケータ変数は、他の自動変数と同じように機能します。ファイル操作に関しては、最も標準的なプログラミング言語の標準 I/O の一部として利用できるファイル記述子と同じように機能します。BFILE ロケータを定義および初期化し、このロケータによって参照されるファイルをオープンした場合、このファイルをクローズするまでのすべての操作は、このロケータまたはこのロケータのローカル・コピーを使用して、同じプログラム・ブロック内から実行する必要があることを意味します。

ガイドライン

- **ファイルは、同じネスト・レベルにある、同じプログラム・ブロックからオープンおよびクローズします。** BFILE ロケータ変数は、スカラーと同様、他のプロシージャ、メンバ・メソッドまたは外部関数コールアウトのパラメータとして使用できます。ただし、ファイルのオープンおよびクローズは、同じネスト・レベルにある、同じプログラム・ブロックから行うことをお勧めします。

- **オブジェクトをデータベースにフラッシュする前に、BFILE 値を設定します。** オブジェクトに BFILE が含まれる場合、オブジェクトをデータベースにフラッシュする前に BFILE 値を設定する必要があります。それによって、新しい行を挿入できます。言い換えると、OCIObjectNew() の後でかつ OCIObjectFlush() の前に、OCILOBFileNameSet() をコールする必要があります。
- **BFILE を挿入または更新する前に、ディレクトリ別名およびファイル名を指定します。** ディレクトリ別名およびファイル名を指定せずに BFILE を INSERT または UPDATE を行うとエラーになります。

この規則は、INSERT/UPDATE 文で BFILE に対して OCI バインド変数を使用するユーザーにも適用されます。INSERT 文または UPDATE 文を発行する前に、OCI バインド変数をディレクトリ別名およびファイル名で初期化する必要があります。
- 挿入または更新する前に BFILE を初期化します

注意: OCISetAttr() では、BFILE ロケータを NULL に設定できません。

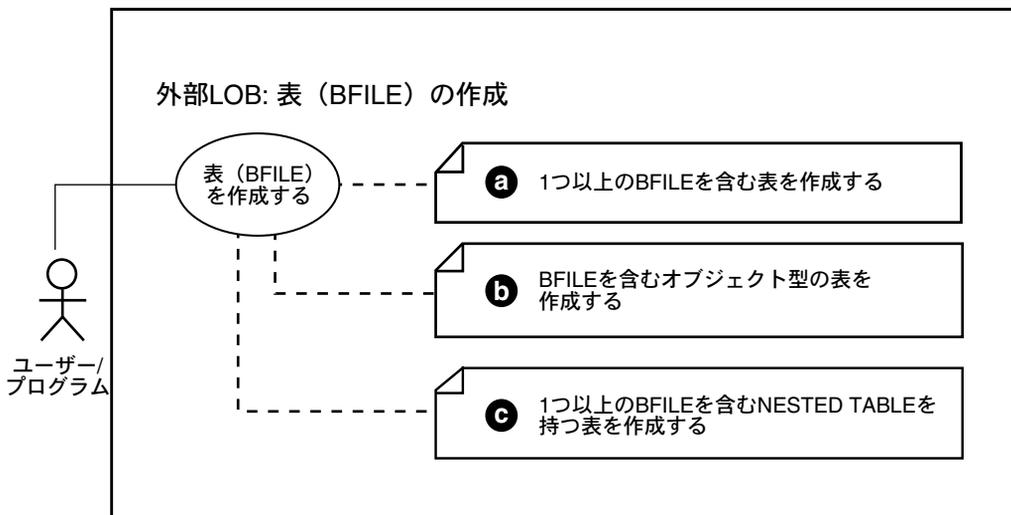
一般的な規則

BFILE を持つ行を挿入または更新するための SQL を使用する前に、ユーザーは BFILE を次のいずれかに初期化する必要があります。

- NULL (OCI バインド変数を使用している場合は不可)
- ディレクトリ別名およびファイル名

BFILE を含む表を作成する 3 つの方法

図 11-2 ユースケース図：1 つ以上の BFILE 列を含む表を作成する 3 つの方法



参照： 外部 LOB (BFILE) に関するすべての基本操作については、11-2 ページの「ユースケース・モデル: 外部 LOB (BFILE)」を参照してください。

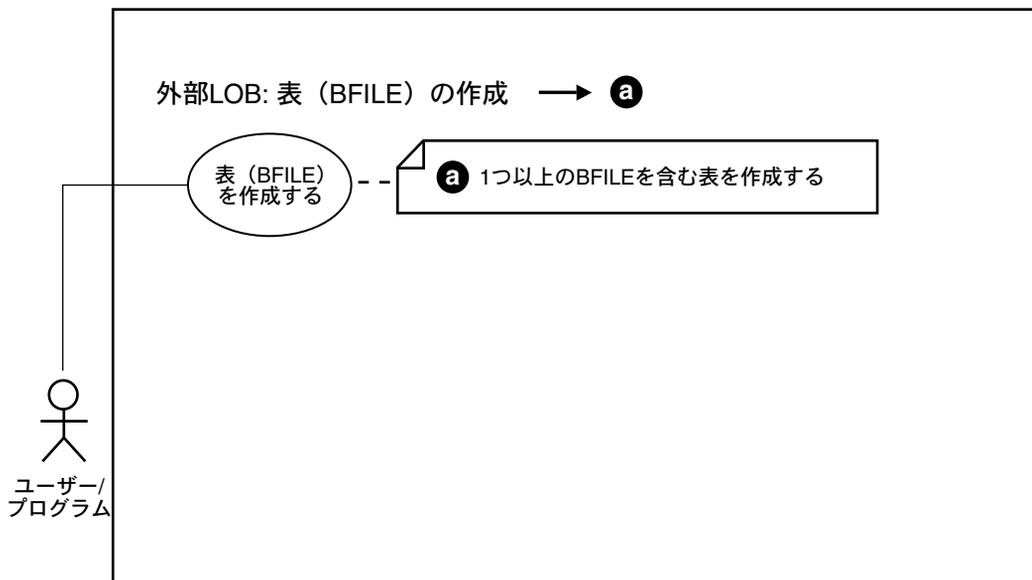
BFILE を表に取り込むには、次の 3 つの方法があります。

- 表の列として取り込む方法: 11-15 ページの「[BFILE 属性を持つオブジェクト型の表の作成](#)」を参照してください。
- オブジェクト型の属性として取り込む方法: 11-18 ページの「[BFILE 属性を持つオブジェクト型の表の作成](#)」を参照してください。
- NESTED TABLE 内に含める方法: 11-21 ページの「[BFILE を含む NESTED TABLE を持つ表の作成](#)」を参照してください。

いずれの場合も SQL データ定義言語 (DDL) を使用して、表内の BFILE 列およびオブジェクト型の BFILE 属性を定義します。

1つ以上の BFILE 列を含む表の作成

図 11-3 ユースケース図: 1つ以上の BFILE 列を含む表の作成



参照: 外部 LOB (BFILE) に関するすべての基本操作については、11-2 ページの「ユースケース・モデル: 外部 LOB (BFILE)」を参照してください。

用途

1つ以上の BFILE 列を含む表を作成します。

使用上の注意

ありません。

構文

次のマニュアルの項を参照してください。

- SQL: 『Oracle8i SQL リファレンス』の第7章「SQL文」の「CREATE TABLE」

使用例

仮想アプリケーションの中心は、Multimedia_tab 表です。この表の列を構成する様々な型によって、クリップの作成に使用される何種類ものマルチメディア要素を1つにまとめることができます。

例

例が SQL で示されています。この例は、すべてのプログラム環境に適用されます。

- SQL: 1つ以上の BFILE 列を含む表の作成

SQL: 1つ以上の BFILE 列を含む表の作成

次のようなデータ構造を設定しないと機能しない例もあります。

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

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

CREATE TABLE VoiceoverLib_tab of Voiced_typ
( Script DEFAULT EMPTY_CLOB(),
  CONSTRAINT TakeLib CHECK (Take IS NOT NULL),
  Recording DEFAULT NULL
);
CONNECT samp/samp
CREATE TABLE a_table (blob_col BLOB);
CREATE TYPE Voiced_typ AS OBJECT
( Originator      VARCHAR2(30),
```

```

Script          CLOB,
Actor           VARCHAR2(30),
Take            NUMBER,
Recording        BFILE );

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

CREATE TYPE InSeg_tab AS TABLE of InSeg_typ;

CREATE TYPE Map_typ AS OBJECT
( Region        VARCHAR2(30),
  NW             NUMBER,
  NE             NUMBER,
  SW             NUMBER,
  SE             NUMBER,
  Drawing        BLOB,
  Aerial         BFILE);

CREATE TABLE Map_Libtab of Map_typ;
CREATE TABLE Voiceover_tab of Voiced_typ
( Script DEFAULT EMPTY_CLOB(),
  CONSTRAINT Take CHECK (Take IS NOT NULL),
  Recording DEFAULT NULL);

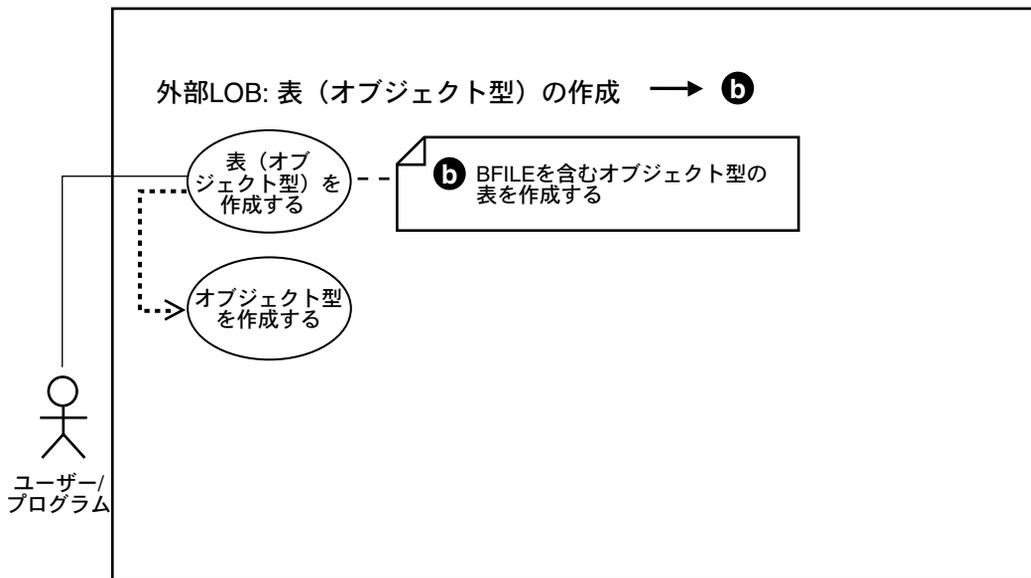
SQL DDL を直接使用して、LOB 列が 1 つ以上含まれる表を作成できるため、DBMS_LOB
パッケージを使用する必要はありません。

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

```

BFILE 属性を持つオブジェクト型の表の作成

図 11-4 ユースケース図：BFILE を含む表の作成



参照： 外部 LOB (BFILE) に関するすべての基本操作については、11-2 ページの「[ユースケース・モデル: 外部 LOB \(BFILE\)](#)」を参照してください。

用途

BFILE 属性を持つオブジェクト型の表を作成します。

使用上の注意

図に示されているように、BFILE 属性を含むオブジェクト型を作成してから、そのオブジェクト型を使用する表を作成する必要があります。

構文

次のマニュアルの項を参照してください。

- SQL: 『Oracle8i SQL リファレンス』の第7章「SQL文」の「CREATE TABLE」、「CREATE TYPE」

NCLOB は、オブジェクト型の属性になることができないことに注意してください。

使用例

次の例には、オブジェクト型に BFILE を含むことのできる 2 つの異なる方法が含まれています。

- Multimedia_tab には、Voiced_typ 型に基づく VoiceOver_tab 表の中の行オブジェクトを参照する Voiced_ref 列が含まれます。この型には、CLOB および BFILE の 2 種類の LOB が含まれます。CLOB は俳優が読む台本を格納し、BFILE は音声録音を格納します。
- Multimedia_tab 表は、Map_typ 型の列オブジェクトを含む Map_obj 列を含みます。Map_typ 型は、地域の航空写真を格納するために BFILE データ型を使用します。

例

例が SQL で示されています。この例は、すべてのプログラム環境に適用されます。

- [SQL: BFILE 属性を持つオブジェクト型の表の作成](#)

SQL: BFILE 属性を持つオブジェクト型の表の作成

/* SQL DDL を使用して、型 Voiced_typ をナレーションの記録を含むことができる表の基礎として作成します。*/

```
CREATE TYPE Voiced_typ AS OBJECT
( Originator    VARCHAR2(30),
  Script       CLOB,
  Actor        VARCHAR2(30),
  Take         NUMBER,
  Recording     BFILE
);
```

/* SQL DDL を使用して、Voiceover_tab 表を作成します。*/

```
CREATE TABLE Voiceover_tab OF Voiced_typ
( Script DEFAULT EMPTY_CLOB(),
  CONSTRAINT Take CHECK (Take IS NOT NULL),
  Recording DEFAULT NULL
);
```

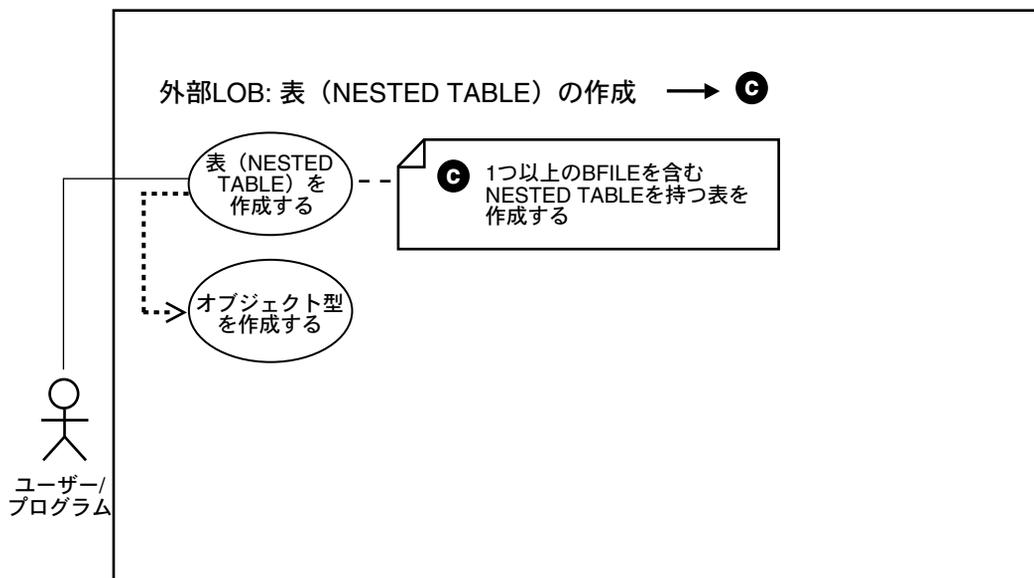
BFILE 属性を持つオブジェクト型の表の作成

```
/* SQL DDLを使用して、型 Map_typ を列オブジェクトを含む表の基礎として作成します。*/
CREATE TYPE Map_typ AS OBJECT
( Region          VARCHAR2(30),
  NW              NUMBER,
  NE              NUMBER,
  SW              NUMBER,
  SE              NUMBER,
  Drawing         BLOB,
  Aerial          BFILE
);

/* SQL DDLを使用して、サポート表 MapLib_tab をマップのアーカイブとして作成します。*/
CREATE TABLE Map_tab of MapLib_typ;
```

BFILE を含む NESTED TABLE を持つ表の作成

図 11-5 ユースケース図：BFILE を含む NESTED TABLE を持つ表の作成



参照： 外部 LOB (BFILE) に関するすべての基本操作については、11-2 ページの「ユースケース・モデル: 外部 LOB (BFILE)」を参照してください。

用途

BFILE を含む NESTED TABLE を持つ表を作成します。

使用上の注意

図で示されるように、BFILE 属性を含むオブジェクト型を作成してから、そのオブジェクト型を使用する NESTED TABLE を作成する必要があります。

構文

次のマニュアルの項を参照してください。

- SQL: 『Oracle8i SQL リファレンス』の第7章「SQL文」の「CREATE TABLE」、「CREATE TYPE」

使用例

次の例の `Multimedia_tab` は、`InSeg_typ` 型を含む NESTED TABLE、`Inseg_ntab` を含んでいます。この型は2種類の LOB データベースを使用しています。BFILE はインタビューの音声を録音し、CLOB は録音の台本用のものです。

BFILE 列を持つ表を作成する方法については、11-15 ページの「[1つ以上の BFILE 列を含む表の作成](#)」を参照してください。ここでは、基礎となるオブジェクト型を作成するための SQL 構文のみを説明します。

例

例が SQL で示されています。この例は、すべてのプログラム環境に適用されます。

- [SQL: BFILE を含む NESTED TABLE を持つ表の作成](#)

SQL: BFILE を含む NESTED TABLE を持つ表の作成

SQL DDL を直接使用して表を作成するため、`DBMS_LOB` パッケージを使用する必要はありません。

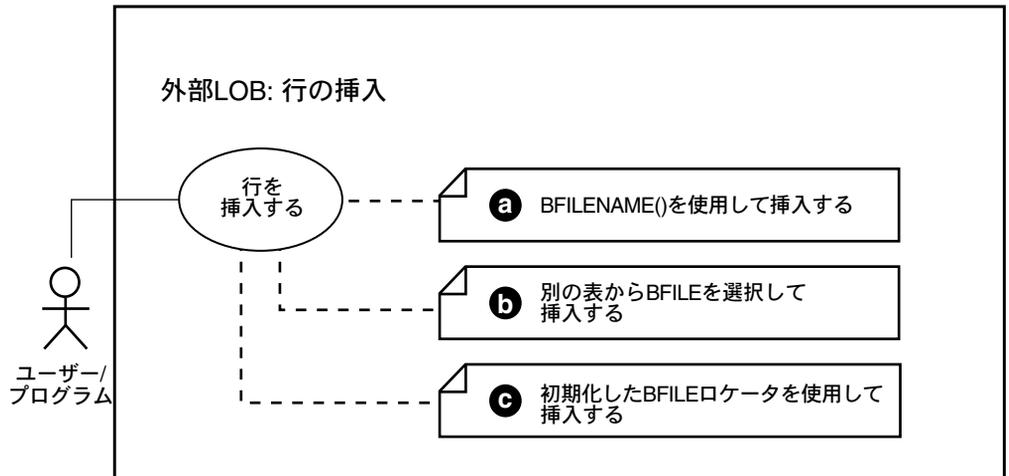
```
CREATE TYPE InSeg_typ AS OBJECT
(
  Segment          NUMBER,
  Interview_Date   DATE,
  Interviewer      VARCHAR2(30),
  Interviewee      VARCHAR2(30),
  Recording        BFILE,
  Transcript       CLOB
);
```

NESTED TABLE の埋込みは、その表を含む構造が定義されたときに行われます。この例では、`Multimedia_tab` が作成される時点で、次の文によって行われます。

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

BFILE を含む行を挿入する 3 つの方法

図 11-6 ユースケース図：BFILE を含む列を挿入する 3 つの方法



参照： 外部 LOB (BFILE) に関するすべての基本操作については、11-2 ページの「ユースケース・モデル: 外部 LOB (BFILE)」を参照してください。

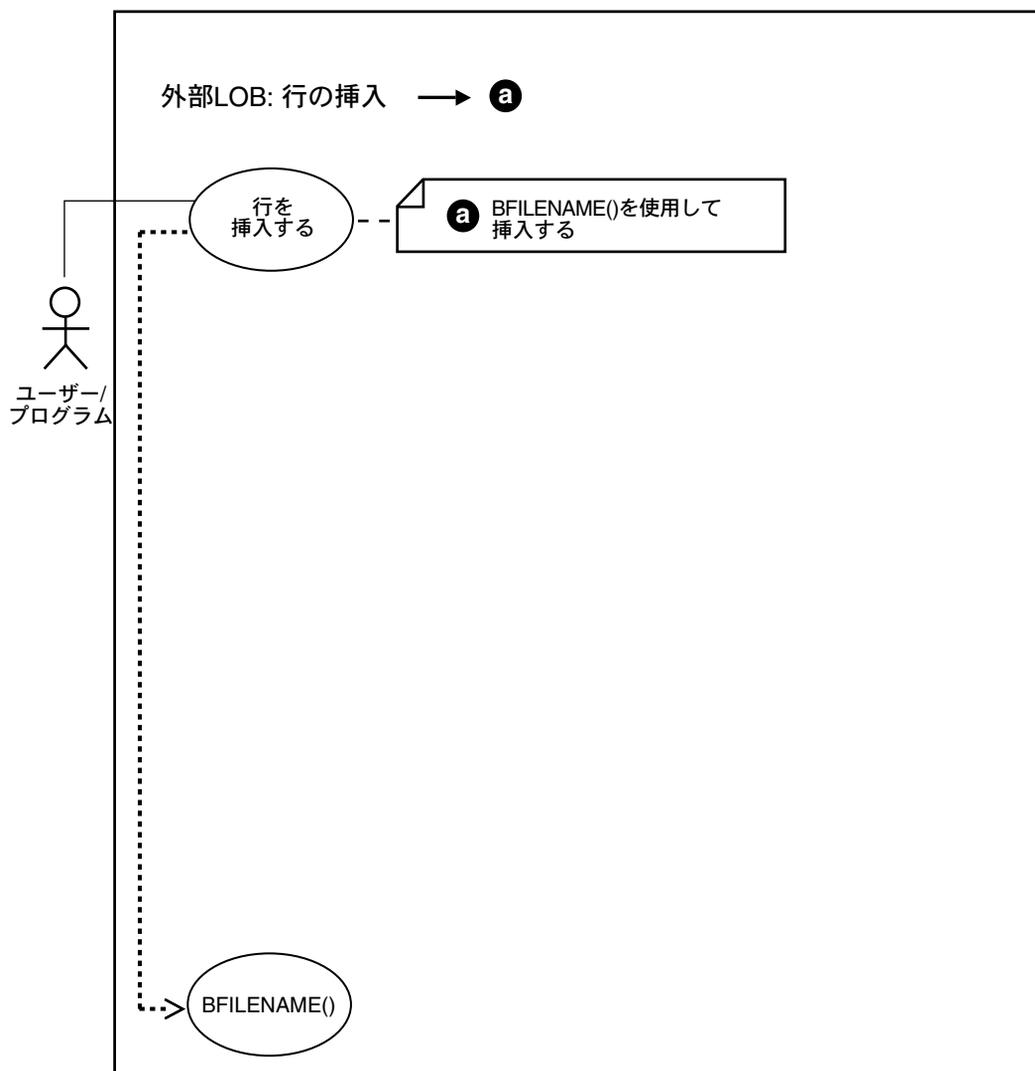
注意： BFILE は、挿入する前に、NULL、またはディレクトリ別名およびファイル名に初期化する必要があります。

BFILE を含む行を挿入するには、次の 3 つの方法があります。

- BFILENAME() を使用した行の挿入 (11-24 ページ)
- 別の表からの BFILE の選択による BFILE 行の挿入 (11-34 ページ)
- 初期化した BFILE ロケータを使用した BFILE を含む行の挿入 (11-34 ページ)

BFILENAME() を使用した行の挿入

図 11-7 ユースケース図 : BFILENAME() を使用した行の挿入



参照： 外部 LOB (BFILE) に関するすべての基本操作については、11-2 ページの「[ユースケース・モデル:外部 LOB \(BFILE\)](#)」を参照してください。

用途

BFILENAME() を使用して行を挿入します。

使用上の注意

BFILENAME() 関数は、特定の行の BFILE 列または BFILE 属性をサーバーのファイル・システム内の物理ファイルに対応付けることによって初期化する INSERT の一部としてコールします。

BFILENAME() への `directory_alias` パラメータで表される DIRECTORY オブジェクトは、BFILENAME() が SQL または PL/SQL プログラムでコールされる前に定義されている必要はありませんが、DIRECTORY オブジェクトおよびオペレーティング・システム・ファイルは、実際に BFILE ロケータを使用する時点までに存在している必要があります。たとえば、次の操作のいずれかへのパラメータとして使用されるときです。

- OCILobFileOpen()
- DBMS_LOB.FILEOPEN()
- OCILobOpen()
- DBMS_LOB.OPEN()

注意： BFILENAME() では、この DIRECTORY オブジェクトに対する権限の妥当性チェックは行われず、DIRECTORY オブジェクトが表す物理ディレクトリが実際に存在するかどうかはチェックされません。このようなチェックは、BFILENAME() によって初期化された BFILE ロケータを使用するファイル・アクセス時に限り実行されます。

BFILE 列またはロケータ変数を初期化するための BFILENAME() の使用方法

次の方法で BFILENAME() を使用して、BFILE 列を初期化できます。

- SQL の INSERT 文の一部として
- UPDATE 文の一部として

BFILENAME() を使用してプログラム・インタフェース・プログラムの BFILE ロケータ変数を初期化し、そのロケータをファイル操作に使用することができます。ただし、対応するディレクトリ別名またはファイル名（あるいはその両方）が存在しない場合、この変数を使用する PL/SQL の DBMS_LOB またはその他の関連ルーチンでエラーが発生します。

BFILENAME() 関数の `directory_alias` パラメータは、ディレクトリ名の大 / 小文字を正確に指定する必要があります。

参照: 11-8 ページの「[ディレクトリ名の指定](#)」を参照してください。

構文

各プログラム環境で使用可能な機能のリストは、[第 3 章「LOB プログラム環境](#)」を参照してください。各プログラム環境における構文については、次のマニュアルの項を参照してください。

- SQL: 『Oracle8i SQL リファレンス』の第 7 章「SQL 文」の「INSERT」
- C (OCI) : 『Oracle8i コール・インタフェース・プログラマーズ・ガイド』の第 7 章「LOB および FILE 操作」の使用上の注意、および第 15 章「OCI リレーショナル関数」の「LOB 関数」
- COBOL (Pro*COBOL) : 『Oracle8i Pro*COBOL プリコンパイラ・プログラマーズ・ガイド』の第 13 章「ラージ・オブジェクト (LOB)」の「LOB 文」の使用上の注意、付録 F「埋込み SQL 文およびプリコンパイラ宣言文」、および『Oracle8i SQL リファレンス』の第 7 章「SQL 文」の「INSERT」
- C/C++ (Pro*C/C++) : 『Oracle8i Pro*C/C++ プリコンパイラ・プログラマーズ・ガイド』の第 16 章「ラージ・オブジェクト (LOB)」の「LOB 文」の使用上の注意、付録 F「埋込み SQL 文およびプリコンパイラ宣言文」、および『Oracle8i SQL リファレンス』の第 7 章「SQL 文」の「INSERT」
- Visual Basic (Oracle Objects for OLE オンライン・ヘルプ) : 「Help」の「Contents」タブから、「OO4O Automation Server」>「OBJECTS」>「OraBfile」>「METHODS」>「AddNew」、および「OBJECTS」>「OraBfile」>「METHODS」>「various methods」を選択してください。
- Java (JDBC) : 『Oracle8i JDBC 開発者ガイドおよびリファレンス』の第 7 章「LOB と BFILE の操作」の「BLOB と CLOB の操作」の「BLOB または CLOB 列の作成と移入」、および『Oracle8i SQLJ 開発者ガイドおよびリファレンス』の第 5 章「型のサポート」の「Oracle の拡張型」の「BLOB、CLOB および BFILE のサポート」

使用例

次のプログラム環境の例が示されています。

- SQL: BFILENAME() を使用した行の挿入 (11-27 ページ)

- C (OCI) : BFILENAME() を使用した行の挿入 (11-27 ページ)
- COBOL (Pro*COBOL) : BFILENAME() を使用した行の挿入 (11-28 ページ)
- C/C++ (Pro*C/C++) : BFILENAME() を使用した行の挿入 (11-29 ページ)
- Visual Basic (OO4O) : BFILENAME() を使用した行の挿入 (11-30 ページ)
- Java (JDBC) : BFILENAME() を使用した行の挿入 (11-30 ページ)

例

次の例は、BFILENAME() を使用して行を挿入する方法を示しています。

SQL: BFILENAME() を使用した行の挿入

/* これは、内部永続 LOB に適用した INSERT 文と同じですが、BFILE 列を初期化するために BFILENAME() 関数を追加していることに注意してください。*/

```
INSERT INTO Multimedia_tab VALUES (1, EMPTY_CLOB(), EMPTY_CLOB(),
FILENAME('PHOTO_DIR', 'LINCOLN_PHOTO'),
EMPTY_BLOB(), EMPTY_BLOB(),
VOICED_TYP('Abraham Lincoln', EMPTY_CLOB(), 'James Earl Jones', 1, NULL),
NULL, BFILENAME('AUDIO_DIR', 'LINCOLN_AUDIO'),
MAP_TYP('Gettysburg', 23, 34, 45, 56, EMPTY_BLOB(), NULL));
```

C (OCI) : BFILENAME() を使用した行の挿入

```
/* BFILENAME を使用して行を挿入します。*/
void insertUsingBfilename(svchp, stmthp, errhp)
OCISvcCtx *svchp;
OCIStmt *stmthp;
OCIError *errhp;
{
    text *insstmt =
        (text *) "INSERT INTO Multimedia_tab VALUES (3, EMPTY_CLOB(), \
EMPTY_CLOB(), BFILENAME('PHOTO_DIR', 'Lincoln_photo'), \
EMPTY_BLOB(), EMPTY_BLOB(), NULL, \
NULL, BFILENAME('AUDIO_DIR', 'Lincoln_audio'), \
```

```
        MAP_TYP('Gettysburg', 23, 34, 45, 56, EMPTY_BLOB(), NULL));

/* SQL 文を準備します。*/
checkerr (errhp, OCIStmtPrepare(stmthp, errhp, insstmt, (ub4)
                                strlen((char *) insstmt),
                                (ub4) OCI_NTV_SYNTAX, (ub4)OCI_DEFAULT));

/* SQL 文を実行します。*/
checkerr (errhp, OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                                (CONST OCISnapshot*) 0, (OCISnapshot*) 0,
                                (ub4) OCI_DEFAULT));

}
```

COBOL (Pro*COBOL) : BFILENAME() を使用した行の挿入

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

01  USERID          PIC X(11) VALUES "SAMP/SAMP".
01  ORASLNRD        PIC 9(4).

EXEC SQL INCLUDE SQLCA END-EXEC.
EXEC ORACLE OPTION (ORACA=YES) END-EXEC.
EXEC SQL INCLUDE ORACA END-EXEC.

PROCEDURE DIVISION.
BFILE-INSERT.

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

EXEC SQL
INSERT INTO MULTIMEDIA_TAB (CLIP_ID, PHOTO)
VALUES (1, BFILENAME('PHOTO_DIR', 'LINCOLN_PHOTO'))
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++) : BFILENAME() を使用した行の挿入

```

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

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

void BFILENAMEInsert_proc()
{
    EXEC SQL WHENEVER SQLERROR DO Sample_Error();
    EXEC SQL WHENEVER NOT FOUND CONTINUE;
    /* すべての既存の行を削除します。*/
    EXEC SQL DELETE FROM Multimedia_tab WHERE Clip_ID = 1;
    /* BFILENAME() を使用して、BFILE に対して新しい行を挿入します。*/
    EXEC SQL INSERT INTO Multimedia_tab
        VALUES (1, EMPTY_CLOB(), EMPTY_CLOB(),
            BFILENAME('PHOTO_DIR', 'Lincoln_photo'),
            EMPTY_BLOB(), EMPTY_BLOB(), NULL,
            InSeg_tab(InSeg_typ(1, NULL, 'Ted Koppell', 'Abraham Lincoln',
                BFILENAME('AUDIO_DIR', 'Lincoln_audio'),
                EMPTY_CLOB()))),
            BFILENAME('AUDIO_DIR', 'Lincoln_audio'),
            Map_typ('Moon Mountain', 23, 34, 45, 56, EMPTY_BLOB()),

```

```
        BFILENAME('PHOTO_DIR', 'Lincoln_photo')));
    printf("Inserted %d row\n", sqlca.sqlerrd[2]);
}

void main()
{
    char *samp = "samp/samp";
    EXEC SQL CONNECT :samp;
    BFILENAMEInsert_proc();
    EXEC SQL ROLLBACK WORK RELEASE;
}
```

Visual Basic (OO4O) : BFILENAME() を使用した行の挿入

```
Dim OraDyn as OraDynaset, OraPhoto as OraBFile, OraMusic as OraBFile

Set OraDyn = OraDb.CreateDynaset("select * from Multimedia_tab", ORADYN_DEFAULT)
Set OraMusic = OraDyn.Fields("Music").Value
Set OraPhoto = OraDyn.Fields("Photo").Value
OraDyn.AddNew
OraDyn.Fields("Clip_ID").value = 1
OraDyn.Fields("Story").value = Empty 'This is equivalent to EMPTY_BLOB() in SQL
OraDyn.Fields("FLSub").value = Empty
'BFILe データを初期化します。
OraPhoto.DirectoryName = "PHOTO_DIR"
OraPhoto.FileName = "LINCOLN_PHOTO"
OraDyn.Fields("Frame").Value = Empty
OraDyn.Fields("Sound").Value = Empty
'BFILe データを初期化します。
OraMusic.DirectoryName = "AUDIO_DIR"
OraMusic.FileName = "LINCOLN_AUDIO"
OraDyn.Update
' 表に行を追加します。
```

Java (JDBC) : BFILENAME() を使用した行の挿入

```
import java.io.InputStream;
import java.io.OutputStream;
// コア JDBC クラス :
import java.sql.DriverManager;
import java.sql.Connection;
import java.sql.Statement;
import java.sql.PreparedStatement;
```

```
import java.sql.ResultSet;
import java.sql.SQLException;

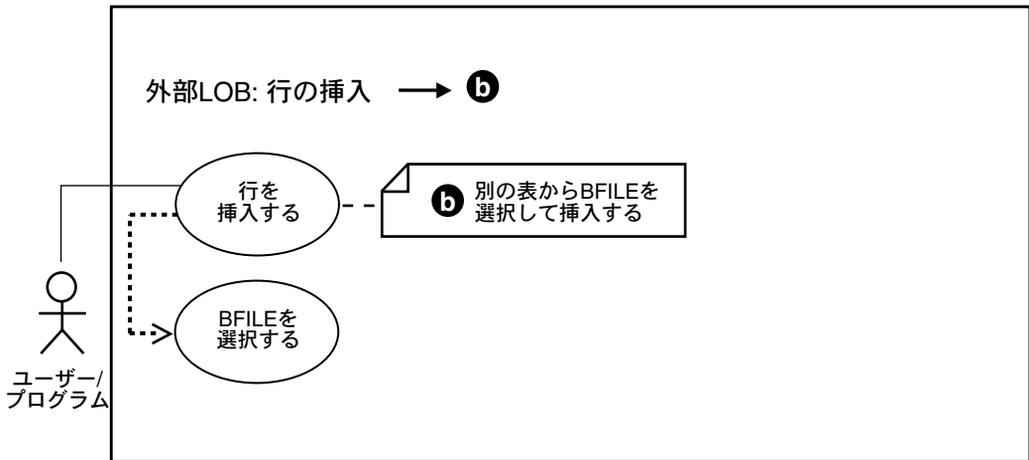
// Oracle 固有の JDBC クラス :
import oracle.sql.*;
import oracle.jdbc.driver.*;
public class Ex4_21
{
    public static void main (String args [])
        throws Exception
    {
        // Oracle JDBC Driver をロードします。
        DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

        // データベースに接続します。
        Connection conn =
            DriverManager.getConnection ("jdbc:oracle:oci8:@", "samp", "samp");
        conn.setAutoCommit (false);

        // 文を作成します。
        Statement stmt = conn.createStatement ();
        try
        {
            stmt.execute("INSERT INTO multimedia_tab "
                +"VALUES (99, EMPTY_CLOB(), EMPTY_CLOB(), "
                +"BFILENAME ('PHOTO_DIR', 'Lincoln_photo'), "
                +"EMPTY_BLOB(), EMPTY_BLOB(), "
                +"(SELECT REF(Vref) FROM Voiceover_tab Vref "
                +" WHERE Actor = 'James Earl Jones'), NULL, "
                +"BFILENAME('AUDIO_DIR', 'Lincoln_audio'), "
                +"MAP_TYP('Gettysburg', 23, 34, 45, 56, EMPTY_BLOB(), NULL))");
            // トランザクションをコミットします。
            conn.commit();
            stmt.close();
            conn.close();
        }
        catch (SQLException e)
        {
            e.printStackTrace();
        }
    }
}
```

別の表からの BFILE の選択による BFILE 行の挿入

図 11-8 ユースケース図：別の表からの BFILE の選択による BFILE を含む行の挿入 (INSERT ... AS ... SELECT)



参照： 外部 LOB (BFILE) に関するすべての基本操作については、11-2 ページの「ユースケース・モデル:外部 LOB (BFILE)」を参照してください。

用途

別の表から BFILE を選択することによって、BFILE を含む行を挿入します。

使用上の注意

LOB に関してオブジェクト・リレーショナル技法を使用すると、型を関連する表の共通テンプレートとして定義できるというメリットがあります。たとえば、アーカイブ・データを格納する表と、これらのライブラリを使用する作業表の両方が共通の構造を持つことには意味があります。後述の「使用例」を参照してください。

構文

次のマニュアルの項を参照してください。

- SQL: 『Oracle8i SQL リファレンス』の第7章「SQL文」の「INSERT」

使用例

次の例は、ライブラリ表 `VoiceoverLib_tab` が、`Multimedia_tab` 表の `Voiced_ref` 列によって参照される `Voiceover_tab` と同じ型 (`Voiced_typ`) であるということに基づいています。

ここでは、BFILE の選択によって、ライブラリ表から `Multimedia_tab` に値が挿入されます。

例

例が SQL で示されています。この例は、すべてのプログラム環境に適用されます。

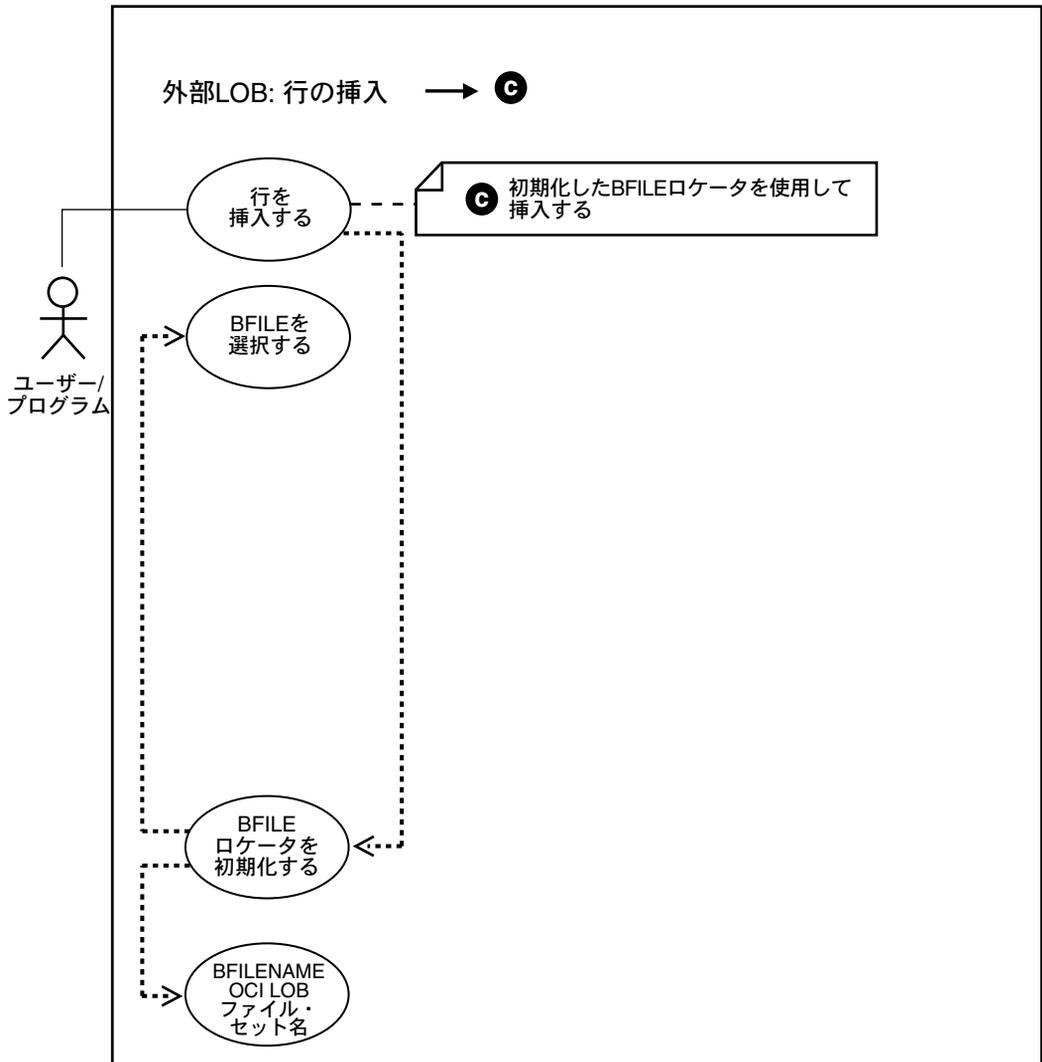
- [SQL: 別の表からの BFILE の選択による BFILE を含む行の挿入](#)

SQL: 別の表からの BFILE の選択による BFILE を含む行の挿入

```
INSERT INTO Voiceover_tab
  (SELECT * from VoiceoverLib_tab
   WHERE Take = 12345);
```

初期化した BFILE ロケータを使用した BFILE を含む行の挿入

図 11-9 ユースケース図：初期化した BFILE ロケータを使用した行の挿入



参照： 外部 LOB (BFILE) に関するすべての基本操作については、11-2 ページの「ユースケース・モデル:外部 LOB (BFILE)」を参照してください。

用途

初期化した BFILE ロケータを使用して、BFILE を含む行を挿入します。

使用上の注意

注意： INSERT 文を発行する前に、BFILE ロケータのバインド変数をディレクトリ別名およびファイル名に初期化する必要があります。

構文

各プログラム環境で使用可能な機能のリストは、第 3 章「LOB プログラム環境」を参照してください。各プログラム環境における構文については、次のマニュアルの項を参照してください。

- SQL: 『Oracle8i SQL リファレンス』の第 7 章「SQL 文」の「INSERT」
- C (OCI) : 『Oracle8i コール・インタフェース・プログラマーズ・ガイド』の第 7 章「LOB および FILE 操作」および第 15 章「OCI リレーショナル関数」の「LOB 関数」
- COBOL (Pro*COBOL) : 『Oracle8i Pro*COBOL プリコンパイラ・プログラマーズ・ガイド』の第 13 章「ラージ・オブジェクト (LOB)」の「LOB 文」の使用上の注意、付録 F「埋込み SQL 文およびプリコンパイラ宣言文」、および『Oracle8i SQL リファレンス』の第 7 章「SQL 文」の「INSERT」
- C/C++ (Pro*C/C++) : 『Oracle8i Pro*C/C++ プリコンパイラ・プログラマーズ・ガイド』の第 16 章「ラージ・オブジェクト (LOB)」の「LOB 文」の使用上の注意、付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB FILE SET (実行可能埋込み SQL 拡張要素)」、および『Oracle8i SQL リファレンス』の第 7 章「SQL 文」の「INSERT」
- Visual Basic (Oracle Objects for OLE オンライン・ヘルプ) : 「Help」の「Contents」タブから、「OO4O Automation Server」>「OBJECTS」>「OraBfile」>「METHODS」>「DirectoryName」、「DirectoryName」、および「OBJECTS」>「OraDynaset」>「METHODS」>「Update」を選択してください。
- Java (JDBC) : 『Oracle8i JDBC 開発者ガイドおよびリファレンス』の第 7 章「LOB と BFILE の操作」の「BLOB と CLOB の操作」の「BLOB または CLOB 列の作成と移入」、および『Oracle8i SQL 開発者ガイドおよびリファレンス』の第 5 章「型のサポート」の「Oracle の拡張型」の「BLOB、CLOB および BFILE のサポート」

使用例

この例では、オペレーティング・システムのソース・ファイル (PHOTO_DIR) から Photo を挿入します。

例

次のプログラム環境の例が示されています。

- [PL/SQL: 初期化した BFILE ロケータを使用した BFILE を含む行の挿入 \(11-36 ページ\)](#)
- [C \(OCI\) : 初期化した BFILE ロケータを使用した BFILE を含む行の挿入 \(11-36 ページ\)](#)
- [COBOL \(Pro*COBOL\) : 初期化した BFILE ロケータを使用した BFILE を含む行の挿入 \(11-37 ページ\)](#)
- [C/C++ \(Pro*C/C++\) : 初期化した BFILE ロケータを使用した BFILE を含む行の挿入 \(11-39 ページ\)](#)
- [Visual Basic \(OO4O\) : 初期化した BFILE ロケータを使用した BFILE を含む行の挿入 \(11-39 ページ\)](#)
- [Java \(JDBC\) : 初期化した BFILE ロケータを使用した BFILE を含む行の挿入 \(11-40 ページ\)](#)

PL/SQL: 初期化した BFILE ロケータを使用した BFILE を含む行の挿入

```
DECLARE
  /* BFILE ロケータを初期化します。 */
  Lob_loc BFILE := BFILENAME('PHOTO_DIR', 'Washington_photo');
BEGIN
  INSERT INTO Multimedia_tab (Clip_ID, Photo) VALUES (3, Lob_loc);
  COMMIT;
END;
```

C (OCI) : 初期化した BFILE ロケータを使用した BFILE を含む行の挿入

```
/* BFILE ロケータを使用して行を挿入します。 */
void insertUsingBfileLocator(envhp, svchp, stmthp, errhp)
OCIEnv *envhp;
OCISvcCtx *svchp;
OCIStmt *stmthp;
OCIError *errhp;
{
  text *insstmt =
    (text *) "INSERT INTO Multimedia_tab (Clip_ID, Photo) \
    VALUES (3, :Lob_loc)";
```

```

OCIBind *bndhp;
OCILobLocator *Lob_loc;
OraText *Dir = (OraText *)"PHOTO_DIR", *Name = (OraText *)"Washington_photo";

/* SQL 文を準備します。*/
checkerr (errhp, OCIStmtPrepare(stmthp, errhp, insstmt, (ub4)
                                strlen((char *) insstmt),
                                (ub4) OCI_NTV_SYNTAX, (ub4)OCI_DEFAULT));
/* ロケータ・リソースを割り当てます。*/
(void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &Lob_loc,
                          (ub4)OCI_DTYPE_FILE, (size_t) 0, (dvoid **) 0);
checkerr (errhp, OCILobFileSetName(envhp, errhp, &Lob_loc,
                                   Dir, (ub2)strlen((char *)Dir),
                                   Name, (ub2)strlen((char *)Name)));
checkerr (errhp, OCIBindByPos(stmthp, &bndhp, errhp, (ub4) 1,
                              (dvoid *) &Lob_loc, (sb4) 0, SQLT_BFILE,
                              (dvoid *) 0, (ub2 *)0, (ub2 *)0,
                              (ub4) 0, (ub4 *) 0, (ub4) OCI_DEFAULT));
/* SQL 文を実行します。*/
checkerr (errhp, OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                               (CONST OCISnapshot*) 0, (OCISnapshot*) 0,
                               (ub4) OCI_DEFAULT));
/* LOB リソースを解放します。*/
OCIDescriptorFree((dvoid *) Lob_loc, (ub4) OCI_DTYPE_FILE);
}

```

COBOL (Pro*COBOL) : 初期化した BFILE ロケータを使用した BFILE を含む行の挿入

```

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).

```

初期化した BFILE ロケータを使用した BFILE を含む行の挿入

```
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.
```

* BFILE ロケータの割当ておよび初期化を行います。
EXEC SQL ALLOCATE :SRC-BFILE END-EXEC.

* ディレクトリおよびファイル情報を設定します。
MOVE "PHOTO_DIR" TO DIR-ALIAS-ARR.
MOVE 9 TO DIR-ALIAS-LEN.
MOVE "washington_photo" TO FNAME-ARR.
MOVE 16 TO FNAME-LEN.

* ロケータのディレクトリ別名およびファイル名を設定します。
EXEC SQL
LOB FILE SET :SRC-BFILE DIRECTORY = :DIR-ALIAS,
FILENAME = :FNAME END-EXEC.

```
EXEC SQL  
INSERT INTO MULTIMEDIA_TAB (CLIP_ID, PHOTO)  
VALUES (6, :SRC-BFILE)END-EXEC.  
EXEC SQL 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++) : 初期化した BFILE ロケータを使用した BFILE を含む行の挿入

```
#include <oci.h>
#include <stdio.h>
#include <sqlca.h>
void Sample_Error()
{
    EXEC SQL WHENEVER SQLERROR CONTINUE;
    printf("%.*s\n", sqlca.sqlerrm.sqlerrml, sqlca.sqlerrm.sqlerrmc);
    EXEC SQL ROLLBACK WORK RELEASE;
    exit(1);
}

void insertBFILELocator_proc()
{
    OCIBFileLocator *Lob_loc;
    char *Dir = "PHOTO_DIR", *Name = "Washington_photo";
    EXEC SQL WHENEVER SQLERROR DO Sample_Error();
    /* 入力ロケータを割り当てます。*/
    EXEC SQL ALLOCATE :Lob_loc;
    /* 割り当てた (初期化した) ロケータのディレクトリおよびファイル名を設定します。*/
    EXEC SQL LOB FILE SET :Lob_loc DIRECTORY = :Dir, FILENAME = :Name;
    EXEC SQL INSERT INTO Multimedia_tab (Clip_ID, Photo) VALUES (4, :Lob_loc);
    /* ロケータが保持しているリソースを解放します。*/
    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) : 初期化した BFILE ロケータを使用した BFILE を含む行の挿入

```
Dim OraDyn as OraDynaset, OraPhoto as OraBFile, OraMusic as OraBFile
Set OraDyn = OraDb.CreateDynaset("select * from Multimedia_tab", ORADYN_DEFAULT)
Set OraMusic = OraDyn.Fields("Music").Value
Set OraPhoto = OraDyn.Fields("Photo").Value
```

'最初の行を編集し、"Photo" 列を初期化します。

```
OraDyn.Edit
OraPhoto.DirectoryName = "PHOTO_DIR"
OraPhoto.Filename = "Washington_photo"
OraDyn.Update
```

Java (JDBC) : 初期化した BFILE ロケータを使用した BFILE を含む行の挿入

```
// Java IOクラス :
import java.io.InputStream;
import java.io.OutputStream;

// コア JDBC クラス :
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 固有の JDBC クラス :
import oracle.sql.*;
import oracle.jdbc.driver.*;

public class Ex4_26
{
    public static void main (String args [])
        throws Exception
    {
        // Oracle JDBC Driver をロードします。
        DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

        // データベースに接続します。
        Connection conn =
            DriverManager.getConnection ("jdbc:oracle:oci8:@", "samp", "samp");
        conn.setAutoCommit (false);

        // 文を作成します。
        Statement stmt = conn.createStatement ();
        try
        {
            BFILE src_lob = null;
            ResultSet rset = null;
            OracleCallableStatement cstmt = null;
            rset = stmt.executeQuery (
                "SELECT BFILENAME('PHOTO_DIR', 'Washington_photo') FROM DUAL");
            if (rset.next ())
```

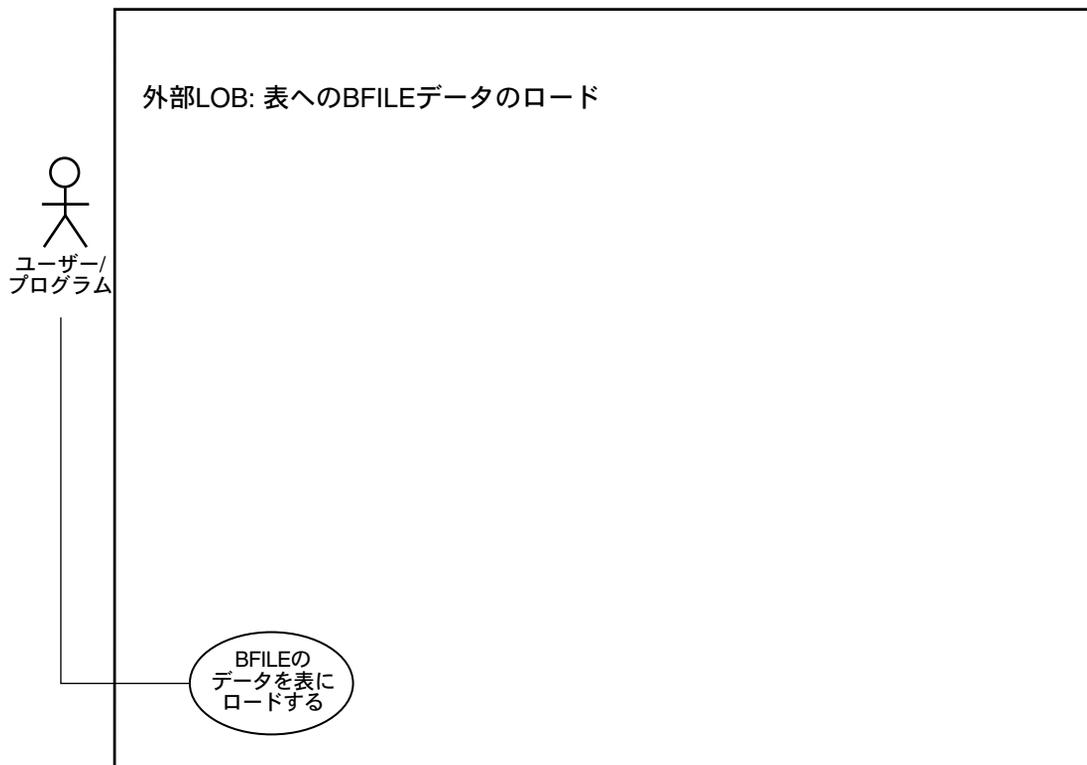
```
{
    src_lob = ((OracleResultSet)rset).getBFILE (1);
}

// CallableStatement を準備して、LOB を READWRITE でオープンします。
cstmt = (OracleCallableStatement) conn.prepareCall (
    "INSERT INTO multimedia_tab (clip_id, photo) VALUES (3, ?)");
cstmt.setBFILE(1, src_lob);
cstmt.execute();

// 文をクローズし、トランザクションをコミットします。
stmt.close();
cstmt.close();
conn.commit();
conn.close();
}
catch (SQLException e)
{
    e.printStackTrace();
}
}
```

外部 LOB (BFILE) へのデータのロード

図 11-10 ユースケース図: 外部 LOB (BFILE) への初期データのロード



参照: 外部 LOB (BFILE) に関するすべての基本操作については、11-2 ページの「ユースケース・モデル: 外部 LOB (BFILE)」を参照してください。

用途

初期データを BFILE に、また BFILE データを表にロードします。

使用上の注意

BFILE データ型では、非構造化バイナリ・データはデータベース外のオペレーティング・システム・ファイルに格納されます。

BFILE 列または BFILE 属性には、データを含んでいるサーバー側の外部ファイルを指すロケータが格納されます。

注意： BFILE としてロードされるファイルは、ロード時に実際に存在する必要はありません。

SQL*Loader は、必要な DIRECTORY オブジェクト（サーバーのファイル・システム上の物理ディレクトリに対する論理別名）がすでに作成されていることを想定しています。

参照： BFILE の詳細は、『Oracle8i アプリケーション開発者ガイド 基礎編』を参照してください。

BFILE 列に対応する制御ファイル・フィールドは、列名およびそれに続く BFILE 指示句で構成されます。

BFILE 指示句は、DIRECTORY オブジェクト名およびそれに続く BFILE 名を引数としてとります。DIRECTORY オブジェクト名および BFILE 名は、文字列定数として指定するか、その他のフィールドを介して動的にソース名を指定できます。

構文

次のマニュアルの項を参照してください。

- 『Oracle8i ユーティリティ・ガイド』の第 II 部「SQL*Loader」
- 4-5 ページの「LOB ロード時の SQL*Loader の使用」

使用例

次の 2 つの例では、BFILE のロードを説明します。最初の例では、ファイル名のみが動的に指定されます。2 番目の例では、BFILE および DIRECTORY オブジェクトが動的に指定されます。

注意： 次のようなデータ構造を設定しないと機能しない場合もあります。

```
CONNECT system/manager
GRANT CREATE ANY DIRECTORY to samp;
CONNECT samp/samp
CREATE OR REPLACE DIRECTORY detective_photo as '/tmp';
CREATE OR REPLACE DIRECTORY photo_dir as '/tmp';
```

例

次の例では、データを BFILE にロードします。

- BFILE へのデータのロード: ファイル名のみの動的な指定
- BFILE へのデータのロード: ファイル名および DIRECTORY オブジェクトの動的な指定

BFILE へのデータのロード: ファイル名のみの動的な指定

制御ファイル

```
LOAD DATA
INFILE sample9.dat
INTO TABLE Multimedia_tab
FIELDS TERMINATED BY ','
(Clip_ID      INTEGER EXTERNAL(5),
 FileName    FILLER CHAR(30),
 Photo       BFILE(CONSTANT "DETECTIVE_PHOTO", FileName))
```

データ・ファイル (sample9.dat)

```
007, JamesBond.jpeg,
008, SherlockHolmes.jpeg,
009, MissMarple.jpeg,
```

注意: サイズが指定されない場合、Clip_ID はデフォルト (255) になります。これはデータ・ファイル内のファイル名にマップされます。DETECTIVE_PHOTO は、すべてのファイルが格納されるディレクトリです。DETECTIVE_PHOTO は、事前に作成された DIRECTORY オブジェクトです。

BFILE へのデータのロード: ファイル名および DIRECTORY オブジェクトの動的な指定

制御ファイル

```
LOAD DATA
INFILE sample10.dat
INTO TABLE Multimedia_tab
replace
FIELDS TERMINATED BY ','
(
  Clip_ID    INTEGER EXTERNAL(5),
  Photo      BFILE (DirName, FileName),
  FileName   FILLER CHAR(30),
  DirName    FILLER CHAR(30)
)
```

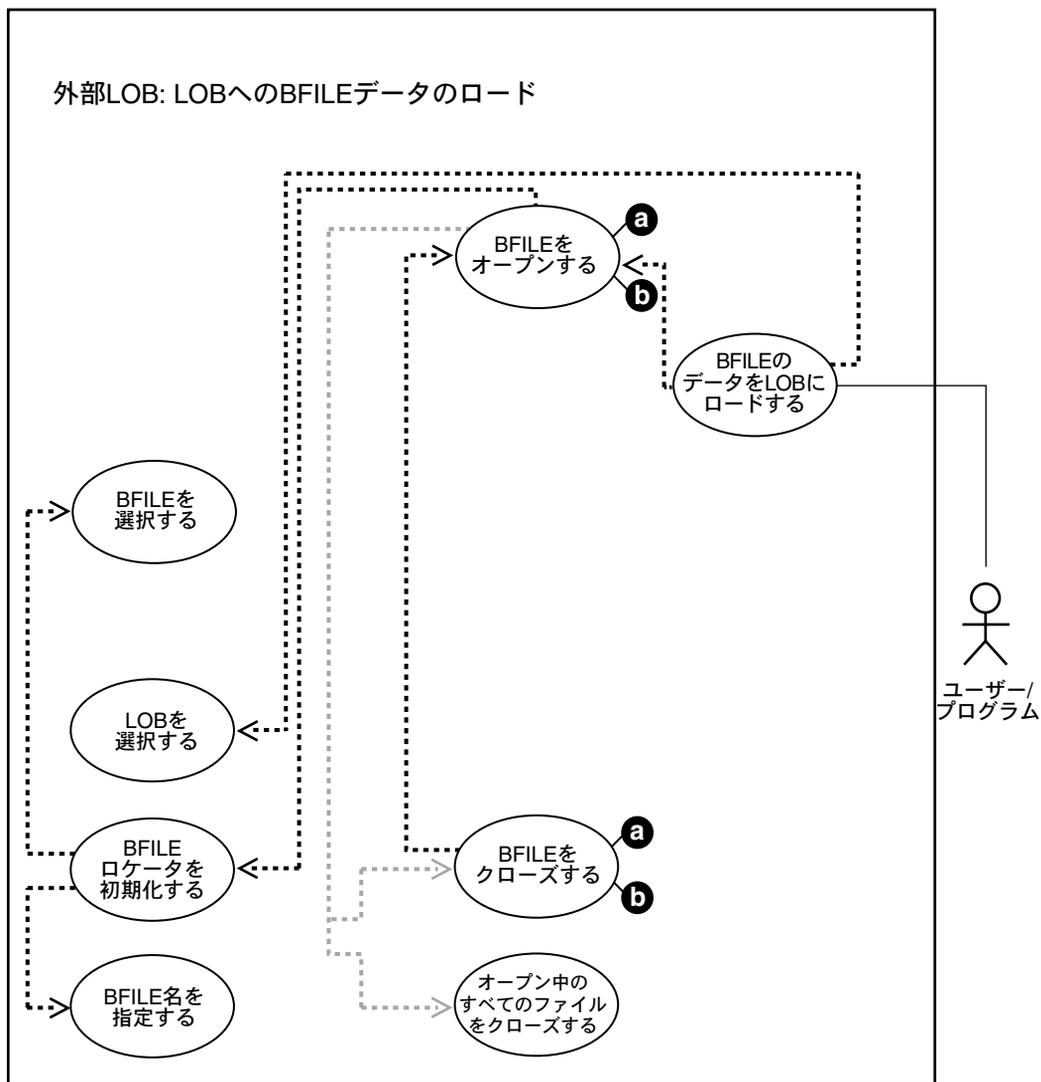
データ・ファイル (sample10.dat)

```
007,JamesBond.jpeg,DETECTIVE_PHOTO,
008,SherlockHolmes.jpeg,DETECTIVE_PHOTO,
009,MissMarple.jpeg,PHOTO_DIR,
```

注意: DirName FILLER CHAR (30) は、ロードされるファイルに対応するディレクトリ名を含むデータ・ファイル・フィールドにマップされます。

LOB への BFILE データのロード

図 11-11 ユースケース図：LOB への BFILE データのロード



参照: 外部 LOB (BFILE) に関するすべての基本操作については、11-2 ページの「ユースケース・モデル:外部 LOB (BFILE)」を参照してください。

用途

BFILE データを LOB にロードします。

使用上の注意

キャラクタ・セットの変換

OCI または OCI 機能にアクセスするプログラム環境を使用する場合、キャラクタ・セットの変換は、1 つのキャラクタ・セットから別のキャラクタ・セットに変換するとき、暗黙的に実行されます。

BFILE から CLOB または NCLOB へのロード: バイナリ・データからキャラクタ・セットへの変換

DBMS_LOB.LOADFROMFILE プロシージャを使用して CLOB または NCLOB に入れる場合は、BFILE のバイナリ・データを LOB に入れることとなります。バイナリ・データからキャラクタ・セットへの場合は、暗黙的な変換は実行されません。

したがって、データを BFILE から CLOB または NCLOB にロードするときは、loadfromfile を使用する前に、BFILE データについて次のことを確認してください。

- BFILE データが、すでにデータベースにある CLOB または NCLOB データと同じキャラクタ・セット (CHAR/NCHAR キャラクタ・セット) になっているか
- BFILE データが、サーバー・マシンの正しいエンディアン形式にコード化されているか

注意: CLOB または NCLOB データベースの CHAR/NCHAR キャラクタ・セットが可変幅の場合、BFILE のデータは ucs-2 キャラクタ・データを含んでいる必要があります。これは、データベースの CHAR/NCHAR キャラクタ・セットが可変幅の場合、CLOB または NCLOB データを ucs-2 形式で格納するためです。

参照: キャラクタ・セットの変換の問題点については、『Oracle8i NLS ガイド』を参照してください。

BFIL より小さいサイズの量パラメータの指定

- **DBMS_LOB.LOADFROMFILE**: 量パラメータを BFILE より大きいサイズに指定することはできません。
- **OCILobLoadFromFile**: 量パラメータを BFILE より大きい長さに指定することはできません。

構文

各プログラム環境で使用可能な機能のリストは、[第 3 章「LOB プログラム環境」](#)を参照してください。各プログラム環境における構文については、次のマニュアルの項を参照してください。

- **PL/SQL (DBMS_LOB)**: 『Oracle8i PL/SQL パッケージ・プロシージャ・リファレンス』の第 18 章「DBMS_LOB」の「サブプログラムの要約」の LOADFROMFILE プロシージャ
- **C (OCI)**: 『Oracle8i コール・インタフェース・プログラマーズ・ガイド』の第 7 章「LOB および FILE 操作」の使用上の注意および例、および第 15 章「OCI リレーショナル関数」の「LOB 関数」の OCILobLoadFromFile()
- **COBOL (Pro*COBOL)**: 『Oracle8i Pro*COBOL プリコンパイラ・プログラマーズ・ガイド』の第 13 章「ラージ・オブジェクト (LOB)」の「LOB 文」の使用上の注意、および付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB LOAD (実行可能埋込み SQL 拡張要素)」
- **C/C++ (Pro*C/C++)**: 『Oracle8i Pro*C/C++ プリコンパイラ・プログラマーズ・ガイド』の第 16 章「ラージ・オブジェクト (LOB)」の「LOB 文」の使用上の注意、および付録 F「埋込み SQL 文およびプリコンパイラ宣言文」、「LOB LOAD (実行可能埋込み SQL 拡張要素)」
- **Visual Basic (Oracle Objects for OLE オンライン・ヘルプ)**: 「Help」の「Contents」タブから、「OO4O Automation Server」>「OBJECTS」>「OraBfile」>「METHODS」>「CopyFromBfile」を選択してください。
- **Java (JDBC)**: 『Oracle8i JDBC 開発者ガイドおよびリファレンス』の第 7 章「LOB と BFILE の操作」の「BLOB と CLOB の操作」の「BLOB または CLOB 列の作成と移入」、および『Oracle8i SQLJ 開発者ガイドおよびリファレンス』の第 5 章「型のサポート」の「Oracle の拡張型」の「BLOB、CLOB および BFILE のサポート」

使用例

次の例は、ターゲット LOB (Music) にロードする LOB データを含むディレクトリ・オブジェクト (AUDIO_DIR) が存在することを想定しています。次のプログラム環境の例が示されています。

例

- PL/SQL (DBMS_LOB パッケージ) : LOB への BFILE データのロード (11-49 ページ)
- C (OCI) : LOB への BFILE データのロード (11-49 ページ)
- COBOL (Pro*COBOL) : LOB への BFILE データのロード (11-51 ページ)
- C/C++ (Pro*C/C++) : LOB への BFILE データのロード (11-52 ページ)
- Visual Basic (OO4O) : LOB への BFILE データのロード (11-54 ページ)
- Java (JDBC) : LOB への BFILE データのロード (11-54 ページ)

PL/SQL (DBMS_LOB パッケージ) : LOB への BFILE データのロード

```

/* 例のプロシージャ loadLOBFromBFILE_proc は、DBMS_LOB パッケージの
一部ではないことに注意してください。*/
CREATE OR REPLACE PROCEDURE loadLOBFromBFILE_proc IS
  Dest_loc      BLOB;
  Src_loc       BFILE := BFILENAME('FRAME_DIR', 'Washington_frame');
  Amount        INTEGER := 4000;
BEGIN
  SELECT Frame INTO Dest_loc FROM Multimedia_tab
     WHERE Clip_ID = 3
     FOR UPDATE;
  /* LOB のオープンは必須です。*/
  DBMS_LOB.OPEN(Src_loc, DBMS_LOB.LOB_READONLY);
  /* LOB のオープンはオプションです。*/
  DBMS_LOB.OPEN(Dest_loc, DBMS_LOB.LOB_READWRITE);
  DBMS_LOB.LOADFROMFILE(Dest_loc, Src_loc, Amount);
  /* LOB をオープンしている場合、その LOB をクローズする必要があります。*/
  DBMS_LOB.CLOSE(Dest_loc);
  DBMS_LOB.CLOSE(Src_loc);
  COMMIT;
END;

```

C (OCI) : LOB への BFILE データのロード

```

/* BFILENAME を使用して行を挿入します。*/
/* マルチメディア表から LOB/BFILE を選択します。*/
void selectLob(Lob_loc, errhp, svchp, stmthp)
OCILOBLocator *Lob_loc;
OCIError *errhp;
OCISvcCtx *svchp;
OCIStmt *stmthp;
{

```

```
char selstmt[150];
OCIDefine *dfnhp;

strcpy(selstmt, (char *) "SELECT FRAME FROM MULTIMEDIA_TAB \
                          WHERE CLIP_ID=3 FOR UPDATE");

/* SQL の SELECT 文を準備します。*/
checkerr (errhp, OCIStmtPrepare(stmthp, errhp, selstmt,
                                (ub4) strlen((char *) selstmt),
                                (ub4) OCI_NTV_SYNTAX, (ub4)OCI_DEFAULT));
/* 選択されている列を定義します。*/
checkerr (errhp, OCIDefineByPos(stmthp, &dfnhp, errhp, 1,
                                (dvoid *)&Lob_loc, 0 , SQLT_BLOB,
                                (dvoid *)0, (ub2 *)0, (ub2 *)0,
                                OCI_DEFAULT));
/* SQL の SELECT 文を実行します。*/
checkerr (errhp, OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                                (CONST OCISnapshot*) 0, (OCISnapshot*) 0,
                                (ub4) OCI_DEFAULT));
}

void loadLobFromBfile(envhp, errhp, svchp, stmthp)
OCIEnv *envhp;
OCIError *errhp;
OCISvcCtx *svchp;
OCIStmt *stmthp;
{
    OCILobLocator *dest_loc;
    OCILobLocator *src_loc;

    /* ロケータを割り当てます。*/
    (void) OCIDescriptorAlloc((dvoid *) envhp,
                              (dvoid **) &dest_loc, (ub4)OCI_DTYPE_FILE,
                              (size_t) 0, (dvoid **) 0);

    (void) OCIDescriptorAlloc((dvoid *) envhp,
                              (dvoid **) &src_loc, (ub4)OCI_DTYPE_FILE,
                              (size_t) 0, (dvoid **) 0);

    checkerr(errhp, OCILobFileSetName(envhp, errhp, &src_loc,
                                      (text *) "PHOTO_DIR", (ub2) strlen("PHOTO_DIR"),
                                      (text *) "Lincoln_photo", (ub2) strlen("Lincoln_photo")));

    selectLob(dest_loc, errhp, svchp, stmthp);

    checkerr(errhp, OCILobFileOpen(svchp, errhp, src_loc,
```

```

                                (ub1)OCI_FILE_READONLY));
checkerr(errhp, OCILobOpen(svchp, errhp, dest_loc, (ub1)OCI_LOB_READWRITE));
checkerr (errhp, OCILobLoadFromFile(svchp, errhp, dest_loc, src_loc,
                                (ub4)4000, (ub4)1, (ub4)1));
checkerr(errhp, OCILobClose(svchp, errhp, dest_loc));
checkerr(errhp, OCILobFileClose(svchp, errhp, src_loc));
}

```

COBOL (Pro*COBOL) : LOB への BFILE データのロード

```

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.

```

```

* LOB ロケータの割当ておよび初期化を行います。
EXEC SQL ALLOCATE :DEST-BLOB END-EXEC.
EXEC SQL ALLOCATE :SRC-BFILE END-EXEC.

```

```

* ディレクトリおよびファイル情報を設定します。
MOVE "AUDIO_DIR" TO DIR-ALIAS-ARR.
MOVE 9 TO DIR-ALIAS-LEN.
MOVE "washington_audio" TO FNAME-ARR.
MOVE 16 TO FNAME-LEN.

```

```

* BFILE を移入します。
EXEC SQL WHENEVER NOT FOUND GOTO END-OF-BFILE END-EXEC.
EXEC SQL

```

```
SELECT PHOTO INTO :SRC-BFILE
FROM MULTIMEDIA_TAB WHERE CLIP_ID = 3 END-EXEC.
```

* ソース BFILE を READ ONLY でオープンします。

* 宛先 BLOB を READ/WRITE でオープンします。

```
EXEC SQL LOB OPEN :SRC-BFILE READ ONLY END-EXEC.
EXEC SQL LOB OPEN :DEST-BLOB READ WRITE END-EXEC.
```

* BLOB に BFILE データをロードします。

```
EXEC SQL
LOB LOAD :AMT FROM FILE :SRC-BFILE INTO :DEST-BLOB END-EXEC.
```

* LOB をクローズします。

```
EXEC SQL LOB CLOSE :SRC-BFILE END-EXEC.
EXEC SQL LOB CLOSE :DEST-BLOB END-EXEC.
```

* LOB ロケータを解放します。

```
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 SQLEERROR CONTINUE
END-EXEC.
MOVE ORASLNR TO ORASLNRD.
DISPLAY " ".
DISPLAY "ORACLE ERROR DETECTED ON LINE ", ORASLNRD, ":".
DISPLAY " ".
DISPLAY SQLEERRMC.
EXEC SQL ROLLBACK WORK RELEASE END-EXEC.
STOP RUN.
```

C/C++ (Pro*C/C++) : LOB への BFILE データのロード

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

void Sample_Error()
{
```

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

void loadLOBFromBFILE_proc()
{
    OCIBlobLocator *Dest_loc;
    OCIBFileLocator *Src_loc;
    char *Dir = "AUDIO_DIR", *Name = "Washington_audio";
    int Amount = 4096;

    EXEC SQL WHENEVER SQLERROR DO Sample_Error();

    /* BFILE ロケータを初期化します。*/
    EXEC SQL ALLOCATE :Src_loc;
    EXEC SQL LOB FILE SET :Src_loc DIRECTORY = :Dir, FILENAME = :Name;

    /* BLOB ロケータを初期化します。*/
    EXEC SQL ALLOCATE :Dest_loc;
    EXEC SQL SELECT Sound INTO :Dest_loc FROM Multimedia_tab
        WHERE Clip_ID = 3 FOR UPDATE;

    /* BFILE のオープンは必須です。*/
    EXEC SQL LOB OPEN :Src_loc READ ONLY;

    /* BLOB のオープンはオプションです。*/
    EXEC SQL LOB OPEN :Dest_loc READ WRITE;
    EXEC SQL LOB LOAD :Amount FROM FILE :Src_loc INTO :Dest_loc;

    /* LOB および BFILE をオープンしている場合、それらをクローズする必要があります。*/
    EXEC SQL LOB CLOSE :Dest_loc;
    EXEC SQL LOB CLOSE :Src_loc;

    /* ロケータが保持しているリソースを解放します。*/
    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 (0040) : LOB への BFILE データのロード

```
Dim OraDyn as OraDynaset, OraDyn2 as OraDynaset, OraPhoto as OraBFile
Dim OraImage as OraBlob

chunksize = 32767
Set OraDyn = OraDb.CreateDynaset("select * from Multimedia_tab", ORADYN_DEFAULT)

Set OraPhoto = OraDyn.Fields("Photo").Value
Set OraFrame = OraDyn.Fields("Frame").Value

OraDyn.Edit
' BFILE のデータを LOB にロードします。
OraFrame.CopyFromBFile (OraPhoto)
OraDyn.Update
```

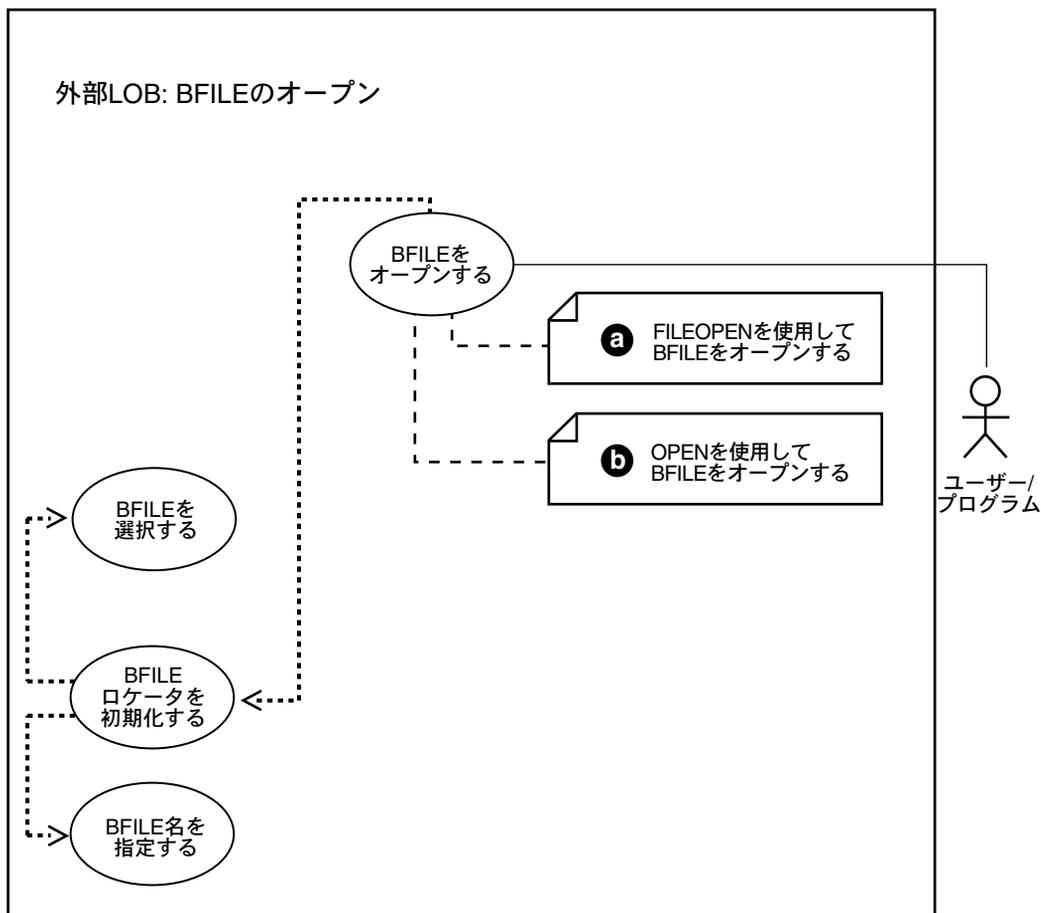
Java (JDBC) : LOB への BFILE データのロード

```
usage: head [-n #] [-#] [filename...]
public class

    // トランザクションをコミットします。
    conn.commit();
}
catch (SQLException e)
{
    e.printStackTrace();
}
}
```

BFILE をオープンする 2 つの方法

図 11-12 ユースケース図：BFILE をオープンする 2 つの方法



参照： 外部LOB (BFILE) に関するすべての基本操作については、11-2ページの「ユースケース・モデル:外部LOB (BFILE)」を参照してください。

推奨事項 : OPEN を使用した BFILE のオープン

コードを比較するとわかるように、これら 2 つのメソッドは、よく似ています。ただし、古い FILEOPEN 形式の使用を継続することもできますが、OPEN を使用すると将来の拡張性が向上するため、OPEN の使用に切り替えることをお勧めします。

- a. 「FILEOPEN を使用した BFILE のオープン」 (11-57 ページ)
- b. 「OPEN を使用した BFILE のオープン」 (11-62 ページ)

BFILE の最大オープン数の指定 : SESSION_MAX_OPEN_FILES

1 回のセッションにつき同時にオープンできる BFILE の数には制限があります。最大数は、初期化パラメータ SESSION_MAX_OPEN_FILES の使用によって指定されます。

SESSION_MAX_OPEN_FILES は、1 回のセッションで同時にオープンできるファイル数の上限を定義します。このパラメータのデフォルト値は 10 です。デフォルト値を使用していれば、1 回のセッションにつき最大 10 ファイルを同時にオープンできます。データベース管理者は、init.ora ファイルでこのパラメータの値を変更できます。次に例を示します。

```
SESSION_MAX_OPEN_FILES=20
```

クローズされていないファイルの数が SESSION_MAX_OPEN_FILES の値を超えると、そのセッションではそれ以上のファイルをオープンできなくなります。

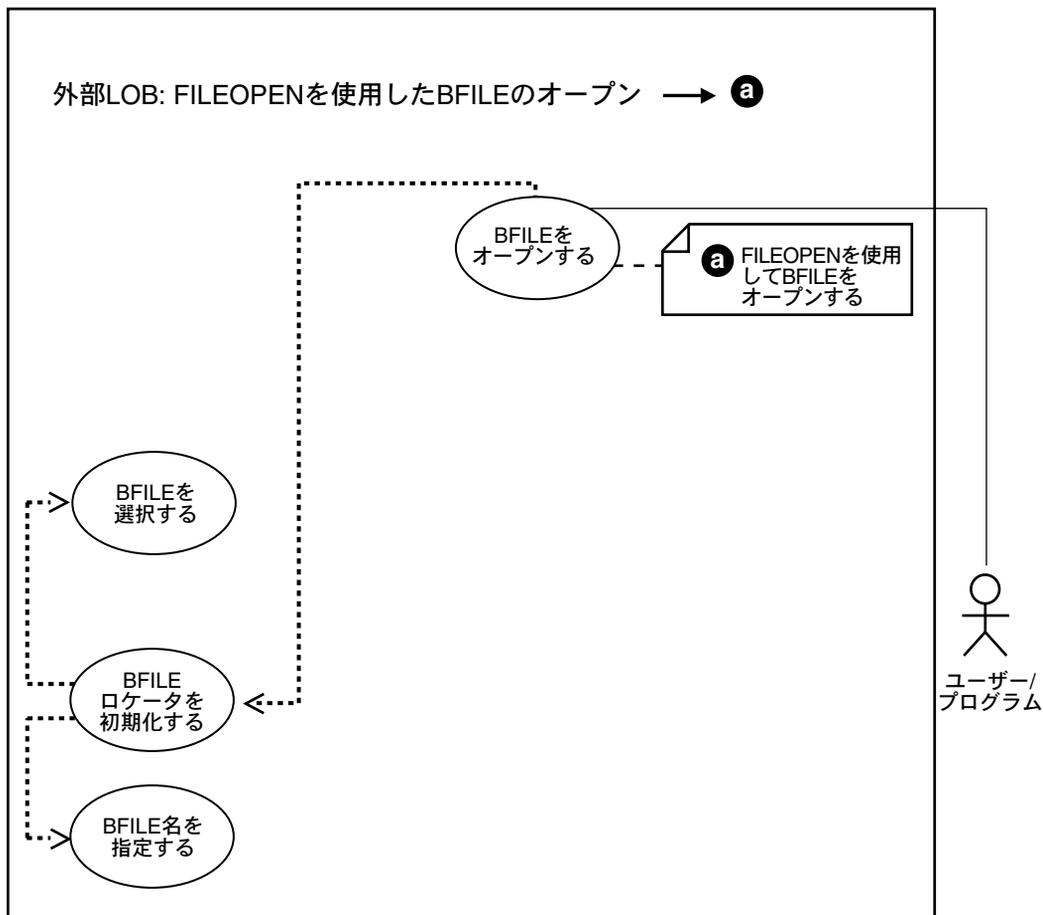
すべてのオープン・ファイルをクローズするには、FILECLOSEALL コールを使用します。

使用後のファイルのクローズ

SESSION_MAX_OPEN_FILES の値を小さく保つために、使用後にファイルをクローズすることをお勧めします。選択する値が大きくなるほど、メモリー使用量が増加します。

FILEOPEN を使用した BFILE のオープン

図 11-13 ユースケース図：FILEOPEN を使用した BFILE のオープン



参照： 外部LOB (BFILE) に関するすべての基本操作については、11-2 ページの「ユースケース・モデル:外部LOB (BFILE)」を参照してください。

用途

FILEOPEN を使用して BFILE をオープンします。

使用上の注意

古い FILEOPEN 形式の使用を継続することもできますが、OPEN を使用すると将来の拡張性が向上するため、OPEN の使用に切り替えることをお勧めします。

構文

各プログラム環境で使用可能な機能のリストは、[第3章「LOB プログラム環境」](#)を参照してください。各プログラム環境における構文については、次のマニュアルの項を参照してください。

- PL/SQL (DBMS_LOB) : 『Oracle8i PL/SQL パッケージ・プロシージャ・リファレンス』の第18章「DBMS_LOB」の「サブプログラムの要約」の FILEOPEN プロシージャ、FILECLOSE プロシージャ
- C (OCI) : 『Oracle8i コール・インタフェース・プログラマーズ・ガイド』の第7章「LOB および FILE 操作」の使用上の注意、および第15章「OCI リレーショナル関数」の「LOB 関数」の OCILobFileOpen()、OCILobFileClose()、OCILobFileName()
- COBOL (Pro*COBOL) : 参照マニュアルはありません。
- C/C++ (Pro*C/C++) : 参照マニュアルはありません。
- Visual Basic (OO4O) : 参照マニュアルはありません。
- Java (JDBC) : 『Oracle8i JDBC 開発者ガイドおよびリファレンス』の第7章「LOB と BFILE の操作」の「BLOB と CLOB の操作」の「BLOB または CLOB 列の作成と移入」、および『Oracle8i SQLJ 開発者ガイドおよびリファレンス』の第5章「型のサポート」の「Oracle の拡張型」の「BLOB、CLOB および BFILE のサポート」

使用例

次の例では、オペレーティング・システム・ファイル PHOTO_DIR 内の Lincoln_photo をオープンします。次のプログラム環境の例が示されています。

例

- [PL/SQL: FILEOPEN を使用した BFILE のオープン](#) (11-59 ページ)
- [C \(OCI\) : FILEOPEN を使用した BFILE のオープン](#) (11-59 ページ)
- COBOL (Pro*COBOL) : 今回のリリースでは例は記載されていません。
- C/C++ (Pro*C/C++) : 今回のリリースでは例は記載されていません。
- [Visual Basic \(OO4O\) : FILEOPEN を使用した BFILE のオープン](#) (11-60 ページ)
- [Java \(JDBC\) : FILEOPEN を使用した BFILE のオープン](#) (11-60 ページ)

PL/SQL: FILEOPEN を使用した BFILE のオープン

```

/* 例のプロシージャ openBFILE_procOne は、DBMS_LOB パッケージの
一部ではないことに注意してください。*/
CREATE OR REPLACE PROCEDURE openBFILE_procOne IS
  Lob_loc    BFILE := BFILENAME('PHOTO_DIR', 'Lincoln_photo');
BEGIN
  /* BFILE をオープンします。*/
  DBMS_LOB.FILEOPEN (Lob_loc, DBMS_LOB.FILE_READONLY);
  /* 何らかの処理を行います。*/
  DBMS_LOB.FILECLOSE (Lob_loc);
END;

```

C (OCI) : FILEOPEN を使用した BFILE のオープン

```

void BfileOpen(envhp, errhp, svchp, stmthp)
OCIEnv *envhp;
OCIError *errhp;
OCISvcCtx *svchp;
OCISmt *stmthp;
{
  OCILobLocator *bfile_loc;

  /* ロケータ記述子を割り当てます。*/
  (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &bfile_loc,
                           (ub4) OCI_DTYPE_FILE,
                           (size_t) 0, (dvoid **) 0);

  /* BFILE ロケータ情報を設定します。*/
  checkerr(errhp, (OCILobFileSetName(envhp, errhp, &bfile_loc,
                                     (OraText *) "PHOTO_DIR",
                                     (ub2) strlen("PHOTO_DIR"),
                                     (OraText *) "Lincoln_photo",
                                     (ub2) strlen("Lincoln_photo"))));
  checkerr(errhp, OCILobFileOpen(svchp, errhp, bfile_loc,
                                (ub1) OCI_FILE_READONLY));
  /* 何らかの処理を行います。*/
  checkerr(errhp, OCILobFileClose(svchp, errhp, bfile_loc));

  /* ロケータ記述子を解放します。*/
  OCIDescriptorFree((dvoid *) bfile_loc, (ub4) OCI_DTYPE_FILE);
}

```

Visual Basic (OO40) : FILEOPEN を使用した BFILE のオープン

注意： 現時点では、OO40 では、OPEN を使用した BFILE のオープン (11-67 ページの「[Visual Basic \(OO40\) : OPEN を使用した BFILE のオープン](#)」参照) のみが提供されています。

Java (JDBC) : FILEOPEN を使用した BFILE のオープン

```
import java.io.OutputStream;
// コア JDBC クラス :
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 固有の JDBC クラス :
import oracle.sql.*;
import oracle.jdbc.driver.*;

public class Ex4_38
{

    public static void main (String args [])
        throws Exception
    {
        // Oracle JDBC Driver をロードします。
        DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

        // データベースに接続します。
        Connection conn =
            DriverManager.getConnection ("jdbc:oracle:oci8:@", "samp", "samp");

        conn.setAutoCommit (false);

        // 文を作成します。
        Statement stmt = conn.createStatement ();
        try
        {
```

```
    BFILE src_lob = null;
    ResultSet rset = null;

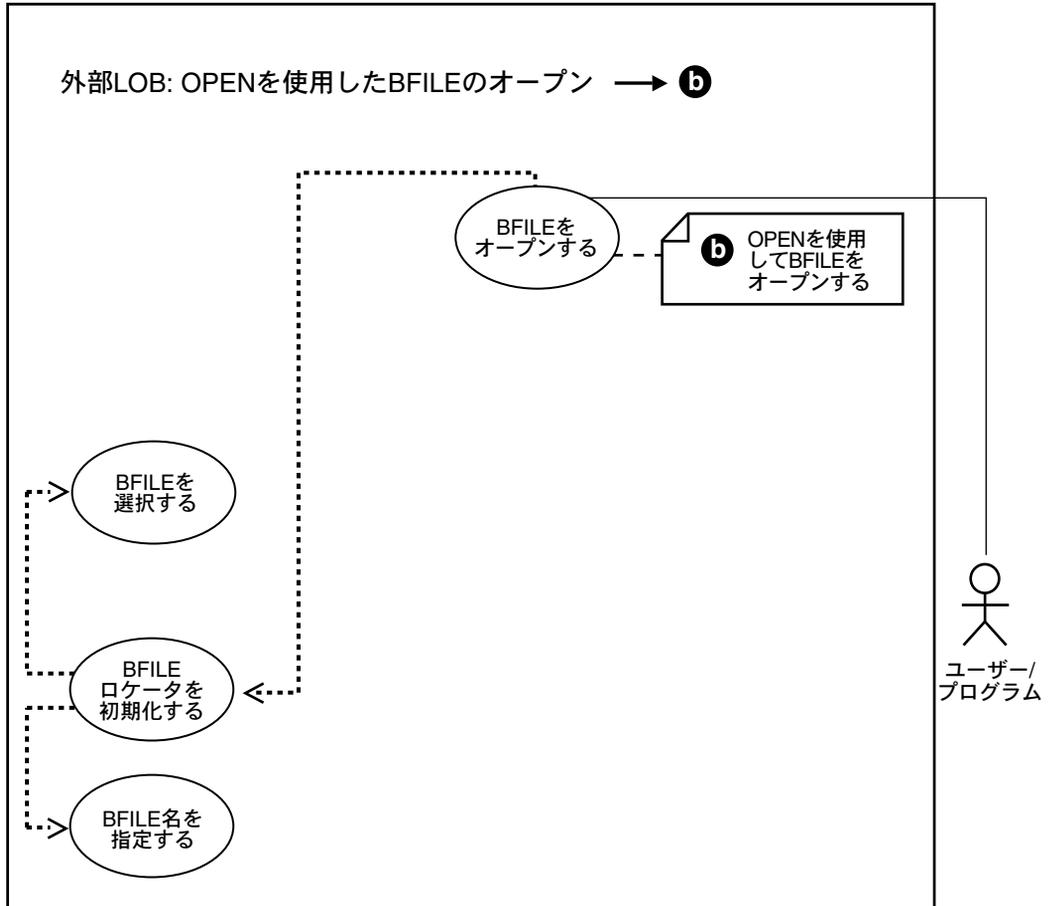
    rset = stmt.executeQuery (
        "SELECT BFILENAME('PHOTO_DIR', 'Lincoln_photo') FROM DUAL");
    if (rset.next())
    {
        src_lob = ((OracleResultSet)rset).getBFILE (1);

        src_lob.openFile();
        System.out.println("The file is now open");
    }

    // BFILE、文および接続をクローズします。
    src_lob.closeFile();
    stmt.close();
    conn.commit();
    conn.close();
}
catch (SQLException e)
{
    e.printStackTrace();
}
}
```

OPEN を使用した BFILE のオープン

図 11-14 ユースケース図：OPEN を使用した BFILE のオープン



参照： 外部 LOB (BFILE) に関するすべての基本操作については、11-2 ページの「ユースケース・モデル:外部 LOB (BFILE)」を参照してください。

用途

OPEN を使用して BFILE をオープンします。

使用上の注意

ありません。

構文

各プログラム環境で使用可能な機能のリストは、第3章「LOB プログラム環境」を参照してください。各プログラム環境における構文については、次のマニュアルの項を参照してください。

- PL/SQL (DBMS_LOB) : 『Oracle8i PL/SQL パッケージ・プロシージャ・リファレンス』の第18章「DBMS_LOB」の「サブプログラムの要約」の OPEN プロシージャ
- C (OCI) : 『Oracle8i コール・インタフェース・プログラマーズ・ガイド』の第7章「LOB および FILE 操作」の使用上の注意、および第15章「OCI リレーショナル関数」の「LOB 関数」の OCILobOpen()、OCILobClose()
- COBOL (Pro*COBOL) : 『Oracle8i Pro*COBOL プリコンパイラ・プログラマーズ・ガイド』の第13章「ラージ・オブジェクト (LOB)」の「LOB 文」の使用上の注意、および付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB OPEN (実行可能埋込み SQL 拡張要素)」
- C/C++ (Pro*C/C++) : 『Oracle8i Pro*C/C++ プリコンパイラ・プログラマーズ・ガイド』の第16章「ラージ・オブジェクト (LOB)」の「LOB 文」の使用上の注意、および付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB OPEN (実行可能埋込み SQL 拡張要素)」
- Visual Basic (Oracle Objects for OLE オンライン・ヘルプ) : 「Help」の「Contents」タブから、「OO4O Automation Server」>「OBJECTS」>「OraBfile」>「METHODS」>「Open」、および「OBJECTS」>「OraDynaset」>「METHODS」>「MoveLast」を選択してください。
- Java (JDBC) : 『Oracle8i JDBC 開発者ガイドおよびリファレンス』の第7章「LOB と BFILE の操作」の「BLOB と CLOB の操作」の「BLOB または CLOB 列の作成と移入」、および『Oracle8i SQLJ 開発者ガイドおよびリファレンス』の第5章「型のサポート」の「Oracle の拡張型」の「BLOB、CLOB および BFILE のサポート」

使用例

この例では、オペレーティング・システム・ファイル PHOTO_DIR 内の Lincoln_photo をオープンします。次のプログラム環境の例が示されています。

例

- PL/SQL: OPEN を使用した BFILE のオープン (11-64 ページ)
- C (OCI) : OPEN を使用した BFILE のオープン (11-64 ページ)
- C/C++ (Pro*C/C++) : OPEN を使用した BFILE のオープン (11-66 ページ)

- [COBOL \(Pro*COBOL\) : OPEN を使用した BFILE のオープン \(11-65 ページ\)](#)
- [Visual Basic \(OO4O\) : OPEN を使用した BFILE のオープン \(11-67 ページ\)](#)
- [Java \(JDBC\) : OPEN を使用した BFILE のオープン \(11-67 ページ\)](#)

PL/SQL: OPEN を使用した BFILE のオープン

```
/* 例のプロシージャ openBFILE_procTwo は、DBMS_LOB パッケージの
一部ではないことに注意してください。*/
CREATE OR REPLACE PROCEDURE openBFILE_procTwo IS
  Lob_loc    BFILE := BFILENAME('PHOTO_DIR', 'Lincoln_photo');
BEGIN
  /* BFILE をオープンします。*/
  DBMS_LOB.OPEN (Lob_loc, DBMS_LOB.LOB_READONLY);
  /* 何らかの処理を行います。*/
  DBMS_LOB.CLOSE (Lob_loc);
END;
```

C (OCI) : OPEN を使用した BFILE のオープン

```
void BfileFileOpen(envhp, errhp, svchp, stmthp)
OCIEnv      *envhp;
OCIError    *errhp;
OCISvcCtx   *svchp;
OCISstmt    *stmthp;
{

    OCILobLocator *bfile_loc;

    /* ロケータ記述子を割り当てます。*/
    (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &bfile_loc,
                              (ub4) OCI_DTYPE_FILE,
                              (size_t) 0, (dvoid **) 0);

    /* BFILE ロケータ情報を設定します。*/
    checkerr(errhp, (OCILobFileSetName(envhp, errhp, &bfile_loc,
                                       (OraText *) "PHOTO_DIR", (ub2)strlen("PHOTO_DIR"),
                                       (OraText *) "Lincoln_photo",
                                       (ub2)strlen("Lincoln_photo"))));
    checkerr(errhp, OCILobOpen(svchp, errhp, bfile_loc,
                              (ub1)OCI_FILE_READONLY));
    /* 何らかの処理を行います。*/
    checkerr(errhp, OCILobClose(svchp, errhp, bfile_loc));
}
```

```

/* ロケータ記述子を解放します。*/
OCIDescriptorFree((dvoid *)bfile_loc, (ub4)OCI_DTYPE_FILE);
}

```

COBOL (Pro*COBOL) : OPEN を使用した BFILE のオープン

```

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.

* BFILE ロケータの割当ておよび初期化を行います。
EXEC SQL ALLOCATE :SRC-BFILE END-EXEC.

* ディレクトリおよびファイル情報を設定します。
MOVE "AUDIO_DIR" TO DIR-ALIAS-ARR.
MOVE 9 TO DIR-ALIAS-LEN.
MOVE "Washington_audio" TO FNAME-ARR.
MOVE 16 TO FNAME-LEN.

* BFILE にディレクトリ別名およびファイル名を割り当てます。
EXEC SQL
LOB FILE SET :SRC-BFILE
DIRECTORY = :DIR-ALIAS, FILENAME = :FNAME END-EXEC.

* BFILE を READ ONLY でオープンします。
EXEC SQL LOB OPEN :SRC-BFILE READ ONLY END-EXEC.

```

```
* LOB をクローズします。
EXEC SQL LOB CLOSE :SRC-BFILE END-EXEC.

* LOB ロケータを解放します。
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++) : OPEN を使用した BFILE のオープン

/* Pro*C/C++ では、BFILE のオープンに使用する OPEN のフォームが1つしかありません。
FILE OPEN はなく、単純な OPEN 文のみです。*/

```
#include <oci.h>
#include <stdio.h>
#include <sqlca.h>
void Sample_Error()
{
    EXEC SQL WHENEVER SQLERROR CONTINUE;
    printf("%.*s\n", sqlca.sqlerrm.sqlerrml, sqlca.sqlerrm.sqlerrmc);
    EXEC SQL ROLLBACK WORK RELEASE;
    exit(1);
}

void openBFILE_proc()
{
    OCIBFileLocator *Lob_loc;
    char *Dir = "PHOTO_DIR", *Name = "Lincoln_photo";

    EXEC SQL WHENEVER SQLERROR DO Sample_Error();
    /* ロケータを初期化します。*/
    EXEC SQL ALLOCATE :Lob_loc;
    EXEC SQL LOB FILE SET :Lob_loc DIRECTORY = :Dir, FILENAME = :Name;
    /* BFILE をオープンします。*/
    EXEC SQL LOB OPEN :Lob_loc READ ONLY;
    /* 何らかの処理を行います。*/
```

```

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 (OO40) : OPEN を使用した BFILE のオープン

```

Dim OraDyn as OraDynaset, OraPhoto as OraBFile, OraMusic as OraBFile
Set OraDyn = OraDb.CreateDynaset("select * from Multimedia_tab",ORADYN_DEFAULT)
Set OraMusic = OraDyn.Fields("Music").Value
Set OraPhoto = OraDyn.Fields("Photo").Value

```

'最後の行に進み、BFILE を読みみ用にオープンします。

```

OraDyn.MoveLast
OraPhoto.Open 'Open Bfile for reading
'何らかの処理を行います。
OraPhoto.Close

```

Java (JDBC) : OPEN を使用した BFILE のオープン

```

import java.io.InputStream;
import java.io.OutputStream;

// コア JDBC クラス :
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 固有の JDBC クラス :
import oracle.sql.*;
import oracle.jdbc.driver.*;
public class Ex4_41
{
    public static void main (String args [])

```

```
        throws Exception
    {
        // Oracle JDBC Driver をロードします。
        DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

        // データベースに接続します。
        Connection conn =
            DriverManager.getConnection ("jdbc:oracle:oci8:@", "samp", "samp");
        conn.setAutoCommit (false);

        // 文を作成します。
        Statement stmt = conn.createStatement ();
        try
        {
            BFILE src_lob = null;
            ResultSet rset = null;

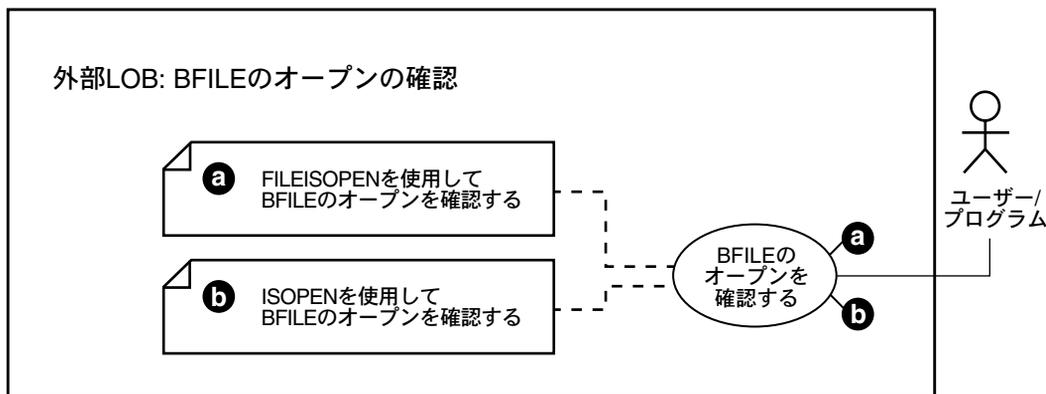
            rset = stmt.executeQuery (
                "SELECT BFILENAME('PHOTO_DIR', 'Lincoln_photo') FROM DUAL");
            if (rset.next())
            {
                src_lob = ((OracleResultSet)rset).getBFILE (1);

                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");
            }

            // BFILE、文および接続をクローズします。
            src_lob.closeFile();
            stmt.close();
            conn.commit();
            conn.close();
        }
        catch (SQLException e)
        {
            e.printStackTrace();
        }
    }
}
```

BFILE のオープンを確認する 2 つの方法

図 11-15 ユースケース図：BFILE のオープンを確認する 2 つの方法



参照： 外部LOB（BFILE）に関するすべての基本操作については、11-2ページの「ユースケース・モデル:外部LOB（BFILE）」を参照してください。

推奨事項：OPEN を使用した BFILE のオープン

コードを比較するとわかるように、これら2つのメソッドは、よく似ています。ただし、古い FILEISOPEN 形式の使用を継続することもできますが、ISOPEN を使用すると将来の拡張性が向上するため、ISOPEN の使用に切り替えることをお勧めします。

- a. [FILEISOPEN を使用した BFILE のオープンの確認](#) (11-71 ページ)
- b. [ISOPEN を使用した BFILE のオープンの確認](#) (11-77 ページ)

BFILE の最大オープン数の指定：SESSION_MAX_OPEN_FILES

1 回のセッションにつき同時にオープンできる BFILE の数には制限があります。最大数は、初期化パラメータ SESSION_MAX_OPEN_FILES の使用によって指定されます。

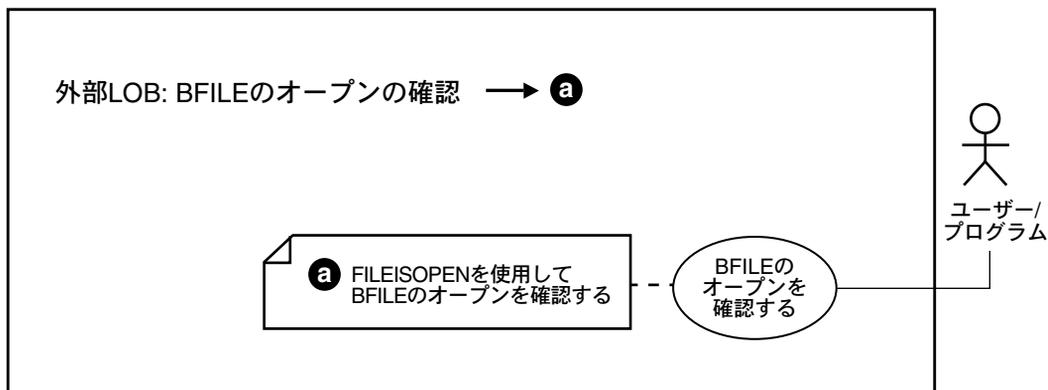
SESSION_MAX_OPEN_FILES は、1 回のセッションで同時にオープンできるファイル数の上限を定義します。このパラメータのデフォルト値は 10 です。デフォルト値を使用していれば、1 回のセッションにつき最大 10 ファイルを同時にオープンできます。データベース管理者は、init.ora ファイルでこのパラメータの値を変更できます。次に例を示します。

```
SESSION_MAX_OPEN_FILES=20
```

クローズされていないファイルの数が `SESSION_MAX_OPEN_FILES` の値を超えると、そのセッションではそれ以上のファイルをオープンできなくなります。すべてのオープンしているファイルをクローズするには、`FILECLOSEALL` コールを使用します。

FILEISOPEN を使用した BFILE のオープンの確認

図 11-16 ユースケース図：FILEISOPEN を使用した BFILE のオープンの確認



参照： 外部 LOB (BFILE) に関するすべての基本操作については、11-2 ページの「ユースケース・モデル:外部 LOB (BFILE)」を参照してください。

用途

FILEISOPEN を使用して、BFILE のオープンを確認します。

使用上の注意

古い FILEOPEN 形式の使用を継続することもできますが、ISOPEN を使用すると将来の拡張性が向上するため、ISOPEN の使用に切り替えることをお勧めします。

構文

各プログラム環境で使用可能な機能のリストは、第 3 章「LOB プログラム環境」を参照してください。各プログラム環境における構文については、次のマニュアルの項を参照してください。

- PL/SQL (DBMS_LOB) : 『Oracle8i PL/SQL パッケージ・プロシージャ・リファレンス』の第 18 章「DBMS_LOB」の「サブプログラムの要約」の FILEISOPEN ファンクション

- C (OCI) : 『Oracle8i コール・インタフェース・プログラマーズ・ガイド』の第7章「LOB および FILE 操作」の使用上の注意、および第15章「OCI リレーショナル関数」の「LOB 関数」の OCILobFileIsOpen()
- COBOL (Pro*COBOL) : 参照マニュアルはありません。
- C/C++ (Pro*C/C++) : 参照マニュアルはありません。
- Visual Basic (OO4O) : 参照マニュアルはありません。
- Java (JDBC) : 『Oracle8i JDBC 開発者ガイドおよびリファレンス』の第7章「LOB と BFILE の操作」の「BLOB と CLOB の操作」の「BLOB または CLOB 列の作成と移入」、および『Oracle8i SQLJ 開発者ガイドおよびリファレンス』の第5章「型のサポート」の「Oracle の拡張型」の「BLOB、CLOB および BFILE のサポート」

使用例

次の例では、Music に対応付けられている BFILE がオープンしているかどうかを問い合わせます。次のプログラム環境の例が示されています。

例

- [PL/SQL \(DBMS_LOB パッケージ\) : FILEISOPEN を使用した BFILE のオープンの確認 \(11-72 ページ\)](#)
- [C \(OCI\) : FILEISOPEN を使用した BFILE のオープンの確認 \(11-73 ページ\)](#)
- C/C++ (Pro*C/C++) : 今回のリリースでは例は記載されていません。
- COBOL (Pro*COBOL) : 今回のリリースでは例は記載されていません。
- Visual Basic (OO4O) : 例はありません。11-74 ページの「注意」を参照してください
- [Java \(JDBC\) : FILEISOPEN を使用した BFILE のオープンの確認 \(11-75 ページ\)](#)

PL/SQL (DBMS_LOB パッケージ) : FILEISOPEN を使用した BFILE のオープンの確認

```
/* 例のプロシージャ seeIfOpenBFILE_procOne は、DBMS_LOB パッケージの
一部ではないことに注意してください。*/
CREATE OR REPLACE PROCEDURE seeIfOpenBFILE_procOne IS
  Lob_loc      BFILE;
  RetVal      INTEGER;
BEGIN
  /* LOB を選択し、BFILE ロケータを初期化します。*/
  SELECT Music INTO Lob_loc FROM Multimedia_tab
     WHERE Clip_ID = 3;
  RetVal := DBMS_LOB.FILEISOPEN(Lob_loc);
```

```

IF (RetVal = 1)
    THEN
        DBMS_OUTPUT.PUT_LINE('File is open');
    ELSE
        DBMS_OUTPUT.PUT_LINE('File is not open');
    END IF;
EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Operation failed');
END;

```

C (OCI) : FILEISOPEN を使用した BFILE のオープンの確認

```

/* マルチメディア表から LOB/BFILE を選択します。*/
void selectLob(Lob_loc, errhp, svchp, stmthp)
OCILobLocator *Lob_loc;
OCIError      *errhp;
OCISvcCtx     *svchp;
OCIStmt       *stmthp;
{
    OCIDefine *dfnhp;
    text *selstmt =
        (text *) "SELECT Music FROM Multimedia_tab WHERE Clip_ID=3";

    /* SQL の SELECT 文を準備します。*/
    checkerr (errhp, OCIStmtPrepare(stmthp, errhp, selstmt,
                                     (ub4) strlen((char *) selstmt),
                                     (ub4) OCI_NTV_SYNTAX, (ub4)OCI_DEFAULT));

    /* BFILE 列に対する定義をコールします。*/
    checkerr (errhp, OCIDefineByPos(stmthp, &dfnhp, errhp, 1,
                                     (dvoid *)&Lob_loc, 0, SQLT_BFILE,
                                     (dvoid *)0, (ub2 *)0, (ub2 *)0,
                                     OCI_DEFAULT));

    /* SQL の SELECT 文を実行します。*/
    checkerr (errhp, OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                                     (CONST OCISnapshot*) 0, (OCISnapshot*) 0,
                                     (ub4) OCI_DEFAULT));
}

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

```

```
OCILOBLocator *bfile_loc;
boolean flag;

/* ロケータ記述子を割り当てます。*/
(void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &bfile_loc,
                          (ub4) OCI_DTYPE_FILE,
                          (size_t) 0, (dvoid **) 0);

/* BFILE を選択します。*/
selectLob(bfile_loc, errhp, svchp, stmp);

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));

/* ロケータ記述子を解放します。*/
OCIDescriptorFree((dvoid *)bfile_loc, (ub4)OCI_DTYPE_FILE);
}
```

Visual Basic (OO4O) : FILEISOPEN を使用した BFILE のオープンの確認

注意： 現時点では、OO4O では、BFILE がオープンしているかどうかをテストするためには ISOPEN のみが提供されています (11-74 ページの「[Visual Basic \(OO4O\) : FILEISOPEN を使用した BFILE のオープンの確認](#)」を参照してください)。

Java (JDBC) : FILEISOPEN を使用した BFILE のオープンの確認

```
import java.io.InputStream;
import java.io.OutputStream;

// コア JDBC クラス :
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 固有の JDBC クラス :
import oracle.sql.*;
import oracle.jdbc.driver.*;
public class Ex4_41
{
    public static void main (String args [])
        throws Exception
    {
        // Oracle JDBC Driver をロードします。
        DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

        // データベースに接続します。
        Connection conn =
        DriverManager.getConnection ("jdbc:oracle:oci8:@", "samp", "samp");
        conn.setAutoCommit (false);

        // 文を作成します。
        Statement stmt = conn.createStatement ();
        try
        {
            BFILE src_lob = null;
            ResultSet rset = null;

            rset = stmt.executeQuery (
                "SELECT BFILENAME('PHOTO_DIR', 'Lincoln_photo') FROM DUAL");
            if (rset.next ())
            {
                src_lob = ((OracleResultSet)rset).getBFILE (1);

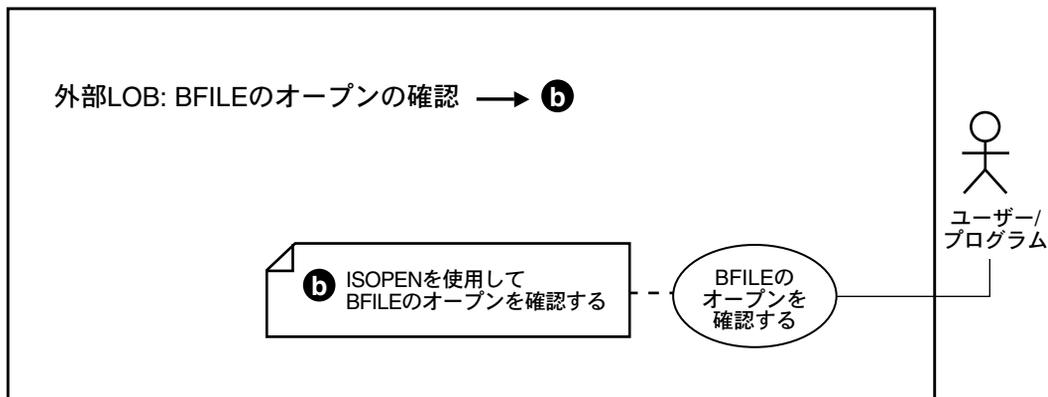
                OracleCallableStatement cstmt = (OracleCallableStatement)
                conn.prepareCall ("begin dbms_lob.open (?,dbms_lob.lob_readonly);
end;");
```

```
        pstmt.registerOutParameter(1,OracleTypes.BFILE);
        pstmt.setBFILE (1, src_lob);
        pstmt.execute();
        src_lob = pstmt.getBFILE(1);
        System.out.println ("the file is now open");
    }

    // BFILE、文および接続をクローズします。
    src_lob.closeFile();
    stmt.close();
    conn.commit();
    conn.close();
}
catch (SQLException e)
{
    e.printStackTrace();
}
}
```

ISOPEN を使用した BFILE のオープンの確認

図 11-17 ユースケース図：ISOPEN を使用した BFILE のオープンの確認



参照： 外部 LOB (BFILE) に関するすべての基本操作については、11-2 ページの「ユースケース・モデル:外部 LOB (BFILE)」を参照してください。

用途

ISOPEN を使用して、BFILE のオープンを確認します。

使用上の注意

ありません。

構文

各プログラム環境で使用可能な機能のリストは、第 3 章「LOB プログラム環境」を参照してください。各プログラム環境における構文については、次のマニュアルの項を参照してください。

- PL/SQL (DBMS_LOB) : 『Oracle8i PL/SQL パッケージ・プロシージャ・リファレンス』の第 18 章「DBMS_LOB」の「サブプログラムの要約」の ISOPEN ファンクション
- C (OCI) : 『Oracle8i コール・インタフェース・プログラマーズ・ガイド』の第 7 章「LOB および FILE 操作」の使用上の注意、および第 15 章「OCI リレーショナル関数」の「LOB 関数」の OCILobFileIsOpen()

- COBOL (Pro*COBOL) : 『Oracle8i Pro*COBOL プリコンパイラ・プログラマーズ・ガイド』の第13章「ラージ・オブジェクト (LOB)」の「LOB 文」の使用上の注意、および付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB DESCRIBE (実行可能埋込み SQL 拡張要素)」、および『Oracle8i PL/SQL パッケージ・プロシージャ・リファレンス』の第18章「DBMS_LOB」の「サブプログラムの要約」の ISOPEN ファンクション
- C/C++ (Pro*C/C++) : 『Oracle8i Pro*C/C++ プリコンパイラ・プログラマーズ・ガイド』の第16章「ラージ・オブジェクト (LOB)」の「LOB 文」の使用上の注意、および付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB DESCRIBE (実行可能埋込み SQL 拡張要素)」、および『Oracle8i PL/SQL パッケージ・プロシージャ・リファレンス』の第18章「DBMS_LOB」の「サブプログラムの要約」の ISOPEN ファンクション
- Visual Basic (Oracle Objects for OLE オンライン・ヘルプ): 「Help」の「Contents」タブから、「OO4O Automation Server」>「OBJECTS」>「OraBfile」>「METHODS」>「IsOpen」、および>「OBJECTS」>「OraDynaset」を選択してください。
- Java (JDBC) : 『Oracle8i JDBC 開発者ガイドおよびリファレンス』の第7章「LOB と BFILE の操作」の「BLOB と CLOB の操作」の「BLOB または CLOB 列の作成と移入」、および『Oracle8i SQLJ 開発者ガイドおよびリファレンス』の第5章「型のサポート」の「Oracle の拡張型」の「BLOB、CLOB および BFILE のサポート」

使用例

次の例では、Music に対応付けられた BFILE がオープンしているかどうかを問い合わせます。

例

次のプログラム環境の例が示されています。

- [PL/SQL \(DBMS_LOB パッケージ\) : ISOPEN を使用した BFILE のオープンの確認 \(11-78 ページ\)](#)
- [C \(OCI\) : ISOPEN を使用した BFILE のオープンの確認 \(11-79 ページ\)](#)
- [COBOL \(Pro*COBOL\) : ISOPEN を使用した BFILE のオープンの確認 \(11-81 ページ\)](#)
- [C/C++ \(Pro*C/C++\) : ISOPEN を使用した BFILE のオープンの確認 \(11-82 ページ\)](#)
- [Visual Basic \(OO4O\) : ISOPEN を使用した BFILE のオープンの確認 \(11-83 ページ\)](#)
- [Java \(JDBC\) : ISOPEN を使用した BFILE のオープンの確認 \(11-83 ページ\)](#)

PL/SQL (DBMS_LOB パッケージ) : ISOPEN を使用した BFILE のオープンの確認

```

/* 例のプロシージャ seeIfOpenBFILE_procTwo は、DBMS_LOB パッケージの
一部ではないことに注意してください。*/
CREATE OR REPLACE PROCEDURE seeIfOpenBFILE_procTwo IS
  Lob_loc      BFILE;
  RetVal       INTEGER;
BEGIN
  /* LOB を選択し、BFILE ロケータを初期化します。*/
  SELECT Music INTO Lob_loc FROM Multimedia_tab
     WHERE Clip_ID = 3;
  RetVal := DBMS_LOB.ISOPEN(Lob_loc);
  IF (RetVal = 1)
  THEN
    DBMS_OUTPUT.PUT_LINE('File is open');
  ELSE
    DBMS_OUTPUT.PUT_LINE('File is not open');
  END IF;
EXCEPTION
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('Operation failed');
END;

```

C (OCI) : ISOPEN を使用した BFILE のオープンの確認

```

/* マルチメディア表から LOB/BFILE を選択します。*/
void selectLob(Lob_loc, errhp, svchp, stmthp)
OCILobLocator *Lob_loc;
OCIError      *errhp;
OCISvcCtx     *svchp;
OCIStmt       *stmthp;
{
  OCIDefine *dfnhp;
  text *selstmt =
    (text *) "SELECT Music FROM Multimedia_tab WHERE Clip_ID=3";

  /* SQL の SELECT 文を準備します。*/
  checkerr (errhp, OCIStmtPrepare(stmthp, errhp, selstmt,
    (ub4) strlen((char *) selstmt),
    (ub4) OCI_NTV_SYNTAX, (ub4) OCI_DEFAULT));

  /* BFILE 列に対する定義をコールします。*/
  checkerr (errhp, OCIDefineByPos(stmthp, &dfnhp, errhp, 1,
    (dvoid *)&Lob_loc, 0, SQLT_BFILE,

```

```
(dvoid *)0, (ub2 *)0, (ub2 *)0,
OCI_DEFAULT));

/* SQL の SELECT 文を実行します。*/
checkerr (errhp, OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                                (CONST OCI_Snapshot*) 0, (OCI_Snapshot*) 0,
                                (ub4) OCI_DEFAULT));
}
void BfileIsOpen(envhp, errhp, svchp, stmthp)
OCIEnv *envhp;
OCIError *errhp;
OCISvcCtx *svchp;
OCIStmt *stmthp;
{
    OCILobLocator *bfile_loc;
    boolean flag;

    /* ロケータ記述子を割り当てます。*/
    (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &bfile_loc,
                              (ub4) OCI_DTYPE_FILE,
                              (size_t) 0, (dvoid **) 0);

    /* BFILE を選択します。*/
    selectLob(bfile_loc, errhp, svchp, stmthp);

    checkerr(errhp, OCILobOpen(svchp, errhp, bfile_loc,
                              (ub1)OCI_FILE_READONLY));

    checkerr(errhp, OCILobIsOpen(svchp, errhp, bfile_loc, &flag));

    if (flag == TRUE)
    {
        printf("File is open\n");
    }
    else
    {
        printf("File is not open\n");
    }

    checkerr(errhp, OCILobFileClose(svchp, errhp, bfile_loc));

    /* ロケータ記述子を解放します。*/
    OCIDescriptorFree((dvoid *)bfile_loc, (ub4)OCI_DTYPE_FILE);
}
}
```

COBOL (Pro*COBOL) : ISOPEN を使用した BFILE のオープンの確認

```
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.
```

* BFILE ロケータの割当ておよび初期化を行います。
EXEC SQL ALLOCATE :SRC-BFILE END-EXEC.

* ディレクトリおよびファイル情報を設定します。
MOVE "AUDIO_DIR" TO DIR-ALIAS-ARR.
MOVE 9 TO DIR-ALIAS-LEN.
MOVE "Washington_audio" TO FNAME-ARR.
MOVE 16 TO FNAME-LEN.

* BFILE にディレクトリ別名およびファイル名を割り当てます。
EXEC SQL
 LOB FILE SET :SRC-BFILE
 DIRECTORY = :DIR-ALIAS, FILENAME = :FNAME
END-EXEC.

* BFILE を READ ONLY でオープンします。
EXEC SQL
 LOB OPEN :SRC-BFILE READ ONLY
END-EXEC.

* LOB をクローズします。
EXEC SQL LOB CLOSE :SRC-BFILE END-EXEC.

```
* LOB ロケータを解放します。
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++) : ISOPEN を使用した BFILE のオープンの確認

/* Pro*C/C++ では、BFILE がオープンしているかどうかの判断に使用する ISOPEN のフォームが 1 つしかありません。FILE IS OPEN はなく、単純な ISOPEN のみです。これは、DESCRIBE 文に使用される属性です。*/

```
#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;
/* ロケータに BFILE を選択します。*/
EXEC SQL SELECT Music INTO :Lob_loc FROM Multimedia_tab
      WHERE Clip_ID = 3;
/* BFILE がオープンしているかどうかを判断します。*/
EXEC SQL LOB DESCRIBE :Lob_loc GET ISOPEN into :isOpen;
if (isOpen)
    printf("BFILE is open\n");
else
    printf("BFILE is not open\n");
/* この例では、BFILE がオープンしていないことに注意してください。*/
EXEC SQL FREE :Lob_loc;
}

void main()
{
    char *samp = "samp/samp";
    EXEC SQL CONNECT :samp;
    seeIfOpenBFILE_proc();
    EXEC SQL ROLLBACK WORK RELEASE;
}

```

Visual Basic (OO40) : ISOPEN を使用した BFILE のオープンの確認

```

Dim OraDyn as OraDynaset, OraMusic as OraBFile, amount_read%, chunksize%, chunk

chunksize = 32767
Set OraDyn = OraDb.CreateDynaset("select * from Multimedia_tab", ORADYN_DEFAULT)
Set OraMusic = OraDyn.Fields("Music").Value

If OraMusic.IsOpen then
    ' ファイルがオープンしている場合の処理が行われます。
Else
    ' ファイルがオープンしていない場合の処理が行われるか、またはエラーが戻されます。
End If

```

Java (JDBC) : ISOPEN を使用した BFILE のオープンの確認

```

import java.io.InputStream;
import java.io.OutputStream;

// コア JDBC クラス :
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 固有の JDBC クラス :
import oracle.sql.*;
import oracle.jdbc.driver.*;

public class Ex4_48
{

    public static void main (String args [])
        throws Exception
    {
        // Oracle JDBC Driver をロードします。
        DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

        // データベースに接続します。
        Connection conn =
        DriverManager.getConnection ("jdbc:oracle:oci8:@", "samp", "samp");
        conn.setAutoCommit (false);

        // 文を作成します。
        Statement stmt = conn.createStatement ();
        try
        {
            BFILE src_lob = null;
            ResultSet rset = null;
            Boolean result = null;
            rset = stmt.executeQuery (
                "SELECT BFILENAME('PHOTO_DIR', 'Lincoln_photo') FROM DUAL");
            if (rset.next())
            {
                src_lob = ((OracleResultSet)rset).getBFILE (1);
            }
            result = new Boolean(src_lob.isFileOpen());
            System.out.println(
                "result of fileIsOpen() before opening file : " + result.toString());
            src_lob.openFile();
            result = new Boolean(src_lob.isFileOpen());
            System.out.println(
                "result of fileIsOpen() after opening file : " + result.toString());
        }
    }
}
```

```
// BFILE、文および接続をクローズします。
src_lob.closeFile();

int i = pstmt.getInt(1);
System.out.println("The result is: " + Integer.toString(i));

OracleCallableStatement pstmt2 = (OracleCallableStatement)
conn.prepareCall (
"BEGIN DBMS_LOB.OPEN(?,DBMS_LOB.LOB_READONLY); END;");
pstmt2.setBFILE(1, bfile);
pstmt2.execute();

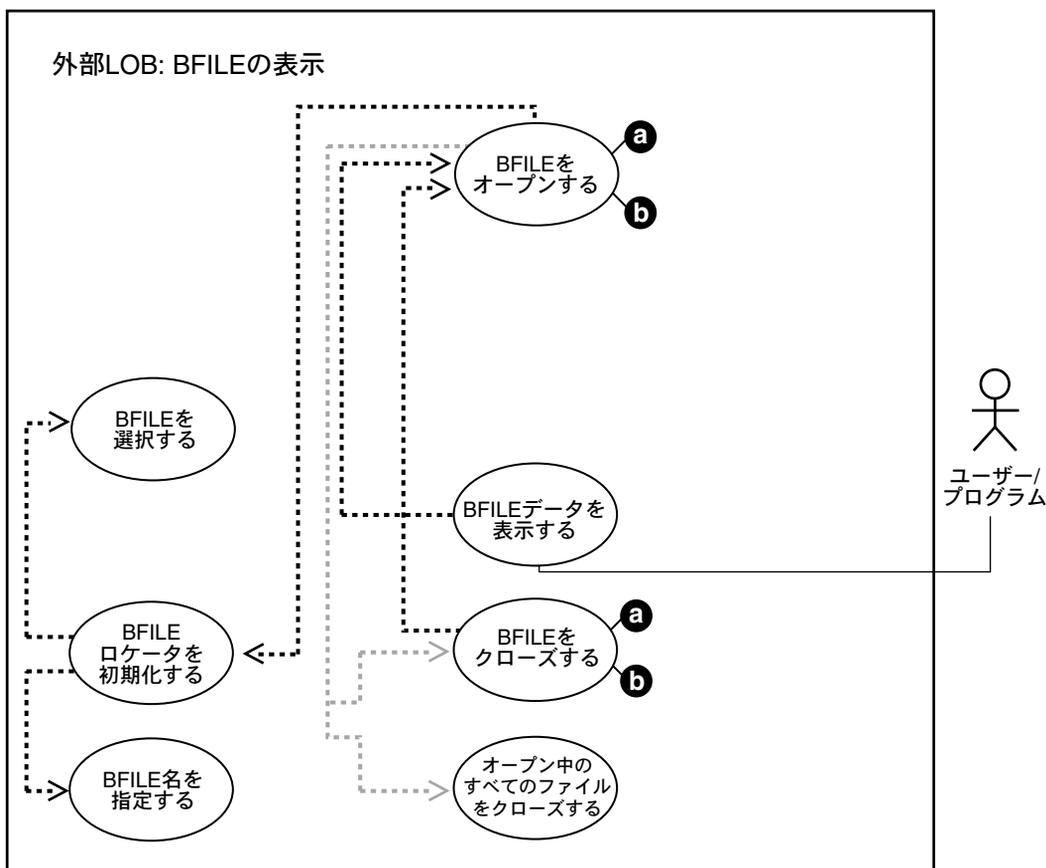
System.out.println("The BFILE has been opened with a call to "
+"DBMS_LOB.OPEN()");

// 既存の pstmt ハンドルを使用して、ロケータの状態を再問合せします。
pstmt.setBFILE(2, bfile);
pstmt.execute();
i = pstmt.getInt(1);
System.out.println("This result is: " + Integer.toString(i));

stmt.close();
conn.commit();
conn.close();
}
catch (SQLException e)
{
    e.printStackTrace();
}
}
```

BFILE データの表示

図 11-18 ユースケース図：BFILE データの表示



参照： 外部 LOB (BFILE) に関するすべての基本操作については、11-2 ページの「ユースケース・モデル: 外部 LOB (BFILE)」を参照してください。

用途

BFILE データを表示します。

使用上の注意

ありません。

構文

各プログラム環境で使用可能な機能のリストは、第3章「LOB プログラム環境」を参照してください。各プログラム環境における構文については、次のマニュアルの項を参照してください。

- PL/SQL (DBMS_LOB) : 『Oracle8i PL/SQL パッケージ・プロシージャ・リファレンス』の第18章「DBMS_LOB」の「サブプログラムの要約」の READ プロシージャ、および第29章「DBMS_OUTPUT」の「サブプログラムの要約」の PUT および PUT_LINE プロシージャ
- C (OCI) : 『Oracle8i コール・インタフェース・プログラマーズ・ガイド』の第7章「LOB および FILE 操作」の使用上の注意、および第15章「OCI リレーショナル関数」の「LOB 関数」の OCILobFileOpen()、OCILobRead()
- COBOL (Pro*COBOL) : 『Oracle8i Pro*COBOL プリコンパイラ・プログラマーズ・ガイド』の第13章「ラージ・オブジェクト (LOB)」の「LOB 文」の使用上の注意、および付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB READ (実行可能埋込み SQL 拡張要素)」、「DISPLAY (実行可能埋込み SQL 拡張要素)」
- C/C++ (Pro*C/C++) : 『Oracle8i Pro*C/C++ プリコンパイラ・プログラマーズ・ガイド』の第16章「ラージ・オブジェクト (LOB)」の「LOB 文」の「READ」
- Visual Basic (Oracle Objects for OLE オンライン・ヘルプ): 「Help」の「Contents」タブから、「OO4O Automation Server」>「OBJECTS」>「OraBfile」>「METHODS」>「Read」、および「OBJECTS」>「OraBfile」>「PROPERTIES」>「Polling」、「Offset」、「Status」を選択してください。または「OBJECTS」>「OraBfile」>「Examples」を選択してください。
- Java (JDBC) : 『Oracle8i JDBC 開発者ガイドおよびリファレンス』の第7章「LOB と BFILE の操作」の「BLOB と CLOB の操作」の「BLOB または CLOB 列の作成と移入」、および『Oracle8i SQLJ 開発者ガイドおよびリファレンス』の第5章「型のサポート」の「Oracle の拡張型」の「BLOB、CLOB および BFILE のサポート」

使用例

次の例では、BFILE データをオープンして表示します。

例

次のプログラム環境の例が示されています。

- [PL/SQL: BFILE データの表示](#) (11-87 ページ)
- [C \(OCI\) : BFILE データの表示](#) (11-88 ページ)
- [COBOL \(Pro*COBOL\) : BFILE データの表示](#) (11-91 ページ)

- [C/C++ \(Pro*C/C++\) : BFILE データの表示 \(11-93 ページ\)](#)
- [Visual Basic \(OO4O\) : BFILE データの表示 \(11-94 ページ\)](#)
- [Java \(JDBC\) : BFILE データの表示 \(11-95 ページ\)](#)

PL/SQL: BFILE データの表示

```
/* 例のプロシージャ displayBFILE_proc は、DBMS_LOB パッケージの
一部ではないことに注意してください。*/
CREATE OR REPLACE PROCEDURE displayBFILE_proc IS
  Lob_loc  BFILE;
  Buffer    RAW(1024);
  Amount   BINARY_INTEGER := 1024;
  Position INTEGER        := 1;
BEGIN
  /* LOB を選択します。*/
  SELECT Music INTO Lob_loc
  FROM Multimedia_tab WHERE Clip_ID = 1;
  /* BFILE をオープンします。*/
  DBMS_LOB.OPEN (Lob_loc, DBMS_LOB.LOB_READONLY);
  LOOP
    DBMS_LOB.READ (Lob_loc, Amount, Position, Buffer);
    /* バッファの内容を表示します。*/
    DBMS_OUTPUT.PUT_LINE(utl_raw.cast_to_varchar2(Buffer));
    Position := Position + Amount;
  END LOOP;
  /* BFILE をクローズします。*/
  DBMS_LOB.CLOSE (Lob_loc);
  EXCEPTION
  WHEN NO_DATA_FOUND THEN
    DBMS_OUTPUT.PUT_LINE('End of data');
END;
```

C (OCI) : BFILE データの表示

```
/* マルチメディア表から LOB/BFILE を選択します。*/
void selectLob(Lob_loc, errhp, svchp, stmhpt)
OCILobLocator *Lob_loc;
OCIError      *errhp;
OCISvcCtx     *svchp;
OCISntmt      *stmhpt;
```

```

{
    OCIDefine *dfnhp;
    text *selstmt =
        (text *) "SELECT Music FROM Multimedia_tab WHERE Clip_ID=3";

    /* SQL の SELECT 文を準備します。*/
    checkerr (errhp, OCIStmtPrepare(stmthp, errhp, selstmt,
                                    (ub4) strlen((char *) selstmt),
                                    (ub4) OCI_NTV_SYNTAX, (ub4)OCI_DEFAULT));

    /* BFILE 列に対する定義をコールします。*/
    checkerr (errhp, OCIDefineByPos(stmthp, &dfnhp, errhp, 1,
                                    (dvoid *)&lob_loc, 0, SQLT_BFILE,
                                    (dvoid *)0, (ub2 *)0, (ub2 *)0,
                                    OCI_DEFAULT));

    /* SQL の SELECT 文を実行します。*/
    checkerr (errhp, OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                                    (CONST OCISnapshot*) 0, (OCISnapshot*) 0,
                                    (ub4) OCI_DEFAULT));
}

#define MAXBUFLen 32767

void BfileDisplay(envhp, errhp, svchp, stmthp)
OCIEnv      *envhp;
OCIError    *errhp;
OCISvcCtx   *svchp;
OCIStmt     *stmthp;
{
    /* 入力としてこのルーチンに渡されたすべてのハンドルは、割当ておよび初期化済で
       あると想定します。*/
    OCILobLocator *bfile_loc;
    ub1 bufp [MAXBUFLen];
    ub4 buflen, amt, offset;
    boolean done;
    ub4 retval;

    /* ロケータ記述子を割り当てます。*/
    (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &bfile_loc,
                              (ub4) OCI_DTYPE_FILE,
                              (size_t) 0, (dvoid **) 0);

    /* BFILE を選択します。*/
    selectLob(bfile_loc, errhp, svchp, stmthp);
}

```

```
checkerr(errhp, OCILobFileOpen(svchp, errhp, bfile_loc,
                               OCI_FILE_READONLY));
/* この例では、標準のポーリング方法を使用して、BFILE の内容全体がピース単位で
   バッファに読み込まれます。このポーリング方法では、各バッファのピースが、
   BFILE 全体が読み込まれるまで、すべての READ 操作後に処理されます。*/
/* amt = 0 (ゼロ) に設定すると、LOB の終わりまで読み込まれます。*/
amt = 0;
buflen = sizeof(bufp);
/* データをピースごとに処理します。*/
offset = 1;
memset(bufp, '\\0', MAXBUFLLEN);
done = FALSE;
while (!done)
{
    retval = OCILobRead(svchp, errhp, bfile_loc,
                       &amt, offset, (dvoid *) bufp,
                       buflen, (dvoid *) 0,
                       (sb4 *) (dvoid *, dvoid *, ub4, ub1)) 0,
                       (ub2) 0, (ub1) SQLCS_IMPLICIT);

    switch (retval)
    {
        case OCI_SUCCESS:          /* 1つのピースか、または最後のピースのみです。*/
            /* bufp のデータを処理します。amt には、bufp に直前に読み込まれたデータ
               の量が示されます。これは、BLOB ではバイト、固定幅 CLOB では文字、および
               可変幅 CLOB ではバイト単位です。*/
            done = TRUE;
            break;
        case OCI_ERROR:
            /* report_error();          ここでは示されない関数です。*/
            done = TRUE;
            break;
        case OCI_NEED_DATA:
            /* 2つ以上のピースがあります。*/
            /* bufp のデータを処理します。amt には、bufp に直前に読み込まれたデータ
               の量が示されます。これは、BFILE ではバイト、固定幅 CLOB では文字、および
               可変幅 CLOB ではバイト単位です。*/
            break;
        default:
            (void) printf("Unexpected ERROR: OCILobRead() LOB.\n");
            done = TRUE;
            break;
    } /* 切り替えます。*/
} /* 待機します。*/
```

```

/* BFILE をオープンしている場合、その BFILE をクローズする必要があります。*/
checkerr (errhp, OCIlobFileClose(svchp, errhp, bfile_loc));

/* ロケータ記述子を解放します。*/
OCIDescriptorFree((dvoid *)bfile_loc, (ub4)OCI_DTYPE_FILE);
}

```

COBOL (Pro*COBOL) : BFILE データの表示

```

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-AMT PIC 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.

* ORACLE に接続します。
   EXEC SQL WHENEVER SQLERROR DO PERFORM SQL-ERROR END-EXEC.
   EXEC SQL
       CONNECT :USERID
   END-EXEC.

* BFILE ロケータの割当ておよび初期化を行います。
   EXEC SQL ALLOCATE :SRC-BFILE END-EXEC.

* BFILE を選択します。
   EXEC SQL SELECT PHOTO INTO :SRC-BFILE
       FROM MULTIMEDIA_TAB WHERE CLIP_ID = 3 END-EXEC.

```

```
* BFILE をオープンします。
EXEC SQL LOB OPEN :SRC-BFILE READ ONLY END-EXEC.

* Amount = 0 (ゼロ) に設定すると、ポーリング方法が開始します。
MOVE 0 TO AMT;
EXEC SQL LOB READ :AMT FROM :SRC-BFILE INTO :BUFFER END-EXEC.

* "BFILE DATA" を表示します。
* AMT を D-AMT に移動させます。
* "First READ (", D-AMT, ")": " バッファを表示します。

* BFILE 全体が読み込まれるまで READ-LOOP を行います。
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.

* AMT を D-AMT に移動させます。
* "Next READ (", D-AMT, ")": " バッファを表示します。

GO TO READ-LOOP.

END-LOOP.
EXEC SQL WHENEVER NOT FOUND CONTINUE END-EXEC.

* LOB をクローズします。
EXEC SQL LOB CLOSE :SRC-BFILE END-EXEC.

* LOB ロケータを解放します。
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++) : BFILE データの表示

/* この例では、標準のポーリング方法を介したストリーミング・メカニズムを使用して、BFILE の内容全体がピース単位でバッファに読み込まれます。このポーリング方法では、各バッファのピースが、BFILE 全体が読み込まれるまで、すべての READ 操作後に表示されます。*/

```
#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;
    /* このデータ型では、データ型を等しくする必要があります。*/
    EXEC SQL VAR Buffer is VARRAW(BufferLength);

    EXEC SQL WHENEVER SQLERROR DO Sample_Error();
    EXEC SQL ALLOCATE :Lob_loc;
    /* BFILE を選択します。*/
    EXEC SQL SELECT Music INTO :Lob_loc
        FROM Multimedia_tab WHERE Clip_ID = 3;
    /* BFILE をオープンします。*/
    EXEC SQL LOB OPEN :Lob_loc READ ONLY;
    /* Amount = 0 (ゼロ) に設定すると、ポーリング方法が開始します。*/
    Amount = 0;
    /* バッファの最大サイズを設定します。*/
    Buffer.Length = BufferLength;
    EXEC SQL WHENEVER NOT FOUND DO break;
    while (TRUE)
    {
```

```
        /* BFILE のピースをバッファに読み込みます。*/
        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 (0040) : BFILE データの表示

```
Dim MySession As OraSession
Dim OraDb As OraDatabase

Dim OraDyn As OraDynaset, OraMusic As OraBfile, amount_read%, chunksize%, chunk As
Variant

Set MySession = CreateObject("OracleInProcServer.XOraSession")
Set OraDb = MySession.OpenDatabase("exampledb", "scott/tiger", 0&)

chunksize = 32767
Set OraDyn = OraDb.CreateDynaset("select * from Multimedia_tab", ORADYN_DEFAULT)
Set OraMusic = OraDyn.Fields("Music").Value

OraMusic.offset = 1
OraMusic.PollingAmount = OraMusic.Size 'Read entire BFILE contents

' BFILE を読みみ用にオープンします。
OraMusic.Open
amount_read = OraMusic.Read(chunk, chunksize)

While OraMusic.Status = ORALOB_NEED_DATA
    amount_read = OraMusic.Read(chunk, chunksize)
Wend
OraMusic.Close
```

Java (JDBC) : BFILE データの表示

```
import java.io.OutputStream;
// コア JDBC クラス :
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 固有の JDBC クラス :
import oracle.sql.*;
import oracle.jdbc.driver.*;

public class Ex4_53
{

    public static void main (String args [])
        throws Exception
    {
        // Oracle JDBC Driver をロードします。
        DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

        // データベースに接続します。
        Connection conn =
            DriverManager.getConnection ("jdbc:oracle:oci8:@", "samp", "samp");

        conn.setAutoCommit (false);

        // 文を作成します。
        Statement stmt = conn.createStatement ();

        try
        {
            BFILE src_lob = null;
            ResultSet rset = null;
            Boolean result = null;
            InputStream in = null;
            byte buf[] = new byte[1000];
            int length = 0;
            boolean alreadyDisplayed = false;
            rset = stmt.executeQuery (
                "SELECT music FROM multimedia_tab WHERE clip_id = 2");
            if (rset.next())
```

```
{
    src_lob = ((OracleResultSet)rset).getBFILE (1);
}

// BFILE をオープンします。
src_lob.openFile();

// ハンドルを取得し、BFILE からデータをストリームの形で流します。
in = src_lob.getBinaryStream();

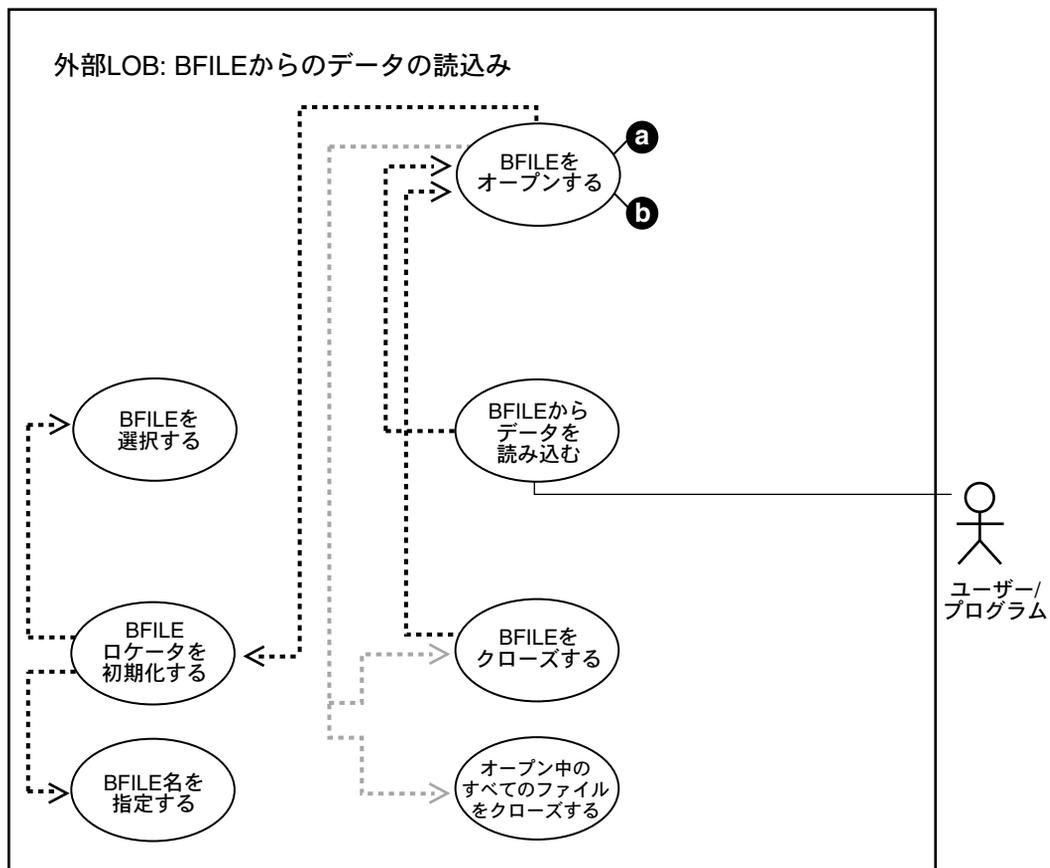
// このループによって buf が繰り返し充填され、InputStream から
// データが取り出されます。
while ((in != null) && ((length = in.read(buf)) != -1))
{
    // データは buf に読み込まれています。

    // この例では、最初の CHUNK のみを表示します。
    if (! alreadyDisplayed)
    {
        System.out.println("Bytes read in: " + Integer.toString(length));
        System.out.println(new String(buf));
        alreadyDisplayed = true;
    }
}

// ストリーム、BFILE、文および接続をクローズします。
in.close();
src_lob.closeFile();
stmt.close();
conn.commit();
conn.close();
}
catch (SQLException e)
{
    e.printStackTrace();
}
}
```

BFILE からのデータの読み込み

図 11-19 ユースケース図：BFILE からのデータの読み込み



参照： 外部LOB (BFILE) に関するすべての基本操作については、11-2ページの「ユースケース・モデル:外部LOB (BFILE)」を参照してください。

用途

BFILE からデータを読み込みます。

使用上の注意

LOB サイズにかかわらず通常 4GB -1 を指定

LOB 値を読み込むとき、LOB の最後を超えて読み込んでもエラーにはなりません。開始オフセットおよび LOB のデータ量にかかわらず、通常 4GB -1 の入力を指定できます。したがって、読み込む量を決定するために、OCILOBGetLength() コールでサーバーへのラウンドトリップを行い、LOB 値の長さを判断する必要はありません。

例

たとえば、LOB の長さが 5,000 バイトで、オフセット 1,000 から開始して LOB 値全体を読み込むとします。また、LOB 値の現在の長さがわからないとします。この場合の OCI 読み込みコールを示します (パラメータの初期化はすべて省略してあります)。

```
#define MAX_LOB_SIZE 4294967295
ub4 amount = MAX_LOB_SIZE;
ub4 offset = 1000;
OCILOBRead(svchp, errhp, locp, &amount, offset, bufp, buf1, 0, 0, 0, 0)
```

注意： 大量の LOB データを最も効率よく読み込む方法は、OCILOBRead() を使ったポーリングやコールバックによるストリーミングです。9-40 ページの「LOB への BFILE データのロード」の「使用上の注意」を参照してください。

量パラメータ

- **DBMS_LOB.READ** では、量パラメータはデータのサイズより大きくできます。PL/SQL では、量パラメータはバッファのサイズ以下である必要があり、バッファ・サイズは 32KB に制限されます。
- **OCILOBRead** では、量パラメータの値は 4GB -1 に指定し、LOB の最後まで読み込むことができます。

構文

各プログラム環境で使用可能な機能のリストは、第 3 章「LOB プログラム環境」を参照してください。各プログラム環境における構文については、次のマニュアルの項を参照してください。

- PL/SQL (DBMS_LOB) : 『Oracle8i PL/SQL パッケージ・プロシージャ・リファレンス』の第 18 章「DBMS_LOB」の「サブプログラムの要約」の READ プロシージャ
- C (OCI) : 『Oracle8i コール・インタフェース・プログラマーズ・ガイド』の第 7 章「LOB および FILE 操作」の使用上の注意、および第 15 章「OCI リレーショナル関数」の「LOB 関数」の OCILobRead()
- COBOL (Pro*COBOL) : 『Oracle8i Pro*COBOL プリコンパイラ・プログラマーズ・ガイド』の第 13 章「ラージ・オブジェクト (LOB)」の「LOB 文」の使用上の注意、および付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB READ (実行可能埋込み SQL 拡張要素)」
- C/C++ (Pro*C/C++) : 『Oracle8i Pro*C/C++ プリコンパイラ・プログラマーズ・ガイド』の第 16 章「ラージ・オブジェクト (LOB)」の「LOB 文」の使用上の注意、および付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB READ (実行可能埋込み SQL 拡張要素)」
- Visual Basic (Oracle Objects for OLE オンライン・ヘルプ): 「Help」の「Contents」タブから、「OO4O Automation Server」>「OBJECTS」>「OraBfile」>「METHODS」>「Read」、および「OBJECTS」>「OraBfile」>「PROPERTIES」>「PollingAmount」、「Offset」、「Status」を選択してください。または「OBJECTS」>「OraBfile」>「Examples」を選択してください。
- Java (JDBC) : 『Oracle8i JDBC 開発者ガイドおよびリファレンス』の第 7 章「LOB と BFILE の操作」の「BLOB と CLOB の操作」の「BLOB または CLOB 列の作成と移入」、および『Oracle8i SQLJ 開発者ガイドおよびリファレンス』の第 5 章「型のサポート」の「Oracle の拡張型」の「BLOB、CLOB および BFILE のサポート」

使用例

次の例では、BFILE 「PHOTO_DIR」から PHOTO に写真を読み込みます。

例

次のプログラム環境の例が示されています。

- [PL/SQL \(DBMS_LOB パッケージ\) : BFILE からのデータの読み込み](#) (11-100 ページ)
- [C \(OCI\) : BFILE からのデータの読み込み](#) (11-100 ページ)
- [COBOL \(Pro*COBOL\) : BFILE からのデータの読み込み](#) (11-102 ページ)
- [C/C++ \(Pro*C/C++\) : BFILE からのデータの読み込み](#) (11-103 ページ)
- [Visual Basic \(OO4O\) : BFILE からのデータの読み込み](#) (11-104 ページ)
- [Java \(JDBC\) : BFILE からのデータの読み込み](#) (11-105 ページ)

PL/SQL (DBMS_LOB パッケージ) : BFILE からのデータの読み込み

```
/* 例のプロシージャ readBFILE_proc は、DBMS_LOB パッケージの
一部ではないことに注意してください。*/
CREATE OR REPLACE PROCEDURE readBFILE_proc IS
  Lob_loc      BFILE := BFILENAME('PHOTO_DIR', 'Jefferson_photo');
  Amount       INTEGER := 32767;
  Position     INTEGER := 1;
  Buffer        RAW(32767);
BEGIN
  /* LOB を選択します。*/
  SELECT Photo INTO Lob_loc FROM Multimedia_tab
     WHERE Clip_ID = 3;
  /* BFILE をオープンします。*/
  DBMS_LOB.OPEN(Lob_loc, DBMS_LOB.LOB_READONLY);
  /* データを読み込みます。*/
  DBMS_LOB.READ(Lob_loc, Amount, Position, Buffer);
  /* BFILE をクローズします。*/
  DBMS_LOB.CLOSE(Lob_loc);
END;
```

C (OCI) : BFILE からのデータの読み込み

```
/* マルチメディア表から LOB/BFILE を選択します。*/
void selectLob(Lob_loc, errhp, svchp, stmthp)
OCILobLocator *Lob_loc;
OCIError      *errhp;
OCISvcCtx     *svchp;
OCIStmt       *stmthp;
{
  OCIDefine *dfnhp;
  text *selstmt =
    (text *) "SELECT Photo FROM Multimedia_tab WHERE Clip_ID=3";

  /* SQL の SELECT 文を準備します。*/
  checkerr (errhp, OCIStmtPrepare(stmthp, errhp, selstmt,
    (ub4) strlen((char *) selstmt),
    (ub4) OCI_NTV_SYNTAX, (ub4) OCI_DEFAULT));

  /* BFILE 列に対する定義をコールします。*/
  checkerr (errhp, OCIDefineByPos(stmthp, &dfnhp, errhp, 1,
    (dvoid *)&Lob_loc, 0, SQLT_BFILE,
    (dvoid *)0, (ub2 *)0, (ub2 *)0,
    OCI_DEFAULT));
}
```

```
/* SQL の SELECT 文を実行します。*/
checkerr (errhp, OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                                (CONST OCISnapshot*) 0, (OCISnapshot*) 0,
                                (ub4) OCI_DEFAULT));
}

#define MAXBUFLen 32767

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

    OCILobLocator *bfile_loc;
    ub1 bufp [MAXBUFLen];
    ub4 buflen, amt, offset;
    ub4 retval;

    /* ロケータ記述子を割り当てます。*/
    (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &bfile_loc,
                              (ub4) OCI_DTYPE_FILE,
                              (size_t) 0, (dvoid **) 0);

    /* BFILE を選択します。*/
    selectLob(bfile_loc, errhp, svchp, stmthp);

    checkerr(errhp, OCILobFileOpen(svchp, errhp, bfile_loc,
                                   OCI_FILE_READONLY));

    /* この例では、標準のポーリング方法を使用して、BFILE の内容全体がピース単位で
       バッファに読み込まれます。このポーリング方法では、各バッファのピースが、
       BFILE 全体が読み込まれるまで、すべての READ 操作後に処理されます。*/
    /* amt = 0 (ゼロ) に設定すると、LOB の終わりまで読み込まれます。*/
    amt = 0;
    buflen = sizeof(bufp);
    /* データをピースごとに処理します。*/
    offset = 1;
    memset(bufp, '\0', MAXBUFLen);

    retval = OCILobRead(svchp, errhp, bfile_loc,
                        &amt, offset, (dvoid *) bufp,
                        buflen, (dvoid *) 0,
                        (sb4 *) (dvoid *, dvoid *, ub4, ub1)) 0,
```

```

        (ub2) 0, (ub1) SQLCS_IMPLICIT);

/* BFILE をオープンしている場合、その BFILE をクローズする必要があります。*/
checkerr (errhp, OCILobFileClose(svchp, errhp, bfile_loc));

/* ロケータ記述子を解放します。*/
OCIDescriptorFree((dvoid *)bfile_loc, (ub4)OCI_DTYPE_FILE);
}

```

COBOL (Pro*COBOL) : BFILE からのデータの読み込み

```

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.

* CLOB ロケータの割当ておよび初期化を行います。
EXEC SQL ALLOCATE :BFILE1 END-EXEC.
EXEC SQL WHENEVER NOT FOUND GOTO END-OF-BFILE END-EXEC.
EXEC SQL
    SELECT MUSIC INTO :BFILE1
    FROM MULTIMEDIA_TAB M WHERE M.CLIP_ID = 3
END-EXEC.

* BFILE をオープンします。
EXEC SQL LOB OPEN :BFILE1 READ ONLY END-EXEC.

* ポーリング読み込みを開始します。
MOVE 0 TO AMT.

EXEC SQL LOB READ :AMT FROM :BFILE1
    INTO :BUFFER2 END-EXEC.

*
*   データを表示します。

```

```
*  
* ロケータをクローズおよび解放します。  
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++) : BFILE からのデータの読み込み

```
#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()  
{  
OCIFFileLocator *Lob_loc;  
/* Amount と BufferLength が等しいため、必要な読み込みは 1 回のみです。*/  
int Amount = BufferLength;  
char Buffer[BufferLength];  
/* このデータ型では、データ型を等しくする必要があります。*/  
EXEC SQL VAR Buffer IS RAW(BufferLength);  
  
EXEC SQL WHENEVER SQLERROR DO Sample_Error();  
EXEC SQL ALLOCATE :Lob_loc;  
EXEC SQL SELECT Photo INTO :Lob_loc  
FROM Multimedia_tab WHERE Clip_ID = 3;  
/* BFILE をオープンします。*/  
EXEC SQL LOB OPEN :Lob_loc READ ONLY;  
EXEC SQL WHENEVER NOT FOUND CONTINUE;  
/* データを読み込みます。*/  
EXEC SQL LOB READ :Amount FROM :Lob_loc INTO :Buffer;
```

```
        printf("Read %d bytes\n", Amount);
        /* BFILE をクローズします。*/
        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 (0040) : BFILE からのデータの読み込み

```
Dim MySession As OraSession
Dim OraDb As OraDatabase

Dim OraDyn As OraDynaset, OraMusic As OraBfile, amount_read%, chunksize%, chunk As
Variant

Set MySession = CreateObject("OracleInProcServer.XOraSession")
Set OraDb = MySession.OpenDatabase("exampledb", "scott/tiger", 0&)

chunksize = 32767
Set OraDyn = OraDb.CreateDynaset("select * from Multimedia_tab", ORADYN_DEFAULT)
Set OraMusic = OraDyn.Fields("Music").Value

OraMusic.offset = 1
OraMusic.PollingAmount = OraMusic.Size 'Read entire BFILE contents

' BFILE を読み込み用にオープンします。
OraMusic.Open
amount_read = OraMusic.Read(chunk, chunksize)
While OraMusic.Status = ORALOB_NEED_DATA
    amount_read = OraMusic.Read(chunk, chunksize)
Wend
OraMusic.Close
```

Java (JDBC) : BFILE からのデータの読み込み

```
import java.io.InputStream;
import java.io.OutputStream;

// コア JDBC クラス :
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 固有の JDBC クラス :
import oracle.sql.*;
import oracle.jdbc.driver.*;

public class Ex4_53
{
    public static void main (String args [])
        throws Exception
    {
        // Oracle JDBC Driver をロードします。
        DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

        // データベースに接続します。
        Connection conn =
            DriverManager.getConnection ("jdbc:oracle:oci8:@", "samp", "samp");
        conn.setAutoCommit (false);

        // 文を作成します。
        Statement stmt = conn.createStatement ();
        try
        {
            BFILE src_lob = null;
            ResultSet rset = null;
            Boolean result = null;
            InputStream in = null;
            byte buf[] = new byte[1000];
            int length = 0;
            boolean alreadyDisplayed = false;
            rset = stmt.executeQuery (
                "SELECT music FROM multimedia_tab WHERE clip_id = 2");
            if (rset.next())
            {
```

```
        src_lob = ((OracleResultSet)rset).getBFILE (1);
    }

    // BFILE をオープンします。
    src_lob.openFile();

    // ハンドルを取得し、BFILE からデータをストリームの形で流します。
    in = src_lob.getBinaryStream();

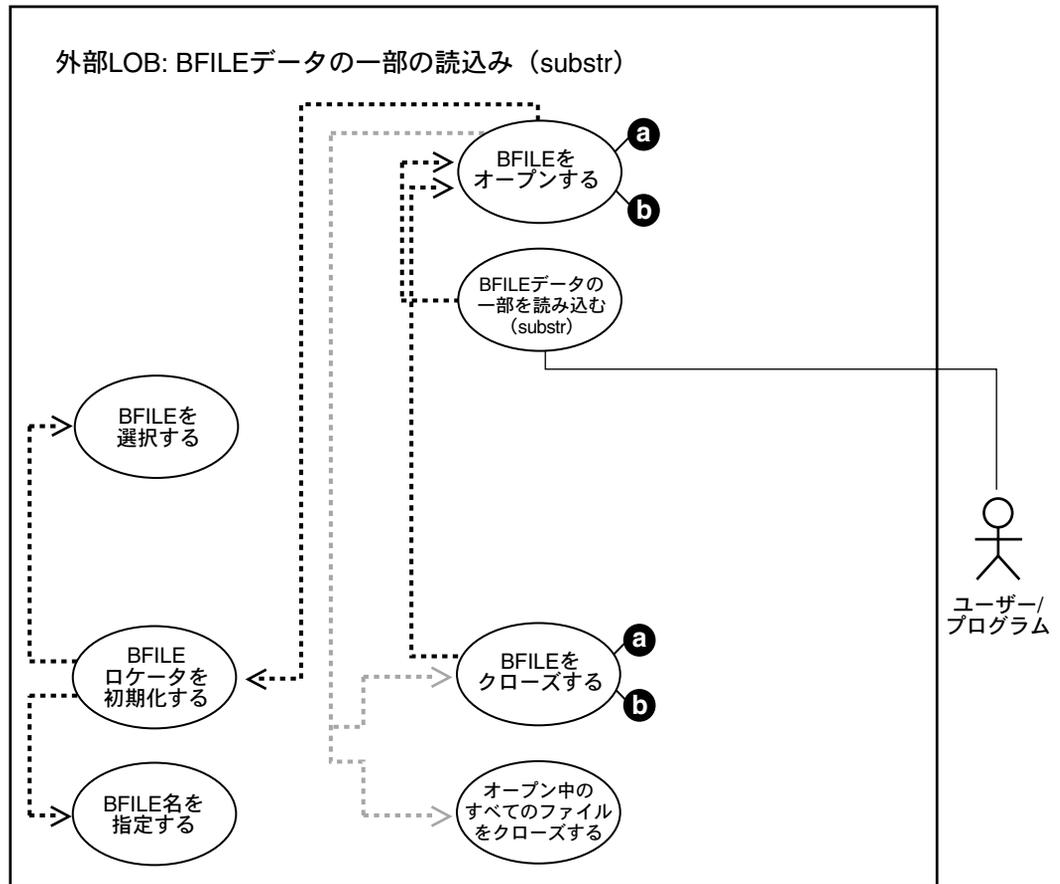
    // このループによって buf が繰り返し充填され、InputStream から
    // データが取り出されます。
    while ((in != null) && ((length = in.read(buf)) != -1))
    {
        // データは buf に読み込まれています。

        // この例では、最初の CHUNK のみを表示します。
        if (! alreadyDisplayed)
        {
            System.out.println("Bytes read in: " + Integer.toString(length));
            System.out.println(new String(buf));
            alreadyDisplayed = true;
        }
    }

    // ストリーム、BFILE、文および接続をクローズします。
    in.close();
    src_lob.closeFile();
    stmt.close();
    conn.commit();
    conn.close();
}
catch (SQLException e)
{
    e.printStackTrace();
}
}
```

BFILE データの一部の読み込み (substr)

図 11-20 ユースケース図：BFILE データの一部の読み込み (substr)



参照： 外部LOB (BFILE) に関するすべての基本操作については、11-2ページの「ユースケース・モデル:外部LOB (BFILE)」を参照してください。

用途

BFILE データの一部を読み込みます (substr)。

使用上の注意

ありません。

構文

各プログラム環境で使用可能な機能のリストは、第3章「LOB プログラム環境」を参照してください。各プログラム環境における構文については、次のマニュアルの項を参照してください。

- PL/SQL (DBMS_LOB) : 『Oracle8i PL/SQL パッケージ・プロシージャ・リファレンス』の第18章「DBMS_LOB」の「サブプログラムの要約」の SUBSTR ファンクション
- C (OCI) : 参照マニュアルはありません。
- COBOL (Pro*COBOL) : 『Oracle8i Pro*COBOL プリコンパイラ・プログラマーズ・ガイド』の第13章「ラージ・オブジェクト (LOB)」の「LOB 文」の使用上の注意、付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB OPEN (実行可能埋込み SQL 拡張要素)」、「LOB CLOSE (実行可能埋込み SQL 拡張要素)」、および『Oracle8i PL/SQL パッケージ・プロシージャ・リファレンス』の第18章「DBMS_LOB」の「サブプログラムの要約」の SUBSTR ファンクション
- C/C++ (Pro*C/C++) : 『Oracle8i Pro*C/C++ プリコンパイラ・プログラマーズ・ガイド』の第16章「ラージ・オブジェクト (LOB)」の「LOB 文」の使用上の注意、付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB OPEN (実行可能埋込み SQL 拡張要素)」、および『Oracle8i PL/SQL パッケージ・プロシージャ・リファレンス』の第18章「DBMS_LOB」の「サブプログラムの要約」の SUBSTR ファンクション
- Visual Basic (Oracle Objects for OLE オンライン・ヘルプ): 「Help」の「Contents」タブから、「OO4O Automation Server」>「OBJECTS」>「OraBfile」>「METHODS」>「Open」、および「OBJECTS」>「OraBfile」>「PROPERTIES」>「PollingAmount」、「Offset」、「Status」を選択してください。または「OBJECTS」>「OraBfile」>「Examples」を選択してください。
- Java (JDBC) : 『Oracle8i JDBC 開発者ガイドおよびリファレンス』の第7章「LOB と BFILE の操作」の「BLOB と CLOB の操作」の「BLOB または CLOB 列の作成と移入」、および『Oracle8i SQLJ 開発者ガイドおよびリファレンス』の第5章「型のサポート」の「Oracle の拡張型」の「BLOB、CLOB および BFILE のサポート」

使用例

次の例では、BFILE 「AUDIO_DIR」から RECORDING に音声録音を読み込みます。

例

次のプログラム環境の例が示されています。

- PL/SQL (DBMS_LOB パッケージ) : BFILE データの一部の読み込み (substr) (11-108 ページ)
- C (OCI) : 今回のリリースでは例は記載されていません。
- COBOL (Pro*COBOL) : BFILE データの一部の読み込み (substr) (11-109 ページ)
- C/C++ (Pro*C/C++) : BFILE データの一部の読み込み (substr) (11-110 ページ)
- Visual Basic (OO4O) : BFILE データの一部の読み込み (substr) (11-111 ページ)
- Java (JDBC) : BFILE データの一部の読み込み (substr) (11-112 ページ)

PL/SQL (DBMS_LOB パッケージ) : BFILE データの一部の読み込み (substr)

```

/* 例のプロシージャ substringBFILE_proc は、DBMS_LOB パッケージの
一部ではないことに注意してください。*/
CREATE OR REPLACE PROCEDURE substringBFILE_proc IS
  Lob_loc          BFILE;
  Position         INTEGER := 1;
  Buffer           RAW(32767);
BEGIN
  /* LOB を選択します。*/
  SELECT Mtab.Voiced_ref.Recording INTO Lob_loc FROM Multimedia_tab Mtab
     WHERE Mtab.Clip_ID = 3;
  /* BFILE をオープンします。*/
  DBMS_LOB.OPEN(Lob_loc, DBMS_LOB.LOB_READONLY);
  Buffer := DBMS_LOB.SUBSTR(Lob_loc, 255, Position);
  /* BFILE をクローズします。*/
  DBMS_LOB.CLOSE(Lob_loc);
END;
```

COBOL (Pro*COBOL) : BFILE データの一部の読み込み (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.
```

BFILE データの一部の読み込み (substr)

```
EXEC SQL INCLUDE SQLCA END-EXEC.
EXEC SQL VAR BUFFER2 IS VARRAW(32767) END-EXEC.

PROCEDURE DIVISION.
BFILE-SUBSTR.

* CLOB ロケータの割当ておよび初期化を行います。
EXEC SQL ALLOCATE :BFILE1 END-EXEC.
EXEC SQL WHENEVER NOT FOUND GOTO END-OF-BFILE END-EXEC.
EXEC SQL
    SELECT MTAB.VOICED_REF.RECORDING INTO :BFILE1
    FROM MULTIMEDIA_TAB MTAB WHERE MTAB.CLIP_ID = 3
END-EXEC.

* BFILE を READ ONLY でオープンします。
EXEC SQL LOB OPEN :BFILE1 READ ONLY END-EXEC.

* PL/SQL を実行して、SUBSTR 機能を使用します。
MOVE 32767 TO AMT.
EXEC SQL EXECUTE
    BEGIN
        :BUFFER2 := DEMS_LOB.SUBSTR(:BFILE1, :AMT, :POS);
    END;
END-EXEC.

* ロケータをクローズおよび解放します。
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++) : BFILE データの一部の読み込み (substr)

```
/* Pro*C/C++ には、DEMS_LOB.SUBSTR() ファンクションに対する同等の埋込み SQL
フォームがありません。ただし、Pro*C/C++ は、この例に示されているように、
Pro*C/C++ プログラムに埋め込まれている無名 PL/SQL ブロックを使用して、PL/SQL と
相互作用できます。*/
#include <oci.h>
#include <stdio.h>
#include <sqlca.h>
void Sample_Error()
{
    EXEC SQL WHENEVER SQLERROR CONTINUE;
    printf("%.*s\n", sqlca.sqlerrm.sqlerrml, sqlca.sqlerrm.sqlerrmc);
}
```

```

EXEC SQL ROLLBACK WORK RELEASE;
exit(1);
}

#define BufferLength 256
void substringBFILE_proc()
{
    OCIBFileLocator *Lob_loc;
    int Position = 1;
    char Buffer[BufferLength];
    EXEC SQL VAR Buffer IS RAW(BufferLength);
    EXEC SQL WHENEVER SQLERROR DO Sample_Error();
    EXEC SQL ALLOCATE :Lob_loc;
    EXEC SQL SELECT Mtab.Voiced_ref.Recording INTO :Lob_loc
        FROM Multimedia_tab Mtab WHERE Mtab.Clip_ID = 3;
    /* BFILE をオープンします。*/
    EXEC SQL LOB OPEN :Lob_loc READ ONLY;
    /* 無名 PL/SQL ブロック内から SUBSTR() を起動します。*/
    EXEC SQL EXECUTE
        BEGIN
            :Buffer := DBMS_LOB.SUBSTR(:Lob_loc, 256, :Position);
        END;
    END-EXEC;
    /* BFILE をクローズします。*/
    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 (0040) : BFILE データの一部の読み込み (substr)

```

Dim MySession As OraSession
Dim OraDb As OraDatabase

Dim OraDyn As OraDynaset, OraMusic As OraBfile, amount_read%, chunksize%, chunk

Set MySession = CreateObject("OracleInProcServer.XOraSession")

```

BFILE データの一部の読み込み (substr)

```
Set OraDb = MySession.OpenDatabase("exampledb", "scott/tiger", 0&)  
  
chunk_size = 32767  
Set OraDyn = OraDb.CreateDynaset("select * from Multimedia_tab", ORADYN_DEFAULT)  
Set OraMusic = OraDyn.Fields("Music").Value  
OraMusic.PollingAmount = OraMusic.Size 'Read entire BFILE contents  
OraMusic.offset = 255 'Read from the 255th position  
'BFILEを読み込み用にオープンします。  
OraMusic.Open  
amount_read = OraMusic.Read(chunk, chunk_size) '戻されたチャンクは、可変の型バイト配列で  
す。  
If amount_read <> chunk_size Then  
    'エラー処理を行います。  
Else  
    'データを処理します。  
End If
```

Java (JDBC) : BFILE データの一部の読み込み (substr)

```
import java.io.OutputStream;  
// コア JDBC クラス :  
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 固有の JDBC クラス :  
import oracle.sql.*;  
import oracle.jdbc.driver.*;  
  
public class Ex4_62  
{  
  
    public static void main (String args [])  
        throws Exception  
    {  
        // Oracle JDBC Driver をロードします。  
        DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());  
  
        // データベースに接続します。  
        Connection conn =  
            DriverManager.getConnection ("jdbc:oracle:oci8:@", "samp", "samp");  
        conn.setAutoCommit (false);  
    }  
}
```

```
// 文を作成します。
Statement stmt = conn.createStatement ();
try
{
BFILE src_lob = null;
    ResultSet rset = null;
InputStream in = null;
byte buf[] = new byte[1000];
int length = 0;
rset = stmt.executeQuery (
    "SELECT music FROM multimedia_tab WHERE clip_id = 2");
if (rset.next())
{
    src_lob = ((OracleResultSet)rset).getBFILE (1);
}

// BFILE をオープンします。
src_lob.openFile();

// ハンドルを取得し、BFILE からデータをストリームの形で流します。
in = src_lob.getBinaryStream();

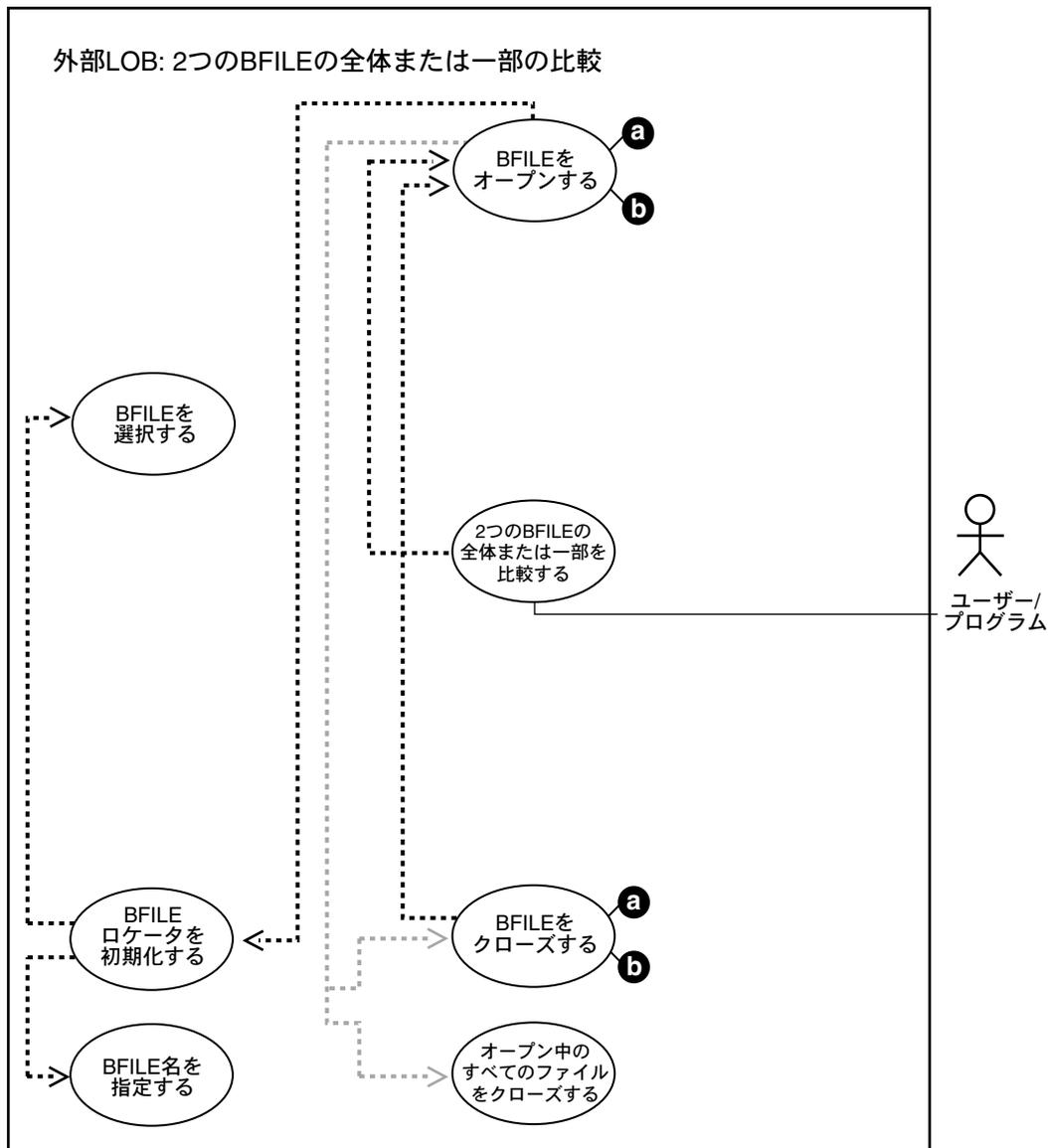
if (in != null)
{
    // オフセット 1 から開始して、buf に 255 バイトを要求します。
    // length = # バイトが、実際にストリームから戻されます。
    length = in.read(buf, 1, 255);
    System.out.println("Bytes read in: " + Integer.toString(length));

    // buf を処理します。
    System.out.println(new String(buf));
}

// ストリーム、BFILE、文および接続をクローズします。
in.close();
src_lob.closeFile();
stmt.close();
conn.commit();
conn.close();
}
catch (SQLException e)
{
    e.printStackTrace();
}
}
```

2つのBFILEの全体または一部の比較

図 11-21 ユースケース図：2つのBFILEの全体または一部の比較



参照: 外部 LOB (BFILE) に関するすべての基本操作については、11-2 ページの「ユースケース・モデル:外部 LOB (BFILE)」を参照してください。

用途

2つの BFILE の全体または一部を比較します。

使用上の注意

ありません。

構文

各プログラム環境で使用可能な機能のリストは、第3章「LOB プログラム環境」を参照してください。各プログラム環境における構文については、次のマニュアルの項を参照してください。

- PL/SQL (DBMS_LOB) : 『Oracle8i PL/SQL パッケージ・プロシージャ・リファレンス』の第18章「DBMS_LOB」の「サブプログラムの要約」の COMPARE ファンクション
- C (OCI) : 参照マニュアルはありません。
- COBOL (Pro*COBOL) : 『Oracle8i Pro*COBOL プリコンパイラ・プログラマーズ・ガイド』の第13章「ラージ・オブジェクト (LOB)」の「LOB 文」の使用上の注意、付録 F 「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB OPEN (実行可能埋込み SQL 拡張要素)」、および『Oracle8i PL/SQL パッケージ・プロシージャ・リファレンス』の第18章「DBMS_LOB」の「サブプログラムの要約」の COMPARE ファンクション
- C/C++ (Pro*C/C++) : 『Oracle8i Pro*C/C++ プリコンパイラ・プログラマーズ・ガイド』の第16章「ラージ・オブジェクト (LOB)」の「LOB 文」の使用上の注意、付録 F 「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB OPEN (実行可能埋込み SQL 拡張要素)」、および『Oracle8i PL/SQL パッケージ・プロシージャ・リファレンス』の第18章「DBMS_LOB」の「サブプログラムの要約」の COMPARE ファンクション
- Visual Basic (Oracle Objects for OLE オンライン・ヘルプ): 「Help」の「Contents」タブから、「OO4O Automation Server」>「OBJECTS」>「OraBfile」>「METHODS」>「Open」、「Compare」、および「OBJECTS」>「OraDatabase」>「PROPERTIES」>「Parameters」を選択してください。または「OBJECTS」>「OraBfile」>「Examples」を選択してください。
- Java (JDBC) : 『Oracle8i JDBC 開発者ガイドおよびリファレンス』の第7章「LOB と BFILE の操作」の「BLOB と CLOB の操作」の「BLOB または CLOB 列の作成と移入」、および『Oracle8i SQLJ 開発者ガイドおよびリファレンス』の第5章「型のサポート」の「Oracle の拡張型」の「BLOB、CLOB および BFILE のサポート」

使用例

次の例では、ファイル「PHOTO_DIR」内の写真が特定の PHOTO としてすでに使用されているかどうかを、それぞれのデータ・エンティティをビットごとに比較することで判断します。

注意： 比較を始める前に、それぞれの BFILE の長さを調べる必要がないように、LOBMAXSIZE は 4GB に設定されます。

例

次のプログラム環境の例が示されています。

- [PL/SQL \(DBMS_LOB パッケージ\) : 2 つの BFILE の全体または一部の比較 \(11-116 ページ\)](#)
- C (OCI) : 今回のリリースでは例は記載されていません。
- [COBOL \(Pro*COBOL\) : 2 つの BFILE の全体または一部の比較 \(11-117 ページ\)](#)
- [C/C++ \(Pro*C/C++\) : 2 つの BFILE の全体または一部の比較 \(11-119 ページ\)](#)
- [Visual Basic \(OO4O\) : 2 つの BFILE の全体または一部の比較 \(11-120 ページ\)](#)
- [Java \(JDBC\) : 2 つの BFILE の全体または一部の比較 \(11-121 ページ\)](#)

PL/SQL (DBMS_LOB パッケージ) : 2 つの BFILE の全体または一部の比較

```
/* 例のプロシージャ instrstringBFILE_proc は、DBMS_LOB パッケージの
一部ではないことに注意してください。*/
CREATE OR REPLACE PROCEDURE instrstringBFILE_proc IS
  Lob_loc      BFILE;
  Pattern      RAW(32767);
  Position     INTEGER;
BEGIN
  /* LOB を選択します。*/
  SELECT Intab.Recording INTO Lob_loc
    FROM THE(SELECT Mtab.InSeq_ntab FROM Multimedia_tab Mtab
             WHERE Clip_ID = 3) Intab
         WHERE Segment = 1;
  /* BFILE をオープンします。*/
  DBMS_LOB.OPEN(Lob_loc, DBMS_LOB.LOB_READONLY);
  /* 検索するパターンを初期化し、BFILE の始めから 2 番目に出現するパターンを検索します。*/
  Position := DBMS_LOB.INSTR(Lob_loc, Pattern, 1, 2);
  /* BFILE をクローズします。*/
  DBMS_LOB.CLOSE(Lob_loc);
END;
```

COBOL (Pro*COBOL) : 2 つの BFILE の全体または一部の比較

```

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.

* BLOB ロケータの割当ておよび初期化を行います。
EXEC SQL ALLOCATE :BFILE1 END-EXEC.
EXEC SQL ALLOCATE :BFILE2 END-EXEC.

* ディレクトリおよびファイル情報を設定します。
MOVE "PHOTO_DIR" TO DIR-ALIAS-ARR.
MOVE 9 TO DIR-ALIAS-LEN.
MOVE "fdroosevelt_photo" TO FNAME-ARR.
MOVE 17 TO FNAME-LEN.

EXEC SQL
LOB FILE SET :BFILE1 DIRECTORY = :DIR-ALIAS,
FILENAME = :FNAME
END-EXEC.

EXEC SQL WHENEVER NOT FOUND GOTO END-OF-BFILE END-EXEC.
EXEC SQL
SELECT PHOTO INTO :BFILE2
FROM MULTIMEDIA_TAB WHERE CLIP_ID = 3

```

```
END-EXEC.

* BLOB を READ ONLY でオープンします。
EXEC SQL LOB OPEN :BFILE1 READ ONLY END-EXEC.
EXEC SQL LOB OPEN :BFILE2 READ ONLY END-EXEC.

* PL/SQL を実行し、COMPARE 機能を取得します。
MOVE 5 TO AMT.
EXEC SQL EXECUTE
BEGIN
:RET := DBMS_LOB.COMPARE(:BFILE1, :BFILE2,
:AMT, 1, 1);
END;
END-EXEC.

IF RET = 0
*   同等の BFILE の論理がここに入ります。
   DISPLAY "BFILES are equal"
ELSE
*   同等でない BFILE の論理がここに入ります。
   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++) : 2 つの BFILE の全体または一部の比較

/* Pro*C/C++ には、DBMS_LOB.COMPARE() ファンクションに対する同等の埋込み SQL フォームがありません。ただし、DBMS_LOB.SUBSTR() と同様に、Pro*C/C++ は、この例に示されているように、無名 PL/SQL ブロック内で DBMS_LOB.COMPARE() を起動できます。*/

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

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

void compareBFILES_proc()
{
    OCIBFileLocator *Lob_loc1, *Lob_loc2;
    int Retval = 1;
    char *Dir1 = "PHOTO_DIR", *Name1 = "RooseveltFDR_photo";

    EXEC SQL WHENEVER SQLERROR DO Sample_Error();
    EXEC SQL ALLOCATE :Lob_loc1;
    EXEC SQL LOB FILE SET :Lob_loc1 DIRECTORY = :Dir1, FILENAME = :Name1;
    EXEC SQL ALLOCATE :Lob_loc2;
    EXEC SQL SELECT Photo INTO :Lob_loc2 FROM Multimedia_tab
        WHERE Clip_ID = 3;
    /* BFILE をオープンします。*/
    EXEC SQL LOB OPEN :Lob_loc1 READ ONLY;
    EXEC SQL LOB OPEN :Lob_loc2 READ ONLY;
    /* DBMS_LOB.COMPARE() を使用して、PL/SQL 内で BFILE を比較します。*/
    EXEC SQL EXECUTE
        BEGIN
            :Retval := DBMS_LOB.COMPARE(
                :Lob_loc2, :Lob_loc1, DBMS_LOB.LOBMAXSIZE, 1, 1);
        END;
    END-EXEC;
    /* BFILE をクローズします。*/
    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");
/* ロケータが使用しているリソースを解放します。*/
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 (0040) : 2つの BFILE の全体または一部の比較

ここで示す PL/SQL パッケージおよび表は、標準の 0040 インストールの一部ではないことに注意してください。

```
Dim MySession As OraSession
Dim OraDb As OraDatabase
Dim OraDyn As OraDynaset, OraMusic As OraBfile, OraMyMusic As OraBfile, OraSql As OraSqlStmt
```

```
Set MySession = CreateObject("OracleInProcServer.XOraSession")
Set OraDb = MySession.OpenDatabase("exampledb", "scott/tiger", 0&)
```

```
OraDb.Connection.BeginTrans
```

```
Set OraParameters = OraDb.Parameters
```

```
OraParameters.Add "id", 1001, ORAPARM_INPUT
```

‘BFILE 型の OUT パラメータを定義します。

```
OraParameters.Add "MyMusic", Null, ORAPARM_OUTPUT
OraParameters("MyMusic").ServerType = ORATYPE_BFILE
```

```
Set OraSql =
    OraDb.CreateSql(
        "BEGIN SELECT music INTO :MyMusic FROM multimedia_tab WHERE clip_id = :id;
        END;", ORASQL_FAILEXEC)
```

```
Set OraMyMusic = OraParameters("MyMusic").Value
```

‘ダイナセットを作成します。

```
Set OraDyn =
    OraDb.CreateDynaset(
```

```

        "SELECT * FROM Multimedia_tab WHERE Clip_Id = 1001", ORADYN_DEFAULT)
Set OraMusic = OraDyn.Fields("Music").Value

' BFILE を読み用にオープンします。
OraMusic.Open
OraMyMusic.Open

If OraMusic.Compare(OraMyMusic) Then
    ' データを処理します。
Else
    ' エラー処理を行います。
End If
OraDb.Connection.CommitTrans

```

Java (JDBC) : 2 つの BFILE の全体または一部の比較

```

import java.io.InputStream;
import java.io.OutputStream;

// コア JDBC クラス :
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 固有の JDBC クラス :
import oracle.sql.*;
import oracle.jdbc.driver.*;

public class Ex4_66
{
    static final int MAXBUFSIZE = 32767;

    public static void main (String args [])
        throws Exception
    {
        // Oracle JDBC Driver をロードします。
        DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

        // データベースに接続します。
        Connection conn =
            DriverManager.getConnection ("jdbc:oracle:oci8:@", "samp", "samp");
    }
}

```

```
// 自動コミットがオフの場合、高速になります。
conn.setAutoCommit (false);

// 文を作成します。
Statement stmt = conn.createStatement ();

try
{
    BFILE lob_loc1 = null;
    BFILE lob_loc2 = null;
    ResultSet rset = null;

    rset = stmt.executeQuery (
        "SELECT photo FROM multimedia_tab WHERE clip_id = 2");
    if (rset.next())
    {
        lob_loc1 = ((OracleResultSet)rset).getBFILE (1);
    }

    rset = stmt.executeQuery (
        "SELECT BFILENAME('PHOTO_DIR', 'music') FROM DUAL");
    if (rset.next())
    {
        lob_loc2 = ((OracleResultSet)rset).getBFILE (1);
    }

    lob_loc1.openFile ();
    lob_loc2.openFile ();

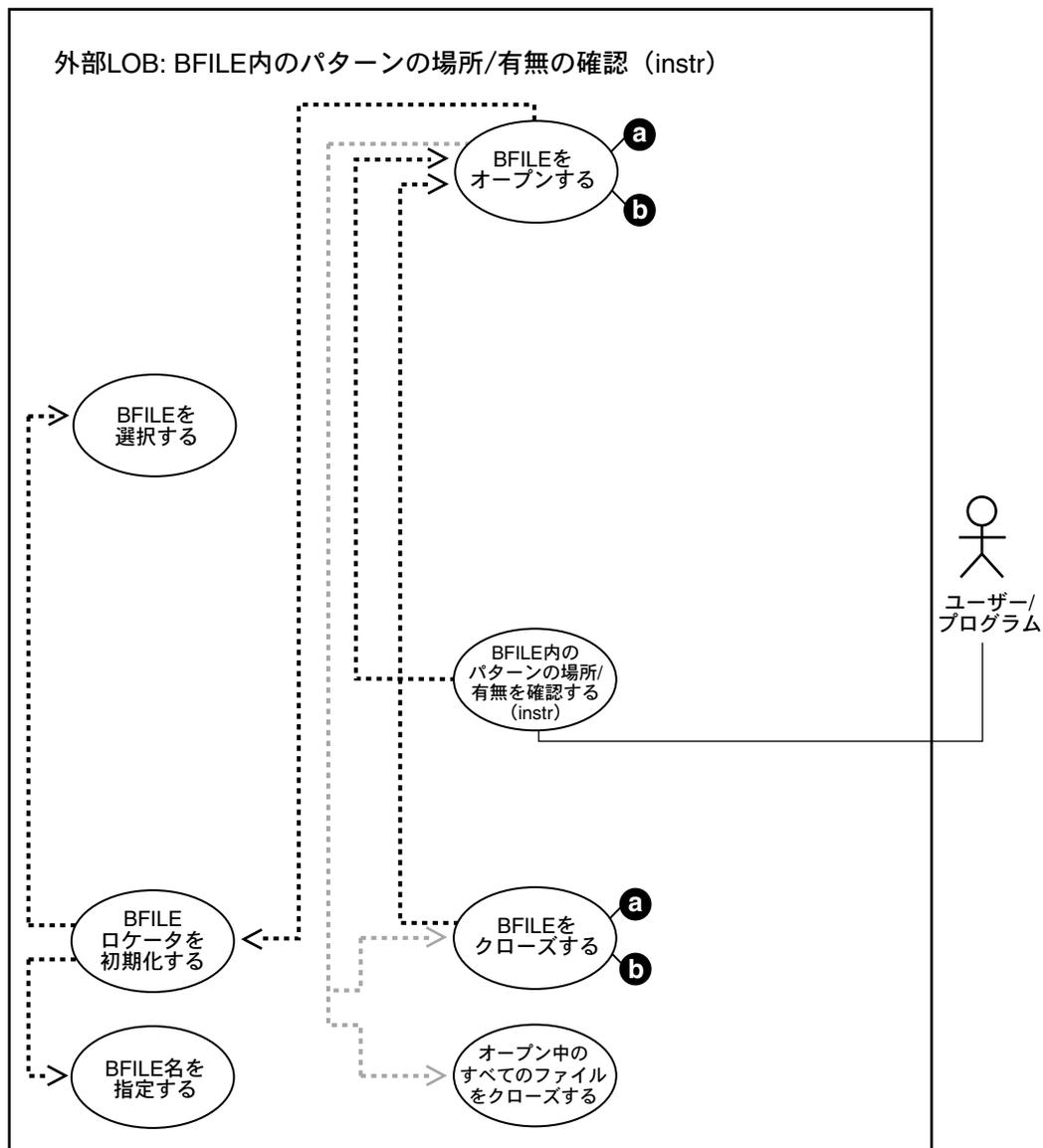
    if (lob_loc1.length() > lob_loc2.length())
        System.out.println("Looking for LOB2 inside LOB1. result = " +
            lob_loc1.position(lob_loc2, 1));
    else
        System.out.println("Looking for LOB1 inside LOB2. result = " +
            lob_loc2.position(lob_loc1, 1));

    stmt.close();
    conn.commit();
    conn.close();

}
catch (SQLException e)
{
    e.printStackTrace();
}
}
```

BFILE 内のパターンの有無の確認 (instr)

図 11-22 ユースケース図：BFILE 内のパターンの有無の確認 (instr)



参照： 外部 LOB (BFILE) に関するすべての基本操作については、11-2 ページの「[ユースケース・モデル:外部 LOB \(BFILE\)](#)」を参照してください。

用途

BFILE 内のパターンの有無を確認します (instr)。

使用上の注意

ありません。

構文

各プログラム環境で使用可能な機能のリストは、[第 3 章「LOB プログラム環境」](#)を参照してください。各プログラム環境における構文については、次のマニュアルの項を参照してください。

- PL/SQL (DBMS_LOB) : 『Oracle8i PL/SQL パッケージ・プロシージャ・リファレンス』の第 18 章「DBMS_LOB」の「サブプログラムの要約」の INSTR ファンクション
- C (OCI) : 参照マニュアルはありません。
- COBOL (Pro*COBOL) : 『Oracle8i Pro*COBOL プリコンパイラ・プログラマーズ・ガイド』の第 13 章「ラージ・オブジェクト (LOB)」の「LOB 文」の使用上の注意、付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB OPEN (実行可能埋込み SQL 拡張要素)」、および『Oracle8i PL/SQL パッケージ・プロシージャ・リファレンス』の第 18 章「DBMS_LOB」の「サブプログラムの要約」の INSTR ファンクション
- C/C++ (Pro*C/C++) : 『Oracle8i Pro*C/C++ プリコンパイラ・プログラマーズ・ガイド』の第 16 章「ラージ・オブジェクト (LOB)」の「LOB 文」の使用上の注意、および付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB OPEN (実行可能埋込み SQL 拡張要素)」、および『Oracle8i PL/SQL パッケージ・プロシージャ・リファレンス』の第 18 章「DBMS_LOB」の「サブプログラムの要約」の INSTR ファンクション
- Visual Basic (OO4O) : 参照マニュアルはありません。
- Java (JDBC) : 『Oracle8i JDBC 開発者ガイドおよびリファレンス』の第 7 章「LOB と BFILE の操作」の「BLOB と CLOB の操作」の「BLOB または CLOB 列の作成と移入」、および『Oracle8i SQLJ 開発者ガイドおよびリファレンス』の第 5 章「型のサポート」の「Oracle の拡張型」の「BLOB、CLOB および BFILE のサポート」

使用例

次の例では、インタビュー Recording 内でオーディオ・データのパターンを検索します。ここでは、オーディオ・シグネチャは識別可能なビット・パターンによって表されていると想定しています。

例

次のプログラム環境の例が示されています。

- [PL/SQL \(DBMS_LOB パッケージ\) : BFILE 内のパターンの有無の確認 \(instr\) \(11-125 ページ\)](#)
- C (OCI) : 今回のリリースでは例は記載されていません。
- [COBOL \(Pro*COBOL\) : BFILE 内のパターンの有無の確認 \(instr\) \(11-126 ページ\)](#)
- [C/C++ \(Pro*C/C++\) : BFILE 内のパターンの有無の確認 \(instr\) \(11-127 ページ\)](#)
- Visual Basic (OO4O) : 今回のリリースでは例は記載されていません。
- [Java \(JDBC\) : BFILE 内のパターンの有無の確認 \(instr\) \(11-129 ページ\)](#)

PL/SQL (DBMS_LOB パッケージ) : BFILE 内のパターンの有無の確認 (instr)

```

/* 例のプロシージャ compareBFILES_proc は、DBMS_LOB パッケージの
一部ではないことに注意してください。*/
CREATE OR REPLACE PROCEDURE compareBFILES_proc IS
  /* BFILE ロケータを初期化します。*/
  Lob_loc1      BFILE := BFILENAME('PHOTO_DIR', 'RooseveltFDR_photo');
  Lob_loc2      BFILE;
  Retval        INTEGER;
BEGIN
  /* LOB を選択します。*/
  SELECT Photo INTO Lob_loc2 FROM Multimedia_tab
     WHERE Clip_ID = 3;
  /* BFILE をオープンします。*/
  DBMS_LOB.OPEN(Lob_loc1, DBMS_LOB.LOB_READONLY);
  DBMS_LOB.OPEN(Lob_loc2, DBMS_LOB.LOB_READONLY);
  Retval := DBMS_LOB.COMPARE(Lob_loc2, Lob_loc1, DBMS_LOB.LOBMAXSIZE, 1, 1);
  /* BFILE をクローズします。*/
  DBMS_LOB.CLOSE(Lob_loc1);
  DBMS_LOB.CLOSE(Lob_loc2);
END;
```

COBOL (Pro*COBOL) : BFILE 内のパターンの有無の確認 (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.

* パターンの長さは任意に選択されています。
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.

* BFILE ロケータの割当ておよび初期化を行います。
    EXEC SQL ALLOCATE :BFILE1 END-EXEC.

    EXEC SQL WHENEVER NOT FOUND GOTO END-OF-BFILE END-EXEC.
    EXEC SQL
        SELECT PHOTO INTO :BFILE1
        FROM MULTIMEDIA_TAB WHERE CLIP_ID = 3 END-EXEC.

* CLOB を READ ONLY でオープンします。
    EXEC SQL LOB OPEN :BFILE1 READ ONLY END-EXEC.

* PL/SQL を実行して、INSTR 機能を取得します。
    EXEC SQL EXECUTE
        BEGIN
            :POS := DBMS_LOB.INSTR(:BFILE1, :PATTERN, 1, 2); END; END-EXEC.

    IF POS = 0
```

```

*          パターンが見つからない場合の論理はここにあります。
          DISPLAY "Pattern is not found."
ELSE
*          Pos には、パターンが見つかった位置が含まれます。
          DISPLAY "Pattern is found."
END-IF.

* LOB をクローズおよび解放します。
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++) : BFILE 内のパターンの有無の確認 (instr)

/* Pro*C/C++ には、DBMS_LOB.INSTR() ファンクションに対する同等の埋込み SQL フォームがありません。ただし、SUBSTR() および COMPARE() と同様に、Pro*C/C++ は、この例に示されているように、無名 PL/SQL ブロック内から DBMS_LOB.INSTR() をコールできます。*/

```

#include <sql2oci.h>
#include <stdio.h>
#include <string.h>
#include <sqlca.h>

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

#define PatternSize 5

```

```
void instringBFILE_proc()
{
    OCIBFileLocator *Lob_loc;
    unsigned int Position = 0;
    int Clip_ID = 3, Segment = 1;
    char Pattern[PatternSize];
    /* このデータ型では、データ型を等しくする必要があります。*/
    EXEC SQL VAR Pattern IS RAW(PatternSize);

    EXEC SQL WHENEVER SQLERROR DO Sample_Error();
    EXEC SQL ALLOCATE :Lob_loc;
    /* 動的 SQL を使用して、BFILE ロケータを取り出します。*/
    EXEC SQL PREPARE S FROM
        'SELECT Intab.Recording \
         FROM TABLE(SELECT Mtab.InSeg_ntab FROM Multimedia_tab Mtab \
          WHERE Clip_ID = :cid) Intab \
          WHERE Intab.Segment = :seg';
    EXEC SQL DECLARE C CURSOR FOR S;
    EXEC SQL OPEN C USING :Clip_ID, :Segment;
    EXEC SQL FETCH C INTO :Lob_loc;
    EXEC SQL CLOSE C;
    /* BFILE をオープンします。*/
    EXEC SQL LOB OPEN :Lob_loc READ ONLY;
    memset((void *)Pattern, 0, PatternSize);
    /* PL/SQL を使用して、BFILE の始めから最初に出現するパターンを検索します。*/
    EXEC SQL EXECUTE
        BEGIN
            :Position := DBMS_LOB.INSTR(:Lob_loc, :Pattern, 1, 1);
        END;
    END-EXEC;
    /* BFILE をクローズします。*/
    EXEC SQL LOB CLOSE :Lob_loc;
    if (0 == Position)
        printf("Pattern not found\n");
    else
        printf("The pattern occurs at %d\n", Position);
    EXEC SQL FREE :Lob_loc;
}

void main()
{
    char *samp = "samp/samp";
    EXEC SQL CONNECT :samp;
```

```
instr(BFILE_proc());  
EXEC SQL ROLLBACK WORK RELEASE;  
}
```

Java (JDBC) : BFILE 内のパターンの有無の確認 (instr)

```
import java.io.OutputStream;  
// コア JDBC クラス :  
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 固有の JDBC クラス :  
import oracle.sql.*;  
import oracle.jdbc.driver.*;  
  
public class Ex4_70  
{  
  
    static final int MAXBUFSIZE = 32767;  
  
    public static void main (String args [])  
        throws Exception  
    {  
        // Oracle JDBC Driver をロードします。  
        DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());  
  
        // データベースに接続します。  
        Connection conn =  
            DriverManager.getConnection ("jdbc:oracle:oci8:@", "samp", "samp");  
  
        // 自動コミットがオフの場合、高速になります。  
        conn.setAutoCommit (false);  
  
        // 文を作成します。  
        Statement stmt = conn.createStatement ();  
  
        try  
        {  
            BFILE lob_loc = null;
```

```
// BFILE 内で検索するパターンです。
String pattern = new String("children");

ResultSet rset = stmt.executeQuery (
    "SELECT photo FROM multimedia_tab WHERE clip_id = 3");
if (rset.next())
{
    lob_loc = ((OracleResultSet)rset).getBFILE (1);
}

// LOB をオープンします。
lob_loc.openFile();
// オフセット 1 から開始して、BFILE 内のパターン文字列の位置を検索します。
long result = lob_loc.position(pattern.getBytes(), 1);
System.out.println(
    "Results of Pattern Comparison : " + Long.toString(result));

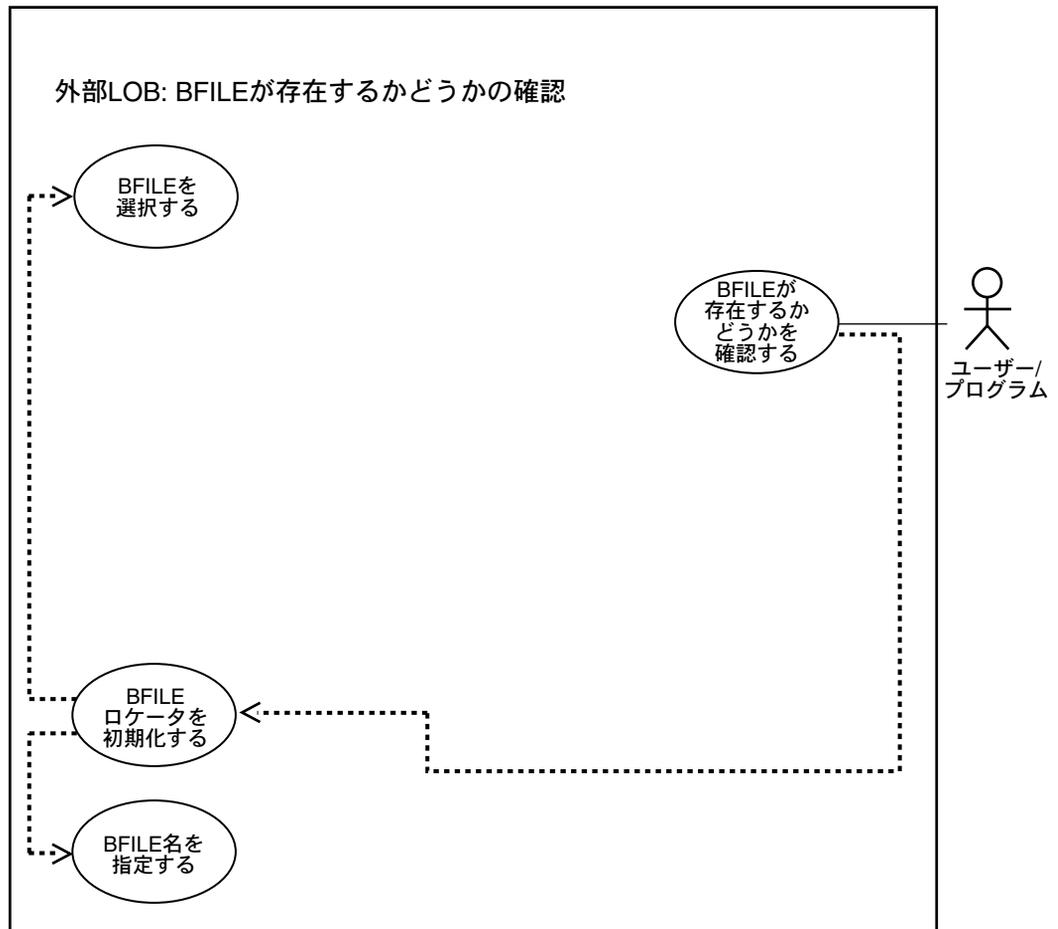
// LOB をクローズします。
lob_loc.closeFile();

stmt.close();
conn.commit();
conn.close();

}
catch (SQLException e)
{
    e.printStackTrace();
}
}
```

BFILE が存在するかどうかの確認

図 11-23 ユースケース図：BFILE が存在するかどうかの確認



参照： 外部LOB (BFILE) に関するすべての基本操作については、11-2ページの「ユースケース・モデル:外部LOB (BFILE)」を参照してください。

用途

BFILE が存在するかどうかを確認します。

使用上の注意

ありません。

構文

各プログラム環境で使用可能な機能のリストは、第3章「LOB プログラム環境」を参照してください。各プログラム環境における構文については、次のマニュアルの項を参照してください。

- PL/SQL (DBMS_LOB) : 『Oracle8i PL/SQL パッケージ・プロシージャ・リファレンス』の第18章「DBMS_LOB」の「サブプログラムの要約」の FILEEXISTS ファンクション
- C (OCI) : 『Oracle8i コール・インタフェース・プログラマーズ・ガイド』の第7章「LOB および FILE 操作」の使用上の注意、および第15章「OCI リレーショナル関数」の「LOB 関数」の OCILobFileExists()
- COBOL (Pro*COBOL) : 『Oracle8i Pro*COBOL プリコンパイラ・プログラマーズ・ガイド』の第13章「ラージ・オブジェクト (LOB)」の「LOB 文」の使用上の注意、および付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB DESCRIBE (実行可能埋込み SQL 拡張要素)」、および『Oracle8i PL/SQL パッケージ・プロシージャ・リファレンス』の第18章「DBMS_LOB」の「サブプログラムの要約」の FILEEXISTS ファンクション
- C/C++ (Pro*C/C++) : 『Oracle8i Pro*C/C++ プリコンパイラ・プログラマーズ・ガイド』の第16章「ラージ・オブジェクト (LOB)」の「LOB 文」の使用上の注意、および付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB DESCRIBE (実行可能埋込み SQL 拡張要素)」、および『Oracle8i PL/SQL パッケージ・プロシージャ・リファレンス』の第18章「DBMS_LOB」の「サブプログラムの要約」の FILEEXISTS ファンクション
- Visual Basic (Oracle Objects for OLE オンライン・ヘルプ): 「Help」の「Contents」タブから、「OO4O Automation Server」>「OBJECTS」>「OraBfile」>「PROPERTIES」>「Exists」、および「OBJECTS」>「OraDatabase」>「PROPERTIES」>「Parameters」を選択してください。または「OBJECTS」>「OraBfile」、「OraDatabase」>「Examples」を選択してください。
- Java (JDBC) : 『Oracle8i JDBC 開発者ガイドおよびリファレンス』の第7章「LOB と BFILE の操作」の「BLOB と CLOB の操作」の「BLOB または CLOB 列の作成と移入」、および『Oracle8i SQLJ 開発者ガイドおよびリファレンス』の第5章「型のサポート」の「Oracle の拡張型」の「BLOB、CLOB および BFILE のサポート」

使用例

次の例では、Recording に対応付けられた BFILE が存在するかどうかを問い合わせます。

例

次のプログラム環境の例が示されています。

- PL/SQL (DBMS_LOB パッケージ) : BFILE が存在するかどうかの確認 (11-133 ページ)
- C (OCI) : BFILE が存在するかどうかの確認 (11-134 ページ)
- COBOL (Pro*COBOL) : BFILE が存在するかどうかの確認 (11-135 ページ)
- C/C++ (Pro*C/C++) : BFILE が存在するかどうかの確認 (11-136 ページ)
- Visual Basic (OO4O) : BFILE が存在するかどうかの確認 (11-137 ページ)
- Java (JDBC) : BFILE が存在するかどうかの確認 (11-138 ページ)

PL/SQL (DBMS_LOB パッケージ) : BFILE が存在するかどうかの確認

```

/* 例のプロシージャ seeIfExistsBFILE_proc は、DBMS_LOB パッケージの
一部ではないことに注意してください。*/
CREATE OR REPLACE PROCEDURE seeIfExistsBFILE_proc IS
  Lob_loc      BFILE;
BEGIN
  /* LOB を選択します。*/
  SELECT Intab.Recording INTO Lob_loc
    FROM THE (SELECT Mtab.InSeg_ntab FROM Multimedia_tab Mtab
              WHERE Mtab.Clip_ID = 3) Intab
           WHERE Intab.Segment = 1;
  /* BFILE が存在するかどうかを確認します。*/
  IF (DBMS_LOB.FILEEXISTS(Lob_loc) != 0)
  THEN
    DBMS_OUTPUT.PUT_LINE('Processing given that the BFILE exists');
  ELSE
    DBMS_OUTPUT.PUT_LINE('Processing given that the BFILE does not exist');
  END IF;
EXCEPTION
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('Operation failed');
END;
```

C (OCI) : BFILE が存在するかどうかの確認

```

/* マルチメディア表から LOB/BFILE を選択します。*/
void selectLob(Lob_loc, errhp, svchp, stmthp)
OCIlobLocator *Lob_loc;
OCIError      *errhp;
OCISvcCtx     *svchp;
OCIStmt       *stmthp;
{
    OCIDefine *dfnhp;
    text *selstmt = (text *) "SELECT Photo FROM Multimedia_tab \
                               WHERE Clip_ID = 3";

    /* SQL の SELECT 文を準備します。*/
    checkerr (errhp, OCIStmtPrepare(stmthp, errhp, selstmt,
                                     (ub4) strlen((char *) selstmt),
                                     (ub4) OCI_NTV_SYNTAX, (ub4)OCI_DEFAULT));

    /* BFILE 列に対する定義をコールします。*/
    checkerr (errhp, OCIDefineByPos(stmthp, &dfnhp, errhp, 1,
                                     (dvoid *)&Lob_loc, 0, SQLT_BFILE,
                                     (dvoid *)0, (ub2 *)0, (ub2 *)0,
                                     OCI_DEFAULT));

    /* SQL の SELECT 文を実行します。*/
    checkerr (errhp, OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                                     (CONST OCISnapshot*) 0, (OCISnapshot*) 0,
                                     (ub4) OCI_DEFAULT));
}

void BfileExists(envhp, errhp, svchp, stmthp)
OCIEnv      *envhp;
OCIError    *errhp;
OCISvcCtx   *svchp;
OCIStmt     *stmthp;
{
    OCIlobLocator *bfile_loc;
    boolean is_exist;

    /* ロケータ記述子を割り当てます。*/
    (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &bfile_loc,
                              (ub4) OCI_DTYPE_FILE,
                              (size_t) 0, (dvoid **) 0);

    /* BFILE を選択します。*/
    selectLob(bfile_loc, errhp, svchp, stmthp);
}

```

```

checkerr (errhp, OCILobFileExists(svchp, errhp, bfile_loc, &is_exist));

if (is_exist == TRUE)
{
    printf("File exists\n");
}
else
{
    printf("File does not exist\n");
}

/* ロケータ記述子を解放します。*/
OCIDescriptorFree((dvoid *)bfile_loc, (ub4)OCI_DTYPE_FILE);
}

```

COBOL (Pro*COBOL) : BFILE が存在するかどうかの確認

```

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.

```

* BFILE ロケータの割当ておよび初期化を行います。

```

EXEC SQL ALLOCATE :BFILE1 END-EXEC.

EXEC SQL WHENEVER NOT FOUND GOTO END-OF-BFILE END-EXEC.
EXEC SQL
    SELECT PHOTO INTO :BFILE1

```

```
        FROM MULTIMEDIA_TAB WHERE CLIP_ID = 3
END-EXEC.

EXEC SQL
    LOB DESCRIBE :BFILE1 GET FILEEXISTS INTO :FEXISTS
END-EXEC.

IF FEXISTS = 1
*   ファイルが存在する場合の論理がここにあります。
    DISPLAY "File exists"
ELSE
*   ファイルが存在しない場合の論理がここにあります。
    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++) : BFILE が存在するかどうかの確認

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

void Sample_Error()
{
    EXEC SQL WHENEVER SQLERROR CONTINUE;
    printf("%.*s\n", sqlca.sqlerm.sqlerm1, sqlca.sqlerm.sqlermc);
    EXEC SQL ROLLBACK WORK RELEASE;
    exit(1);
}

void seeIfBFILEExists_proc()
```

```

{
  OCIBFileLocator *Lob_loc;
  unsigned int Exists = 0;

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  EXEC SQL ALLOCATE :Lob_loc;
  EXEC SQL SELECT Mtab.Voiced_ref.Recording INTO :Lob_loc
    FROM Multimedia_tab Mtab WHERE Mtab.Clip_ID = 3;
  /* BFILE が存在するかどうかを確認します。*/
  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 (0040) : BFILE が存在するかどうかの確認

ここで示す PL/SQL パッケージおよび表は、標準の 0040 インストールの一部では、ないことに注意してください。

```

Dim MySession As OraSession
Dim OraDb As OraDatabase
Dim OraMusic As OraBfile, OraSql As OraSqlStmt

Set MySession = CreateObject("OracleInProcServer.XOraSession")
Set OraDb = MySession.OpenDatabase("exampledb", "scott/tiger", 0&)

OraDb.Connection.BeginTrans

Set OraParameters = OraDb.Parameters

OraParameters.Add "id", 1001, ORAPARM_INPUT

' BFILE 型の OUT パラメータを定義します。
OraParameters.Add "MyMusic", Null, ORAPARM_OUTPUT
OraParameters("MyMusic").ServerType = ORATYPE_BFILE

Set OraSql =

```

```
OraDb.CreateSql (
    "BEGIN SELECT music INTO :MyMusic FROM multimedia_tab WHERE clip_id = :id;
    END;", ORASQL_FAILEXEC)

Set OraMusic = OraParameters("MyMusic").Value

If OraMusic.Exists Then
    'データを処理します。
Else
    'エラー処理を行います。
End If
OraDb.Connection.CommitTrans
```

Java (JDBC) : BFILE が存在するかどうかの確認

```
import java.io.InputStream;
import java.io.OutputStream;

// コア JDBC クラス :
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 固有の JDBC クラス :
import oracle.sql.*;
import oracle.jdbc.driver.*;

public class Ex4_74
{
    static final int MAXBUFSIZE = 32767;

    public static void main (String args [])
        throws Exception
    {
        // Oracle JDBC Driver をロードします。
        DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

        // データベースに接続します。
        Connection conn =
            DriverManager.getConnection ("jdbc:oracle:oci8:@", "samp", "samp");
```

```
// 自動コミットがオフの場合、高速になります。
conn.setAutoCommit (false);

// 文を作成します。
Statement stmt = conn.createStatement ();
try
{
    BFILE lob_loc = null;
    ResultSet rset = stmt.executeQuery (
        "SELECT photo FROM multimedia_tab WHERE clip_id = 3");
    if (rset.next ())
    {
        lob_loc = ((OracleResultSet)rset).getBFILE (1);
    }

    // BFILE が存在するかどうかを確認します。
    System.out.println("Result from fileExists(): " + lob_loc.fileExists());

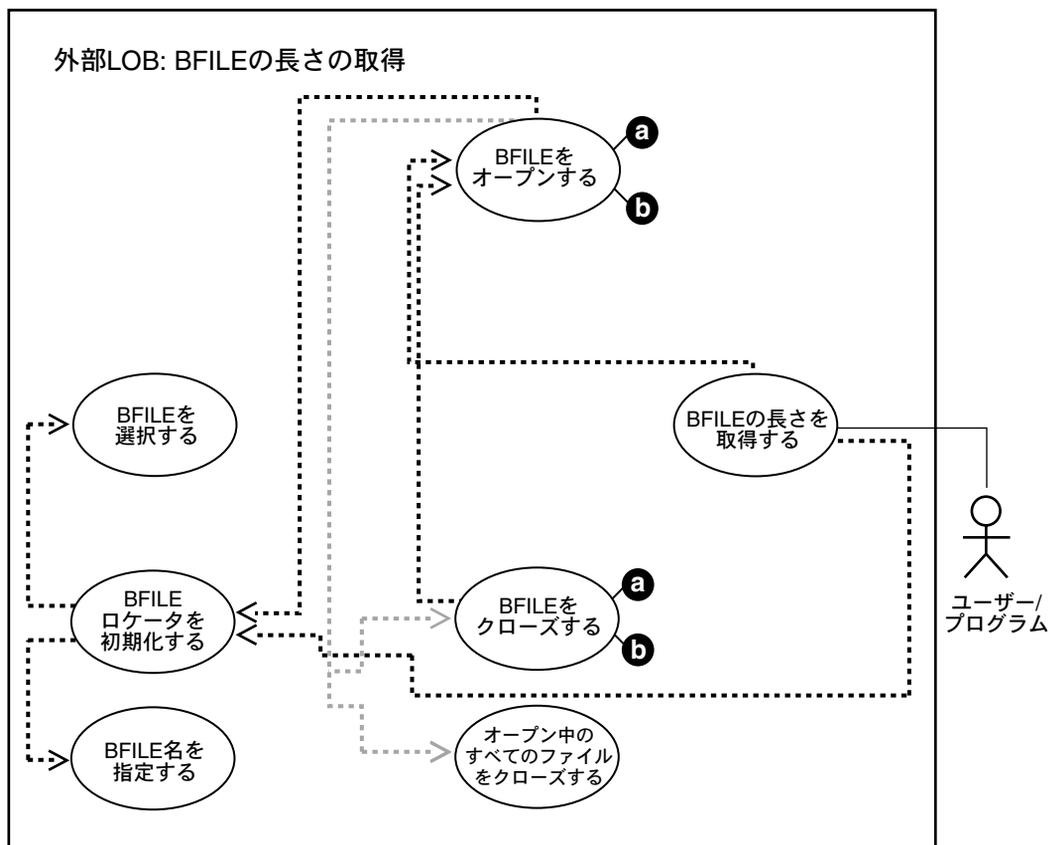
    // BFILE の長さを戻します。
    long length = lob_loc.length();
    System.out.println("Length of BFILE: " + length);

    // このBFILE のディレクトリ別名を取得します。
    System.out.println("Directory alias: " + lob_loc.getDirAlias());

    // このBFILE のファイル名を取得します。
    System.out.println("File name: " + lob_loc.getName());
    stmt.close();
    conn.commit();
    conn.close();
}
catch (SQLException e)
{
    e.printStackTrace();
}
}
```

BFILE の長さの取得

図 11-24 ユースケース図：BFILE の長さの取得



参照： 外部LOB (BFILE) に関するすべての基本操作については、11-2ページの「ユースケース・モデル: 外部LOB (BFILE)」を参照してください。

用途

BFILE の長さを取得します。

使用上の注意

ありません。

構文

各プログラム環境で使用可能な機能のリストは、第3章「LOB プログラム環境」を参照してください。各プログラム環境における構文については、次のマニュアルの項を参照してください。

- PL/SQL (DBMS_LOB) : 『Oracle8i PL/SQL パッケージ・プロシージャ・リファレンス』の第18章「DBMS_LOB」の「サブプログラムの要約」の GETLENGTH ファンクション
- C (OCI) : 『Oracle8i コール・インタフェース・プログラマーズ・ガイド』の第7章「LOB および FILE 操作」第15章「OCI リレーショナル関数」の「LOB 関数」の OCILobGetLength()
- COBOL (Pro*COBOL) : 『Oracle8i Pro*COBOL プリコンパイラ・プログラマーズ・ガイド』の第13章「ラージ・オブジェクト (LOB)」の「LOB 文」の使用上の注意、および付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB DESCRIBE (実行可能埋込み SQL 拡張要素)」、および『Oracle8i PL/SQL パッケージ・プロシージャ・リファレンス』の第18章「DBMS_LOB」の「サブプログラムの要約」の GETLENGTH ファンクション
- C/C++ (Pro*C/C++) : 『Oracle8i Pro*C/C++ プリコンパイラ・プログラマーズ・ガイド』の第16章「ラージ・オブジェクト (LOB)」の「LOB 文」の使用上の注意、および付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB DESCRIBE (実行可能埋込み SQL 拡張要素)」、および『Oracle8i PL/SQL パッケージ・プロシージャ・リファレンス』の第18章「DBMS_LOB」の「サブプログラムの要約」の GETLENGTH ファンクション
- Visual Basic (Oracle Objects for OLE オンライン・ヘルプ): 「Help」の「Contents」タブから、「OO4O Automation Server」>「OBJECTS」>「OraBfile」>「PROPERTIES」>「Size」、および「OBJECTS」>「OraBfile」>「Examples」を選択してください。
- Java (JDBC) : 『Oracle8i JDBC 開発者ガイドおよびリファレンス』の第7章「LOB と BFILE の操作」の「BLOB と CLOB の操作」の「BLOB または CLOB 列の作成と移入」、および『Oracle8i SQLJ 開発者ガイドおよびリファレンス』の第5章「型のサポート」の「Oracle の拡張型」の「BLOB、CLOB および BFILE のサポート」

使用例

次の例では、Recording に対応付けられた BFILE の長さを取得します。

例

次のプログラム環境の例が示されています。

- PL/SQL (DBMS_LOB パッケージ) : BFILE の長さの取得 (11-140 ページ)
- C (OCI) : BFILE の長さの取得 (11-143 ページ)
- COBOL (Pro*COBOL) : BFILE の長さの取得 (11-144 ページ)
- C/C++ (Pro*C/C++) : BFILE の長さの取得 (11-145 ページ)
- Visual Basic (OO4O) : BFILE の長さの取得 (11-146 ページ)
- Java (JDBC) : BFILE の長さの取得 (11-147 ページ)

PL/SQL (DBMS_LOB パッケージ) : BFILE の長さの取得

```
/* 例のプロシージャ getLengthBFILE_proc は、DBMS_LOB パッケージの
一部ではないことに注意してください。*/
CREATE OR REPLACE PROCEDURE getLengthBFILE_proc IS
  Lob_loc      BFILE;
  Length       INTEGER;
BEGIN
  /* LOB を選択して、BFILE ロケータを初期化します。*/
  SELECT Mtab.Voiced_ref.Recording INTO Lob_loc FROM Multimedia_tab Mtab
     WHERE Mtab.Clip_ID = 3;
  /* BFILE をオープンします。*/
  DBMS_LOB.OPEN(Lob_loc, DBMS_LOB.LOB_READONLY);
  /* LOB の長さを取得します。*/
  Length := DBMS_LOB.GETLENGTH(Lob_loc);
  IF Length IS NULL THEN
    DBMS_OUTPUT.PUT_LINE('BFILE is null. ');
  ELSE
    DBMS_OUTPUT.PUT_LINE('The length is ' || length);
  END IF;
  /* BFILE をクローズします。*/
  DBMS_LOB.CLOSE(Lob_loc);
END;
```

C (OCI) : BFILE の長さの取得

```

/* マルチメディア表から LOB/BFILE を選択します。*/
void selectLob(Lob_loc, errhp, svchp, stmthp)
OCILobLocator *Lob_loc;
OCIError      *errhp;
OCISvcCtx     *svchp;
OCIStmt      *stmthp;
{
    OCIDefine *dfnhp;
    text *selstmt = (text *) "SELECT Photo FROM Multimedia_tab \
                               WHERE Clip_ID = 3";

    /* SQL の SELECT 文を準備します。*/
    checkerr (errhp, OCIStmtPrepare(stmthp, errhp, selstmt,
                                     (ub4) strlen((char *) selstmt),
                                     (ub4) OCI_NTV_SYNTAX, (ub4)OCI_DEFAULT));

    /* BFILE 列に対する定義をコールします。*/
    checkerr (errhp, OCIDefineByPos(stmthp, &dfnhp, errhp, 1,
                                     (dvoid *)&Lob_loc, 0 , SQLT_BFILE,
                                     (dvoid *)0, (ub2 *)0, (ub2 *)0,
                                     OCI_DEFAULT));

    /* SQL の SELECT 文を実行します。*/
    checkerr (errhp, OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                                     (CONST OCISnapshot*) 0, (OCISnapshot*) 0,
                                     (ub4) OCI_DEFAULT));
}

void BfileLength(envhp, errhp, svchp, stmthp)
OCIEnv      *envhp;
OCIError    *errhp;
OCISvcCtx   *svchp;
OCIStmt     *stmthp;
{
    OCILobLocator *bfile_loc;
    ub4 len;

    /* ロケータ記述子を割り当てます。*/
    (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &bfile_loc,
                              (ub4) OCI_DTYPE_FILE,
                              (size_t) 0, (dvoid **) 0);
}

```

```
/* BFILE を選択します。*/
selectLob(bfile_loc, errhp, svchp, stmthp);

checkerr (errhp, OCILobFileOpen(svchp, errhp, bfile_loc,
                               (ub1) OCI_FILE_READONLY));

checkerr (errhp, OCILobGetLength(svchp, errhp, bfile_loc, &len));

printf("Length of bfile = %d\n", len);

checkerr (errhp, OCILobFileClose(svchp, errhp, bfile_loc));

/* ロケータ記述子を解放します。*/
OCIDescriptorFree((dvoid *)bfile_loc, (ub4)OCI_DTYPE_FILE);
}
```

COBOL (Pro*COBOL) : BFILE の長さの取得

```
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.

* BFILE ロケータの割当ておよび初期化を行います。
EXEC SQL ALLOCATE :BFILE1 END-EXEC.
EXEC SQL WHENEVER NOT FOUND GOTO END-OF-BFILE END-EXEC.
```

```

EXEC SQL
    SELECT PHOTO INTO :BFILE1
    FROM MULTIMEDIA_TAB WHERE CLIP_ID = 3
END-EXEC.

* LOB DESCRIBE を使用して、LOB の長さを取得します。
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++) : BFILE の長さの取得

```

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

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

void getLengthBFILE_proc()
{
    OCIBFileLocator *Lob_loc;

```

```
unsigned int Length = 0;

EXEC SQL WHENEVER SQLERROR DO Sample_Error();
EXEC SQL ALLOCATE :Lob_loc;
EXEC SQL SELECT Mtab.Voiced_ref.Recording INTO :Lob_loc
      FROM Multimedia_tab Mtab WHERE Mtab.Clip_ID = 3;
/* BFILE をオープンします。*/
EXEC SQL LOB OPEN :Lob_loc READ ONLY;
/* 長さを取得します。*/
EXEC SQL LOB DESCRIBE :Lob_loc GET LENGTH INTO :Length;
/* BFILE が NULL であるか、または初期化されていない場合、長さは未定義です。*/
printf("Length is %d bytes\n", Length);
/* BFILE をクローズします。*/
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 (0040) : BFILE の長さの取得

ここで示す PL/SQL パッケージおよび表は、標準の 0040 インストールの一部ではないことに注意してください。

```
Dim MySession As OraSession
Dim OraDb As OraDatabase

Set MySession = CreateObject("OracleInProcServer.XOraSession")
Set OraDb = MySession.OpenDatabase("exampledb", "scott/tiger", 0&)

OraDb.Connection.BeginTrans

Set OraParameters = OraDb.Parameters

OraParameters.Add "id", 1001, ORAPARM_INPUT

' BFILE 型の OUT パラメータを定義します。
OraParameters.Add "MyMusic", Null, ORAPARM_OUTPUT
```

```
OraParameters("MyMusic").ServerType = ORATYPE_BFILE

Set OraSql =
  OraDb.CreateSql(
    "BEGIN SELECT music INTO :MyMusic FROM multimedia_tab WHERE clip_id = :id;
    END;", ORASQL_FAILEXEC)

Set OraMusic = OraParameters("MyMusic").Value

If OraMusic.Size = 0 Then
  MsgBox "BFile size is 0"
Else
  MsgBox "BFile size is " & OraMusic.Size
End If
OraDb.Connection.CommitTrans
```

Java (JDBC) : BFILE の長さの取得

```
import java.io.InputStream;
import java.io.OutputStream;

// コア JDBC クラス :
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 固有の JDBC クラス :
import oracle.sql.*;
import oracle.jdbc.driver.*;

public class Ex4_74
{
    static final int MAXBUFSIZE = 32767;
    public static void main (String args [])
        throws Exception
    {
        // Oracle JDBC Driver をロードします。
        DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
    }
}
```

```
// データベースに接続します。
Connection conn =
    DriverManager.getConnection ("jdbc:oracle:oci8:@", "samp", "samp");

// 自動コミットがオフの場合、高速になります。
conn.setAutoCommit (false);

// 文を作成します。
Statement stmt = conn.createStatement ();
try
{
    BFILE lob_loc = null;

    ResultSet rset = stmt.executeQuery (
        "SELECT photo FROM multimedia_tab WHERE clip_id = 3");
    if (rset.next ())
    {
        lob_loc = ((OracleResultSet)rset).getBFILE (1);
    }

    // BFILE が存在するかどうかを確認します。
    System.out.println("Result from fileExists(): " + lob_loc.fileExists());

    // BFILE の長さを戻します。
    long length = lob_loc.length();
    System.out.println("Length of BFILE: " + length);

    // この BFILE のディレクトリ別名を取得します。
    System.out.println("Directory alias: " + lob_loc.getDirAlias());

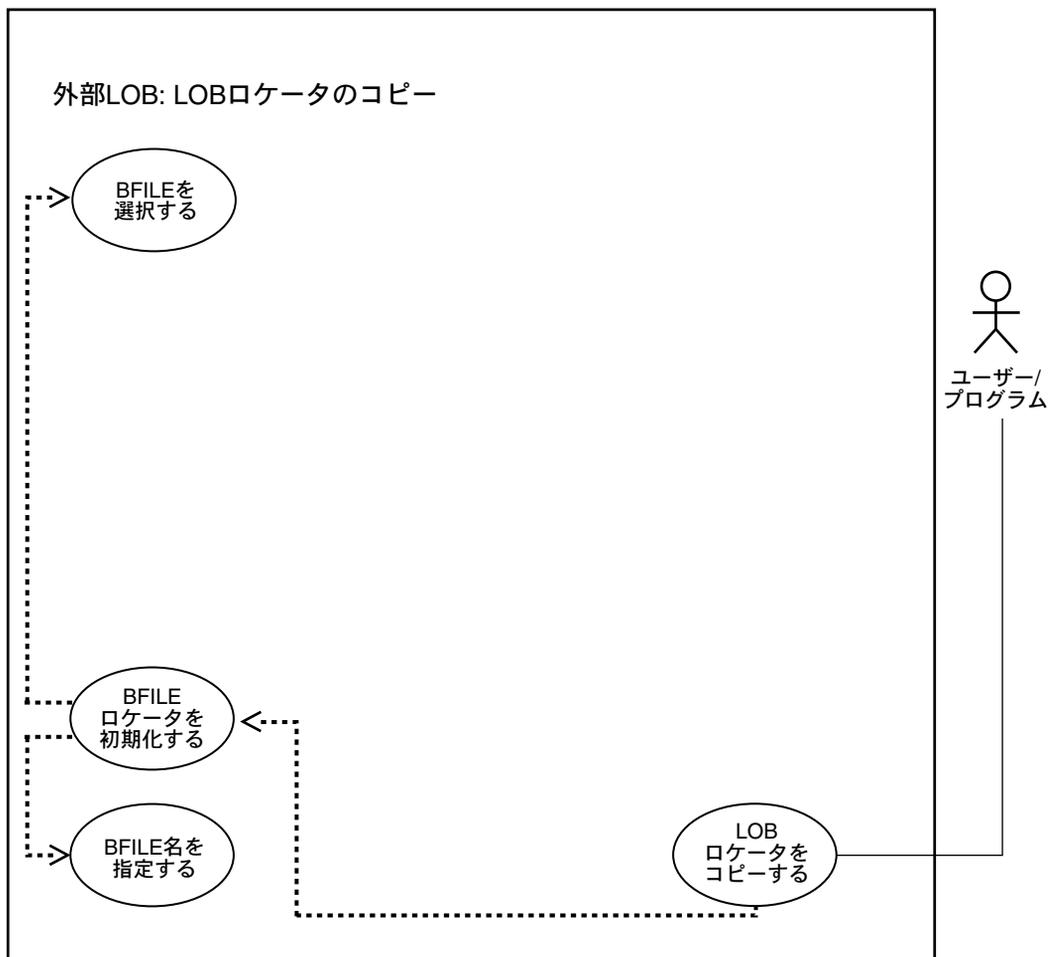
    // この BFILE のファイル名を取得します。
    System.out.println("File name: " + lob_loc.getName());

    stmt.close ();
    conn.commit ();
    conn.close ();

}
catch (SQLException e)
{
    e.printStackTrace ();
}
}
```

BFILE 用の LOB ロケータのコピー

図 11-25 ユースケース図：BFILE 用の LOB ロケータのコピー



参照： 外部 LOB (BFILE) に関するすべての基本操作については、11-2 ページの「ユースケース・モデル:外部 LOB (BFILE)」を参照してください。

用途

BFILE 用の LOB ロケータをコピーします。

使用上の注意

ありません。

構文

各プログラム環境で使用可能な機能のリストは、[第 3 章「LOB プログラム環境」](#)を参照してください。各プログラム環境における構文については、次のマニュアルの項を参照してください。

- SQL: 『Oracle8i SQL リファレンス』の第 7 章「SQL 文」の「CREATE PROCEDURE」
- C (OCI) : 『Oracle8i コール・インタフェース・プログラマーズ・ガイド』の第 7 章「LOB および FILE 操作」の使用上の注意、および第 15 章「OCI リレーショナル関数」の「LOB 関数」の OCILobLocatorAssign()
- COBOL (Pro*COBOL) : 『Oracle8i Pro*COBOL プリコンパイラ・プログラマーズ・ガイド』の第 13 章「ラージ・オブジェクト (LOB)」の「LOB 文」の使用上の注意、および付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB ASSIGN (実行可能埋込み SQL 拡張要素)」
- C/C++ (Pro*C/C++) : 『Oracle8i Pro*C/C++ プリコンパイラ・プログラマーズ・ガイド』の第 16 章「ラージ・オブジェクト (LOB)」の「LOB 文」の使用上の注意、および付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB ASSIGN (実行可能埋込み SQL 拡張要素)」
- Visual Basic (OO4O) : 参照マニュアルはありません。
- Java (JDBC) : 『Oracle8i JDBC 開発者ガイドおよびリファレンス』の第 7 章「LOB と BFILE の操作」の「BLOB と CLOB の操作」の「BLOB または CLOB 列の作成と移入」、および『Oracle8i SQLJ 開発者ガイドおよびリファレンス』の第 5 章「型のサポート」の「Oracle の拡張型」の「BLOB、CLOB および BFILE のサポート」

使用例

次の例では、ある BFILE ロケータを、Photo に関連する別のロケータに割り当てます。

例

次のプログラム環境の例が示されています。

- [PL/SQL: BFILE 用の LOB ロケータのコピー](#) (11-145 ページ)
- [C \(OCI\) : BFILE 用の LOB ロケータのコピー](#) (11-151 ページ)

- COBOL (Pro*COBOL) : BFILE 用の LOB ロケータのコピー (11-153 ページ)
- C/C++ (Pro*C/C++) : BFILE 用の LOB ロケータのコピー (11-154 ページ)
- Visual Basic: 今回のリリースでは例は記載されていません。
- Java (JDBC) : BFILE 用の LOB ロケータのコピー (11-154 ページ)

PL/SQL: BFILE 用の LOB ロケータのコピー

注意: PL/SQL で、1 つの BFILE を別の BFILE に割り当てる場合、「=」符号を使用します。詳細は、5-2 ページの「[読取り一貫性のあるロケータ](#)」を参照してください。

```

/* 例のプロシージャ BFILEAssign_proc は、DBMS_LOB パッケージの
一部ではないことに注意してください。*/
CREATE OR REPLACE PROCEDURE BFILEAssign_proc IS
  Lob_loc1  BFILE;
  Lob_loc2  BFILE;
BEGIN
  SELECT Photo INTO Lob_loc1 FROM Multimedia_tab WHERE Clip_ID = 3
  FOR UPDATE;
  /* Lob_loc1 を Lob_loc2 に割り当て、両方が同じオペレーティング・システム・
  ファイルを参照するようにします。*/
  Lob_loc2 := Lob_loc1;
  /* これで、Lob_loc1 または Lob_loc2 の一方から BFILE を読み込むことができます。*/
END;
```

C (OCI) : BFILE 用の LOB ロケータのコピー

```

/* マルチメディア表から LOB/BFILE を選択します。*/
void selectLob(Lob_loc, errhp, svchp, stmthp)
OCILobLocator *Lob_loc;
OCIError      *errhp;
OCISvcCtx     *svchp;
OCISmt       *stmthp;
{
  OCIDefine *dfnhp;
  text *selstmt = (text *) "SELECT Photo FROM Multimedia_tab \
                          WHERE Clip_ID = 3";

  /* SQL の SELECT 文を準備します。*/
  checkerr (errhp, OCISmtPrepare(stmthp, errhp, selstmt,
                                (ub4) strlen((char *) selstmt),
                                (ub4) OCI_NTV_SYNTAX, (ub4) OCI_DEFAULT));
```

```
/* BFILE 列に対する定義をコールします。*/
checkerr (errhp, OCIDefineByPos(stmtthp, &dfnhp, errhp, 1,
                                (dvoid *)&Lob_loc, 0, SQLT_BFILE,
                                (dvoid *)0, (ub2 *)0, (ub2 *)0,
                                OCI_DEFAULT));

/* SQL の SELECT 文を実行します。*/
checkerr (errhp, OCIStmtExecute(svchp, stmtthp, errhp, (ub4) 1, (ub4) 0,
                                (CONST OCISnapshot*) 0, (OCISnapshot*) 0,
                                (ub4) OCI_DEFAULT));
}

void BfileAssign(envhp, errhp, svchp, stmtthp)
OCIEnv      *envhp;
OCIError    *errhp;
OCISvcCtx   *svchp;
OCIStmt     *stmtthp;
{
    OCILobLocator *src_loc;
    OCILobLocator *dest_loc;

    /* ロケータ記述子を割り当てます。*/
    (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &src_loc,
                              (ub4) OCI_DTYPE_FILE,
                              (size_t) 0, (dvoid **) 0);

    (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &dest_loc,
                              (ub4) OCI_DTYPE_FILE,
                              (size_t) 0, (dvoid **) 0);

    /* BFILE を選択します。*/
    selectLob(src_loc, errhp, svchp, stmtthp);

    checkerr(errhp, OCILobLocatorAssign(svchp, errhp, src_loc, &dest_loc));

    /* ロケータ記述子を解放します。*/
    OCIDescriptorFree((dvoid *)src_loc, (ub4)OCI_DTYPE_FILE);
    OCIDescriptorFree((dvoid *)dest_loc, (ub4)OCI_DTYPE_FILE);
}
```

COBOL (Pro*COBOL) : BFILE 用の LOB ロケータのコピー

```
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.

* BFILE ロケータの割当ておよび初期化を行います。
EXEC SQL ALLOCATE :BFILE1 END-EXEC.
EXEC SQL ALLOCATE :BFILE2 END-EXEC.

EXEC SQL WHENEVER NOT FOUND GOTO END-OF-BFILE END-EXEC.
EXEC SQL
    SELECT PHOTO INTO :BFILE1
    FROM MULTIMEDIA_TAB WHERE CLIP_ID = 3 END-EXEC.
EXEC 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 ORASLN RD TO ORASLN RD.
DISPLAY " ".
DISPLAY "ORACLE ERROR DETECTED ON LINE ", ORASLN RD, ":".
DISPLAY " ".
DISPLAY SQLERRMC.
EXEC SQL ROLLBACK WORK RELEASE END-EXEC.
STOP RUN.
```

C/C++ (Pro*C/C++) : BFILE 用の LOB ロケータのコピー

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

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

void BFILEAssign_proc()
{
    OCIBFileLocator *Lob_loc1, *Lob_loc2;

    EXEC SQL WHENEVER SQLERROR DO Sample_Error();
    EXEC SQL ALLOCATE :Lob_loc1;
    EXEC SQL ALLOCATE :Lob_loc2;
    EXEC SQL SELECT Photo INTO :Lob_loc1
        FROM Multimedia_tab WHERE Clip_ID = 3;
    /* Lob_loc1 を Lob_loc2 に割り当て、両方が同じオペレーティング・システム・
       ファイルを参照するようにします。*/
    EXEC SQL LOB ASSIGN :Lob_loc1 TO :Lob_loc2;
    /* これで、Lob_loc1 または Lob_loc2 の一方から BFILE を読み込むことができます。*/
}

void main()
{
    char *samp = "samp/samp";
    EXEC SQL CONNECT :samp;
    BFILEAssign_proc();
    EXEC SQL ROLLBACK WORK RELEASE;
}
```

Java (JDBC) : BFILE 用の LOB ロケータのコピー

```
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 固有の JDBC クラス :
import oracle.sql.*;
import oracle.jdbc.driver.*;
public class Ex4_81
{
    public static void main (String args [])
        throws Exception
    {
        // Oracle JDBC Driver をロードします。
        DriverManager.registerDriver (new oracle.jdbc.driver.OracleDriver());

        // データベースに接続します。
        Connection conn =
            DriverManager.getConnection ("jdbc:oracle:oci8:@", "samp", "samp");

        // 自動コミットがオフの場合、高速になります。
        conn.setAutoCommit (false);

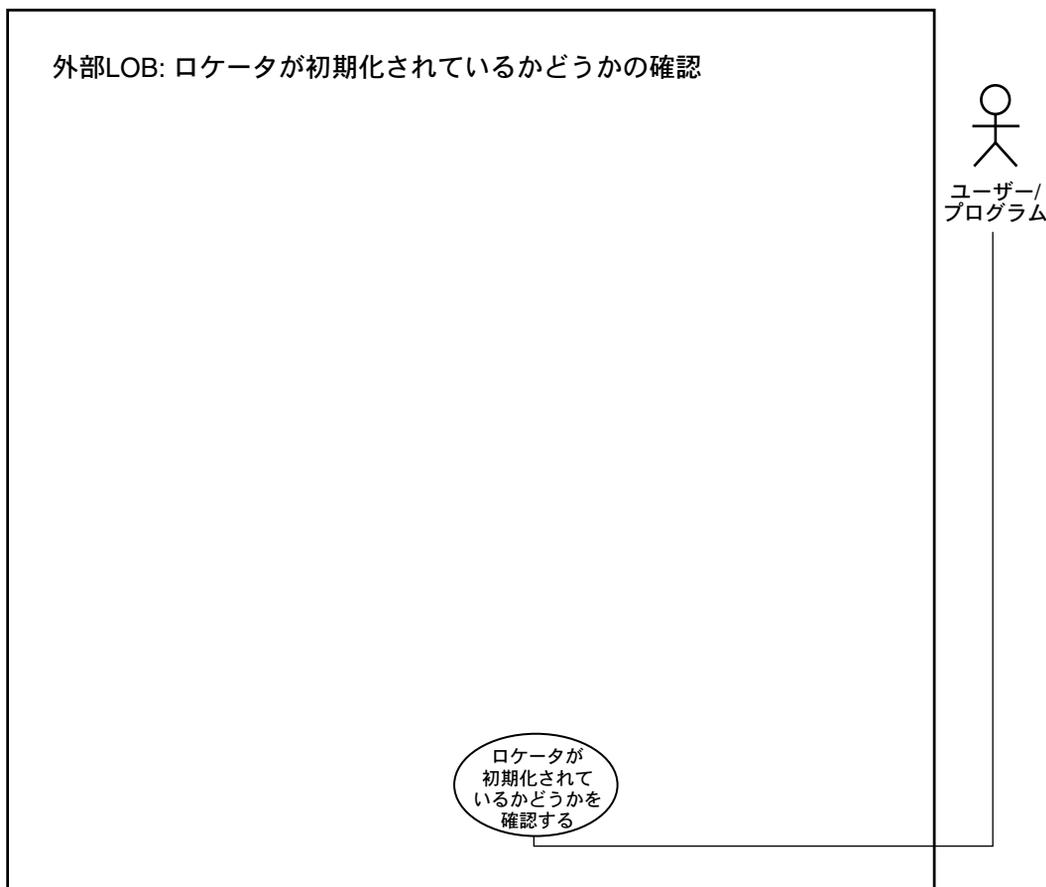
        // 文を作成します。
        Statement stmt = conn.createStatement ();
        try
        {
            BFILE lob_loc1 = null;
            BFILE lob_loc2 = null;

            ResultSet rset = stmt.executeQuery (
                "SELECT photo FROM multimedia_tab WHERE clip_id = 3");
            if (rset.next())
            {
                lob_loc1 = ((OracleResultSet)rset).getBFILE (1);
            }

            // Lob_loc1 を Lob_loc2 に割り当て、両方が同じオペレーティング・システム・
            // ファイルを参照するようにします。
            // これで、Lob_loc1 または Lob_loc2 の一方から BFILE を読み込むことができます。
            lob_loc2 = lob_loc1;
            stmt.close();
            conn.commit();
            conn.close();
        }
        //catch (SQLException e) を実行します。
        catch (Exception e)
        {
            e.printStackTrace();
        }
    }
}
```

BFILE の LOB ロケータが初期化されているかどうかの確認

図 11-26 ユースケース図：LOB ロケータが初期化されているかどうかの確認



参照： 外部 LOB (BFILE) に関するすべての基本操作については、11-2 ページの「ユースケース・モデル:外部 LOB (BFILE)」を参照してください。

用途

BFILE の LOB ロケータが初期化されているかどうかを確認します。

使用上の注意

クライアント側では、OCILOB* インタフェース (OCILOBWrite など)、または OCILOB* インタフェースを使用するプログラム環境をコールする前に、まず、SELECT などを使用して LOB ロケータを初期化します。

アプリケーションで、ある関数から別の関数にロケータを渡す必要がある場合、ロケータがすでに初期化されているかどうかを確認する場合があります。ロケータが初期化されていない場合、OCILOB* インタフェースをコールする前にアプリケーションがエラーを戻すか、または SELECT を実行するように設計できます。

構文

各プログラム環境で使用可能な機能のリストは、[第 3 章「LOB プログラム環境」](#)を参照してください。各プログラム環境における構文については、次のマニュアルの項を参照してください。

- PL/SQL (DBMS_LOB) : 参照マニュアルはありません。
- C (OCI) : 『Oracle8i コール・インタフェース・プログラマーズ・ガイド』の第 7 章「LOB および FILE 操作」の使用上の注意、および第 15 章「OCI リレーショナル関数」の「LOB 関数」の OCILOBLocatorIsInit()
- COBOL (Pro*COBOL) : 参照マニュアルはありません。
- C/C++ (Pro*C/C++) : 『Oracle8i Pro*C/C++ プリコンパイラ・プログラマーズ・ガイド』の第 16 章「ラージ・オブジェクト (LOB)」の「LOB 文」の使用上の注意、付録 F「埋込み SQL 文およびプリコンパイラ宣言文」、および『Oracle8i コール・インタフェース・プログラマーズ・ガイド』の第 7 章「LOB および FILE 操作」の使用上の注意、および第 15 章「OCI リレーショナル関数」の「LOB 関数」の OCILOBLocatorIsInit()
- Visual Basic (OO4O) : 参照マニュアルはありません。
- Java (JDBC) : 参照マニュアルはありません。

使用例

ありません。

例

次のプログラム環境の例が示されています。

- PL/SQL (DBMS_LOB) : 今回のリリースでは例は記載されていません。
- C (OCI) : [BFILE の LOB ロケータが初期化されているかどうかの確認](#) (11-157 ページ)
- COBOL (Pro*COBOL) : 今回のリリースでは例は記載されていません。

- C/C++ (Pro*C/C++) : BFILE の LOB ロケータが初期化されているかどうかの確認 (11-159 ページ)
- Visual Basic (OO4O) : 今回のリリースでは例は記載されていません。
- Java (JDBC) : 今回のリリースでは例は記載されていません。

C (OCI) : BFILE の LOB ロケータが初期化されているかどうかの確認

```

/* マルチメディア表から LOB/BFILE を選択します。*/
void selectLob(Lob_loc, errhp, svchp, stmthp)
OCILobLocator *Lob_loc;
OCIError      *errhp;
OCISvcCtx     *svchp;
OCIStmt       *stmthp;
{
    OCIDefine *dfnhp;
    text *selstmt = (text *) "SELECT Photo FROM Multimedia_tab \
                               WHERE Clip_ID = 3";

    /* SQL の SELECT 文を準備します。*/
    checkerr (errhp, OCIStmtPrepare(stmthp, errhp, selstmt,
                                    (ub4) strlen((char *) selstmt),
                                    (ub4) OCI_NTV_SYNTAX, (ub4)OCI_DEFAULT));

    /* BFILE 列に対する定義をコールします。*/
    checkerr (errhp, OCIDefineByPos(stmthp, &dfnhp, errhp, 1,
                                    (dvoid *)&Lob_loc, 0, SQLT_BFILE,
                                    (dvoid *)0, (ub2 *)0, (ub2 *)0,
                                    OCI_DEFAULT));

    /* SQL の SELECT 文を実行します。*/
    checkerr (errhp, OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                                    (CONST OCISnapshot*) 0, (OCISnapshot*) 0,
                                    (ub4) OCI_DEFAULT));
}

void BfileIsInit(envhp, errhp, svchp, stmthp)
OCIEnv      *envhp;
OCIError    *errhp;
OCISvcCtx   *svchp;
OCIStmt     *stmthp;
{
    OCILobLocator *bfile_loc;

```

```

boolean is_init;

/* ロケータ記述子を割り当てます。*/
(void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &bfile_loc,
                          (ub4) OCI_DTYPE_FILE,
                          (size_t) 0, (dvoid **) 0);

/* BFILE を選択します。*/
selectLob(bfile_loc, errhp, svchp, stmthp);

checkerr(errhp, OCILobLocatorIsInit(envhp, errhp, bfile_loc, &is_init));

if (is_init == TRUE)
{
    printf("Locator is initialized\n");
}
else
{
    printf("Locator is not initialized\n");
}
/* ロケータ記述子を解放します。*/
OCIDescriptorFree((dvoid *)bfile_loc, (ub4)OCI_DTYPE_FILE);
}

```

C/C++ (Pro*C/C++) : BFILE の LOB ロケータが初期化されているかどうかの確認

/* Pro*C/C++ には、BFILE ロケータが初期化されているかどうかを判断するための埋込み SQL フォームがありません。Pro*C/C++ のロケータは、EXEC SQL ALLOCATE 文を介して割り当てられた場合に初期化されます。ただし、この例に示されているように、埋込み SQL および OCI を使用する例を書くことができます。*/

```

#include <sql2oci.h>
#include <stdio.h>
#include <sqlca.h>

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

void BFILELocatorIsInit_proc()
{
    OCIBFileLocator *Lob_loc;

```

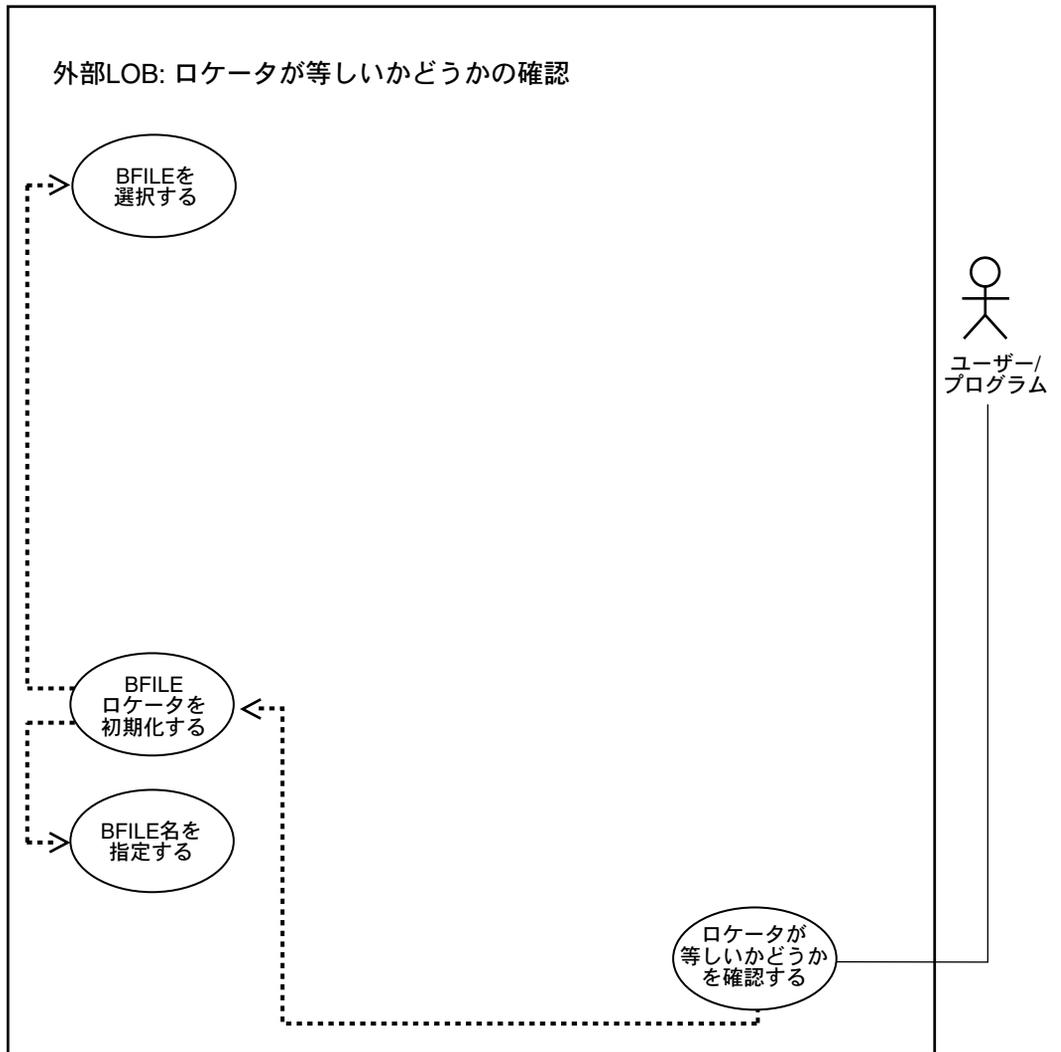
```
OCIEnv *oeh;
OCIError *err;
boolean isInitialized = 0;

EXEC SQL WHENEVER SQLERROR DO Sample_Error();
EXEC SQL ALLOCATE :Lob_loc;
EXEC SQL SELECT Mtab.Voiced_ref.Recording INTO :Lob_loc
FROM Multimedia_tab Mtab WHERE Mtab.Clip_ID = 3;
/* SQLLIB ルーチンを使用して、OCI 環境ハンドルを取得します。*/
(void) SQLEnvGet(SQL_SINGLE_RCTX, &oeh);
/* OCI エラー・ハンドルを割り当てます。*/
(void) OCIHandleAlloc((dvoid *)oeh, (dvoid **)&err,
                     (ub4)OCI_HTYPE_ERROR, (ub4)0, (dvoid **)0);
/* OCI を使用して、ロケータが初期化されているかどうかを判断します。*/
(void) OCILobLocatorIsInit(oeh, err, Lob_loc, &isInitialized);
if (isInitialized)
    printf("Locator is initialized\n");
else
    printf("Locator is not initialized\n");
/* この例では、ロケータが初期化されていることに注意してください。*/
/* OCI エラー・ハンドルの割当てを解除します。*/
(void) OCIHandleFree(err, OCI_HTYPE_ERROR);
/* ロケータが保持しているリソースを解放します。*/
EXEC SQL FREE :Lob_loc;
}

void main()
{
    char *samp = "samp/samp";
    EXEC SQL CONNECT :samp;
    BFILELocatorIsInit_proc();
    EXEC SQL ROLLBACK WORK RELEASE;
}
```

BFILE の LOB ロケータが他と等しいかどうかの確認

図 11-27 ユースケース図：BFILE の LOB ロケータが他と等しいかどうかの確認



参照： 外部 LOB (BFILF) に関するすべての基本操作については、11-2 ページの「ユースケース・モデル:外部 LOB (BFILF)」を参照してください。

用途

BFILF の LOB ロケータが他と等しいかどうかを確認します。

使用上の注意

ありません。

構文

各プログラム環境で使用可能な機能のリストは、第 3 章「LOB プログラム環境」を参照してください。各プログラム環境における構文については、次のマニュアルの項を参照してください。

- PL/SQL (DBMS_LOB) : 参照マニュアルはありません。
- C (OCI) : 『Oracle8i コール・インタフェース・プログラマーズ・ガイド』の第 7 章「LOB および FILE 操作」の使用上の注意、および第 15 章「OCI リレーショナル関数」の「LOB 関数」の OCILobIsEqual()
- COBOL (Pro*COBOL) : 参照マニュアルはありません。
- C/C++ (Pro*C/C++) : 『Oracle8i Pro*C/C++ プリコンパイラ・プログラマーズ・ガイド』の第 16 章「ラージ・オブジェクト (LOB)」の「LOB 文」の使用上の注意、付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB ASSIGN (実行可能埋込み SQL 拡張要素)」、および『Oracle8i コール・インタフェース・プログラマーズ・ガイド』の第 7 章「LOB および FILE 操作」の使用上の注意、および第 15 章「OCI リレーショナル関数」の「LOB 関数」の OCILobIsEqual()
- Visual Basic (OO4O) : 参照マニュアルはありません。
- Java (JDBC) : 『Oracle8i JDBC 開発者ガイドおよびリファレンス』の第 7 章「LOB と BFILF の操作」の「BLOB と CLOB の操作」の「BLOB または CLOB 列の作成と移入」、および『Oracle8i SQLJ 開発者ガイドおよびリファレンス』の第 5 章「型のサポート」の「Oracle の拡張型」の「BLOB、CLOB および BFILF のサポート」

使用例

2 つのロケータが等しい場合、それらが同じバージョンの LOB データを参照していることを意味します (5-2 ページの「読取り一貫性のあるロケータ」を参照)。

例

次のプログラム環境の例が示されています。

- PL/SQL: 今回のリリースでは例は記載されていません。
- [C \(OCI\) : BFILE の LOB ロケータが他と等しいかどうかの確認](#) (11-158 ページ)
- COBOL (Pro*COBOL) : 今回のリリースでは例は記載されていません。
- [C/C++ \(Pro*C/C++\) : BFILE の LOB ロケータが他と等しいかどうかの確認](#) (11-165 ページ)
- Visual Basic (OO4O) : 今回のリリースでは例は記載されていません。
- [Java \(JDBC\) : BFILE の LOB ロケータが他と等しいかどうかの確認](#) (11-166 ページ)

C (OCI) : BFILE の LOB ロケータが他と等しいかどうかの確認

```

/* マルチメディア表から LOB/BFILE を選択します。*/
void selectLob(Lob_loc, errhp, svchp, stmthp)
OCILobLocator *Lob_loc;
OCIError      *errhp;
OCISvcCtx     *svchp;
OCIStmt       *stmthp;
{
    OCIDefine *dfnhp;
    text *selstmt = (text *) "SELECT Photo FROM Multimedia_tab \
                               WHERE Clip_ID = 3";

    /* SQL の SELECT 文を準備します。*/
    checkerr (errhp, OCIStmtPrepare(stmthp, errhp, selstmt,
                                     (ub4) strlen((char *) selstmt),
                                     (ub4) OCI_MTV_SYNTAX, (ub4)OCI_DEFAULT));

    /* BFILE 列に対する定義をコールします。*/
    checkerr (errhp, OCIDefineByPos(stmthp, &dfnhp, errhp, 1,
                                     (dvoid *)&Lob_loc, 0, SQLT_BFILE,
                                     (dvoid *)0, (ub2 *)0, (ub2 *)0,
                                     OCI_DEFAULT));

    /* SQL の SELECT 文を実行します。*/
    checkerr (errhp, OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                                     (CONST OCISnapshot*) 0, (OCISnapshot*) 0,
                                     (ub4) OCI_DEFAULT));
}

```

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

    OCILobLocator *bfile_loc1;
    OCILobLocator *bfile_loc2;
    boolean is_equal;

    /* ロケータ記述子を割り当てます。 */
    (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &bfile_loc1,
                              (ub4) OCI_DTYPE_FILE,
                              (size_t) 0, (dvoid **) 0);

    (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &bfile_loc2,
                              (ub4) OCI_DTYPE_FILE,
                              (size_t) 0, (dvoid **) 0);

    /* BFILE を選択します。 */
    selectLob(bfile_loc1, errhp, svchp, stmthp);

    checkerr(errhp,
             OCILobLocatorAssign(svchp, errhp, bfile_loc1, &bfile_loc2));

    checkerr(errhp, OCILobIsEqual(envhp, bfile_loc1, bfile_loc2, &is_equal));

    if (is_equal == TRUE)
    {
        printf("Locators are equal\n");
    }
    else
    {
        printf("Locators are not equal\n");
    }

    /* ロケータ記述子を解放します。 */
    OCIDescriptorFree((dvoid *)bfile_loc1, (ub4)OCI_DTYPE_FILE);
    OCIDescriptorFree((dvoid *)bfile_loc2, (ub4)OCI_DTYPE_FILE);
}
```

C/C++ (Pro*C/C++) : BFILE の LOB ロケータが他と等しいかどうかの確認

/* Pro*C/C++ では、2つのロケータが等しいかどうかをテストするメカニズムが提供されません。
ただし、この例に示すように、OCI を直接使用することによって2つのロケータを比較し、これらが等しいかどうかを判断できます。*/

```
#include <sql2oci.h>
#include <stdio.h>
#include <sqlca.h>

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

void BFILELocatorIsEqual_proc()
{
    OCIBFileLocator *Lob_loc1, *Lob_loc2;
    OCIEnv *oeh;
    boolean isEqual = 0;

    EXEC SQL WHENEVER SQLERROR DO Sample_Error();
    EXEC SQL ALLOCATE :Lob_loc1;
    EXEC SQL ALLOCATE :Lob_loc2;
    EXEC SQL SELECT Photo INTO :Lob_loc1
        FROM Multimedia_tab WHERE Clip_ID = 3;
    EXEC SQL LOB ASSIGN :Lob_loc1 TO :Lob_loc2;
    /* これで、Lob_loc1 または Lob_loc2 の一方から BFILE を読み込むことができます。*/
    /* SQLLIB ルーチンを使用して、OCI 環境ハンドルを取得します。*/
    (void) SQLEnvGet(SQL_SINGLE_RCTX, &oeh);
    /* OCI をコールして、2つのロケータが等しいかどうかを確認します。*/
    (void) OCILobIsEqual(oeh, Lob_loc1, Lob_loc2, &isEqual);
    if (isEqual)
        printf("Locators are equal\n");
    else
        printf("Locators are not equal\n");
    /* この例では、LOB ロケータが等しいことに注意してください。*/
    EXEC SQL FREE :Lob_loc1;
    EXEC SQL FREE :Lob_loc2;
}
```

```
void main()
{
    char *samp = "samp/samp";
    EXEC SQL CONNECT :samp;
    BFILELocatorIsEqual_proc();
    EXEC SQL ROLLBACK WORK RELEASE;
}
```

Java (JDBC) : BFILE の LOB ロケータが他と等しいかどうかの確認

```
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 固有の JDBC クラス :
import oracle.sql.*;
import oracle.jdbc.driver.*;

public class Ex4_89
{

    public static void main (String args [])
        throws Exception
    {
        // Oracle JDBC Driver をロードします。
        DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

        // データベースに接続します。
        Connection conn =
            DriverManager.getConnection ("jdbc:oracle:oci8:@", "samp", "samp");

        // 自動コミットがオフの場合、高速になります。
        conn.setAutoCommit (false);

        // 文を作成します。
        Statement stmt = conn.createStatement ();

        try
        {
            BFILE lob_loc1 = null;
            BFILE lob_loc2 = null;
```

```
ResultSet rset = stmt.executeQuery (
    "SELECT photo FROM multimedia_tab WHERE clip_id = 3");
if (rset.next())
{
    lob_loc1 = ((OracleResultSet)rset).getBFILE (1);
}

// 両方の LOB が同じ BFILE を参照するように設定します。
lob_loc2 = lob_loc1;

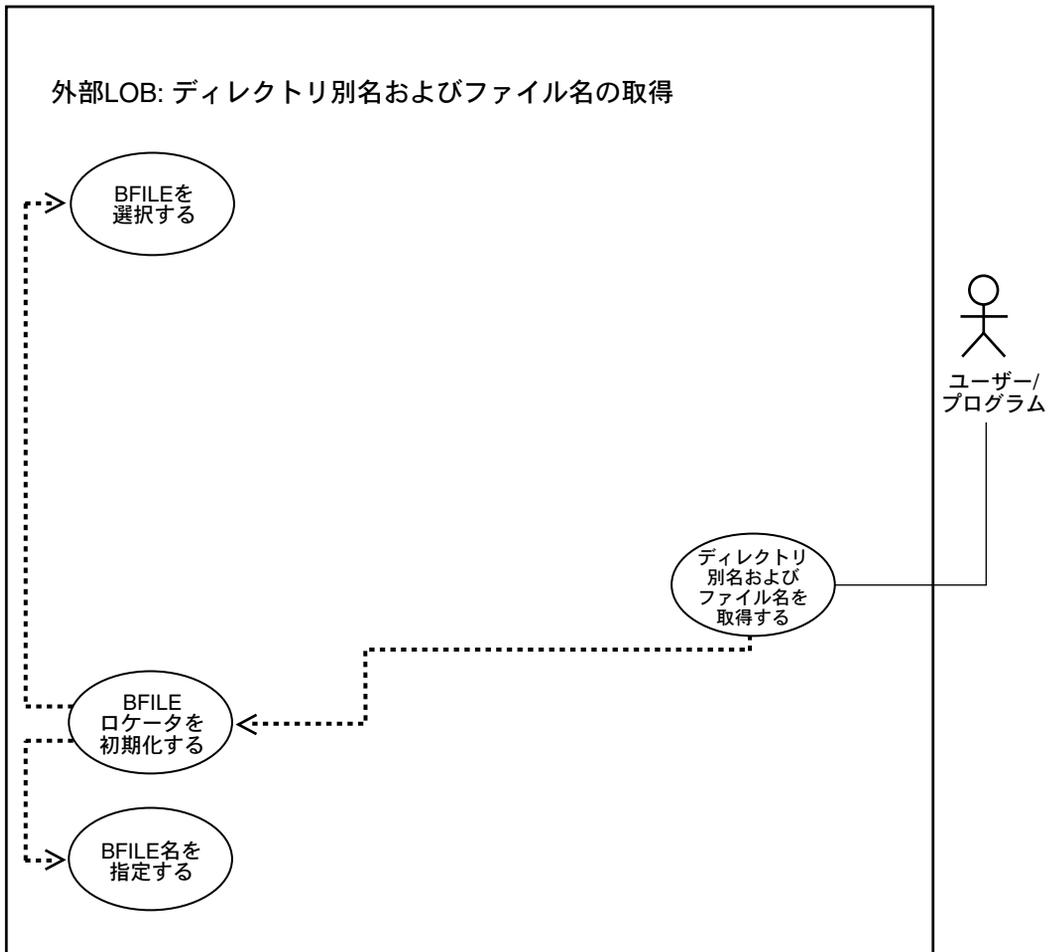
// この例では、ロケータが等しいことに注意してください。
if (lob_loc1.equals(lob_loc2))
{
    // ロケータが等しい場合の論理がここにあります。
    System.out.println("The BFILEs are equal");
}
else
{
    // ロケータが等しくない場合の論理がここにあります。
    System.out.println("The BFILEs are NOT equal");
}

stmt.close();
conn.commit();
conn.close();

}
catch (SQLException e)
{
    e.printStackTrace();
}
}
```

ディレクトリ別名およびファイル名の取得

図 11-28 ユースケース図：ディレクトリ別名およびファイル名の取得



参照： 外部 LOB (BFILE) に関するすべての基本操作については、11-2 ページの「ユースケース・モデル: 外部 LOB (BFILE)」を参照してください。

用途

ディレクトリ別名およびファイル名を取得します。

使用上の注意

ありません。

構文

各プログラム環境で使用可能な機能のリストは、第3章「LOB プログラム環境」を参照してください。各プログラム環境における構文については、次のマニュアルの項を参照してください。

- PL/SQL (DBMS_LOB) : 『Oracle8i PL/SQL パッケージ・プロシージャ・リファレンス』の第18章「DBMS_LOB」の「サブプログラムの要約」の FILEGETNAME プロシージャ
- C (OCI) : 『Oracle8i コール・インタフェース・プログラマーズ・ガイド』の第7章「LOB および FILE 操作」の使用上の注意、および第15章「OCI リレーショナル関数」の「LOB 関数」の OCILobFileName()
- COBOL (Pro*COBOL) : 『Oracle8i Pro*COBOL プリコンパイラ・プログラマーズ・ガイド』の第13章「ラージ・オブジェクト (LOB)」の「LOB 文」の使用上の注意、および付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB DESCRIBE (実行可能埋込み SQL 拡張要素)」
- C/C++ (Pro*C/C++) : 『Oracle8i Pro*C/C++ プリコンパイラ・プログラマーズ・ガイド』の第16章「ラージ・オブジェクト (LOB)」の「LOB 文」の使用上の注意、および付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB DESCRIBE (実行可能埋込み SQL 拡張要素)」
- Visual Basic (Oracle Objects for OLE オンライン・ヘルプ): 「Help」から、「PROPERTIES」>「DirectoryName」、「FileName」、および「OBJECTS」>「OraBfile」>「Examples」を選択してください。
- Java (JDBC) : 『Oracle8i JDBC 開発者ガイドおよびリファレンス』の第7章「LOB と BFILE の操作」の「BLOB と CLOB の操作」の「BLOB または CLOB 列の作成と移入」、および『Oracle8i SQLJ 開発者ガイドおよびリファレンス』の第5章「型のサポート」の「Oracle の拡張型」の「BLOB、CLOB および BFILE のサポート」

使用例

次の例では、BFILE、Music に関連するディレクトリ別名およびファイル名を取り出します。

例

次のプログラム環境の例が示されています。

- [PL/SQL \(DBMS_LOB パッケージ\) : ディレクトリ別名およびファイル名の取得 \(11-165 ページ\)](#)
- [C \(OCI\) : ディレクトリ別名およびファイル名の取得 \(11-170 ページ\)](#)
- [COBOL \(Pro*COBOL\) : ディレクトリ別名およびファイル名の取得 \(11-172 ページ\)](#)
- [C/C++ \(Pro*C/C++\) : ディレクトリ別名およびファイル名の取得 \(11-173 ページ\)](#)
- [Visual Basic \(OO4O\) : ディレクトリ別名およびファイル名の取得 \(11-174 ページ\)](#)
- [Java \(JDBC\) : ディレクトリ別名およびファイル名の取得 \(11-175 ページ\)](#)

PL/SQL (DBMS_LOB パッケージ) : ディレクトリ別名およびファイル名の取得

```
CREATE OR REPLACE PROCEDURE getNameBFILE_proc IS
  Lob_loc          BFILE;
  DirAlias_name    VARCHAR2(30);
  File_name        VARCHAR2(40);
BEGIN
  SELECT Music INTO Lob_loc FROM Multimedia_tab WHERE Clip_ID = 3;
  DBMS_LOB.FILEGETNAME(Lob_loc, DirAlias_name, File_name);
  /* ディレクトリ別名およびファイル名に基づいて、何らかの処理を行います。*/
END;
```

C (OCI) : ディレクトリ別名およびファイル名の取得

```
/* マルチメディア表から LOB/BFILE を選択します。*/
void selectLob(Lob_loc, errhp, svchp, stmthp)
OCILOBLocator *Lob_loc;
OCIError *errhp;
OCISvcCtx *svchp;
OCIStmt *stmthp;
{
  OCIDefine *dfnhp;
  text *selstmt = (text *) "SELECT Photo FROM Multimedia_tab \
                           WHERE Clip_ID = 3";

  /* SQL の SELECT 文を準備します。*/
  checkerr (errhp, OCIStmtPrepare(stmthp, errhp, selstmt,
                                  (ub4) strlen((char *) selstmt),
                                  (ub4) OCI_NTV_SYNTAX, (ub4) OCI_DEFAULT));
```

```
/* BFILE 列に対する定義をコールします。*/
checkerr (errhp, OCIDefineByPos(stmthp, &dfnbp, errhp, 1,
                                (dvoid *)&Lob_loc, 0, SQLT_BFILE,
                                (dvoid *)0, (ub2 *)0, (ub2 *)0,
                                OCI_DEFAULT));

/* SQL の SELECT 文を実行します。*/
checkerr (errhp, OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                                (CONST OCISnapshot*) 0, (OCISnapshot*) 0,
                                (ub4) OCI_DEFAULT));
}

void BfileGetDirFile(envhp, errhp, svchp, stmthp)
OCIEnv      *envhp;
OCIError    *errhp;
OCISvcCtx   *svchp;
OCIStmt     *stmthp;
{
    OCILobLocator *bfile_loc;
    OraText dir_alias[32] = NULL;
    OraText filename[256] = NULL;
    ub2 d_length = 32;
    ub2 f_length = 256;

    /* ロケータ記述子を割り当てます。*/
    (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &bfile_loc,
                              (ub4) OCI_DTYPE_FILE,
                              (size_t) 0, (dvoid **) 0);

    /* BFILE を選択します。*/
    selectLob(bfile_loc, errhp, svchp, stmthp);

    checkerr(errhp, OCILobFileGetName(envhp, errhp, bfile_loc,
                                      dir_alias, &d_length, filename, &f_length));

    printf("Directory Alias : [%s]\n", dir_alias);
    printf("File name : [%s]\n", filename);

    /* ロケータ記述子を解放します。*/
    OCIDescriptorFree((dvoid *)bfile_loc, (ub4)OCI_DTYPE_FILE);
}
}
```

COBOL (Pro*COBOL) : ディレクトリ別名およびファイル名の取得

```
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.

* BFILE ロケータの割当ておよび初期化を行います。
EXEC SQL ALLOCATE :BFILE1 END-EXEC.
EXEC SQL WHENEVER NOT FOUND GOTO END-OF-BFILE END-EXEC.

* BFILE ロケータを移入します。
EXEC SQL
    SELECT PHOTO INTO :BFILE1
    FROM MULTIMEDIA_TAB WHERE CLIP_ID = 3 END-EXEC.

* LOB DESCRIBE 機能を使用して、ディレクトリ別名およびファイル名を取得します。
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++) : ディレクトリ別名およびファイル名の取得

```
#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];  
    /* データ型を等しくするのはオプションです。*/  
    EXEC SQL VAR Directory IS STRING;  
    EXEC SQL VAR Filename IS STRING;  
    EXEC SQL WHENEVER SQLERROR DO Sample_Error();  
    EXEC SQL ALLOCATE :Lob_loc;  
  
    /* BFILE を選択します。*/  
    EXEC SQL SELECT Photo INTO :Lob_loc  
        FROM Multimedia_tab WHERE Clip_ID = 3;  
    /* BFILE をオープンします。*/  
    EXEC SQL LOB OPEN :Lob_loc READ ONLY;  
    /* ディレクトリ別名およびファイル名を取得します。*/  
    EXEC SQL LOB DESCRIBE :Lob_loc  
        GET DIRECTORY, FILENAME INTO :Directory, :Filename;  
  
    /* BFILE をクローズします。*/  
    EXEC SQL LOB CLOSE :Lob_loc;
```

```
printf("Directory Alias: %s\n", Directory);
printf("Filename: %s\n", Filename);
/* ロケータが保持しているリソースを解放します。*/
EXEC SQL FREE :Lob_loc;
}

void main()
{
    char *samp = "samp/samp";
    EXEC SQL CONNECT :samp;
    getBFILEDirectoryAndFilename_proc();
    EXEC SQL ROLLBACK WORK RELEASE;
}
```

Visual Basic (0040) : ディレクトリ別名およびファイル名の取得

ここで示す PL/SQL パッケージおよび表は、標準の 0040 インストールの一部ではないことに注意してください。

```
Dim MySession As OraSession
Dim OraDb As OraDatabase
Dim OraMusic1 As OraBfile, OraSql As OraSqlStmt

Set MySession = CreateObject("OracleInProcServer.XOraSession")
Set OraDb = MySession.OpenDatabase("exampledb", "scott/tiger", 0&)
OraDb.Connection.BeginTrans
Set OraParameters = OraDb.Parameters
OraParameters.Add "id", 1001, ORAPARM_INPUT

' BFILE 型の OUT パラメータを定義します。
OraParameters.Add "MyMusic", Null, ORAPARM_OUTPUT
OraParameters("MyMusic").ServerType = ORATYPE_BFILE

Set OraSql =
    OraDb.CreateSql(
        "BEGIN SELECT music INTO :MyMusic FROM multimedia_tab WHERE clip_id = :id;
        END;", ORASQL_FAILEXEC)

Set OraMusic1 = OraParameters("MyMusic").Value
' ディレクトリ別名およびファイル名を取得します。
MsgBox " Directory alias is " & OraMusic1.DirectoryName &
    " Filename is " & OraMusic1.filename

OraDb.Connection.CommitTrans
```

Java (JDBC) : ディレクトリ別名およびファイル名の取得

```
import java.io.InputStream;
import java.io.OutputStream;
// コア JDBC クラス :
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 固有の JDBC クラス :
import oracle.sql.*;
import oracle.jdbc.driver.*;
public class Ex4_74
{
    static final int MAXBUFSIZE = 32767;
    public static void main (String args [])
        throws Exception
    {
        // Oracle JDBC Driver をロードします。
        DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

        // データベースに接続します。
        Connection conn =
            DriverManager.getConnection ("jdbc:oracle:oci8:@", "samp", "samp");

        // 自動コミットがオフの場合、高速になります。
        conn.setAutoCommit (false);

        // 文を作成します。
        Statement stmt = conn.createStatement ();
        try
        {
            BFILE lob_loc = null;
            ResultSet rset = stmt.executeQuery (
                "SELECT photo FROM multimedia_tab WHERE clip_id = 3");
            if (rset.next())
            {
                lob_loc = ((OracleResultSet)rset).getBFILE (1);
            }
        }
    }
}
```

ディレクトリ別名およびファイル名の取得

```
// BFILEが存在するかどうかを確認します。
System.out.println("Result from fileExists(): " + lob_loc.fileExists());

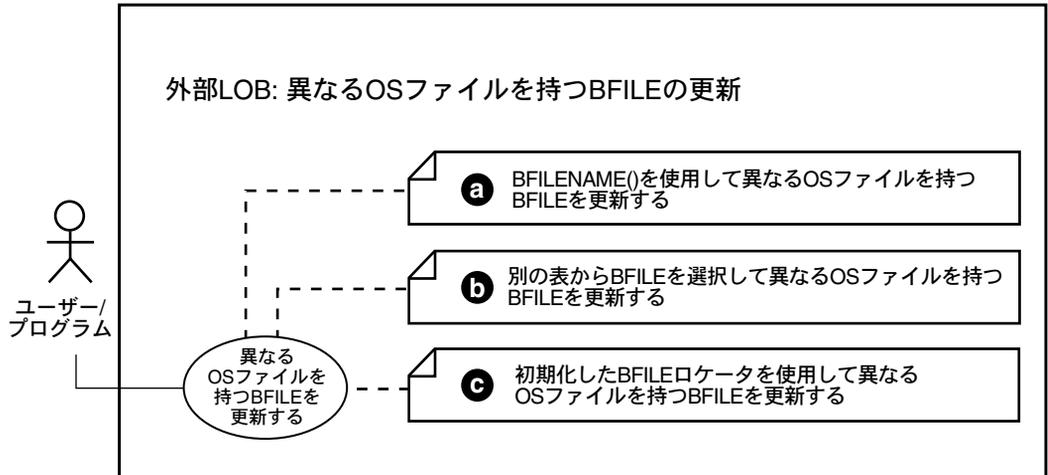
// BFILEの長さを戻します。
long length = lob_loc.length();
System.out.println("Length of BFILE: " + length);

// このBFILEのディレクトリ別名を取得します。
System.out.println("Directory alias: " + lob_loc.getDirAlias());

// このBFILEのファイル名を取得します。
System.out.println("File name: " + lob_loc.getName());
stmt.close();
conn.commit();
conn.close();
}
catch (SQLException e)
{
    e.printStackTrace();
}
}
```

BFILE を含む行を更新する 3 つの方法

図 11-29 ユースケース図：BFILE を含む行を更新する 3 つの方法



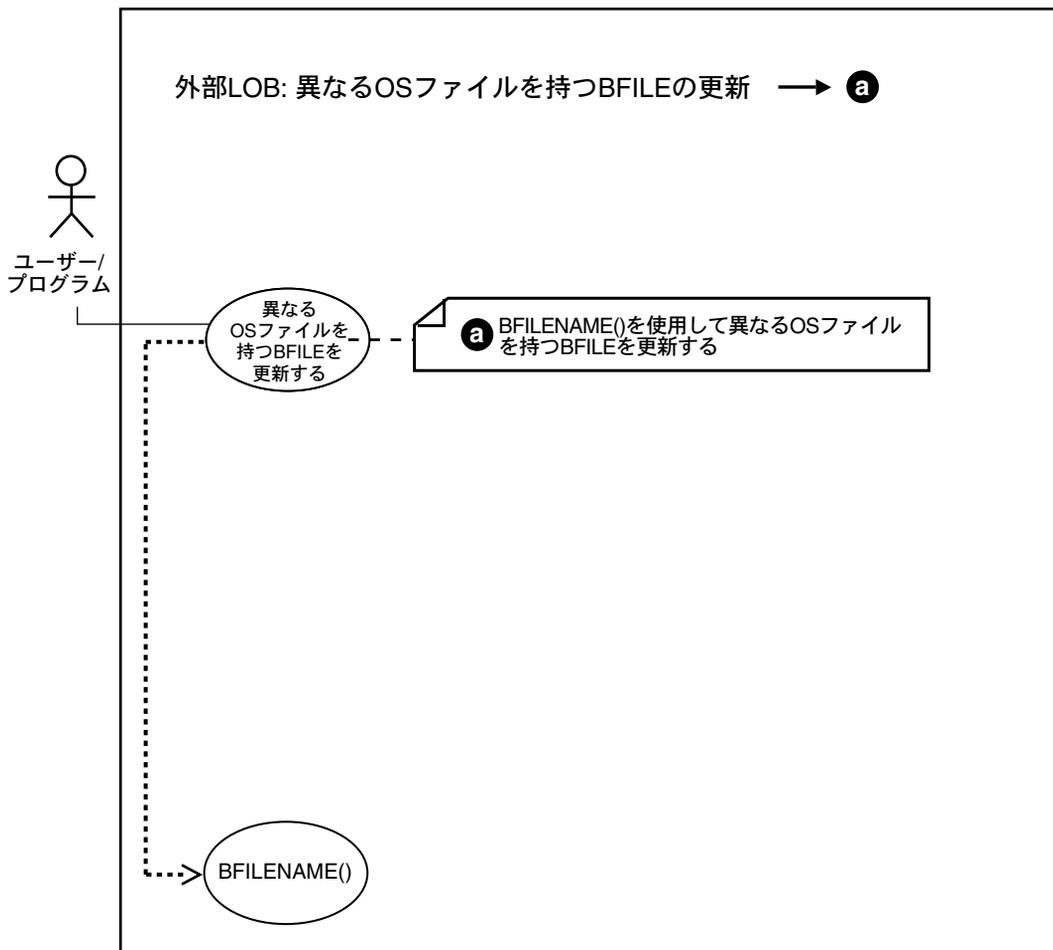
参照： 外部 LOB (BFILE) に関するすべての基本操作については、11-2 ページの「ユースケース・モデル:外部 LOB (BFILE)」を参照してください。

BFILE は、NULL、またはディレクトリ別名およびファイル名に初期化する必要があることに注意してください。

- a. [BFILENAME\(\) を使用した BFILE の更新 \(11-178 ページ\)](#)
- b. [別の表からの BFILE の選択による BFILE の更新 \(11-181 ページ\)](#)
- c. [初期化した BFILE ロケータを使用した BFILE の更新 \(11-183 ページ\)](#)

BFILENAME() を使用した BFILE の更新

図 11-30 ユースケース図：BFILENAME() を使用した BFILE の更新



参照： 外部LOB (BFILE) に関するすべての基本操作については、11-2 ページの「ユースケース・モデル：外部LOB (BFILE)」を参照してください。

使用上の注意

BFILENAME() 関数

BFILENAME() 関数は、特定の行の BFILE 列または BFILE 属性をサーバーのファイル・システム内の物理ファイルに対応付けることによって初期化する、INSERT または UPDATE の一部としてコールできます。

この関数への `directory_alias` パラメータで表される DIRECTORY オブジェクトは、BFILENAME() 関数が SQL DML または PL/SQL プログラムでコールされる前に、SQL DDL を使用して定義されている必要はありません。ただし、ディレクトリ・オブジェクトおよびオペレーティング・システム・ファイルは、実際に BFILE ロケータを使用する時点で（たとえば、`OCILobFileOpen()`、`DBMS_LOB.FILEOPEN()`、`OCILobOpen()` または `DBMS_LOB.OPEN()` のような操作にパラメータとして使用される場合）存在している必要があります。

BFILENAME() では、この DIRECTORY オブジェクトに対する権限の妥当性チェックは行われず、DIRECTORY オブジェクトが表す物理ディレクトリが実際に存在するかどうかもチェックされないことに注意してください。このようなチェックは、BFILENAME() 関数によって初期化された BFILE ロケータを使用するファイル・アクセス時に限り実行されます。

BFILE 列を初期化するために、SQL の INSERT 文および UPDATE 文の一部として、BFILENAME() を使用できます。また、BFILENAME() を使用して PL/SQL プログラムの BFILE ロケータ変数を初期化し、そのロケータをファイル操作に使用することもできます。ただし、対応するディレクトリ別名またはファイル名（あるいはその両方）が存在しない場合、この変数を使用する PL/SQL DBMS_LOB ルーチンでエラーが発生します。

BFILENAME() 関数の `directory_alias` パラメータは、ディレクトリ名の大 / 小文字を正確に指定する必要があります。

構文

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

参照： ディレクトリ名に大文字を使用する際の詳細は、11-8 ページの「[ディレクトリ名の指定](#)」を、同等の OCI ベースのルーチンの詳細は、『Oracle8i コール・インタフェース・プログラマーズ・ガイド』の第 15 章「OCI リレーショナル関数」の「LOB 関数」の `OCILobFileSetName()` を参照してください。

構文

次のマニュアルの項を参照してください。

- SQL: 『Oracle8i SQL リファレンス』の第7章「SQL文」の「UPDATE」、および第4章「関数」の「BFILENAME」

使用例

次の例では、BFILENAME 関数によって Multimedia_tab を更新します。

例

例が SQL で示されています。この例は、すべてのプログラム環境に適用されます。

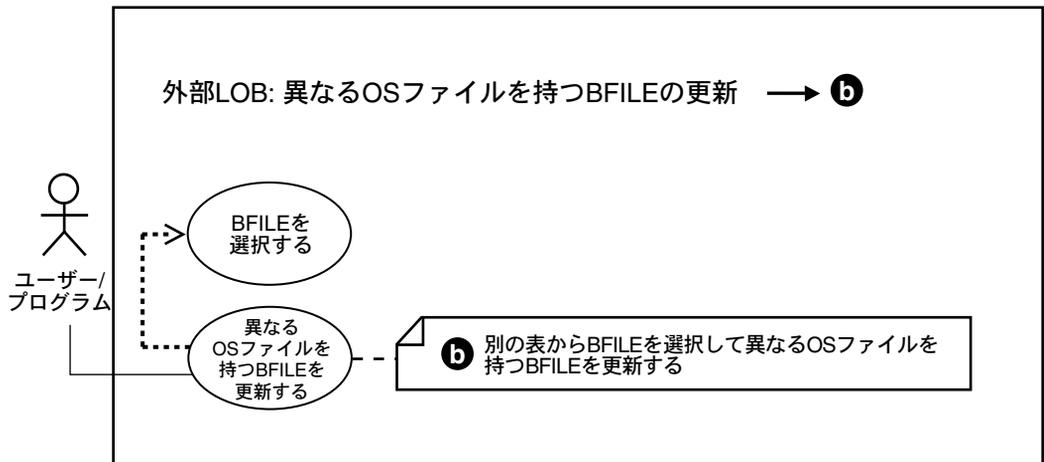
- [SQL: BFILENAME\(\) を使用した BFILE の更新](#)

SQL: BFILENAME() を使用した BFILE の更新

```
UPDATE Multimedia_tab
SET Photo = BFILENAME('PHOTO_DIR', 'Nixon_photo') where Clip_ID = 3;
```

別の表からの BFILE の選択による BFILE の更新

図 11-31 ユースケース図：別の表からの BFILE の選択による BFILE の更新



参照： 外部LOB (BFILE) に関するすべての基本操作については、11-2 ページの「ユースケース・モデル: 外部LOB (BFILE)」を参照してください。

用途

別の表から BFILE を選択することによって、BFILE を更新します。

使用上の注意

BFILE にはコピー機能がないため、BFILE をある場所から別の場所にコピーする場合は、別の表から選択した BFILE を更新に使用する必要があります。BFILE は、コピー・セマンティクスのかわりに参照セマンティクスを使用するため、BFILE ロケータのみがある行から別の行にコピーされます。これは、オペレーティング・システムのコマンドを発行して、そのオペレーティング・システム・ファイルをコピーしなければ、外部LOBの値のコピーを作成できないことを意味します。

構文

次のマニュアルの項を参照してください。

- SQL: 『Oracle8i SQL リファレンス』の第7章「SQL文」の「UPDATE」

使用例

次の例では、アーカイブ記憶表 VoiceoverLib_tab から選択することによって、Voiceover_tab 表を更新します。

例

例が SQL で示されています。この例は、すべてのプログラム環境に適用されます。

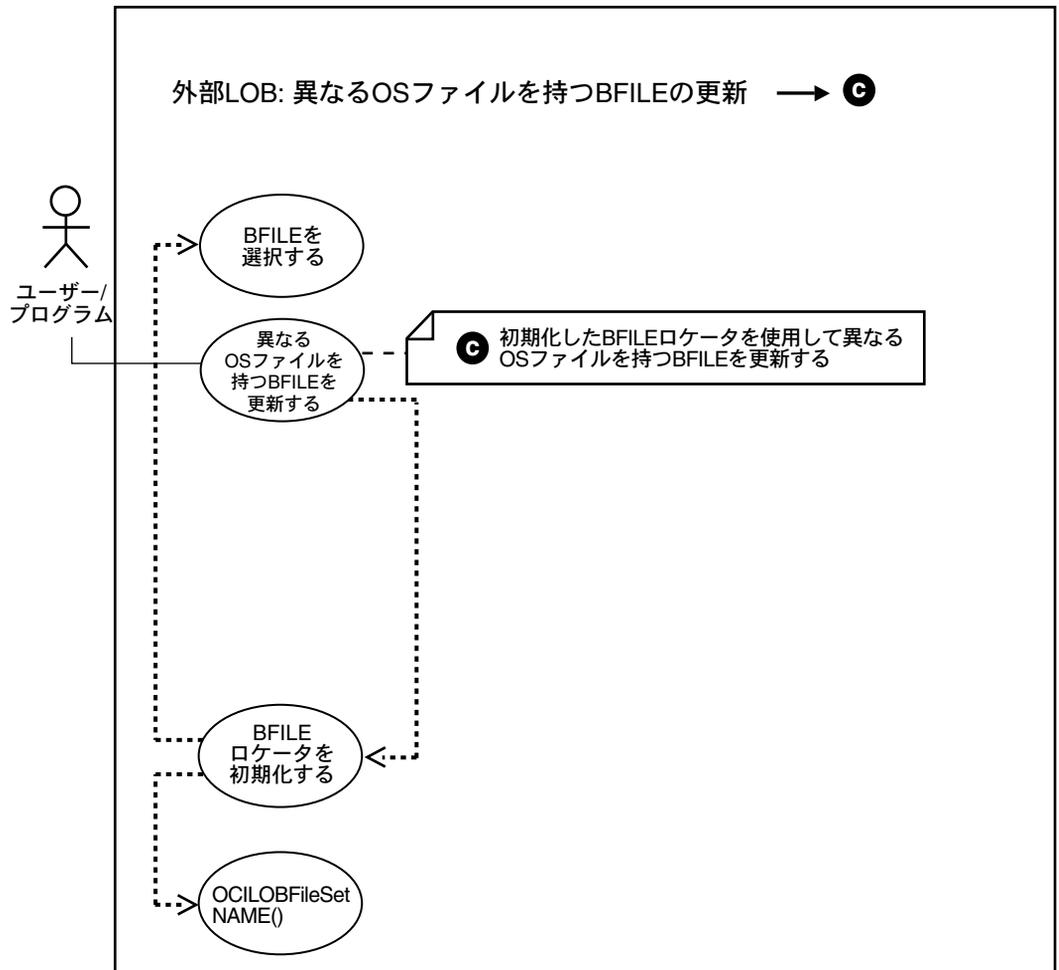
- [SQL: 別の表からの BFILE の選択による BFILE の更新](#)

SQL: 別の表からの BFILE の選択による BFILE の更新

```
UPDATE Voiceover_tab
   SET (originator,script,actor,take,recording) =
       (SELECT * FROM VoiceoverLib_tab VLtab WHERE VLtab.Take = 101);
```

初期化した BFILE ロケータを使用した BFILE の更新

図 11-32 ユースケース図：初期化した BFILE ロケータを使用した BFILE の更新



参照： 外部LOB (BFILE) に関するすべての基本操作については、11-2ページの「ユースケース・モデル:外部LOB (BFILE)」を参照してください。

用途

初期化した BFILE ロケータを使用して、BFILE を更新します。

使用上の注意

UPDATE 文を発行する前に、BFILE ロケータのバインド変数をディレクトリ別名およびファイル名に初期化する必要があります。

構文

各プログラム環境で使用可能な機能のリストは、[第 3 章「LOB プログラム環境」](#)を参照してください。各プログラム環境における構文については、次のマニュアルの項を参照してください。

- PL/SQL (DBMS_LOB) : 『Oracle8i SQL リファレンス』の第 7 章「SQL 文」の「UPDATE」
- C (OCI) : 『Oracle8i コール・インタフェース・プログラマーズ・ガイド』の第 7 章「LOB および FILE 操作」の使用上の注意、および第 15 章「OCI リレーショナル関数」の「LOB 関数」の OCILobFileSetName()
- COBOL (Pro*COBOL) : 『Oracle8i Pro*COBOL プリコンパイラ・プログラマーズ・ガイド』の第 13 章「ラージ・オブジェクト (LOB)」の「LOB 文」の使用上の注意、付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「ALLOCATE (実行可能埋込み SQL 拡張要素)」、および『Oracle8i SQL リファレンス』の第 7 章「SQL 文」の「UPDATE」
- C/C++ (Pro*C/C++) : 『Oracle8i Pro*C/C++ プリコンパイラ・プログラマーズ・ガイド』の第 16 章「ラージ・オブジェクト (LOB)」の「LOB 文」の使用上の注意、付録 F「埋込み SQL 文およびプリコンパイラ宣言文」、および『Oracle8i SQL リファレンス』の第 7 章「SQL 文」の「UPDATE」
- Visual Basic (Oracle Objects for OLE オンライン・ヘルプ): 「Help」の「Contents」タブから、「OO4O Automation Server」>「OBJECTS」>「OraBfile」>「PROPERTIES」>「DirectoryFileName」、「FileName」、および「OBJECTS」>「OraDatabase」>「METHODS」>「ExecuteSQL」を選択してください。または「OBJECTS」>「OraBfile」、「OraDatabase」>「Examples」を選択してください。
- Java (JDBC) : 『Oracle8i JDBC 開発者ガイドおよびリファレンス』の第 7 章「LOB と BFILE の操作」の「BLOB と CLOB の操作」の「BLOB または CLOB 列の作成と移入」、および『Oracle8i SQLJ 開発者ガイドおよびリファレンス』の第 5 章「型のサポート」の「Oracle の拡張型」の「BLOB、CLOB および BFILE のサポート」

使用例

ありません。

例

次のプログラム環境の例が示されています。

- [PL/SQL: 初期化した BFILE ロケータを使用した BFILE の更新](#) (11-183 ページ)
- [C \(OCI\) : 初期化した BFILE ロケータを使用した BFILE の更新](#) (11-185 ページ)
- [COBOL \(Pro*COBOL\) : 初期化した BFILE ロケータを使用した BFILE の更新](#) (11-186 ページ)
- [C/C++ \(Pro*C/C++\) : 初期化した BFILE ロケータを使用した BFILE の更新](#) (11-188 ページ)
- [Visual Basic \(OO4O\) : 初期化した BFILE ロケータを使用した BFILE の更新](#) (11-188 ページ)
- [Java \(JDBC\) : 初期化した BFILE ロケータを使用した BFILE の更新](#) (11-189 ページ)

PL/SQL: 初期化した BFILE ロケータを使用した BFILE の更新

```

/* 例のプロシージャ updateUseBindVariable_proc は、DBMS_LOB パッケージの
一部ではないことに注意してください。*/
CREATE OR REPLACE PROCEDURE updateUseBindVariable_proc (Lob_loc BFILE) IS
BEGIN
    UPDATE Multimedia_tab SET Photo = Lob_loc WHERE Clip_ID = 3;
END;

DECLARE
    Lob_loc BFILE;
BEGIN
    SELECT Photo INTO Lob_loc
    FROM Multimedia_tab
    WHERE Clip_ID = 1;
    updateUseBindVariable_proc (Lob_loc);
    COMMIT;
END;

```

C (OCI) : 初期化した BFILE ロケータを使用した BFILE の更新

```

void BfileUpdate(envhp, errhp, svchp, stmthp)
OCIEnv *envhp;
OCIError *errhp;
OCISvcCtx *svchp;
OCIStmt *stmthp;
{
    OCILobLocator *Lob_loc;
    OCIBind *bndhp;

```

```
text *updstmt =
    (text *) "UPDATE Multimedia_tab SET Photo = :Lob_loc WHERE Clip_ID = 1";

OraText *Dir = (OraText *) "PHOTO_DIR", *Name = (OraText *) "Washington_photo";

/* SQL 文を準備します。*/
checkerr (errhp, OCIStmtPrepare(stmthp, errhp, updstmt, (ub4)
                                strlen((char *) updstmt),
                                (ub4) OCI_NTV_SYNTAX, (ub4) OCI_DEFAULT));

/* ロケータ・リソースを割り当てます。*/
(void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &Lob_loc,
                          (ub4) OCI_DTYPE_FILE, (size_t) 0, (dvoid **) 0);

checkerr (errhp, OCILobFileSetName(envhp, errhp, &Lob_loc,
                                   Dir, (ub2) strlen((char *) Dir),
                                   Name, (ub2) strlen((char *) Name)));

checkerr (errhp, OCIBindByPos(stmthp, &bndhp, errhp, (ub4) 1,
                              (dvoid *) &Lob_loc, (sb4) 0, SQLT_BFILE,
                              (dvoid *) 0, (ub2 *) 0, (ub2 *) 0,
                              (ub4) 0, (ub4 *) 0, (ub4) OCI_DEFAULT));

/* SQL 文を実行します。*/
checkerr (errhp, OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                               (CONST OCISnapshot*) 0, (OCISnapshot*) 0,
                               (ub4) OCI_DEFAULT));

/* LOB リソースを解放します。*/
OCIDescriptorFree((dvoid *) Lob_loc, (ub4) OCI_DTYPE_FILE);
}
```

COBOL (Pro*COBOL) : 初期化した BFILE ロケータを使用した BFILE の更新

```
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.

* BFILE ロケータの割当ておよび初期化を行います。
EXEC SQL ALLOCATE :BFILE1 END-EXEC.

* BFILE を移入します。
EXEC SQL WHENEVER NOT FOUND GOTO END-OF-BFILE END-EXEC.
EXEC ORACLE OPTION (SELECT_ERROR=NO) END-EXEC.
EXEC SQL
    SELECT PHOTO INTO :BFILE1:BFILE-IND
    FROM MULTIMEDIA_TAB WHERE CLIP_ID = 1 END-EXEC.

* clip_id=3 に対応付けられた写真を clip_id=1 と同じにします。
EXEC SQL
    UPDATE MULTIMEDIA_TAB SET PHOTO = :BFILE1:BFILE-IND
    WHERE CLIP_ID = 3 END-EXEC.

* BFILE を解放します。
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++) : 初期化した BFILE ロケータを使用した BFILE の更新

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

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

void updateUseBindVariable_proc(Lob_loc)
    OCIBFileLocator *Lob_loc;
{
    EXEC SQL WHENEVER SQLERROR DO Sample_Error();
    EXEC SQL UPDATE Multimedia_tab SET Photo = :Lob_loc WHERE Clip_ID = 3;
}

void updateBFILE_proc()
{
    OCIBFileLocator *Lob_loc;

    EXEC SQL ALLOCATE :Lob_loc;
    EXEC SQL SELECT Photo INTO :Lob_loc
        FROM Multimedia_tab WHERE Clip_ID = 1;
    updateUseBindVariable_proc(Lob_loc);
    EXEC SQL FREE :Lob_loc;
}

void main()
{
    char *samp = "samp/samp";
    EXEC SQL CONNECT :samp;
    updateBFILE_proc();
    EXEC SQL ROLLBACK WORK RELEASE;
}
```

Visual Basic (OO4O) : 初期化した BFILE ロケータを使用した BFILE の更新

```
Dim MySession As OraSession
Dim OraDb As OraDatabase
Dim OraParameters As OraParameters, OraPhoto As OraBfile
```

```

Set MySession = CreateObject("OracleInProcServer.XOraSession")
Set OraDb = MySession.OpenDatabase("exampledb", "scott/tiger", 0&)

OraDb.Connection.BeginTrans

Set OraParameters = OraDb.Parameters

' BFILE 型の IN OUT パラメータを定義します。
OraParameters.Add "MyPhoto", Null, ORAPARM_BOTH, ORATYPE_BFILE

' BFILE 型の OUT パラメータを定義します。
OraDb.ExecuteSQL (
    "BEGIN SELECT Photo INTO :MyPhoto FROM Multimedia_tab WHERE Clip_ID = 1;
    END;")

' clip_id=1 の写真 BFILE を clip_id=1001 に更新します。
OraDb.ExecuteSQL (
    "UPDATE Multimedia_tab SET Photo = :MyPhoto WHERE Clip_ID = 1001")

' ディレクトリ別名およびファイル名を取得します。
'MsgBox " Directory alias is " & OraMusic1.DirectoryName & " Filename is " &
OraMusic1.filename

OraDb.Connection.CommitTrans

```

Java (JDBC) : 初期化した BFILE ロケータを使用した BFILE の更新

```

import java.io.InputStream;
import java.io.OutputStream;

// コア JDBC クラス :
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 固有の JDBC クラス :
import oracle.sql.*;
import oracle.jdbc.driver.*;

public class Ex4_100
{

    public static void main (String args [])

```

```

        throws Exception
    {
        // Oracle JDBC Driver をロードします。
        DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

        // データベースに接続します。
        Connection conn =
            DriverManager.getConnection ("jdbc:oracle:oci8:@", "samp", "samp");

        conn.setAutoCommit (false);

        // 文を作成します。
        Statement stmt = conn.createStatement ();

        try
        {
            BFILE src_lob = null;
            ResultSet rset = null;
            OraclePreparedStatement pstmt = null;

            rset = stmt.executeQuery (
                "SELECT photo FROM multimedia_tab WHERE clip_id = 3");
            if (rset.next())
            {
                src_lob = ((OracleResultSet)rset).getBFILE (1);
            }

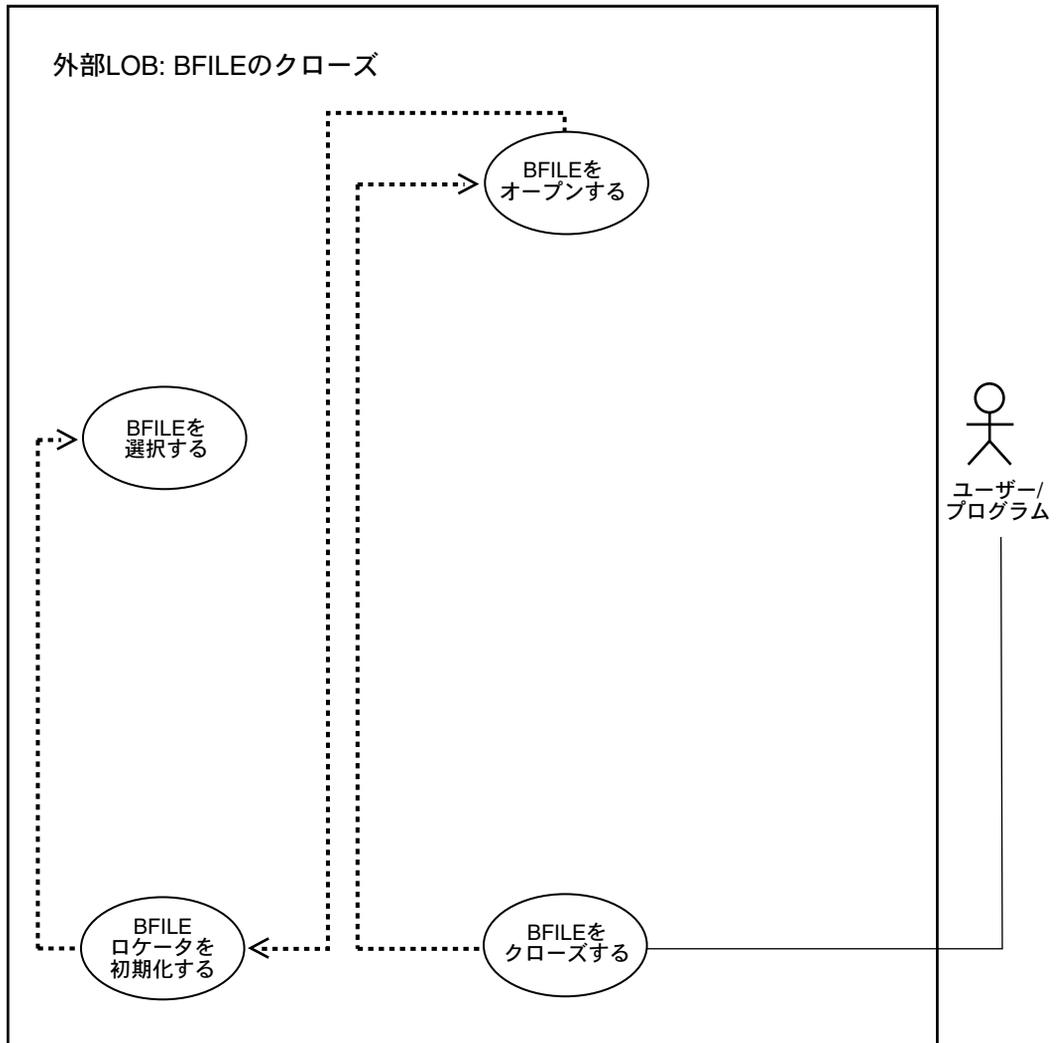
            // CallableStatement を準備し、LOB を READWRITE でオープンします。
            pstmt = (OraclePreparedStatement) conn.prepareStatement (
                "UPDATE multimedia_tab SET photo = ? WHERE clip_id = 1");
            pstmt.setBFILE(1, src_lob);
            pstmt.execute();

            // 文をクローズし、トランザクションをコミットします。
            stmt.close();
            pstmt.close();
            conn.commit();
            conn.close();
        }
        catch (SQLException e)
        {
            e.printStackTrace();
        }
    }
}

```

BFILE をクローズする 2 つの方法

図 11-33 ユースケース図: BFILE をクローズする 2 つの方法



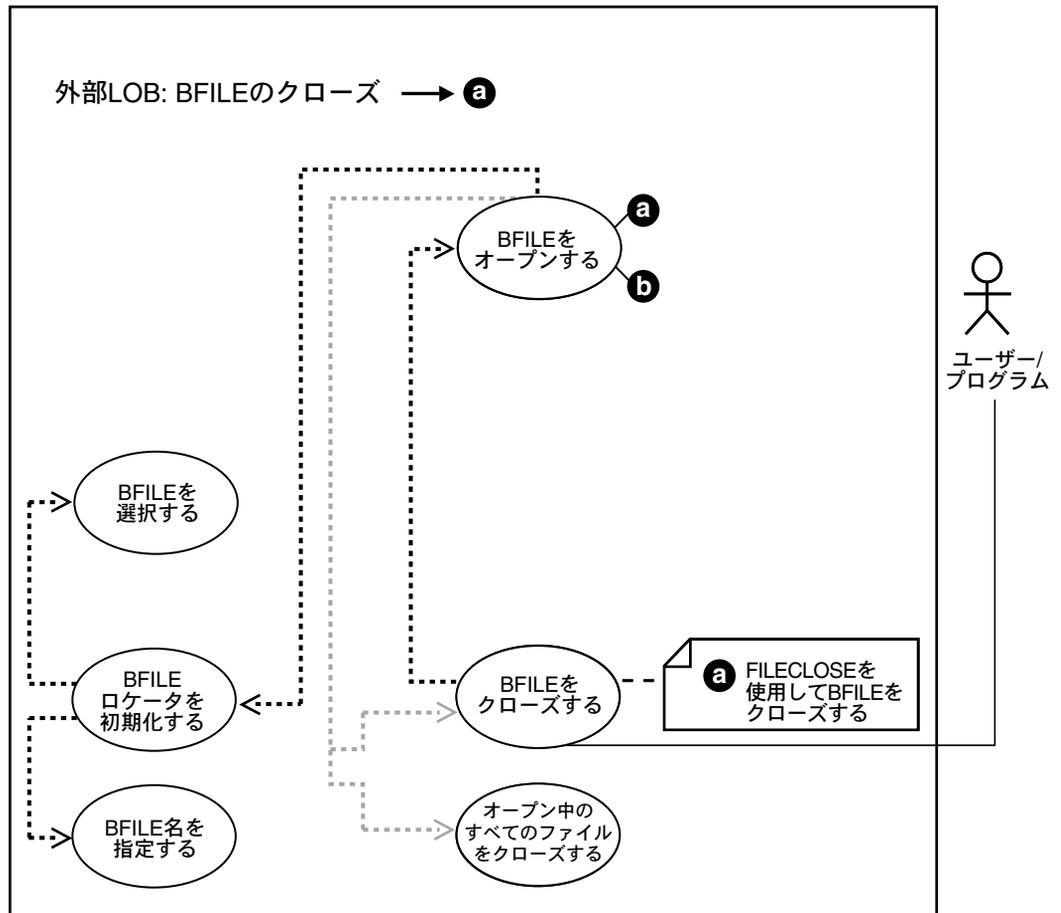
参照: 外部LOB (BFILE) に関するすべての基本操作については、11-2ページの「ユースケース・モデル: 外部LOB (BFILE)」を参照してください。

コードを比較するとわかるように、これら 2 つの方法は、よく似ています。ただし、古い FILECLOSE 形式の使用を継続することもできますが、CLOSE を使用すると将来の拡張性が向上するため、CLOSE の使用に切り替えることをお勧めします。

- a. [FILECLOSE を使用した BFILE のクローズ](#) (11-193 ページ)
- b. [CLOSE を使用した BFILE のクローズ](#) (11-198 ページ)

FILECLOSE を使用した BFILE のクローズ

図 11-34 ユースケース図：FILECLOSE を使用した BFILE のクローズ



参照： 外部LOB (BFILE) に関するすべての基本操作については、11-2ページの「ユースケース・モデル:外部LOB (BFILE)」を参照してください。

用途

FILECLOSE を使用して BFILE をクローズします。

使用上の注意

ありません。

構文

各プログラム環境で使用可能な機能のリストは、[第 3 章「LOB プログラム環境」](#)を参照してください。各プログラム環境における構文については、次のマニュアルの項を参照してください。

- PL/SQL (DBMS_LOB) : 『Oracle8i PL/SQL パッケージ・プロシージャ・リファレンス』の第 18 章「DBMS_LOB」の「サブプログラムの要約」の FILEOPEN プロシージャ、FILECLOSE プロシージャ
- C (OCI) : 『Oracle8i コール・インタフェース・プログラマーズ・ガイド』の第 7 章「LOB および FILE 操作」の使用上の注意、および第 15 章「OCI リレーショナル関数」の「LOB 関数」の OCILobFileClose()
- COBOL (Pro*COBOL) : 参照マニュアルはありません。
- C/C++ (Pro*C/C++) : 参照マニュアルはありません。
- Visual Basic (OO4O) : 参照マニュアルはありません。
- Java (JDBC) : 『Oracle8i JDBC 開発者ガイドおよびリファレンス』の第 7 章「LOB と BFILE の操作」の「BLOB と CLOB の操作」の「BLOB または CLOB 列の作成と移入」、および『Oracle8i SQLJ 開発者ガイドおよびリファレンス』の第 5 章「型のサポート」の「Oracle の拡張型」の「BLOB、CLOB および BFILE のサポート」

使用例

古い FILECLOSE 形式の使用を継続することもできますが、CLOSE を使用すると将来の拡張性が向上するため、CLOSE の使用に切り替えることをお勧めします。この例は、BFILE のオープンとの例と関連しています。

例

次の各プログラム環境の例が示されています。

- [PL/SQL \(DBMS_LOB パッケージ\) : FILECLOSE を使用した BFILE のクローズ](#) (11-193 ページ)
- [C \(OCI\) : FILECLOSE を使用した BFILE のクローズ](#) (11-193 ページ)
- COBOL (Pro*COBOL) : 今回のリリースでは例は記載されていません。
- C/C++ (Pro*C/C++) : 今回のリリースでは例は記載されていません。
- Visual Basic (OO4O) : 例はありません。11-196 ページの「注意」を参照してください。

- [Java \(JDBC\) : FILECLOSE を使用した BFILE のクローズ \(11-196 ページ\)](#)

PL/SQL (DBMS_LOB パッケージ) : FILECLOSE を使用した BFILE のクローズ

```

/* 例のプロシージャ closeBFILE_procOne は、DBMS_LOB パッケージの
   一部ではないことに注意してください。*/
CREATE OR REPLACE PROCEDURE closeBFILE_procOne IS
  Lob_loc    BFILE := BFILENAME('PHOTO_DIR', 'Lincoln_photo');
BEGIN
  DBMS_LOB.FILEOPEN(Lob_loc, DBMS_LOB.FILE_READONLY);
  /* 何らかの処理を行います。*/
  DBMS_LOB.FILECLOSE(Lob_loc);
END;

```

C (OCI) : FILECLOSE を使用した BFILE のクローズ

```

void BfileFileClose(envhp, errhp, svchp, stmthp)
OCIEnv      *envhp;
OCIError    *errhp;
OCISvcCtx   *svchp;
OCIStmt     *stmthp;
{
  OCILobLocator *bfile_loc;

  /* ロケータ記述子を割り当てます。*/
  (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &bfile_loc,
                           (ub4) OCI_DTYPE_FILE,
                           (size_t) 0, (dvoid **) 0);

  checkerr(errhp, OCILobFileSetName(envhp, errhp, &bfile_loc,
                                    (OraText *) "PHOTO_DIR", (ub2) strlen("PHOTO_DIR"),
                                    (OraText *) "Lincoln_photo",
                                    (ub2) strlen("Lincoln_photo")));

  checkerr(errhp, OCILobFileOpen(svchp, errhp, bfile_loc,
                                 (ub1) OCI_FILE_READONLY));

  checkerr(errhp, OCILobFileClose(svchp, errhp, bfile_loc));

  /* ロケータ記述子を解放します。*/
  OCIDescriptorFree((dvoid *)bfile_loc, (ub4)OCI_DTYPE_FILE);
}

```

Visual Basic (OO4O) : FILECLOSE を使用した BFILE のクローズ

注意： 現在、OO4O では、CLOSE による BFILE のクローズ (次を参照) のみが提供されています。

Java (JDBC) : FILECLOSE を使用した BFILE のクローズ

```
import java.io.InputStream;
import java.io.OutputStream;

// コア JDBC クラス :
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 固有の JDBC クラス :
import oracle.sql.*;
import oracle.jdbc.driver.*;
public class Ex4_45
{
    public static void main (String args [])
        throws Exception
    {
        // Oracle JDBC Driver をロードします。
        DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

        // データベースに接続します。
        Connection conn =
            DriverManager.getConnection ("jdbc:oracle:oci8:@", "samp", "samp");

        conn.setAutoCommit (false);

        // 文を作成します。
        Statement stmt = conn.createStatement ();
```

```
try
{
    BFILE src_lob = null;
    ResultSet rset = null;
    boolean result = false;

    rset = stmt.executeQuery (
        "SELECT BFILENAME('PHOTO_DIR', 'Lincoln_photo') 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);

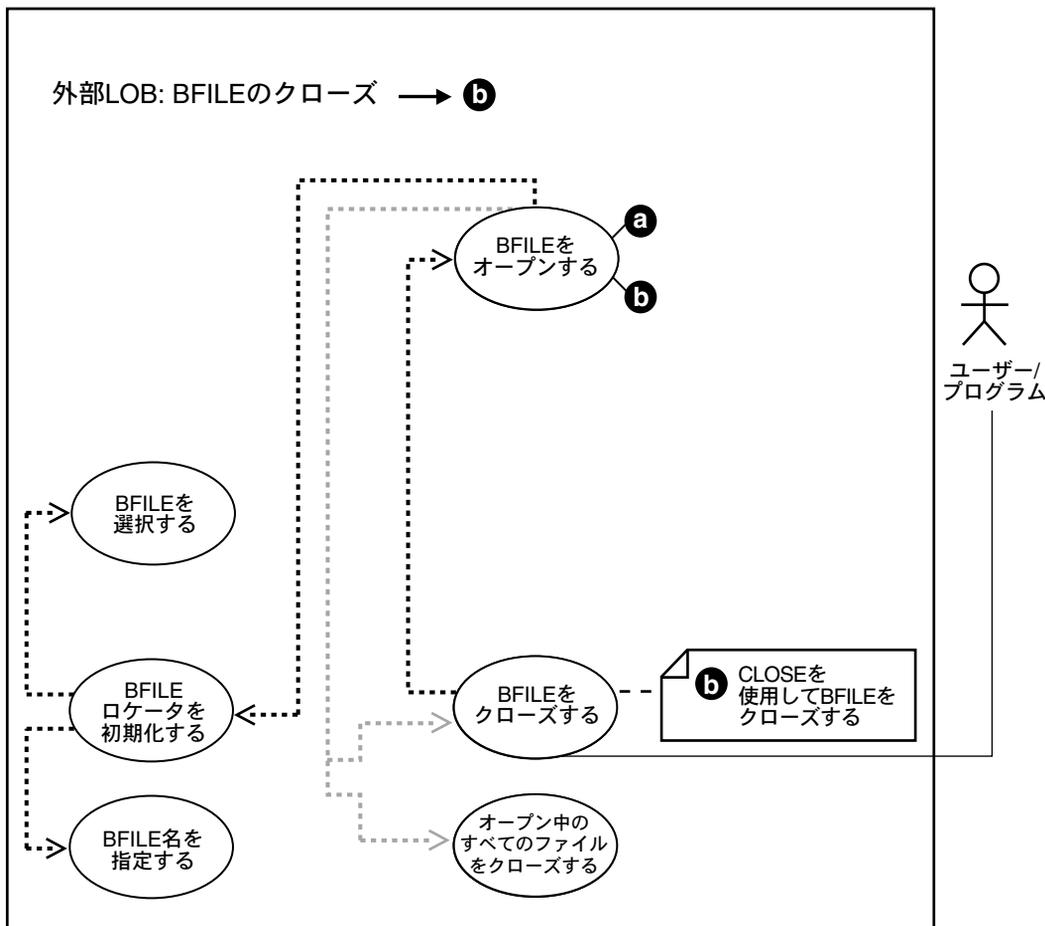
    src_lob.openFile();

    result = src_lob.isFileOpen();
    System.out.println(
        "result of fileIsOpen() after opening file : " + result);

    // BFILE、文および接続をクローズします。
    src_lob.closeFile();
    stmt.close();
    conn.commit();
    conn.close();
}
catch (SQLException e)
{
    e.printStackTrace();
}
}
```

CLOSE を使用した BFILE のクローズ

図 11-35 ユースケース図：CLOSE を使用したオープン中の BFILE のクローズ



参照： 外部 LOB (BFILE) に関するすべての基本操作については、11-2 ページの「ユースケース・モデル:外部 LOB (BFILE)」を参照してください。

用途

CLOSE を使用して BFILE をクローズします。

使用上の注意

ありません。

構文

各プログラム環境で使用可能な機能のリストは、第3章「LOB プログラム環境」を参照してください。各プログラム環境における構文については、次のマニュアルの項を参照してください。

- PL/SQL (DBMS_LOB) : 『Oracle8i PL/SQL パッケージ・プロシージャ・リファレンス』の第18章「DBMS_LOB」の「サブプログラムの要約」のCLOSE プロシージャ
- C (OCI) : 『Oracle8i コール・インタフェース・プログラマーズ・ガイド』の第7章「LOB および FILE 操作」の使用上の注意、および第15章「OCI リレーショナル関数」の「LOB 関数」の OCILobClose()
- COBOL (Pro*COBOL) : 『Oracle8i Pro*COBOL プリコンパイラ・プログラマーズ・ガイド』の第13章「ラージ・オブジェクト (LOB)」の「LOB 文」の使用上の注意、および付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB CLOSE (実行可能埋込み SQL 拡張要素)」
- C/C++ (Pro*C/C++) : 『Oracle8i Pro*C/C++ プリコンパイラ・プログラマーズ・ガイド』の第16章「ラージ・オブジェクト (LOB)」の「LOB 文」の使用上の注意、および付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB CLOSE (実行可能埋込み SQL 拡張要素)」
- Visual Basic (Oracle Objects for OLE オンライン・ヘルプ): 「Help」の「Contents」タブから、「OO4O Automation Server」>「OBJECTS」>「OraBfile」>「METHODS」>「IsOpen」、「Close」、および「OBJECTS」>「OraBfile」>「Examples」を選択してください。
- Java (JDBC) : 『Oracle8i JDBC 開発者ガイドおよびリファレンス』の第7章「LOB と BFILE の操作」の「BLOB と CLOB の操作」の「BLOB または CLOB 列の作成と移入」、および『Oracle8i SQLJ 開発者ガイドおよびリファレンス』の第5章「型のサポート」の「Oracle の拡張型」の「BLOB、CLOB および BFILE のサポート」

使用例

次の例は、BFILE のオープンの例と関連しています。ここでは、Lincoln_photo と対応付けられた BFILE をクローズします。

例

- PL/SQL (DBMS_LOB パッケージ) : CLOSE を使用した BFILE のクローズ (11-200 ページ)
- C (OCI) : CLOSE を使用した BFILE のクローズ (11-200 ページ)

- [COBOL \(Pro*COBOL\) : CLOSE を使用した BFILE のクローズ \(11-201 ページ\)](#)
- [C/C++ \(Pro*C/C++\) : CLOSE を使用した BFILE のクローズ \(11-202 ページ\)](#)
- [Visual Basic \(OO4O\) : CLOSE を使用した BFILE のクローズ \(11-203 ページ\)](#)
- [Java \(JDBC\) : CLOSE を使用した BFILE のクローズ \(11-203 ページ\)](#)

PL/SQL (DBMS_LOB パッケージ) : CLOSE を使用した BFILE のクローズ

```
/* 例のプロシージャ closeBFILE_procTwo は、DBMS_LOB パッケージの
一部ではないことに注意してください。*/
CREATE OR REPLACE PROCEDURE closeBFILE_procTwo IS
  Lob_loc   BFILE := BFILENAME('PHOTO_DIR', 'Lincoln_photo');
BEGIN
  DBMS_LOB.OPEN(Lob_loc, DBMS_LOB.LOB_READONLY);
  /* 何らかの処理を行います。*/
  DBMS_LOB.CLOSE(Lob_loc);
END;
```

C (OCI) : CLOSE を使用した BFILE のクローズ

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

  OCILobLocator *bfile_loc;

  /* ロケータ記述子を割り当てます。*/
  (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &bfile_loc,
                           (ub4) OCI_DTYPE_FILE,
                           (size_t) 0, (dvoid **) 0);

  checkerr(errhp, OCILobFileSetName(envhp, errhp, &bfile_loc,
                                    (OraText *) "PHOTO_DIR", (ub2) strlen("PHOTO_DIR"),
                                    (OraText *) "Lincoln_photo",
                                    (ub2) strlen("Lincoln_photo")));

  checkerr(errhp, OCILobOpen(svchp, errhp, bfile_loc,
                             (ub1) OCI_LOB_READONLY));

  checkerr(errhp, OCILobClose(svchp, errhp, bfile_loc));
```

```

/* ロケータ記述子を解放します。*/
OCIDescriptorFree((dvoid *)bfile_loc, (ub4)OCI_DTYPE_FILE);
}

```

COBOL (Pro*COBOL) : CLOSE を使用した BFILE のクローズ

```

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.

* BFILE ロケータの割当ておよび初期化を行います。
EXEC SQL ALLOCATE :BFILE1 END-EXEC.

* ディレクトリおよびファイル情報を設定します。
MOVE "PHOTO_DIR" TO DIR-ALIAS-ARR.
MOVE 9 TO DIR-ALIAS-LEN.
MOVE "Lincoln_photo" TO FNAME-ARR.
MOVE 13 TO FNAME-LEN.

EXEC SQL
LOB FILE SET :BFILE1
DIRECTORY = :DIR-ALIAS, FILENAME = :FNAME END-EXEC.

EXEC SQL
LOB OPEN :BFILE1 READ ONLY END-EXEC.

* LOB をクローズします。
EXEC SQL LOB CLOSE :BFILE1 END-EXEC.

```

```
* LOB ロケータを解放します。
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++) : CLOSE を使用した BFILE のクローズ

/* Pro*C/C++ では、BFILE に対する CLOSE のフォームが 1 つしかありません。
FILE CLOSE 文はありません。そのかわりに、単純な CLOSE 文が使用されます。*/

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

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

void closeBFILE_proc()
{
OCIBFileLocator *Lob_loc;
char *Dir = "PHOTO_DIR", *Name = "Lincoln_photo";

EXEC SQL WHENEVER SQLERROR DO Sample_Error();
EXEC SQL ALLOCATE :Lob_loc;
EXEC SQL LOB FILE SET :Lob_loc DIRECTORY = :Dir, FILENAME = :Name;
EXEC SQL LOB OPEN :Lob_loc READ ONLY;
/* 何らかの処理を行います。*/
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 (OO40) : CLOSE を使用した BFILE のクローズ

```

Dim MySession As OraSession
Dim OraDb As OraDatabase

Dim OraDyn As OraDynaset, OraMusic As OraBfile, amount_read%, chunksize%, chunk

Set MySession = CreateObject("OracleInProcServer.XOraSession")
Set OraDb = MySession.OpenDatabase("exampledb", "scott/tiger", 0&)

chunksize = 32767
Set OraDyn = OraDb.CreateDynaset("select * from Multimedia_tab", ORADYN_DEFAULT)
Set OraMusic = OraDyn.Fields("Music").Value

If OraMusic.IsOpen Then
    ' ファイルがオープンされている場合の処理が行われます。
    OraMusic.Close
End If

```

Java (JDBC) : CLOSE を使用した BFILE のクローズ

```

import java.io.InputStream;
import java.io.OutputStream;

// コア JDBC クラス :
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 固有の JDBC クラス :
import oracle.sql.*;
import oracle.jdbc.driver.*;

public class Ex4_48
{
    public static void main (String args [])
        throws Exception

```

```
{
// Oracle JDBC Driver をロードします。
DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

// データベースに接続します。
Connection conn =
    DriverManager.getConnection ("jdbc:oracle:oci8:@", "samp", "samp");

conn.setAutoCommit (false);

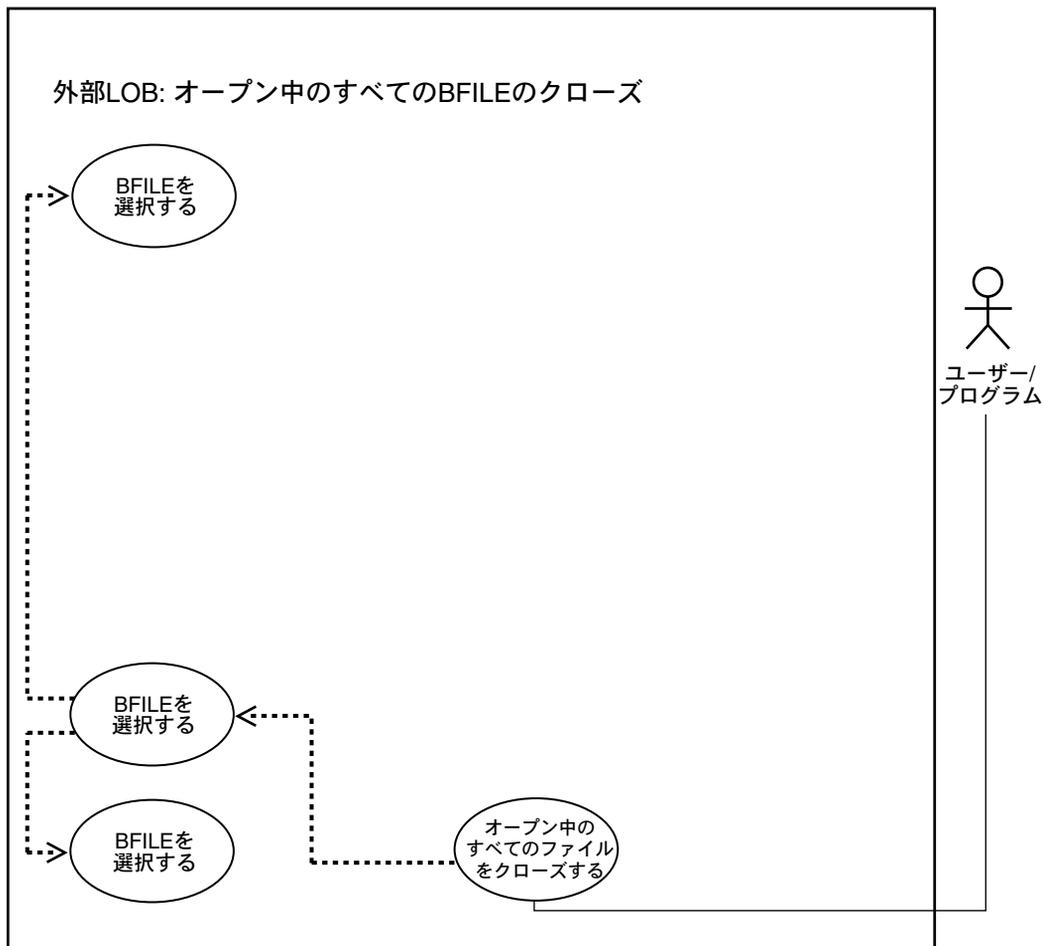
// 文を作成します。
Statement stmt = conn.createStatement ();
try
{
    BFILE src_lob = null;
    ResultSet rset = null;

    rset = stmt.executeQuery (
        "SELECT BFILENAME('PHOTO_DIR', 'Lincoln_photo') 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");
    }

    // BFILE、文および接続をクローズします。
    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();
}
}
```

オープン中のすべての BFILE のクローズ

図 11-36 ユースケース図：オープン中のすべての BFILE のクローズ



参照： 外部 LOB (BFILE) に関するすべての基本操作については、11-2 ページの「ユースケース・モデル:外部 LOB (BFILE)」を参照してください。

PL/SQL プログラム・ブロックまたは OCI プログラムが正常終了または異常終了した後で、オープンしているすべてのファイルをクローズすることは、ユーザーの責任で行う必要があります。このため、たとえばある BFILE に対するすべての DBMS_LOB.FILEOPEN() コールまたは DBMS_LOB.OPEN() コールに対して、対応する DBMS_LOB.FILECLOSE() コールまたは DBMS_LOB.CLOSE() コールがある必要があります。PL/SQL ブロックまたは OCI プログラムの終了前、およびエラーの発生時に、オープン・ファイルをクローズする必要があります。例外ハンドラは、例外または異常終了の発生に備えて、オープンされていたファイルをクローズできる必要があります。

これが行われない場合、Oracle は、それらのファイルはクローズされていないとみなしません。

参照： 11-56 ページの「[BFILE の最大オープン数の指定：SESSION_MAX_OPEN_FILES](#)」を参照してください。

用途

すべての BFILE をクローズします。

使用上の注意

ありません。

構文

各プログラム環境で使用可能な機能のリストは、[第 3 章「LOB プログラム環境」](#)を参照してください。各プログラム環境における構文については、次のマニュアルの項を参照してください。

- PL/SQL (DBMS_LOB) : 『Oracle8i PL/SQL パッケージ・プロシージャ・リファレンス』の第 18 章「DBMS_LOB」の「サブプログラムの要約」の FILECLOSEALL プロシージャ
- C (OCI) : 『Oracle8i コール・インタフェース・プログラマーズ・ガイド』の第 7 章「LOB および FILE 操作」の使用上の注意、および第 15 章「OCI リレーショナル関数」の「LOB 関数」の OCILobFileCloseAll()
- COBOL (Pro*COBOL) : 『Oracle8i Pro*COBOL プリコンパイラ・プログラマーズ・ガイド』の第 13 章「ラージ・オブジェクト (LOB)」の「LOB 文」の使用上の注意、および付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB FILE CLOSE ALL (実行可能埋込み SQL 拡張要素)」
- C/C++ (Pro*C/C++) : 『Oracle8i Pro*C/C++ プリコンパイラ・プログラマーズ・ガイド』の第 16 章「ラージ・オブジェクト (LOB)」の「LOB 文」の使用上の注意、および付録 F「埋込み SQL 文およびプリコンパイラ宣言文」の「LOB FILE CLOSE ALL (実行可能埋込み SQL 拡張要素)」

- Visual Basic (Oracle Objects for OLE オンライン・ヘルプ): 「Help」の「Contents」タブから、「OO4O Automation Server」>「OBJECTS」>「OraBfile」>「METHODS」>「CloseAll」、および「OBJECTS」>「OraBfile」>「Examples」を選択してください。
- Java (JDBC) : 『Oracle8i JDBC 開発者ガイドおよびリファレンス』の第7章「LOB と BFILE の操作」の「BLOB と CLOB の操作」の「BLOB または CLOB 列の作成と移入」、および『Oracle8i SQLJ 開発者ガイドおよびリファレンス』の第5章「型のサポート」の「Oracle の拡張型」の「BLOB、CLOB および BFILE のサポート」

使用例

ありません。

例

次の各プログラム環境の例が記載されています。

- [PL/SQL \(DBMS_LOB パッケージ\) : オープン中のすべての BFILE のクローズ \(11-207 ページ\)](#)
- [C \(OCI\) : オープン中のすべての BFILE のクローズ \(11-207 ページ\)](#)
- [COBOL \(Pro*COBOL\) : オープン中のすべての BFILE のクローズ \(11-208 ページ\)](#)
- [C/C++ \(Pro*C/C++\) : オープン中のすべての BFILE のクローズ \(11-210 ページ\)](#)
- [Visual Basic \(OO4O\) : オープン中のすべての BFILE のクローズ \(11-211 ページ\)](#)
- [Java \(JDBC\) : オープン中のすべての BFILE のクローズ \(11-211 ページ\)](#)

PL/SQL (DBMS_LOB パッケージ) : オープン中のすべての BFILE のクローズ

```

/* 例のプロシージャ closeAllOpenFilesBFILE_proc は、DBMS_LOB パッケージの
一部ではないことに注意してください。*/
CREATE OR REPLACE PROCEDURE closeAllOpenFilesBFILE_proc IS
BEGIN
  /* オープン中のすべての BFILE をクローズします。*/
  DBMS_LOB.FILECLOSEALL;
END;

```

C (OCI) : オープン中のすべての BFILE のクローズ

```

void BfileCloseAll(envhp, errhp, svchp, stmt hp)
OCIEnv      *envhp;
OCIError    *errhp;
OCISvcCtx   *svchp;
OCIStmt     *stmt hp;
{

```

```
OCILOBLocator *bfile_loc1;
OCILOBLocator *bfile_loc2;

/* ロケータ記述子を割り当てます。*/
(void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &bfile_loc1,
                          (ub4) OCI_DTYPE_FILE,
                          (size_t) 0, (dvoid **) 0);

(void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &bfile_loc2,
                          (ub4) OCI_DTYPE_FILE,
                          (size_t) 0, (dvoid **) 0);

checkerr(errhp, OCILOBFileSetName(envhp, errhp, &bfile_loc1,
                                  (OraText *) "PHOTO_DIR", (ub2) strlen("PHOTO_DIR"),
                                  (OraText *) "Lincoln_photo",
                                  (ub2) strlen("Lincoln_photo")));

checkerr(errhp, OCILOBFileSetName(envhp, errhp, &bfile_loc2,
                                  (OraText *) "PHOTO_DIR", (ub2) strlen("PHOTO_DIR"),
                                  (OraText *) "Washington_photo",
                                  (ub2) strlen("Washington_photo")));

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));

/* ロケータ記述子を解放します。*/
OCIDescriptorFree((dvoid *)bfile_loc1, (ub4)OCI_DTYPE_FILE);
OCIDescriptorFree((dvoid *)bfile_loc2, (ub4)OCI_DTYPE_FILE);
}
```

COBOL (Pro*COBOL) : オープン中のすべての BFILE のクローズ

```
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 ORASLNDR      PIC 9(4).

```

```

EXEC SQL INCLUDE SQLCA END-EXEC.
EXEC ORACLE OPTION (ORACA=YES) END-EXEC.
EXEC SQL INCLUDE ORACA END-EXEC.

```

```

PROCEDURE DIVISION.
BFILE-CLOSE-ALL.

```

```

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

```

* BFILE を割り当てます。

```

EXEC SQL ALLOCATE :BFILE1 END-EXEC.
EXEC SQL ALLOCATE :BFILE2 END-EXEC.

```

* ディレクトリおよびファイル情報を設定します。

```

MOVE "AUDIO_DIR" TO DIR-ALIAS1-ARR.
MOVE 9 TO DIR-ALIAS1-LEN.
MOVE "Washington_audio" TO FNAME1-ARR.
MOVE 16 TO FNAME1-LEN.

```

```

EXEC SQL
LOB FILE SET :BFILE1
DIRECTORY = :DIR-ALIAS1, FILENAME = :FNAME1 END-EXEC.
EXEC SQL LOB OPEN :BFILE1 READ ONLY END-EXEC.

```

* ディレクトリおよびファイル情報を設定します。

```

MOVE "PHOTO_DIR" TO DIR-ALIAS2-ARR.
MOVE 9 TO DIR-ALIAS2-LEN.
MOVE "Lincoln_photo" TO FNAME2-ARR.
MOVE 13 TO FNAME2-LEN.
EXEC SQL LOB FILE SET :BFILE2
DIRECTORY = :DIR-ALIAS2, FILENAME = :FNAME2 END-EXEC.
EXEC SQL LOB OPEN :BFILE2 READ ONLY END-EXEC.

```

* BFILE1 と BFILE2 の両方をクローズします。

```

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++) : オープン中のすべての BFILE のクローズ

```
#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;
    /* ロケータを移入します。*/
    EXEC SQL SELECT Music INTO :Lob_loc1
        FROM Multimedia_tab WHERE Clip_ID = 3;
    EXEC SQL SELECT Mtab.Voiced_ref.Recording INTO Lob_loc2
        FROM Multimedia_tab Mtab WHERE Mtab.Clip_ID = 3;
    /* 両方の BFILE をオープンします。*/
    EXEC SQL LOB OPEN :Lob_loc1 READ ONLY;
    EXEC SQL LOB OPEN :Lob_loc2 READ ONLY;
    /* オープン中のすべての BFILE をクローズします。*/
    EXEC SQL LOB FILE CLOSE ALL;
```

```

/* ロケータが保持しているリソースを解放します。*/
EXEC SQL FREE :Lob_loc1;
EXEC SQL FREE :Lob_loc2;
}

void main()
{
    char *samp = "samp/samp";
    EXEC SQL CONNECT :samp;
    closeAllOpenBFILEs_proc();
    EXEC SQL ROLLBACK WORK RELEASE;
}

```

Visual Basic (OO40) : オープン中のすべての BFILE のクローズ

```

Dim OraParameters as OraParameters, OraPhoto as OraBFile
OraConnection.BeginTrans

Set OraParameters = OraDatabase.Parameters

' BFILE 型の IN OUT パラメータを定義します。
OraParameters.Add "MyPhoto", Null, ORAPARAM_BOTH, ORATYPE_BFILE

' clip_id 1 の写真 BFILE を選択します。
OraDatabase.ExecuteSQL("Begin SELECT Photo INTO :MyPhoto FROM
Multimedia_tab WHERE Clip_ID = 1; END; ")

' BFILE photo 列を取得します。
set OraPhoto = OraParameters("MyPhoto").Value

' OraPhoto をオープンします。
OraPhoto.Open

' OraPhoto に何らかの処理を行います。

' OraPhoto に対応付けられたすべての BFILE をクローズします。
OraPhoto.CloseAll

```

Java (JDBC) : オープン中のすべての BFILE のクローズ

```

import java.io.InputStream;
import java.io.OutputStream;

```

```
// コア JDBC クラス :
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 固有の JDBC クラス :
import oracle.sql.*;
import oracle.jdbc.driver.*;
public class Ex4_66
{
    static final int MAXBUFSIZE = 32767;
    public static void main (String args [])
        throws Exception
    {
        // Oracle JDBC Driver をロードします。
        DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

        // データベースに接続します。
        Connection conn =
            DriverManager.getConnection ("jdbc:oracle:oci8:@", "samp", "samp");

        // 自動コミットがオフの場合、高速になります。
        conn.setAutoCommit (false);

        // 文を作成します。
        Statement stmt = conn.createStatement ();
        try
        {
            BFILE lob_loc1 = null;
            BFILE lob_loc2 = null;
            ResultSet rset = null;
            OracleCallableStatement cstmt = null;
            rset = stmt.executeQuery (
                "SELECT photo FROM multimedia_tab WHERE clip_id = 3");
            if (rset.next())
            {
                lob_loc1 = ((OracleResultSet)rset).getBFILE (1);
            }

            rset = stmt.executeQuery (
                "SELECT BFILENAME('PHOTO_DIR', 'RooseveltFDR_photo') FROM DUAL");
```

```

if (rset.next())
{
    lob_loc2 = ((OracleResultSet)rset).getBFILE (1);
}

cstmt = (OracleCallableStatement) conn.prepareCall (
    "BEGIN DBMS_LOB.FILEOPEN(?,DBMS_LOB.LOB_READONLY); END;");
// 最初のLOBをオープンします。
cstmt.setBFILE(1, lob_loc1);
cstmt.execute();

cstmt = (OracleCallableStatement) conn.prepareCall (
    "BEGIN DBMS_LOB.FILEOPEN(?,DBMS_LOB.LOB_READONLY); END;");
// 同じCallableStatementを使用して、2番目のLOBをオープンします。
cstmt.setBFILE(1, lob_loc2);
cstmt.execute();

lob_loc1.openFile ();
lob_loc2.openFile ();

// lob_loc1 と lob_loc2 の両方の最初のバイトから開始して、
// MAXBUFSIZE バイトを比較します。
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));

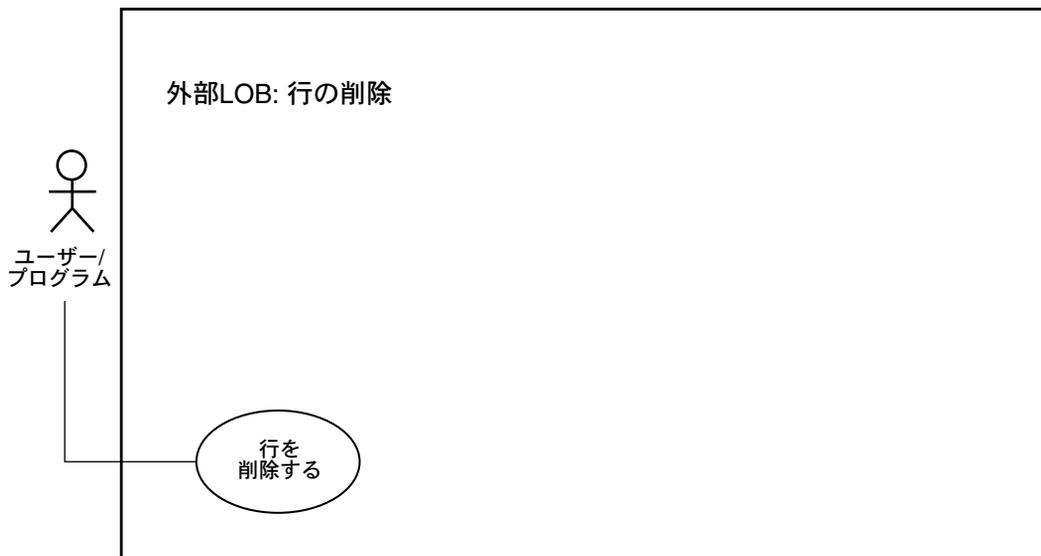
// すべてのBFILEをクローズします。
stmt.execute("BEGIN DBMS_LOB.FILECLOSEALL; END;");

stmt.close();
cstmt.close();
conn.commit();
conn.close();
}
catch (SQLException e)
{
    e.printStackTrace();
}
}
}

```

BFILE を含む表の行の削除

図 11-37 ユースケース図：BFILE を含む表の行の削除



参照： 外部 LOB (BFILE) に関するすべての基本操作については、11-2 ページの「[ユースケース・モデル:外部 LOB \(BFILE\)](#)」を参照してください。

用途

BFILE を含む表の行を削除します。

使用上の注意

内部永続 LOB と異なり、BFILE 内の LOB 値は、SQL DDL または SQL DML コマンドの使用では削除されません。これらのコマンドでは、BFILE ロケータのみが削除されます。BFILE 列を含むレコードを削除すると、物理オペレーティング・システム・ファイルそのものが削除されるのではなく、そのレコードと既存のファイルとのリンクが解除されます。ある行に対して DELETE 文を発行すると、その行の BFILE ロケータが削除されるため、オペレーティング・システム・ファイルへの参照を削除することになります。

構文

次のマニュアルの項を参照してください。

- SQL: 『Oracle8i SQL リファレンス』の第7章「SQL文」の「DELETE」、「DROP TABLE」、「TRUNCATE」

使用例

次に示す DELETE、DROP TABLE または TRUNCATE TABLE 文は、行 (Image1.gif を参照する BFILE ロケータ) を削除しますが、ファイル・システムにあるオペレーティング・システム・ファイルは残ります。

例

例が SQL で示されています。この例は、すべてのプログラム環境に適用されます。

- [SQL: 表からの行の削除](#)

SQL: 表からの行の削除

DELETE

```
DELETE FROM Multimedia_tab
WHERE Clip_ID = 3;
```

DROP

```
DROP TABLE Multimedia_tab;
```

TRUNCATE

```
TRUNCATE TABLE Multimedia_tab;
```


索引

数字

16 進文字列

DBMS_LOB.WRITE() への渡し, 9-199, 10-149

8.0 または 8.1.5 へのダウングレード、CACHE READS
LOB の使用, 4-16

B

BFILE, 2-2

BFILENAME() を使用した初期化, 2-6

CLOB または NCLOB への変換, 11-47

DBMS_LOB での読み込み, 3-8

DBMS_LOB、バイトで示されるオフセット・パラ
メータおよび量パラメータ, 3-7

DBMS_LOB 読み込み専用プロシージャ, 3-9

OCI 読み込み専用関数, 3-15

Oracle Objects for OLE (OO4O)

プロパティ, 3-35

メソッドのオープン / クローズ, 3-33

読み込み専用メソッド, 3-35

Pro*C/C++ プリコンパイラ文, 3-23

Pro*COBOL プリコンパイラ埋込み SQL 文, 3-27

アクセス, 11-5

値を読み込み / テストする OCI 関数, 3-15

オープンおよびクローズのための Pro*C/C++ プリ
コンパイラの使用, 3-24

記憶デバイス, 2-2

クローズ, 11-191

最大オープン数, 4-2, 11-140

参照セマンティクス, 2-3

セキュリティ, 11-8, 11-9

データ型, 2-2, 2-3

等しいロケータのチェック, 11-161

マルチスレッド・サーバー (MTS), 11-12

読み込み専用のサポート, 4-16

読み込み / テストのための JDBC の使用, 3-41

ロケータ, 2-5

BFILENAME(), 11-24, 11-179

使用するメリット, 11-7

BFILE クラス

「JDBC」を参照

BFILE バッファリング

「JDBC」を参照

BLOB

BLOB 値を読み込み / テストする JDBC の使用, 3-38

DBMS_LOB、バイトで示されるオフセット・パラ
メータおよび量パラメータ, 3-7

DBMS_LOB を介しての変更, 3-8

クラス

「JDBC」を参照

データ型, 2-2

変更のための JDBC の使用, 3-38

変更のための oracle.sql.BLOB メソッドの使用,
3-38

BLOB バッファリング

「JDBC」を参照

C

C

「OCI」を参照

C++

「Pro*C/C++ プリコンパイラ」を参照

CACHE / NOCACHE, 7-8

CHUNK, 7-10

CLOB

DBMS_LOB、文字で示されるオフセット・パラ
メータおよび量パラメータ, 3-7

DBMS_LOB を介しての変更, 3-8

JDBC の読み込み / テスト, 3-40

可変幅, 2-4

データ型, 2-2

可変幅列, 2-4

変更のための JDBC の使用, 3-39

列

可変幅文字データ, 2-4

CLOB クラス

「JDBC」を参照

CLOB バッファリング

「JDBC」を参照

Clone メソッド

「Oracle Objects for OLE (OO4O)」を参照

COBOL

「Pro*COBOL プリコンパイラ」を参照

CSID パラメータ

OCILobRead および OCILobWrite の OCI_UCS2ID
への設定, 3-11

D

DBMS_LOB

WRITE()

16 進文字列の渡し, 9-199, 10-149

DBMS_LOB()

READ, 9-100

DBMS_LOB パッケージ

BFILE 用の読み込み専用ファンクション / プロシ
ージャ, 3-9

BLOB、CLOB および NCLOB を変更するファンク
ション / プロシージャ, 3-8

CREATETEMPORARY(), 10-16

LOADFROMFILE(), 11-48

WRITE()

ガイドライン, 9-199

テンポラリ LOB のガイドライン, 10-149

オフセット・パラメータおよび量パラメータの規
則, 3-7

起動前の LOB ロケータの提供, 3-6

クライアント・プロシージャによる DBMS_LOB の
コール不可, 3-6

使用可能な LOB プロシージャ / ファンクション,
3-3

テンポラリ LOB, 3-9

内部および外部 LOB のオープン / クローズ, 3-9

内部および外部 LOB を読み込み / テストするファン
クション / プロシージャ, 3-8

マルチスレッド・サーバー (MTS), 11-12

DBMS_LOB パッケージ LOB の作業、使用, 3-6

directory_alias パラメータ, 11-26

DIRECTORY オブジェクト, 11-5

Windows NT 上の名前, 11-8

カタログ・ビュー, 11-10

個々のファイルではなくオブジェクトに対する

READ 権限, 11-9

使用のガイドライン, 11-11

別名およびファイル名の取得, 11-168

命名規則, 11-8

ロケータ使用前に OS ファイルの存在が必要, 11-25

DIRECTORY オブジェクトのビュー, 11-10

DIRECTORY 名の指定, 11-8

E

EMPTY_BLOB()/EMPTY_CLOB()

使用時期の FAQ, 6-7

EMPTY_CLOB()/BLOB()

BFILE の初期化, 2-6

内部 LOB の初期化, 2-6

F

FILECLOSEALL(), 11-11, 11-56, 11-70

FOR UPDATE 句

LOB, 2-7

LOB ロケータ, 5-2

FREETEMPORARY(), 10-28

I

INSERT 文

4,000 バイトを超えるバインド, 7-14

J

Java

「JDBC」を参照

JDBC

8.1.x で使用できない OracleBlob、OracleClob, 3-42

BFILE クラス, 3-36

BFILE バッファリング, 3-42

BLOB および CLOB クラス, 3-36

BLOB 値の変更, 3-38

BLOB 値の読み込み / テスト, 3-38

- BLOB バッファリングのためのメソッド / プロパティ, 3-39
 - CLOB 値の変更, 3-39
 - CLOB 値の読み込み / テスト, 3-40
 - CLOB バッファリングのためのメソッド / プロパティ, 3-40
 - DBMS_LOB パッケージのコール, 3-36
 - Java を使用した内部 LOB および外部 LOB (BFILE) の読み込み, 3-36
 - LOB の参照, 3-37
 - LOB を参照するための OraclePreparedStatement の OUT パラメータの使用, 3-37
 - LOB を参照するための OracleResultSet の使用, 3-37
 - オブジェクト oracle.sql.BLOB/CLOB を使用した Java での内部 LOB の変更, 3-36
 - 外部 LOB (BFILE) の値の読み込み / テスト, 3-41
 - 空の LOB ロケータを持つ行の表への挿入, 6-8
 - 構文参照, 3-37
 - 使用可能な LOB メソッド / プロパティ, 3-3
 - ロケータのカプセル化, 3-36
- JPublisher
- 空の LOB の作成, 6-9
- ## L
-
- LBS
- 「LOB バッファリング・サブシステム (LBS)」を参照
- LOADFROMFILE()
- 使用前に必要な BFILE キャラクタ・セット変換, 10-34
- LOB
- interMedia*, 1-4
 - LONG データ型, 1-3
 - NULL への設定, 2-8
 - SELECT の実行, 2-7
 - 値, 2-5
 - 移行の問題, 1-5
 - インタフェース
 - 「プログラム環境」を参照
 - オブジェクト・キャッシュ, 5-18
 - 外部 (BFILE), 2-2
 - 可変幅文字データ, 7-3
 - 行内記憶域, 2-5
 - 区分的操作, 5-6
 - 更新済 LOB ロケータ, 5-6
 - 互換性, 1-5
 - 使用する理由, 1-2
 - 使用例, 8-2
 - 内部 LOB
 - CACHE / NOCACHE, 7-8
 - CHUNK, 7-10
 - ENABLE | DISABLE STORAGE IN ROW, 7-11
 - LOGGING / NOLOGGING, 7-9
 - Oracle Objects for OLE (OO4O)、メソッドの変更, 3-32
 - PCTVERSION, 7-7
 - 空に設定, 2-9
 - 更新前のロック, 9-142, 9-179, 9-189, 9-198, 9-214, 9-224
 - コピー・セマンティクス, 2-3
 - 初期化, 11-97
 - トランザクション, 2-2
 - 表領域および LOB 索引, 7-7
 - 表領域および記憶特性, 7-5
 - ロケータ, 2-5
 - パーティション表内, 7-24
 - バッファリング
 - 通告, 5-19
 - ページング処理ができるページ, 5-24
 - バッファリング・サブシステム, 5-19
 - バッファリング使用方法, 5-21
 - パフォーマンス向上のための最善策, 7-31
 - 非構造化データ, 1-2
 - 表
 - 索引作成, 7-28
 - 作成, 7-26
 - パーティション化, 7-26
 - パーティションの移動, 7-29
 - パーティションの交換, 7-28
 - パーティションの追加, 7-29
 - パーティションの分割, 7-29
 - パーティションのマージ, 7-29
 - フラッシュ, 5-19
 - 読取り一貫性のあるロケータ, 5-2
 - ロケータ, 2-5, 5-2
 - 複数のトランザクションにまたがることはできない, 7-13
 - ロケータによるアクセス, 2-7
 - ロケータを含めるための設定, 2-6
 - LOB データ型への変換, 6-3
 - LOB の ANSI 規格, 10-10
 - LOB の値, 2-5

LOB のためのインタフェース
「プログラム環境」を参照

LOB のチェックアウト
内部永続 LOB, 9-64

LOB のチェックイン
内部永続 LOB, 9-75

LOB バッファリング
JDBC での BLOB バッファリング, 3-39
LBS を介した更新済 LOB のフラッシュ, 5-24
OCILobFlushBuffer(), 5-24
OCI 関数, 3-16
OCI 例, 5-26
Oracle Objects for OLE (OO4O)
内部 LOB のためのメソッド, 3-34
Pro*C/C++ プリコンパイラ文, 3-24
Pro*COBOL プリコンパイラ文, 3-28
ガイドライン, 5-19
使用方法, 5-21
テンポラリ LOB
CACHE、NOCACHE、CACHE READS, 10-8
テンポラリ LOB に対する使用禁止化, 10-188
テンポラリ LOB に対するフラッシュ, 10-182
バッファリング対応のロケータ, 5-25
バッファの物理構造, 5-21
バッファのフラッシュ, 5-23
例, 5-22

LOB バッファリング・サブシステム (LBS), 5-19
OCI を使用したバッファリングの例, 5-26
ガイドライン, 5-19
更新済 LOB のフラッシュ, 5-24
使用方法, 5-21
バッファのフラッシュ, 5-23
バッファリング対応のロケータ, 5-24
メリット, 5-19
リセレクトを避けるためのロケータ状態の保存,
5-25
例, 5-22

LOB ファイルのディスク・ストライプ化, 8-3

LOB ロケータ
外部 LOB (BFILE), 2-3
コピー・セマンティクス, 2-3
参照セマンティクス, 2-3
内部 LOB, 2-3

LOGGING / NOLOGGING, 7-9

LONG データ型と LOB データ型, 1-3

M

Multimedia_tab, 9-1, 11-1
表構造, 8-5

N

NCLOB
DBMS_LOB、文字で示されるオフセット・パラ
メータおよび量パラメータ, 3-7
DBMS_LOB を介しての変更, 3-8
可変幅, 2-4
データ型, 2-2

NOCOPY 制限事項, 10-12

O

OCI
BFILE 固有の関数, 3-15
Lob バッファリング関数, 3-16
LOB ロケータ関数, 3-16
LOB を使用するための使用, 3-11
NCLOB パラメータ, 3-13
OCILobFileGetLength
CLOB および NCLOB の入力および出力の長さ,
3-12

OCILobRead
可変幅 CLOB および NCLOB の入力および量の
量, 3-12

OCILobRead、OCILobWrite の OCI_UCS2ID への
設定, 3-11

OCILobWrite
可変幅 CLOB および NCLOB の入力および量の
量, 3-12

seeIfLOBOpen および main() の使用方法, 3-17
オフセット・パラメータおよび量パラメータの規則
固定幅キャラクタ・セット, 3-11

使用可能な LOB 関数, 3-3

テンポラリ LOB, 10-12

テンポラリ LOB のための関数, 3-15

内部 LOB の値を変更する関数, 3-14

内部および外部 LOB の値を読み込みまたはテストす
る関数, 3-15

内部および外部 LOB をオープン / クローズする関
数, 3-16

バッファリング例, 5-26

ロケータ, 2-7

論理バケットへのテンポラリ LOB のグループ化,
10-8

OCIBindByName(), 7-14

OCIBindByPos(), 7-14

OCIDuration(), 10-8

OCIDurationEnd(), 10-8, 10-12, 10-28

OCILOBAssign(), 5-20, 10-10

OCILOBFileSetName(), 11-7, 11-13

OCILOBFlushBuffer(), 5-24

OCILOBFreeTemporary(), 10-28

OCILOBGetLength(), 11-98

OCILOBLoadFromFile(), 11-48

OCILOBRead(), 9-90, 9-100, 10-57, 11-98
大量の LOB データの読み込み, 9-65
量, 6-6

OCILOBWrite(), 10-149
大量の LOB データの書き込み, 9-76

OCILOBWriteAppend(), 9-189

OCIObjectFlush(), 11-13

OCIObjectNew(), 11-13

OCISetAttr(), 11-13

OO4O
「Oracle Objects for OLE (OO4O)」を参照

OraBfile
「Oracle Objects for OLE (OO4O)」を参照

OraBlob
「Oracle Objects for OLE (OO4O)」を参照

Oracle Call Interface
「OCI」を参照

Oracle Objects for OLE (OO4O)
LOB での操作のためのプロパティ, 3-34

OraBfile の例, 3-30

OraBlob の例, 3-30

OraBlob、OraClob および OraBfile によるロケータ
のカプセル化, 3-30

外部 LOB (BFILE) での操作のためのプロパティ,
3-35

外部 LOB (BFILE) のオープン / クローズ, 3-33

外部 LOB (BFILE) のための読み込み専用メソッド,
3-35

構文参照, 3-29

使用可能な LOB メソッド / プロパティ, 3-3

ダイナセットからロケータの独立を維持するための
Clone メソッドの使用, 3-30

内部 LOB および外部 LOB (BFILE) の値の読み込み
 / テスト, 3-33

内部 LOB の変更, 3-32

内部 LOB バッファリング, 3-34

OraclePreparedStatement
「JDBC」を参照

OracleResultSet
「JDBC」を参照

oracle.sql.BFILE
BFILE バッファリング, 3-42
BFILE を読み込み / テストする JDBC メソッド, 3-41

oracle.sql.BLOB
BLOB 値の変更のための, 3-38
BLOB 値の読み込み / テスト, 3-38
BLOB バッファリング, 3-39
「JDBC」を参照

oracle.sql.CLOB
CLOB 値の変更, 3-39
CLOB 値を読み込み / テストする JDBC メソッド,
3-40
CLOB バッファリング, 3-40
「JDBC」を参照

P

PCTVERSION, 7-7

PL/SQL, 3-2
テンポラリ LOB の他への割当て, 10-109
テンポラリ LOB のロケータを再度割り当てる場合
のパフォーマンス, 10-12

PL/SQL プロシージャ
DBMS_LOB をコールできないクライアント側,
4-16

Pro*C/C++ プリコンパイラ
BFILE のための文, 3-23
LOB バッファリング, 3-24
使用可能な LOB 関数, 3-3
テンポラリ LOB のための文, 3-23
内部 LOB および外部 LOB (BFILE) のオープンお
よびクローズ, 3-24
内部 LOB の値の変更, 3-22
内部および外部 LOB の値の読み込みまたはテスト,
3-22
ロケータ, 3-23
割り当てられた入力ロケータ・ポインタの提供,
3-21

Pro*COBOL プリコンパイラ
BFILE のための文, 3-27
LOB バッファリング, 3-28
使用可能な LOB 関数, 3-3

テンポラリ LOB, 3-27
内部 LOB および外部 LOB (BFILE) のオープン /
クローズ, 3-28
内部 LOB の値の変更, 3-26
内部および外部 LOB の読み込みまたはテスト, 3-27
ロケータ, 3-28
割り当てられた入力ロケータの提供, 3-25

S

SELECT 文
FOR UPDATE, 2-7
読取り一貫性, 5-2
SESSION_MAX_OPEN_FILES パラメータ, 4-2,
11-56, 11-69
setData
JPublisher を使用しての EMPTY_BLOB() への設定,
6-9
SQL DDL
BFILE セキュリティ, 11-10
SQL DML
BFILE セキュリティ, 11-10
SQL*Loader
行内 LOB データのロード, 4-7
内部 LOB のパフォーマンス, 4-6

T

TO_LOB
制限事項, 9-60

U

UPDATE 文
4,000 バイトを超えるバインド, 7-14

V

VARRAY
「VARRAY の作成」を参照
サポートされない LOB, 4-15
VARRAY の作成
LOB への参照を含む, 5-28
Visual Basic
「Oracle Objects for OLE (OO4O)」を参照

い

移行, 1-5
一時表領域
4,000 バイトを超えるバインド, 7-14

う

埋込み SQL 文
「Pro*C/C++ プリコンパイラ」および
「Pro*COBOL プリコンパイラ」を参照

お

オープン
BFILE, 11-55
FILEISOPEN() を使用したオープン中の BFILE の
チェック, 11-71
FILEOPEN を使用した BFILE, 11-57
ISOPEN を使用した BFILE のオープンのチェック,
11-77
LOB がオープンしているかどうかの確認, 9-51
OPEN を使用した BFILE, 11-62
オープン中の BFILE のチェック, 11-69
テンポラリ LOB がオープンしているかどうかの
チェック, 10-40
オブジェクト・キャッシュ
LOB, 5-18
オブジェクト・リレーショナル設計, 8-4

か

外部 LOB (BFILE)
「BFILE」を参照
外部 LOB へのアクセス, 11-5
外部コールアウト, 5-24
解放
テンポラリ LOB, 10-28
書込み
1 つずつまたはピース単位, 9-189
少量のデータ、バッファリングの使用可能化,
9-233
ストリーム, 10-149
データを LOB に
内部永続 LOB, 9-197
データをテンポラリ LOB に, 10-148

カタログ・ビュー
 v\$temporary_lob, 10-13
各国語サポート
 NCLOB, 2-2
可変幅文字データ, 2-4

き

キャッシュ
 オブジェクト・キャッシュ, 5-18
キャラクタ・セット ID
 「CSID パラメータ」を参照
 取得
 内部永続 LOB, 9-172
 テンポラリ LOB、取得, 10-125
キャラクタ・セット・フォーム
 取得
 内部永続 LOB, 9-175
切捨て
 LOB データ
 内部永続 LOB, 9-213
 テンポラリ LOB データ, 10-158

く

クローズ
 BFILE, 11-191
 CLOSE を使用した BFILE, 11-198
 FILECLOSE を使用した BFILE, 11-193
 オープン中のすべての BFILE, 11-205

こ

更新
 BFILENAME() を使用した BFILE, 11-178
 BFILE を含む行, 11-177
 EMPTY_CLOB()/EMPTY_BLOB() の使用
 内部永続 LOB, 9-254
 PL/SQL バインド変数を使用した LOB, 5-10
 異なるロケータを使用した LOB の回避, 5-8
 初期化した LOB ロケータ・バインド変数の使用
 内部永続 LOB, 9-259
 別の表からの BFILE の選択による BFILE, 11-181
 別の表からの LOB の選択による行
 内部永続 LOB, 9-257
 ロック, 9-142, 9-179, 9-214, 9-224
更新済ロケータ, 5-2, 5-6, 5-10, 5-13, 5-23

構文
 値, 10-10
コード
 サンプル・プログラム, 1-5
 デモ・プログラムのリスト, 1-5
コールバック, 9-65, 9-76, 9-101, 9-189, 10-57,
 10-149
互換性, 1-5
コピー
 BFILE にはコピー機能がない, 11-181
 BFILE 用の LOB ロケータ, 11-149
 LOB に対する LONG, 6-3
 LOB の全体または一部を別の LOB に
 内部永続 LOB, 9-141
 LOB ロケータ
 内部永続 LOB, 9-152
 LONG を LOB に, 9-59
 TO_LOB 制限事項, 9-60
 テンポラリ LOB の全体または一部を他へ, 10-97
 テンポラリ LOB ロケータ, 10-107
コピー・セマンティクス, 2-3
 内部 LOB, 9-27

さ

サーバーへのラウンドトリップ、回避, 5-19, 5-25
索引構成表
 LOB に対する行内記憶域および, 6-6
削除
 LOB を含む行
 内部永続 LOB, 9-267
参照セマンティクス, 2-3, 9-27
 1 つのレコード内に複数の BFILE を持つことが可
 能, 11-7
サンプル・プログラム, 1-5

し

システム所有のオブジェクト
 「DIRECTORY オブジェクト」を参照
消去
 LOB の一部
 内部永続 LOB, 9-223
 テンポラリ LOB の一部, 10-166
初期化
 EMPTY_CLOB()、EMPTY_BLOB() の使用, 2-6

内部 LOB

「LOB」を参照

BFILENAME() を使用した BFILE 列またはロケータ
変数, 11-25

BFILE の LOB ロケータのチェック, 11-156

CREATE TABLE または INSERT 中, 9-24

EMPTY_BLOB() を使用する BLOB 属性 FAQ, 6-8

外部 LOB, 2-6

す

ストリーム, 9-76, 9-90

書込み, 9-198, 10-149

バッファリングの使用禁止化、使用時, 9-233

読込み

テンポラリ LOB, 10-57

せ

制限

データのバインディング、INSERTS および
UPDATES に対して削除された, 4-17

制限事項

4,000 バイトを超えるバインド, 7-16

セキュリティ

BFILE, 11-8, 11-9

SQL DDL を使用した BFILE, 11-10

SQL DML を使用した BFILE, 11-10

設定

LOB を NULL に, 2-8

NLS_LANG 変数に対するオーバーライド, 3-11

内部 LOB を空に, 2-9

セマンティクス

BFILE に対する参照ベース, 11-7

疑似参照, 10-10

内部 LOB に対するコピー・ベース, 9-27

そ

挿入

1 つ以上の LOB 値を行に, 9-21

4,000 バイトを超えるバインド, 9-22

BFILENAME() を使用した行, 11-24

BFILE を含む列, 11-23

EMPTY_CLOB()/EMPTY_BLOB() を使用した LOB
値の

内部永続 LOB, 9-23

初期化した BFILE ロケータを使用した行, 11-34

初期化した LOB ロケータを使用した行

内部永続 LOB, 9-29

別の表からの BFILE の選択によって BFILE を含む
行, 11-32

別の表からの LOB の選択による行

内部永続 LOB, 9-27

存在

BFILE のチェック, 11-131

ち

チャンク・サイズ, 9-198

複数、パフォーマンスの改善, 9-101

つ

追加

LOB への追加の書込み

内部永続 LOB, 9-188

LOB を他の LOB に

内部永続 LOB, 9-178

テンポラリ LOB を他へ, 10-131

追加の書込み

テンポラリ LOB, 10-140

て

ディレクトリ

カタログ・ビュー, 11-10

使用のガイドライン, 11-11

所有権および権限, 11-9

ディレクトリ・オブジェクト, 11-5

データ型

LOB への変換 FAQ, 6-3

データを内部 LOB にバインディング、制限削除, 4-17

デモ・プログラム, 1-5

テンポラリ LOB

DBMS_LOB の使用可能なファンクション / プロ
シージャ, 3-9

IN 値として使用できるロケータ, 10-6

LOB がテンポラリ LOB であるかどうかのチェック,
10-22

OCI および論理バケット, 10-8

OCI 関数, 3-15

Pro*C/C++ プリコンパイラ埋込み SQL 文, 3-23

Pro*COBOL プリコンパイラ文, 3-27

一時表領域に格納されたデータ, 10-7
永続 LOB に使用される類似の関数, 10-7
永続 LOB の場合と同じロケータ操作, 10-6
永続 LOB ロケータで上書きする前の明示的な解放,
10-9
機能, 10-10
キャラクタ・セット ID, 10-125
切捨て, 10-158
クライアントではなくサーバーへの常駐, 10-8
サポートされないトランザクションおよび読取り一
貫性, 10-7
使用されない行内および行外, 10-7
操作しない SQL DML, 10-6
存続期間, 10-8
追加での書込み, 10-140
バッファリングの使用禁止化, 10-188
パフォーマンス, 10-10
メモリー操作, 10-8
テンポラリ LOB の作成, 10-14
テンポラリ LOB ロケータ内への永続 LOB の選択,
10-9

と

等価
テンポラリ LOB ロケータが他と, 10-116
トランザクション
移行, 5-24
完全に関連する内部 LOB, 2-2
関連しない外部 LOB, 2-3
シリアライズ可能でないロケータ, 5-15
シリアライズ可能なロケータ, 5-15
またがることができない LOB ロケータ, 7-13
ロケータの ID, 5-15
トランザクションの境界
LOB ロケータ, 5-15
トリガー
LOB 列、更新された場合の通知方法, 6-4

な

長さ
BFILE の取得, 11-140
テンポラリ LOB, 10-88
内部永続 LOB, 9-133

は

バインド
4,000 バイトを超える更新
内部永続 LOB, 9-252
HEX から RAW または RAW から HEX の変換,
7-14
「INSERT 文」および「UPDATE 文」を参照
パターン
instr を使用した BFILE 内のパターンの有無の
チェック, 11-123
instr を使用した LOB 内のパターンの有無の確認
内部永続 LOB, 9-126
テンポラリ LOB
有無のチェック, 10-81
バッファのフラッシュ, 5-19
テンポラリ LOB, 10-182
バッファリング
使用可能化
内部永続 LOB, 9-232
使用禁止化
内部永続 LOB, 9-244
フラッシュ
内部永続 LOB, 9-238
バッファリングの使用禁止化
「LOB バッファリング」を参照
パフォーマンス
OCI およびテンポラリ LOB, 10-12
同じテンポラリ LOB への複数ロケータの割当て、
影響, 10-10

ひ

非 NULL
LOB 列への書込み前
内部永続 LOB, 9-255
比較
2 つの BFILE の全体または一部, 11-114
2 つのテンポラリ LOB の全体または一部, 10-74
2 つの LOB の全体または一部
内部永続 LOB, 9-118
非構造化データ, 1-2
等しい
1 つの LOB ロケータと別の LOB ロケータ
内部永続 LOB, 9-160

等しいロケータ

BFILE の LOB ロケータが他と等しいかどうかの
チェック, 11-161

表示

テンポラリ LOB データ, 10-46
内部永続 LOB の LOB データ, 9-89

表の作成

1 つ以上の BFILE 列を含む, 11-15
1 つ以上の LOB 列を含む
内部永続 LOB, 9-8
BFILE 属性を持つオブジェクト型, 11-18
BFILE を含む, 11-14
BFILE を含む NESTED TABLE, 11-21
LOB 属性を持つオブジェクト型を含む
内部永続 LOB, 9-13
NESTED、LOB を含む
内部永続 LOB, 9-18

表領域

一時, 10-8
格納されたテンポラリ LOB データ, 10-7

ふ

プログラム環境, 3-2

使用可能な関数, 3-3
比較, 3-3

へ

変換

LOADFROMFILE() の使用前に BFILE 上で必要な
キャラクタ・セットの変換, 10-34
「HEX から RAW のバインド」を参照
キャラクタ・セット, 11-47
バイナリ・データからキャラクタ・セットへ,
11-47

ほ

ポーリング, 9-65, 9-76, 9-101, 9-189, 10-149

ま

マルチスレッド・サーバー (MTS)

BFILE, 11-12

マルチメディア

コンテンツ収集, 8-2

も

文字データ

可変幅, 2-4

ゆ

ユースケース

図の解釈方法, xlv
内部永続 LOB のすべてのリスト, 9-2
モデル、図形概要, 9-1

よ

読込み

BFILE

LOB にかかわらず 4GB -1 を指定, 11-98
LOB からのデータ
内部永続 LOB, 9-99
substr を使用した BFILE データの一部, 11-107
substr を使用した LOB の一部
内部永続 LOB, 9-110
少量のデータ、バッファリングの使用可能化,
9-233
ストリーミングを使用した大量の LOB データ,
9-65
テンポラリ LOB からのデータ, 10-56
テンポラリ LOB の一部, 10-67

読取り一貫性

LOB, 5-2

読取り一貫性のあるロケータ, 5-2, 5-3, 5-10, 5-13,
5-23, 5-25, 5-28

り

量パラメータ

BFILE での使用, 11-48
LOB データの読込みおよびロード、サイズ, 6-4

れ

例

PL/SQL 変数を使用した LOB の更新, 5-10
SQL DML と DBMS_LOB の混合による影響, 5-6
更新済 LOB ロケータ, 5-8
デモ・プログラム, 1-5
読取り一貫性のあるロケータ, 5-3

ろ

ロード

- BFILE のデータを LOB に, 11-46, 9-40
- BFILE のデータをテンポラリ LOB に, 10-33
- 外部 LOB (BFILE) データを表に, 11-42
- データを内部 LOB に, 9-38

ロケータ, 2-5

- BFILE, 11-12
 - 2つの行による同じファイルの参照, 11-12
 - ガイドライン, 11-12
- LOB 初期化または含めるための BFILE, 2-6
- LOB へのアクセス, 2-7
- LOB ロケータが初期化されているかどうかの確認
 - 内部永続 LOB, 9-167
- OCI 関数, 3-16
- Pro*COBOL プリコンパイラ文, 3-28
- Pro*COBOL プリコンパイラへの提供, 3-25
- SELECT 実行時期に関係なく同じ LOB 値を提供する読取り一貫性のあるロケータ, 5-3
- 外部 LOB (BFILE), 2-5
- 更新済, 5-2, 5-6, 5-10, 5-13, 5-23
- 再選択を避けるための状態の保存, 5-25
- 使用した LOB への読込みおよび書込み, 5-15
- 選択, 2-7
- テンポラリ LOB のコピー, 10-107
- テンポラリ、永続 LOB の選択, 10-9
- トランザクションの境界, 5-15
- バッファリング対応, 5-25
- 複数の, 5-2
- 複数のトランザクションにまたがることはできない, 7-13
- 含めるための列または属性の設定, 2-6
- 読取り一貫性のある, 5-2, 5-3, 5-10, 5-13, 5-23, 5-25, 5-26, 5-28
- 読取り一貫性のあるロケータ, 5-2

わ

割当て

- テンポラリ LOB でのコレクションから他のコレクションへ, 10-11
- テンポラリ LOB を他のテンポラリ LOB に, 10-11

