Oracle9*i*

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 2（9.2）

2002 年 7 月
部品番号 : J06287-01

ORACLE®

Oracle9*i* 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 2（9.2）

部品番号：J06287-01

原本名：Oracle9*i* Application Developer's Guide - Large Objects (LOBs), Release 2 (9.2)

原本部品番号：A96591-01

原本著者：Eric Paapanen、Shelley Higgins、Susan Kotsovolos、Den Raphaely

原本協力者：K. Akiyama、Geeta Arora、S. Banerjee、Yujie Cao、T. H. Chang、E. Chong、S. Das、C. Freiwald、C. Iyer、M. Jagannath、R. Krishnan、M. Krishnaprasad、S. Lari、Li-Sen Liu、D. Mullen、V. Nimani、A. Roy、S. Shah、A. Shivarudraiah、J. Srinivasan、R. Toohey、Anh-Tuan Tran、G. Viswana、A. Yalamanchi、J. Balaji、D. Cruceanu、M. Chien、G. Edmiston、M. Fry、J. Kalogeropoulos、V. Karra、P. Manavazhi、S. Muthulingam、R. Ratnam、C. Shay、A. Shehade、E. Shirk、Jan Syssauw、S. Vedala、E. Wan、J. Yang

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

目次

2 基本 LOB コンポーネント

3 様々なプログラム環境での LOB のサポート

4 LOB の管理

5 ラージ・オブジェクト（LOB）: 詳細事項

6 LOB に関する FAQ

7 モデリングおよび設計

8 LONG から LOB への移行

9 LOB: 効果的な使用方法

10 内部永続 LOB

11 一時 LOB

12 外部 LOB（BFILE）

13 OraOLEDB を使用した LOB の操作

14 LOB の事例

A Unified Modeling Language 図

B マルチメディア・スキーマ

索引

はじめに

このマニュアルでは、Oracle9*i* 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に適用されるもので、システム固有の情報は含みません。

ここでは、次の項目について説明します。

機能範囲および可用性

このマニュアルでは、Oracle9*i* および Oracle9*i* Enterprise Edition 製品の特長および機能について説明します。Oracle9*i* と Oracle9*i* 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ったり、Enterprise Edition でもオプション扱いとなっているものもあります。パーティション機能を使用するには、Partitioning Option を選択します。

注意： Oracle9*i* Enterprise Edition の今回のリリースから、オブジェクトおよび機能をインストールするためにオブジェクト・オプションを選択する必要がなくなりました。

必要なオプション

LOB の取扱いについて、システム上の特別な制限はありません。

参照： 制限の詳細は、次の項を参照してください。

- xli ページの「削除された制限」
- 4-16 ページの「LOB の制限」
- 5-31 ページの「パーティション化された索引構成表内の LOB に対する制限」

対象読者

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。

このマニュアルの構成

このマニュアルは、次のように構成されています。

第 1 章「LOB の概要」

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 LOB コンポーネント」

この章では、内部永続 LOB と一時 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「様々なプログラム環境での LOB のサポート」

この章では、LOB を操作するために使用する、次の 8 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（『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照）
- **Oracle Call Interface（OCI）**を使用した C（『Oracle Call Interface プログラマーズ・ガイド』を参照）
- **Oracle C++ Call Interface（OCCI）**を使用した C++（『Oracle C++ Call Interface 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/C++（『Pro*C/C++ Precompiler 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（『Pro*COBOL Precompiler 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 Application Programmers Interface（API）**を使用した Java（『Oracle9*i* JDBC 開発者ガイドおよびリファレンス』を参照）
- Oracle Provider for OLE DB（OraOLEDB）を使用した OLEDB（『Oracle Provider for OLE DB 開発者ガイド』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「ラージ・オブジェクト（LOB）：詳細事項」

この章では、他のすべての章に関係する詳細事項について説明します。特に、読込み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、オブジェクト・キャッシュ内の LOB、およびパーティション化された索引構成表（PIOT）での LOB の使用を中心に説明します。

第 6 章「LOB に関する 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 LONG RAW 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、IOT およびパーティション表についても説明します。この章では、LOB に対する SQL セマンティクスの使用および LOB 列の索引付けについても説明します。

第 8 章「LONG から LOB への移行」

この章では、LOB 用の LONG API を使用した LONG から LOB への移行について説明します。この API を使用すると、LONG 列を LOB に変更したときに、既存のアプリケーションを変更する必要はほとんどありません。

第 9 章「LOB: 効果的な使用方法」

この章では、SQL*Loader を使用した LOB のロードのガイドライン、および LOB と一時 LOB のパフォーマンスのガイドラインについて説明します。

第 10 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 9 章に概要を示した使用例に沿って、関連する問題とともに説明します。また、Unified Modeling Language（UML）の表記について、利用方法の点から紹介します。各基本操作は、利用例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、付録 A を参照してください。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 11 章「一時 LOB」

この章では、第 10 章と同じ形式で、一時 LOB について説明します。今回のリリースの一時 LOB 用の新しい JDBC API には、一時 LOB の作成、BLOB/CLOB が一時的かどうかの確認、一時 BLOB/CLOB の解放、一時 LOB の比較および一時 LOB の切捨てが含まれます。一時 LOB の Visual Basic（OO4O）サンプル・スクリプトは、今回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 12 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 10 章、第 11 章と同じ形式で説明します。各操作は、利用例として扱われ、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。

第 13 章「OraOLEDB を使用した LOB の操作」

この章では、ADO レコードセットおよび OraOLEDB を使用した LOB の処理方法について説明します。

第 14 章「LOB の事例」

この章では、LOB を使用したマルチメディア・リポジトリの構築方法について説明します。また、LOB ベースの Web サイトを構築する場合に行う最初の手順についても説明します。

付録 A「Unified Modeling Language 図」

この付録では、第 10 章、第 11 章および第 12 章の利用図に使用されている UML 構文の使用方法について説明します。

付録 B「マルチメディア・スキーマ」

この付録では、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。

関連文書

詳細は、次の Oracle ドキュメントを参照してください。

- 『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』: PL/SQL は、オラクル社が開発した、SQL の手続き型拡張機能です。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。
- 『Oracle Call Interface プログラマーズ・ガイド』: OCI について説明しています。OCI を使用すると、Oracle サーバーにアクセスする C または C++ の 3GL アプリケーションを作成できます。
- 『Oracle C++ Call Interface プログラマーズ・ガイド』

- 『Pro*C/C++ Precompiler 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。このプリコンパイラを使用することで、アプリケーション・プログラムに SQL および PL/SQL を組み込むことができます。
- 『Pro*COBOL Precompiler 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 サーバーにアクセスするための COBOL プログラムに SQL および PL/SQL を組み込むことができます。
- **Java:** Oracle9*i* では、データベース内で Java を使用できます。
  - 『Oracle9*i* JDBC 開発者ガイドおよびリファレンス』
  - 『Oracle9*i* JPublisher ユーザーズ・ガイド』
  - 『Oracle9*i* Java Stored Procedures Developer's Guide』

マルチメディア

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

- Oracle *inter*Media 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。
  - 『Oracle *inter*Media ユーザーズ・ガイドおよびリファレンス』
  - 『Oracle *inter*Media Java Classes ユーザーズ・ガイドおよびリファレンス』
  - 『Oracle Text リファレンス』
  - 『Oracle Text アプリケーション開発者ガイド』

基本的な参照

- SQL については、『Oracle9*i* SQL リファレンス』および『Oracle9*i* データベース管理者ガイド』を参照してください。
- LOB データを含む Oracle XML SQL については、『Oracle9*i* アドバンスト・レプリケーション』を参照してください。
- Oracle の基本概念については、『Oracle9*i* データベース概要』を参照してください。
- 『Oracle9*i* データベース・ユーティリティ』

リリース・ノート、インストール・マニュアル、ホワイト・ペーパーまたはその他の関連文書は、OTN-J（Oracle Technology Network Japan）に接続すれば、無償でダウンロードできます。OTN-J を使用するには、オンラインでの登録が必要です。次の URL で登録できます。

`http://otn.oracle.co.jp/membership/`

すでに OTN-J のユーザー名およびパスワードを取得済であれば、次の OTN-J Web サイトの文書セクションに直接接続できます。

`http://otn.oracle.co.jp/document/`

表記規則

この項では、このマニュアルの本文およびコード例で使用される表記規則について説明します。この項の内容は次のとおりです。

本文中の表記規則

本文では、特別な用語をより迅速に識別できるように、様々な表記規則を使用します。次の表に、それらの表記規則を説明し、その使用例を示します。

規則	意味	例
太字	太字は、本文中で定義されている用語を示します。	この句を指定すると、**索引構成表**が作成されます。
固定幅フォントの大文字	固定幅フォントの大文字は、システムが提供する要素を示します。このような要素には、パラメータ、権限、データ型、Recovery Manager のキーワード、SQL キーワード、SQL*Plus またはユーティリティ・コマンド、パッケージおよびメソッドが含まれます。また、システムが提供する列名、データベース・オブジェクト、データベース構造、ユーザー名およびロールも含まれます。	NUMBER 列に対してのみに、この句を指定できます。 BACKUP コマンドを使用して、データベースのバックアップを取ることができます。 USER_TABLES データ・ディクショナリ・ビュー内の TABLE_NAME 列を問い合せます。 DBMS_STATS.GENERATE_STATS プロシージャを使用します。
固定幅フォントの小文字	固定幅フォントの小文字は、実行可能ファイル、ファイル名、ディレクトリ名およびユーザーが提供する要素のサンプルを示します。このような要素には、コンピュータ名、データベース名、ネット・サービス名および接続識別子が含まれます。また、ユーザーが提供するデータベース・オブジェクト、データベース構造、列名、パッケージ、クラス、ユーザー名、ロール、プログラム・ユニットおよびパラメータの値も含まれます。 **注意：**大文字と小文字を組み合せて使用するプログラム要素もあります。これらの要素は、記載されているとおり入力してください。	sqlplus と入力して、SQL*Plus をオープンします。 パスワードは、orapwd ファイルで指定します。 /disk1/oracle/dbs ディレクトリ内のデータ・ファイルおよび制御ファイルのバックアップを取ります。 hr.departments 表には、department_id、department_name および location_id 列があります。 QUERY_REWRITE_ENABLED 初期化パラメータを true に設定します。 oe ユーザーとして接続します。 JRepUtil クラスが次のメソッドを実装します。

規則	意味	例
固定幅フォントの小文字のイタリック	固定幅フォントの小文字のイタリックは、プレースホルダまたは変数を示します。	*parallel_clause* を指定できます。 U*old_release*.SQL を実行します。ここで、*old_release* とはアップグレード前にインストールしたリリースを示します。

コード例中の表記規則

コード例は、SQL、PL/SQL、SQL*Plus または他のコマンドライン文を説明します。コード例は、固定幅フォントで表示され、この例に示すとおり通常のテキストと区別されます。

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

次の表に、コード例で使用される表記規則を説明し、その使用例を示します。

規則	意味	例
[ ]	大カッコは、任意に選択する 1 つ以上の項目を囲みます。大カッコは、入力しないでください。	DECIMAL (*digits* [, *precision*])
{ }	中カッコは、2 つ以上の項目を囲み、そのうちの 1 つの項目は必須です。中カッコは、入力しないでください。	{ENABLE | DISABLE}
|	縦線は、大カッコまたは中カッコ内の 2 つ以上のオプションの選択項目を表します。オプションのうちの 1 つを入力します。縦線は、入力しないでください。	{ENABLE | DISABLE} [COMPRESS | NOCOMPRESS]
...	水平省略記号は、次のいずれかを示します。	
	■ 例に直接関連しないコードの一部が省略されている。	CREATE TABLE ... AS *subquery*;
	■ コードの一部を繰り返すことができる。	SELECT *col1*, *col2*, ..., *coln* FROM employees;
. . .	垂直の省略記号は、例に直接関連しない複数の行が省略されていることを示します。	SQL> SELECT NAME FROM V$DATAFILE; NAME ---------- /fsl/dbs/tbs_01.dbf /fs1/dbs/tbs_02.dbf . . . /fsl/dbs/tbs_09.dbf 9 rows selected.

規則	意味	例
その他の句読点	大カッコ、中カッコ、縦線および省略記号以外の句読点は、表示されているとおり入力する必要があります。	`acctbal NUMBER(11,2);` `acct CONSTANT NUMBER(4) := 3;`
イタリック体	イタリック体は、プレースホルダまたは特定の値を指定する必要のある変数を示します。	`CONNECT SYSTEM/`*`system_password`* `DB_NAME = `*`database_name`*

大文字	大文字は、システムが提供する要素を示します。これらの用語は、ユーザー定義の用語と区別するために大文字で示されます。用語が大カッコ内にないかぎり、表示されているとおりの順序および綴りで入力します。ただし、これらの用語は大文字 / 小文字が区別されないため、小文字でも入力できます。	`SELECT last_name, employee_id FROM employees;`
		`SELECT * FROM USER_TABLES;`
		`DROP TABLE hr.employees;`

小文字	小文字は、ユーザーが提供するプログラム要素を示します。たとえば、表名、列名、ファイル名などです。
	注意：大文字と小文字を組み合せて使用するプログラム要素もあります。これらの要素は、記載されているとおり入力してください。
		`SELECT last_name, employee_id FROM employees;`
		`sqlplus hr/hr`
		`CREATE USER mjones IDENTIFIED BY ty3MU9;`

ラージ・オブジェクト（LOB）の新機能

ここでは、次のリリースで導入された LOB の新機能について説明します。

- Oracle9i リリース 2（9.2）で導入された LOB の新機能
- Oracle9i リリース 1（9.0.1）で導入された LOB の新機能
- Oracle8i リリース 8.1.6 で導入された LOB の機能
- Oracle8i リリース 8.1.5 で導入された LOB の機能

Oracle9*i* リリース 2（9.2）で導入された LOB の新機能

今回のリリースでは、新しい PL/SQL API が導入されていますが、これらの API では、LOB からバイナリ・データおよび文字データをロードするための機能が改善されています。

- **DBMS_LOB.LOADBLOBFROMFILE**

  この API によって、バイナリ・ラージ・オブジェクト（BLOB）をオペレーティング・システム・ファイルから内部永続 LOB、一時 LOB および外部 LOB（BFILE）にロードできます。

  参照：

  - 10-42 ページの「内部永続 BLOB への BFILE バイナリ・データのロード」を参照してください。
  - 11-46 ページの「一時 BLOB への BFILE のバイナリ・データのロード」を参照してください。
  - 12-55 ページの「BLOB への BFILE データのロード」を参照してください。

- **DBMS_LOB.LOADCLOBFROMFILE**

  この API によって、キャラクタ・ラージ・オブジェクト（CLOB）をオペレーティング・システム・ファイルから内部永続 LOB、一時 LOB および外部 LOB（BFILE）にロードできます。この API では、BFILE データのキャラクタ・セットから宛先の CLOB/NCLOB キャラクタ・セットへの変換が適切に実行されます。

  参照：

  - 10-46 ページの「内部永続 CLOB への BFILE データのロード」を参照してください。
  - 11-50 ページの「一時 CLOB/NCLOB へのファイルのキャラクタ・データのロード」を参照してください。
  - 12-59 ページの「CLOB への BFILE データのロード」を参照してください。

削除された制限

Oracle9*i* リリース 2 では、次の制限が削除されました。

- **トリガー**

  今回のリリースでは、DML BEFORE ROW トリガー :new が LOB に対してサポートされています。これは、LOB に対するトリガーが、他のすべての列の型に対するトリガーと同じ規則に従うことを意味します。

- **ローカル管理表領域**

  ローカル管理表領域に LOB 列を作成できるようになりました。

- **LOB および自動セグメント管理の表領域**

  自動セグメント管理の表領域に、LOB を格納できるようになりました。

- **NCLOB パラメータ**

  NCLOB パラメータを、オブジェクト型の属性として使用できるようになりました。

- **パーティション化された索引構成表**

  パーティション化された索引構成表（PIOT）がサポートされるようになりました。

- **クライアント側の PL/SQL DBMS_LOB プロシージャ**

  クライアント側の PL/SQL DBMS_LOB プロシージャがサポートされるようになりました。

- **PL/SQL トリガー本体**

  以前のリリースの場合、DML BEFORE ROW トリガーの PL/SQL トリガー本体では、LOB の :old 値を読み込むことはできましたが、:new 値を読み込むことはできませんでした。

- **トリガーのサポート**

  以前のリリースでは、LOB 列を含むビューに INSTEAD OF トリガーが存在する場合、その LOB 列に INSERT または UPDATE 文字列は指定できませんでしたが、今回のリリースでは、この制限が削除されました。次に例を示します。

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

Oracle9*i* リリース 1（9.0.1）で導入された LOB の新機能

この項では、Oracle9*i* の LOB の新機能について説明します。

- **LONG から LOB への移行 API**

  今回のリリースから、LOB への移行を補助するために、LOB 用の LONG API がサポートされます。この API を使用すると、LONG 列を LOB に変更したときに、既存のアプリケーションを変更する必要はほとんどありません。LOB の方がメリットが多いため、可能な場合は、LONG ではなく LOB を使用するように既存のアプリケーションを変更してください。

  参照： 第 8 章「LONG から LOB への移行」を参照してください。

- **SQL セマンティクスでの LOB の使用**

  今回のリリースから、SQL 文字列演算子や SQL ファンクションなどの SQL VARCHAR2 セマンティクスを使用して、（内部永続）LOB にアクセスできるようになりました。ユーザーが使い慣れた SQL インタフェースが提供されるため、LOB データへ簡単にアクセスできます。これらのセマンティクスは、サイズが小さい LOB（10 ～ 100KB）に対して使用することをお薦めします。

  参照： 第 7 章「モデリングおよび設計」を参照してください。

- **Oracle C++ Call Interface（OCCI）での LOB の使用**

  Oracle C++ Call Interface（OCCI）は、Oracle データベース内のデータを操作するための新しい C++ API です。OCCI は、使いやすい C++ クラスのコレクションとして構成されています。これらの C++ クラスによって、C++ プログラムで、データベースへの接続、SQL 文の実行、データベース表への値の挿入とデータベース表の値の更新、問合せの結果の取出し、データベース内でのストアド・プロシージャの実行、データベース・スキーマ・オブジェクトのメタデータへのアクセスを行うことができます。OCCI API には、JDBC および Open DataBase Connectivity（ODBC）より多くのメリットがあります。

  参照：

  - 第 3 章「様々なプログラム環境での LOB のサポート」を参照してください。
  - 第 10 章「内部永続 LOB」を参照してください。

- **JDBC LOB の新機能**

  JDBC LOB に関連する新機能は次のとおりです。

  - 一時 LOB API: 一時 LOB を作成および破棄します。
  - TRIM API: 指定された長さまで LOB を切り捨てます。
  - OPEN API および CLOSE API: LOB を明示的にオープンおよびクローズします。
  - 新しいストリーミング API: 指定されたオフセットから、Java ストリームとして LOB の読込みおよび書込みを行います。
  - JDBC を使用した空の LOB インスタンスの作成 : これらのインスタンスは、データベースのラウンドトリップを必要としません。

    参照 :

    - 第 3 章「様々なプログラム環境での LOB のサポート」を参照してください。
    - 第 10 章「内部永続 LOB」を参照してください。
    - 第 11 章「一時 LOB」
    - 第 12 章「外部 LOB（BFILE）」を参照してください。

- **パーティション化された索引構成表での LOB のサポート**

  Oracle9*i* では、パーティション化された索引構成表内で、LOB、LOB として格納された VARRAY 列、および BFILE を使用できます。これらの表での LOB 列の動作は、従来の（ヒープ構成）パーティション表での動作に似ていますが、わずかに違います。

  参照 : 第 5 章「ラージ・オブジェクト（LOB）: 詳細事項」を参照してください。

- **OLE DB および LOB の使用**

  OLE DB は、異なるストアから様々な型のデータに同じ方法でアクセスするためのオープン仕様です。OLE DB は、LOB 型に対する次の機能をサポートします。

  - **永続 LOB:** 行セット全体での読込み / 書込み
  - **BFILE**: 行セット全体での読込み専用

    参照 : 第 13 章「OraOLEDB を使用した LOB の操作」を参照してください。

Oracle8*i* リリース 8.1.6 で導入された LOB の機能

注意： Oracle8*i* リリース 8.1.6 と Oracle8*i* リリース 8.1.7 では、LOB の機能は同じです。

Oracle8*i* リリース 8.1.6 で導入された LOB の新機能は、次のとおりです。

- **CACHE READS**

  LOB 列に対する CACHE READS オプション

- **4,000 バイト制限の削除**

  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**

  Oracle8*i* リリース 8.1.6 以上では、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がサポートされています。

  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。

Oracle8*i* リリース 8.1.5 で導入された LOB の機能

Oracle8*i* リリース 8.1.5 で導入された LOB の新機能は、次のとおりです。

- 一時 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 に対する新規 API（open/close/isopen、writeappend、getchunksize）
- 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

1

LOB の概要

この章の内容は次のとおりです。

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、次のような種類のデータを処理する必要性が増大しています。

- 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

これまで、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。アドバンスト・キューイングなどのキューイング・テクノロジによって、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します。

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を見ることができるように、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（OS）・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

Oracle では、次の 2 種類の LOB をサポートしています。

- 表のインラインまたは別々のセグメントや表領域（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
- OS ファイル（BFILE など）として格納される LOB

XML、LOB および Oracle Text（*inter*Media Text）の使用

CLOB または BFILE を使用した非構造化データの格納

CLOB は大量の文字データを格納できるため、非構造化 XML 文書を格納する場合に有効です。また、外部ファイル参照である BFILE も、マルチメディア・データの格納に有効です。この場合、XML は RDBMS の外部で格納および管理されますが、サーバー上での問合せには使用できます。

Oracle Text（*inter*Media Text）の索引付けによる XML 要素の内容検索のサポート

CLOB 列に Oracle Text （*inter*Media Text）の索引を作成し、XML で問合せを実行できます。

参照： 詳細は、次のマニュアルを参照してください。

- 『Oracle9*i* XML Developer's Kit ガイド - XDK』
- 『Oracle Text リファレンス』
- 『Oracle Text アプリケーション開発者ガイド』

LOB 対応の Oracle Text（*inter*Media Text）

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、Oracle8*i* および Oracle9*i* 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、オーディオおよびビデオ・データがあります。

Oracle8*i* では、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・データをサポートする *inter*Media をバンドルしています。SQL 問合せを使用して *inter*Media オブジェクトへアクセスし、その内容を操作（イメージの切捨てなど）し、内容の読込みおよび書込みを行い、あるフォーマットから別のフォーマットへデータを変換できます。

また、*inter*Media は Oracle 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。Oracle *inter*Media 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。これによってアプリケーション開発が簡単になります。

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。

Oracle8*i* および Oracle9*i* 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 のサポートより優れています。

- **LOB 容量**: Oracle8 および Oracle8*i* 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。
- **表当たりの LOB 列数**: Oracle8、Oracle8*i* または Oracle9*i* 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 列には、異なる型を指定できます。Oracle7 リリース 7.3 以上では、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をサポートします。

LOB 列

注意： LOB はオブジェクトの属性にもなります。

LOB（BLOB、CLOB、NCLOB または BFILE）型の列には、ロケータという値または参照が格納されます。ロケータは、LOB の位置を指定します。

LOB 列は、ロケータを格納するのみではない LOB 列では、LOB ロケータはインラインに格納されます。ユーザー定義の SQL データ定義言語（DDL）記憶域パラメータによっては、Oracle9*i* は、約 4KB より小さい LOB を表のインラインに格納できます。LOB が約 4KB より大きくなると、Oracle9*i* 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、Oracle9*i* は、LOB ロケータのみでなく、LOB データをインラインに格納します。

BLOB、CLOB および NCLOB のデータは、データベース内でアウトラインに格納されます。BFILE データは、データベース外の OS ファイルに格納されます。Oracle9*i* は、LOB へのアクセスおよび LOB で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。

LONG から LOB への移行 API

Oracle9*i* は、LONG データ型の他に LOB データ型もサポートします。LOB の方がメリットが多いため、できるだけ、LONG ではなく LOB を使用するように既存のアプリケーションを変更してください。

LONG から LOB へ移行すると、LONG 列にアクセスしている既存のアプリケーションを簡単に移行して LOB 列を使用できます。移行は、次の 2 段階で行います。

- データの移行
- アプリケーションの移行

参照： 詳細は、次の項を参照してください。

- 7-2 ページの「LOB 型と LONG 型および LONG RAW 型との比較」
- 第 8 章「LONG から LOB への移行」

LOB に対する SQL セマンティクスのサポート

今回のリリースから、SQL 文字列演算子や SQL ファンクションなどの SQL VARCHAR2 セマンティクスを使用して、LOB にアクセスできるようになりました。

ユーザーには使い慣れた SQL インタフェースが提供されるため、LOB データへのアクセスが大幅に簡単になります。この追加機能は、次の 2 つの場合に有効です。

- サイズが小さい LOB（10 ～ 100KB）を使用してデータを格納し、SQL 問合せでその LOB データにアクセスする必要がある場合。構文は、VARCHAR2 の構文と同じです。
- LONG 列を LOB に移行した場合。今回のリリースでは、LONG-to-LOB への移行 API を使用して、より簡単に移行プロセスを実行できます。詳細は第 8 章「LONG から LOB への移行」を参照してください。

参照： 7-33 ページの「LOB に対する SQL セマンティクスのサポート」を参照してください。

パーティション化された索引構成表（PIOT）および LOB

Oracle9*i* では、パーティション化された索引構成表内で、LOB、LOB として格納された VARRAY 列、および BFILE を使用できます。これらの表での LOB 列の動作は、従来の（ヒープ構成）パーティション表での動作に似ていますが、次の項目で違いがあります。

- 表領域のマッピング
- インライン LOB およびアウトライン LOB

LOB 列は、レンジ・パーティション化された索引構成表でのみサポートされます。

参照： 第 5 章「ラージ・オブジェクト（LOB）：詳細事項」の「パーティション化された索引構成表内の LOB」を参照してください。

LOB に対する拡張索引作成機能

Oracle は、拡張索引作成機能を持つ拡張可能サーバーを提供します。この機能によって、必要に応じて新しい索引タイプを定義できます。これは、協調的索引作成の概念に基づいています。この場合、データ・カートリッジおよび Oracle9*i* は、オンライン分析処理（OLAP）のために、テキストや空間などのデータ型用の索引を作成およびメンテナンスします。

カートリッジは、索引構造の定義、ロード操作および更新操作中の索引内容のメンテナンス、問合せ処理中の索引の検索を行います。索引構造は、ヒープ構成表または索引構成表として Oracle に格納するか、OS ファイルとして外部に格納できます。

これを行うために、Oracle は索引タイプの概念を導入しています。索引タイプの目的は、データ・カートリッジを使用して、テキスト、空間、イメージ、OLAP などの複合ドメインを効率的に検索および取出しできるようにすることです。索引タイプは、Oracle サーバーに組み込まれているソート済の索引またはビットマップ索引に似ています。相違点は、組込み索引は Oracle カーネルによって実装されますが、索引タイプはデータ・カートリッジの開発者によって実装されることです。データ・カートリッジの開発者が新しい索引タイプを実装すると、データ・カートリッジのエンド・ユーザーは、組込み索引タイプと同様にその索引タイプを使用できます。

データベース・システムがドメイン索引の物理的な格納を処理する場合、データ・カートリッジは次のことを行います。

- 索引のフォーマットおよび内容の定義。これによって、カートリッジが複合データ・オジェクトを保存できる索引構造を定義できます。
- ドメイン索引の作成、削除および更新。カートリッジは、索引構造の構築およびメンテナンスを処理します。これは、単純な SQL データ型用に提供されている医療業界向け索引付け機能から大幅に発展した機能です。また、索引は一定の数の要素で構成される集合として作成されるため、置換更新が直接サポートされます。
- 索引の内容へのアクセスおよび解析。この機能を持つデータ・カートリッジは、問合せの処理に不可欠なコンポーネントです。データベースへの問合せの内容に関連する句は、データ・カートリッジが処理します。

Oracle9*i* では、拡張索引作成機能がサポートされるため、LOB などの複合データ型にアクセスするパフォーマンスの高いソリューションを非常に簡単に開発できます。

拡張可能オプティマイザ

ユーザー定義ファンクションおよび索引の作成者は、拡張可能オプティマイザの機能を使用して、統計集計ファンクション、選択ファンクションおよびコスト・ファンクションを作成できます。オプティマイザは、この情報を使用して問合せ計画を選択します。そのため、コストベースのオプティマイザは、ユーザーが提供する情報を使用するように拡張されています。ルールベース・オプティマイザは変更されていません。

拡張索引作成機能を使用すると、新しい演算子、索引タイプおよびドメイン索引を定義できます。これらのユーザー定義演算子およびドメイン索引に対しては、拡張可能オプティマイザの機能を使用して、オプティマイザが実行計画を選択するために使用する 3 つの主な構成要素（統計、選択性、コスト）を制御できます。

LOB に対するファンクション索引付け

ファンクション索引とは、式に対して作成する索引です。ファンクション索引によって、列に対する索引付けより優れた機能を提供します。ファンクション索引によって、データへのアクセス方法が多様化されます。

現在、ファンクション索引はネストした表には作成できません。ただし、LOB 列および VARRAY には作成できます。

> **参照：** ファンクション索引の使用の詳細は、『Oracle9*i* 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。

CLOB として XMLType 列に格納できる XML 文書

構成済 XML 文書は CLOB に格納できます。`XMLType` 列では、CLOB を使用して格納します。

> **参照：** XMLType および XML の LOB への格納方法の詳細は、『Oracle9*i* XML Developer's Kit ガイド - XDK』の第 1 章を参照してください。

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、『Oracle9*i* データベース移行ガイド』の次の項を参照してください。

- CLOB および NCLOB の可変幅キャラクタ・セットについては、第 5 章「互換性および相互運用性」の「リリース 2（9.2）とリリース 1（9.0.1）間の互換性および相互運用性の問題」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。
- 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 7 章「以前の Oracle リリースへのデータベースのダウングレード」の「非互換性の削除」の「リリース 1（9.0.1）の非互換性の削除」の「LOB を含むパーティション索引構成表」の「パーティション索引構成表のすべての LOB 列の削除」および「パーティション索引構成表のすべての VARRAY 列の削除」を参照してください。
- ダウングレード :（LOB に対して）パーティション化された索引構成表については、第 7 章「以前の Oracle リリースへのデータベースのダウングレード」を参照してください。
- ダウングレード : LOB に対するファンクション索引については、第 7 章「リリース 8.1 へダウングレードする前の非互換性の削除」を参照してください。
- ダウングレード : LONG から LOB へのデータおよびアプリケーションの移行については、第 7 章「リリース 8.1 へダウングレードする前の非互換性の削除」を参照してください。

このマニュアルでの例

このマニュアルの例では、次のサンプル・スキーマが使用されています。

- 製品メディア（PM）のサンプル・スキーマ。
- マルチメディアのサンプル・スキーマは使用できません。

マルチメディアのサンプル・スキーマは使用できないため、Oracle9*i* のサンプル・スキーマでは提供されません。LOB のほとんどの例では、マルチメディア・スキーマのかわりに、製品メディア・スキーマがサンプル・スキーマとして使用されます。

このマニュアルの第 10 章、第 11 章および第 12 章の例は、PM スキーマに移行されています。他のほとんどの例は、移行されていないため、マルチメディア・スキーマのままです。これらの例の詳細は、付録 B「マルチメディア・スキーマ」を参照してください。

製品メディアのサンプル・スキーマには、LOB データベース機能の説明に適した表およびサンプル・データが含まれています。このデータのほとんどは、LOB API が大量の非構造化データを処理するように設計されているため、構造化されていません。「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。

このマニュアルの PM スキーマを基にした例では、サンプル表 `Print_media` が使用されています。

> **参照：** 製品メディアのサンプル・スキーマの詳細は、『Oracle9*i* サンプル・スキーマ』を参照してください。

2

基本 LOB 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

Oracle9*i* 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 LOB と外部 LOB の両方に適用できます。

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[1] 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`**: Oracle9*i* 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`**: Oracle9*i* 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

内部 LOB には、**永続 LOB** および**一時 LOB** の 2 種類があります。

外部 LOB（BFILE）

外部 LOB（BFILE）は、データベース表領域外の OS 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 OS 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 次記憶デバイスに加えて、CD-ROM、フォト CD、DVD などの 3 次ブロック記憶デバイスにも格納できます。

`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、読込み専用のバイト・ストリーム・アクセスを可能にします。

[1] アクセス制御情報ディレクトリ。これは、ユーザーのアクセスの種類およびアクセス先のディレクトリ・データを決定する属性です。この属性には、構造アクセス項目およびコンテンツ・アクセス項目に対する一連の規則が含まれます。詳細は、『Oracle Internet Directory 管理者ガイド』を参照してください。

Oracle では、基礎となるサーバーの OS が、これらの OS 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、トランザクションには関係ありません。LOB の整合性および耐久性は、OS 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。

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 OS 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 か一時 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。実際の OS ファイルはコピーされません。

可変幅文字データ

- 次の LOB 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

OCI にアクセスする DBMS_LOB.LOADFROMFILE およびファンクションの使用

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に、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。

ただし、LOADFROMFILE のかわりに SQL*Loader を使用して、データを CLOB/NCLOB にロードすることをお薦めします。

参照 :

- 10-46 ページの「内部永続 CLOB への BFILE データのロード」を参照してください。
- 11-50 ページの「一時 CLOB/NCLOB へのファイルのキャラクタ・データのロード」を参照してください。
- 12-59 ページの「CLOB への BFILE データのロード」を参照してください。

クライアント・キャラクタ・セットと UCS2 の間の変換

カートリッジ・サービスには、クライアント・キャラクタ・セットと UCS2 の間で変換を実行できる次の API があります。

- OCIUnicodeToCharSet()
- OCICharSetToUnicode()

参照：

- 第 3 章「様々なプログラム環境での LOB のサポート」を参照してください。
- 3-8 ページの「オフセット・パラメータおよび量パラメータ : DBMS_LOB パッケージに対する固定幅と可変幅（文字またはバイト）」を参照してください。
- 3-12 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（文字またはバイト）」を参照してください。

LOB の値およびロケータ

LOB 値のインライン記憶域

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インライン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 4,000 バイトより小さいと、その値はインラインに格納されます。それ以外の場合は、アウトラインに格納されます。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 が混在する場合のみに、インライン記憶域が効果的です。

7-11 ページの「ENABLE | DISABLE STORAGE IN ROW」に示すとおり、LOB 値は、約 4,000 バイトを超えると自動的にアウトラインに移されます。

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、ロケータはインラインに格納されます。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 OS 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 OS ファイルを指す BFILE ロケータを含むことができます。

ロケータを含む LOB 列および LOB 属性の設定

内部 LOB

サポートされているプログラム環境[1]（PL/SQL、OCI、OCCI、Pro*C/C++、Pro*COBOL、Visual Basic、Java または OLE DB）を使用してデータを内部 LOB に書き込む前に、LOB 列および LOB 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() ファンクション、CLOB および NCLOB に対しては EMPTY_CLOB() ファンクションを使用して、INSERT 文および 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：** 第 10 章「内部永続 LOB」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

サポートされているプログラム環境を使用して外部 LOB（BFILE）へのアクセスを開始する前に、BFILE 列および BFILE 属性を NULL 以外にする必要があります。BFILENAME() ファンクションを使用して、外部 OS ファイルを指す BFILE 列を初期化できます。

> **参照：** 第 12 章「外部 LOB（BFILE）」の「BFILENAME() を使用した行の挿入」を参照してください。

[1] **注意：** NULL が含まれていても LOB 属性であるかぎり、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、サポートされているプログラム環境は使用できません。

EMPTY_BLOB() または EMPTY_CLOB() ファンクションを単独で起動しても、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（『Oracle9*i* SQL リファレンス』の CREATE DIRECTORY 文を参照）。

注意： 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。

Multimedia_tab 表の定義は、付録 B「マルチメディア・スキーマ」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 の中の 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。

```
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));
```

注意： 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。

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

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「様々なプログラム環境での LOB のサポート」および『Oracle Call Interface 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込み一貫性のあるロケータの詳細は、第 5 章「ラージ・オブジェクト（LOB）：詳細事項」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場合は、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（詳細は、第 7 章「モデリングおよび設計」の「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 列または LOB 属性から SELECT すると、OCI、PL/SQL（DBMS_LOB）などのサポートされているプログラム環境を使用して、LOB にデータを移入するためのロケータを入手できます。第 3 章「様々なプログラム環境での LOB のサポート」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 文を発行する場合は、行の挿入時に内部 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;
```

この文によって、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。

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

この文によって、録音する必要があるセグメントを確認できます。

ただし、OCI 関数または DBMS_LOB ファンクションは NULL LOB ではコールできません。そのため、SQL UPDATE 文を発行して、NULL の LOB 列を、内部 LOB に対しては EMPTY_BLOB()、EMPTY_CLOB() または値（Denzel Washington など）に、外部 LOB に対してはファイル名に再設定する必要があります。

つまり、NULL に設定された LOB に対しては、サポートされているプログラム環境からはファンクションをコールできません。これらの関数またはファンクション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 EMPTY_BLOB() または EMPTY_CLOB() ファンクションを使用して 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 をコールして LOB にデータを移入すると、（次の SELECT に必要な、処理のラウンドトリップが避けられるため）効果的です。

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

Multimedia_tab 表を使用した LOB の例の初期化

次の INSERT 文を使用して、Multimedia_tab の中の 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）。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。

> **参照：** 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() ファンクションを使用して NULL またはファイル名に初期化できます。

> **参照：** 第 12 章「外部 LOB（BFILE）」の「ディレクトリ・オブジェクト」の「BFILE ロケータの初期化」を参照してください。

3

様々なプログラム環境での LOB のサポート

この章の内容は次のとおりです。

LOB で操作される 8 つのプログラム環境

表 3-1 に、LOB 機能をサポートする 8 つのプログラム環境（言語）を示します。第 10 章、第 11 章および第 12 章では、サポートされている LOB 機能を利用方法の点から説明します。ほとんどの LOB の利用例では、各プログラム環境の例を示します。

表 3-1 LOB の 8 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	この章での参照先
PL/SQL	DBMS_LOB パッケージ	『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』	3-7 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	OCI	『Oracle Call Interface 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++	OCCI	『Oracle C++ Call Interface プログラマーズ・ガイド』	3-22 ページの「C++（OCCI）を使用した LOB の作業」
C/C++	Pro*C/C++ プリコンパイラ	『Pro*C/C++ Precompiler プログラマーズ・ガイド』	3-28 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	『Pro*COBOL Precompiler プログラマーズ・ガイド』	3-32 ページの「COBOL（Pro*COBOL）を使用した LOB の作業」
Visual Basic	OO4O	OO4O は、Oracle9*i* Client for Windows 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、Oracle9*i* インストールの「Application Development」サブメニューから利用できます。	3-35 ページの「Visual Basic（OO4O）を使用した LOB の作業」
Java	JDBC API	『Oracle9*i* SQLJ 開発者ガイドおよびリファレンス』および『Oracle9*i* JDBC 開発者ガイドおよびリファレンス』	3-41 ページの「Java（JDBC）を使用した LOB の作業」
OLE DB	OraOLEDB（Oracle の OLE DB プロバイダ）	『Oracle Provider for OLE DB 開発者ガイド』	

LOB インタフェースの比較

表 3-2 および表 3-3 に、LOB で操作するために使用するファンクションおよびメソッドを示し、8 つの LOB インタフェースを比較します。表を 2 つに分けてあるのは、8 つのインタフェースをすべて示すためです。LOB に関するインタフェースの機能は、次の項で説明します。

表 3-2 LOB インタフェースの比較

PL/SQL: DBMS_LOB（dbmslob.sql）	C（OCI）（ociap.h）	C++（OCCI）（occiData.h）OCCIClob および OCCIBfile クラスの場合も含む	Pro*C/C++ および Pro*COBOL
DBMS_LOB.COMPARE	利用不可	利用不可	利用不可
DBMS_LOB.INSTR	利用不可	利用不可	利用不可
DBMS_LOB.SUBSTR	利用不可	利用不可	利用不可
DBMS_LOB.APPEND	OCILob.Append	OCCIBlob.append()	APPEND
利用不可（PL/SQL 割当て演算子を使用）	OCILob.Assign		ASSIGN
利用不可	OCILob.CharSetForm	OCCIClob.getCharsetForm（CLOB のみ）	利用不可
利用不可	OCILob.CharSetId	OCCIClob.getCharsetId()（CLOB のみ）	利用不可
DBMS_LOB.CLOSE	OCILob.Close	OCCIBlob.close()	CLOSE
利用不可	利用不可	OCCIClob.closeStream()	利用不可
DBMS_LOB.COPY	OCILob.Copy	OCCIBlob.copy()	COPY
利用不可	OCILob.DisableBuffering	利用不可	DISABLE BUFFERING
利用不可	OCILob.EnableBuffering	利用不可	ENABLE BUFFERING
DBMS_LOB.ERASE	OCILob.Erase	利用不可	ERASE
DBMS_LOB.FILECLOSE	OCILob.FileClose	OCCIClob.close()	CLOSE
DBMS_LOB.FILECLOSEALL	OCILob.FileCloseAll	利用不可	FILE CLOSE ALL
DBMS_LOB.FILEEXISTS	OCILob.FileExists	OCCIBfile.fileExists()	DESCRIBE [FILEEXISTS]

表 3-2 LOB インタフェースの比較（続き）

PL/SQL: DBMS_LOB（dbmslob.sql）	C（OCI）（ociap.h）	C++（OCCI）（occiData.h）OCCIClob および OCCIBfile クラスの場合も含む	Pro*C/C++ および Pro*COBOL
DBMS_LOB.GETCHUNKSIZE	OCILob.GetChunkSize	OCCIBlob.getChunkSize()	DESCRIBE [CHUNKSIZE]
DBMS_LOB.FILEGETNAME	OCILob.FileGetName	OCCIBfile.getFileName() および OCCIBfile.getDirAlias()	DESCRIBE [DIRECTORY, FILENAME]
DBMS_LOB.FILEISOPEN	OCILob.FileIsOpen	OCCIBfile.isOpen()	DESCRIBE [ISOPEN]
DBMS_LOB.FILEOPEN	OCILob.FileOpen	OCCIBfile.open()	OPEN
利用不可（BFILENAME 演算子を使用）	OCILob.FileSetName	OCCIBfile.setName()	FILE SET
利用不可	OCILob.FlushBuffer	利用不可	FLUSH BUFFER
DBMS_LOB.GETLENGTH	OCILob.GetLength	OCCIBlob.length()	DESCRIBE [LENGTH]
利用不可	OCILob.IsEqual	演算子 = ( )=/!= を使用	利用不可
DBMS_LOB.ISOPEN	OCILob.IsOpen	OCCIBlob.isOpen()	DESCRIBE [ISOPEN]
DBMS_LOB.LOADFROMFILE	OCILob.LoadFromFile	overloadedcopy() メソッドを使用	LOAD FROM FILE
N/A	OCILob.LocatorIsInit	OCCIClob.isinitialized()	利用不可
DBMS_LOB.OPEN	OCILob.Open	OCCIBlob.open	OPEN
DBMS_LOB.READ	OCILob.Read	OCCIBlob.read	READ
DBMS_LOB.TRIM	OCILob.Trim	OCCIBlob.trim	TRIM
DBMS_LOB.WRITE	OCILob.Write	OCCIBlob.write	WRITEORALOB
DBMS_LOB.WRITEAPPEND	OCILob.WriteAppend	利用不可	WRITE APPEND
DBMS_LOB.CREATETEMPORARY	OCILob.CreateTemporary	利用不可	利用不可
DBMS_LOB.FREETEMPORARY	OCILob.FreeTemporary	利用不可	利用不可
DBMS_LOB.ISTEMPORARY	OCILob.IsTemporary	利用不可	利用不可
	OCILob.LocatorAssign	演算子 = ( ) を使用、またはコンストラクタをコピー	利用不可

表 3-3 LOB インタフェースの比較

PL/SQL: DBMS_LOB（dbmslob.sql）	Visual Basic（OO4O）	Java（JDBC）	OLE DB
DBMS_LOB.COMPARE	ORALOB.Compare	DBMS_LOB を使用	利用不可
DBMS_LOB.INSTR	ORALOB.Matchpos	position	利用不可
DBMS_LOB.SUBSTR	利用不可	BLOB または BFILE 用の getBytes CLOB 用の getSubString	利用不可
DBMS_LOB.APPEND	ORALOB.Append	length の後で putBytes または PutString を使用	利用不可
利用不可（PL/SQL 割当て演算子を使用）	ORALOB.Clone	利用不可（等号を使用）	利用不可
利用不可	利用不可	利用不可	利用不可
利用不可	利用不可	利用不可	利用不可
DBMS_LOB.CLOSE	利用不可	DBMS_LOB を使用	利用不可
DBMS_LOB.COPY	ORALOB.Copy	read および write を使用	利用不可
利用不可	ORALOB.DisableBuffering	利用不可	利用不可
利用不可	ORALOB.EnableBuffering	利用不可	利用不可
DBMS_LOB.ERASE	ORALOB.Erase	DBMS_LOB を使用	利用不可
DBMS_LOB.FILECLOSE	ORABFILE.Close	closeFile	利用不可
DBMS_LOB.FILECLOSEALL	ORABFILE.CloseAll	DBMS_LOB を使用	利用不可
DBMS_LOB.FILEEXISTS	ORABFILE.Exist	fileExists	利用不可
DBMS_LOB.GETCHUNKSIZE	利用不可	getChunkSize	利用不可
DBMS_LOB.FILEGETNAME	ORABFILE.DirectoryName ORABFILE.FileName	getDirAlias getName	利用不可
DBMS_LOB.FILEISOPEN	ORABFILE.IsOpen	DBMS_LOB.ISOPEN を使用	利用不可
DBMS_LOB.FILEOPEN	ORABFILE.Open	openFile	利用不可

表 3-3 LOB インタフェースの比較（続き）

PL/SQL: DBMS_LOB（dbmslob.sql）	Visual Basic（OO4O）	Java（JDBC）	OLE DB
利用不可（BFILENAME 演算子を使用）	DirectoryName FileName	BFILENAME を使用	利用不可
利用不可	ORALOB.FlushBuffer	利用不可	利用不可
DBMS_LOB.GETLENGTH	ORALOB.Size	length	利用不可
利用不可	N/A	equals	利用不可
DBMS_LOB.ISOPEN	ORALOB.IsOpen	DBMS_LOB.IsOpen を使用	利用不可
DBMS_LOB.LOADFROMFILE	ORALOB.CopyFromBfile	read の後で write を使用	利用不可
DBMS_LOB.OPEN	ORALOB.open	DBMS_LOB を使用	利用不可
DBMS_LOB.READ	ORALOB.Read	BLOB または BFILE: getBytes および getBinaryStream	IRowset::GetData and ISequentialStream::Read
		CLOB:getString、getSubString および getCharacterStream	
DBMS_LOB.TRIM	ORALOB.Trim	DBMS_LOB を使用	利用不可
DBMS_LOB.WRITE	ORALOB.Write	BLOB または BFILE: putBytes および getBinaryOutputStream	IRowsetChange::SetData および ISequentialStream::Write
		CLOB: putString および getCharacterOutputStream	
DBMS_LOB.WRITEAPPEND	利用不可	length の後で putString または putBytes を使用	利用不可
DBMS_LOB.CREATETEMPORARY	利用不可	利用不可	利用不可
DBMS_LOB.FREETEMPORARY	利用不可	利用不可	利用不可
DBMS_LOB.ISTEMPORARY	利用不可	利用不可	利用不可

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 および一時 LOB**: 全体またはピース単位の読込みおよび変更操作
- **BFILE**: 読込み操作

> **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、『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・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、ルーチンをコール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、ディレクトリ・オブジェクトを定義します。これらのファイルは存在し、Oracle サーバーで処理を行うための読込み権限を所有している必要があります。OS がパス名の大 / 小文字を区別する場合には、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、12-4 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を使用して、前述の操作について説明します。

PL/SQL - LOB ガイドライン

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： DBMS_LOB パッケージに対する固定幅と可変幅（文字またはバイト）

DBMS_LOB パッケージ（固定幅と可変幅の両方のキャラクタ・セット）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、常に文字で表されます。
- BLOB および BFILE: オフセット・パラメータおよび量パラメータは、常にバイトで表されます。

DBMS_LOB.LOADFROMFILE: 量パラメータの値

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。（ただし、BFILE 全体をロードするための量パラメータの値には、LOBMAXSIZE 定数を指定することができます。）

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す。PL/SQL では、量パラメータをバッファ・サイズ以下に指定する必要があります。また、バッファ・サイズは **32KB** 以下に制限されます。

参照：

- 第 10 章の「内部永続 BLOB への BFILE バイナリ・データのロード」および「内部永続 CLOB への BFILE データのロード」を参照してください。
- 第 11 章の「一時 BLOB への BFILE のバイナリ・データのロード」および「一時 CLOB/NCLOB へのファイルのキャラクタ・データのロード」を参照してください。
- 第 12 章の「BLOB への BFILE データのロード」および「CLOB への BFILE データのロード」を参照してください。

LOB を操作する PL/SQL ファンクションおよびプロシージャ

BLOB、CLOB、NCLOB および BFILE を操作する 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 3-4 を参照してください。
- LOB 値の読込みまたはテストは、表 3-5 を参照してください。
- 一時 LOB の作成、解放または確認は、表 3-6 を参照してください。
- 外部 LOB（BFILE）での読込み専用ファンクションは、表 3-7 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確認は、表 3-8 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およびプロシージャ

表 3-4 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。
LOADCLOBFROMFILE()	キャラクタ・セットをファイルから LOB にロードします。
LOADBLOBFROMFILE()	バイナリ・データをファイルから 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 の終わりに書き込みます。

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

表 3-5 PL/SQL: 内部 LOB および外部 LOB の値の読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()	読込みおよび書込み用にチャンク・サイズを取得します。これは内部 LOB にのみ適用され、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: 一時 LOB を操作するファンクションおよびプロシージャ

表 3-6 PL/SQL: 一時 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()	一時 LOB を作成します。
ISTEMPORARY()	LOB ロケータが一時 LOB を参照するかどうかを確認します。
FREETEMPORARY()	一時 LOB を解放します。

PL/SQL: BFILE 固有の読込み専用ファンクションおよびプロシージャ

表 3-7 PL/SQL: BFILE 固有の DBMS_LOB 読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。CLOSE() でも可能です。
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確認します。
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確認します。ISOPEN() でも可能です。
FILEOPEN()	ファイルをオープンします。OPEN() でも可能です。

PL/SQL: 内部 LOB 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およびプロシージャ

表 3-8　PL/SQL: 内部 LOB 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確認します。
CLOSE()	LOB をクローズします。

次の章では、特定の LOB 操作（LOB を含む行の挿入など）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 10 章「内部永続 LOB」
- 第 11 章「一時 LOB」
- 第 12 章「外部 LOB（BFILE）」

これらの章では、$ORACLE_HOME/rdbms/demo/lobs/plsql にある PL/SQL LOB の多くのサンプル・スクリプトを参照できます。

C（OCI）を使用した LOB の作業

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 LOB および外部 LOB（BFILE）からの読込み
- 内部 LOB への書込み

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 3-9 ～表 3-15 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 における読込み / 書込みのためのキャラクタ・セット ID（CSID）パラメータの OCI_UCS2ID への設定

データを 2 バイトの Unicode（UCS2）フォーマットで読込みまたは書込みする場合は、OCILobRead および OCILobWrite の CSID 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 CSID を示します。`CSID` パラメータは、すべての CSID に設定できます。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、戻り値およびサンプル・コードの詳細は、『Oracle Call Interface 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、『Oracle9*i* Database グローバリゼーション・サポート・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（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、常に文字で表されます。
- BLOB および BFILE: オフセット・パラメータおよび量パラメータは、常にバイトで表されます。

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 :** 『Oracle9*i* Database グローバリゼーション・サポート・ガイド』を参照してください。

NCLOB

NCLOB パラメータは、メソッドで使用できます。

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

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。また、LOB の終わりまで読み込みます。

OCI LOB の例

OCI の例は、次の章にも記載されています。

- 第 10 章「内部永続 LOB」
- 第 11 章「一時 LOB」
- 第 12 章「外部 LOB（BFILE）」

OCI LOB のほとんどのサンプル・スクリプトは、Oracle9*i* 配布ソフトウェアの $ORACLE_HOME/rdbms/demo/lobs/oci から参照できます。

追加の OCI サンプル・スクリプトが、次のディレクトリに存在します。

- UNIX システムの場合
  - `/ORACLE_HOME/rdbms/demo/demolb.c`
  - `/ORACLE_HOME/rdbms/demo/demolb2.c`
  - `/ORACLE_HOME/rdbms/demo/demolbs.c`
- Windows NT の場合
  - `¥ORACLE_HOME¥Oci¥Samples¥demolb.c,....`

これ以外の OCI のデモ・スクリプトについては、『Oracle Call Interface 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: BLOB、CLOB、NCLOB および BFILE を操作する関数

BLOB、CLOB、NCLOB および BFILE を操作する OCI 関数は、次のとおりです。

- 内部 LOB の変更は、表 3-9 を参照してください。
- LOB 値の読込みまたはテストは、表 3-10 を参照してください。
- 一時 LOB の作成または解放、または一時 LOB が存在するかどうかの確認は、表 3-11 を参照してください。
- 外部 LOB（BFILE）における読込み専用関数については、表 3-12 を参照してください。
- LOB ロケータでの操作は、表 3-13 を参照してください。
- LOB バッファリングについては、表 3-14 を参照してください。
- LOB のオープンおよびクローズは、表 3-15 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 3-9 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()`	指定の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()`	バッファのデータを LOB の終わりに書き込みます。

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

表 3-10 OCI: 内部 LOB および外部 LOB（BFILE）の値の読込みまたはテストを行う関数

関数 / プロシージャ	説明
OCILobGetChunkSize()	読込みおよび書込み用にチャンク・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 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。

OCI: 一時 LOB を操作する関数

表 3-11 OCI: 一時 LOB を操作する関数

関数 / プロシージャ	説明
OCILobCreateTemporary()	一時 LOB を作成します。
OCILobIsTemporary()	一時 LOB が存在するかどうかを確認します。
OCILobFreeTemporary()	一時 LOB を解放します。

OCI: BFILE 固有の読込み専用関数

表 3-12 OCI: BFILE 固有の読込み専用関数

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確認します。
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確認します。
OCILobFileOpen()	BFILE をオープンします。

OCI: LOB ロケータ関数

表 3-13 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確認します。
OCILobLocatorIsInit()	LOB ロケータが初期化されているかどうかを確認します。

OCI: LOB バッファリング関数

表 3-14 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()	後続の LOB データの読込み / 書込みに、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 LOB および外部 LOB をオープンおよびクローズする関数

表 3-15 OCI: 内部 LOB および外部 LOB をオープンおよびクローズする関数

関数 / 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確認します。
OCILobClose()	LOB をクローズします。

OCI の例ー LOB がオープンしているかどうかの確認 : main() および seeIfLOBIsOpen

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、seeIfLOBIsOpen を例に説明します。

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

  return 0;
}

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

  switch (status)
  {
  case OCI_SUCCESS:
    break;
  case OCI_SUCCESS_WITH_INFO:
```

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

/* Select the locator into a locator variable */

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

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

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

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

  return (0);
}

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

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

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

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

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

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

C++（OCCI）を使用した LOB の作業

Oracle C++ Call Interface（OCCI）は、Oracle データベース内のデータを操作するための C++ API です。OCCI は、使いやすい C++ クラスのコレクションとして構成されています。これらの C++ クラスによって、C++ プログラムで、データベースへの接続、SQL 文の実行、データベース表への値の挿入とデータベース表の値の更新、問合せの結果の取出し、データベース内でのストアド・プロシージャの実行、データベース・スキーマ・オブジェクトのメタデータへのアクセスを行うことができます。また、OCCI は、ユーザー定義型オブジェクトを C++ クラスのインスタンスとして操作するためのシームレスなインタフェースを提供します。

OCCI は、OCI とともに使用することによって、アプリケーションを構築できます。

OCCI API には、JDBC および ODBC と比べて次のメリットがあります。

- OCCI には JDBC より多くの Oracle 機能があります。OCCI は、JDBC にないすべての OCI 機能を提供します。
- OCCI は、コンパイル型のパフォーマンスを提供します。コンパイル型のプログラムでは、ソース・コードは、最初からできるかぎりマシン言語に近いコードで記述されています。JDBC はインタプリタ型 API であるため、コンパイル型 API ほどのパフォーマンスを提供できません。インタプリタ型のプログラムでは、コードの各行を 1 行ずつマシン言語に近いコードに解析する必要があるため、パフォーマンスが低下します。
- OCCI では、スマート・ポインタを使用してメモリーを管理できます。OCCI オブジェクトのメモリー管理に注意する必要がありません。そのため、アプリケーション・コードのパフォーマンスが大幅に向上します。
- OCCI のナビゲーショナル・アクセスによって、直観的にオブジェクトにアクセスし、メソッドをコールできます。オブジェクトへの変更を持続するために、対応する SQL 文を記述する必要はありません。クライアント側のキャッシュを使用する場合、オブジェクト・インタフェースよりナビゲーショナル・インタフェースの方がパフォーマンスが高くなります。
- ODBC に関しては、OCCI API の方が簡単に使用できます。ODBC は C 言語で構築されていますが、OCCI には、C にはない C++ のすべてのメリットがあります。さらに、ODBC の習得は簡単ではないとされています。それに対して OCCI は、使いやすさを基に設計されています。

OCCI 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 LOB および外部 LOB（BFILE）からの読込み
- 内部 LOB への書込み

各 LOB 型に対する個別のクラス

OCI は BLOB、CLOB および BFILE の操作に共通の API を使用しますが、OCCI には、次のとおり各 LOB 型に対して個別のクラスがあります。

- **`OCCIClob` クラス**: 内部 CLOB および内部 NCLOB に格納されているデータにアクセスし、データを変更します。
- **`OCCIBlob` クラス**: 内部 BLOB に格納されているデータにアクセスし、データを変更します。
- **`OCCIBfile` クラス**: 外部 LOB（BFILE）に格納されているデータにアクセスし、データを読み込みます。

OCCIClob クラス

`OCCIClob` ドライバは、SQL ロケータ（CLOB）を使用して CLOB オブジェクトを実装します。これは、CLOB オブジェクトに、データ自体ではなく SQL CLOB データへの論理ポインタが含まれていることを意味します。

CLOB インタフェースでは、SQL CLOB 値の長さの取得、クライアント上の CLOB 値のマテリアライズ化、および部分文字列の取得を実行するためのメソッドが提供されます。OCCIResultSet インタフェースおよび OCCIStatement インタフェースの `getClob()`、`setClob()` などのメソッドを使用すると、SQL CLOB 値にアクセスできます。

OCCIBlob クラス

`OCCIResultSet` インタフェースおよび `OCCIStatement` インタフェースの `getBlob()` や `setBlob()` などのメソッドを使用すると、SQL BLOB 値にアクセスできます。`OCCIBlob` インタフェースでは、SQL BLOB 値の長さの取得、クライアント上の BLOB 値のマテリアライズ化、および BLOB の一部の抽出を実行するためのメソッドが提供されます。

表 3-16 ～表 3-20 に、これらのメソッドを示します。

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、『Oracle C++ Call Interface 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、『Oracle9*i* Database グローバリゼーション・サポート・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（文字またはバイト）

固定幅キャラクタ・セットの規則

OCCI 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。

- `OCCIClob`: オフセット・パラメータおよび量パラメータは、常に文字で表されます。
- `OCCIBlob`: オフセット・パラメータおよび量パラメータは、常にバイトで表されます。
- `OCCIBfile`: オフセット・パラメータおよび量パラメータは、常にバイトで表されます。

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。
  - `OCCIClob()`: 文字で表されます。
  - `OCCIBlob()`: バイトで表されます。
  - `OCCIBfile()`: 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。
  - `OCCIClob`: サーバー側 LOB を参照する場合は文字で表されます。
  - `OCCIBlob`: クライアント側バッファを参照する場合はバイトで表されます。
  - `OCCIBfile`: クライアント側バッファを参照する場合はバイトで表されます。
- **`length()`**: 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。
  - `OCCIClob.length()`: 文字で表されます。
  - `OCCIBlob.length()`: バイトで表されます。
  - `OCCIBfile.length()`: バイトで表されます。
- **`OCCIClob.read()`** および **`OCCIBlob.read()`**: 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、OCCI バッファ・パラメータ `buffer` に読み込まれたバイト数を表します。

- **OCCIClob.write()** および **OCCIBlob.write()**: 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、OCCI 入力バッファ buffer 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 OCCI 操作のためのオフセット・パラメータおよび量パラメータ

他のすべての OCCI 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。

- OCCIClob.copy()
- OCCIClob.erase()
- OCCIClob.trim()
- LoadFromFile 機能用にオーバーロードされた OCCICLob.copy()

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：**『Oracle9*i* Database グローバリゼーション・サポート・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。

OCCIClob.copy() および OCCIBlob.copy() でのファイルからのロード : 量パラメータ

OCCI の LoadFromFile 機能は、OCCIClob.copy() メソッドおよび OCCIBlob.copy() メソッドを介して提供されます。これらのメソッドは、OCCIBfile 引数を取ります。

BFILE の長さより長い値を指定することはできません。BFILE の長さより小さいサイズの量パラメータを指定する必要があります。

OCCIClob.read()、OCCIBlob.read() および OCCIBfile.read(): 量パラメータ

OCCIClob、OCCIBlob および OCCIBfile で読み込む場合、量パラメータを 4GB - 1 に指定します。これによって、LOB の終わりまでが読み込まれます。

OCCI の詳細

参照：『Oracle C++ Call Interface プログラマーズ・ガイド』を参照してください。

OCCI: BLOB、CLOB、NCLOB および BFILE を操作するメソッド

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

- 内部 LOB の変更は、表 3-16 を参照してください。
- LOB 値の読込みまたはテストは、表 3-17 を参照してください。
- 外部 LOB（BFILE）における読込み専用メソッドは、表 3-18 を参照してください。
- 他の LOB OCCI メソッドは、表 3-19 を参照してください。
- LOB のオープンおよびクローズは、表 3-20 を参照してください。

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

表 3-16 OCCIClob および OCCIBlob: 内部 LOB（BLOB、CLOB および NCLOB）の値を変更するメソッド

関数 / プロシージャ	説明
`OCCIBlob.append()`	CLOB または BLOB の値を別の LOB に追加します。
`OCCIBlob.copy()`	CLOB または BLOB の全体または一部を別の LOB にコピーします。
`OCCIBlob.copy()`	BFILE データを内部 LOB にロードします。
`OCCIBlob.trim()`	LOB または BLOB を切り捨てます。
`OCCIBlob.write()`	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。

OCCI: 内部 LOB および BFILE の値の読込みまたはテストを行うメソッド

表 3-17 OCCIBlob、OCCIClob および OCCIBfile: 内部 LOB および外部 LOB（BFILE）の値の読込みまたはテストを行うメソッド

関数 / プロシージャ	説明
OCCIBlob.getChunkSize()	読込みおよび書込み用にチャンク・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。
OCCIBlob.length()	LOB または BFILE の長さを戻します。
OCCIBlob.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。

OCCI: BFILE 固有の読込み専用メソッド

表 3-18 OCCI: BFILE 固有の読込み専用メソッド

関数 / プロシージャ	説明
OCCIBfile.close()	オープンしている BFILE をクローズします。
OCCIBfile.fileExists()	BFILE が存在するかどうかを確認します。
OCCIBfile.getFileName()	BFILE の名前を戻します。
OCCIBfile.getDirAlias()	ディレクトリ別名を取得します。
OCCIBfile.isOpen()	BFILE がオープンしているかどうかを確認します。
OCCIBfile.open()	BFILE をオープンします。

OCCI: 他の LOB メソッド

表 3-19 OCCI: 他の LOB メソッド

メソッド	説明
OCCIClob/Blob/Bfile.=	LOB ロケータを別の LOB ロケータに割り当てます。= またはコピー・コンストラクタを使用します。
OCCIClob.getCharSetForm()	LOB のキャラクタ・セット・フォームを戻します。
OCCIClob.getCharSetId()	LOB のキャラクタ・セット ID を戻します。
OCCIBfile.setName()	BFILE の名前を設定します。
OCCIClob/Blob/Bfile.IsEqual()	2 つの LOB が同じ LOB を参照しているかどうかを確認します。
OCCIClob/Blob/Bfile.isInitialized()	LOB が初期化されているかどうかを確認します。

OCCI: 内部 LOB および外部 LOB をオープンおよびクローズするメソッド

表 3-20 OCCI: 内部 LOB および外部 LOB をオープンおよびクローズするメソッド

関数 / プロシージャ	説明
`OCCIClob/Blob/Bfile.Open()`	LOB をオープンします。
`OCCIClob/Blob/Bfile.IsOpen()`	LOB がオープンしているかどうかを確認します。
`OCCIClob/Blob/Bfile.Close()`	LOB をクローズします。

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、終わりの部分を変更できます。内部 LOB と外部 LOB の両方に読込み目的のアクセスが可能で、内部 LOB には書込みも可能です。

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 3-21 ～表 3-27 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、『Pro*C/C++ Precompiler 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 文を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、次の章にも記載されています。

- 第 10 章「内部永続 LOB」
- 第 11 章「一時 LOB」
- 第 12 章「外部 LOB（BFILE）」

これらの Pro*C/C++ LOB サンプル・スクリプトには、$ORACLE_HOME/rdbms/demo/lobs/ からアクセスできます。

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 3-21 を参照してください。
- LOB 値の読込みまたはテストは、表 3-22 を参照してください。
- 一時 LOB の作成または解放、または一時 LOB が存在するかどうかの確認は、表 3-23 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 3-24 を参照してください。
- LOB ロケータでの操作は、表 3-25 を参照してください。
- LOB バッファリングについては、表 3-26 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 3-27 を参照してください。

Pro*C/C++: 内部 LOB の値を変更する埋込み SQL 文

表 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

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 の終わりに書き込みます。

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

表 3-22 Pro*C/C++: 内部 LOB および外部 LOB の値の読込みまたはテストを行う埋込み SQL 文

文	説明
DESCRIBE [CHUNKSIZE]	書込み用のチャンク・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。
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++: 一時 LOB を操作する埋込み SQL 文

表 3-23 Pro*C/C++: 一時 LOB を操作する埋込み SQL 文

文	説明
CREATE TEMPORARY	一時 LOB を作成します。
DESCRIBE [ISTEMPORARY]	LOB ロケータが一時 LOB を参照しているかどうかをチェックします。
FREE TEMPORARY	一時 LOB を解放します。

Pro*C/C++: BFILE 固有の埋込み SQL 文

表 3-24 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確認します。
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 3-25 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 3-26 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	後続の LOB データの読込み / 書込みに、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 3-27 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確認します。
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、終わりの部分を変更できます。内部 LOB と外部 LOB の両方に読込み目的のアクセスが可能で、内部 LOB には書込みも可能です。

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 3-28 ～表 3-34 に示します。詳細は、この章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。埋込み SQL LOB 文を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、次の章にも記載されています。

- 第 10 章「内部永続 LOB」
- 第 11 章「一時 LOB」
- 第 12 章「外部 LOB（BFILE）」

これらの Pro*COBOL LOB のサンプル・スクリプトには、$ORACLE_HOME/rdbms/demo/lobs/ からアクセスできます。

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 OS 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、『Pro*COBOL Precompiler 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 3-28 を参照してください。
- 内部 LOB および外部 LOB の値の読込みまたはテストは、表 3-29 を参照してください。
- 一時 LOB の作成、解放、または LOB ロケータの確認は、表 3-30 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 3-31 を参照してください。
- LOB ロケータでの操作は、表 3-32 を参照してください。
- LOB バッファリングについては、表 3-33 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 3-34 を参照してください。

Pro*COBOL: 内部 LOB の値を変更する埋込み SQL 文

表 3-28 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 の終わりに書き込みます。

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

表 3-29 Pro*COBOL: 内部 LOB および外部 LOB の値の読込みまたはテストを行う埋込み SQL 文

文	説明
DESCRIBE [CHUNKSIZE]	書込み用のチャンク・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: 一時 LOB を操作する埋込み SQL 文

表 3-30 Pro*COBOL: 一時 LOB を操作する埋込み SQL 文

文	説明
CREATE TEMPORARY	一時 LOB を作成します。
DESCRIBE [ISTEMPORARY]	LOB ロケータが一時 LOB を参照しているかどうかをチェックします。
FREE TEMPORARY	一時 LOB を解放します。

Pro*COBOL: BFILE 固有の埋込み SQL 文

表 3-31 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確認します。
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 3-32 Pro*COBOL: 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*COBOL: LOB バッファリング埋込み SQL 文

表 3-33 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 データの読込み / 書込みに、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 BFILE をオープンおよびクローズするための埋込み SQL 文

表 3-34 Pro*COBOL: 内部 LOB および 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確認します。
CLOSE	LOB または BFILE をクローズします。

Visual Basic（OO4O）を使用した LOB の作業

OO4O は、プログラム可能な COM オブジェクトのコレクションです。これによって、Oracle データベースとの通信用に設計されたアプリケーションの開発が簡単になります。OO4O は、データベースへのアクセスにおいて高いパフォーマンスを実現します。また、他の ODBC または OLE DB ベースのコンポーネント（ADO など）からでは簡単かつ効率的に使用できなかった Oracle 固有の機能にも簡単にアクセスできます。

次のオブジェクト・インタフェースの 1 つを使用すると、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**: OS 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：** *Oracle*Blob および *Oracle*Clob 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

OO4O の構文および詳細は、OO4O オンライン・ヘルプを参照してください。OO4O は、Oracle9*i* Client for Windows 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、Oracle9*i* インストールの「Application Development」サブメニューから利用できます。ヘルプから特定のメソッドおよびプロパティを表示するには、「目次」タブから、>「OO4O オートメーション・サーバー・リファレンス」>「メソッド」または「プロパティ」を選択します。

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。

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

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

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

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

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

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

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

OraConnection.CommitTrans
```

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: LOB 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

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するメソッドおよびプロパティが含まれます。

- 第 10 章「内部永続 LOB」
- 第 12 章「外部 LOB（BFILE）」

OO4O LOB のサンプル・スクリプトには、%ORACLE_HOME%¥rdbms¥demo¥lobs¥ からアクセスできます。

> **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、OO4O オンライン・ヘルプを参照してください。OO4O は、Oracle9*i* Client for Windows 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。OO4O オンライン・ヘルプは、Oracle9*i* 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 3-35 を参照してください。
- 内部 LOB および外部 LOB の値の読込みまたはテストは、表 3-36 を参照してください。
- BFILE のオープンおよびクローズは、表 3-37 を参照してください。
- LOB バッファリングについては、表 3-38 を参照してください。
- LOB が NULL かどうかの確認、またはポーリング量の取得または設定を行うプロパティについては、表 3-39 を参照してください。
- 読込み専用 BFILE メソッドについては、表 3-40 を参照してください。
- BFILE プロパティについては、表 3-41 を参照してください。

OO4O: BLOB、CLOB および NCLOB の値を変更するメソッド

表 3-35 OO4O: 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。
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 LOB および外部 LOB の値の読込みまたはテストを行うメソッド

表 3-36 OO4O: 内部 LOB および外部 LOB の値の読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 3-37 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。

OO4O: 内部 LOB バッファリング・メソッド

表 3-38 OO4O: 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、データベースへフラッシュします。
OraClob.FlushBuffer	CLOB バッファリング・サブシステムへの変更を、データベース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。

OO4O: LOB プロパティ

表 3-39 OO4O: 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)	読込み / 書込みポーリング操作の合計を取得 / 設定します。
Offset(Read/Write)	読込み / 書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)	ポーリングの状態を戻します。戻される値は次のとおりです。 ■ ORALOB_NEED_DATA: 読込みまたは書込みを行うデータが残っています。 ■ ORALOB_NO_DATA: 読込みまたは書込みを行うデータは残っていません。 ■ ORALOB_SUCCESS_LOB: データは正常に読込み / 書込みされました。
Size(Read)	LOB データの長さを戻します。

OO4O: 外部 LOB（BFILE）固有の読込み専用メソッド

表 3-40 OO4O: 外部 LOB（BFILE）固有の読込み専用メソッド

メソッド	説明
OraBFile.Close	オープンしている BFILE をクローズします。
OraBFile.CloseAll	オープンしているすべての BFILE をクローズします。
OraBFile.Open	BFILE をオープンします。
OraBFile.IsOpen	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 3-41 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
OraBFile.DirectoryName	サーバー側のディレクトリ別名を取得 / 設定します。
OraBFile.FileName(Read/Write)	サーバー側のファイル名を取得 / 設定します。
OraBFile.Exists	BFILE が存在するかどうかを確認します。

Java（JDBC）を使用した LOB の作業

Java（JDBC）を使用して、LOB に対して次の作業を行うことができます。

- Java を使用した内部永続 LOB の変更
- Java を使用した内部永続 LOB および外部 LOB（BFILE）の読込み
- Java（JDBC）からの DBMS_LOB パッケージのコール
- Java（JDBC）を使用した LOB の参照

Java を使用した内部永続 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 を使用した内部永続 LOB および外部 LOB（BFILE）の読込み

JDBC インタフェースでは、Java を使用して内部永続 LOB と外部 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 LOB 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。

Java（JDBC）からの 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。この方法は、このマニュアルの例で繰り返し使用されています。

Java（JDBC）を使用した 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

LOB での JDBC の使用に関する JDBC 構文および詳細は、次のドキュメントを参照してください。

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、『Oracle9*i* 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。
- 『Oracle9*i* SQLJ 開発者ガイドおよびリファレンス』を参照してください。

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 3-42 を参照してください。
  - BLOB 値の読込みまたはテストは、表 3-43 を参照してください。
  - BLOB バッファリングについては、表 3-44 を参照してください。
- 一時 BLOB
  - BLOB の作成、BLOB がオープンしているかどうかの確認および一時 BLOB の解放については、表 3-52 を参照してください。
  - BLOB のオープン、クローズ、および BLOB がオープンしているかどうかの確認は、表 3-52 を参照してください。
  - BLOB の切捨ては、表 3-55 を参照してください。
  - BLOB のストリーミング API については、表 3-57 を参照してください。
- CLOB
  - CLOB 値の読込みまたはテストは、表 3-46 を参照してください。
  - CLOB バッファリングについては、表 3-47 を参照してください。
  - CLOB 値の変更は、表 3-57 を参照してください。
- 一時 CLOB
  - CLOB のオープン、クローズおよび CLOB がオープンしているかどうかの確認は、表 3-53 を参照してください。
  - CLOB の切捨ては、表 3-56 を参照してください。
  - CLOB のストリーミング API については、表 3-58 を参照してください。
- BFILE
  - BFILE の読込みまたはテストは、表 3-48 を参照してください。
  - BFILE バッファリングについては、表 3-49 を参照してください。
  - BFILE のオープン、クローズおよび BFILE がオープンしているかどうかの確認は、表 3-54 を参照してください。
  - BFILE のストリーミング API については、表 3-59 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 3-42 JDBC: BLOB 値を変更する oracle.sql.BLOB メソッド

メソッド	説明
`int putBytes(long, byte[])`	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 値の読込みまたはテストを行う oracle.sql.BLOB メソッド

表 3-43 JDBC: BLOB 値の読込みまたはテストを行う oracle.sql.BLOB メソッド

メソッド	説明
`byte[] getBytes(long, int)`	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。
`long position(byte[],long)`	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。
`long position(Blob,long)`	指定された BLOB を LOB 内で検索します。
`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 のチャンク・サイズを戻します。

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 3-44 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 3-45 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 値の読込みまたはテストを行う oracle.sql.CLOB メソッド

表 3-46 JDBC: CLOB 値の読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。
`long position(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 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 3-47 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

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）値の読込みまたはテストを行う oracle.sql.BFILE メソッド

表 3-48 JDBC: 外部 LOB（BFILE）値の読込みまたはテストを行う oracle.sql.BFILE メソッド

メソッド	説明
`byte[] getBytes(long, int)`	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。
`int getBytes(long, int, byte[])`	BFILE のサブセットをバイト配列に読み込みます。
`long position(oracle.sql.BFILE, long)`	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。
`long position(byte[], long)`	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。
`boolean equals(java.lang.Object)`	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。
`long length()`	BFILE の長さを戻します。
`boolean fileExists()`	この BFILE が参照している OS ファイルが存在するかどうかを確認します。
`public void openFile()`	この BFILE が参照している OS ファイルをオープンします。
`public void closeFile()`	この BFILE が参照している OS ファイルをクローズします。
`public boolean isFileOpen()`	この BFILE がすでにオープンしているかどうかを確認します。
`public java.lang.String getDirAlias()`	この BFILE のディレクトリ別名を取得します。
`public java.lang.String getName()`	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 3-49 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

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

JDBC: Oracle8*i* リリース 8.1.x 以上で使用できない 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 固有の機能です。Oracle8*i* リリース 8.1.x 以上では、`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、Oracle8*i* 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 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。」

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、『Oracle9*i* 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、Oracle9*i* に付属の LOB のサンプル・コードを参照してください。

JDBC: 一時 LOB API

Oracle9*i* の JDBC ドライバには、一時 LOB を作成およびクローズするための API が含まれます。これらの API によって、以前のリリースで使用していた DBMS_LOB PL/SQL パッケージの次のプロシージャを使用する必要がなくなります。

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

表 3-50 JDBC: 一時 BLOB API

メソッド	説明
`public static BLOB createTemporary(Connection conn, boolean cache, int duration) throws SQLException`	一時 BLOB を作成します。
`public static boolean isTemporary(BLOB blob) throws SQLException`	指定の BLOB ロケータが一時 BLOB を参照するかどうかを確認します。
`public boolean isTemporary() throws SQLException`	現行の BLOB ロケータが一時 BLOB を参照するかどうかを確認します。
`public static void freeTemporary(BLOB temp_blob) throws SQLException`	指定の一時 BLOB を解放します。
`public void freeTemporary() throws SQLException`	一時 BLOB を解放します。

JDBC: LOB のオープンおよびクローズ

oracle.sql.CLOB クラスは、標準 JDBC java.sql.Clob インタフェースの Oracle JDBC ドライバの実装です。表 3-51 に、oracle.sql.CLOB にある、一時 CLOB にアクセスするための新しい Oracle 拡張 API を示します。

Oracle9*i* の JDBC ドライバには、明示的に LOB をオープンおよびクローズするための API が含まれます。これらの API によって、以前の DBMS_LOB.open() および DBMS_LOB.close() を使用する必要がなくなります。

表 3-51 JDBC: 一時 CLOB API

メソッド	説明
`public static CLOB createTemporary(Connection conn, boolean cache, int duration) throws SQLException`	一時 CLOB を作成します。
`public static boolean isTemporary(CLOB clob) throws SQLException`	指定の CLOB ロケータが一時 CLOB を参照するかどうかを確認します。
`public boolean isTemporary() throws SQLException`	現行の CLOB ロケータが一時 CLOB を参照するかどうかを確認します。
`public static void freeTemporary(CLOB temp_clob) throws SQLException`	指定の一時 CLOB を解放します。
`public void freeTemporary() throws SQLException`	一時 CLOB を解放します。

参照： 第 11 章「一時 LOB」を参照してください。

JDBC: BLOB のオープンおよびクローズ

oracle.sql.BLOB クラスは、標準 JDBC java.sql.Blob インタフェースの Oracle JDBC ドライバの実装です。表 3-52 に、oracle.sql.BLOB にある、BLOB をオープンおよびクローズするための Oracle 拡張 API を示します。これらの API は、今回のリリースで新しく追加されています。

表 3-52 JDBC:BLOB のオープンおよびクローズ

メソッド	説明
`public void open(int mode) throws SQLException`	BLOB をオープンします。
`public boolean isOpen() throws SQLException`	BLOB がオープンしているかどうかを確認します。
`public void close() throws SQLException`	BLOB をクローズします。

BLOB のオープン

JDBC アプリケーションは、oracle.sql.BLOB クラスに定義されている OPEN メソッドを使用して BLOB をオープンできます。OPEN メソッドの定義は次のとおりです。

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

MODE パラメータに指定可能な値は次のとおりです。

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

OPEN をコールするたびに BLOB がオープンします。次に例を示します。

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

BLOB がオープンしているかどうかの確認

JDBC アプリケーションは、oracle.sql.BLOB に定義されている ISOPEN メソッドを使用して、BLOB がオープンしているかどうかを確認できます。戻されたブール値は、以前に BLOB がオープンされているかどうかを示します。ISOPEN メソッドの定義は次のとおりです。

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

次に例を示します。

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

BLOB のクローズ

JDBC アプリケーションは、oracle.sql.BLOB に定義されている CLOSE メソッドを使用して BLOB をクローズできます。CLOSE API の定義は次のとおりです。

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

次に例を示します。

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

JDBC: CLOB のオープンおよびクローズ

oracle.sql.CLOB クラスは、標準 JDBC java.sql.Clob インタフェースの Oracle JDBC ドライバの実装です。表 3-53 に、oracle.sql.CLOB にある、CLOB をオープンおよびクローズするための新しい Oracle 拡張 API を示します。

表 3-53 JDBC: CLOB のオープンおよびクローズ

メソッド	説明
`public void open(int mode) throws SQLException`	CLOB をオープンします。
`public boolean isOpen() throws SQLExceptio`	CLOB がオープンしているかどうかを確認します。
`public void close() throws SQLException`	CLOB をクローズします。

CLOB のオープン

JDBC アプリケーションは、oracle.sql.CLOB クラスに定義されている OPEN メソッドを使用して CLOB をオープンできます。OPEN メソッドの定義は次のとおりです。

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

MODE パラメータに指定可能な値は次のとおりです。

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

OPEN をコールするたびに CLOB がオープンします。次に例を示します。

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

CLOB がオープンしているかどうかの確認

JDBC アプリケーションは、oracle.sql.CLOB に定義されている ISOPEN メソッドを使用して CLOB がオープンしているかどうかを確認できます。戻されたブール値は、以前に CLOB がオープンされているかどうかを示します。ISOPEN メソッドの定義は次のとおりです。

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

次に例を示します。

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

CLOB のクローズ

JDBC アプリケーションは、oracle.sql.CLOB に定義されている CLOSE メソッドを使用して CLOB をクローズできます。CLOSE API の定義は次のとおりです。

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

次に例を示します。

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

JDBC: BFILE のオープンおよびクローズ

oracle.sql.BFILE クラスは、データベース BFILE オブジェクトをラップします。表 3-54 に、oracle.sql.BFILE にある、CLOB をオープンおよびクローズするための新しい Oracle 拡張 API を示します。

表 3-54 BFILE をオープンおよびクローズするための JDBC 拡張 API

メソッド	説明
`public void open() throws SQLException`	BFILE をオープンします。
`public void open(int mode) throws SQLException`	BFILE をオープンします。
`public boolean isOpen() throws SQLException`	BFILE がオープンしているかどうかを確認します。
`public void close() throws SQLException`	BFILE をクローズします。

BFILE のオープン

JDBC アプリケーションは、oracle.sql.BFILE クラスに定義されている OPEN メソッドを使用して BFILE をオープンできます。OPEN メソッドの定義は次のとおりです。

```
/**
 * Open a external LOB in the readonly mode. It is an error
 * to open the same LOB twice.
 */
public void open () throws SQLException

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

MODE パラメータに指定可能な値は次のとおりです。

```
public static final int MODE_READONLY
```

OPEN をコールするたびに BFILE がオープンします。次に例を示します。

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

BFILE がオープンしているかどうかのチェック

JDBC アプリケーションは、oracle.sql.BFILE に定義されている ISOPEN メソッドを使用して BFILE がオープンしているかどうかを確認できます。戻されたブール値は、以前に BFILE がオープンされているかどうかを示します。ISOPEN メソッドの定義は次のとおりです。

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

次に例を示します。

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

BFILE のクローズ

JDBC アプリケーションは、oracle.sql.BFILE に定義されている CLOSE メソッドを使用して BFILE をクローズできます。CLOSE API の定義は次のとおりです。

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

次に例を示します。

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

使用例（OpenCloseLob.java）

```
/*
 * This sample shows how to open/close BLOB and CLOB.
 */

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

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

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

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

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

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

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

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

// Populate the table
stmt.execute (
    "insert into basic_lob_table values"
    + " ('one', '0101010101010101010101010101', 'onetwothreefour')");

    // Select the lobs
    ResultSet rset = stmt.executeQuery ("select * from basic_lob_table");
```

```
    while (rset.next ())
    {
      // Get the lobs
      BLOB blob = (BLOB) rset.getObject (2);
      CLOB clob = (CLOB) rset.getObject (3);

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

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

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

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

    // Close the ResultSet
    rset.close ();

    // Close the Statement
    stmt.close ();

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

JDBC を使用した LOB の切捨て

Oracle9*i* の JDBC ドライバには、LOB を切り捨てるための API が含まれています。これによって、以前の DBMS_LOB.trim() を使用する必要がなくなります。

JDBC:BLOB の切捨て

oracle.sql.BLOB クラスは、標準 JDBC java.sql.Blob インタフェースの Oracle JDBC ドライバの実装です。表 3-55 に、oracle.sql.BLOB にある、BLOB を切り捨てるための新しい Oracle 拡張 API を示します。

表 3-55 JDBC: BLOB の切捨て

メソッド	説明
public void trim(long newlen) throws SQLException	BLOB を切り捨てます。

TRIM API の定義は次のとおりです。

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

newlen パラメータに BLOB の新しい長さを指定します。

JDBC:CLOB の切捨て

oracle.sql.CLOB クラスは、標準 JDBC java.sql.Clob インタフェースの Oracle JDBC ドライバの実装です。表 3-56 に、oracle.sql.CLOB にある、CLOB を切り捨てるための新しい Oracle 拡張 API を示します。

表 3-56 JDBC: CLOB の切捨て

メソッド	説明
public void trim(long newlen) throws SQLException	CLOB を切り捨てます。

TRIM API の定義は次のとおりです。

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

newlen パラメータに CLOB の新しい長さを指定します。

> **参照：** 10-225 ページの「Java（JDBC）: LOB データの切捨て」を参照してください。

JDBC: 新しい LOB ストリーミング API

Oracle9*i* の JDBC ドライバには、要求された位置で Java ストリームからの LOB の読込みおよび書込みを行うための、新しい LOB ストリーミング API が含まれています。以前のリリースでは、LOB ストリーミング API でオフセットを指定できませんでした。

新しい JDBC BLOB ストリーミング API

`oracle.sql.BLOB` クラスは、標準 JDBC java.sql.Blob インタフェースの Oracle JDBC ドライバの実装です。表 3-57 に、`oracle.sql.BLOB` にある、要求された位置から BLOB の内容を操作するための新しい Oracle 拡張 API を示します。

表 3-57 JDBC: 新しい BLOB ストリーミング API

メソッド	説明
`public java.io.OutputStream getBinaryOutputStream (long pos) throws SQLException`	ストリームから BLOB に書き込みます。
`public java.io.InputStream getBinaryStream(long pos) throws SQLException`	BLOB からストリームとして読み込みます。

これらの API の定義は次のとおりです。

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

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

新しい CLOB ストリーミング API

oracle.sql.CLOB クラスは、標準 JDBC java.sql.Clob インタフェースの Oracle JDBC ドライバの実装です。表 3-58 に、oracle.sql.CLOB にある、要求された位置から CLOB の内容を操作するための新しい Oracle 拡張 API を示します。

表 3-58 JDBC: 新しい CLOB ストリーミング API

メソッド	説明
`public java.io.OutputStream getAsciiOutputStream (long pos) throws SQLException`	ASCII ストリームから CLOB に書き込みます。
`public java.io.Writer getCharacterOutputStream(long pos) throws SQLException`	文字ストリームから CLOB に書き込みます。
`public java.io.InputStream getAsciiStream(long pos) throws SQLException`	CLOB から ASCII ストリームとして読み込みます。
`public java.io.Reader getCharacterStream(long pos) throws SQLException`	CLOB から文字ストリームとして読み込みます。

これらの API の定義は次のとおりです。

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

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

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

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

新しい BFILE ストリーミング API

`oracle.sql.BFILE` クラスは、データベース BFILE をラップします。表 3-59 に、`oracle.sql.BFILE` にある、要求された位置から BFILE の内容を読み込むための新しい Oracle 拡張 API を示します。

表 3-59 JDBC: 新しい BFILE ストリーミング API

メソッド	説明
`public java.io.InputStream getBinaryStream(long pos) throws SQLException`	BFILE からストリームとして読み込みます。

これらの API の定義は次のとおりです。

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

JDBC BFILE ストリーミングの例（NewStreamLob.java）

注意： この例の Java コードの文字列（引用符で囲まれている）は、基本的には 1 行で記述しますが、複数行で記述されているものがあります（たとえば、stmt.execute の行）。このコードを使用する場合は、必ず文字列を 1 行で記述してください。

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

import java.io.*;

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

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

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

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

```
  // and use the default
}

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

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

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

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

// Populate the table
stmt.execute (
     "insert into basic_lob_table values"
     + "('one', '0101010101010101010101010101', 'onetwothreefour')");

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

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

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

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

```
    fillBlob (conn, blob, 11, 50);
  }
  rset.close ();

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

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

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

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

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

  // length of characters read
  int length = 0;

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

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

  // Close input stream
```

```
  instream.close();
}

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

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

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

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

  int i = 0;
  int chunk = 10;

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

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

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

    int i = 0;
    int chunk = 10;

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

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

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

JDBC および空の LOB

oracle.sql.BLOB にある次の API を使用すると、空の BLOB を作成できます。

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

同様に、oracle.sql.CLOB にある次の API を使用すると、空の CLOB を作成できます。

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

JDBC ドライバを使用すると、データベースをラウンドトリップさせることなく、空の LOB インスタンスを作成できます。空の LOB は、次の場合に使用できます。

- PreparedStatement の「設定」API
- 更新可能な結果セットの「更新」API
- STRUCT の属性値
- ARRAY の要素値

注意： 空の LOB は特別なマーカーとしての LOB であり、実際の LOB 値ではありません。

JDBC アプリケーションでは、前述の API で作成された空の LOB の読込みまたは書込みを行うことができません。アプリケーションで空の LOB の読込みまたは書込みを行うと、エラー・メッセージ「ORA-17098: 空の LOB 操作は無効です。」が表示されます。

OLEDB（OraOLEDB）

OraOLEDB は、OLE DB および ADO の開発者に高いパフォーマンス、および Oracle データへの効率的なアクセスを提供します。Visual Basic、C++ または任意の COM クライアントを使用してプログラムを作成する開発者は、OraOLEDB を使用して Oracle データベースにアクセスできます。

OraOLEDB は、Oracle のための OLE DB プロバイダです。OraOLEDB は、高いパフォーマンス、および LOB などの Oracle データへの効率的なアクセスを提供します。また、OraOLEDB を使用して特定の LOB 型に更新することもできます。

OraOLEDB は、次の LOB 型をサポートします。

- **永続 LOB:** 行セット全体での読込み / 書込み
- **BFILE**: 行セット全体での読込み専用
- **一時 LOB**: 行セット全体では未サポート

> **参照：** 第 13 章「OraOLEDB を使用した LOB の操作」を参照してください。

4

LOB の管理

この章の内容は次のとおりです。

ディレクトリ・オブジェクトおよび BFILE の使用規則

ディレクトリ・オブジェクトまたは BFILE を作成する場合、次の条件が満たされている必要があります。

- OS ファイルは、シンボリック・リンクまたはハード・リンクではない。
- Oracle ディレクトリ・オブジェクトに指定されている OS ディレクトリ・パスは、既存の OS ディレクトリ・パスである。
- Oracle ディレクトリ・オブジェクトに指定されている OS ディレクトリ・パスのコンポーネントに、シンボリック・リンクは含まれていない。

LOB を使用する前に必要な DBA アクション

この項では、LOB の作業の前に、ユーザーまたはデータベース管理者が行う必要があるアクションについて説明します。

オープンできる BFILE 数の上限の設定

1 回のセッションで同時にオープンが可能な 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 個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。すべてのオープンしているファイルをクローズするには、DBMS_LOB.FILECLOSEALL コールを使用します。

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「様々なプログラム環境での LOB のサポート」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。または、SQL VARCHAR2 ファンクションを使用して、CLOB の文字列操作を実行することもできます。

  参照： 第 7 章「モデリングおよび設計」を参照してください。

  利用例については、第 10 章「内部永続 LOB」の次の項を参照してください。

  - INSERT:
    - 10-16 ページの「EMPTY_CLOB() または EMPTY_BLOB() を使用した LOB 値の挿入」
    - 10-20 ページの「別の表からの LOB の選択による行の挿入」
    - 10-23 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:
    - 10-260 ページの「EMPTY_CLOB() または EMPTY_BLOB() を使用した LOB の更新」
    - 10-263 ページの「別の表からの LOB の選択による行の更新」
    - 10-265 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:
    - 10-274 ページの「LOB を含む表の行の削除」
- **外部 LOB（BFILE）**: Oracle8*i* 以上では、外部 LOB の読込み専用操作がサポートされています。詳細は、第 12 章「外部 LOB（BFILE）」を参照してください。
  - INSERT:
    - 12-22 ページの「BFILENAME() を使用した行の挿入」
    - 12-31 ページの「別の表からの BFILE の選択による BFILE 行の挿入」
    - 12-33 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込みができます。
  - 12-191 ページの「BFILENAME() を使用した BFILE の更新」
  - 12-194 ページの「別の表からの BFILE の選択による BFILE の更新」
  - 12-196 ページの「初期化した BFILE ロケータを使用した BFILE の更新」
- DELETE:
  - 12-228 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。

Oracle8*i* および Oracle9*i*

- **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` 構文には、`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。
- LOB 記憶域句には、`LOB_storage_clause` および `modify_LOB_storage_clause` の 2 種類があります。`ALTER TABLE MODIFY LOB` 文に指定できるのは、`modify_LOB_storage_clause` のみです。

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

- **ALTER TABLE ... MOVE の使用** : 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);
```

一時 LOB の管理

一時 LOB の管理およびセキュリティの問題は、第 11 章「一時 LOB」の次の項を参照してください。

- 11-11 ページの「一時 LOB の管理」
- 11-10 ページの「一時 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

SQL*Loader を使用して、LOB をバルク・ロードできます。LOB 型をロードする SQL*Loader 制御ファイル DDL の使用に関する詳細は、『Oracle9*i* データベース・ユーティリティ』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アウトラインに置く必要がある場合があります。LOBFILE を使用することによって、長いデータを分割できます。

LOBFILE

LOBFILE は、LOB ロードを簡単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す。
- ファイル全体の内容が読み込める単一の 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。

インライン LOB とアウトライン LOB

インライン LOB とは、プライマリ・データ・ファイルの値を持つ LOB です。

アウトライン LOB とは、LOBFILE の値を持つ LOB です。

SQL*Loader を使用したインラインおよびアウトライン・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インラインおよびアウトライン・データを内部 LOB にロードする手順を説明します。

- インライン LOB データのロード
  - 既定サイズ・フィールド内のインライン LOB データのロード
  - デリミタ付きフィールド内のインライン LOB データのロード
  - Length-Value Pair フィールド内のインライン LOB データのロード
- アウトライン 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・フィールド内のアウトライン LOB データのロード
  - デリミタ付きフィールド内のアウトライン LOB データのロード
  - Length-Value Pair フィールド内のアウトライン LOB データのロード

次の項目についても説明します。

- SQL*Loader による LOB のロードのヒント

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

インラインまたはアウトライン・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

参照 : 第 9 章「LOB: 効果的な使用方法」を参照してください。

インライン LOB データのロード

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

既定サイズ・フィールド内のインライン 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かぎりません。

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 を同じ長さにすることです。後続の空白の切捨てについては、『Oracle9*i* データベース・ユーティリティ』の「空白の切捨て」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
    APPEND
    (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 句ではなく NULLIF 句を使用する場合に生成されます。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。

> **注意：** NULLIF は DEFAULTIF より優先順位が高いですが、同一のフィールドに NULLIF と DEFAULTIF の両方を指定できます。

デリミタ付きフィールド内のインライン 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）にロードしても問題はありません。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。ローダーがデータ全体をスキャンし、デリミタ文字列を検索するため、このフォーマットでのロードには少し時間がかかります。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>' '<endrec>'
008,    <startlob>     Once upon another time ....The end. '<endlob>' '<endrec>'
```

Length-Value Pair フィールド内のインライン LOB データのロード

VARCHAR（『Oracle9*i* データベース・ユーティリティ』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。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>\r\n'"
INTO TABLE Multimedia_tab
APPEND
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）と記述され、Length 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 Length サブフィールドは 0（ゼロ）です（Value 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 LF にしてください。
- レコードが LF（UNIX の場合）または CR + LF（Windows の場合）で終了しているデータ・ストリームをロードする場合は、\n （UNIX の場合）または ¥r¥n （Windows の場合）を指定して、これらの文字が次のレコードの先頭と解釈されないようにする必要があります。

アウトライン LOB データのロード

この項の内容は次のとおりです。

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 をセカンダリ・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・インスタンスはフィールド（既定サイズ、デリミタ付き、Length-Value Pair）に存在しているとみなされます。ただし、これらのフィールドはレコードには編成されていません（LOBFILE には、レコードという概念はありません）。したがって、レコードの処理に伴う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,
```

セカンダリ・データ・ファイル（FirstStory.txt）

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

セカンダリ・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 がファイル全体にまたがることをローダーに伝えます。
- SQL*Plus の場合、SQL*Plus の long パラメータを使用して CLOB データを表示すると、long パラメータのデフォルト値が 80 であるため、表示されるのは CLOB の最初の 80 文字のみです。
- 詳細は、『Oracle9*i* データベース・ユーティリティ』を参照してください。

既定サイズ・フィールド内のアウトライン 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,
```

セカンダリ・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 2,000 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アウトライン 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。ローダーがデータ全体をスキャンし、デリミタ文字列を検索するため、このフォーマットでのロードには少し時間がかかります。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>\n")
```

Windows の場合は、次の文字列を使用して制御ファイルを終了します。

```
TERMINATED BY "<endlob>¥r¥n")
```

データ・ファイル（sample5.dat）

```
007,
008,
```

セカンダリ・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。
- レコードが LF（UNIX の場合）または CR + LF（Windows の場合）で終了しているデータ・ストリームをロードする場合は、\n（UNIX の場合）または ¥r¥n（Windows の場合）を指定してこれらの文字が次のレコードの先頭と解釈されないようにする必要があります。

Length-Value Pair フィールド内のアウトライン LOB データのロード

LOBFILE 内の各 LOB の前に長さを指定する必要があります。VARCHAR（『Oracle9*i* データベース・ユーティリティ』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。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,
```

セカンダリ・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）。

SQL*Loader による LOB のロードのヒント

- SQL*Loader の従来型パス・ロードでは、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。

  SQL*Loader のダイレクト・パス・ロードでは、LOB が空になったり、切り捨てられる可能性があります。LOB は、ロード時にピース単位でサーバーに送信されます。LOB ピースにエラーがある場合、そのピースは廃棄され、その LOB の残りはロードされません。エラーのある LOB 全体が最初のピースに含まれる場合、LOB 列は空になるか、切り捨てられます。

- 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。

- SQL*Loader のダイレクト・パス・ロードを使用する場合、LOB のロードに大量のメモリーが使用されます。LOB のロード時に、メッセージ「SQL*Loader-00700 必須割当て中にメモリー不足が発生しました [*number*]」（*number* には数値が入ります）が表示された場合、内部コードによってロードのコール 1 回当たりにバッチされる行の数が、OS およびプロセス・メモリーがサポートする数を超えている可能性があります。この問題を解決するには、ROWS オプションを使用して、データ・セーブ 1 回当たりに読み込む行数を少なくします。

- ダイレクト・パス API を使用して、LOB をロードすることもできます。

参照：

- 『Oracle9*i* データベース・ユーティリティ』の第 7 章および第 9 章を参照してください。
- 第 9 章「LOB: 効果的な使用方法」および「SQL*Loader の使用」を参照してください。

LOB の制限

この項では、LOB の制限の詳細について説明します。

参照：

- 特定のリリースで削除された制限については、xxxix ページの「ラージ・オブジェクト（LOB）の新機能」を参照してください。
- 5-31 ページの「パーティション化された索引構成表内の LOB に対する制限」を参照してください。
- クラスタ化表、レプリケーション、トリガー、ドメイン索引およびファンクション索引における LONG から LOB への移行の制限については、8-10 ページの「LONG から LOB への移行に関する制限事項」を参照してください。

LOB 列には次の制限があります。

- 分散 LOB はサポートされていません。このため、問合せの SELECT 句や WHERE 句または DBMS_LOB パッケージのファンクションではリモート・ロケータを使用できません。

  次の構文は、LOB ではサポートされていません。

```
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;
```

  この文を実行すると、「ORA-22992 リモート表から選択された LOB ロケータは使用できません。」というエラー・メッセージが表示されます。

  ただし、LOB を参照する問合せの別の部分では、リモート・ロケータを使用できます。次の構文は、リモート LOB 列ではサポートされています。

```
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_site ...
UPDATE table1@remote_site ...
DELETE FROM table1@remote_site ...
```

  副問合せが含まれている最初の 3 つの文では、選択リストにスタンドアロンの LOB 列のみを指定できます。LOB に対する SQL ファンクションまたは DBMS_LOB API は、サポートされていません。たとえば、次の文はサポートされています。

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

ただし、次の文はサポートされていません。

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

- クラスタには、キー列またはキー列以外の列として、LOB を含めることはできません。LOB を含めると、「ORA-02335 クラスタ列のデータ型が無効です。」というエラー・メッセージが表示されます。
- VARRAY の LOB は作成できません。作成しようとすると、「ORA-02348: 埋込み LOB で VARRAY 列を作成できません。」というエラー・メッセージが表示されます。
- 問合せの ORDER BY 句や GROUP BY 句または集計ファンクションには、LOB 列を指定できません。指定すると、「ORA-00932: 一貫性のないデータ型です。」というエラー・メッセージが表示されます。
- SELECT... DISTINCT 文、SELECT... UNIQUE 文または結合には、LOB 列を指定できません。ただし、列のオブジェクト型にその型で定義された MAP ファンクションまたは ORDER ファンクションが含まれている場合、SELECT... DISTINCT 文、または UNION または MINUS 集合演算子を使用する問合せには、オブジェクト型列の LOB 属性を指定できます。
- 表を作成する場合、オブジェクト型の属性として NCLOB を指定できません。ただし、メソッドには NCLOB パラメータを指定できます。
- ANALYZE... COMPUTE または ANALYZE... ESTIMATE 文には LOB 列を指定できません。
- LOB 列には、UPDATE DML トリガーを定義できません。
- LOB は主キー列として指定できません。
- LOB は索引キーの一部として指定できません。ただし、ファンクション索引のファンクション、またはドメイン索引の索引タイプ指定には、LOB 列を指定できます。また、Oracle Text を使用すると、CLOB 列に索引を定義できます。

  参照： ドメイン・インデックスに対するトリガーの定義については、『Oracle9*i* Data Cartridge Developer's Guide』を参照してください。

- INSERT 操作または UPDATE 操作では、任意のサイズのデータを LOB 列にバインドできますが、オブジェクト型の LOB 属性にはデータをバインドできません。INSERT... AS SELECT 操作では、LOB 列に最大 4,000 バイトのデータをバインドできます。

  参照：

  - 7-33 ページの「LOB に対する SQL セマンティクスのサポート」を参照してください。
  - LOB の使用に対するその他のセマンティクスについては、『Oracle9*i* SQL リファレンス』にある、個々の SQL 文における「キーワードとパラメータ」の項を参照してください。

- 表に LONG 列と LOB 列の両方がある場合、同一の SQL 文で、LONG 列と LOB 列の両方に 4,000 バイトを超えるデータをバインドすることはできません。ただし、LONG 列または LOB 列のいずれかには、4,000 バイトを超えるデータをバインドすることができます。
- **すべての LOB セグメントの最初のエクステント（初期エクステント）には、3 つ以上のブロックが必要です。**この制限は、Oracle リリース 8.0 で LOB が実装されてから適用されました。

  すべてのセグメントの最初のエクステント（初期エクステント）には、2 つ以上のブロックが必要です（FREELIST GROUPS が 0 の場合）。これは、最初のエクステントに 3 つ以上のブロックが必要な LOB セグメントとは異なります。永続ディクショナリで管理される表領域には、初期エクステントが 2 ブロックの LOB セグメントを作成できます。これは、永続ディクショナリで管理される表領域内のセグメントが、表領域のデフォルト記憶域設定をオーバーライドできるためです。

  ただし、ローカルで均一に管理される一時表領域やディクショナリで管理される一時表領域、またはローカルで管理される一時表領域内のエクステント・サイズが 2 ブロックである場合は、これらの表領域内に LOB セグメントを作成できません。これは、これらのタイプの表領域では、エクステント・サイズが固定され、表領域のデフォルト記憶域設定が無視されるためです。

  この場合、LOB セグメントを作成しようとすると、「ORA-03237 指定サイズの初期エクステントを割り当てられません。」というエラー・メッセージが表示されます。このエラーは、特に表領域に多くの空き領域がある場合は混乱しやすいため注意してください。
- Oracle9*i* では、共有サーバー（マルチスレッド・サーバー : MTS）・モードでの BFILE のセッション移行は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、共有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 操作が含まれる共有サーバーのセッションは、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、次回のリリースではなくなります。

注意： Oracle8*i* リリース 8.1.6 以上では、LOB に対する CACHE READS の設定がサポートされます。このような LOB を使用し、以前のリリースにダウングレードする場合、Oracle は警告を生成し、LOB を CACHE READS から CACHE LOGGING に変換します。その後、LOB を NOCACHE LOGGING または NOCACHE NOLOGGING のいずれかに変更できます。詳細は、7-9 ページの「CACHE / NOCACHE / CACHE READS」を参照してください。OCI 関数または DBMS_LOB ルーチンを使用して、LOB 列の値またはオブジェクト型列の LOB 属性を変更する場合、DML トリガーを定義した表では、DML トリガーが起動されません。

その他の一般的な LOB の制限は、次のとおりです。

- LOB は、レンジ・パーティション化された索引構成表でのみサポートされます。詳細は、1-6 ページの「パーティション化された索引構成表（PIOT）および LOB」を参照してください。
- クライアント PL/SQL プロシージャは、DBMS_LOB ルーチンをコールできません。詳細は、3-7 ページの「PL/SQL - LOB ガイドライン」を参照してください。
- `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。詳細は、3-7 ページの「PL/SQL - LOB ガイドライン」を参照してください。
- 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す。詳細は、3-7 ページの「PL/SQL - LOB ガイドライン」を参照してください。
- LOBFILE から読み込まれたフィールドは、句への引数としては使用できません。詳細は、4-5 ページの「LOBFILE」を参照してください。

参照：

- 4-15 ページの「SQL*Loader による LOB のロードのヒント」を参照してください。
- 5-31 ページの「パーティション化された索引構成表内の LOB に対する制限」を参照してください。

5

ラージ・オブジェクト（LOB）：詳細事項

この章の内容は次のとおりです。

注意：この章に示す例は、付録 B「マルチメディア・スキーマ」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 の概要 : 詳細事項

この章では、後の章で説明されている利用方法を補足し、詳細を説明します。ここで説明されている項目は、後の章の利用例を参照しておくと、より簡単に理解できます。

読込み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込みおよび更新処理と同様の、読込み一貫性メカニズムを LOB に対して提供しています。読込み一貫性に関する一般的な説明は、『Oracle9*i* データベース概要』を参照してください。LOB ロケータの場合、読込み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。これらの用途については次の項で説明します。

読込み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込み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込み一貫性のあるロケータとして存在します。読込み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 操作によって読込み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、次のこと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。
- 読込み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 と同様に読込み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てください。

LOB の更新および読込み一貫性

読込み一貫性のあるロケータでは、SELECT 実行時期に関係なく同じ LOB 値が提供されます。

次に、読込み一貫性と更新の関係を簡単な例で示します。付録 B「マルチメディア・スキーマ」で定義されている 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

次のコード例では、t1 ～ t6 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 および 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、同じ値を持つ読込み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）、これが読込み一貫性のあるロケータで、SELECT の実行時と同じ値が引き続き参照されることを示します。
- 同様に、clob_copied を介した値の DBMS_LOB.READ() は（t6）、これが読込み一貫性のあるロケータで、clob_selected と同じ値が引き続き参照されることを示します。

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_selected     CLOB;
  clob_updated      CLOB;
```

```
  clob_copied       CLOB;
  read_amount       INTEGER;
  read_offset       INTEGER;
  write_amount      INTEGER;
  write_offset      INTEGER;
  buffer            VARCHAR2(20);

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

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

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

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

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

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

  -- At time t4:
  write_amount := 3;
```

```
  write_offset := 5;
  buffer := 'efg';
  dbms_lob.write(clob_updated, write_amount, write_offset,
       buffer);

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

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

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

更新済ロケータを介した LOB の更新

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かぎり、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 値を更新する場合のみに、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 や 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 句を使用するかの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は、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** のピース単位操作を混合しないように注意してください。

> **参照：**『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

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

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

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

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

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

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

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 ～ t5 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。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

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

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

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

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

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

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


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

```
  -- At time t5:
  clob_copied := clob_updated;

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

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）が読込み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 ～ t5 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

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

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


  -- At time t2:
  clob_copied := clob_updated;


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

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

```
  -- clob_updated


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

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

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

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、読込み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。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、シリアル化可能なトランザクション内でない場合は、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、5-14 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。

- 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、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
          -- At time t1:
     SELECT      story
     INTO        clob_updated
     FROM        Multimedia_tab
     WHERE       clip_id = 1
     FOR UPDATE;
     read_amount := 10;
     read_offset := 1;
     dbms_lob.read(clob_updated, read_amount, read_offset,
          buffer);
     dbms_output.put_line('clob_updated value: ' || buffer);
     -- This produces the output 'abcd'

     -- At time t2:
     write_amount := 3;
     write_offset := 5;
```

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

    -- At time t3:
    COMMIT;

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

LOB 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 LOB 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。
  - DML 文の実行前にロケータを選択した場合 : 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。したがって、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 に対する読込みおよび書込み

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。
- シリアル化可能なトランザクションを含むロケータを使用して読込みまたは書込みができない場合 : 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ル化可能な場合、そのロケータを使用した読込みまたは書込みはできません。
- シリアル化可能でないトランザクションを含むロケータを使用して読込みはできるが、書込みはできない場合 : トランザクションがシリアル化可能でない場合、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、ロケータとシリアル化可能でないトランザクションの関係を示します。

シリアル化可能でない例 : 現行トランザクションを持たないロケータの選択

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 の読込みまたは書込み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のロケータの選択

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 データの読込みの詳細は、5-2 ページの「読込み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 の読込みまたは書込みを行います。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 データの読込みの詳細は、5-2 ページの「読込み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 属性および外部 LOB 属性には、次のようなオブジェクト・キャッシュの問題があります。

- **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 LOB に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、同一の LOB 値を参照する、同一のロケータが含まれることになります。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「LOB の更新および読込み一貫性」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（LBS）

Oracle8*i* および Oracle9*i* では、LBS を提供しており、Data Cartridge、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 回の読込みまたは書込み操作に対して指定できる最大サイズでもあります。

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込みおよび書込みを実行する（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込みが可能になります。LOB バッファへの書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 操作では、次のことに注意する必要があります。

- **LOB バッファの内容を明示的にフラッシュします。**Oracle8*i* および Oracle9*i* 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。Oracle 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込みまたは書込み操作のエラー・レポートは、次回のサーバー・ベースの LOB へのアクセスまで行われません。
- **LOB バッファリングは、シングル・ユーザー、シングル・スレッドです。**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、シングル・スレッドのシステムであるためです。
- **ロールバックするための論理セーブポイントを保持します。**Oracle 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・サブシステムを使用しない同一トランザクションの他の操作によって、LOB が更新されないことを確認します。**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。Oracle 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。
- **バッファリングを使用可能にした LOB ロケータを更新します。**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、読込み一貫性のある LOB を提供します。5-2 ページの「読込み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を介してのみ実行可能になるということです。Oracle 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。5-5 ページの「更新済ロケータを介した LOB の更新」を参照してください。

- **バッファリングを使用可能にした LOB ロケータに IN OUT または OUT パラメータを渡します。**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込み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。

- **複数のロケータが同じ LOB を指す場合は、1 つのみのロケータでバッファリングを使用可能にします。**複数の異なるロケータが同じ LOB を指す場合は、1 つのみのロケータでバッファリングが使用可能であることを確認する必要があります。複数のロケータでバッファリングが使用可能である場合、LOB の内容は保証されません。

- **バッファリングを使用可能にした LOB では、バイト値 0（ゼロ）の充填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。**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）の終わりから 1 バイト（または 1 文字）の場合にのみ可能です。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）の充填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、Oracle では、クライアント側とサーバーの LOB の、ロケータ・バインド変数のキャラクタ・セット・フォーム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方法

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 × CHUNKSIZE = 約 32KB 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 セッション当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込みまたは書込み操作に指定可能なサイズの最大値は 512KB で、それぞれの環境によって、読込み / 書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。Oracle8*i* および Oracle9*i* では、入力オフセット、および読込みまたは書込み操作用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込みまたは書込み操作によって、読込みまたは書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、指定された読込み / 書込みの量が 30000 の場合、Oracle 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、読込み / 書込みの量が 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、読込み / 書込みの量が 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、Oracle 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と、Oracle 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、Oracle 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。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。Oracle では、この状態をエラーとしてユーザーに通知します。Oracle では、変更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、Oracle 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 書込みのために使用されると、Oracle 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と、Oracle 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 に別の読込みまたは書込みを行うと、次のエラーが発生します。

```
Error 22280: 操作に必要なバッファは、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。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、読込み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、読込み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。両方とも読込み一貫性のあるロケータで、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、その時点で、別のバージョンの LOB 値を参照しています。サーバーの LOB を更新する場合は、L1 を使用して、L2 で得た読込み一貫性のある状態を維持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。読込み一貫性のあるロケータをフラッシュしようとすると、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込みまたは書込みでは、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・アウトされます。

---

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、読込み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薦めします。

- 外部コールアウトを呼び出します。
- 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込み / 書込みを行います。
- 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。

Oracle 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 がある場合のみに使用できます。

    `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 は読込み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。この割当てによって、ロケータを L2 に選択しなおすために、サーバーにアクセスする必要がなくなります。

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            bufl;

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

   init_OCI();

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

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

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

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

          -- Write 4K of the LOB throgh lbs_loc2 starting from
          -- offset 1:
```

```
    amount = 4096; offset = 1; bufl = 4096;
    buffer = populate_buffer(4096);
    OCILobWrite(.., lbs_loc2, offset, amount, buffer,
        bufl, ..);

if (exception)
    goto exception_handler;
    -- This will read the first 32K bytes of the LOB from
    -- the server into a new page (call it page_B) in the
    -- LOB's buffer, and modify the contents of this page
    -- with input buffer contents.
    -- lbs_loc2 is an updated locator.

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

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

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

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

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

if (exception)
```

```
        goto exception_handler;
        -- No two locators can be used to update a buffered LOB
        -- through the buffering subsystem

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

    if (exception)
        goto exception_handler;
        -- Locators enabled for buffering cannot be used with
        -- operations like Append, Copy, Trim etc.
        -- When done, flush LOB's buffer to the server:
    OCILobFlushBuffer(.., lbs_loc2, OCI_LOB_BUFFER_NOFREE);

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

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

    if (exception)
       goto exception_handler;
       -- This disables the three locators for buffered access,
       -- and frees up the LOB's buffer resources.
      exception_handler:
    handle_exception_reporting();
    exec_statement("rollback to savepoint lbs_savepoint");
}
```

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、付録 B「マルチメディア・スキーマ」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、10-7 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。

LOB 参照を含む VARRAY の作成 : 例

各マルチメディア・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、次の手順を実行します。

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;
```

パーティション化された索引構成表内の LOB

Oracle9*i* では、パーティション化された索引構成表（PIOT）内で、LOB、LOB として格納された VARRAY 列、および BFILE を使用できます。これらの表での LOB 列の動作は、従来の（ヒープ構成）パーティション表での動作に似ていますが、わずかに違います。

- **表領域のマッピング**

  LOB 列に対する表領域が、個々のパーティション・レベルまたは表レベルで指定されてない場合、そのパーティションに対する LOB のデータ・セグメントおよび索引セグメントは、索引構成表の対応するパーティションの主キー索引セグメントが作成された表領域に作成されます。これらのセグメントは、同一レベル・パーティション化され、対応する主キー索引セグメントと連結されます。

- **インラインとアウトライン**

  パーティション化された索引構成表がオーバーフロー・セグメントなしで定義された場合、LOB 列をインラインに作成するように定義または変更することはできません。この要件は、非パーティション索引構成表の要件と同じです。

LOB 列は、レンジ・パーティション化された索引構成表でのみサポートされます。

パーティション化された索引構成表内の LOB 列の例

この項では、付録 B の Multimedia_tab の例を使用して、パーティション化された索引構成表内の LOB に関する前述の相違点を示します。

Multimedia_tab は、レンジ・パーティション化された索引構成表として、次のように作成されたと想定します。

```
CREATE TABLE Multimedia_tab (
  CLIP_ID    INTEGER PRIMARY KEY,
  CLIP_DATE  DATE,
  STORY      CLOB,
  FLSUB      NCLOB,
  PHOTO      BFILE,
  FRAME      BLOB,
  SOUND      BLOB,
  ...
  )
  ORGANIZATION INDEX
    TABLESPACE TBS_IDX
  OVERFLOW
    TABLESPACE TBS_OVF
  LOB (FRAME, SOUND) STORE AS (TABLESPACE TBS_LOB)
  PARTITION BY RANGE (CLIP_DATE)
  (PARTITION Jan_Multimedia_tab VALUES LESS THAN (01-FEB-2000)
     LOB (STORY) STORE AS (TABLESPACE TBS_LOB),
   PARTITION Feb_Multimedia_tab VALUES LESS THAN (01-MAR-2000)
     LOB (FLSUB) STORE AS (TABLESPACE TBS_LOB
                           ENABLE STORAGE IN ROW)
  );
```

この例では、LOB 列 FRAME および SOUND は、TBS_LOB 表領域内のすべてのパーティションにまたがって格納されます。

- パーティション Jan_Multimedia_tab では、パーティション固有のオーバーライドによって、STORY 列が TBS_LOB に格納されます。ただし、FLSUB 列は、主キー索引セグメントの表領域である TBS_IDX に格納されます。
- パーティション Feb_Multimedia_tab では、STORY 列が TBS_IDX に格納されます。また、パーティション固有のオーバーライドによって、FLSUB 列が TBS_LOB に格納されます。

注意： Multimedia_tab がオーバーフロー・セグメント付きで作成されているため、LOB 列をインラインに格納することができます。オーバーフロー・セグメントがない場合、Feb_Multimedia_tab の FLSUB に対する ENABLE STORAGE IN ROW 句によってエラーが発生します。

残りの LOB 物理属性に対する継承のセマンティクスは、従来の表内に LOB を持つインライン形式になります。

参照： CREATE TABLE の lob_storage_clause の詳細は、『Oracle9*i* SQL リファレンス』を参照してください。

パーティション化された索引構成表内の LOB に対する制限

レンジ・パーティション化された索引構成表内の LOB に対する制限

サポートされない列型

レンジ・パーティション化された索引構成表内の次の列型はサポートされません。

- インライン LOB またはアウトライン LOB として格納された VARRAY 列
- LOB 属性を持つ抽象データ型（ADT）
- （LOB 属性を持つ）ADT の（LOB として格納された）VARRAY
- LOB 列を持つネストした表

オブジェクト・レンジ・パーティション化された索引構成表でサポートされない列型

オブジェクト・レンジ・パーティション化された索引構成表内の次の列型はサポートされません。

- LOB 属性を持つ ADT
- インライン LOB またはアウトライン LOB として格納された VARRAY 列
- （LOB 属性を持つ）ADT の（LOB として格納された）VARRAY
- LOB 列を持つネストした表

ハッシュ・パーティション化された索引構成表の LOB に対する制限

ハッシュ・パーティション化された索引構成表では、LOB 列がサポートされません。

6

LOB に関する FAQ

この章では、次に示すよくある質問（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？

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 つの手順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

たとえば、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 LONG RAW 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、いくつかの条件を確認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、どのように確認するのですか？ 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：この例は、トップレベルの LOB 列に対してのみ有効です。

LOB データの読込みおよびロード：量パラメータのサイズは？

質問

古いリリースの『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 の終わり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。読込み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。」

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、無効または範囲外です
```

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。

クラッシュ後に LOB データは失われますか？

質問

BLOB 列を含む表があります。高速化の目的で NOLOGGING を使用するため、BLOB チャンクが REDO ログに保存されません。サーバーがクラッシュするとどうなりますか？ リカバリ時、表データは失われたり、表が破損しますか？

回答

データ・ファイルに書き込まれていないすべての LOB データは、REDO ログからリカバリできません。再度ロードする必要があります。LOB 索引の更新は REDO に書き込まれるため、LOB が存在しない（REDO からリカバリされなかった）にもかかわらず、索引は LOB が存在すると示します。そのため、データ破損のエラーが発生することが考えられます。

索引構成表（IOT）および LOB

インライン記憶域は索引構成表の LOB に使用できますか？

質問

インライン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、インライン 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

Oracle9*i* では、データが 4KB 未満であるかぎり、INSERT 文を使用して、データによって 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 文を使用して、4KB 未満のデータによって LOB を更新することもできます。LOB が 4KB を超える場合、次の手順を実行します。

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() または 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。
- データを使用して LOB 値を初期化するか、EMPTY_BLOB()/EMPTY_CLOB() を使用して初期化するかは、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かぎり）その値はインラインに格納されます。LOB 値が 4KB よりも大きくなると、アウトラインに移されます。

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 Oracle9*i* 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ることです。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
       ('0101010101010101010101010101', 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 Oracle8*i* または Oracle9*i* で使用できますか？

質問

OracleBlob および OracleClob は、Oracle8*i* または Oracle9*i* 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。Oracle8*i* リリース 8.1.x（および Oracle9*i* 以上などの後続リリース）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 データにアクセスすると、標準エラー・メッセージが表示されます。たとえば、JDBC Thin Driver で LOB を操作しようとすると、次のエラーが表示されます。

「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。」

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、『Oracle9*i* 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。

Oracle JDBC Thin Driver で LOB を操作する方法は？

質問

Oracle JDBC Thin Driver で LOB を操作しようとすると、次のようなエラーが表示されました。

```
Dumping lobs
java.sql.SQLException: ORA-03115: unsupported network datatype or representation
at oracle.jdbc.ttc7.TTIoer.processError(TTIoer.java:181)
at oracle.jdbc.ttc7.Odscrarr.receive(Compiled Code)
at oracle.jdbc.ttc7.TTC7Protocol.describe(Compiled Code)
at oracle.jdbc.ttc7.TTC7Protocol.parseExecuteDescribe(TTC7Protocol.java: 516)
at oracle.jdbc.driver.OracleStatement.doExecuteQuery(OracleStatement.java:1002)
at oracle.jdbc.driver.OracleStatement.doExecute(OracleStatement.java:1163)
at oracle.jdbc.driver.OracleStatement.doExecuteWithTimeout(OracleStateme
nt.java:1211)
at oracle.jdbc.driver.OracleStatement.executeQuery(OracleStatement.java: 201)
at LobExample.main(Compiled Code)
---------------------------------
```

使用しているコードは、Oracle8 リリース 8.0.5 に付属の LobExample.java です。これは OCI8 のサンプルですが、使用している Thin Driver とインスタンスは、後続リリースのものです。

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。

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、『Oracle9*i* 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。

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。

DBMS_LOB.ERASE を使用すると何ができますか？

質問

DBMS_LOB.ERASE を使用すると何ができますか？

回答

CLOB のセグメントを消去するのみです。CLOB の短縮は行いません。そのため、DBMS_LOB.ERASE を使用した後も CLOB の長さは同じです。DBMS_LOB.TRIM を使用すると、CLOB を短縮できます。

putChars() を使用できますか？

質問

oracle.sql.CLOB.putChars() を使用できますか？

回答

使用できます。ただし、位置と長さの引数が正しいことを確認する必要があります。putChars をコールする、推奨の OutputStream インタフェースを使用することもできます。

JDBC で CLOB CharSetId を操作できますか？

質問

OCI には、CLOB CharSetId を操作する機能があります。JDBC に同等の機能がありますか？

回答

JDBC では、CLOB は常に USC2（Java の CHAR 型に対応する Oracle キャラクタ・セット）です。そのため、OCI CLOB CharSetId に同等のものはありません。

LONG RAW より BLOB への挿入に時間がかかるのはなぜですか？

質問

LONG RAW への挿入より BLOB への書込みに時間がかかるのはなぜですか？

回答

JDBC Thin ドライバを使用してデータを BLOB に挿入するには、DBMS_LOB パッケージが使用されるため時間がかかります。JDBC OCI 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。

LONG 列に LobLength を実行すると、ORA-03127 エラーが発生するのはなぜですか？

質問

LONG 列で LobLength を取得すると、ORA-03127 エラーが発生するのはなぜですか？

回答

これは正常な動作です。LONG 列は、所定の場所（インライン）でフェッチされません。LONG 列は他の場所でフェッチされ、明示的に読み込まれるまでパイプ内に存在します。この場合、LobLocator（getBlob()）を取得し、LONG 列を読み込む前に、この LOB の長さを取得しようとします。パイプ内は空でないため、前述の例外が発生します。これを解決するには、BLOB の操作を行う前に、LONG 列の読込みを完了してください。

Java プログラムで CLOB オブジェクトを作成する方法は？

質問

JDBC を使用して、CLOB で次の手順を実行する予定です。

1. Java プログラムで CLOB オブジェクトを作成します。
2. プログラムに渡された文字列の文字を CLOB に移入します。
3. CLOB をパラメータとして受信するストアド・プロシージャへのコールを準備します。
4. Java CLOB を使用して、パラメータを設定します。
5. 実行します。

`SQLUtil.makeOracleDatum()` メソッドを試しましたが、正常に動作しませんでした。型が無効であることを示すエラー・メッセージが表示されます。参照した唯一の Oracle 例では、SQL オブジェクトを使用して Oracle から CLOB オブジェクトを読み込むことによって、CLOB オブジェクトが作成されています。Java プログラムで CLOB を作成する必要があります。

回答

質問の方法では CLOB オブジェクトを作成できません。`oracle.sql.CLOB` クラスは、CLOB 用の実際のデータではなく、CLOB ロケータをカプセル化しますが、CLOB ロケータはデータベースから取得する必要があります。現在、クライアントに CLOB ロケータを構築する方法はありません。empty_clob() を表に挿入し、ロケータを取り出してから、データを CLOB に書き込む必要があります。

PL/SQL プロシージャは、この特定の機能に適切でない可能性があります。

PL/SQL パラメータを CLOB 型にする場合、PL/SQL パラメータは CLOB ロケータを表すため、他のインタフェースを使用して、データを CLOB に書き込む必要があります。また、すべてのデータを VARCHAR2 または LONG パラメータとして PL/SQL へ渡すことも問題です。これは、PL/SQL パラメータは 32KB に制限されており、この場合は実用的でないためです。

標準の JDBC API を使用して、CLOB を処理することをお薦めします。

クライアントの LOB をロードするアプリケーションおよび LOB へのデータ挿入で起動される単一のストアド・プロシージャに、CLOB を挿入するために必要なすべての機能をカプセル化する必要があります。

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。『Oracle9*i* データベース・ユーティリティ』、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・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 ドライバ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 ドライバを使用して 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 ドライバ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 の実行によって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 ドライバ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 の使用をお薦めします。LOB のストリーム・アクセスを使用することによって、JDBC ドライバは 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。

```
/*
 * This sample writes the GIF file john.gif to a BLOB.
 */
import java.sql.*;
import java.io.*;
import java.util.*;

// Importing the Oracle Jdbc driver package makes the code more readable
import oracle.jdbc.driver.*;

//needed for new CLOB and BLOB classes
import oracle.sql.*;

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

```
    // Connect to the database
    // You can put a database name after the @ sign in the connection URL.
    Connection conn =
      DriverManager.getConnection ("jdbc:oracle:oci8:@", "scott", "tiger");

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

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

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

    // Create a table containing a BLOB and a CLOB
    stmt.execute ("create table persons (name varchar2 (30), picture blob)");

    // Populate the table
    stmt.execute ("insert into persons values ('John', EMPTY_BLOB())");

    // Select the BLOB
    ResultSet rset = stmt.executeQuery ("select picture from persons where name =
'John'");
    if (rset.next ())
    {
      // Get the BLOB locator from the table
      BLOB blob = ((OracleResultSet)rset).getBLOB (1);

      // Declare a file handler for the john.gif file
      File binaryFile = new File ("john.gif");

      // Create a FileInputStream object to read the contents of the GIF file
      FileInputStream istream = new FileInputStream (binaryFile);

      // Create an OutputStram object to write the BLOB as a stream
      OutputStream ostream = blob.getBinaryOutputStream ();

      // Create a tempory buffer
      byte[] buffer = new byte[1024];
      int length = 0;
```

```
      // Use the read() method to read the GIF file to the byte
      // array buffer, then use the write() method to write it to
      // the BLOB.
      while ((length = istream.read(buffer)) != -1)
        ostream.write(buffer, 0, length);

      // Close the inputstream and outputstream
      istream.close();
      ostream.close();

      // Check the BLOB size
      System.out.println ("Number of bytes written = "+blob.length());
    }

    // Close all resources
    rset.close();
    stmt.close();
    conn.close();
  }
}
```

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 : 現行の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、名前の変更、再作成または変更ができないため、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。
- 一時 LOB:

  一時 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す。

  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 TABLESPACE 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、データは表のインラインに格納されます。約 4KB より大きくなると、LOB 値は指定された表領域に移されます。

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、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 からは書き込めません。
  - 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。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、繰返しの読込みおよび複数の読込みが高速化されます。
  - BLOB のピース単位および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

全表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されますか ? 4KB を超える BLOB は指定されたセグメントに移されますか ?

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 はインラインに格納されますか？

質問

『Oracle9*i* SQL リファレンス』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を引いた長さより小さければ、LOB 値がインラインに格納されます。これはデフォルト値です。

インライン 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インラインに格納されるため、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 値がある場合に、次のコマンド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 がある場合に、次のコマンド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。チャンク・サイズ以上の領域が必要です。

   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？

回答

第 7 章「モデリングおよび設計」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 を含む行を数千行挿入してください。

Oracle9*i* 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。

CLOB のインライン格納 : DISABLING STORAGE および使用領域

質問

CLOB を行の外にインライン格納することについて質問があります。LOB 列を含む表を作成する場合、DISABLE STORAGE IN ROW または ENABLE STORAGE IN ROW を指定できることを知っています。次の質問があります。

1. ENABLE STORAGE IN ROW を指定する場合、これはその行と同じブロックに LOB 情報を格納するということですか？
2. 両方の表には、CLOB 列用の別々のセグメントがあります。ただし、ENABLE STORAGE IN ROW を指定した表自体のサイズ（CLOB セグメントは含まない）は、DISABLE STORAGE IN ROW を指定した表のサイズよりかなり大きくなります。これはなぜですか？

回答

1. LOB 値が約 4KB 未満の場合、その値は表にインライン格納されます。行のサイズによって、行全体が 1 つのブロックに格納されるかどうかが決まります。大きい行は複数のブロックにまたがります。LOB が 4KB を超える場合、LOB 値は別のセグメントに格納されます。
2. これは、4KB 未満の LOB が、表のセグメントにインライン格納されるためです。

VARRAY 列を含む表を作成する場合、LOB 記憶域句を含める必要がありますか？

質問

VARRAY 列を含む表を作成する場合、LOB 記憶域を含める場合と含めない場合の影響は何ですか？ マニュアルには、VARRAY はそのサイズによって、インラインまたは LOB に格納されると記述されているため、LOB 記憶域を含めない場合は、こういう結果になると思います。LOB 記憶域を含めると、LOB は常に使用されますか？

なぜ LOB に名前を付けるのでしょうか？ ネストした表や記憶表に名前を付けるのは、索引付けや変更などを行うために有効であると理解していますが、LOB に名前を付ける必要性がわかりません。考えられるのは、LOB をキャッシュするために LOB を変更することのみです。

回答

マニュアルには、次のように記述されています。「VARRAY は、RAW 値または BLOB として列に格納されます。Oracle では、宣言された VARRAY の LIMIT を使用して計算された VARRAY の最大サイズに基づいて、VARRAY が定義された場合の格納方法を決定します。サイズが約 4,000 バイトを超える場合、VARRAY は BLOB に格納されます。サイズが 4,000 バイト未満の場合、VARRAY は RAW 値として列自体に格納されます。さらに、Oracle はインライン LOB をサポートしています。そのため、大規模 VARRAY の最初の 4,000 バイトに収まる（LOB ロケータ用に数バイトが予約されています）要素は、行自体の列に格納されます。」

したがって、データが約 4,000 バイト未満で、VARRAY に LOB 記憶域句を指定しない場合、そのデータはロー・データとしてインラインに格納されます。

また、『Oracle9*i* SQL リファレンス』には、次のように記述されています。

「varray_storage_clause: VARRAY が保存される LOB に対して、個々の記憶特性を指定すると、インラインに格納できるほど小さい値でも、VARRAY は必ず LOB に格納されます。」

したがって、varray_storage_clause を指定すると、VARRAY は常に LOB に格納されます。ただし、最初の段落に、VARRAY はインライン LOB もサポートしていると記述されているため、デフォルトで、最初の 4,000 バイトが表の行にインライン格納され、他のデータがインライン LOB として格納されます。余分な LOB オーバーヘッドもあります。

varray store as LOB を指定し、定義した列の最大サイズが 4,000 バイト未満である場合、VARRAY はインライン LOB として格納されます。概要は次のとおりです。

列の最大サイズを計算します。余分なオーバーヘッドがあるため、サイズが 1,000 バイトの要素が 10 個ある場合、最大サイズは 10 × 1000 より少し大きくなることに注意してください。

- 最大サイズが 4,000 バイトから数バイト引いた値未満で、varray store as LOB を指定していない場合、VARRAY は行にインライン格納されます。
- 最大サイズが 4,000 バイトから数バイト引いた値未満で、varray store as LOB を指定し、DISABLE STORAGE IN ROW を指定していない場合、VARRAY はインライン LOB として格納されます。
- 最大サイズが 4,000 バイトから数バイト引いた値未満で、varray store as LOB および DISABLE STORAGE IN ROW を指定している場合、VARRAY は LOB にアウトライン格納されます。
- 最大サイズが 4,000 バイトを超える場合、varray store as LOB 句を指定しない場合でも、VARRAY は常に LOB に格納されます。VARRAY のサイズおよび DISABLE STORAGE IN ROW を指定しているかどうかによって、インライン LOB かアウトライン LOB のいずれかになります。
- LOB 記憶域句を指定していないか、または DISABLE STORAGE IN ROW を指定しない記憶域句がある場合、システムは自動的に小さい LOB をインラインに、約 4,000 バイトを超える LOB のみアウトラインに格納しようとします。DISABLE STORAGE IN ROW を指定すると、LOB は常にアウトラインに格納されます。

LONG から LOB への移行

アプリケーションを停止できない場合に LONG を LOB に移行させる方法は？

質問

現在の表は、3 つのフィールド（順序、冗長性チェック番号および LONG RAW フィールド）を持つレコードで構成されています。LONG RAW フィールドのサイズは約 100KB ですが、最大で 300KB にできます。ファイル全体は 160GB で、サーバーの最大サイズは 200GB です。このデータベースを Oracle7 リリース 7.3.4 から Oracle9*i* に移行してから、アプリケーション・プログラムの LONG RAW フィールドが正常に動作しません。このフィールドは BLOB に変換する必要があります。BLOB への移行中は、アプリケーションを停止できません。どのような方法がありますか？

回答

Oracle9*i* では、ALTER TABLE を使用して、データを LONG から LOB にコピーできます。詳細は、第 8 章「LONG から LOB への移行」を参照してください。ただし、ALTER TABLE コマンドを使用すると、ALTER の実行中は表が使用できなくなります。

また、Oracle8*i* 以上で導入された TO_LOB 演算子を使用して、データを LONG から LOB にコピーする方法もあります。詳細は、『Oracle9*i* データベース移行ガイド』の第 4 章の「LONG から LOB へのコピー」を参照してください。この場合、表が使用できない期間がさらに短くなります。

参照： 第 8 章「LONG から LOB への移行」を参照してください。

異なる LOB 型の間での変換

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 変換は行われないのですか？ 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.....
```

Oracle8*i* および Oracle9*i* 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 未満であるかぎり、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

Oracle8*i* 以上では、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と、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パスを調べてください。

質問 2

I/O デバイス / パス : 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**をお薦めします。

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 文で使用できるファンクションです。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

LOB パフォーマンスの問題については、第 9 章「LOB: 効果的な使用方法」を参照してください。第 7 章「モデリングおよび設計」も参照してください。

全体の読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は、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は、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 をインライン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 をインライン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 のインライン格納はデフォルトで、ほとんどの場合お薦めします。Oracle9*i* では、LOB 値が約 4KB 未満の場合に、LOB がインラインに格納されます。そのため、値をアウトライン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されますが、LOB 値をすばやく検索するためのメタ情報はインラインに格納されます。そのため、多くの場合、インライン格納によってパフォーマンスが最大になります。

ただし、全表スキャン、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および選択など、多くの実表処理を実行する場合は、LOB をインラインに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？

回答

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。

コールされたルーチンで一時 LOB を作成すると、パフォーマンスに影響するのはなぜですか？

質問

コールされたルーチンでの一時 LOB の作成に関連して、深刻なパフォーマンス上の問題が発生しています。次の 2 つの作業でその問題を説明します。

- createlob=FALSE を指定して RUNLOB() をコールすると（一時 LOB は作成されるが、内部ルーチンの LOB は作成されない）、1 秒未満で実行されます。
- createlob=TRUE を指定して RUNLOB() を実行すると（外部ルーチン LOB と内部ルーチン LOB の両方が作成される）、実行に約 30 秒かかり、ループのサイズに比例して長くなります。ログは次のとおりです。

```
SQL> set serveroutput on size 20000;
SQL> execute runlob(FALSE);
Start time (seconds): 52089
End time (seconds): 52089
PL/SQL procedure successfully completed.

SQL> execute runlob(TRUE);
Start time (seconds): 52102
End time (seconds): 52131
PL/SQL procedure successfully completed.
```

これによって、API での DDL 作成のパフォーマンスが低下しています。何が原因かわかりますか？

```
CREATE OR REPLACE PROCEDURE lob(createlob BOOLEAN)
 IS
 doc     CLOB;
 BEGIN
   IF createlob THEN
     DBMS_LOB.CREATETEMPORARY(doc, TRUE);
     DBMS_LOB.FREETEMPORARY(doc);
   END IF;
 END;
 /
 CREATE OR REPLACE PROCEDURE RUNLOB(createlob BOOLEAN DEFAULT FALSE) AS
 doc    CLOB;
 BEGIN
   dbms_output.put_line('Start time (seconds):
'||to_char(sysdate,'SSSSS'));
   FOR i IN 1..400 LOOP
     DBMS_LOB.CREATETEMPORARY(doc, TRUE);
     lob(createlob);
```

```
    DBMS_LOB.FREETEMPORARY(doc);
  END LOOP;
  dbms_output.put_line('End time (seconds):
'||to_char(sysdate,'SSSSS'));
END;
/
```

回答

このテスト・ケースにおける、一時 LOB を RUNLOB() で作成する場合と LOB() で作成する場合の違いは次のとおりです。

- RUNLOB() では、ファンクション・フレームに有効期限があり、すべての新しい一時 LOB は、ファンクションの最初の一時 LOB を使用して作成された単一の duration state object（DSO）からリンクされています。
- ただし、LOB() では、一時 LOB を作成するたびに、ファンクション・フレームの有効期限が新規であるため、Oracle は新しい DSO を割り当てます。

kdlt_add_dso_link() は、kdlt における他の一時 LOB 作成サイクルと比較して、コストがかかる操作です。LOB() に対する DSO の割当て（割当て解除）には、オーバーヘッドが発生します。kdlt_add_dso_link() では、対応付けられたメモリー割当ておよび制御構造の初期化のために、新しい DSO を割り当てる必要があります。余分のコード・パスにはコストがかかります。

新しい DSO 作成を回避するために、LOB() において、ローカルではなくパッケージ変数 tmplob ロケータを使用できますか？ 次の変更済スクリプトを使用してください。このスクリプトでは、パフォーマンスに影響はありません。

```
create or replace package pk is
   tmplob clob;
 end pk;
/

 CREATE OR REPLACE PROCEDURE lob(createlob BOOLEAN)
 IS
 doc     CLOB;
 BEGIN
   IF createlob THEN
     DBMS_LOB.CREATETEMPORARY(pk.tmplob, TRUE);
     DBMS_LOB.FREETEMPORARY(pk.tmplob);
     null;
  END IF;
 END;
 /

 CREATE OR REPLACE PROCEDURE RUNLOB(createlob BOOLEAN DEFAULT FALSE) AS
 doc    CLOB;
```

```
BEGIN
  dbms_output.put_line('Start time (seconds):
'||to_char(sysdate,'SSSSS'));
  FOR i IN 1..400 LOOP
    DBMS_LOB.CREATETEMPORARY(doc, TRUE);
    lob(createlob);
    DBMS_LOB.FREETEMPORARY(doc);
  END LOOP;
  dbms_output.put_line('End time (seconds):
'||to_char(sysdate,'SSSSS'));
END;
/
```

応答

ありがとうございます。有効範囲がパッケージの一時 LOB を、現在ローカルで機能する LOB があるほぼすべての場所で使用できます。

PL/SQL

UPLOAD_AS_BLOB

質問

「UPLOAD_AS_BLOB」とは何ですか？

回答

UPLOAD_AS_BLOB とは、データベース・アクセス記述子（DAD）の属性で、PL/SQL Web Gateway インタフェースを使用してドキュメントを BLOB 型の表の列にアップロードする場合に使用されます。

7

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

データ型を選択する場合、次の項目を考慮します。

- 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 : 可変幅文字データおよびマルチバイト固定幅文字データ

LOB 型と LONG 型および LONG RAW 型との比較

表 7-1 に、LOB 型と LONG 型および LONG RAW 型との類似点と相違点を示します。

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 行当たり 1 つの LONG または LONG RAW のみを格納します。
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。 インライン LOB では、表列の中に 4,000 バイト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、ロケータまたはデータのフェッチが選択できます。	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、OS 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 の最大サイズは 2GB 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 を使用する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、柔軟性はありません。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（『Oracle9*i* アドバンスト・レプリケーション』を参照）。

レプリケーション

Oracle は、LONG および LONG RAW データ型を使用する列のレプリケーションをサポートしていません。これらのデータ型を含む列は、レプリケート表から省略されます。Oracle9*i* では、LONG データ型を LOB に変換してからレプリケートする必要があります。

LONG 列の LOB への変換

既存の LONG 列は、次の方法のいずれかを使用して、LOB に変換できます。

- LONG-to-LOB API（第 8 章「LONG から LOB への移行」を参照）
- TO_LOB() ファンクション（10-60 ページの「LONG-to-LOB API を使用した LONG から LOB への移行」を参照）

注意： Oracle9*i* は、LOB から LONG への再変換はサポートしません。

キャラクタ・セットの変換 : 可変幅文字データおよびマルチバイト固定幅文字データ

OCI または OCI 機能にアクセスするプログラム環境では、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、『Oracle9*i* Database グローバリゼーション・サポート・ガイド』を参照してください。

注意： 表に CLOB 列または NCLOB 列がある場合、ALTER DATABASE コマンドは機能しません。

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 LOB 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
- 索引構成表内（IOT）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、インラインに格納されます。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 ロケータがインラインに格納されます。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、インラインに LOB ロケータがあります。これは、LOB が ENABLE STORAGE IN ROW または DISABLE STORAGE IN ROW のどちらで作成された場合にも該当します。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: BLOB 列が ENABLE STORAGE IN ROW として作成された場合、Oracle8*i* 以上では、1 バイトのデータの格納に、インラインの 1 バイトの記憶域のみを余分に使用します。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。

表領域および記憶特性の定義例 1

```
CREATE TABLE ContainsLOB_tab (n NUMBER, c CLOB)
      lob (c) STORE AS SEGNAME (TABLESPACE lobtbs1 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： いくつかの記憶領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 文を使用して 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

7-5 ページの「表領域および記憶特性の定義例 1」に示すとおり、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（『Oracle9*i* データベース・リファレンス』を参照）を問い合せる場合、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 LOB 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、『Oracle9*i* SQL リファレンス』の「*storage_clause*」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 記憶域句内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

Oracle9*i* 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す。

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 値の読込み一貫性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なおして再利用します。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、次のことを考慮します。

- LOB を更新する頻度
- 更新された LOB を読み込む頻度

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	更新された 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利用しないため、LOB 記憶域は大きくなります。

例 2:

LOB を作成し、書込みを 1 回のみ行い、その後は主に読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、旧バージョンの LOB データの使用に必要な領域が少なくなります。既存の LOB が読込み専用とわかっている場合は、旧バージョンのデータ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	読込み	書込み
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ないか、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、読込み中のみバッファ・キャッシュに入れられ、書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 以上で 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ます（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 はバルク・ロードまたは挿入に効果的　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、通常の LOB 操作に対する LOB データ・セグメントの LOB 記憶特性を、CACHE や NOCACHE LOGGING に変更できます。

注意： CACHE を設定すると、LOGGING も行われます。

CHUNK

CHUNK には、データベース・ブロック・サイズの倍数で、LOB データの合計バイト数（LOB 値への 1 回のアクセスで OCILobRead()、OCILobWrite()、DBMS_LOB.READ() または DBMS_LOB.WRITE() を使用して読込みまたは書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 Oracle ブロックで、プラットフォームによって変わることはありません。

1 回に 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。

つまり、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
- CHUNK <= INITIAL

ENABLE | DISABLE STORAGE IN ROW

ENABLE | DISABLE STORAGE IN ROW 句を使用して、LOB をインラインまたはアウトライン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。

ENABLE または DISABLE STORAGE IN ROW のガイドライン

インラインに格納される LOB データの最大サイズは、最大 VARCHAR2 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 をインライン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 値は自動的にアウトラインに移動されます。

このため、次のようなガイドラインが必要になります。

次の理由から、通常、デフォルトの ENABLE STORAGE IN ROW が最適です。

- **小さい LOB**: LOB が小さい場合（4,000 バイト未満の場合）、行を読み込むときに、余分なディスク I/O なしで LOB 全体を読み込めます。
- **大きい LOB**: LOB が大きい場合（4,000 バイトを超える場合）、LOB データがアウトラインに移動されても、ENABLE STORAGE IN ROW が設定されている場合は、制御情報はインラインに格納されています。この制御情報によって、アウトライン LOB データを高速に読み込むことができます。

ただし、DISABLE STORAGE IN ROW の方が適している場合もあります。LOB をインラインに格納すると、行のサイズが大きくなるためです。行サイズが大きいと、ユーザーが全表スキャン、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 などの実表の処理を頻繁に行う場合、パフォーマンスが低下します。

GB の LOB の作成方法

Oracle8*i* 以上では、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 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・ファイルを追加する必要があります。
- **PCTINCREASE 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**: MAXEXTENTS パラメータは、LOB 列用に使用可能なエクステントの数を制限します。LOB のサイズが大きくなると、多数のエクステントが段階的に作成されます。このため、パラメータをその列に対するすべての LOB を保持するために十分大きい値に設定する必要があります。また、UNLIMITED に設定することもできます。
- **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、Oracle 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 UNDO 情報を生成します。エクステントの数が多い場合、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利用します。

例 1: 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 10 章「内部永続 LOB」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);
ALTER TABLESPACE lobtbs1 add datafile '/your/own/data/directory/lobtbs_2.dat'  size
2000M reuse;

CREATE TABLE Multimedia_tab (
   Clip_ID          NUMBER NOT NULL,
   Story            CLOB default EMPTY_CLOB(),
   FLSub            NCLOB default EMPTY_CLOB(),
   Photo            BFILE default NULL,
   Frame            BLOB default EMPTY_BLOB(),
   Sound            BLOB default EMPTY_BLOB(),
   Voiced_ref       REF Voiced_typ,
   InSeg_ntab       InSeg_tab,
   Music            BFILE default NULL,
   Map_obj          Map_typ,
   Comments         LONG
 )
 NESTED TABLE    InSeg_ntab STORE AS InSeg_nestedtab
 LOB(Frame) store as (tablespace lobtbs1 chunk 32768 pctversion 0 NOCACHE
NOLOGGING
 storage(initial 100M next 100M maxextents unlimited pctincrease 0));
```

例 2: GB の LOB を格納するための表領域および表の作成

例 1 とこの例の違いは、例 1 では CREATE TABLE 実行中に記憶域句を指定するのに対して、この例では CREATE TABLESPACE 中に記憶域句を指定することです。

- **一時 LOB** の場合、STORAGE 句は、一時表領域を作成するときに指定する必要があります。
- **永続 LOB** の場合、STORAGE 句は、表領域または表を作成するときのいずれでも指定できます。

一時 LOB COPY または APPEND への影響

PCTINCREASE パラメータを 0（ゼロ）に設定することが重要です。0（ゼロ）に設定しない場合は、デフォルト値は 50% です。4GB の LOB が一杯になっている場合、エクステント・サイズは次のとおり、表領域が一杯になるまで徐々に拡大されます。

```
1st extent: 100M, 2nd 100M, 3rd, 150M, 4th 225M...
```

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 LOB 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込み一貫性のあるロケータの使用方法の詳細は、第 5 章「ラージ・オブジェクト（LOB）: 詳細事項」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ため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。拡張可能な一時表領域を作成するには、次の文を使用します。

```
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、付録 B「マルチメディア・スキーマ」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 varchar2(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
`INSERT INTO Multimedia_tab (story, sound) VALUES (charbuf, rawbuf);`	`INSERT INTO Multimedia_tab(sound) VALUES(charbuf);` 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。
-	`INSERT INTO Multimedia_tab(story) VALUES (rawbuf);` 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。
-	`INSERT INTO Multimedia_tab(sound) VALUES( utl_raw.cast_to_raw(charbuf));` この文では、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せることができないため、処理されません。

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     VARCHAR2(32767);
     smalltext   VARCHAR2(2000);
     bigraw      RAW (32767);
BEGIN
     bigtext     := LPAD('a', 32767, 'a');
     smalltext   := LPAD('a', 2000, 'a');
     bigraw      := utlraw.cast_to_raw (bigtext);

/* The following is allowed: */
     INSERT INTO Multimedia_tab(clip_id, story, frame, comments)
         VALUES (1,bigtext, bigraw,smalltext);
/* The following is allowed: */
     INSERT INTO Multimedia_tab (clip_id, story, comments)
         VALUES (2,smalltext, bigtext);

     bigtext    := LPAD('b', 32767, 'b');
     smalltext  := LPAD('b', 20, 'a');
     bigraw     := utlraw.cast_to_raw (bigtext);

/* The following is allowed: */
     UPDATE Multimedia_tab SET story = bigtext, frame = bigraw,
         comments = smalltext;

/* The following is allowed */
     UPDATE Multimedia_tab set story = smalltext, comments = bigtext;

/* The following is NOT allowed because we are trying to insert more than
```

```
        4000 bytes of data in a LONG and a LOB column: */
      INSERT INTO Multimedia_tab (clip_id, story, comments)
        VALUES (5, bigtext, bigtext);

/* The following is NOT allowed because we are trying to insert
       data into LOB attribute */
     INSERT into Multimedia_tab (clip_id,map_obj)
       VALUES (10,map_typ(NULL, NULL, NULL, NULL, NULL,bigtext, NULL));

/* The following is not allowed because we try to perform INSERT AS
     SELECT data INTO LOB */
    INSERT INTO Multimedia_tab (story) AS SELECT bigtext FROM foo;
 END;
```

例 : PL/SQL - 4,000 バイトを超えるバインド（挿入は未サポート）

4,000 バイトを超えるバインドの場合、HEX と RAW の間の変換がサポートされていないため、挿入はできません。

```
/*  Oracle does not do any implicit conversion (e.g., HEX to RAW or RAW to HEX
    etc.) for data of more than 4000 bytes. Hence, the following cases will not
    work :  */

  declare
    charbuf   varchar2(32767);
    rawbuf    raw(32767);
  begin
    charbuf := lpad ('a', 12000, 'a');
    rawbuf  := utl_raw.cast_to_raw(charbuf);

/* The following is allowed ... */
   INSERT INTO Multimedia_tab (story, sound) VALUES (charbuf, rawbuf);

/* The following is not allowed because Oracle won't do implicit
   hex to raw conversion. */
   INSERT INTO Multimedia_tab (sound) VALUES (charbuf);

/* The following is not allowed because Oracle won't do implicit
   raw to hex conversion. */
   INSERT INTO Multimedia_tab (story) VALUES (rawbuf);

/* The following is not allowed because we can't combine the
   utl_raw.cast_to_raw() operator with the bind of more than 4,000 bytes. */
   INSERT INTO Multimedia_tab (sound) VALUES (utl_raw.cast_to_raw(charbuf));

end;
/
```

例 : PL/SQL - 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,
/* The following command inserts only 4,000 bytes because the result of
   LPAD is limited to 4,000 bytes */
   INSERT INTO Multimedia_tab (story) VALUES (lpad('a', 5000, 'a'));

/* The following command inserts only 2,000 bytes because the result of
   LPAD is limited to 4,000 bytes, and the implicit hex to raw conversion
   converts it to 2,000 bytes of RAW data. */
   INSERT INTO Multimedia_tab (sound) VALUES (lpad('a', 5000, 'a'));
```

例 : C（OCI）- 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 );
void insert()         /* A function in an OCI program */
{
/* The following is allowed */
   ub1 buffer[8000];
   text *insert_sql = "INSERT INTO Print_media(ad_sourcetext, ad_composite,
comments)
               VALUES (:1, :2, :3)";
   OCIStmtPrepare(stmthp, errhp, insert_sql, strlen((char*)insert_sql),
              (ub4) OCI_NTV_SYNTAX, (ub4) OCI_DEFAULT);
   OCIBindByPos(stmthp, &bindhp[0], errhp, 1, (dvoid *)buffer, 8000,
              SQLT_LNG, 0, 0, 0, 0, 0, (ub4) OCI_DEFAULT);
   OCIBindByPos(stmthp, &bindhp[1], errhp, 2, (dvoid *)buffer, 8000,
              SQLT_LBI, 0, 0, 0, 0, 0, (ub4) OCI_DEFAULT);
   OCIBindByPos(stmthp, &bindhp[2], errhp, 3, (dvoid *)buffer, 2000,
              SQLT_LNG, 0, 0, 0, 0, 0, (ub4) OCI_DEFAULT);
   OCIStmtExecute(svchp, stmthp, errhp, 1, 0, OCI_DEFAULT);
}

void insert()
{
/* The following is allowed */
   ub1 buffer[8000];
   text *insert_sql = "INSERT INTO Print_media (ad_sourcetext,comments)
               VALUES (:1, :2)";
   OCIStmtPrepare(stmthp, errhp, insert_sql, strlen((char*)insert_sql),
              (ub4) OCI_NTV_SYNTAX, (ub4) OCI_DEFAULT);
   OCIBindByPos(stmthp, &bindhp[0], errhp, 1, (dvoid *)buffer, 2000,
              SQLT_LNG, 0, 0, 0, 0, 0, (ub4) OCI_DEFAULT);
```

```
   OCIBindByPos(stmthp, &bindhp[1], errhp, 2, (dvoid *)buffer, 8000,
              SQLT_LNG, 0, 0, 0, 0, 0, (ub4) OCI_DEFAULT);
   OCIStmtExecute(svchp, stmthp, errhp, 1, 0, OCI_DEFAULT);
}

void insert()
{
/* The following is allowed, no matter how many rows it updates */
   ub1 buffer[8000];
   text *insert_sql = (text *)"UPDATE Print_media SET
              ad_sourcetext = :1, ad_photo=:2, comments=:3";
   OCIStmtPrepare(stmthp, errhp, insert_sql, strlen((char*)insert_sql),
              (ub4) OCI_NTV_SYNTAX, (ub4) OCI_DEFAULT);
   OCIBindByPos(stmthp, &bindhp[0], errhp, 1, (dvoid *)buffer, 8000,
              SQLT_LNG, 0, 0, 0, 0, 0, (ub4) OCI_DEFAULT);
   OCIBindByPos(stmthp, &bindhp[1], errhp, 2, (dvoid *)buffer, 8000,
              SQLT_LBI, 0, 0, 0, 0, 0, (ub4) OCI_DEFAULT);
   OCIBindByPos(stmthp, &bindhp[2], errhp, 3, (dvoid *)buffer, 2000,
              SQLT_LNG, 0, 0, 0, 0, 0, (ub4) OCI_DEFAULT);
   OCIStmtExecute(svchp, stmthp, errhp, 1, 0, OCI_DEFAULT);
}

void insert()
{
/* The following is allowed, no matter how many rows it updates */
   ub1 buffer[8000];
   text *insert_sql = (text *)"UPDATE Print_media SET
              ad_sourcetext = :1, ad_photo=:2, comments=:3";
   OCIStmtPrepare(stmthp, errhp, insert_sql, strlen((char*)insert_sql),
             (ub4) OCI_NTV_SYNTAX, (ub4) OCI_DEFAULT);
   OCIBindByPos(stmthp, &bindhp[0], errhp, 1, (dvoid *)buffer, 2000,
             SQLT_LNG, 0, 0, 0, 0, 0, (ub4) OCI_DEFAULT);
   OCIBindByPos(stmthp, &bindhp[1], errhp, 2, (dvoid *)buffer, 2000,
             SQLT_LNG, 0, 0, 0, 0, 0, (ub4) OCI_DEFAULT);
   OCIBindByPos(stmthp, &bindhp[2], errhp, 3, (dvoid *)buffer, 8000,
             SQLT_LNG, 0, 0, 0, 0, 0, (ub4) OCI_DEFAULT);
   OCIStmtExecute(svchp, stmthp, errhp, 1, 0, OCI_DEFAULT);
}

void insert()
{
/* Piecewise, callback and array insert/update operations similar to
   the allowed regular insert/update operations are also allowed */
}

void insert()
{
/* The following is NOT allowed because we try to insert >4000 bytes
```

```
   to both LOB and LONG columns */
ub1 buffer[8000];
text *insert_sql = (text *)"INSERT INTO Print_media (ad_composite, comments)
            VALUES (:1, :2)";
OCIStmtPrepare(stmthp, errhp, insert_sql, strlen((char*)insert_sql),
           (ub4) OCI_NTV_SYNTAX, (ub4) OCI_DEFAULT);
OCIBindByPos(stmthp, &bindhp[0], errhp, 1, (dvoid *)buffer, 8000,
            SQLT_LNG, 0, 0, 0, 0, 0, (ub4) OCI_DEFAULT);
OCIBindByPos(stmthp, &bindhp[1], errhp, 2, (dvoid *)buffer, 8000,
            SQLT_LNG, 0, 0, 0, 0, 0, (ub4) OCI_DEFAULT);
OCIStmtExecute(svchp, stmthp, errhp, 1, 0, OCI_DEFAULT);
}

void insert()
{
/* The following is NOT allowed because we try to insert data into
   LOB attributes */
   ub1 buffer[8000];
   text *insert_sql = (text *)"INSERT INTO Print_media (adheader_typ)
            VALUES (adheader_typ(NULL, NULL, NULL, NULL, NULL,:1, NULL))";
   OCIStmtPrepare(stmthp, errhp, insert_sql, strlen((char*)insert_sql),
           (ub4) OCI_NTV_SYNTAX, (ub4) OCI_DEFAULT);
   OCIBindByPos(stmthp, &bindhp[0], errhp, 1, (dvoid *)buffer, 2000,
            SQLT_LNG, 0, 0, 0, 0, 0, (ub4) OCI_DEFAULT);
   OCIStmtExecute(svchp, stmthp, errhp, 1, 0, OCI_DEFAULT);
}

void insert()
{
/* The following is NOT allowed because we try to do insert as
   select character data into LOB column */
   ub1 buffer[8000];
   text *insert_sql = (text *)"INSERT INTO Print_media (ad_sourcetext)
            SELECT :1 from FOO";
   OCIStmtPrepare(stmthp, errhp, insert_sql,strlen((char*)insert_sql),
            (ub4) OCI_NTV_SYNTAX, (ub4) OCI_DEFAULT);
   OCIBindByPos(stmthp, &bindhp[0], errhp, 1, (dvoid *)buffer, 8000,
            SQLT_LNG, 0, 0, 0, 0, 0, (ub4) OCI_DEFAULT);
   OCIStmtExecute(svchp, stmthp, errhp, 1, 0, OCI_DEFAULT);
}

void insert()
{
/* Other update operations similar to the disallowed insert operations are also
   not allowed. Piecewise and callback insert/update operations similar to the
   disallowed regular insert/update operations are also not allowed  */
}
```

内部 LOB 用の OPEN、CLOSE および ISOPEN インタフェース

OPEN、CLOSE および ISOPEN 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。

注意： オープン性は、ロケータではなく、LOB に対応付けられています。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 は暗黙的にオープン、クローズされ、ドメイン索引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索引は、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 ドメイン索引は常に有効で、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、ドメイン索引上のトリガーは 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索引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索引が OPEN/CLOSE コールの間では有効にならないことを示します。

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、ドメイン索引およびファンクション索引は更新されません。この場合、LOB 列にファンクション索引およびドメイン索引を再作成してください。

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 に対して次のことを意味します。

- LOB がクローズしない
- ドメイン索引上のトリガーが起動しない

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は、次のいずれかに該当します。

- 「トランザクションを開始する DML 文（SELECT ... FOR UPDATE を含む）」とコミットとの間
- 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、ドメイン索引上のトリガーは起動されません。

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: トランザクションでの LOB に対する 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ける 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
   /* Select a LOB: */
   SELECT Story INTO Lob_loc1 FROM Multimedia_tab WHERE Clip_ID = 1;

   /* The following statement opens the LOB outside of a transaction
      so it must be closed before the session ends: */
   DBMS_LOB.OPEN(Lob_loc1, DBMS_LOB.LOB_READONLY);
   /* The following statement begins a transaction. Note that Lob_loc1 and
      Lob_loc2 point to the same LOB:  */
```

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

例 2: トランザクションでの LOB に対する 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
   /* Note that the FOR UPDATE clause starts a transaction: */
   SELECT Story INTO Lob_loc FROM Multimedia_tab WHERE Clip_ID = 1 for update;
   DBMS_LOB.OPEN(Lob_loc, DBMS_LOB.LOB_READONLY);
   /* COMMIT returns an error because there is still an open LOB associated
   with this transaction: */
   COMMIT;
END;
```

索引構成表内（IOT）の LOB

現在、索引構成表（IOT）は内部 LOB 列および外部 LOB 列をサポートしています。IOT 内の LOB に対する SQL の DDL、DML およびピース単位操作は、従来の表での操作と同様に動作します。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かぎり、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。
- **インライン記憶域とアウトライン記憶域**：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、デフォルトでアウトラインに格納されます。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 に対して強制的に 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 は従来の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）。

  > **参照：** 第 5 章「ラージ・オブジェクト（LOB）：詳細事項」の「パーティション化された索引構成表内の LOB」を参照してください。

LOB 列を含む索引構成表の例

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 ioto_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インラインに格納される CLOB（列 C3）
- 明示的にアウトラインに強制格納される 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アウトラインに格納されます。

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の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メリットを 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、メンテナンスしやすくなります。

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。

付録 B「マルチメディア・スキーマ」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

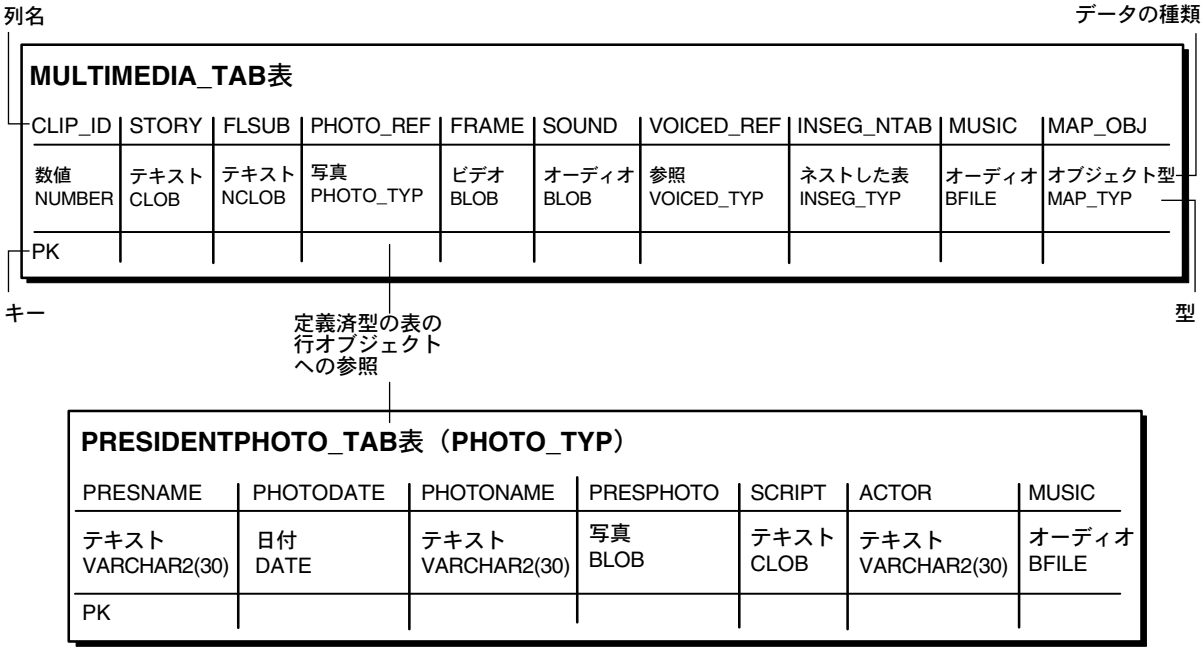


表 7-5 Multimedia_tab 列

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。写真が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説明や、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 期務めた大統領には 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： 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 TABLESPACE, DROP TABLESPACE TO scott;
CONNECT scott/tiger
CREATE TABLESPACE EarlyPresidents_tbs DATAFILE
'disk1:moredata01' SIZE 1M;
CREATE TABLESPACE EarlyPresidentsPhotos_tbs DATAFILE
'disk1:moredata99' SIZE 1M;
CREATE TABLESPACE EarlyPresidentsScripts_tbs DATAFILE
'disk1:moredata03' SIZE 1M;
CREATE TABLESPACE RichardNixon1_tbs DATAFILE
'disk1:moredata04' SIZE 1M;
CREATE TABLESPACE Post1960PresidentsPhotos_tbs DATAFILE
'disk1:moredata05' SIZE 1M;
CREATE TABLESPACE Post1960PresidentsScripts_tbs DATAFILE
'disk1:moredata06' SIZE 1M;
CREATE TABLESPACE RichardNixon2_tbs DATAFILE
'disk1:moredata07' SIZE 1M;
CREATE TABLESPACE GeraldFord1_tbs DATAFILE
'disk1:moredata97' SIZE 1M;
CREATE TABLESPACE RichardNixonPhotos_tbs DATAFILE
'disk1:moredata08' SIZE 2M;
CREATE TABLESPACE RichardNixonBigger2_tbs DATAFILE
'disk1:moredata48' SIZE 2M;
CREATE TABLE Mirrorlob_tab(
   PresName VARCHAR2(30),
   PhotoDate DATE,
   PhotoName VARCHAR2(30),
   PresPhoto BLOB,
   Script CLOB,
   Actor VARCHAR2(30),
   Music BFILE);
```

```
CREATE TABLE Presidentphoto_tab(PresName VARCHAR2(30), PhotoDate DATE,
                                PhotoName VARCHAR2(30), PresPhoto BLOB,
                                Script CLOB, Actor VARCHAR2(30), Music BFILE)
   STORAGE (INITIAL 100K  NEXT 100K PCTINCREASE 0)
   LOB (PresPhoto) STORE AS (CHUNK 4096)
   LOB (Script) STORE AS (CHUNK 2048)
   PARTITION BY RANGE(PhotoDate)
(PARTITION GeorgeWashington1_part
   /* Use photos to the end of Washington's first term */
   VALUES LESS THAN (TO_DATE('19-mar-1792', 'DD-MON-YYYY'))
```

```
   TABLESPACE EarlyPresidents_tbs
   LOB (PresPhoto) store as (TABLESPACE EarlyPresidentsPhotos_tbs)
   LOB (Script) store as (TABLESPACE EarlyPresidentsScripts_tbs),
PARTITION GeorgeWashington2_part
   /* Use photos to the end of Washington's second term */
   VALUES LESS THAN (TO_DATE('19-mar-1796', 'DD-MON-YYYY'))
   TABLESPACE EarlyPresidents_tbs
   LOB (PresPhoto) store as (TABLESPACE EarlyPresidentsPhotos_tbs)
   LOB (Script) store as (TABLESPACE EarlyPresidentsScripts_tbs),
PARTITION JohnAdams1_part
   /* Use photos to the end of Adams' only term */
   VALUES LESS THAN (TO_DATE('19-mar-1800', 'DD-MON-YYYY'))
   TABLESPACE EarlyPresidents_tbs
   LOB (PresPhoto) store as (TABLESPACE EarlyPresidentsPhotos_tbs)
   LOB (Script) store as (TABLESPACE EarlyPresidentsScripts_tbs),
/* ...intervening presidents... */
PARTITION RichardNixon1_part
   /* Use photos to the end of Nixon's first term */
   VALUES LESS THAN (TO_DATE('20-jan-1972', 'DD-MON-YYYY'))
   TABLESPACE RichardNixon1_tbs
   LOB (PresPhoto) store as (TABLESPACE Post1960PresidentsPhotos_tbs)
   LOB (Script) store as (TABLESPACE Post1960PresidentsScripts_tbs)
);
```

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、一意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

Oracle8 リリース 8.0 から Oracle8*i* リリース 8.1 以上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

ドキュメンタリのプロデューサは、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 ツリー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、ドメイン専用に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。Oracle8*i* 以上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、ドメイン索引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、『Oracle9*i* Data Cartridge Developer's Guide』を参照してください。

LOB 列の内容の性質によっては、Oracle *inter*Media 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 *inter*Media オプションの詳細は、『Oracle *inter*Media ユーザーズ・ガイドおよびリファレンス』および『Oracle Text リファレンス』を参照してください。

LOB 列のファンクション索引

Oracle9*i* では、LOB 列のファンクション索引をサポートしています。LOB 列の拡張索引およびドメイン索引と同様に、DML 操作が LOB 列で実行されると、ファンクション索引も自動的に更新されます。

注意： ファンクション索引が LOB 列に存在する場合、拡張索引を更新すると、ファンクション索引も更新されます。

参照：『Oracle9*i* 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。

LOB に対する SQL セマンティクスのサポート

この項の内容は次のとおりです。

- SQL VARCHAR2/RAW セマンティクスの CLOB/BLOB への適用方法
- SQL RAW 型および BLOB
- LOB に対する SQL DML の変更
- VARCHAR2/RAW および CLOB/BLOB の SQL ファンクション / 演算子
- PL/SQL 文および PL/SQL 変数 : 新しく変更されたセマンティクス
- PL/SQL CLOB の比較規則

以前のリリースでは、データベースに格納された LOB には、様々な言語インタフェース（C、C++、OO4O、Java、COBOL、PL/SQL）の一連の API で LOB ロケータを使用してしかアクセスできませんでした。LOB は、SQL 文字ファンクションでは使用できませんでした。

LOB の使いやすさの向上 : SQL 文字ファンクションを使用した LOB へのアクセス

Oracle9*i* では、SQL 文字列演算子や SQL ファンクションなどの SQL VARCHAR2 セマンティクスを使用して LOB にアクセスできるようになりました。

ユーザーが使い慣れた SQL インタフェースが提供されるため、LOB データへ簡単にアクセスできます。この追加機能は、次の 2 つの場合に有効です。

- 小さいサイズの LOB（10 ～ 100KB）を使用して、API を介してデータを格納し、LOB に対する SQL サポートを必要とする場合。
- LONG 列を LOB に移行した場合。今回のリリースでは、LONG-to-LOB 移行 API によってよって簡単になった移行プロセスのメリットを利用できます（第 8 章「LONG から LOB への移行」を参照）。

ランダム・アクセスやピース単位のフェッチなどの機能を利用する必要がある上級 LOB ユーザーは、既存の LOB API インタフェースを使用してください。

標準サイズから大きいサイズ（1MB を超える）の LOB のユーザーに対しては、パフォーマンスが低下する可能性があるため、この SQL インタフェースはお薦めしません。

前述の説明は、内部永続 LOB のみに該当します。今回のリリースでは、BFILE に対する SQL はサポートしていません。

注意： SQL セマンティクスのサポートは、現在の LOB の使用に影響しません。LOB API を使用する既存の LOB アプリケーションを変更する必要はありません。

CLOB に対して使用可能になった SQL および PL/SQL VARCHAR2 ファンクションおよび演算子

次の SQL VARCHAR2 ファンクションおよび演算子が、CLOB に対して使用可能になりました（表 7-6 を参照）。

- INSTR 関係演算子およびファンクション
  - INSTR() およびそのオプション・タイプ（表 7-7 を参照）
  - LIKE
  - REPLACE()
- CONCAT および ||
- LENGTH() およびそのオプション・タイプ（表 7-7 を参照）
- SUBSTR() およびそのオプション・タイプ（表 7-7 を参照）
- TRIM()、LTRIM() および RTRIM()
- LOWER()、UPPER()、NLS_LOWER() および NLS_UPPER()
- LPAD() および RPAD()

LOB に対して使用可能になった PL/SQL 関係演算子

LONG から LOB への移行において、PL/SQL の次の関係演算子が、LONG および LOB に対して使用可能になりました。

- 演算子 : >、<、= および !=
- IN および BETWEEN
- GREATEST および LEAST
- NLSSORT

表 7-6 にも、これらの演算子を示します。

CHAR と CLOB 間の SQL および PL/SQL 変換ファンクション

CHAR と CLOB 間の次の変換ファンクションが、LOB に対して使用可能になりました。

- TO_CHAR() および TO_NCHAR(): CLOB または NCLOB を CHAR または NCHAR に変換します。
- TO_CLOB() および TO_NCLOB(): CHAR または NCHAR を CLOB または NCLOB に変換します。

LOB に対してサポートされていない SQL の機能

次の SQL の機能は、ほとんど使用しないか、またはより簡単な方法があるため、LOB に対してサポートされていません。

- **LOB 列の INDEX**: かわりに、Oracle Text（*inter*Media Text）を使用します。
- **SQL 変換ファンクション**: TO_DATE、TO_NUMBER、TO_TIMESTAMP、CHARTOROWID、TO_MULTI_BYTE、TO_SINGLE_BYTE（ほとんど使用しません。）
- **他の SQL ファンクション**: GROUPING（ほとんど使用しません。）
- **次の比較ファンクションおよび演算子**:

  注意： これらの演算子は、PL/SQL で使用できますが、PL/SQL ブロックで発行された SQL 問合せでは使用できません。

  - 演算子 : >、<、= および !=
  - IN、SOME、ANY、ALL および BETWEEN
  - MAX、MIN、GREATEST および LEAST
  - SELECT DISTINCT、GROUP BY、ORDER BY（SORT）、UNION、INTERSECT および MINUS
  - JOIN
  - 他の SQL ファンクション : INITCAP、NLS_INITCAP、DUMP、TRANSLATE、VSIZE および DECODE

CLOB での VARCHAR2 に対する SQL ファンクション / 演算子の使用

表 7-6 に、VARCHAR2 をオペランドおよび引数として取得したり、VARCHAR2 の値を戻す、すべての SQL 演算子および SQL ファンクションを示します。IS [NOT] NULL 演算子以外の演算子およびファンクションは、以前のリリースでは CLOB に対して機能しません。

表 7-6 の 4 列目の SQL 列は、SQL 演算子およびファンクションが、Oracle9*i* の CLOB に対してサポートされているかどうかを示しています。

表 7-6 に示されているほとんどのファンクションは、PL/SQL の組込みファンクション（提供されるパッケージ）でも使用できます。5 列目の PL/SQL 列は、CLOB に対する演算子およびファンクションが PL/SQL で使用可能であるかどうかを示しています。

Oracle9*i* では、CLOB 型と CHAR 型間の暗黙的な変換が可能です。そのため、CLOB に対して使用できないファンクションでも、暗黙的な変換を介して CLOB を処理できます。この場合、ファンクションをコールする前に CLOB を CHAR または VARCHAR2 に変換します。CLOB のサイズが 4KB を超える場合、4KB のみが CHAR または VARCHAR2 に変換されます。

表 7-6 では、暗黙的な変換を介して CLOB パラメータをとるファンクションは「CNV」と示されています。

表 7-6 SQL VARCHAR2 ファンクションおよび演算子

カテゴリ	演算子および ファンクション	CLOB 列に対する SQL 例	SQL	PL/SQL
連結演算子	||、CONCAT()	`Select clobCol || clobCol2 from tab;`	はい	はい
比較演算子	=、!=、>、>=、<、<=、<>、^=	`if clobCol=clobCol2 then...`	いいえ	はい
	IN、NOT IN	`if clobCol NOT IN (clob1, clob2, clob3) then...`	いいえ	はい
	SOME、ANY、ALL	`if clobCol < SOME (select clobCol2 from...) then...`	いいえ	利用不可
	BETWEEN	`if clobCol BETWEEN clobCol2 and clobCol3 then...`	いいえ	はい
	LIKE [ESCAPE] およびそのオプション・タイプ。表 7-7 を参照。	`if clobCol LIKE '%pattern%' then...`	はい	はい
	IS [NOT] NULL	`where clobCol IS NOT NULL`	はい	はい
文字ファンクション	INITCAP、NLS_INITCAP	`select INITCAP(clobCol) from...`	CNV	CNV
	LOWER、NLS_LOWER、UPPER、NLS_UPPER	`...where LOWER(clobCol1) = LOWER(clobCol2)`	はい	はい
	LPAD、RPAD	`select RPAD(clobCol, 20, ' La') from...`	はい	はい
	TRIM、LTRIM、RTRIM	`...where RTRIM(LTRIM(clobCol,'ab'), 'xy') = 'cd'`	はい	はい
	REPLACE	`select REPLACE(clobCol, 'orig','new') from...`	はい	はい
	SOUNDEX	`...where SOUNDEX(clobCOl) = SOUNDEX('SMYTHE')`	CNV	CNV
	SUBSTR およびそのオプション・タイプ。表 7-7 を参照。	`...where substr(clobCol, 1,4) = 'THIS'`	はい	はい

表 7-6 SQL VARCHAR2 ファンクションおよび演算子 （続き）

カテゴリ	演算子および ファンクション	CLOB 列に対する SQL 例	SQL	PL/SQL
文字ファンクション （続き）	TRANSLATE	`select TRANSLATE(clobCol, '123abc','NC') from...`	CNV	CNV
	ASCII	`select ASCII(clobCol) from...`	CNV	CNV
	INSTR およびそのオプション・タイプ。表 7-7 を参照。	`...where instr(clobCol, 'book') = 11`	はい	はい
	LENGTH およびそのオプション・タイプ。表 7-7 を参照。	`...where length(clobCol) != 7;`	はい	はい
	NLSSORT	`...where NLSSORT (clobCol,'NLS_SORT = German') > NLSSORT ('S','NLS_SORT = German')`	CNV	CNV
変換ファンクション	CHARTOROWID	`CHARTOROWID(clobCol)`	CNV	CNV
	HEXTORAW	`HEXTORAW(CLOB)`	いいえ	CNV
	CONVERT	`select CONVERT(clobCol,'WE8DEC','WE8HP') from...`	はい	CNV
	TO_DATE	`TO_DATE(clobCol)`	CNV	CNV
	TO_NUMBER	`TO_NUMBER(clobCol)`	CNV	CNV
	TO_TIMESTAMP	`TO_TIMESTAMP(clobCol)`	いいえ	CNV
	TO_MULTI_BYTE TO_SINGLE_BYTE	`TO_MULTI_BYTE(clobCol)` `TO_SINGLE_BYTE(clobCol)`	CNV	CNV
	TO_CHAR	`TO_CHAR(clobCol)`	はい	はい
	TO_NCHAR	`TO_NCHAR(clobCol)`	はい	はい
	TO_LOB	`INSERT INTO... SELECT TO_LOB(longCol)...` **注意 :** TO_LOB は、LONG 列を持つ表を SELECT FROM に指定して、LOB 列を持つ表を作成またはその表に挿入する場合にのみ使用できます。	利用不可	利用不可

表 7-6 SQL VARCHAR2 ファンクションおよび演算子 （続き）

カテゴリ	演算子および ファンクション	CLOB 列に対する SQL 例	SQL	PL/SQL
変換ファンクション (続き)	TO_CLOB	`TO_CLOB(varchar2Col)`	はい	はい
	TO_NCLOB	`TO_NCLOB(varchar2Clob)`	はい	はい
集計ファンクション	COUNT	`select count(clobCol) from...`	いいえ	利用不可
	MAX、MIN	`select MAX(clobCol) from...`	いいえ	利用不可
	GROUPING	`select grouping(clobCol) from... group by cube (clobCol);`	いいえ	利用不可
他のファンクション	GREATEST、LEAST	`select GREATEST (clobCol1, clobCol2) from...`	いいえ	CNV
	DECODE	`select DECODE(clobCol, condition1, value1, defaultValue) from...`	CNV	CNV
	NVL	`select NVL(clobCol,'NULL') from...`	はい	はい
	DUMP	`select DUMP(clobCol) from...`	いいえ	利用不可
	VSIZE	`select VSIZE(clobCol) from...`	いいえ	利用不可

参照：『Oracle9*i* SQL リファレンス』の第 6 章「ファンクション」を参照してください。

VARCHAR2 および CLOB の Unicode サポート

今回のリリースでは、データベースが VARCHAR2（Unicode）に対して Unicode をサポートするため、INSTR、SUBSTR、LENGTH および LIKE ファンクションの Unicode オプション・タイプがいくつか提供されます。これらの Unicode ファンクションは、CLOB または NCLOB ではサポートされていません（表 7-7 を参照）。

参照：

- 『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。
- 『Oracle9*i* 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。
- 『Oracle9*i* SQL リファレンス』を参照してください。
- 『Oracle9*i* Database グローバリゼーション・サポート・ガイド』を参照してください。

表 7-7 Unicode 関連の SQL ファンクション（CLOB=CLOB サポート）

SQL ファンクション	コメント	CLOB
INSTRB、SUBSTRB、LENGTHB	Oracle9*i* 以下のリリースに存在するバイト・ベースのファンクション	いいえ
INSTR2、SUBSTR2、LENGTH2、LIKE2	今回のリリースで提供する UCS2 キャラクタ・セット・ベースのファンクション	いいえ
INSTR4、SUBSTR4、LENGTH4、LIKE4	今回のリリースで提供する UCS4 キャラクタ・セット・ベースのファンクション	いいえ
INSTRC、SUBSTRC、LENGTHC、LIKEC	今回のリリースで提供するキャラクタ・ベースのファンクション	いいえ

LOB が使用できない場合の SQL の機能

表 7-8 に、LOB が使用できない場合の SQL の他の機能を示します。詳細は、第 4 章「LOB の管理」の「LOB の制限」を参照してください。

表 7-8 LOB が使用できない場合の SQL の機能

SQL の機能	CLOB 列の例
SELECT DISTINCT	`SELECT DISTINCT clobCol from...`
SELECT 句 ORDER BY	`SELECT... ORDER BY clobCol`
SELECT 句 GROUP BY	`SELECT avg(num) FROM...` `GROUP BY clobCol`
UNION、INTERSECT、MINUS **注意 :** UNION ALL は、LOB に対して有効です。	`SELECT clobCol1 from tab1 UNION SELECT clobCol2 from tab2;`
JOIN	`SELECT... FROM... WHERE tab1.clobCol = tab2.clobCol`
INDEX	`CREATE INDEX clobIndx ON tab(clobCol)...`

SQL VARCHAR2/RAW セマンティクスの CLOB/BLOB への適用方法

CLOB に対する CHAR バッファの定義

Oracle9*i* では、特別な LOB API を使用することなく、SQL を使用して直接 LOB からデータを取り出せます。

PL/SQL では、CLOB 列に対して VARCHAR2 を、BLOB 列に対して RAW を定義できます。VARCHAR2 列に対して CLOB を、RAW 列に対して BLOB を定義することもできます。

CHAR バッファまたは CLOB への CLOB 列の選択

PL/SQL では、CLOB 列がローカルの VARCHAR2 変数に選択された場合、CLOB 列に格納されたデータを取り出して、CHAR バッファに格納します。バッファがすべての CLOB データを格納するために十分大きくない場合、切捨てエラーが発生し、データはバッファへ書き込まれません。SELECT の後、VARCHAR2 変数は正規の文字バッファと同じ動作をします。

対照的に、CLOB 列がローカルの CLOB 変数に選択された場合、CLOB ロケータがフェッチされます。以前は VARCHAR2 のみを取得していた PL/SQL 組込みファンクションは、引数として CLOB ロケータも取得できるようになりました。1 次引数が CLOB の場合、ファンクションが戻す型は CLOB です。また、CLOB のローカル変数が DBMS_LOB API に渡されると、LOB ロケータとして機能する場合があります。

前述の説明は、RAW および BLOB にも該当します。

VARCHAR2 演算子 / ファンクションでの CLOB の指定

現在、VARCHAR2 列をオペランドまたは引数としてとる SQL 演算子および SQL ファンクションが、CLOB 列を指定できるようになりました。Oracle8*i* では、LOB の比較は、LOB ファンクションと LOB の IS [NOT] NULL 演算子のみ可能でした。今回のリリースでは、PL/SQL での LOB 自体の比較は可能になりましたが、パフォーマンスの問題のため、SQL 問合せの比較はできません。

CLOB 値を戻す SQL ファンクション / 演算子

以前は VARCHAR2 を戻していた SQL 演算子および SQL ファンクションが、入力パラメータのタイプによって、CLOB または VARCHAR2 のいずれかを戻すようになりました。

VARCHAR2 の戻し

VARCHAR2 のみが引数として渡された場合、演算子およびファンクションは、以前と同様に VARCHAR2 を戻します。VARCHAR2 パラメータのみを持つファンクションは、CLOB を戻しません。

CLOB の戻し

通常最初のパラメータである 1 次引数が CLOB として渡された場合、演算子およびファンクションは CLOB を戻します。たとえば、次の SQL 文は、結果を CLOB 型として選択します。

```
SELECT SUBSTR(clobCol, 1,4) FROM .... WHERE LENGTH(clobCol)>4;
SELECT clobCol1 || charCol1 FROM ...;
```

注意： CONCAT() や || のように、正しく定義された 1 次引数がないファンクションは、パラメータに LOB が指定されると LOB を戻します。

一時 LOB ロケータとして戻される LOB

LOB が戻されると、選択リストからの結果は一時 LOB ロケータ形式となります。アプリケーションは、一時 LOB を、SELECT によって戻された CHAR 文字列用のローカル記憶域として参照する必要があります。PL/SQL では、一時 LOB の存続期間は、他のローカル PL/SQL プログラム変数と同じです。一時 LOB は、後続の SQL や PL/SQL VARCHAR2 ファンクションまたは問合せに渡すことができます。

- PL/SQL ローカル変数として、一時 LOB は、常駐するプログラム・ブロックの終了時に適用範囲外となります。その後、LOB データは自動的に解放されます。これは、他の PL/SQL VARCHAR2 変数と同じ動作です。ただし、`DBMS_LOB.FREETEMPORARY()` コールを発行することで、ローカルの一時 LOB が使用しているリソースをいつでも解放できます。
- OCI では、SQL 問合せから戻された一時 LOB は、常にセッションの存続期間中は存続します。ただし、ユーザー定義の存続期間が存在する場合、一時 LOB はユーザー定義の存続期間中に存続します。

警告： 一時表領域が、プログラムの問合せから戻されたすべての一時 LOB 結果を格納するために十分大きいことを確認してください。

また、次のいずれかが発生した場合、宣言されたサイズの正規の CHAR バッファが結果として戻されます。

- CLOB 列を VARCHAR2 に選択した
- CLOB を戻すファンクションが VARCHAR2 バッファに格納された

VARCHAR2 バッファが、LOB からのデータを収めるために十分な大きさでない場合、切捨てエラーが発生します。

SQL 問合せ例 1: CLOB を VARCHAR2 に選択するための SQL の使用

次に、CLOB 列を VARCHAR2 に選択し、結果を宣言したサイズの CHAR バッファとして戻す例を示します。

```
DECLARE
  vc1 VARCHAR2(32000);
  lb1 CLOB;
  lb2 CLOB;
BEGIN
  SELECT clobCol1 INTO vc1 FROM tab WHERE colID=1;
  -- lb1 is a temporary LOB
  SELECT clobCol2 || clobCol3 INTO lb1 FROM tab WHERE colID=2;

  lb2 := vc1|| lb1;
```

```
  -- lb2 is a still temporary LOB, so the persistent data in the database
  -- is not modified. An update is necessary to modify the table data.
  UPDATE tab SET clobCol1 = lb2 WHERE colID = 1;

DBMS_LOB.FREETEMPORARY(lb2); -- Free up the space taken by lb2

       <... some more queries ...>

   END; -- at the end of the block, lb1 is automatically freed
```

VARCHAR2 および CLOB の IS [NOT] NULL

LOB 列に対しては、IS [NOT] NULL 演算子が Oracle8 から使用可能です。この演算子は、LOB ロケータが表の行に格納されているかどうかを確認します。

VARCHAR2 列に対しては、IS NULL 演算子が空の文字列または NULL 文字列を示します。

注意： IS NULL セマンティックの相違

SQL 92 標準では、長さ 0（ゼロ）の文字列は、NULL 文字列とは異なります。

初期化された、長さ 0（ゼロ）の LOB の場合、IS NULL は 0（ゼロ）（FALSE）を戻すことが予想されます。これは、標準に準拠した正常な動作です。対照的に、長さ 0（ゼロ）の VARCHAR2 は、IS NULL に対して TRUE を戻します。

また、LENGTH() ファンクションの場合、次のものが戻されます。

- 長さ 0（ゼロ）の文字列が入力されると、LENGTH() は *NULL* を戻します。
- 長さ 0（ゼロ）の CLOB の場合、SQL および PL/SQL の LENGTH および DBMS_LOB.GETLENGTH() は、0（ゼロ）の EMPTY_CLOB() を戻します。

これらは間違いやすいため、このセマンティクスの相違に注意してください。

SQL RAW 型および BLOB

SQL RAW 型および BLOB は、次のように処理されます。

- **PL/SQL:** BLOB は RAW に選択されます。
- **OCI および他のインタフェース :** C の CHAR 配列など、ローカルのメモリー・バッファを BLOB に定義することが可能です。BLOB 列に対して、C プログラムのすべての型のバッファを SQLT_RAW として定義できます。

LOB に対する SQL DML の変更

Oracle9*i* では、LOB に関して SQL DML への大きな変更はありません。関連する変更が、UPDATE および DELETE の WHERE 句にあるのみです。以前のリリースでは、UPDATE、DELETE および SELECT の WHERE 句で LOB が使用できませんでした。今回のリリースでは、LOB 値の比較を含まない LOB の SQL ファンクションが、WHERE 句の述語で使用可能です（`length()`、`insert()` など）。

VARCHAR2/RAW および CLOB/BLOB の SQL ファンクション / 演算子

表 7-6 ～表 7-8 に示すとおり、VARCHAR2/RAW に対する SQL ファンクションおよび SQL 演算子が拡張され、CLOB 列または BLOB 列で機能するようになりました。

SQL ファンクションの戻り型は、入力する型によって決まります。詳細は、7-42 ページの「CLOB 値を戻す SQL ファンクション / 演算子」を参照してください。

次の例は、CLOB の VARCHAR2 セマンティクスのメリットを得ている問合せを示します。以前のリリースでは、これらの問合せは、PL/SQL コードで DBMS_LOB コールを使用して実行されていました。VARCHAR2 と同じインタフェースを使用すると、より効率的にデータにアクセスできます。

注意 : これらの例は、付録 B「マルチメディア・スキーマ」および第 10 章「内部永続 LOB」の「1 つ以上の LOB 列を含む表の作成」で説明する、マルチメディア・アプリケーション・スキーマの次の改訂バージョンを基にしています。

```
CREATE TABLE Multimedia_tab (
   Clip_ID NUMBER NOT NULL,
   Story           CLOB default EMPTY_CLOB(),
   Gist            VARCHAR2(100),
   ......
}
```

SQL 問合せ例 2: CLOB の SQL 問合せ

```
SELECT Gist||Story FROM Multimedia_tab WHERE Story LIKE Gist;

SELECT SUBSTR(Story, 20, 1), LENGTH(Story) FROM Multimedia_tab WHERE Gist NOT IN
Story;
-- A temp LOB is created and returned for 'Gist||Story' and 'SUBSTR(Story,20,1)'
-- because story is a CLOB.
```

PL/SQL 文および PL/SQL 変数 : 新しく変更されたセマンティクス

PL/SQL では、前述したとおり、多数のセマンティクスが変更されました。

注意 : 特に指定のないかぎり、CLOB および VARCHAR2 に関する次の説明は、BLOB および RAW にも該当します。説明では、BLOB および RAW については特に記載しません。

次の項で説明する、新規の PL/SQL セマンティクスに対するサポートは次のとおりです。

- CLOB と VARCHAR2 の間の暗黙的な変換
  - PL/SQL 例 1: CLOB/VARCHAR2 アプリケーションに対する以前のリリースの SQL インタフェース
  - PL/SQL 例 2: VARCHAR2 として処理された場合の CLOB データへのアクセス
  - PL/SQL 例 3: VARCHAR2 での CLOB 変数の定義
- 明示的な変換ファンクション

CLOB と VARCHAR2 の間の暗黙的な変換

CLOB から VARCHAR2 および VARCHAR2 から CLOB の双方向への暗黙的な変換によって、CLOB と VARCHAR2 間の次の操作が可能になりました。

- VARCHAR2 PL/SQL 変数への CLOB 列の選択
- CLOB 変数への VARCHAR2 列の選択
- CLOB と VARCHAR2 間の割当ておよびパラメータの受渡し

PL/SQL 例 1: CLOB/VARCHAR2 アプリケーションに対する以前のリリースの SQL インタフェース

次の例は、以前のリリースでの CLOB データへのアクセス方法を示します。このアプリケーションは、Multimedia_tab 表から Gist および Story の両方を表示します。

```
declare
   myStoryLOB CLOB;
   myStoryBuf VARCHAR2(4001);
   amt NUMBER:=4001;
   offset NUMBER := 1;
begin
   SELECT Story INTO myStoryLOB FROM Multimedia_tab WHERE Clip_ID = 10;
   DBMS_LOB.READ(myStoryLOB, amt, offset, myStoryBuf);
   -- Display Gist and Story by printing 'myStoryBuf'.
end;
```

PL/SQL 例 2: VARCHAR2 として処理された場合の CLOB データへのアクセス

次の例は、CLOB が VARCHAR2 として処理された場合、今回のリリースでの CLOB データへのアクセス方法を示します。

```
declare
   myStoryBuf VARCHAR2(4001);
begin
   SELECT Story INTO myStoryBuf FROM Multimedia_tab WHERE Clip_ID = 10;
   -- Display Story by printing myStoryBuf directly
end;
```

PL/SQL 例 3: VARCHAR2 での CLOB 変数の定義

```
declare
    myGistLOB CLOB;
begin
    SELECT Gist INTO myGistLOB FROM Multimedia_tab WHERE Clip_ID = 10;
    -- myGistLOB is a temporary LOB.
    -- Use myGistLOB as a lob locator
end;
```

注意 : 以前のリリースでは、PL/SQL で一時 LOB を使用する前に、`DBMS_LOB.CREATETEMPORARY()` コールを発行する必要がありました。今回のリリースから、一時 LOB は、割当ておよび定義によって暗黙的に作成されます。

明示的な変換ファンクション

SQL および PL/SQL では、LONG から LOB への移行において、他のデータ型を CLOB、NCLOB および BLOB に変換するために、次の新規の明示的変換ファンクションが追加されました。

- TO_CLOB(): VARCHAR2、NVARCHAR2 または NCLOB から CLOB への変換
- TO_NCLOB: VARCHAR2、NVARCHAR2 または CLOB から NCLOB への変換
- TO_BLOB(): RAW から BLOB への変換
- TO_CHAR() を使用して、CLOB を CHAR 型へ変換できるようになりました。
- TO_NCHAR(): NCLOB から NCHAR 型への変換

TO_NUMBER() などの他の明示的変換ファンクションは、今回のリリースではサポートされていません（表 7-6 を参照）。変換ファンクションの詳細は、第 8 章「LONG から LOB への移行」を参照してください。

PL/SQL 組込みファンクションでの VARCHAR2 および CLOB

CLOB および VARCHAR2 は、2 つの個別の型です。ただし、使用方法によっては、CLOB を SQL および PL/SQL VARCHAR2 組込みファンクションに渡して、VARCHAR2 と同様に動作させることができます。または、変数を DBMS_LOB API に渡して、LOB ロケータと同様に動作させることができます。次の、「PL/SQL 例 4: PL/SQL の CLOB 変数」の複合的な例を参照してください。

PL/SQL VARCHAR2 ファンクションおよび演算子は、CLOB を引数またはオペランドとして取得する必要があります。

VARCHAR2 変数のサイズが、CLOB を戻すファンクションの結果、または CLOB 列に対する SELECT の結果を含むために十分大きくない場合、エラーが発生し、操作は実行されません。これは、現行の VARCHAR2 動作と一貫性があります。

PL/SQL 例 4: PL/SQL の CLOB 変数

```
1 declare
2   myStory CLOB;
3   revisedStory CLOB;
4   myGist VARCHAR2(100);
5   revisedGist VARCHAR2(100);
6 begin
7  -- select a CLOB column into a CLOB variable
8  SELECT Story INTO myStory FROM Multimedia_tab WHERE clip_id=10;
9  -- perform VARCHAR2 operations on a CLOB variable
10 revisedStory := UPPER(SUBSTR(myStory, 100, 1));
11 -- revisedStory is a temporary LOB
12 -- Concat a VARCHAR2 at the end of a CLOB
13 revisedStory := revisedStory || myGist;

14 -- The following statement will raise an error since myStory is
15 -- longer than 100 bytes
16 myGist := myStory;
17 end;
```

「PL/SQL 例 4: PL/SQL の CLOB 変数」の 10 行目で一時 CLOB が暗黙的に作成され、`revisedStory` CLOB ロケータによって指されていることに注意してください。現行のインタフェースでは、この行を次のように展開できます。

```
buffer VARCHAR2(32000)
DBMS_LOB.CREATETEMPORARY(revisedStory);
buffer := UPPER(DBMS_LOB.SUBSTR(myStory,100,1));
DBMS_LOB.WRITE(revisedStory,length(buffer),1, buffer);
```

13 行目では、myGist が一時 LOB の終わりに追加されます。これは、次の文と同じ影響があります。

```
DBMS_LOB.WRITEAPPEND(revisedStory, myGist, length(myGist));
```

場合によっては、PL/SQL 文で暗黙的に作成された一時 LOB は、以前に定義した LOB ロケータの表現を変更できます。次の例について考えてみます。

PL/SQL 例 5: ロケータとデータのリンケージの変更

```
1 declare
2 myStory CLOB;
3 amt number:=100;
4 buffer VARCHAR2(100):='some data';
5 begin
6 -- select a CLOB column into a CLOB variable
7 SELECT Story INTO myStory FROM Multimedia_tab WHERE clip_id=10;
8 DBMS_LOB.WRITE(myStory, amt, 1, buf);
9 -- write to the persistent LOB in the table
10
11 myStory:= UPPER(SUBSTR(myStory, 100, 1));
12 -- perform VARCHAR2 operations on a CLOB variable, temporary LOB created. Changes
13 -- will not be reflected in the database table from this point on.
14
15 update Multimedia_tab set Story = myStory WHERE clip_id = 10;
16 -- an update is necessary to synchronize the data in the table.
17 end;
```

7 行目以降、myStory は、Multimedia_tab 内の永続 LOB を表します。

8 行目の DBMS_LOB.WRITE() コールは、データを直接表に書き込みます。

UPDATE 文は必要ありません。続いて、11 行目で一時 LOB が作成され、myStory に割り当てられます。これは、myStory がローカルの VARCHAR2 と同様の動作をするようになったためです。LOB ロケータである myStory は、新規に作成された一時 LOB を指します。

したがって、myStory への変更は、これ以降はデータベースに反映されません。変更をデータベース表へ伝播するには、UPDATE 文が必要になります。以前の永続 LOB には、UPDATE が必要でなかったことに注意してください。

SELECT または割当ての結果としてプログラム・ブロックで作成された一時 LOB は、PL/SQL ブロック、ファンクションまたはプロシージャの終了時に自動的に解放されます。CLOB 変数に対して DBMS_LOB.FREETEMPORARY() をコールして、明示的に一時 LOB を解放し、システム・リソースおよび一時表領域を再生することもできます。

PL/SQL 例 6: 一時 LOB の自動的および手動による解放

```
declare
   Story1 CLOB;
   Story2 CLOB;
   StoryCombined CLOB;
   StoryLower CLOB;
begin
   SELECT Story INTO Story1 FROM Multimedia_tab WHERE Clip_ID = 1;
   SELECT Story INTO Story2 FROM Multimedia_tab WHERE Clip_ID = 2;
   StoryCombined := Story1 || Story2; -- StoryCombined is a temporary LOB
   -- Free the StoryCombined manually to free up space taken
   DBMS_LOB.FREETEMPORARY(StoryCombined);
   StoryLower := LOWER(Story1) || LOWER(Story2);
   end; -- At the end of block, StoryLower is freed.
```

PL/SQL CLOB の比較規則

VARCHAR2 と同様に、CLOB を他の CLOB や VARCHAR2 と比較する場合、一連の規則によって比較が制御されます。規則は通常、「照合順番」といいます。Oracle では、CHAR および VARCHAR2 は、CHAR の空白埋めのため、順序がわずかに異なります。

VARCHAR2 の照合順番に従う CLOB

通常、CLOB は VARCHAR2 と同じ照合順番に従います。つまり、CLOB を比較するとき、CLOB データの内容を VARCHAR2 バッファに取り出して、その VARCHAR2 を比較する場合は、結果は一貫しています。この規則は、CLOB 間、CLOB と VARCHAR2 間および CLOB と CHAR 間の比較を含む、すべての場合に適用されます。

注意： CLOB を CHAR 文字列と比較する場合、常に CLOB の文字データが CHAR 文字列と比較されます。同様に、2 つの CLOB を比較する場合、それぞれの LOB ロケータではなく、2 つの CLOB のデータ内容が比較されます。

CLOB を、文字以外のデータまたは BLOB と比較する必要はありません。比較すると、エラーが発生します。

OCI および Java インタフェースの SQL および PL/SQL との相互作用

OCI インタフェースおよび Java インタフェースによって、LOB を指定した SQL 文および PL/SQL 文に対して VARCHAR2 変数をバインドおよび定義できます。

参照：

- 『Oracle Call Interface プログラマーズ・ガイド』を参照してください。
- 『Oracle9*i* 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。

今回のリリースでは、逆方向への変数の定義はできません。つまり、クライアント側の LOB ロケータを VARCHAR2 列に定義することはできません。

注意： OCI では、SQL 問合せによって、通常、一時 LOB がセッションの存続期間内に戻されます。

LOB で SQL セマンティクスを使用する場合のパフォーマンス属性

LOB で SQL セマンティクスを使用する場合、次のパフォーマンスの問題に注意してください。

LOB 列への 4KB を超えるデータの挿入

Oracle9*i* では、SQL 問合せを処理する場合のすべての列データおよびバッファ・サイズに対する最大長の制限を、4KB を超えるサイズにできます。SQL では、4GB までの LOB データを処理できます。

データが 4KB を超える場合、一時 LOB を内部で使用して、中間結果を格納します。

注意： これによって、パフォーマンスが低下する可能性があります。大量の中間結果を扱うコストおよび一時 LOB へのアクセス効率の低さの両方によって、問合せの処理に余分な負荷が発生します。

大きな VARCHAR については、SQL 問合せが、以前の一連の LOB API を介して CLOB にアクセスするときと同様の方法を実行します。

一時 LOB の作成および割当て解除

PL/SQL、C（OCI）および Java では、SQL 問合せは、LOB 列に対する操作およびファンクション・コールの結果として、一時 LOB を戻します。次に例を示します。

```
SELECT substr(CLOB_Column, 4001, 32000) FROM ...
```

戻された一時 LOB は、PL/SQL プログラム・ブロック終了時に自動的に解放されます。

不要な一時 LOB は、いつでも解放するように選択して、システム・リソースおよび一時表領域を解放できます。SQL 問合せから戻された一時 LOB の割当てを適切に解除しないと、一時表領域が一杯となり、パフォーマンスが低下します。一時 LOB を明示的に解放する例については、「PL/SQL 例 6: 一時 LOB の自動的および手動による解放」を参照してください。

パフォーマンス測定

CLOB 列に対する SQL 問合せの実行パフォーマンスを、同じサイズの VARCHAR2 または LONG に対する問合せの実行パフォーマンスと比較する必要があります。LOB でのパフォーマンスは、VARCHAR2 または LONG でのパフォーマンスの 80% 以上である必要があります。

注意： システムおよびデータベースの管理 : 新しく提供された拡張 SQL セマンティクス機能をアプリケーションで使用すると、以前よりさらに多くの一時 LOB が暗黙的に作成されます。**これらの一時 LOB を格納する一時表領域の大きさが、アプリケーションに対して十分であること**を確認してください。

ユーザー定義集計および LOB

ユーザー定義集計（UDAG）は、アプリケーション開発者、カートリッジ開発者およびエンド・ユーザーに、オブジェクト型および不透明型と同様に（LOB を含む）スカラー・データ型を介して、新規の集計ファンクションを実装および配置するメカニズムを提供します。

ユーザー定義集計（UDAG）用の 2 つのアプリケーション例を次に示します。

- **Spatial Cartridge** Oracle Spatial Cartridge では、いくつかのファンクションが、空間問合せの結果としてジオメトリを戻します。これらのファンクションは、ユーザー定義集計ファンクションを使用してのみ集計できます。Web 配信などの場合、個別のジオメトリを複数送るより、問合せの結果を集計して、1 つの集計ジオメトリを送る方が効果的です。

  たとえば、各州の境界の和集合を求めることで州の境界を検索する問合せは、次のとおりです。

  ```
  SELECT SDO_AGGR_UNION(county.geometry)
  FROM COUNTIES
  GROUP BY county.state;
  ```

- **Trusted Oracle** Trusted Oracle では、行のラベルが LBAC_LABEL 不透明型としてモデル化されます。下限（GLB）や上限（LUB）などの行ラベルを介して集計ファンクションを定義するには、次のような問合せを効果的に実行できます。

  ```
  SELECT group_column, GLB(rowlabel)
  FROM x
  GROUP BY group_column.
  ```

ユーザー定義集計（UDAG）ファンクションは、ユーザー指定の集計セマンティクスを使用して集計ファンクションを参照します。新規の集計ファンクションを作成し、一連のルーチンを使用して集計ロジックを提供できます。ユーザー定義集計ファンクションを一度作成すると、組込みファンクションと同様に SQL DML 文で使用できます。

複合データは、通常、オブジェクト型、不透明型または LOB を使用してデータベースに格納されます。ユーザー定義集計は、これらのデータのドメインにわたる集計を指定する際に効果的です。

ユーザー定義集計を使用すると、金融アプリケーションまたは科学アプリケーション用の従来のスカラー・データ型を介して、新規の集計ファンクションも作成できます。すべての形式の集計にシステム固有のサポートを提供することはできませんが、この機能によって、新規の集計ファンクションを柔軟に追加できます。

集計ファンクションは、一連の値を入力としてとり、単一の値を戻します。集計用の一連の値は、通常 GROUP BY 句によって識別します。次に例を示します。

```
SELECT AVG(T.Sales)
  FROM AnnualSales T
  GROUP BY T.State
```

ユーザー定義集計を使用すると、前述の基本形の操作に対する固有の（新規）実装を指定することで、新規の集計ファンクションを登録できます。

UDAG: DDL のサポート

ユーザー定義集計ファンクションは、次の DDL をサポートします。

- スカラー・データ型（LOB を含む）およびユーザー定義データ型（オブジェクト型および不透明型）を介した、集計ファンクションの作成および削除
- ユーザー提供ルーチンの形式での集計ロジックの指定
- PL/SQL、C/C++ または Java の組合せでの、実装ルーチンの指定

UDAG: DML および問合せのサポート

ユーザー定義集計ファンクションは、次の DML および問合せをサポートします。

- SELECT リスト内の式の一部として、または HAVING 句の述語の一部として、UDAG をコールする SQL 文
- UDAG のシリアル評価およびパラレル評価の両方
- UDAG に対する入力パラメータに DISTINCT キーワードおよび ALL（デフォルト）キーワードの指定
- GROUP BY 拡張子を指定した UDAG - CUBE、ROLLUP およびグループ・セット
- メモリーをほとんど使用しないリフレッシュ・オプション付きの、UDAG を指定したマテリアライズド・ビュー
- UDAG を指定したウィンドウ・ファンクション

参照： 『Oracle9*i* Data Cartridge Developer's Guide』を参照してください。

注意： PL/SQL では、UDAG をプロシージャ文で使用できませんが、埋込み SQL では使用できます。

8

LONG から LOB への移行

この章の内容は次のとおりです。

LONG から LOB への移行の概要

Oracle は、LOB への移行を補助するために、LOB 用の LONG API をサポートしています。この API を使用すると、LONG 列を LOB に変更したときに、既存のアプリケーションを変更する必要はほとんどありません。

「LONG API」という用語は、DML、および LONG データ型の問合せを意味します。LONG API の例を次に示します。

- **OCI:** OCIBindByName()、OCIBindDynamic()、OCIDefineByPos()、OCIDefineDynamic() など
- **PL/SQL:** substr、instr など、およびパラメータの受渡し

> **注意：** LONG API は、LONG 以外のデータ型にも使用できます。ただし、この章では、LOB 用の API についてのみ説明します。ここでは、この API を「LONG-to-LOB API」といいます。

Oracle9*i* では、LOB データ型および LONG データ型をサポートしています。LOB の方がメリットが多いため、可能な場合は、LONG ではなく LOB を使用するように既存のアプリケーションを変更してください。詳細は、7-2 ページの「LOB 型と LONG 型および LONG RAW 型との比較」を参照してください。

この章では、LOB に対する LONG-to-LOB API の使用方法について説明します。

> **注意：** LONG-to-LOB API を使用すると、いくつかの LOB 操作のパフォーマンスが改善される場合があります。詳細は、8-42 ページの「パフォーマンス」を参照してください。

LONG-to-LOB API を使用した簡単な移行

LONG から LOB へ移行すると、LONG 列にアクセスしている既存のアプリケーションを簡単に移行して LOB 列を使用できます。移行は、次の 2 段階で行います。

- **データの移行 :** LOB を使用するために、LONG 列を含む既存の表を移動する手順です。詳細は、8-7 ページの「既存の表の LONG から LOB への移行」を参照してください。
- **アプリケーションの移行 :** LOB を使用するために、既存の LONG アプリケーションを変更する手順です。ただし、ほとんどの場合、アプリケーションを変更する必要はありません。LOB 用に実装された LONG-to-LOB API については、8-13 ページの「OCI での LONG-to-LOB API の使用」および 8-17 ページの「SQL および PL/SQL を使用した LONG および LOB へのアクセス」を参照してください。

注意： LONG には様々な制限があります。たとえば、1 つの表に含めることができる LONG 列は 1 つのみです。

一方、LOB にはこのような制限がありません。LOB を使用するために LONG 表を移行した後は、LONG の制限はなくなります。1 つの表に複数の LOB 列を含めたり、単一の INSERT または UPDATE で、4KB を超える複数のバインドが可能になります。

LONG-to-LOB API の使用のガイドライン

LONG-to-LOB API の使用のガイドラインを次に示します。

ALTER TABLE の使用

既存の表の LONG 列を LOB に変換するには、ALTER TABLE を使用します。8-7 ページの「LONG から LOB への移行 : ALTER TABLE を使用した LONG 列の LOB 型への変更」を参照してください。

LONG-to-LOB API および OCI

OCI でのバインド

以前のリリースでは、4,000 バイトを超えるデータの VARCHAR2 バッファをバインドできるのは LONG 列に対してのみでした。Oracle9*i* では、LONG-to-LOB API を使用することで、LOB に対してもこの機能を使用できるようになりました。LONG-to-LOB API を使用すると、次の場合に有効です。

- INSERT および UPDATE に対する標準、ピース単位およびコールバックのバインド
- INSERT および UPDATE に対する配列のバインド
- PL/SQL と OCI の境界でのパラメータの受渡し

次の OCI 関数は LONG-to-LOB API の一部です。

- OCIBindByPos()
- OCIBindByName()
- OCIBindDynamic()
- OCIStmtSetPieceInfo()
- OCIStmtGetPieceInfo()

これらの関数では、LOB 列の INSERT または UPDATE で次のデータ型を指定できます。

- CLOB 列 : SQLT_CHR および SQLT_LNG
- BLOB 列 : SQLT_BIN および SQLT_LBI

参照 :

- 8-14 ページの「OCI 関数を使用した LOB の INSERT または UPDATE」を参照してください。
- 8-20 ページの「OCI からの PL/SQL バインドおよび C バインド」を参照してください。

OCI での定義

LONG-to-LOB API を使用すると、次の OCI 関数に、VARCHAR2 バッファ、`SQLT_CHR`、`SQLT_LNG`、`SQLT_LBI` および `SQLT_BIN` データ型を LOB 列の出力として指定できます。

- OCIDefineByPos()
- OCIDefineDynamic()
- OCIStmtSetPieceInfo()
- OCIStmtGetPieceInfo()

これを行うと、LOB ロケータではなく LOB データが選択され、ユーザーのバッファに移動されます。

注意 : OCI の LONG-to-LOB API では、読込み量を指定できません。指定できるのは、バッファ長のみです。Oracle はユーザーのバッファに収まる量をすべて読み込みます。

OCI 関数による LOB でのピース単位および配列の INSERT、UPDATE またはフェッチ

前述の OCI 関数を使用して、LOB に対してピース単位の INSERT、UPDATE、フェッチ、および配列の INSERT、UPDATE、フェッチを実行できます。これらの関数によって、LOB への INSERT および UPDATE の実行時に、動的にデータを指定できます。

以前のリリースでは、LOB に対するバインド（INSERT および UPDATE）関数の動作は、LONG に対する動作と同じでした。

参照 : 詳細は、8-14 ページの「OCI 関数を使用した LOB の INSERT または UPDATE」を参照してください。

今回のリリースから、LOB に対して標準、ピース単位、コールバックおよび配列モードの定義（SELECT）が可能になりました。

> **参照：** 8-15 ページの「OCI 関数を使用した LOB のフェッチ」を参照してください。

マルチバイト・キャラクタ・セット（OCI）

クライアントのキャラクタ・セットがマルチバイトである場合、これらの関数は LONG の場合と同様に動作します。

- マルチバイト・キャラクタ・セットでのピース単位のフェッチでは、マルチバイト・キャラクタを 2 つに分割して、前半のバイトを 1 つのバッファに格納し、後半のバイトを次のバッファに格納できます。
- 標準のフェッチでは、最後の文字の一部をバッファに格納できない場合、Oracle はバッファに格納可能なサイズのバイト（部分的な文字）を戻します。

LONG-to-LOB API および PL/SQL

LOB 列の INSERT および UPDATE（PL/SQL）

以前のリリースでは、PL/SQL で、LOB 列に対して次のデータの INSERT または UPDATE が可能でした。

- CLOB 列 : VARCHAR2、CHAR、LONG などの文字データ
- BLOB 列 : RAW、LONG RAW などのバイナリ・データ

詳細は、8-41 ページの「PL/SQL インタフェース」を参照してください。

LOB 列の SELECT（PL/SQL）

PL/SQL では、CLOB 列に対して SELECT 文を実行できます。この場合、INTO 句に VARCHAR2、CHAR、LONG などの文字変数を指定します。詳細は、8-17 ページの「SQL および PL/SQL を使用した LOB へのアクセス」を参照してください。BLOB 列に対して SELECT 文を実行する場合も同様に、RAW、LONG RAW などのバイナリ変数を INTO 句に指定します。

> **注意：** PL/SQL の LONG-to-LOB API では、読込み量は指定できません。指定できるのは、バッファ長のみです。バッファ長が LOB データより短い場合、例外が発生します。

割当ておよびパラメータの受渡し（PL/SQL）

PL/SQL では、次の変数の**暗黙的な型変換**および割当てが可能です。

- CLOB 変数から VARCHAR2 変数、CHAR 変数、LONG 変数（またはその逆）
- BLOB 変数から RAW 変数および LONG RAW 変数（またはその逆）

パラメータの受渡しの場合も同様に、PL/SQL で次の受渡しが可能です。

- 実パラメータ CLOB から、仮パラメータが VARCHAR2、CHAR、LONG などのキャラクタ・タイプであるファンクション（またはその逆）
- 実パラメータ BLOB から、仮パラメータが RAW、LONG RAW などのバイナリ・タイプであるファンクションまたはプロシージャ（またはその逆）

参照：

- 8-23 ページの「NUMBER、DATE、ROW_ID、BINARY_INTEGER、PLS_INTEGER から LOB への暗黙的な変換の非サポート」を参照してください。
- 8-23 ページの「BLOB から VARCHAR2、CHAR、または CLOB から RAW、LONG RAW への暗黙的な変換の非サポート」を参照してください。

今回のリリースから、VARCHAR2 引数をとる PL/SQL **組込みファンクションおよび演算子**が、CLOB 引数もとるようになりました。このような PL/SQL 組込みファンクションおよび演算子には、INSTR、SUBSTR、比較演算子などがあります。

参照： これらの PL/SQL 組込みファンクションおよび演算子の詳細なリストは、8-19 ページの「PL/SQL 組込みファンクションでの VARCHAR2 および CLOB」を参照してください。

既存の表の LONG から LOB への移行

この項では、既存の表を LONG データ型から LOB データ型へ移行する方法を説明します。

LONG から LOB への移行 : ALTER TABLE を使用した LONG 列の LOB 型への変更

今回のリリースから、ALTER TABLE を使用して LONG 列を CLOB または NCLOB に、LONG_RAW 列を BLOB に変更できるようになりました。構文は次のとおりです。

```
ALTER TABLE [<schema>.]<table_name>
   MODIFY ( <long_column_name> { CLOB | BLOB | NCLOB }
  [DEFAULT <default_value>]) [LOB_storage_clause];
```

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);
```

Long_tab 表の long_col 列を CLOB データ型に変更するには、次のとおり指定します。

```
ALTER TABLE Long_tab MODIFY ( long_col CLOB );
```

注意 : 新しい ALTER TABLE 文によって可能な変更は、次のいずれかのみです。

- LONG 列の COLB 列または NCLOB 列への変更
- LONG RAW 列の BLOB 列への変更

VARCHAR 列または RAW 列は変更できません。

注意 : 新しい ALTER TABLE 文では、LONG 列を LOB に変更する以外には、次の操作のみが可能です。

- LOB 列に対するデフォルト値の指定
- LONG から LOB に変更された列に対する LOB 記憶域句の指定

LONG から LOB への移行に関しては、前述以外の ALTER TABLE 操作は行えません。

注意： LONG から LOB への移行 : Oracle8*i* で使用した方法

前述の LONG から LOB への移行方法は、Oracle8*i* で使用した方法にかわる今回のリリースからの新しい方法です。Oracle8*i* では、LONG に TO_LOB() という新しい演算子を追加し、この演算子を使用して LONG を LOB にコピーしました。TO_LOB 演算子を持つ CREATE TABLE AS SELECT 文または INSERT AS SELECT 文を使用して、データを LONG から LOB にコピーします。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);
```

次の文を入力します。

```
CREATE TABLE Lob_tab (id NUMBER, clob_col CLOB);
INSERT INTO Lob_tab SELECT id, TO_LOB(long_col) FROM long_tab;
DROP TABLE Long_tab;
CREATE VIEW Long_tab (id, long_col) AS SELECT * from Lob_tab;
```

この一連の操作は、Long_tab 表の Long_col 列のデータ型を LONG から CLOB に変更する操作と同じです。この方法（以前のリリースの方法）では、新しい表にすべての制約、トリガーおよび索引を再作成する必要があります。

LONG 列のすべての制約の保持

以前の LONG 列に対するすべての制約は、新しい LOB 列においても保持されます。LONG 列で有効な制約は、NULL および NOT-NULL のみです。これらの列に対する制約、またはこの表の他の列やプロパティを変更するには、移行後に ALTER TABLE 文を使用します。

LONG のデフォルト値の LOB へのコピー

デフォルト値を指定しない場合、LONG 列のデフォルト値が新しい LOB 列にコピーされます。

有効なトリガーの保持

表に対するほとんどの既存のトリガーは引き続き使用できますが、2 種類のトリガーで問題が発生する可能性があります。

参照： 詳細は、8-10 ページの「LONG から LOB への移行に関する制限事項」を参照してください。

索引の再作成ー ALTER INDEX...REBUILD の使用

LONG 列のドメイン索引は、LONG 列を LOB に変更する前に削除する必要があります。

表のすべての列にあるドメイン索引およびファンクション索引を含む他のすべての索引は使用不可になるため、ALTER INDEX <index name> REBUILD 文を使用して再作成する必要があります。

LONG から LOB への移行後の索引の再作成

任意の表に索引を再作成するには、LONG から LOB に移行した後、次の手順に従います。

1. LONG 列にドメイン索引がある場合、そのドメイン索引を削除します。
2. 次の文を実行します。

```
ALTER TABLE Long_tab MODIFY ( long_col CLOB...)...;
```

3. 次の文を実行します。

```
SELECT index_name FROM user_indexes WHERE table_name='LONG_TAB';
```

> **注意：** この問合せでは、表名を大文字で指定する必要があります。

4. 手順 3 でリストされたすべての索引 <index> に対して、次の文を実行します。

```
ALTER INDEX <index> REBUILD
```

5. 必要に応じて、LOB 列にドメイン索引を作成します。

一時的に 2 倍になる領域要件

ALTER TABLE MODIFY LONG->LOB 文は、表の内容を新しい領域にコピーし、操作の終了時に古い領域を解放します。このため、必要な領域が一時的に 2 倍になります。ただし、変換後に NULL が表に埋め込まれていないため、その後の DML または問合せのパフォーマンスが向上するというメリットがあります。

LOGGING

移行中の表に対する REDO がログに記録されるのは、表に LOGGING が指定されている場合のみです。また、LONG から LOB へ変換中の列に対する REDO がログに記録されるのは、LOB の記憶特性が LOGGING に指定されている場合のみです。LOB の LOGGING|NOLOGGING のデフォルト値は、LOB が作成されている表領域から継承されます。

移行中の REDO 領域の生成を回避して、正常に移行するには、次の手順に従います。

1. 次の文を実行します。

```
ALTER TABLE Long_tab NOLOGGING;
```

2. 次の文を実行します。

```
ALTER TABLE Long_tab MODIFY ( long_col CLOB [default <default_val>]) LOB
(long_col) STORE AS (... NOLOGGING...);
```

3. 次の文を実行します。

```
ALTER TABLE Long_tab MODIFY LOB long_col STORE AS (...LOGGING...);
```

4. 次の文を実行します。

```
ALTER TABLE Long_tab LOGGING;
```

5. 表および LOB を含む表領域のバックアップを取ります。

LONG から LOB への移行に関する制限事項

LONG から LOB への移行では、次のことに注意してください。

クラスタ化表

クラスタ化表には、LOB は作成できませんが、LONG は作成できます。そのため、表がクラスタの一部である場合、その表の LONG 列または LONG RAW 列は、LOB に変更できません。

レプリケーション

Oracle は、LONG および LONG RAW データ型を使用する列のレプリケーションをサポートしていません。これらのデータ型を含む列は、レプリケート表から削除されます。Oracle8*i* では、まず LONG データ型を LOB に変更してからレプリケートする必要があります。

これは、LONG から LOB への移行に関する制限事項ではありません。LONG から LOB へ移行すると、これらの列をレプリケートできます。

表がレプリケートされている場合、または表にマテリアライズド・ビューがある場合に LONG 列を LOB に変更するときは、手動でレプリカを修正する必要があります。

トリガー

トリガーには次のような問題があります。

UPDATE OF トリガーの UPDATE OF リストには、LOB 列を作成できません。LONG 列は作成できます。たとえば、次のような文は無効です。

```
create table t(lobcol CLOB);
create trigger trig before/after update of lobcol on t ...;
```

そのため、次のような場合にトリガーは無効になり、再コンパイルできません。

```
create table t(lobcol LONG);
create or replace trigger trig before (or after) update of lobcol on t
for each row
begin
  dbms_output.put_line('lmn');
end;
/
insert into t values('abc');
UPDATE t SET lobcol = 'xyz';

ALTER TABLE t MODIFY (lobcol CLOB); -- invalidates the trigger
UPDATE t SET lobcol = 'xyz'; -- doesn't execute because trigger
                            -- can't be revalidated
```

この制限事項は、今後のリリースではなくなる予定です。他のすべてのトリガーは問題なく機能します。

索引

移行する表のすべての列の**索引**を、手動で再作成する必要があります。これには、ファンクション索引およびドメイン索引も含まれます。

- LONG 列の**ドメイン索引**は、LONG 列を LOB に変更する前に削除する必要があります。
- LOB に変換された LONG 列の**ファンクション索引**は、コードを変更しなくても機能します。

LONG、LOB および NULL

LONG および LOB は、NULL の場合と長さ 0（ゼロ）の場合では動作が異なります。ただし、次に説明するとおり、LONG から LOB に移行するアプリケーションには影響しません。

次の 2 つの表、long_tab と lob_tab について考えてみます。

```
CREATE TABLE long_tab(id NUMBER, long_col LONG);
CREATE TABLE lob_tab(id NUMBER, lob_col LOB);
```

NULL の LONG と長さ 0（ゼロ）の LONG

長さ 0（ゼロ）の LONG と NULL の LONG は同じです。そのため、次の 2 つの文の結果は同じであり、どちらも LONG 列に NULL を挿入します。

```
INSERT INTO long_tab values(1, NULL);
INSERT INTO long_tab values(1, ''); -- Zero length string inserts NULL into the LONG
column
```

NULL の LOB と長さ 0（ゼロ）の LOB

LOB の場合も同様に、次の 2 つの文は、どちらも LOB 列に NULL を挿入します。

```
INSERT INTO lob_tab values(1, NULL);
INSERT INTO lob_tab values(1, '');  -- A zero length string inserts NULL into LOB
column
```

ただし、実際に empty_clob() コンストラクタを使用して長さ 0（ゼロ）の LOB を挿入した場合、LOB 列は NULL 以外になります。

```
INSERT INTO lob_tab values(1, empty_clob());  -- A zero length LOB is not the same
as NULL
```

OCI での LONG-to-LOB API の使用

以前のリリースの OCI では、LONG データに対してピース単位の INSERT、UPDATE およびフェッチを実行する場合、インタフェース・コールを提供していました。これらの API では、配列の INSERT または配列の UPDATE の場合、バインド値の静的配列を提供するかわりに、動的にデータを提供することも可能です。これらのピース単位の操作は、ポーリングまたはコールバックによって実行できます。

今回のリリースから、LOB に対して次の関数がサポートされるようになったため、LOB ロケータを使用することなく直接 LOB データを INSERT、UPDATE およびフェッチできます。

- `OCIBindByName()` または `OCIBindByPos()`: これらの関数を使用して、INSERT または UPDATE 用に SQL 文または PL/SQL ブロック内のプログラム変数とプレースホルダ間の対応付けを行います。
- `OCIBindDynamic()`: このコールを使用して、INSERT または UPDATE 用に動的にデータを割り当てるためのユーザー・コールバックを登録します。
- `OCIDefineByPos()`: このコールを使用して、選択リストにある項目を型および出力データ・バッファに対応付けます。
- `OCIDefineDynamic()`: このコールを使用して、`OCIDefineByPos()` で OCI_DYNAMIC_FETCH モードが選択された場合、SELECT に対してユーザー・コールバックを登録します。
- `OCIStmtGetPieceInfo()` および `OCIStmtSetPieceInfo()`: これらのコールを使用して、ピース単位操作のためのピース情報を取得および設定します。

> **参照：** API の詳細は、『Oracle Call Interface プログラマーズ・ガイド』の「ランタイム・データ割当てとピース単位操作」を参照してください。

OCI での LONG-to-LOB API の使用のガイドライン

前述した、今回のリリースの OCI 関数は、LOB に対して LONG と同様に動作します。この項では、これらの関数を使用して、データの INSERT、UPDATE およびフェッチを実行する方法について説明します。

注意： CLOB、BLOB、LONG および LONG RAW に対して前述の関数を使用する場合、次のようにデータ型を指定します。

- CLOB および LONG: SQLT_LNG および SQLT_CHR
- BLOB および LONG RAW: SQLT_LBI および SQLT_BIN

OCI 関数を使用した LOB の INSERT または UPDATE

LOB データを INSERT または UPDATE するには、様々な方法があります。

> **注意：** この項で説明する方法は、第 10 章「内部永続 LOB」で説明する LOB ロケータの挿入方法とは異なります。

この項で説明するすべての方法では、事前に OCI 環境を初期化し、すべての必要なハンドルを割り当てていることを想定しています。

1 つのピースでの標準の INSERT または UPDATE

1 つのピースで標準の INSERT および UPDATE を行うには、次の手順に従います。

1. `OCIStmtPrepare()` を OCI_DEFAULT モードで使用して、文を準備します。
2. `OCIBindByName()` または `OCIBindbyPos()` を OCI_DEFAULT モードで使用して、LOB を CHAR または BIN としてバインドするためにプレースホルダをバインドします。
3. `OCIStmtExecute()` を使用して、INSERT または UPDATE を実行します。

ポーリングによるピース単位の INSERT および UPDATE

ポーリングを使用してピース単位の INSERT または UPDATE を行うには、次の手順に従います。

1. `OCIStmtPrepare()` を OCI_DEFAULT モードで使用して、文を準備します。
2. `OCIBindByName()` または `OCIBindbyPos()` を OCI_DATA_AT_EXEC モードで使用して、LOB を CHAR または BIN としてバインドするためにプレースホルダをバインドします。
3. `OCIStmtExecute()` をデフォルト・モードで使用します。これによって OCI_NEED_DATA が戻されます。
4. OCI_NEED_DATA が戻されたら、次のことを行います。
   * `OCIStmtGetPieceInfo()` を使用して、挿入するピースの情報を取り出します。
   * `OCIStmtSetPieceInfo()` を使用して、挿入するピースの情報を設定します。
   * `OCIStmtExecute` を使用します。戻り値が OCI_SUCCESS である場合、手順は完了です。

コールバックによるピース単位の INSERT および UPDATE

コールバックを使用して INSERT および UPDATE を行うには、次の手順に従います。

1. `OCIStmtPrepare()` を OCI_DEFAULT モードで使用して、文を準備します。
2. `OCIBindByName()` または `OCIBindbyPos()` を OCI_DATA_AT_EXEC モードで使用して、LOB を CHAR または BIN としてバインドするために、プレースホルダをバインドします。
3. `OCIBindDynamic()` を使用してコールバックを指定します。
4. `OCIStmtExecute()` をデフォルト・モードで使用します。

配列の INSERT および UPDATE

前述のいずれかのモードを、`OCIBindArrayOfStruct()` と組み合せて使用するか、または `OCIStmtExecute()` コールに 1 より大きい繰返し値（iter）を指定して使用します。

OCI 関数を使用した LOB のフェッチ

LOB データをフェッチするには、次の 3 つの方法があります。

> **注意：** この項で説明する方法は、第 10 章「内部永続 LOB」で説明する LOB ロケータのフェッチ方法とは異なります。

- 1 つのピースでの標準のフェッチ
- ピース単位のフェッチ
- 配列のフェッチ

1 つのピースでの標準のフェッチ

1 つのピースで LOB の標準のフェッチを行うには、次の手順に従います。

1. `OCIStmtPrepare()` を OCI_DEFAULT モードで使用して、SELECT 文を準備します。
2. `OCIDefineByPos()` を OCI_DEFAULT モードで使用して、LOB を CHAR または BIN として定義するために、選択リストの位置を定義します。
3. `OCIStmtExecute()` を使用して SELECT 文を実行します。
4. `OCIStmtFetch()` を使用してフェッチを実行します。

ポーリングによるピース単位のフェッチ

ポーリングを使用してピース単位のフェッチを行うには、次の手順に従います。

1. OCIStmtPrepare() を OCI_DEFAULT モードで使用して、SELECT 文を準備します。
2. OCIDefinebyPos() を OCI_DYNAMIC_FETCH モードで使用して、LOB を CHAR または BIN として定義するために、選択リストの位置を定義します。
3. OCIStmtExecute() を使用して SELECT 文を実行します。
4. OCIStmtFetch() をデフォルト・モードで使用します。これによって OCI_NEED_DATA が戻されます。
5. OCI_NEED_DATA が戻されたら、次のことを行います。
   * OCIStmtGetPieceInfo() を使用して、フェッチするピースの情報を取り出します。
   * OCIStmtSetPieceInfo() を使用して、フェッチするピースの情報を設定します。
   * OCIStmtFetch を実行します。戻り値が OCI_SUCCESS である場合、手順は完了です。

コールバックによるピース単位のフェッチ

コールバックを使用してピース単位のフェッチを行うには、次の手順に従います。

1. OCIStmtPrepare() を OCI_DEFAULT モードで使用して、文を準備します。
2. OCIDefinebyPos() を OCI_DYNAMIC_FETCH モードで使用して、LOB を CHAR または BIN として定義するために、選択リストの位置を定義します。
3. OCIStmtExecute() を使用して SELECT 文を実行します。
4. OCIDefineDynamic() を使用して、コールバックを指定します。
5. OCIStmtFetch() をデフォルト・モードで使用します。

配列のフェッチ

前述のいずれかのモードを、OCIDefineArrayOfStruct() と組み合せて使用するか、または OCIStmtExecute() コールに 1 より大きい繰返し値（iter）を指定して使用します。

SQL および PL/SQL を使用した LONG および LOB へのアクセス

この項の内容は次のとおりです。

SQL および PL/SQL を使用した LOB へのアクセス

CLOB 列および BLOB 列からのデータは、INSERT 文、UPDATE 文、SELECT 文などの通常の SQL 文を使用して参照できます。

PL/SQL には、ピース単位の INSERT、UPDATE、フェッチ用のルーチンがありません。そのため、LOB 列からアクセスできるデータ量は、文字バッファ・サイズの最大値によって制限されます。Oracle9*i* では、PL/SQL は文字バッファ・サイズを 32,767 バイトまでサポートします。このため PL/SQL アプリケーションがアクセスできるのは、サイズが 32,767 バイト以下の LOB のみです。

32KB を超える LOB にアクセスする必要がある場合は、PL/SQL コードから OCI コールアウトを実行し、API を使用してピース単位の INSERT およびフェッチを行う必要があります。

LOB 列にアクセスするためのガイドラインは次のとおりです。

INSERT

LOB 列を含む表には、VALUE 句で通常の INSERT を使用してデータを挿入できます。LOB 列のフィールドには、PL/SQL の文字バッファ変数またはバイナリ・バッファ変数（CHAR、VARCHAR2、RAW など）、文字列リテラルまたは LOB ロケータを指定できます。

UPDATE

LOB 列は、UPDATE...SET 文によって全体を更新できます。LOB 列に格納されているデータは、ランダムにアクセスできません。SET 句には、新しい値として、リテラル、PL/SQL の文字変数またはバイナリ変数、LOB ロケータが指定できます。

4,000 バイトを超える LONG RAW バッファおよび RAW バッファに対する制限事項　LONG および LOB に対するバインドには制限事項があります。SQL は 4,000 バイトを超えるバッファに対して暗黙的な HEXTORAW 変換を行わないため、バッファ・サイズが 4,000 バイトを超える場合、LONG RAW または BLOB 列に VARCHAR2 バッファをバインドできません。同様に、SQL は 4,000 バイトを超えるバッファに対して暗黙的な RAWTOHEX 変換を行わないため、バッファ・サイズが 4,000 バイトを超える場合、LONG または CLOB 列に VARCHAR2 バッファをバインドできません。

SELECT

以前のリリースでは、フェッチの際に、LOB 列に文字変数をバインドするために SELECT INTO を使用することはできませんでした。SELECT INTO は、LOB ロケータを列にバインドするために使用していました。この制約はなくなりました。

PL/SQL では、LOB 列を選択して文字バッファまたはバイナリ・バッファに格納できます。LOB 列がバッファ・サイズより大きい場合、データはバッファに格納されず、例外が発生します。LOB 列は、LOB ロケータとしても選択できます。

暗黙的な割当ておよびパラメータの受渡し

LONG-to-LOB 移行 API は、LONG（LONG RAW）変数または VARCHAR2（RAW）変数への CLOB（BLOB）変数の割当て、およびその逆もサポートします。これは、PL/SQL に %type および %rowtype のデータ型が存在するためです。この割当てには、パラメータの受渡しも含まれます。この項では、これらの機能について説明します。

CLOB/CHAR および BLOB/RAW 間の変数の割当て

CLOB と CHAR 間および BLOB と RAW 間では、次のような変数の割当てが可能です。

```
CLOB_VAR := CHAR_VAR;
CHAR_VAR := CLOB_VAR;
BLOB_VAR := RAW_VAR;
RAW_VAR := BLOB_VAR;
```

これは、既存のコードに %type および %rowtype が指定されている場合に可能です。次に例を示します。

```
CREATE TABLE t (long_col LONG); -- Alter this table to change LONG column to LOB
DECLARE
   a VARCHAR2(100);
   b t.long_col%type; -- This variable changes from LONG to CLOB
BEGIN
   SELECT * INTO b FROM t;
   a := b;  -- This changes from "VARCHAR2 := LONG to VARCHAR2 := CLOB
   b := a;  -- This changes from "LONG := VARCHAR2 to CLOB := VARCHAR2
END;
```

ファンクション / プロシージャのパラメータの受渡し

この機能によって、VARCHAR2、LONG、RAW および LONG RAW が仮パラメータである場合、すべてのユーザー定義プロシージャおよびファンクションが CLOB および BLOB を実パラメータとして使用できます。またその逆も可能です。さらに、INSTR のような PL/SQL 組込みファンクションに、文字列以外に CLOB データを指定することもできます。次に例を示します。

```
CREATE PROCEDURE FOO ( a IN OUT t.long_col%type) IS......
CREATE PROCEDURE BAR (b IN OUT VARCHAR2) IS ...
DECLARE
    a VARCHAR2(100);
    b t.long_col%type  -- This changes to CLOB
BEGIN
    a := 'abc';
    SELECT long_col into b from t;
    FOO(a); -- Actual parameter is VARCHAR2, formal parameter is CLOB
    BAR(b); -- Actual parameter is CLOB, formal parameter is VARCHAR2
END;
```

明示的な変換ファンクション

PL/SQL では、LONG から LOB への移行の一部としてその他のデータ型を CLOB および BLOB に変換するために、次の 2 つの明示的な変換ファンクションが追加されました。

- TO_CLOB(): LONG、VARCHAR2 および CHAR を CLOB に変換
- TO_BLOB(): LONG RAW および RAW を BLOB に変換

TO_CHAR() を使用して、CLOB を CHAR 型へ変換できるようになりました。

PL/SQL 組込みファンクションでの VARCHAR2 および CLOB

PL/SQL の VARCHAR2 ファンクションおよび演算子は、CLOB を引数またはオペランドとしてとります。CLOB を SQL および PL/SQL の VARCHAR2 組込みファンクションに渡して、VARCHAR2 と同じように動作させることができます。または VARCHAR2 変数を DBMS_LOB API に渡して、LOB ロケータのように動作させることができます。

CLOB パラメータを指定するか、または CLOB 出力を戻す PL/SQL 組込みファンクションを次に示します。

- LENGTH、LENGTHB
- INSTR、INSTRB
- SUBSTR、SUBSTRB
- CONCAT、||
- LPAD、RPAD

- LTRIM、RTRIM、TRIM
- LIKE
- REPLACE
- LOWER、UPPER
- NLS_LOWER、NLS_UPPER
- NVL
- 比較演算子（>、=、<、!=）

ファンクションが戻した CLOB が VARCHAR2 変数に割り当てられますが、VARCHAR2 変数がその CLOB を格納に十分な大きさを持っていない場合、エラーが発生し、操作は行われません。CLOB を VARCHAR2 に指定して SELECT を実行する場合も同じです。これは現行の VARCHAR2 の動作と一致しています。

これらのファンクションは、暗黙的に一時 LOB を作成します。このため、永続から一時に変更される LOB ロケータもあります。その結果、（一時）LOB ロケータが指すデータに対する変更は、LOB ロケータが最初に指していた永続 LOB に反映されません。

これらの一時 LOB は PL/SQL ブロックの終了時に自動的に解放されます。CLOB 変数に対して `DBMS_LOB.FREE_TEMPORARY()` をコールして、明示的に一時 LOB を解放し、システム・リソースおよび一時表領域を再生することもできます。

> **参照：** 7-33 ページの「LOB に対する SQL セマンティクスのサポート」を参照してください。

OCI からの PL/SQL バインドおよび C バインド

OCI から PL/SQL プロシージャをコールして、インバインドまたはアウトバインド（あるいはその両方）を行う場合、次のいずれかを行えます。

- PL/SQL プロシージャの仮パラメータが SQLT_CLOB である場合、変数を SQLT_CHR または SQLT_LNG としてバインドする。
- 仮パラメータが SQLT_BLOB である場合、変数を SQLT_BIN または SQLT_LBI としてバインドする。

次に、有効な 2 つの例を示します。

「begin foo(:1); end;」形式での PL/SQL アウトバインドのコール

「begin foo(:1); end;」形式で PL/SQL アウトバインドをコールする例を次に示します。

```
text *sqlstmt = (text *)"BEGIN get_lob(:c); END; " ;
```

「call foo(:1);」形式での PL/SQL アウトバインドのコール

「call foo(:1);」形式で PL/SQL アウトバインドをコールする例を次に示します。

```
text *sqlstmt = (text *)"CALL get_lob( :c );" ;
```

どちらの例でも、プログラムの後半は次のようになります。

```
OCIStmtPrepare(stmthp, errhp, sqlstmt, (ub4)strlen((char *)sqlstmt),
        ub4) OCI_NTV_SYNTAX, (ub4) OCI_DEFAULT);
   curlen = 0;
OCIBindByName(stmthp, &bndhp[3], errhp,
        (text *) ":c", (sb4) strlen((char *) ":c"),
        (dvoid *) buf5, (sb4) LONGLEN, SQLT_CHR,
        (dvoid *) 0, (ub2 *) 0, (ub2 *) 0,
        (ub4) 1, (ub4 *) &curlen, (ub4) OCI_DATA_AT_EXEC);
```

PL/SQL プロシージャ get_lob() を次に示します。

```
procedure get_lob(c INOUT CLOB) is  -- This might have been column%type
  begin
  ... /* The procedure body could be in PL/SQL or C*/
  end;
```

SQL または PL/SQL からの PL/SQL プロシージャおよび C プロシージャのコール

SQL からのコール

SQL から PL/SQL プロシージャをコールする場合、LONG パラメータは使用できません。つまり、このコールは LONG から LOB への変換処理の一部ではありません。

PL/SQL からのコール

PL/SQL から PL/SQL プロシージャまたは C プロシージャをコールできます。CHR が仮パラメータの場合、CLOB を実パラメータとして渡せます。またその逆も可能です。BLOB および RAW の場合も同様です。

このようなコールは、仮パラメータか実パラメータのどちらかがアンカー型（table%type）である場合に実行されます。

LONG から LOB に変換する場合のアプリケーションの変更

暗黙的に LOB へ変換する場合でも、アプリケーションを変更する必要がある場合があります。この項では、アプリケーションを変更する必要がある場合について説明します。

アンカー型でのオーバーロード

アンカー型を使用したアプリケーションでは、LOB への変換中に、オーバーロードが暗黙的に別のターゲットに変換される場合があります。次に例を示します。

```
procedure p(l long) is ...;       -- (1)
procedure p(c clob) is ...;       -- (2)
```

次のようにコールするとします。

```
declare
     var  longtab.longcol%type;
   begin
     ...
   p(var);
     ...
end;
```

LOB へ移行する前は、このコールはオーバーロード（1）に変換されます。longtab を LOB に移行すると、このコールは暗黙的にオーバーロード（2）に変換されます。（1）のパラメータが CHAR、VARCHAR2、RAW、LONG RAW である場合も、同様の問題が発生します。

LONG 列を LOB に移行する場合、依存するアプリケーションを手動で検査して修正する必要があります。

このような新しい変換動作のため、LOB 引数を含む、プロシージャのオーバーロードを持つ既存のアプリケーションが正常に動作しなくなる場合があります。これには、LONG のアンカー型を使用しないアプリケーションも含まれます。次に例を示します。

```
procedure p(n number) is ...;       -- (1)
procedure p(c clob) is ...;         -- (2)

p('abc');
```

以前のリリースでは、CHAR から NUMBER への変換のみが可能であったため、(1) が選択されていました。今回のリリースでは、両方の変換が可能なため、コールの解釈が不明確になり、オーバーロード・エラーが発生します。

NUMBER、DATE、ROW_ID、BINARY_INTEGER、PLS_INTEGER から LOB への暗黙的な変換の非サポート

PL/SQL では、現在、NUMBER、DATE、ROW_ID、BINARY_INTEGER、PLS_INTEGER から LONG への変換が可能です。これらの型から LOB への明示的または暗黙的な変換をサポートする予定はありません。そのため、このような変換が必要な場合は、これらのデータ型を明示的に TO_CHAR に変換する必要があります。次のような形式の割当てがあるとします。

```
number_var := long_var;  -- The RHS becomes a LOB variable after conversion
```

この場合、次のようにコードを明示的に変更する必要があります。

```
number_var := TO_CHAR(long_var); -- Note that long_var is of type CLOB after
conversion
```

BLOB から VARCHAR2、CHAR、または CLOB から RAW、LONG RAW への暗黙的な変換の非サポート

次のような暗黙的な変換もサポートされません。

- BLOB から VARCHAR2、CHAR または LONG
- CLOB から RAW または LONG RAW

次のようなコードがあるとします。

```
SELECT <long raw column> INTO <varchar2> VARIABLE FROM <table>
```

ここで LONG RAW 列を BLOB に変換すると、この SELECT 文は機能しません。選択した変数に、TO_RAW または TO_CHAR 変換演算子を追加する必要があります。次に例を示します。

```
SELECT TO_RAW(<long raw column>) INTO <varchar2> VARIABLE FROM <table>
-- note that the long raw column is now a BLOB column
```

CLOB を RAW 変数に選択する場合、CLOB を RAW に割り当てる場合、および BLOB を VARCHAR2 に割り当てる場合も同様です。

utldtree.sql を使用したアプリケーションの変更部分の判断

LONG 表を LOB に ALTER する場合、アプリケーションの変更が必要な部分を判断するには、rdbms/admin/utldtree.sql ユーティリティを使用します。

utldtree.sql を使用すると、指定のオブジェクトに（再帰的に）依存するすべてのオブジェクトを参照できます。さらに、ユーザーに権限があるオブジェクトのみを参照できます。

utldtree.sql の使用方法は、utldtree.sql ファイル内に記載されています。このファイルを使用して、LONG 列を持つ表に依存するすべてのオブジェクトを参照し、それらのオブジェクトを 8-22 ページの「LONG から LOB に変換する場合のアプリケーションの変更」に示す例と比較して、アプリケーションを変更する必要があるかどうかを判断できます。

utldtree.sql が必要となるのは PL/SQL の場合のみです。SQL および OCI の場合は、アプリケーションを変更する必要はありません。

Multimedia_tab 表を使用した LONG から LOB への変換例

Multimedia_tab スキーマの詳細は、付録 B「マルチメディア・スキーマ」を参照してください。次の例で使用するフィールドを次に示します。

```
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;
```

MULTIMEDIA_TAB 表の STORY 列が LONG 型であったと想定します。つまり、MULTIMEDIA_TAB 表は次のとおり作成されたとします。

```
CREATE TABLE MULTIMEDIA_TAB (CLIP_ID NUMBER,
    STORY  LONG,
    .... );
```

ALTER TABLE を使用した LONG から CLOB への変換

LONG 列を CLOB に変換するには、次のとおり ALTER TABLE を使用します。

```
ALTER TABLE multimedia_tab MODIFY ( story CLOB );
```

これで変換が実行されます。

MULTIMEDIA_TAB 表を使用したすべての既存のアプリケーションは、STORY 列を CLOB 型に変更した後も、ほとんど変更することなく使用できます。

LONG で使用していて、8-22 ページの「LONG から LOB に変換する場合のアプリケーションの変更」で説明する変更を行うことで LOB でも使用できるすべての操作（バインドおよび定義）の例を次に示します。

LONG から LOB への変換例 1: 4KB を超えるバインドおよび標準の INSERT

4KB を超えるバインドおよび標準の INSERT を使用する場合の、LONG から LOB への変換の例を次に示します。

```
word buflen, buf1 = 0;
text buf2[5000];
text *insstmt = (text *)
"INSERT INTO MULTIMEDIA_TAB(CLIP_ID, STORY) VALUES
(:CLIP_ID, :STORY)";

if (OCIStmtPrepare(stmthp, errhp, insstmt,
(ub4)strlen((char *)insstmt), (ub4) OCI_NTV_SYNTAX,
(ub4) OCI_DEFAULT))
{
    DISCARD printf("FAILED: OCIStmtPrepare()\n");
    report_error(errhp);
    return;
}

if (OCIBindByName(stmthp, &bndhp[0], errhp,
  (text *) ":CLIP_ID", (sb4) strlen((char *) ":CLIP_ID"),
  (dvoid *) &buf1, (sb4) sizeof(buf1), SQLT_INT,
  (dvoid *) 0, (ub2 *) 0, (ub2 *) 0,
  (ub4) 0, (ub4 *) 0, (ub4) OCI_DEFAULT)
  ||  OCIBindByName(stmthp, &bndhp[1], errhp,
  (text *) ":STORY", (sb4) strlen((char *) ":STORY"),
  (dvoid *) buf2, (sb4) sizeof(buf2), SQLT_CHR,
  (dvoid *) 0, (ub2 *) 0, (ub2 *) 0,
  (ub4) 0, (ub4 *) 0, (ub4) OCI_DEFAULT))
{
    DISCARD printf("FAILED: OCIBindByName()\n");
    report_error(errhp);
```

```
    return;
}

buf1 = 101;
memset((void *)buf2, (int)'A', (size_t)5000);

if (OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
           (const OCISnapshot*) 0, (OCISnapshot*) 0,
           (ub4) OCI_DEFAULT))
{
      DISCARD printf("FAILED: OCIStmtExecute()\n");
      report_error(errhp);
      return;
}
```

LONG から LOB への変換例 2: ポーリングによるピース単位の INSERT

前述の例の続きです。

```
text *sqlstmt  = (text *)"INSERT INTO MULTIMEDIA_TAB VALUES (:1, :2)";
ub2 rcode;
ub1 piece, i;

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

OCIBindByPos(stmthp, &bndhp[0], errhp, (ub4) 1,
                  (dvoid *) &buf1, (sb4) sizeof(buf1), SQLT_INT,
                  (dvoid *) 0, (ub2 *)0, (ub2 *)0,
                  (ub4) 0, (ub4 *) 0, (ub4) OCI_DEFAULT);

OCIBindByPos(stmthp, &bndhp[1], errhp, (ub4) 2,
                  (dvoid *) 0, (sb4) 15000, SQLT_LNG,
                  (dvoid *) 0, (ub2 *)0, (ub2 *)0,
                  (ub4) 0, (ub4 *) 0, (ub4) OCI_DATA_AT_EXEC);
buf1 = 101;
i = 0;
while (1)
{
    i++;
    retval = OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                  (CONST OCISnapshot*) 0, (OCISnapshot*) 0,
                  (ub4) OCI_DEFAULT);

switch(retval)
```

```
    {
          case OCI_NEED_DATA:

        memset((void *)buf2, (int)'A'+i, (size_t)5000);
        buflen = 5000;
        if (i == 1) piece = OCI_ONE_PIECE
            else if (i == 3) piece = OCI_LAST_PIECE
            else piece = OCI_NEXT_PIECE;

        if (OCIStmtSetPieceInfo((dvoid *)bndhp[1],
                  (ub4)OCI_HTYPE_BIND, errhp, (dvoid *)buf2,
                  &buflen, piece, (dvoid *) 0, &rcode))
        {
          DISCARD printf("ERROR: OCIStmtSetPieceInfo: %d \n", retval);
          break;
        }

        retval = OCI_NEED_DATA;
        break;
      case OCI_SUCCESS:
        break;
      default:
        DISCARD printf( "oci exec returned %d \n", retval);
        report_error(errhp);
        retval = 0;
    }    /* end switch */
    if (!retval) break;
} /* end while(1) */
```

LONG から LOB への変換例 3: コールバックによるピース単位の INSERT

コールバックによるピース単位の INSERT を使用する場合の、LONG から LOB への変換の例を次に示します。

```
void insert_data()
{
text *sqlstmt = (text *) "INSERT INTO MULTIMEDIA_TAB VALUES (:1, :2)";
OCIStmtPrepare(stmthp, errhp, sqlstmt, (ub4)strlen((char *)sqlstmt),
                    (ub4) OCI_NTV_SYNTAX, (ub4) OCI_DEFAULT)
/* bind input */
if (OCIBindByPos(stmthp, &bndhp[0], errhp, (ub4) 1,
                      (dvoid *) 0, (sb4) sizeof(buf1), SQLT_INT,
                      (dvoid *) 0, (ub2 *)0, (ub2 *)0,
                      (ub4) 0, (ub4 *) 0, (ub4) OCI_DATA_AT_EXEC)
   || OCIBindByPos(stmthp, &bndhp[1], errhp, (ub4) 2,
                      (dvoid *) 0, (sb4) 3 * sizeof(buf2), SQLT_CHR,
```

```
                    (dvoid *) 0, (ub2 *)0, (ub2 *)0,
                    (ub4) 0, (ub4 *) 0, (ub4) OCI_DATA_AT_EXEC))
{
    DISCARD printf("FAILED: OCIBindByPos()\n");
    report_error(errhp);
    return OCI_ERROR;
}
for (i = 0; i < MAXCOLS; i++)
    pos[i] = i+1;
if (OCIBindDynamic(bndhp[0], errhp, (dvoid *) (dvoid *) &pos[0],
        cbf_in_data,(dvoid *) 0, (OCICallbackOutBind) 0)
  ||  OCIBindDynamic(bndhp[1], errhp, (dvoid *) (dvoid *) &pos[1],
        cbf_in_data, (dvoid *) 0, (OCICallbackOutBind) 0))
{
    DISCARD printf("FAILED: OCIBindDynamic()\n");
    report_error(errhp);
    return OCI_ERROR;
}
OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                    (const OCISnapshot*) 0, (OCISnapshot*) 0,
                    (ub4) OCI_DEFAULT)
} /* end insert_data() */

/* Inbind callback to specify input data.                          */
STATICF sb4 cbf_in_data(ctxp, bindp, iter, index, bufpp, alenpp,
                                        piecep, indpp)
dvoid *ctxp;
OCIBind *bindp;
ub4 iter;
ub4 index;
dvoid **bufpp;
ub4 *alenpp;
ub1 *piecep;
dvoid **indpp;
{
  static int a = 0;
  word j;
  ub4  inpos = *((ub4 *)ctxp);
    switch(inpos)
    {
    case 1:
    buf1 = 175;
    *bufpp = (dvoid *) &buf1;
    *alenpp = sizeof(buf1);
    break;
    case 2:
    memset((void *)buf2, (int) 'A'+a, (size_t) 5000);
```

```
    *bufpp = (dvoid *) buf2;
    *alenpp = 5000 ;
     a++;
    break;
    default: printf("ERROR: invalid position number: %d\n", pos);
    }
    *indpp = (dvoid *) 0;
      *piecep = OCI_ONE_PIECE;
      if (inpos == 2)
    {
    if (a<=1)
    {
      *piecep = OCI_FIRST_PIECE;
      printf("Insert callback: 1st piece\n");
    }
    else if (a<3)
    {
      *piecep = OCI_NEXT_PIECE;
      printf("Insert callback: %d'th piece\n", a);
    }
    else {
      *piecep = OCI_LAST_PIECE;
      printf("Insert callback: %d'th piece\n", a);
      a = 0;
    }
    }
    return OCI_CONTINUE;
}
```

LONG から LOB への変換例 4: 配列の INSERT

配列の INSERT を使用する場合の、LONG から LOB への変換の例を次に示します。

```
word buflen;
word arrbuf1[5];
text arrbuf2[5][5000];
text *insstmt = (text *)
"INSERT INTO MULTIMEDIA_TAB(CLIP_ID, STORY) VALUES
(:CLIP_ID, :STORY)";

if (OCIStmtPrepare(stmthp, errhp, insstmt,
(ub4)strlen((char *)insstmt), (ub4) OCI_NTV_SYNTAX,
(ub4) OCI_DEFAULT))
{
    DISCARD printf("FAILED: OCIStmtPrepare()\n");
    report_error(errhp);
```

```
    return;
}

if (OCIBindByName(stmthp, &bndhp[0], errhp,
  (text *) ":CLIP_ID", (sb4) strlen((char *) ":CLIP_ID"),
  (dvoid *) &arrbuf1[0], (sb4) sizeof(arrbuf1[0]), SQLT_INT,(dvoid *) 0, (ub2 *)
   0, (ub2 *) 0,
  (ub4) 0, (ub4 *) 0, (ub4) OCI_DEFAULT)
  ||  OCIBindByName(stmthp, &bndhp[1], errhp,
  (text *) ":STORY", (sb4) strlen((char *) ":STORY"),
  (dvoid *) arrbuf2[0], (sb4) sizeof(arrbuf2[0]), SQLT_CHR,
  (dvoid *) 0, (ub2 *) 0, (ub2 *) 0,
  (ub4) 0, (ub4 *) 0, (ub4) OCI_DEFAULT))
{
    DISCARD printf("FAILED: OCIBindByName()\n");
    report_error(errhp);
    return;
}
OCIBindArrayOfStruct(bndhp[0], ERRH, sizeof(arrbuf1[0]),
                indsk, rlsk, rcsk);
OCIBindArrayOfStruct(bndhp[1], ERRH, sizeof(arrbuf2[0]),
                indsk, rlsk, rcsk);
for (i=0; i<5; i++)
{
    arrbuf1[i] = 101+i;
    memset((void *)arrbuf2[i], (int)'A'+i, (size_t)5000);
}

if (OCIStmtExecute(svchp, stmthp, errhp, (ub4) 5, (ub4) 0,
                (const OCISnapshot*) 0, (OCISnapshot*) 0,
                (ub4) OCI_DEFAULT))
{
      DISCARD printf("FAILED: OCIStmtExecute()\n");
      report_error(errhp);
      return;
}
```

LONG から LOB への変換例 5: 標準のフェッチ

標準のフェッチを使用する場合の、LONG から LOB への変換の例を次に示します。

```
word   i, buf1 = 0;
text   buf2[5000];

text *selstmt = (text *) "SELECT CLIP_ID, STORY FROM MULTIMEDIA_TAB
                         ORDER BY CLIP_ID";
OCIStmtPrepare(stmthp, errhp, selstmt, (ub4)strlen((char *)selstmt),
                        (ub4) OCI_NTV_SYNTAX, (ub4) OCI_DEFAULT);
retval = OCIStmtExecute(svchp, stmthp, errhp, (ub4) 0, (ub4) 0,
                        (const OCISnapshot*) 0, (OCISnapshot*) 0,
                        (ub4) OCI_DEFAULT);

while (retval == OCI_SUCCESS || retval == OCI_SUCCESS_WITH_INFO)
{
    OCIDefineByPos(stmthp, &defhp1, errhp, (ub4) 1, (dvoid *) &buf1,
                        (sb4) sizeof(buf1), (ub2) SQLT_INT, (dvoid *) 0,
                        (ub2 *) 0,(ub2 *) 0, (ub4) OCI_DEFAULT);
    OCIDefineByPos(stmthp, &defhp2, errhp, (ub4) 2, (dvoid *) buf2,
                        (sb4) sizeof(buf2), (ub2) SQLT_CHR, (dvoid *) 0,
                        (ub2 *) 0, (ub2 *) 0, (ub4) OCI_DEFAULT);
    retval = OCIStmtFetch(stmthp, errhp, (ub4) 1,
                        (ub4) OCI_FETCH_NEXT, (ub4) OCI_DEFAULT);
    if (retval == OCI_SUCCESS || retval == OCI_SUCCESS_WITH_INFO)
       DISCARD printf("buf1 = %d,  buf2 = %.*s\n", buf1, 30, buf2);
}
```

LONG から LOB への変換例 6: ポーリングによるピース単位のフェッチ

ポーリングによるピース単位のフェッチを使用する場合、LONG から LOB への変換の例を次に示します。

```
text *selstmt = (text *) "SELECT STORY FROM MULTIMEDIA_TAB
                            ORDER BY CLIP_ID";
OCIStmtPrepare(stmthp, errhp, sqlstmt,
                  (ub4) strlen((char *)sqlstmt),
                  (ub4) OCI_NTV_SYNTAX, (ub4) OCI_DEFAULT);
OCIDefineByPos(stmthp, &dfnhp[1], errhp, (ub4) 1,
                  (dvoid *) NULL, (sb4) 100000, SQLT_LNG,
                  (dvoid *) 0, (ub2 *) 0,
                  (ub2 *) 0, (ub4) OCI_DYNAMIC_FETCH);
retval = OCIStmtExecute(svchp, stmthp, errhp, (ub4) 0, (ub4) 0,
                                 (CONST OCISnapshot*) 0, (OCISnapshot*) 0,
                                     (ub4) OCI_DEFAULT);
retval = OCIStmtFetch(stmthp, errhp, (ub4) 1 ,
```

```
                                (ub2) OCI_FETCH_NEXT, (ub4) OCI_DEFAULT);

while (retval != OCI_NO_DATA && retval != OCI_SUCCESS)
{
    ub1    piece;
    ub4    iter, buflen;
    ub4    idx;
     genclr((void *)buf2, 5000);

    switch(retval)
    {
     case OCI_NEED_DATA:
        OCIStmtGetPieceInfo(stmthp, errhp, &hdlptr, &hdltype,
                            &in_out, &iter, &idx, &piece);
        OCIStmtSetPieceInfo(hdlptr, hdltype, errhp,
                              (dvoid *) buf2, &buflen, piece,
                                    (CONST dvoid *) &indp1, (ub2 *) 0));
        retval = OCI_NEED_DATA;
            break;
    default:
      DISCARD printf("ERROR: piece-wise fetching\n");
        return;
    } /* end switch */

    retval = OCIStmtFetch(stmthp, errhp, (ub4) 1 ,
                              (ub2) OCI_FETCH_NEXT, (ub4) OCI_DEFAULT);
    printf("Data : %s\n"; buf2);

} /* end while */
```

LONG から LOB への変換例 7: コールバックによるピース単位のフェッチ

コールバックによるピース単位のフェッチを使用する場合の、LONG から LOB への変換の例を次に示します。

```
select()
{
text *sqlstmt = (text *) "SELECT CLIP_ID, STORY FROM MULTIMEDIA_TAB";

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

OCIDefineByPos(stmthp, &dfnhp[0], errhp, (ub4) 1,
                    (dvoid *) 0, (sb4) sizeof(buf1), SQLT_INT,
                    (dvoid *) 0, (ub2 *)0, (ub2 *)0,
                    (ub4) OCI_DYNAMIC_FETCH);
```

```
OCIDefineByPos(stmthp, &dfnhp[1], errhp, (ub4) 2,
                    (dvoid *) 0, (sb4)3 * sizeof(buf2), SQLT_CHR,
                    (dvoid *) 0, (ub2 *)0, (ub2 *)0,
                    (ub4) OCI_DYNAMIC_FETCH);
OCIDefineDynamic(dfnhp[0], errhp, (dvoid *) &outpos,
                      (OCICallbackDefine) cbf_get_data);
OCIDefineDynamic(dfnhp[1], errhp, (dvoid *) &outpos2,
                      (OCICallbackDefine) cbf_get_data);
OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                    (const OCISnapshot*) 0, (OCISnapshot*) 0,
                    (ub4) OCI_DEFAULT);
buf2[ 4999 ] = '\0';
printf("Select callback: Last piece: %s\n", buf2);
}

/* ------------------------------------------------------------- */
/* Fetch callback to specify buffers.                            */
/* ------------------------------------------------------------- */
STATICF sb4 cbf_get_data(ctxp, dfnhp, iter, bufpp, alenpp, piecep,
                                 indpp, rcpp)
dvoid *ctxp;
OCIDefine *dfnhp;
ub4 iter;
dvoid **bufpp;
ub4 **alenpp;
ub1 *piecep;
dvoid **indpp;
ub2 **rcpp;
{
  static int a = 0;
  ub4  outpos = *((ub4 *)ctxp);
  len = sizeof(buf1);
  len2 = 5000;

      switch(outpos)
    {
      case 1:
    *bufpp = (dvoid *) &buf1;
    *alenpp = &len;
    break;
      case 2:
    a ++;
    *bufpp = (dvoid *) buf2;
    *alenpp = &len2;
    break;
      default:
    *bufpp = (dvoid *) 0;
```

```
    *alenpp = (ub4 *) 0;
    DISCARD printf("ERROR: invalid position number: %d\n", pos);
  }

  *indpp = (dvoid *) 0;
  *rcpp = (ub2 *) 0;

  if (outpos == 1)
    *piecep = (ub1)OCI_ONE_PIECE;
  if (outpos == 2)
  {
    out2[len2] = '\0';
    if (a<=1)
    {
      *piecep = OCI_FIRST_PIECE;
        printf("Select callback: 0th piece\n");
    }
    else if (a<3)
    {
      *piecep = OCI_NEXT_PIECE;
      printf("Select callback: %d'th piece: %s\n", a-1, out2);
    }
    else {
      *piecep = OCI_LAST_PIECE;
      printf("Select callback: %d'th piece: %s\n", a-1, out2);
      a = 0;
    }
  }

  return OCI_CONTINUE;
}
```

LONG から LOB への変換例 8: 配列のフェッチ

配列のフェッチを使用する場合の、LONG から LOB への変換の例を次に示します。

```
word   i;
word arrbuf1[5] = 0;
text   arrbuf2[5][5000];

text *selstmt = (text *) "SELECT CLIP_ID, STORY FROM MULTIMEDIA_TAB
                    ORDER BY CLIP_ID";
OCIStmtPrepare(stmthp, errhp, selstmt, (ub4)strlen((char *)selstmt),
                    (ub4) OCI_NTV_SYNTAX, (ub4) OCI_DEFAULT);
retval = OCIStmtExecute(svchp, stmthp, errhp, (ub4) 0, (ub4) 0,
                    (const OCISnapshot*) 0, (OCISnapshot*) 0,
```

```
                    (ub4) OCI_DEFAULT);

while (retval == OCI_SUCCESS || retval == OCI_SUCCESS_WITH_INFO)
{
    OCIDefineByPos(stmthp, &defhp1, errhp, (ub4) 1,
                    (dvoid *) &arrbuf1[0], (sb4) sizeof(arrbuf1[0]),
                    (ub2) SQLT_INT, (dvoid *) 0,
                    (ub2 *) 0,(ub2 *) 0, (ub4) OCI_DEFAULT);
    OCIDefineByPos(stmthp, &defhp2, errhp, (ub4) 2,
                    (dvoid *) arrbuf2[0], (sb4) sizeof(arrbuf2[0]),
                    (ub2) SQLT_CHR, (dvoid *) 0,
                    (ub2 *) 0, (ub2 *) 0, (ub4) OCI_DEFAULT);
    OCIDefineArrayOfStruct(dfnhp[0], ERRH, sizeof(arrbuf1[0]), indsk,
                     rlsk, rcsk);
    OCIDefineArrayOfStruct(dfnhp[1], ERRH, sizeof(arrbuf2[0]), indsk,
                     rlsk, rcsk);

    retval = OCIStmtFetch(stmthp, errhp, (ub4) 5,
                     (ub4) OCI_FETCH_NEXT, (ub4) OCI_DEFAULT);
    if (retval == OCI_SUCCESS || retval == OCI_SUCCESS_WITH_INFO)
    {
        DISCARD printf("%d, %s\n", arrbuf1[0], arrbuf2[0]);
        DISCARD printf("%d, %s\n", arrbuf1[1], arrbuf2[1]);
        DISCARD printf("%d, %s\n", arrbuf1[2], arrbuf2[2]);
        DISCARD printf("%d, %s\n", arrbuf1[3], arrbuf2[3]);
        DISCARD printf("%d, %s\n", arrbuf1[4], arrbuf2[4]);

    }
}
```

LONG から LOB への変換例 9: INSERT、UPDATE および SELECT での PL/SQL の使用

LOB に対する INSERT 文および UPDATE 文の使用方法は、LONG に対する使用方法と同じです。次に例を示します。

```
BEGIN
  INSERT INTO Multimedia_tab VALUES (1, 'A wonderful story', NULL, EMPTY_BLOB,
      EMPTY_BLOB(), NULL, NULL, NULL, NULL, NULL);
  UPDATE Multimedia_tab SET Story = 'A wonderful story';
END;
```

LONG-to-LOB API を使用すると、SELECT 文によって LOB 列に文字変数をバインドできます。

```
BEGIN
story_buffer VARCHAR2(100);
/* This will get the LOB column if it is upto 100 bytes, otherwise it will
raise an exception */
SELECT Story INTO story_buffer FROM Multimedia_tab WHERE Clip_ID = 1;
END;
```

LONG から LOB への変換例 10: PL/SQL での割当ておよびパラメータの受渡し

LONG-to-LOB API を使用すると、パラメータの受渡しを含む、LOB の VARCHAR2、RAW などへの暗黙的な割当てが可能です。次に例を示します。

```
CREATE TABLE t (clob_col CLOB, blob_col BLOB);
INSERT INTO t VALUES('abcdefg', 'aaaaaa');
DECLARE
    var_buf VARCHAR2(100);
    clob_buf CLOB;
    raw_buf RAW(100);
    blob_buf BLOB;
BEGIN
    SELECT * INTO clob_buf, blob_buf FROM t;
    var_buf := clob_buf;
    clob_buf:= var_buf;
    raw_buf := blob_buf;
    blob_buf := raw_buf;
  END;

CREATE PROCEDURE FOO ( a IN OUT CLOB) IS......

CREATE PROCEDURE BAR (b IN OUT VARCHAR2) IS .....
DECLARE
    a VARCHAR2(100) := '1234567';
    b CLOB;
BEGIN
    FOO(a);
    SELECT clob_col INTO b FROM t;
    BAR(b);
END;
```

LONG から LOB への変換例 11: PL/SQL 組込みファンクションでの CLOB の使用

PL/SQL 組込みファンクションで CLOB を使用した場合の、LONG から LOB への変換の例を次に示します。

```
DECLARE
   myStory CLOB;
   revisedStory CLOB;
   myGist VARCHAR2(100):= 'This is my gist.';
   revisedGist VARCHAR2(100);
BEGIN
   -- select a CLOB column into a CLOB variable
   SELECT Story INTO myStory FROM Multimedia_tab WHERE clip_id=10;

   -- perform VARCHAR2 operations on a CLOB variable
   revisedStory := UPPER(SUBSTR(myStory, 100, 1));

   -- revisedStory is a temporary LOB
   -- Concat a VARCHAR2 at the end of a CLOB
   revisedStory := revisedStory || myGist;
   -- The following statement will raise an error since myStory is
   -- longer than 100 bytes
   myGist := myStory;
END;
```

LONG から LOB への変換例 12: OCI からの PL/SQL バインドの LOB での使用

LONG-to-LOB API を使用すると、OCI からの PL/SQL バインドを、LOB に対して次のように使用できます。

OCI から PL/SQL プロシージャをコールして、インバインドまたはアウトバインド（あるいはその両方）を実行すると、PL/SQL プロシージャの仮パラメータが SQLT_CLOB である場合、変数を SQLT_CHR としてバインドできます。

> **注意：** C プロシージャは PL/SQL スタブにラップされています。そのため、OCI アプリケーションは常に PL/SQL スタブを起動します。

OCI をコールするプログラムでは、次のような場合が考えられます。

「begin foo(:1); end;」形式を使用した PL/SQL アウトバインドのコール

次に例を示します。

```
text *sqlstmt = (text *)"BEGIN PKG1.P5 (:c); END; " ;
```

「call foo(:1);」形式を使用した PL/SQL アウトバインドのコール

次に例を示します。

```
text *sqlstmt = (text *)"CALL PKG1.P5( :c );" ;
```

どちらの例でも、プログラムの後半は次のようになります。

```
   OCIStmtPrepare(stmthp, errhp, sqlstmt, (ub4)strlen((char *)sqlstmt),
            (ub4) OCI_NTV_SYNTAX, (ub4) OCI_DEFAULT);
   curlen = 0;

   OCIBindByName(stmthp, &bndhp[3], errhp,
            (text *) ":c4", (sb4) strlen((char *) ":c"),
            (dvoid *) buf5, (sb4) LONGLEN, SQLT_CHR,
            (dvoid *) 0, (ub2 *) 0, (ub2 *) 0,
            (ub4) 1, (ub4 *) &curlen, (ub4) OCI_DATA_AT_EXEC);

    OCIStmtExecute(svchp, stmthp, errhp,(ub4) 0,(ub4) 0, (const OCISnapshot*) 0,
    (OCISnapshot*) 0,(ub4) OCI_DEFAULT);
```

PL/SQL プロシージャ PKG1.P5 を次に示します。

```
   CREATE OR REPLACE PACKAGE BODY pkg1 AS
     ...
   procedure p5 (c OUT CLOB) is
     -- This might have been table%rowtype (so it   is CLOB now)
   BEGIN
     ...
   END p5;

END pkg1;
```

LONG から LOB への変換例 13: PL/SQL からの PL/SQL プロシージャおよび C プロシージャのコール

PL/SQL プロシージャまたはファンクションには、CLOB または VARCHAR2 を仮パラメータとして指定できます。たとえば、PL/SQL プロシージャは次のいずれかのように指定できます。

- **仮パラメータが CLOB の場合 :**

```
CREATE OR REPLACE PROCEDURE get_lob(table_name IN VARCHAR2, lob INOUT
CLOB) AS
   ...
BEGIN
  ...
END;
/
```

- **仮パラメータが VARCHAR2 の場合 :**

```
CREATE OR REPLACE PROCEDURE get_lob(table_name IN VARCHAR2, lob INOUT
VARCHAR2) AS
   ...
BEGIN
  ...
END;
/
```

コールするファンクションは次のいずれかのタイプです。

- **実パラメータが CHR の場合 :**

```
create procedure ...
declare
c VARCHAR2[200];
begin
  get_lob('table_name', c);
end;
```

- **実パラメータが CLOB の場合 :**

```
create procedure ...
declare
c table_name.column_name%type   -- This is a CLOB now
begin
  get_lob('table_name', c);
end;
```

PL/SQL のどちらのスタブも、両方の例の実パラメータで機能します。

LONG-to-LOB API と対応付けられた新機能の概要

OCI 関数

OCIDefineByPos() 関数に、次の型を指定できるようになりました。

- CLOB 列 : SQLT_CHR および SQLT_LNG
- BLOB 列 : SQLT_BIN および SQLT_LBI

これによって、LOB 列に対して VARCHAR2 バッファを定義した後、OCIStmtFetch() コマンドを使用して CLOB または BLOB データを格納したバッファを取得できます。

OCIBindByPos() 関数および OCIBindByName() 関数は、最大 4GB のバッファを処理できるようになりました。

SQL 文

次の構文が新しく追加されました。

```
ALTER TABLE [<schema>.]<table_name>
MODIFY ( <long_column_name> { CLOB | BLOB | NCLOB } [DEFAULT <default_value> )
[LOB_storage_clause];
```

詳細は、8-7 ページの「LONG から LOB への移行 : ALTER TABLE を使用した LONG 列の LOB 型への変更」を参照してください。ALTER TABLE 構文の変更点を次に示します。

- 次のとおりデータ型を指定できます。
  - **CLOB/NCLOB**: 「long_column_name」の元のデータ型が **LONG** であった場合
  - **BLOB**: 元のデータ型が **LONG RAW** であった場合
- この LONG から LOB への変換中のみ、**LOB_storage_clause** 句を MODIFY 句に指定できます。
- LONG から LOB への変換中に指定できるのは、[DEFAULT "expr"] のみです。この操作で実行できるのは、前述の ALTER TABLE 操作のみです。

PL/SQL インタフェース

次の PL/SQL の SELECT 文を使用できるようになりました。

- CLOB 列に対して SELECT を実行し、CHAR、LONG、VARCHAR2 などの文字バッファ変数に入れる SELECT INTO 文
- BLOB 列に対して SELECT を実行し、RAW や LONG RAW などのバイナリ・バッファ変数に入れる SELECT INTO 文

次のような割当てを行うこともできます。

- CLOB（BLOB）の VARCHAR2（RAW）変数への割当て
- VARCHAR2（RAW）変数の CLOB（BLOB）への割当て

さらに、CLOB（BLOB）を実パラメータとして、仮パラメータに VARCHAR2（RAW）を持つファンクションに渡すことができます。その逆も可能です。また、LOB に対して PL/SQL 組込みファンクションをコールすることもできます。

参照： 7-33 ページの「LOB に対する SQL セマンティクスのサポート」を参照してください。

互換性および移行

ALTER TABLE を使用して LONG 列を LOB に変更すると、その表には元から LONG 列ではなく LOB 列があったように見えます。すべての LONG データを LOB に移動すると、表に ALTER を実行して LONG に戻すことはできません。

表 8-1 に、今回のリリースおよび以前のリリースでの様々なクライアント / サーバーの組合せによる動作の概要を示します。

表 8-1 Oracle9*i* および Oracle9*i* より前のクライアント / サーバーの組合せ

クライアント	Oracle9*i* リリース 9.0.x のサーバー	Oracle9*i* より前のサーバー
CHAR を使用したリリース 9.0.x	データを送信する	クライアントにエラーが発生する
LOB を使用したリリース 9.0.x	ロケータを送信する	ロケータを送信する
CHAR を使用した 9.0.x より前のリリース	クライアントにエラーが発生する	クライアントにエラーが発生する
LOB を使用した 9.0.x より前のリリース	ロケータを送信する	ロケータを送信する

パフォーマンス

INSERT およびフェッチの優れたパフォーマンス

LONG-to-LOB API を使用した LOB のピース単位の INSERT またはフェッチは、`OCILobRead()` や `OCILobWrite()` などの既存の関数を使用したピース単位の INSERT およびフェッチと同等のパフォーマンスを発揮します。

Oracle では、一度の OCI コールで 4KB を超えるデータを LOB に挿入できるため、サーバーへのラウンドトリップを削減できます。

また、最初に LOB ロケータをフェッチして `OCILobRead()` を実行するかわりに、一度の `OCIStmtFetch()` コールで LOB データを読み取ることができるようになりました。これによって、LOB データを最初から読み取る場合、（`OCIStmtFetch()` はオフセット 1 からデータを戻すため）パフォーマンスが向上します。このような理由から、LONG-to-LOB API を使用すると LOB の INSERT およびフェッチのパフォーマンスが向上します。

PL/SQL

VARCHAR2 バッファを LOB 変数に割り当てる場合、一時 LOB を作成する必要があるため、VARCHAR2 を LONG 変数に割り当てる場合よりもパフォーマンスが低下します。その結果、アプリケーションのパフォーマンスが暗黙的に低下します。

FAQ: LONG から LOB への移行

LOB から LONG への移動

質問

表に対して ALTER を実行して LONG 列から LOB に変更し、すべての LONG データを LOB に移動させると、その列に ALTER を実行して LONG に戻すことができません。解決策はありますか？

回答

あります。LONG 列を追加し、OCI アプリケーションを使用して LOB 列からデータを読み込み、LONG 列に挿入します。その後、LOB 列を削除します。

CREATE VIEW の必要性

質問

LONG から LOB に移行した場合でも、CREATE VIEW は必要ですか？

回答

移行後は CREATE VIEW を使用する必要はありません。ALTER TABLE 文を使用します。

OCI LOB ルーチンの廃止

質問

`OCIStmtFetch()` は、LOB 列でどのように機能しますか ?LONG 列の場合と同様に OCI_NEED_DATA を戻しますか？また、他の列のデータを使用する前に、LOB データを完全にフェッチする必要がありますか ?`OCILobRead()` や `OCILobWrite()` などの LOB 用の OCI ルーチンは廃止されますか？

回答

`OCIStmtFetch()` は、データ型が定義に SQLT_LNG、SQLT_CHR などのように指定されている場合は、以前の LONG に対する場合と同様に LOB に対して機能します。データ型が SQLT_CLOB または SQLT_BLOB として指定されている場合、`OCIStmtFetch()` コールで LOB ロケータをフェッチし、`OCILobRead()` をコールして LOB データを読み込むことができます。LOB 用の OCI コールは廃止しません。

LOB 列のデータ型が SQLT_LNG または SQLT_CHR である場合、他のデータを使用する前に LOB データを完全にフェッチする必要があります。SQL*Plus でこの問題を回避するには、既存の LOB 用の OCI インタフェースを継続して使用してください。

PL/SQL に関する問題

質問

LOB 列（サイズが 32KB を超える）をフェッチして PL/SQL の CHAR、RAW、LONG または LONG RAW 型のバッファに格納すると、例外が発生しますか？

回答

`OCIDefineByPos()` および PL/SQL の `SELECT INTO` 文では、必要な読込み量を指定できません。指定できるのはバッファ長のみです。LOB のサイズにかかわらず、バッファをオーバーフローすることなく、正確な量がフェッチされます。列全体をフェッチしない場合、OCI では切捨てエラーが戻され、PL/SQL では例外が発生します。

この動作は LONG および VARCHAR2 に対する既存の動作と一貫性があります。

32KB 未満のイメージ全体の取出し

質問

最初にロケータを取り出さなくても、LOB データを SELECT できるようになりました。PL/SQL では、イメージが 32KB 未満の場合、一度の SELECT でイメージ全体を取り出せますか？

回答

取り出せます。

LONG および LOB でのトリガー

質問

トリガーには、LOB ではサポートされていないが LONG ではサポートされている機能がいくつかあります。これによって問題が発生しますか？

回答

LOB でトリガーを使用するには 2 つの制限事項があります。詳細は、8-10 ページの「LONG から LOB への移行に関する制限事項」を参照してください。

9

LOB: 効果的な使用方法

この章の内容は次のとおりです。

SQL*Loader の使用

SQL*Loader を使用して、LOB をバルク・ロードできます。

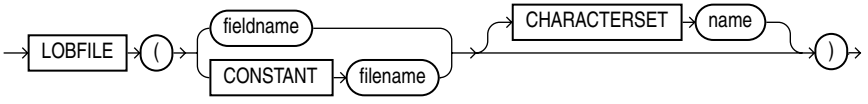
参照：

- SQL*Loader の使用例および詳細は、第 4 章「LOB の管理」の「LOB ロード時の SQL*Loader の使用」を参照してください。
- オブジェクト、LOB、コレクションをロードするための SQL*Loader の使用の詳細は、『Oracle9*i* データベース・ユーティリティ』を参照してください。

SQL*Loader による XML 文書の LOB へのロード

LOB は非常に大きくなる場合があるため、SQL*Loader は、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インライン）または LOBFILE のいずれかからロードできます。図 9-1 に、LOBFILE 構文を示します。

図 9-1 LOBFILE 構文



LOB データは非常に大きくなる場合があるため、LOBFILE からロードする必要があります。LOBFILE でも、LOB データ・インスタンスは（事前にサイズが決まっており、デリミタ付きで、Length-Value である）フィールドに存在するとみなされます。ただし、これらのフィールドはレコードに編成されていません（レコードという概念は LOBFILE 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。

LOBFILE からの LOB は、メモリーに収まる必要はありません。SQL*Loader は、64KB のチャンクで LOBFILE を読み込みます。64KB を超える物理レコードをロードするには、READSIZE パラメータを使用して、さらに大きなサイズを指定できます。

LOBFILE を使用して、XMLType 列または CLOB に XML データを含む列をロードする方法が最適です。

- **XML が有効な場合**
  LOBFILE の XML データが大きく、そのデータが有効な XML であるとわかっている場合は、ダイレクト・パス・ロードを使用してください。これによって、すべての XML 検証処理を回避できます。
- **XML に妥当性チェックが必要な場合**
  XML データの妥当性チェックが不可欠な場合、従来型パス・ロードを使用してください（ダイレクト・パス・ロードほど効果的でないことに注意してください）。

従来型パス・ロードは、SQL INSERT 文を実行して、表を Oracle データベースに移入します。ダイレクト・パス・ロードは、Oracle データ・ブロックをフォーマットし、データ・ブロックを直接データベース・ファイルに書き込むことによって、Oracle データベースのオーバーヘッドの大半を排除します。

ダイレクト・パス・ロードでは、データベース・リソースに関して他のユーザーと競合することがないため、通常、ほぼディスク・スピードでデータをロードできます。制限事項、セキュリティおよびバックアップなどを含む、ダイレクト・パス・ロード固有の考慮事項については、『Oracle9*i* データベース・ユーティリティ』の第 9 章を参照してください。

図 9-2 に、SQL*Loader のダイレクト・パス・ロードおよび従来型パス・ロードを示します。

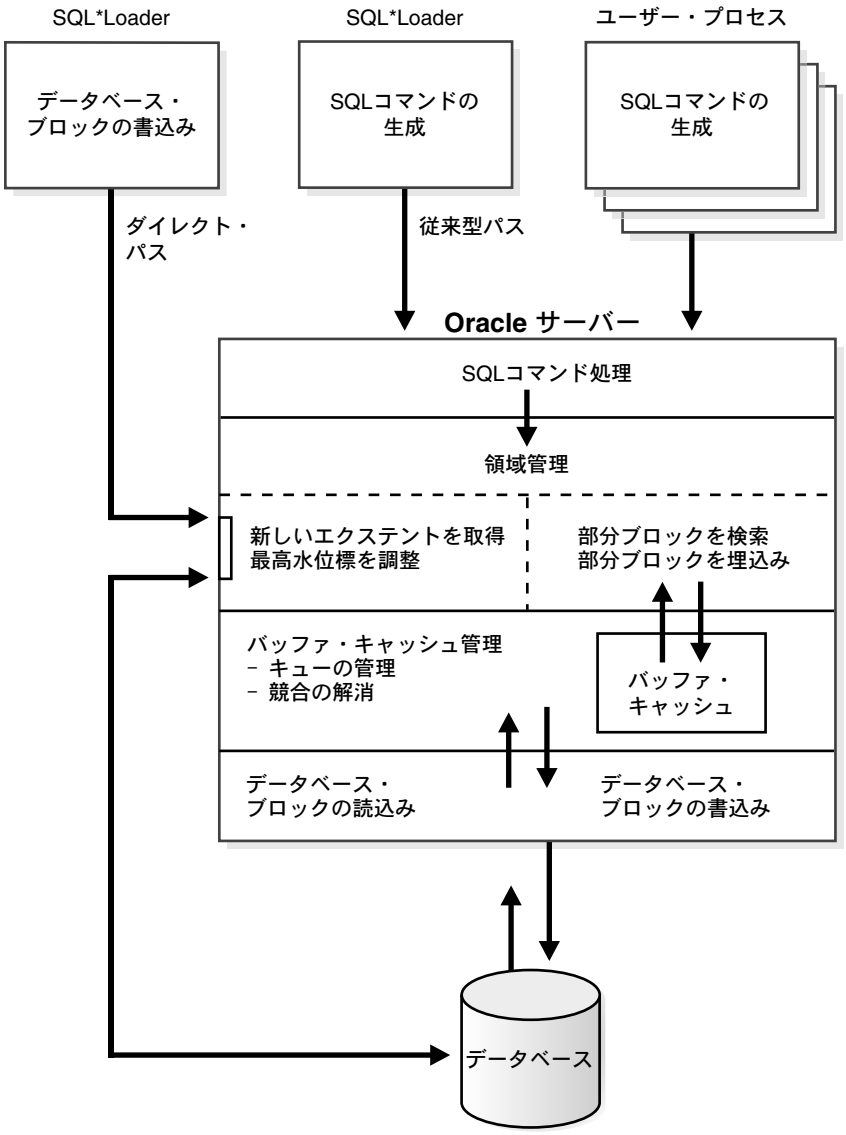
ロードされる表は、事前にデータベースに存在している必要があります。SQL*Loader が表を作成することはありません。SQL*Loader は、すでにデータを含んでいるか、または空である既存の表をロードします。

ロードには次の権限が必要です。

- ロードされる表に関する INSERT 権限
- ロードされる表に関する DELETE 権限（同じ場所に新規のデータをロードする前に、REPLACE または TRUNCATE オプションを使用して、表の以前のデータを空にする場合）

> **参照：** オブジェクト、LOB、コレクションのロードの詳細および SQL*Loader の事例は、『Oracle9*i* データベース・ユーティリティ』を参照してください。

図 9-2 SQL*Loader: ダイレクト・パス・ロードおよび従来型パス・ロード



LOB パフォーマンスのガイドライン

一般的なパフォーマンスに関するガイドライン

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。

- できるかぎり、一度に大きいデータ・チャンクの読込みおよび書込みを行います。LOB はサイズが大きいため、LOB 値の大きいデータ・チャンクを一度に読込みおよび書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有効です。

  a. クライアント側から LOB にアクセスし、クライアントがサーバーと異なるノードにある場合、大量に読込みおよび書込みを行うことでネットワーク・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、少量の読込みおよび書込みを行うたびに I/O が発生します。大量に読込みおよび書込みを行うことで I/O が削減されます。

  c. LOB に書込みを行うと、新しいバージョンの LOB CHUNK が作成されます。そのため、一度の書込み量が少ないと、書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、小さいデータ・チャンクの読込みおよび書込みを行います。クライアント側の小さい LOB データの読込みおよび書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 データの読込みおよび書込みを行う前に、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、5-18 ページの「LOB バッファリング・サブシステム（LBS）」を参照してください。

- コールバックとともに **`OCILobRead()`** および **`OCILobWrite()`** を使用します。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、amount 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。できるかぎり、LOB チャンクの倍数で読込みおよび書込みを行います。

- **LOB** に対してチェックアウト / チェックイン・モデルを使用します。LOB は次の操作に対して最適化されます。
  - LOB 値全体を置換する SQL の `UPDATE`。
  - 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。
- 頻繁に変更をコミットします。

> **参照：** LOB で SQL セマンティクスを使用する場合のパフォーマンスの問題については、7-52 ページの「LOB で SQL セマンティクスを使用する場合のパフォーマンス属性」を参照してください。

パフォーマンスの数値

表 9-1 に、ペイロードの LOB 部分にチャンク・サイズ 8KB を使用して 500 個のメッセージをエンキューしたパフォーマンス・テストの結果を示します。このパフォーマンス・テストでは、Oracle8*i* リリース 8.1.7 を使用しています。DB_BLOCKSIZE = 8192（8KB）は、OS のブロック・サイズと同じです。

表 9-1 CACHE および LOGGING を指定した状態と指定しない状態で、500 個のメッセージをエンキューした場合の応答時間

チャンク・サイズ	CACHE（あり / なし）	LOGGING（あり / なし）	MESSAGE_SIZE	応答時間
8KB	NOCACHE	NOLOGGING	3,900 バイト	01:33 秒
8KB	NOCACHE	NOLOGGING	4,000 バイト	06:19 秒
8KB	NOCACHE	NOLOGGING	4,000 バイト	04:36 秒
8KB	CACHE		3,900 バイト	01:22 秒
8KB	CACHE		4,000 バイト	01:22 秒
8KB	CACHE		4,000 バイト	01:83 秒
8KB	CACHE		20,000 バイト	02:33 秒

4,000 バイトのメッセージに対して、チャンク・サイズ 16KB を使用して、NOCACHE および NOLOGGING を指定した場合の、以前の応答時間は 12:28 秒でした。

これらの結果は、CACHE パラメータによって大幅にパフォーマンスが向上したことを示しています。

一時 LOB パフォーマンスのガイドライン

前述の一般的な LOB パフォーマンスに関するガイドライン「LOB パフォーマンスのガイドライン」に加えて、一時 LOB の使用に対するガイドラインを次に示します。

- **デフォルトのシステム表領域のかわりに、一時 LOB 記憶域に別の一時表領域を使用します。**

  これによって、データを永続 LOB から一時 LOB にコピーする場合のデバイスの競合を回避します。

  アプリケーションで、新しく提供された拡張 SQL セマンティクス機能を使用する場合、以前よりさらに多くの一時 LOB が SQL および PL/SQL で暗黙的に作成されます。これらの一時 LOB を格納する**一時表領域**が、アプリケーションに対して十分な大きさであることを確認してください。特に、これらの一時 LOB が暗黙的に作成されるのは、次の事項を使用または実行する場合です。

  - LOB に対する SQL ファンクション
  - LOB に対する PL/SQL 組込み文字ファンクション
  - VARCHAR2 から CLOB、RAW から BLOB への変数割当て
  - LONG から LOB への移行

- **PL/SQL では、NOCOPY を使用して、できるかぎり参照によって一時 LOB パラメータを渡します。**

  参照によるパラメータの受渡しおよびパラメータのエイリアシングの詳細は、『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。

- **PL/SQL プロシージャ・ループで一時 LOB を使用します。**

  PL/SQL プロシージャをループで使用して、繰返し一時 LOB を作成する場合、ローカル変数のかわりに、PL/SQL パッケージ LOB ロケータ変数をプロシージャ内で使用すると、パフォーマンスが向上します。

  これは、セッション中持続するパッケージ変数を使用することによって、プロシージャ・コールごとに一時 LOB を管理するための余分なメモリーを割り当てる必要がなくなるためです。

注意： セッション・ロケータを使用して作成された一時 LOB は、ファンクション・コールまたはプロシージャ・コールの最後に自動的にクリーンアップされません。DBMS_LOB.FREETEMPORARY() をコールして、一時 LOB を明示的に解放する必要があります。

```
CREATE OR REPLACE PACKAGE pk is
  tmplob clob;
END pk;
/
CREATE OR REPLACE PROCEDURE temp_lob_proc
IS
BEGIN
  -- instead of using a local LOB variabe, use a package variable here
  DBMS_LOB.CREATETEMPORARY(pk.tmplob, TRUE);
  -- Do some LOB data maniputation here
  DBMS_LOB.FREETEMPORARY(pk.tmplob);
END;
/
DECLARE
  doc CLOB;
BEGIN
  FOR i IN 1..400 LOOP
    temp_lob_proc();
  END LOOP;
END;
/
```

- **一時 LOB に対してバッファ・キャッシュを利用します。**

  CACHE パラメータを TRUE に設定して作成した一時 LOB は、バッファ・キャッシュを介して移動できます。CACHE パラメータを設定しなかった場合、一時 LOB の読込みおよび書込みはディスクから直接行われます。

- **一時 LOB 変数を割り当てると、コストがかかることに注意してください。**

  一時 LOB は、割当て時に新しく自身のコピーを作成します。次に例を示します。

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

  このコードによって、LOCATOR1 が指す一時 LOB のコピーが作成されます。一時 LOB パラメータをプロシージャまたはファンクションに渡す場合、PL/SQL の参照セマンティクスによる受渡しを考慮する必要がある場合もあります。

OCI では、LOB ロケータおよびデータのコピー・セマンティクスを確認するために、OCILobLocatorAssign を使用して LOB データおよび一時 LOB ロケータをコピーします。ディープ・コピーを回避するために、ロケータ・コピーのコピー・セマンティクスを必要としない場合は、ポインタを割り当てることができます。次に例を示します。

```
OCILOBLocator *LOC1;
OCILOBLocator *LOC2;
OCILOBCREATETEMPORARY (LOC1,TRUE,OCIDURATIONSESSION);
LOC2 = LOC1;
```

- **一時 LOB に対して OCI_OBJECT モードを使用します。**

  LOB 割当てでの一時 LOB のパフォーマンスを向上させるには、OCILobLocatorAssign に対して OCI_OBJECT モードを使用します。OCI_OBJECT モードの場合、Oracle はディープ・コピー回数を最小限にしようとします。そのため、OCI_OBJECT モードでソース一時 LOB に対して OCILobLocatorAssign が実行された後、ソース・ロケータおよび宛先ロケータは、いずれかの LOB ロケータにより変更されるまで、同一 LOB を指すようになります。

- **SQL 問合せおよび PL/SQL プログラムから戻された一時 LOB を解放します。**

  PL/SQL、C（OCI）、Java および他のプログラム・インタフェースでは、SQL 問合せの結果または PL/SQL プログラムの実行によって、LOB のオペレーション・コールおよびファンクション・コールに対して一時 LOB が戻されます。次に例を示します。

  ```
  SELECT substr(CLOB_Column, 4001, 32000) FROM ...
  ```

  PL/SQL で問合せが実行される場合、戻された一時 LOB は、PL/SQL プログラム・ブロックの終了時に自動的に解放されます。任意の時点で、明示的に一時 LOB を解放することもできます。OCI および Java では、戻された一時 LOB をユーザーが明示的に解放する必要があります。

  SQL 問合せから戻された一時 LOB の割当てを適切に解除しないと、一時表領域が一杯となり、パフォーマンスが低下します。

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 を使用して挿入します。
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. コミットします。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

注意：

- 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、手順 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薦めします。

1. 空の LOB を挿入し、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. コミットします。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。

参照： 第 14 章「LOB の事例」を参照してください。

LONG から LOB への移行

LONG から LOB への移行中、表に対する REDO がログに記録されるのは、表に LOGGING が指定されている場合のみです。また、LONG から LOB へ変換中の列に対する REDO がログに記録されるのは、LOB の記憶特性が LOGGING に指定されている場合のみです。LOB の LOGGING | NOLOGGING のデフォルト値は、LOB が作成されている表領域から継承されます。

移行中の REDO 領域作成の回避

移行中の REDO 領域の生成を回避して、正常に移行するには、次の手順に従います。

1. 次の文を実行します。

```
ALTER TABLE Long_tab NOLOGGING;
```

2. 次の文を実行します。

```
ALTER TABLE Long_tab MODIFY ( long_col CLOB [default <default_val>])   LOB
(long_col) STORE AS (... NOLOGGING ...);
```

3. 次の文を実行します。

```
ALTER TABLE Long_tab MODIFY LOB long_col STORE AS (...LOGGING...);
```

4. 次の文を実行します。

```
ALTER TABLE Long_tab LOGGING;
```

5. 表および LOB を含む表領域のバックアップを取ります。

参照： 第 8 章「LONG から LOB への移行」を参照してください。

10

内部永続 LOB

利用モデル

この章では、LOB に対する操作（LOB へのデータの書込みなど）を、利用方法に沿って説明します。表 10-1「利用モデル : 内部永続 LOB」に、すべての利用方法を示します。

個々の利用方法

内部永続 LOB の個々の利用方法を、次のように説明します。

- 利用図 : 利用方法を表す UML 図。図の見方については、付録 A「Unified Modeling Language 図」を参照してください。
- 用途 : LOB に関するこの利用方法の用途。
- 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 関連操作の実行に使用する主な構文。
- 使用例 : 利用方法の実装例について、サンプル・スキーマの点から説明した例。サンプル・スキーマに関する詳細は、『Oracle9*i* サンプル・スキーマ』を参照してください。
- 例 : 使用するサンプル・スキーマに基づく各利用方法の適用法。

利用モデル : 内部永続 LOB

表 10-1 の「**+**」は、特定の利用方法で、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 がその利用方法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 を使用した C
- **CP** − OCCI 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 を使用した Visual Basic
- **J** − JDBC を使用した Java

表 10-1 利用モデル : 内部永続 LOB

LOB の利用方法およびページ	P	O	CP	B	C	V	J
10-183 ページの「他の LOB への LOB の追加」	+	+	-	+	+	+	+
10-191 ページの「Java（JDBC）: 他の LOB への LOB の追加」	+	+	-	+	+	-	+
10-179 ページの「キャラクタ・セットフォーム : キャラクタ・セット・フォームの取得」	-	+	-	-	-	-	-
10-176 ページの「キャラクタ・セット ID: キャラクタ・セット ID の取得」	-	+	-	-	-	-	-
10-78 ページの「LOB のチェックイン」	+	+	-	+	+	+	+
10-67 ページの「LOB のチェックアウト」	+	+	-	+	+	+	+
LOB のクローズについては、第 3 章「様々なプログラム環境での LOB のサポート」を参照	-	-	-	-	-	-	-
10-122 ページの「2 つの LOB の全体または一部の比較」	+	-	-	+	+	+	+
10-156 ページの「LOB ロケータのコピー」	+	+	-	+	+	+	+
10-145 ページの「他の LOB への LOB の全体または一部のコピー」	+	+	-	+	+	+	+
10-13 ページの「LOB を含むネストした表の作成」	S	S	-	S	S	S	S
10-10 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	S

表 10-1 利用モデル : 内部永続 LOB （続き）

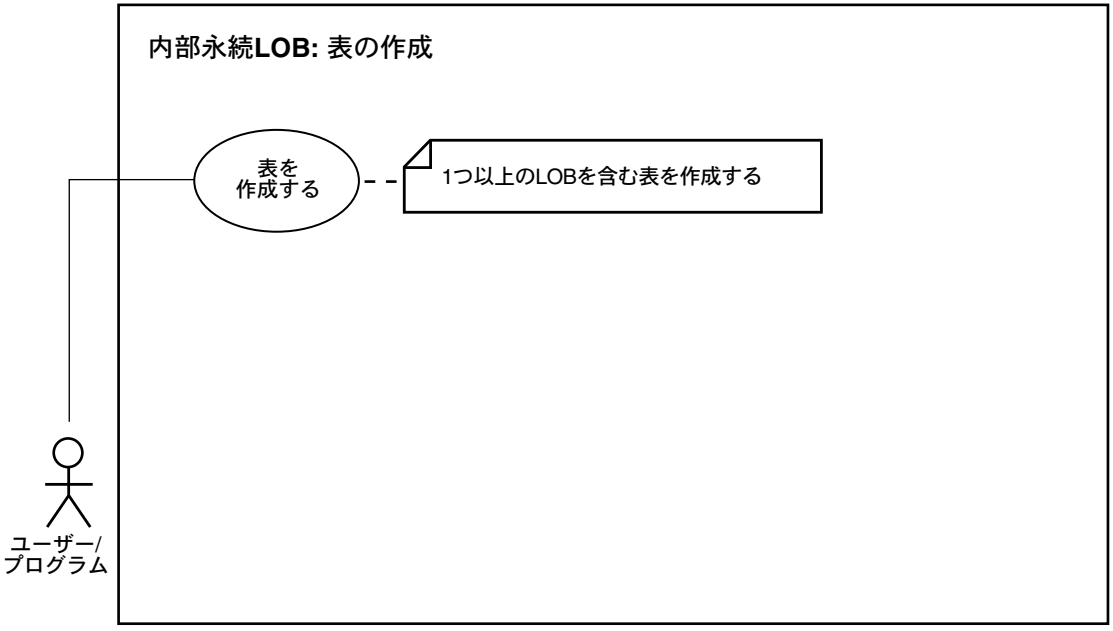
LOB の利用方法およびページ	P	O	CP	B	C	V	J
10-5 ページの「1 つ以上の LOB 列を含む表の作成」	S	S	S	S	S	S	S
LOB への参照を含む VARRAY の作成については、5-29 ページの「LOB への参照を含む VARRAY の作成」を参照	S	S	S	S	S	S	S
10-274 ページの「LOB を含む表の行の削除」	S	S	S	S	S	S	S
10-252 ページの「LOB バッファリングの使用禁止化」	-	+	-	+	+	+	-
10-92 ページの「LOB データの表示」	+	+	-	+	+	+	+
10-240 ページの「LOB バッファリングの使用可能化」	-	-	-	+	+	+	-
10-164 ページの「2 つの LOB ロケータが等しいかどうかの確認」	-	+	-	-	+	-	+
10-230 ページの「LOB の一部の消去」	+	+	-	+	+	+	+
10-246 ページの「バッファのフラッシュ」	-	+	-	+	+	-	-
10-171 ページの「ロケータの初期化 : LOB ロケータが初期化されているかどうかの確認」	-	+	-	-	+	-	-
10-16 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	S	+
10-23 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	+	-	+	+	+	+
10-20 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	S
10-137 ページの「長さ : LOB の長さの取得」	+	+	-	+	+	+	+
10-32 ページの「LOB への BFILE データのロード」	+	+	-	+	+	+	+
10-42 ページの「内部永続 BLOB への BFILE バイナリ・データのロード」	+	-	-	-	-	-	-
10-46 ページの「内部永続 CLOB への BFILE データのロード」	+	-	-	-	-	-	-
10-63 ページの「TO_LOB 演算子を使用した LONG から LOB へのコピー」	S	S	S	S	S	S	S
10-60 ページの「LONG-to-LOB API を使用した LONG から LOB への移行」	+	+	-	-	-	-	-
10-191 ページの「Java（JDBC）: 他の LOB への LOB の追加」	+	+	-	+	+	-	+
LOB のオープンについては、第 3 章「様々なプログラム環境での LOB のサポート」を参照	-	-	-	-	-	-	-
10-130 ページの「パターン : LOB 内のパターンの有無の確認（instr）」	+	-	-	+	+	-	+

表 10-1 利用モデル : 内部永続 LOB （続き）

LOB の利用方法およびページ	P	O	CP	B	C	V	J
10-114 ページの「LOB の一部の読込み（substr）」	+	-	-	+	+	+	+
10-103 ページの「LOB からのデータの読込み」	+	+	-	+	+	+	+
LOB データのストリーミングについては、3-60 ページの「JDBC: 新しい LOB ストリーミング API」を参照 **注意 :** この API の利用方法は、この章には含まれていません。次回のリリースを参照してください。	-	-	-	-	-	-	+
10-218 ページの「LOB データの切捨て」	+	+	-	+	+	+	+
10-260 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	S
10-263 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	S
10-265 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	+	-	+	+	+	+
書込みの追加 : 10-193 ページの「LOB の終わりへの追加の書込み」を参照	-	-	-	-	-	-	-
10-202 ページの「LOB へのデータの書込み」	+	+	+	+	+	+	+

1 つ以上の LOB 列を含む表の作成

図 10-1 利用図 : 1 つ以上の LOB 列を含む表の作成



参照 :

- 10-2 ページの表 10-1「利用モデル : 内部永続 LOB」を参照してください。
- 10-10 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。
- 10-13 ページの「LOB を含むネストした表の作成」を参照してください。
- 5-29 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。

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() ファンクションを使用すると、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、10-16 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ネストした表の作成の詳細は、10-13 ページの「LOB を含むネストした表の作成」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、10-10 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。

LOB を含む表を作成する場合は、次に示すガイドラインおよび例を使用してください。

- 第 2 章「基本 LOB 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」

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、『Oracle9*i* 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 ファンクション
- 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:『Oracle9*i* SQL リファレンス』の第 15 章「SQL 文 : CREATE SYNONYM ～ CREATE TRIGGER」の「CREATE TABLE」

使用例

この例では、次の Oracle9*i* サンプル・スキーマを使用します。

- 人事管理（HR）
- 注文入力（OE）
- 製品メディア（PM）

HR スキーマと OE スキーマを作成してから、PM スキーマを作成する必要があります。これらのスキーマに関する詳細は、『Oracle9*i* サンプル・スキーマ』を参照してください。

例

LOB 列を含む表を作成する方法の例を、SQL で示します。

- 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。

> **注意：** SQL DDL を使用して 1 つ以上の LOB 列を含む表を直接作成できるため、DBMS_LOB パッケージを使用する必要はありません。

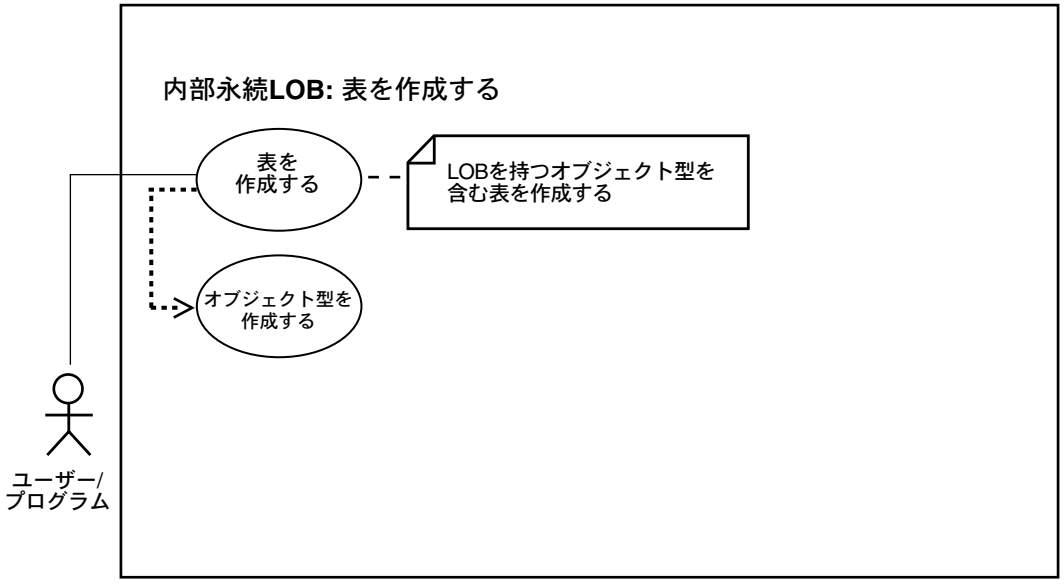
```
/*  Setup script for creating Print_media,
    Online_media and associated structures
*/

DROP USER pm CASCADE;
DROP DIRECTORY ADPHOTO_DIR;
DROP DIRECTORY ADCOMPOSITE_DIR;
DROP DIRECTORY ADGRAPHIC_DIR;
DROP INDEX onlinemedia CASCADE CONSTRAINTS;
DROP INDEX printmedia CASCADE CONSTRAINTS;
DROP TABLE online_media CASCADE CONSTRAINTS;
DROP TABLE print_media CASCADE CONSTRAINTS;
DROP TYPE textdoc_typ;
DROP TYPE textdoc_tab;
DROP TYPE adheader_typ;
DROP TABLE adheader_typ;
CREATE USER pm;
GRANT CONNECT, RESOURCE to pm;

CREATE DIRECTORY ADPHOTO_DIR AS '/tmp/';
CREATE DIRECTORY ADCOMPOSITE_DIR AS '/tmp/';
```

```
CREATE DIRECTORY ADGRAPHIC_DIR AS '/tmp/';
CREATE DIRECTORY media_dir AS '/tmp/';
GRANT READ ON DIRECTORY ADPHOTO_DIR to pm;
GRANT READ ON DIRECTORY ADCOMPOSITE_DIR to pm;
GRANT READ ON DIRECTORY ADGRAPHIC_DIR to pm;
GRANT READ ON DIRECTORY media_dir to pm;

CONNECT pm/pm  (or &pass);
COMMIT;

CREATE TABLE a_table (blob_col BLOB);

CREATE TYPE adheader_typ AS OBJECT (
   header_name    VARCHAR2(256),
   creation_date  DATE,
   header_text    VARCHAR(1024),
   logo           BLOB );

CREATE TYPE textdoc_typ AS OBJECT (
   document_typ   VARCHAR2(32),
   formatted_doc  BLOB);

CREATE TYPE Textdoc_ntab AS TABLE of textdoc_typ;

CREATE TABLE adheader_tab of adheader_typ (
Ad_finaltext DEFAULT EMPTY_CLOB(), CONSTRAINT
Take CHECK (Take IS NOT NULL),  DEFAULT NULL);

CREATE TABLE online_media
( product_id  NUMBER(6),
product_photo ORDSYS.ORDImage,
product_photo_signature ORDSYS.ORDImageSignature,
product_thumbnail ORDSYS.ORDImage,
product_video ORDSYS.ORDVideo,
product_audio ORDSYS.ORDAudio,
product_text CLOB,
product_testimonials ORDSYS.ORDDoc);

CREATE UNIQUE INDEX onlinemedia_pk
  ON online_media (product_id);

ALTER TABLE online_media
ADD (CONSTRAINT onlinemedia_pk
PRIMARY KEY (product_id), CONSTRAINT loc_c_id_fk
FOREIGN KEY (product_id)  REFERENCES oe.product_information(product_id)
);
```

```
CREATE TABLE print_media
(product_id NUMBER(6),
ad_id NUMBER(6),
ad_composite BLOB,
ad_sourcetext CLOB,
ad_finaltext CLOB,
ad_fktextn NCLOB,
ad_testdocs_ntab textdoc_tab,
ad_photo BLOB,
ad_graphic BFILE,
ad_header adheader_typ,
press_release LONG) NESTED TABLE ad_textdocs_ntab STORE AS textdocs_nestedtab;

CREATE UNIQUE INDEX printmedia_pk
  ON print_media (product_id, ad_id);

ALTER TABLE print_media
ADD (CONSTRAINT printmedia_pk
PRIMARY KEY (product_id, ad_id),
CONSTRAINT printmedia_fk FOREIGN KEY (product_id)
REFERENCES oe.product_information(product_id)
);
```

LOB 属性を持つオブジェクト型を含む表の作成

図 10-2 利用図 : LOB 属性を持つオブジェクト型を含む表の作成



参照 :

- 10-2 ページの表 10-1「利用モデル : 内部永続 LOB」を参照してください。
- 10-5 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。
- 10-13 ページの「LOB を含むネストした表の作成」を参照してください。
- 5-29 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。

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

LOB を含む表を作成する場合は、次に示すガイドラインおよび例を使用してください。

- 第 2 章「基本 LOB コンポーネント」、「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」

構文

次のマニュアルの項を参照してください。

- SQL: 『Oracle9*i* SQL リファレンス』の第 15 章「SQL 文 : CREATE SYNONYM ～ CREATE TRIGGER」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を使用する表を作成する必要があります。

この例では、Oracle9*i* サンプル・スキーマに含まれる製品メディア・スキーマを使用します。このスキーマに関する詳細は、『Oracle9*i* サンプル・スキーマ』を参照してください。

例

例を SQL で示します。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

この例で使用するヘッダーまたはタイトル、およびロゴを含む `adheader_typ` を表の基礎として作成します。

```
CREATE TYPE adheader_typ AS OBJECT (
   header_name    VARCHAR2(256),
   creation_date  DATE,
   header_text    VARCHAR(1024),
   logo           BLOB );

/* Create table adheader_tab Using SQL DDL: */
CREATE TABLE adheader_tab of adheader_typ (
   logo DEFAULT EMPTY_BLOB() CONSTRAINT Take CHECK (Take IS NOT NULL),
   creation_date DATE );
```

SQL DDL を使用して、列オブジェクトを含む adheader_typ 型を表の基礎として作成します。

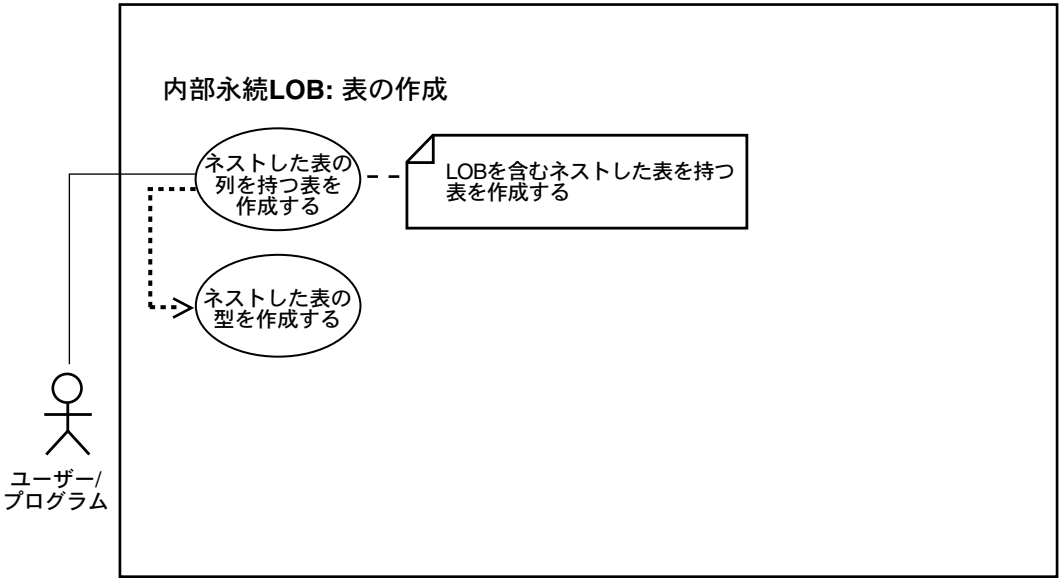
```
DROP TYPE adheader;
DROP TABLE adheader_tab;
CREATE TYPE adheader_typ AS OBJECT (
   header_name    VARCHAR2(256),
   creation_date  DATE,
   header_text    VARCHAR(1024),
   logo           BLOB );

/*  Create support table adheader_tab as an archive of
    ad headers using SQL DDL: */
CREATE TABLE adheader_tab of adheader_typ;
```

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、『Oracle9*i* 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 を含むネストした表の作成

図 10-3 利用図 : LOB を含むネストした表の作成



参照 :

- 10-2 ページの表 10-1「利用モデル : 内部永続 LOB」を参照してください。
- 10-5 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。
- 10-10 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。
- 5-29 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。

用途

LOB を含むネストした表を作成します。

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 LOB 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」

構文

次のマニュアルの項を参照してください。

- SQL:『Oracle9*i* SQL リファレンス』の第 15 章「SQL 文 : CREATE SYNONYM ～ CREATE TRIGGER」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ネストした表を作成します。この例の `Print_media` 表には、`textdoc_tab` 型を持つネストした表 `ad_textdoc_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。

- BFILE - 広告用のグラフィック
- CLOB - 広告用の台本

LOB 列を伴う表の作成方法は、前項で説明したとおりです（10-5 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。

例

「SQL: LOB を含むネストした表の作成」では、SQL プログラム環境での例を示します。

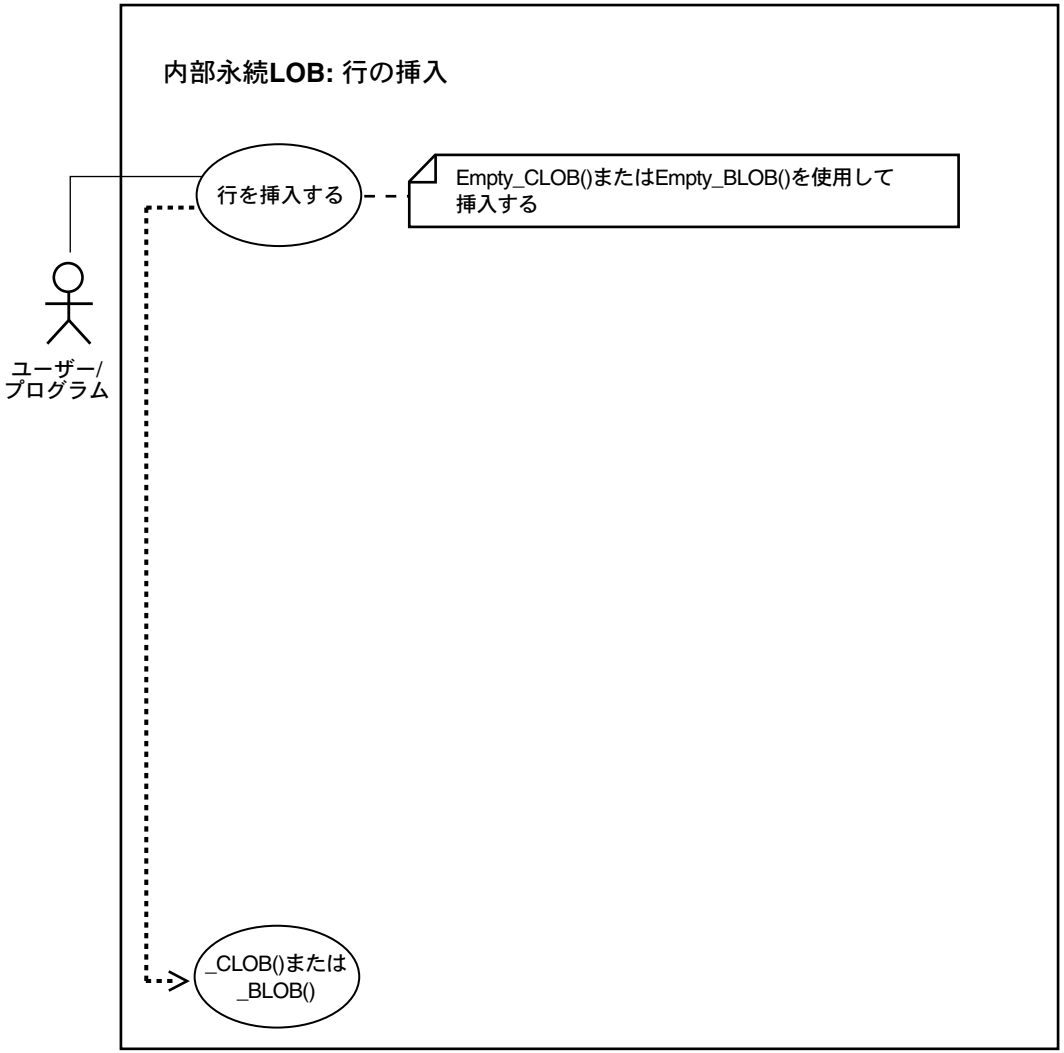
SQL: LOB を含むネストした表の作成

```
/* Create type textdoc_typ as the base type
   for the nested table textdoc_ntab,
   where textdoc_ntab contains a LOB:
*/
DROP TYPE textdoc_typ force;
DROP TYPE textdoc_ntab;
DROP TABLE textdoc_ntable;
CREATE TYPE textdoc_typ AS OBJECT
(
   document_typ    VARCHAR2(32),
   formatted_doc   BLOB
);

/* The type has been created. Now you need a */
/* nested table of that type to embed in */
/* table Print_media, so: */
CREATE TYPE textdoc_ntab AS TABLE of textdoc_typ;
CREATE TABLE textdoc_ntable (
   id number,
   textdoc_ntab textdoc_typ)
NESTED TABLE textdoc_ntab STORE AS textdoc_nestedtab;
```

ネストした表の実際の埋込みは、その表を含む構造が定義されるときに行われます。この例では、Print_media 表が作成されるときに、NESTED TABLE 文によって行われます。

EMPTY_CLOB() または EMPTY_BLOB() を使用した LOB 値の挿入

図 10-4 利用図 : EMPTY_CLOB() または EMPTY_BLOB() を使用した行の挿入



参照：

- 10-2 ページの表 10-1「利用モデル : 内部永続 LOB」を参照してください。
- 10-20 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。
- 10-23 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。

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

次に、LOB 挿入のガイドラインを示します。

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() 関数またはファンクションを使用して、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。

```
INSERT INTO Print_media (product_id, ad_id, ad_sourcetext)
     VALUES (1, 1, 'This is a One Line Advertisement');
```

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。

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」
- 7-15 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）」
- 7-17 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」
- 7-18 ページの「例 : PL/SQL - 4,000 バイトを超えるバインド（挿入は未サポート）」
- 7-19 ページの「例 : PL/SQL - 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」

構文

各プログラム環境で使用可能な関数またはファンクションのリストは、第 3 章「様々なプログラム環境での LOB のサポート」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:『Oracle9*i* SQL リファレンス』の第 17 章「SQL 文 : DROP SEQUENCE ～ ROLLBACK」の「INSERT」
- C（OCI）: 参照マニュアルはありません。
- C++（OCCI）: 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）:『Oracle9*i* JDBC 開発者ガイドおよびリファレンス』の第 8 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『Oracle9*i* 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「JDBC 2.0 LOB 型と Oracle 拡張型のサポート」の「BLOB、CLOB および BFILE のサポート」

使用例

参照： この例で使用される PM スキーマおよび Print_media 表の詳細は、『Oracle9*i* サンプル・スキーマ』を参照してください。

例

次のプログラム環境での例を示します。

- SQL: EMPTY_CLOB() / EMPTY_BLOB() を使用した値の挿入（10-19 ページ）
- C（OCI）: 今回のリリースでは例は提供されません。
- C++（OCCI）: 今回のリリースでは例は提供されません。
- C/C++（Pro*C/C++）: 今回のリリースでは例は提供されません。
- COBOL（Pro*COBOL）: 今回のリリースでは例は提供されません。
- Visual Basic（OO4O）: 今回のリリースでは例は提供されません。
- Java（JDBC）: 今回のリリースでは例は提供されません。

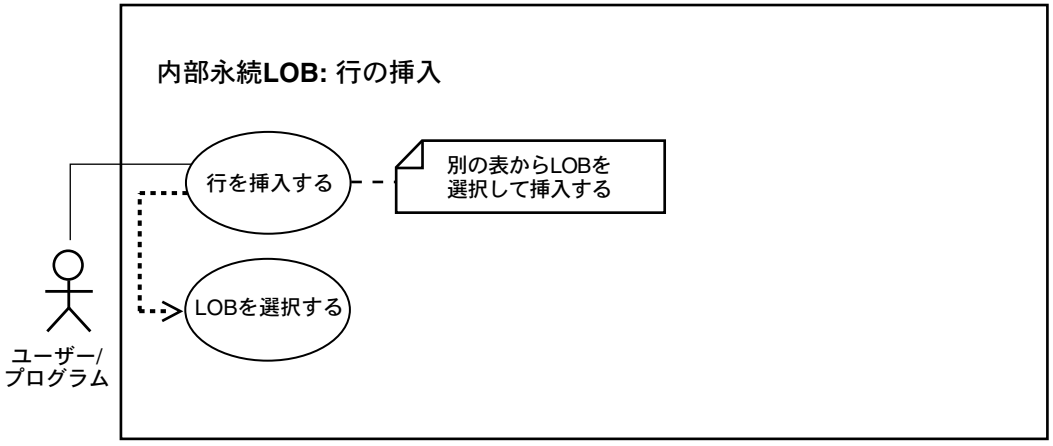
SQL: EMPTY_CLOB() / EMPTY_BLOB() を使用した値の挿入

これらのファンクションは、Oracle SQL では特別ファンクション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。

```
/* In the new row of table Print_media,
   the columns ad_sourcetext and ad_fltextn are initialized using EMPTY_CLOB(),
   the columns ad_composite and ad_photo are initialized using EMPTY_BLOB(),
   the column formatted-doc in the nested table is initialized using EMPTY_BLOB(),
   the column logo in the column object is initialized using EMPTY_BLOB(): */
INSERT INTO Print_media
   VALUES (3060,11001, EMPTY_BLOB(), EMPTY_CLOB(),EMPTY_CLOB(),EMPTY_CLOB(),
   textdoc_tab(textdoc_typ ('HTML', EMPTY_BLOB())), EMPTY_BLOB(), NULL,
   adheader_typ('any header name', <any date>, 'ad header text goes here',
EMPTY_BLOB()),
   'Press release goes here');
```

別の表からの LOB の選択による行の挿入

図 10-5 利用図 : 別の表からの LOB の選択による行の挿入



参照 :

- 10-2 ページの表 10-1「利用モデル : 内部永続 LOB」を参照してください。
- 10-16 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。
- 10-23 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。

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、`Print_media` および `Online_media` が同じスキーマを持つとすると、この文によって `Print_media` 表の中に LOB ロケータが作成され、LOB データが、`Online_media` から `Print_media` 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。

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」
- 7-15 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）」
- 7-17 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」
- 7-18 ページの「例 : PL/SQL - 4,000 バイトを超えるバインド（挿入は未サポート）」
- 7-19 ページの「例 : PL/SQL - 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」

構文

次のマニュアルの項を参照してください。

- SQL: 『Oracle9*i* SQL リファレンス』の第 17 章「SQL 文 : DROP SEQUENCE ～ ROLLBACK」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必要があります。

次のコード例は、Online_media 表が、Print_media 表の ad_textdocs_ntab 列によって参照される Print_media と同じ型であることを前提としています。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 Print_media に挿入します。

> **参照：** この例で使用される PM スキーマおよび Print_media 表の詳細は、『Oracle9*i* サンプル・スキーマ』を参照してください。

例

例を SQL で示します。この例は、すべてのプログラム環境に適用されます。

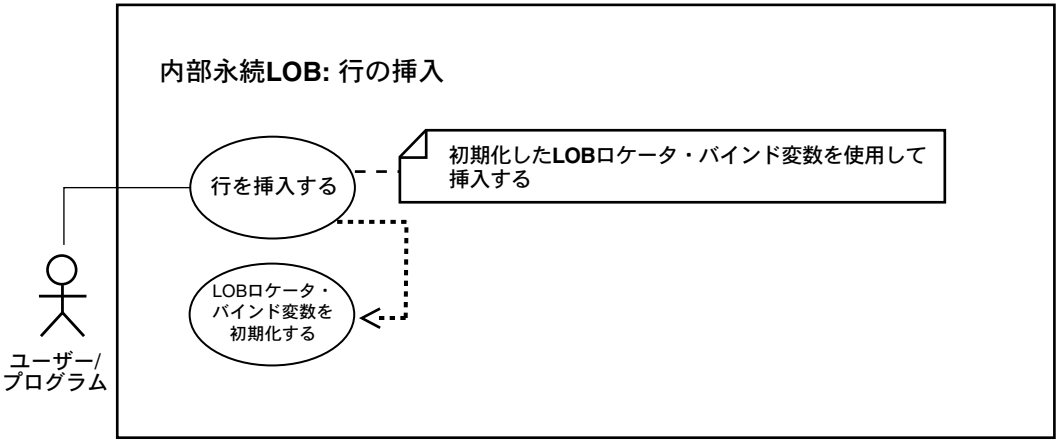
- SQL: 別の表からの LOB の選択による行の挿入（10-22 ページ）

SQL: 別の表からの LOB の選択による行の挿入

```
/* Store records in the archive table Online_media: */
INSERT INTO Online_media
   VALUES (3060, NULL, NULL, NULL, NULL,
           'some text about this CRT Monitor', NULL);

/* Insert values into Print_media by selecting from Online_media: */
INSERT INTO Print_media (product_id, ad_id, ad_sourcetext)
   (SELECT product_id, 11001, product_text
          FROM Online_media where product_id = 3060);
```

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 10-6 利用図 : 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 :

- 10-2 ページの表 10-1「利用モデル : 内部永続 LOB」を参照してください。
- 10-16 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。
- 10-20 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。

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を含む場合の、LOB への挿入および更新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」
- 7-15 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)」

- 7-17 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」
- 7-18 ページの「例 : PL/SQL - 4,000 バイトを超えるバインド（挿入は未サポート）」
- 7-19 ページの「例 : PL/SQL - 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」

構文

各プログラム環境で使用可能な関数またはファンクションのリストは、第 3 章「様々なプログラム環境での LOB のサポート」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:『Oracle9*i* SQL リファレンス』の第 17 章「SQL 文 : DROP SEQUENCE ～ ROLLBACK」の「INSERT」
- C（OCI）:『Oracle Call Interface プログラマーズ・ガイド』の第 16 章「その他の OCI リレーショナル関数」の「LOB 関数」
- C++（OCCI）:『Oracle C++ Call Interface プログラマーズ・ガイド』
- COBOL（Pro*COBOL）:『Pro*COBOL Precompiler プログラマーズ・ガイド』の LOB に関する詳細、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・ディレクティブ」の「INSERT（実行可能埋込み SQL）」
- C/C++（Pro*C/C++）:『Pro*C/C++ Precompiler プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびディレクティブ」の「INSERT（実行可能埋込み SQL）」
- Visual Basic（OO4O オンライン・ヘルプ）: ヘルプの「目次」タブから、「OO4O オートメーション・サーバー・リファレンス」>「OO4O サーバーのオブジェクト」>「OraDynaset」を選択
- Java（JDBC）:『Oracle9*i* JDBC 開発者ガイドおよびリファレンス』の第 8 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『Oracle9*i* 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「JDBC 2.0 LOB 型と Oracle 拡張型のサポート」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、`Print_media` の 1 つの行の中にある `ad_photo` 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での例を示します。

- PL/SQL（DBMS_LOB）：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（10-25 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（10-26 ページ）
- C++（OCCI）：今回のリリースでは例は提供されません。
- 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（10-28 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行の挿入（10-29 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（10-30 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（10-30 ページ）

PL/SQL（DBMS_LOB）：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

```
/* Note that the example procedure insertUseBindVariable_proc is not part of the
   DBMS_LOB package.   */
CREATE OR REPLACE PROCEDURE insertUseBindVariable_proc
   (productnum IN NUMBER, adnum IN NUMBER, Blob_loc IN BLOB) IS
BEGIN
   INSERT INTO Print_media (product_id, ad_id, ad_photo)
      VALUES (productnum, adnum, Blob_loc);
END;

DECLARE
   Blob_loc  BLOB;
BEGIN
   /* Select the LOB from the row where product_id = 3106 and ad_id=13001.
      Initialize the LOB locator bind variable: */
   SELECT ad_photo INTO Blob_loc
      FROM Print_media
      WHERE product_id = 3106 AND ad_id=13001;
  /* Insert into the row where product_id = 2056 AND ad_id=12001 */
  insertUseBindVariable_proc (2056, 12001, Blob_loc);
  COMMIT;
END;
```

C（OCI）：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

```
/* Select the locator into a locator variable */
sb4 select_Printmedia_Locator (Lob_loc, errhp, stmthp, svchp)
OCILobLocator *Lob_loc;
OCIError      *errhp;
OCIStmt       *stmthp;
OCISvcCtx     *svchp;
{
  OCIDefine *defnp1, *defnp2;

  text  *sqlstmt =
    (text *)"SELECT ad_photo FROM Print_media WHERE product_id=2268 AND
ad_id=21001";

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

  /* Define the column being selected */
  checkerr (errhp, OCIDefineByPos(stmthp, &defnp1, errhp, (ub4) 1,
                                  (dvoid *)&Lob_loc, (sb4)0,
                                  (ub2)SQLT_BLOB,(dvoid *)0, (ub2 *)0, (ub2 *)0,
                                  (ub4)OCI_DEFAULT));
  checkerr (errhp, OCIDefineByPos(stmthp, &defnp2, errhp, (ub4) 2,
                                  (dvoid *)&Lob_loc, (sb4)0,
                                  (ub2)SQLT_BLOB,(dvoid *)0, (ub2 *)0, (ub2 *)0,
                                  (ub4)OCI_DEFAULT));

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

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

```
  int            clipid;
  OCILobLocator *Lob_loc;
  OCIBind       *bndhp2;
  OCIBind       *bndhp1;

  text          *insstmt =
   (text *) "INSERT INTO Print_media (product_id, ad_photo) VALUES (:2268,
:3060)";


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

  /* Select a LOB locator from the Print_media table */
  select_Printmedia_Locator(Lob_loc, errhp, stmthp, svchp);


  /* Insert the locator into the Print_media table with product_id=3060 */
  product_id = 3060;

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

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

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

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

COBOL（Pro*COBOL）：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 "PM/PM".
           EXEC SQL INCLUDE SQLCA END-EXEC.

       PROCEDURE DIVISION.
       INSERT-LOB.

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

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

      * Populate the LOB
           EXEC SQL WHENEVER NOT FOUND GOTO END-OF-BLOB END-EXEC.
           EXEC SQL
              SELECT AD_PHOTO INTO :BLOB1
                 FROM PRINT_MEDIA WHERE PRODUCT_ID = 2268 AND AD_ID = 21001
END-EXEC.

      * Insert the value with PRODUCT_ID of 3060
           EXEC SQL
              INSERT INTO PRINT_MEDIA (PRODUCT_ID, AD_PHOTO)
                 VALUES (3060, 11001, :BLOB1)END-EXEC.

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

       SQL-ERROR.
           EXEC SQL WHENEVER SQLERROR CONTINUE END-EXEC.
           DISPLAY " ".
           DISPLAY "ORACLE ERROR DETECTED:".
```

```
DISPLAY " ".
DISPLAY SQLERRMC.
EXEC SQL ROLLBACK WORK RELEASE END-EXEC.
STOP RUN.
```

C/C++（Pro*C/C++）：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, Rownum2;
   OCIBlobLocator *Lob_loc;
{
   EXEC SQL WHENEVER SQLERROR DO Sample_Error();
   EXEC SQL INSERT INTO Print_media (product_id, ad_id, ad_photo)
      VALUES (:Rownum, :Rownum2, :Lob_loc);
}
void insertBLOB_proc()
{
   OCIBlobLocator *Lob_loc;

   /* Initialize the BLOB Locator: */
   EXEC SQL ALLOCATE :Lob_loc;

   /* Select the LOB from the row where product_id = 2268 and ad_id=21001: */
   EXEC SQL SELECT ad_photo INTO :Lob_loc
      FROM Print_media WHERE product_id = 2268 AND ad_id = 21001;

   /* Insert into the row where product_id = 3106 and ad_id = 13001: */
   insertUseBindVariable_proc(3106, 13001, Lob_loc);

   /* Release resources held by the locator: */
  EXEC SQL FREE :Lob_loc;
}
```

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

Visual Basic（OO4O）: 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, OraPhoto1 as OraBLOB, OraPhotoClone as OraBLOB
Set OraDyn = OraDb.CreateDynaset(
   "SELECT * FROM Print_media ORDER BY product_id", ORADYN_DEFAULT)
Set OraPhoto1 = OraDyn.Fields("ad_photo").Value
'Clone it for future reference
Set OraPhotoClone = OraPhoto1

'Go to Next row
OraDyn.MoveNext

'Lets update the current row and set the LOB to OraPhotoClone
OraDyn.Edit
Set OraPhoto1 = OraPhotoClone
OraDyn.Update
```

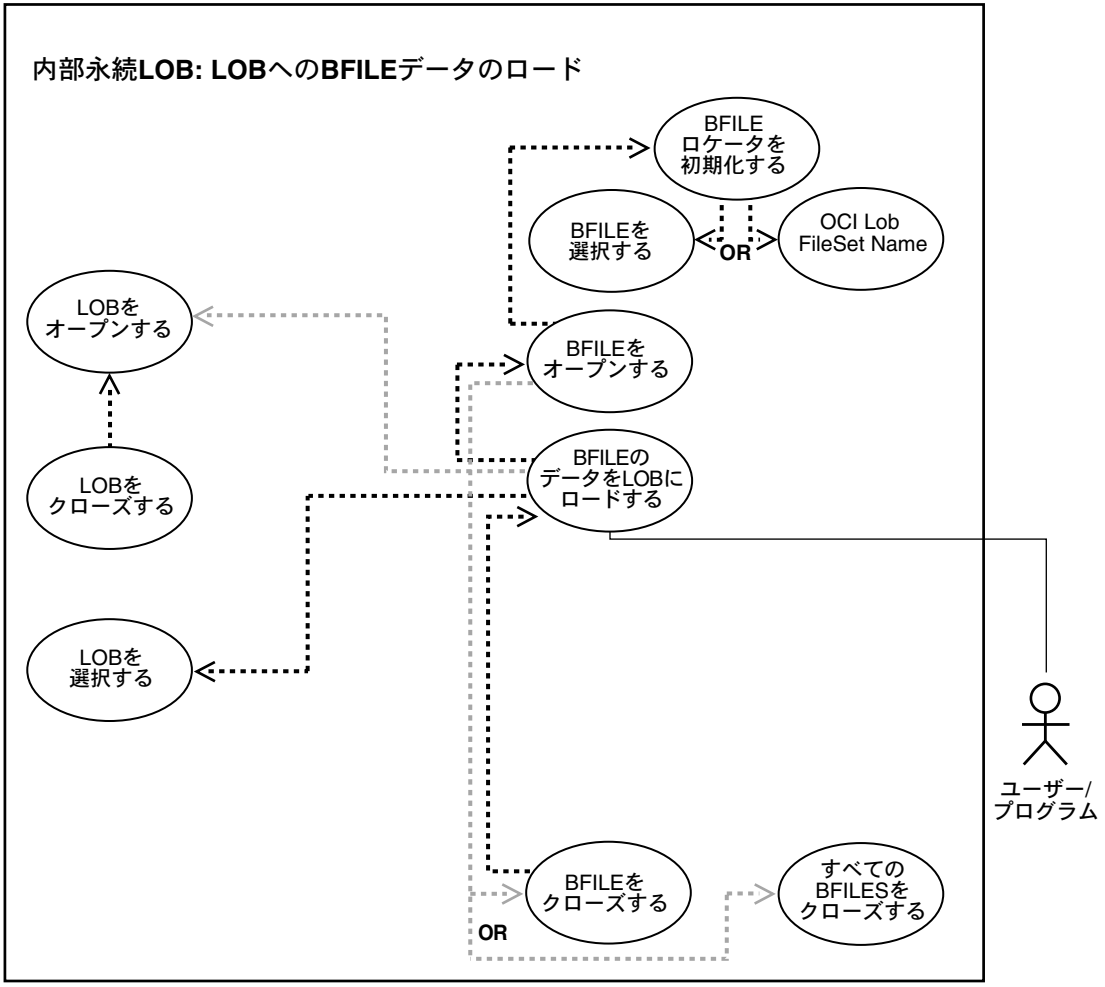
Java（JDBC）: 初期化した LOB ロケータ・バインド変数を使用した行の挿入

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

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

public class Ex2_31
```

```
{
  public static void main (String args [])
       throws Exception
  {
    // Load the Oracle JDBC driver
    DriverManager.registerDriver (new oracle.jdbc.driver.OracleDriver ());
    // Connect to the database:
    Connection conn =
       DriverManager.getConnection ("jdbc:oracle:oci8:@", "pm", "pm");

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

    // Create a Statement:
    Statement stmt = conn.createStatement ();
    try
    {
       ResultSet rset = stmt.executeQuery (
          "SELECT ad_photo FROM Print_media WHERE product_id = 3106 AND ad_id =
13001");
       if (rset.next())
       {
          // retrieve the LOB locator from the ResultSet
          BLOB adphoto_blob = ((OracleResultSet)rset).getBLOB (1);
          OraclePreparedStatement ops =
          (OraclePreparedStatement) conn.prepareStatement(
             "INSERT INTO Print_media (product_id, ad_id, ad_photo) VALUES (2268,
21001, ?)");
          ops.setBlob(1, adphoto_blob);
          ops.execute();
          conn.commit();
          conn.close();
       }
    }
    catch (SQLException e)
    {
       e.printStackTrace();
    }
  }
}
```

LOB への BFILE データのロード

図 10-7 利用図 : LOB への BFILE データのロード



参照：

- 10-2 ページの表 10-1「利用モデル : 内部永続 LOB」を参照してください。
- 10-42 ページの「内部永続 BLOB への BFILE バイナリ・データのロード」を参照してください。
- 10-46 ページの「内部永続 CLOB への BFILE データのロード」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

注意： `LOADBLOBFROMFILE` および `LOADCLOBFROMFILE` プロシージャによって、このプロシージャの機能が実装されるため、バイナリ・データおよび文字データのロード機能が向上します。機能が向上したプロシージャは、PL/SQL 環境でのみ使用できます。できるかぎり、機能が向上したプロシージャを使用することをお薦めします。詳細は、10-42 ページの「内部永続 BLOB への BFILE バイナリ・データのロード」および 10-46 ページの「内部永続 CLOB への BFILE データのロード」を参照してください。

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インライン）か 1 つ以上のセカンダリ・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。

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、第 4 章の「インライン LOB データのロード」および「アウトライン LOB データのロード」を参照してください。

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に、暗黙的に実行されます。DBMS_LOB.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。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。このため、テキストをロードする場合は、LOADCLOBFROMFILE プロシージャを使用してください（10-46 ページの「内部永続 CLOB への BFILE データのロード」を参照）。

サイズを BFILE のサイズより小さく指定

`maxlobsize` を指定して BFILE 全体をロードしない場合は、次に示すとおり、ロード対象として指定する BFILE のデータ量（バイト数）を BFILE のサイズ以下にする必要があります。

- **DBMS_LOB.LOADFROMFILE**: サイズを BFILE のサイズより小さくを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関数またはファンクションのリストは、第 3 章「様々なプログラム環境での LOB のサポート」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）：『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』の第 23 章「DBMS_LOB」の「DBMS_LOB サブプログラムの要約」の「LOADFROMFILE プロシージャ」
- C（OCI）：『Oracle Call Interface プログラマーズ・ガイド』の第 16 章「その他の OCI リレーショナル関数」の「LOB 関数」の OCILobLoadFromFile()
- C++（OCCI）：『Oracle C++ Call Interface プログラマーズ・ガイド』
- COBOL（Pro*COBOL）：『Pro*COBOL Precompiler プログラマーズ・ガイド』の LOB に関する詳細、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・ディレクティブ」の「LOB LOAD（実行可能埋込み SQL 拡張機能）」、「LOB OPEN（実行可能埋込み SQL 拡張機能）」と「LOB CLOSE（実行可能埋込み SQL 拡張機能）」
- C/C++（Pro*C/C++）：『Pro*C/C++ Precompiler プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびディレクティブ」の「LOB LOAD（実行可能埋込み SQL 拡張機能）」

- Visual Basic（OO4O オンライン・ヘルプ）：ヘルプの「目次」タブから、「OO4O オートメーション・サーバー・リファレンス」>「OO4O サーバーのオブジェクト」>「OraBLOB、OraCLOB」>「メソッド」>「CopyFromBFILE」、および「OO4O オートメーション・サーバー・リファレンス」>「OO4O サーバーのオブジェクト」>「OraDynaset」、「OraDatabase」、「OraConnection」を選択
- Java（JDBC）：『Oracle9*i* JDBC 開発者ガイドおよびリファレンス』の第 8 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『Oracle9*i* 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「JDBC 2.0 LOB 型と Oracle 拡張型のサポート」の「BLOB、CLOB および BFILE のサポート」

使用例

次の例は、OS のソース・ファイル（keyboard_3106_13001）が LOB データを含み、ターゲットの LOB（ad_composite）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 ADVERT_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での例を示します。

- PL/SQL（DBMS_LOB）：LOB への BFILE データのロード（10-35 ページ）
- C（OCI）：LOB への BFILE データのロード（10-36 ページ）
- C++（OCCI）：今回のリリースでは例は提供されません。
- COBOL（Pro*COBOL）：LOB への BFILE データのロード（10-38 ページ）
- C/C++（Pro*C/C++）：今回のリリースでは例は提供されません。
- Visual Basic（OO4O）：LOB への BFILE データのロード（10-39 ページ）
- Java（JDBC）：LOB への BFILE データのロード（10-40 ページ）

PL/SQL（DBMS_LOB）：LOB への BFILE データのロード

```
/* Loading a LOB with Data from a BFILE. Note that the example procedure
loadLOBFromBFILE_proc is not part of the
   DBMS_LOB package: */
CREATE OR REPLACE PROCEDURE loadLOBFromBFILE_proc IS
   Dest_loc       BLOB;
   Src_loc        BFILE := BFILENAME('ADPHOTO_DIR', 'keyboard_3106_13001');
   Amount         INTEGER := 4000;
BEGIN
   SELECT ad_photo INTO Dest_loc FROM print_media
      WHERE product_id = 3106 and ad_id=13001 FOR UPDATE;
```

```
   /* Opening the source BFILE is mandatory: */
   DBMS_LOB.OPEN(Src_loc, DBMS_LOB.LOB_READONLY);

   /* Opening the LOB is optional: */
   DBMS_LOB.OPEN(Dest_loc, DBMS_LOB.LOB_READWRITE);
   DBMS_LOB.LOADFROMFILE(Dest_loc, Src_loc, Amount);

   /* Closing the LOB is mandatory if you have opened it: */
   DBMS_LOB.CLOSE(Dest_loc);
   DBMS_LOB.CLOSE(Src_loc);
   COMMIT;
END;
```

C（OCI）: LOB への BFILE データのロード

```
/* Selecting a BLOB from Print_media and loading it with data from a BFILE */
sb4 select_lock_adphoto_locator_3(Lob_loc, errhp, svchp, stmthp)
OCILobLocator *Lob_loc;
OCIError      *errhp;
OCISvcCtx     *svchp;
OCIStmt       *stmthp;
{
  text  *sqlstmt =
     (text *)"SELECT ad_photo FROM Print_media WHERE product_id=2056
              AND ad_id = 12001 FOR UPDATE";
  OCIDefine *defnp1, *defnp2;
  checkerr (errhp, OCIStmtPrepare(stmthp, errhp, sqlstmt,
                                  (ub4)strlen((char *)sqlstmt),
                                  (ub4) OCI_NTV_SYNTAX, (ub4) OCI_DEFAULT));
    checkerr (errhp, OCIDefineByPos(stmthp, &defnp1, errhp, (ub4) 1,
                                  (dvoid *)&Lob_loc, (sb4)0,
                                  (ub2) SQLT_BLOB,(dvoid *) 0,
                                  (ub2 *) 0, (ub2 *) 0, (ub4) OCI_DEFAULT)
                     OCIDefineByPos(stmthp, &defnp2, errhp, (ub4) 2,
                                  (dvoid *)&Lob_loc, (sb4)0,
                                  (ub2) SQLT_BLOB,(dvoid *) 0,
                                  (ub2 *) 0, (ub2 *) 0, (ub4) OCI_DEFAULT));
  /* Execute the select and fetch one row */
  checkerr(errhp, OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                                 (CONST OCISnapshot*) 0, (OCISnapshot*) 0,
                                 (ub4) OCI_DEFAULT));
    return 0;
}

void LoadLobDataFromBFile(envhp, errhp, svchp, stmthp)
OCIEnv    *envhp;
```

```
OCIError  *errhp;
OCISvcCtx *svchp;
OCIStmt   *stmthp;
{
  OCILobLocator *bfile;
  OCILobLocator *blob;
  ub4            amount= 4000;

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

  /* Select a ad_photo locator for update */
  printf (" select the ad_photo locator...\n");
  select_lock_adphoto_locator_2056(blob, errhp, svchp, stmthp);

  /* Set the Directory Alias and File Name of the ad_photo file */
  printf ("  set the file name in bfile\n");
  checkerr (errhp, OCILobFileSetName(envhp, errhp, &bfile, (text*)"ADPHOTO_DIR",
                                     (ub2)strlen("ADPHOTO_DIR"),
                                     (text*)"mousepad_2056_12001",
                                     (ub2)strlen("mousepad_2056_12001")));
    printf (" open the bfile\n");
  /* Opening the BFILE locator is Mandatory */
  checkerr (errhp, (OCILobOpen(svchp, errhp, bfile, OCI_LOB_READONLY)));

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

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

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

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

  return;
}
```

COBOL（Pro*COBOL）: 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.
      * Declare the amount to load.  The value here
      * was chosen arbitrarily
       01  LOB-AMT        PIC S9(9) COMP VALUE 10.
       01  USERID   PIC X(11) VALUES "PM/PM".
           EXEC SQL INCLUDE SQLCA END-EXEC.
       PROCEDURE DIVISION.
       LOB-LOAD.

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

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

      * Set up the directory and file information
           MOVE "ADGRAPHIC_DIR" TO DIR-ALIAS-ARR.
           MOVE 9 TO DIR-ALIAS-LEN.
           MOVE "keyboard_3106_13001" TO FNAME-ARR.
           MOVE 16 TO FNAME-LEN.

           EXEC SQL
              LOB FILE SET :BFILE1 DIRECTORY = :DIR-ALIAS,FILENAME = :FNAME
           END-EXEC.

      * Allocate and initialize the destination BLOB
           EXEC SQL ALLOCATE :DEST END-EXEC.
           EXEC SQL WHENEVER NOT FOUND GOTO END-OF-BLOB END-EXEC.
           EXEC SQL SELECT AD_GRAPHIC INTO :DEST
           FROM PRINT_MEDIA WHERE PRODUCT_ID = 2268 AND AD_ID = 21001 FOR UPDATE
END-EXEC.

      * Open the source BFILE for READ
           EXEC SQL LOB OPEN :BFILE1 READ ONLY END-EXEC.

      * Open the destination BLOB for READ/WRITE
           EXEC SQL LOB OPEN :DEST READ WRITE END-EXEC.
```

```
* Load the destination BLOB from the source BFILE
     EXEC SQL LOB LOAD :LOB-AMT FROM FILE :BFILE1 INTO :DEST END-EXEC.

* Close the source and destination LOBs
     EXEC SQL LOB CLOSE :BFILE1 END-EXEC.
     EXEC SQL LOB CLOSE :DEST END-EXEC.
 END-OF-BLOB.
     EXEC SQL FREE :DEST END-EXEC.
     EXEC SQL FREE :BFILE1 END-EXEC.
     EXEC SQL ROLLBACK WORK RELEASE END-EXEC.
     STOP RUN.
 SQL-ERROR.
     EXEC SQL
        WHENEVER SQLERROR CONTINUE END-EXEC.
     DISPLAY " ".
     DISPLAY "ORACLE ERROR DETECTED:".
     DISPLAY " ".
     DISPLAY SQLERRMC.
     EXEC SQL ROLLBACK WORK RELEASE END-EXEC.
     STOP RUN.
```

Visual Basic （OO4O）: LOB への BFILE データのロード

```
Dim OraDyn as OraDynaset, OraPhoto1 as OraBLOB, OraMyBfile as OraBFile

OraConnection.BeginTrans
Set OraDyn = OraDb.CreateDynaset(
   "SELECT * FROM Print_media ORDER BY product_id, ad_id", ORADYN_DEFAULT)
Set OraPhoto1 = OraDyn.Fields("ad_photo").Value

OraDb.Parameters.Add "id", 3060,ORAPARAM_INPUT
OraDb.Parameters.Add "mybfile", Null,ORAPARAM_OUTPUT
OraDb.Parameters("mybfile").serverType = ORATYPE_BFILE

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

Set OraMyBFile = OraDb.Parameters("mybfile").Value
'Go to Next row
OraDyn.MoveNext

OraDyn.Edit
'Lets update OraPhoto1 data with that from the BFILE
OraPhoto1.CopyFromBFile  OraMyBFile
OraDyn.Update

OraConnection.CommitTrans
```

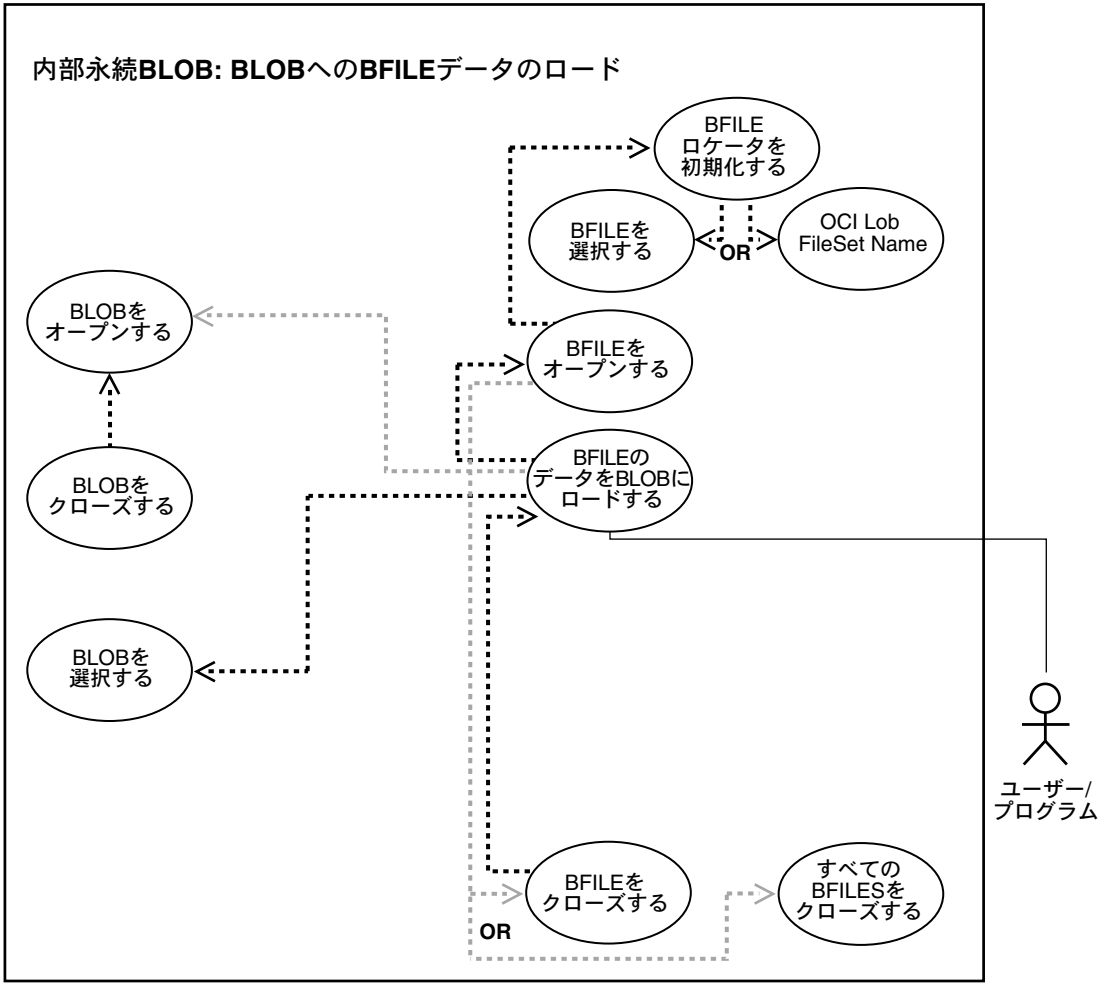
Java（JDBC）: LOB への BFILE データのロード

```
// Java IO classes:
import java.io.InputStream;
import java.io.OutputStream;

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

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

public class Ex2_45
{
  public static void main (String args [])
     throws Exception
  {
    // Load the Oracle JDBC driver:
    DriverManager.registerDriver (new oracle.jdbc.driver.OracleDriver ());
    // Connect to the database:
    Connection conn =
       DriverManager.getConnection ("jdbc:oracle:oci8:@", "pm", "pm");
    conn.setAutoCommit (false);

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

    try
    {
     BFILE src_lob = null;
     BLOB dest_lob = null;
     InputStream in = null;
     OutputStream out = null;
     byte buf[] = new byte[1000];
     ResultSet rset = null;

       rset = stmt.executeQuery (
          "SELECT BFILENAME('ADPHOTO_DIR', 'keyboard_3106_13001') FROM DUAL");
   if (rset.next())
   {
      src_lob = ((OracleResultSet)rset).getBFILE (1);
```

```
      src_lob.openFile();
      in = src_lob.getBinaryStream();
    }

       rset = stmt.executeQuery (
          "SELECT ad_photo FROM Print_media WHERE product_id = 3106
          AND AD_ID = 13001 FOR UPDATE");
    if (rset.next())
    {
      dest_lob = ((OracleResultSet)rset).getBLOB (1);

      // Fetch the output stream for dest_lob:
        out = dest_lob.getBinaryOutputStream();
    }

    int length = 0;
    int pos = 0;
    while ((in != null) && (out != null) && ((length = in.read(buf)) != -1))
    {
      System.out.println(
         "Pos = " + Integer.toString(pos) + ".  Length = " +
         Integer.toString(length));
      pos += length;
      out.write(buf, pos, length);
    }

    // Close all streams and file handles:
    in.close();
    out.flush();
    out.close();
    src_lob.closeFile();

    // Commit the transaction:
    conn.commit();
    conn.close();
     }
     catch (SQLException e)
     {
       e.printStackTrace();
     }
   }
}
```

内部永続 BLOB への BFILE バイナリ・データのロード

図 10-8 利用図 : BLOB への BFILE データのロード



参照：

- 10-2 ページの表 10-1「利用モデル : 内部永続 LOB」を参照してください。
- 10-32 ページの「LOB への BFILE データのロード」を参照してください。
- 10-46 ページの「内部永続 CLOB への BFILE データのロード」を参照してください。

用途

BFILE のバイナリ・データを内部永続 BLOB にロードします。

使用上の注意

バイナリ・データをロードする場合は LOADBLOBFROMFILE を使用し、テキストをロードする場合は LOADCLOBFROMFILE を使用してください。結果は、LOADFROMFILE を使用した場合と同じになります。また、新しいオフセットが戻されます。LOADCLOBFROMFILE API を使用すると、BFILE のキャラクタ・セット ID を指定できるため、BFILE データのキャラクタ・セットから宛先 CLOB/NCLOB のキャラクタ・セットへの変換が適切に実行されます。

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インライン）か 1 つ以上のセカンダリ・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。

SQL*Loader による内部 LOB へのデータのロードに関する詳細およびヒントは、第 4 章の「インライン LOB データのロード」および「アウトライン LOB データのロード」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関数またはファンクションのリストは、第 3 章「様々なプログラム環境での LOB のサポート」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）：『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』の第 23 章「DBMS_LOB」の「DBMS_LOB サブプログラムの要約」の「LOADBLOBFROMFILE プロシージャ」

使用例

この項の例では、製品メディアのサンプル・スキーマの Print_media 表を使用します。また、ターゲットとなる BLOB にロードするバイナリ LOB データを含む OS のソース・ディレクトリが存在することを想定しています。

例

「PL/SQL（DBMS_LOB）: 内部永続 BLOB への BFILE データのロード」の例では、PL/SQL プログラム環境での LOADBLOBFROMFILE の使用方法を示します（その他のプログラム環境はサポートされていません）。

PL/SQL（DBMS_LOB）: 内部永続 BLOB への BFILE データのロード

次の項目の例を示します。

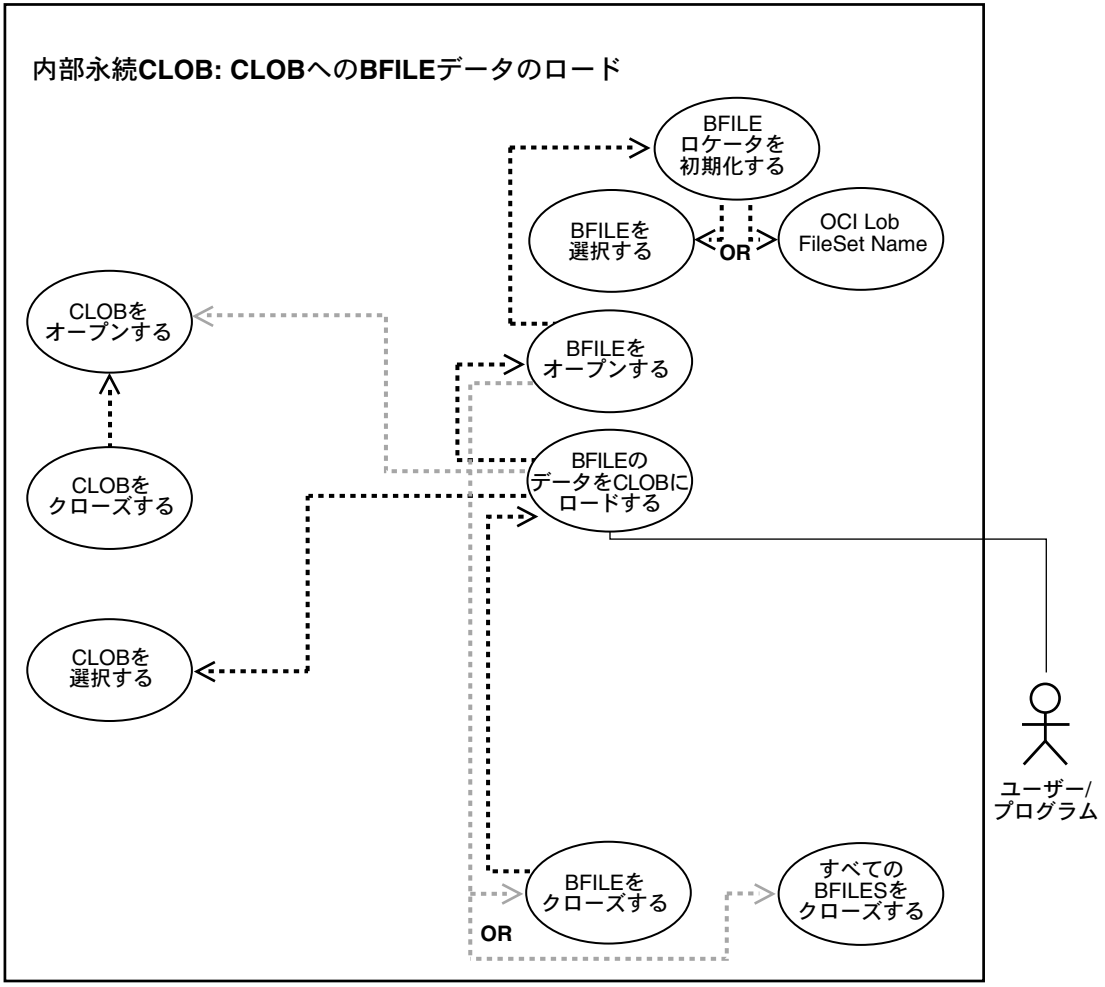
- LOADBLOBFROMFILE を使用して、最初にファイルの長さを取得することなくファイル全体をロードする方法
- オフセットの戻り値を使用して、実際にロードされたデータ量を計算する方法

```
DECLARE
  src_loc     BFILE := bfilename('ADVERT_DIR','display_ad_frame') ;
  dst_loc     BLOB;
  src_offset  NUMBER := 1;
  dst_offset  NUMBER := 1;
  src_osin    NUMBER;
  dst_osin    NUMBER;
  bytes_rd    NUMBER;
  bytes_wt    NUMBER;
BEGIN
  SELECT ad_composite INTO dst_loc FROM Print_media
      WHERE product_id=3106 and ad_id=13001 FOR UPDATE;

  /* Opening the source BFILE is mandatory */
  dbms_lob.fileopen(src_loc, dbms_lob.file_readonly);

  /* Opening the LOB is optional */
  dbms_lob.OPEN(dst_loc, dbms_lob.lob_readwrite);
  /* Save the input source/destination offsets */
  src_osin := src_offset;
  dst_osin := dst_offset;
  /* Use LOBMAXSIZE to indicate loading the entire BFILE */

dbms_lob.LOADBLOBFROMFILE(dst_loc,src_loc,dbms_lob.lobmaxsize,src_offset,dst_offset)
;

  /* Closing the LOB is mandatory if you have opened it */
```

```
  dbms_lob.close(dst_loc);
  dbms_lob.filecloseall();
  COMMIT;

  /* Use the src_offset returned to calculate the actual amount read from the BFILE
*/
  bytes_rd := src_offset - src_osin;
  dbms_output.put_line(' Number of bytes read from the BFILE ' || bytes_rd ) ;
  /* Use the dst_offset returned to calculate the actual amount written to the BLOB
*/
  bytes_wt := dst_offset - dst_osin;
  dbms_output.put_line(' Number of bytes written to the BLOB ' || bytes_wt ) ;
  /* If there is no exception the number of bytes read should equal to the number of
bytes written */

END ;
```

内部永続 CLOB への BFILE データのロード

図 10-9 利用図 : CLOB への BFILE データのロード



参照：

- 10-2 ページの表 10-1「利用モデル : 内部永続 LOB」を参照してください。
- 10-32 ページの「LOB への BFILE データのロード」を参照してください。
- 10-42 ページの「内部永続 BLOB への BFILE バイナリ・データのロード」を参照してください。

用途

BFILE の文字データを内部永続 CLOB または NCLOB にロードします。

使用上の注意

バイナリ・データをロードする場合は LOADBLOBFROMFILE を使用し、テキストをロードする場合は LOADCLOBFROMFILE を使用してください。後者の方法では、BFILE のキャラクタ・セット ID を指定できます。LOADCLOBFROMFILE API を使用すると、BFILE のキャラクタ・セット ID を指定できるため、BFILE データのキャラクタ・セットから宛先 CLOB/NCLOB のキャラクタ・セットへの変換が適切に実行されます。

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インライン）か 1 つ以上のセカンダリ・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。

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、第 4 章の「インライン LOB データのロード」および「アウトライン LOB データのロード」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関数またはファンクションのリストは、第 3 章「様々なプログラム環境での LOB のサポート」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）：『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』の第 23 章「DBMS_LOB」の「DBMS_LOB サブプログラムの要約」の「LOADCLOBFROMFILE プロシージャ」

使用例

この項の例では、製品メディアのサンプル・スキーマの Print_media 表を使用します。また、ターゲットとなる CLOB または NCLOB にロードする LOB の文字データを含む OS のソース・ディレクトリが存在することを想定しています。

例

「PL/SQL（DBMS_LOB）: 内部永続 CLOB への BFILE データのロード」の例では、PL/SQL プログラム環境での LOADCLOBFROMFILE の使用方法を示します（その他のプログラム環境はサポートされていません）。

PL/SQL（DBMS_LOB）: 内部永続 CLOB への BFILE データのロード

例

次の項目の例を示します。

- デフォルト CSIDE（0）の使用方法
- BFILE に対して getlength をコールせずに、ファイル全体をロードする方法
- 戻されたオフセット値を使用して、実際にロードしたデータ量を計算する方法

この例では、ad_source が UTF8 キャラクタ・セット・フォーマットの BFILE で、データベースのキャラクタ・セットが UTF8 であることを想定しています。

```
DECLARE
  src_loc     bfile := bfilename('ADVERT_DIR','ad_source_1000') ;
  dst_loc     clob  ;
  amt         number := dbms_lob.lobmaxsize;
  src_offset  number := 1 ;
  dst_offset  number := 1 ;
  lang_ctx    number := dbms_lob.default_lang_ctx;
  warning     number;
BEGIN
  select ad_sourcetext into dst_loc from Print_media
     where product_id = 3000 and ad_id = 1000 for update ;
  dbms_lob.fileopen(src_loc, dbms_lob.file_readonly);

  /* The default_csid can be used when the BFILE encoding is in the same charset
   * as the destination CLOB/NCLOB charset
   */
  dbms_lob.LOADCLOBFROMFILE(dst_loc,src_loc, amt, dst_offset, src_offset,
       dbms_lob.default_csid, lang_ctx,warning) ;

  commit;

  dbms_output.put_line(' Amount specified ' || amt ) ;
```

```
  dbms_output.put_line(' Number of bytes read from source: ' ||
      (src_offset-1));
  dbms_output.put_line(' Number of characters written to destination: ' ||
      (dst_offset-1) );
  if (warning = dbms_lob.warn_inconvertible_char)
  then
    dbms_output.put_line('Warning: Inconvertible character');
  end if;

  dbms_lob.filecloseall() ;
END ;
```

例

次の項目の例を示します。

- NLS_CHARSET_ID ファンクションを使用して、キャラクタ・セット名からキャラクタ・セット ID を取得する方法
- 戻されたオフセット値および言語コンテキスト lang_ctx を使用して、1 つの BFILE から複数の LOB にストリームのデータをロードする方法
- 警告メッセージの読込み方法

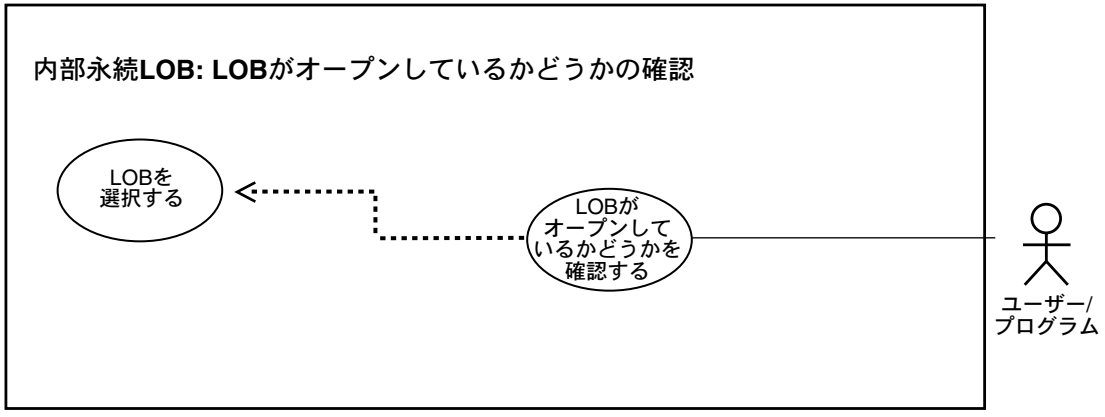
この例では、ad_file_ext_01 が JA16TSTSET フォーマットの BFILE で、データベースの各国語キャラクタ・セットが AL16UTF16 であることを想定しています。

```
DECLARE
  src_loc     bfile := bfilename('ADVERT_DIR','ad_file_ext_01') ;
  dst_loc1    nclob;
  dst_loc2    nclob;
  amt         number := 1000;
  src_offset  number := 1;
  dst_offset  number := 1;
  src_osin    number;
  cs_id       number := NLS_CHARSET_ID('JA16TSTSET'); /* 998 */
  lang_ctx    number := dbms_lob.default_lang_ctx;
  warning     number;
BEGIN
  dbms_lob.fileopen(src_loc, dbms_lob.file_readonly);
  dbms_output.put_line(' BFILE csid is ' || cs_id) ;

  /* Load the first 1KB of the BFILE into dst_loc1 */
  dbms_output.put_line(' ----------------------------' ) ;
  dbms_output.put_line('   First load  ' ) ;
  dbms_output.put_line(' ----------------------------' ) ;

  SELECT ad_fltextn INTO dst_loc1 FROM Print_media
```

```
    WHERE product_id=3106 and ad_id=13000 FOR UPDATE;

  dbms_lob.LOADCLOBFROMFILE(dst_loc1, src_loc, amt, dst_offset, src_offset,
    cs_id, lang_ctx, warning);
  commit;

  /* the number bytes read may or may not be 1k */
  dbms_output.put_line(' Amount specified ' || amt ) ;
  dbms_output.put_line(' Number of bytes read from source: ' ||
    (src_offset-1));
  dbms_output.put_line(' Number of characters written to destination: ' ||
    (dst_offset-1) );
  if (warning = dbms_lob.warn_inconvertible_char)
  then
    dbms_output.put_line('Warning: Inconvertible character');
  end if;

  /* load the next 1KB of the BFILE into the dst_loc2 */
  dbms_output.put_line(' ----------------------------' ) ;
  dbms_output.put_line('   Second load  ' ) ;
  dbms_output.put_line(' ----------------------------' ) ;

  SELECT ad_fltextn INTO dst_loc2 FROM Print_media
    WHERE product_id=3106 and ad_id=13001 FOR UPDATE;

  /* Notice we are using the src_offset and lang_ctx returned from the previous
   * load. We do not use value 1001 as the src_offset here because sometimes the
   * actual amount read may not be the same as the amount specified.
   */

  src_osin := src_offset;
  dst_offset := 1;
  dbms_lob.LOADCLOBFROMFILE(dst_loc2, src_loc, amt, dst_offset, src_offset,
    cs_id, lang_ctx, warning);
  commit ;
  dbms_output.put_line(' Number of bytes read from source: ' ||
    (src_offset-src_osin) );
  dbms_output.put_line(' Number of characters written to destination: ' ||
    (dst_offset-1) );
  if (warning = dbms_lob.warn_inconvertible_char)
  then
    dbms_output.put_line('Warning: Inconvertible character');
  end if;

  dbms_lob.filecloseall() ;

END ;
```

オープン : LOB がオープンしているかどうかの確認

図 10-10 利用図 : LOB がオープンしているかどうかの確認



参照 : 10-2 ページの表 10-1「利用モデル : 内部永続 LOB」を参照してください。

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関数またはファンクションのリストは、第 3 章「様々なプログラム環境での LOB のサポート」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）:『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』の第 23 章「DBMS_LOB」の「DBMS_LOB 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）:『Oracle Call Interface プログラマーズ・ガイド』の第 16 章「その他の OCI リレーショナル関数」の「LOB 関数」の OCILobIsOpen()
- C++（OCCI）:『Oracle C++ Call Interface プログラマーズ・ガイド』

- COBOL（Pro*COBOL）：『Pro*COBOL Precompiler プログラマーズ・ガイド』の LOB に関する詳細、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・ディレクティブ」の「LOB DESCRIBE（実行可能埋込み SQL 拡張機能）」
- C/C++（Pro*C/C++）：『Pro*C/C++ Precompiler プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびディレクティブ」の「LOB DESCRIBE（実行可能埋込み SQL 拡張機能）」
- Visual Basic（OO4O）：参照マニュアルはありません。
- Java（JDBC）：『Oracle9*i* JDBC 開発者ガイドおよびリファレンス』の第 8 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『Oracle9*i* 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「JDBC 2.0 LOB 型と Oracle 拡張型のサポート」の「BLOB、CLOB および BFILE のサポート」

使用例

次の例では、図形イメージ（ad_composite）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での例を示します。

- PL/SQL（DBMS_LOB）: LOB がオープンしているかどうかの確認（10-52 ページ）
- C（OCI）: LOB がオープンしているかどうかの確認（10-53 ページ）
- C++（OCCI）: 今回のリリースでは例は提供されません。
- COBOL（Pro*COBOL）: LOB がオープンしているかどうかの確認（10-54 ページ）
- C/C++（Pro*C/C++）: LOB がオープンしているかどうかの確認（10-56 ページ）
- Visual Basic（OO4O）: 今回のリリースでは例は提供されません。
- Java（JDBC）: LOB がオープンしているかどうかの確認（10-57 ページ）

PL/SQL（DBMS_LOB）: LOB がオープンしているかどうかの確認

```
/* Checking if a LOB is Open. Note that the example procedure lobIsOpen_proc
is not part of the DBMS_LOB package: */
CREATE OR REPLACE PROCEDURE lobIsOpen_proc IS
   Lob_loc     BLOB;
   Retval      INTEGER;
BEGIN
   SELECT ad_composite INTO Lob_loc FROM Print_media
       WHERE product_id = 3106 AND ad_id = 13001;
```

```
   /* Opening the LOB is optional: */
   DBMS_LOB.OPEN (Lob_loc , DBMS_LOB.LOB_READONLY);

   /* See if the LOB is open: */
   Retval := DBMS_LOB.ISOPEN(Lob_loc);
  /* The value of Retval will be 1 meaning that the LOB is open. */
END;
```

C（OCI）: LOB がオープンしているかどうかの確認

```
/* Checking if LOB is Open. */
/* Select the locator into a locator variable */
sb4 select_adcomp_locator(Lob_loc, errhp, svchp, stmthp)
OCILobLocator *Lob_loc;
OCIError      *errhp;
OCISvcCtx     *svchp;
OCIStmt       *stmthp;
{
  text      *sqlstmt =
    (text *)"SELECT ad_composite FROM Print_media
               WHERE product_id=2268 AND ad_id = 21001";
  OCIDefine *defnp1 *defnp2;
  checkerr (errhp, OCIStmtPrepare(stmthp, errhp, sqlstmt,
                                  (ub4)strlen((char *)sqlstmt),
                                  (ub4) OCI_NTV_SYNTAX, (ub4) OCI_DEFAULT));
  checkerr (errhp, OCIDefineByPos(stmthp, &defnp1, errhp, (ub4) 1,
                                  (dvoid *)&Lob_loc, (sb4)0,
                                  (ub2) SQLT_BLOB,(dvoid *) 0,
                                  (ub2 *) 0, (ub2 *) 0, (ub4) OCI_DEFAULT)
                   OCIDefineByPos(stmthp, &defnp2, errhp, (ub4) 2,
                                  (dvoid *)&Lob_loc, (sb4)0,
                                  (ub2) SQLT_BLOB,(dvoid *) 0,
                                  (ub2 *) 0, (ub2 *) 0, (ub4) OCI_DEFAULT));
  /* Execute the select and fetch one row */
  checkerr(errhp, OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                                 (CONST OCISnapshot*) 0, (OCISnapshot*) 0,
                                 (ub4) OCI_DEFAULT));
    return (0);
}

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

```
OCILobLocator *Lob_loc;
int isOpen;

/* Allocate locator resources */
(void) OCIDescriptorAlloc((dvoid *)envhp, (dvoid **)&Lob_loc,
                          (ub4)OCI_DTYPE_LOB, (size_t)0, (dvoid **)0);
/* Select the locator */
(void)select_adcomp_locator(Lob_loc, errhp, svchp, stmthp);

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

if (isOpen)
{
  printf("  Lob is Open\n");
  /* ... Processing given that the LOB has already been Opened */
}
else
{
  printf("  Lob is not Open\n");
  /* ... Processing given that the LOB has not been Opened */
}
/* Free resources held by the locators*/
(void) OCIDescriptorFree((dvoid *)  Lob_loc, (ub4) OCI_DTYPE_LOB);

return;
}
```

COBOL（Pro*COBOL）: LOB がオープンしているかどうかの確認

```
      * Checking if LOB is Open
        IDENTIFICATION DIVISION.
        PROGRAM-ID. LOB-OPEN.
        ENVIRONMENT DIVISION.
        DATA DIVISION.
        WORKING-STORAGE SECTION.

        01  BLOB1          SQL-BLOB.
        01  LOB-ATTR-GRP.
            05 ISOPN       PIC S9(9) COMP.
        01  SRC            SQL-BFILE.
        01  DIR-ALIAS      PIC X(30) VARYING.
        01  FNAME          PIC X(20) VARYING.
        01  DIR-IND        PIC S9(4) COMP.
        01  FNAME-IND      PIC S9(4) COMP.
        01  USERID   PIC X(11) VALUES "PM/PM".
```

```
     EXEC SQL INCLUDE SQLCA END-EXEC.

 PROCEDURE DIVISION.
 LOB-OPEN.
     EXEC SQL WHENEVER SQLERROR DO PERFORM SQL-ERROR END-EXEC.
     EXEC SQL CONNECT :USERID END-EXEC.

* Allocate and initialize the target BLOB
     EXEC SQL ALLOCATE :BLOB1 END-EXEC.
     EXEC SQL WHENEVER NOT FOUND GOTO END-OF-BLOB END-EXEC.
     EXEC SQL SELECT AD_COMPOSITE INTO :BLOB1
        FROM PRINT_MEDIA WHERE PRODUCT_ID = 3060 AND AD_ID = 11001 END-EXEC.

* See if the LOB is OPEN
     EXEC SQL
        LOB DESCRIBE :BLOB1 GET ISOPEN INTO :ISOPN END-EXEC.

     IF ISOPN = 1
*       <Processing for the LOB OPEN case>
        DISPLAY "The LOB is open"
     ELSE
*         <Processing for the LOB NOT OPEN case>
        DISPLAY "The LOB is not open"
     END-IF.

* Free the resources used by the BLOB
 END-OF-BLOB.
     EXEC SQL FREE :BLOB1 END-EXEC.
     EXEC SQL ROLLBACK WORK RELEASE END-EXEC.
     STOP RUN.

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

C/C++（Pro*C/C++）: LOB がオープンしているかどうかの確認

```
/* Checking if LOB is open */
#include <oci.h>
#include <stdio.h>
#include <sqlca.h>

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

void seeIfLOBIsOpen()
{
  OCIBlobLocator *Lob_loc;
  int isOpen = 1;

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  EXEC SQL ALLOCATE :Lob_loc;
  EXEC SQL SELECT ad_composite INTO :Lob_loc
           FROM Print_media WHERE product_id = 3106 and ad_id = 13001;
  /* See if the LOB is Open: */
  EXEC SQL LOB DESCRIBE :Lob_loc GET ISOPEN INTO :isOpen;
  if (isOpen)
    printf("LOB is open\n");
  else
    printf("LOB is not open\n");
  /* Note that in this example, the LOB is not open */
  EXEC SQL FREE :Lob_loc;
}

void main()
{
  char *samp = "pm/pm";
  EXEC SQL CONNECT :pm;
  seeIfLOBIsOpen();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

Java（JDBC）: LOB がオープンしているかどうかの確認

CLOB がオープンしているかどうかの確認

JDBC アプリケーションで CLOB がオープンしているかどうかを確認するには、oracle.sql.CLOB で定義されている isOpen メソッドを使用します。Boolean 型の戻り値によって、CLOB がすでにオープンしているかどうかを確認します。isOpen メソッドは次のように定義します。

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

次に例を示します。

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

BLOB がオープンしているかどうかの確認

JDBC アプリケーションで BLOB がオープンしているかどうかを確認するには、oracle.sql.BLOB で定義されている isOpen メソッドを使用します。Boolean 型の戻り値によって、BLOB がすでにオープンしているかどうかを確認します。isOpen メソッドは次のように定義します。

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

次に例を示します。

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

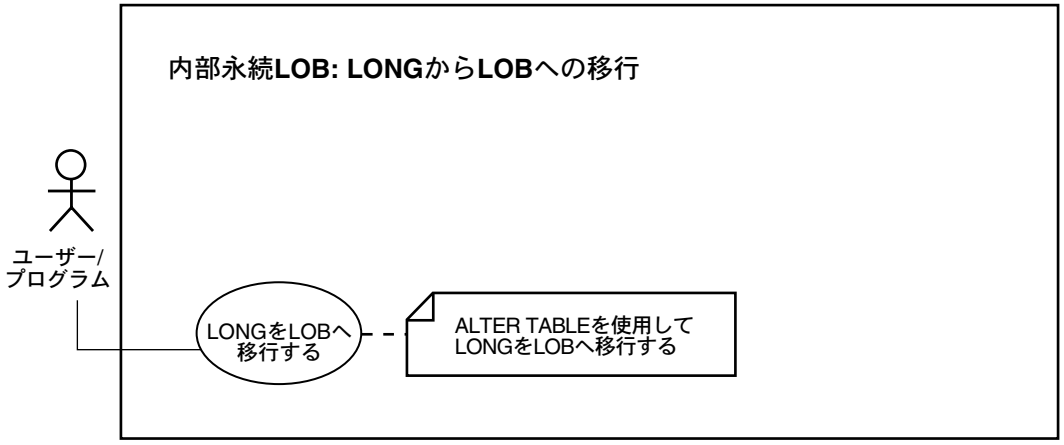
例

```
// Checking if LOB is open
// Core JDBC classes:
import java.io.OutputStream;
import java.sql.DriverManager;
import java.sql.Connection;
import java.sql.Statement;
import java.sql.Types;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

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

public class Ex2_48
{
  public Ex2_48 ()
  {
  }

  public static void main (String args [])
       throws Exception
  {
    // Load the Oracle JDBC driver:
    DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
    // Connect to the database:
    Connection conn =
      DriverManager.getConnection ("jdbc:oracle:oci8:@", "pm", "pm");
    // It's faster when auto commit is off:
    conn.setAutoCommit (false);
    // Create a Statement:
    Statement stmt = conn.createStatement ();

    try
    {
      BLOB blob = null;
      ResultSet rset = stmt.executeQuery (
          "SELECT ad_composite FROM Print_media product_id = 3060 AND ad_id =
11001");
   if (rset.next())
   {
     blob = ((OracleResultSet)rset).getBLOB (1);
   }
       OracleCallableStatement cstmt =
```

```
            (OracleCallableStatement) conn.prepareCall (
            "BEGIN ? := DBMS_LOB.ISOPEN(?); END;");
    cstmt.registerOutParameter (1, Types.NUMERIC);
    cstmt.setBLOB(2, blob);
    cstmt.execute();
    int result = cstmt.getInt(1);
    System.out.println("The result is: " + Integer.toString(result));

    OracleCallableStatement cstmt2 = (OracleCallableStatement)
        conn.prepareCall (
        "BEGIN DBMS_LOB.OPEN(?,DBMS_LOB.LOB_READONLY); END;");
    cstmt2.setBLOB(1, blob);
    cstmt2.execute();

    System.out.println("The LOB has been opened with a call to DBMS_LOB.OPEN()");

    // Use the existing cstmt handle to re-query the status of the locator:
    cstmt.setBLOB(2, blob);
    cstmt.execute();
    result = cstmt.getInt(1);
    System.out.println("This result is: " + Integer.toString(result));

    stmt.close();
    cstmt.close();
    cstmt2.close();
    conn.commit();
    conn.close();
     }
     catch (SQLException e)
     {
       e.printStackTrace();
     }
   }
}
```

LONG-to-LOB API を使用した LONG から LOB への移行

図 10-11 利用図 : LONG-to-LOB API（新機能）を使用した LONG から LOB への移行



参照 :

- 10-2 ページの表 10-1「利用モデル : 内部永続 LOB」を参照してください。
- 10-63 ページの「TO_LOB 演算子を使用した LONG から LOB へのコピー」を参照してください。

用途

LONG-to-LOB API（新機能）を使用して、LONG を LOB へ移行します。

使用上の注意

参照 : LONG-to-LOB API の使用については、第 8 章「LONG から LOB への移行」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:『Oracle9*i* SQL リファレンス』の第 6 章「ファンクション」

使用例

例に使用するフィールドを次に示します。

```
CREATE TABLE Print_media (
   product_id      NUMBER NOT NULL,
   ad_id           NUMBER NOT NULL,
   ad_sourcetext   CLOB default EMPTY_CLOB(),
   ad_fltextn      NCLOB default EMPTY_CLOB(),
   ad_graphic      BFILE default NULL,
   ad_composite    BLOB default EMPTY_BLOB(),
   ad_photo        BLOB default EMPTY_BLOB(),
   ad_graphic      BFILE default NULL,
 );
```

この例では、PRINT_MEDIA 表の ad_sourcetext 列が次のように作成されていると想定します。

```
CREATE TABLE Print_media (
    ...
    ad_sourcetext  LONG,
    ...
);
```

ALTER TABLE を使用した LONG から CLOB への変換

LONG 列を CLOB に変換するには、次のとおり ALTER TABLE を使用します。

```
ALTER TABLE Print_media MODIFY ( ad_sourcetext CLOB );
```

Print_media 表を使用した既存のアプリケーションはすべて、ad_sourcetext 列を CLOB 型に変更した後も、ほとんど変更することなく使用できます。LONG で使用していて、わずかな変更で LOB でも使用できる操作（バインドおよび定義）の例については、第 8 章「LONG から LOB への移行」を参照してください。

例

OCI で LONG-to-LOB API を使用する方法を次に示します。

- C（OCI）: LONG から LOB への移行

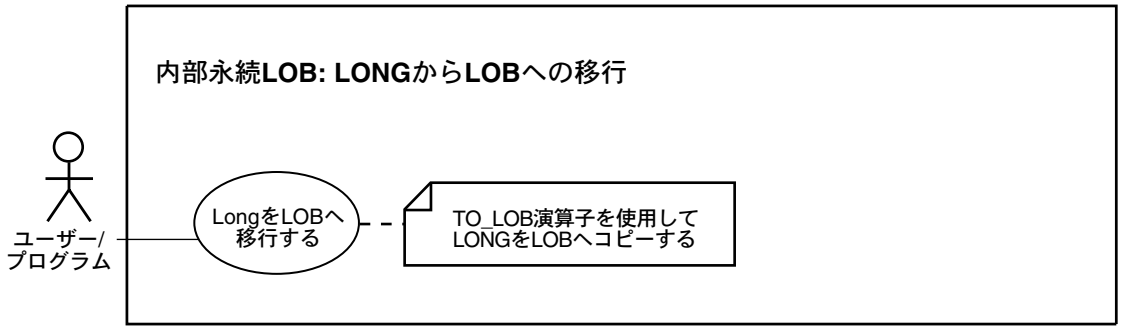
C（OCI）: LONG から LOB への移行

```
/* Migrating from LONG to LOB Using LONG-to-LOB API */

word buflen, pid, aid;
text buf2[5000];
text *insstmt = (text *)
"INSERT INTO Print_media(product_id, ad_id, ad_sourcetext) VALUES
```

```
(:PID, :AID, :ADSOURCE)";

if (OCIStmtPrepare(stmthp, errhp, insstmt,
(ub4)strlen((char *)insstmt), (ub4) OCI_NTV_SYNTAX,
(ub4) OCI_DEFAULT))
{
    DISCARD printf("FAILED: OCIStmtPrepare()\n");
    report_error(errhp);
    return;
}

if (OCIBindByName(stmthp, &bndhp[0], errhp,
  (text *) ":PID", (sb4) strlen((char *) ":PID"),
  (dvoid *) &buf1, (sb4) sizeof(buf1), SQLT_INT,
  (dvoid *) 0, (ub2 *) 0, (ub2 *) 0,
  (ub4) 0, (ub4 *) 0, (ub4) OCI_DEFAULT)
  || OCIBindByName(stmthp, &bndhp[1], errhp,
  (text *) ":AID", (sb4) strlen((char *) ":AID"),
  (dvoid *) &buf1, (sb4) sizeof(buf1), SQLT_INT,
  (dvoid *) 0, (ub2 *) 0, (ub2 *) 1,
  (ub4) 0, (ub4 *) 0, (ub4) OCI_DEFAULT)
  ||  OCIBindByName(stmthp, &bndhp[1], errhp,
  (text *) ":ADSOURCE", (sb4) strlen((char *) ":ADSOURCE"),
  (dvoid *) buf2, (sb4) sizeof(buf2), SQLT_CHR,
  (dvoid *) 0, (ub2 *) 0, (ub2 *) 3,
  (ub4) 0, (ub4 *) 0, (ub4) OCI_DEFAULT))
{
    DISCARD printf("FAILED: OCIBindByName()\n");
    report_error(errhp);
    return;
}

buf1 = 101;
memset((void *)buf2, (int)'A', (size_t)5000);

if (OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
           (const OCISnapshot*) 0, (OCISnapshot*) 0,
           (ub4) OCI_DEFAULT))
{
      DISCARD printf("FAILED: OCIStmtExecute()\n");
      report_error(errhp);
      return;
}
```

TO_LOB 演算子を使用した LONG から LOB へのコピー

図 10-12 利用図 : TO_LOB 演算子を使用した LONG から LOB へのコピー



参照 :

- 10-2 ページの表 10-1「利用モデル : 内部永続 LOB」を参照してください。
- 10-60 ページの「LONG-to-LOB API を使用した LONG から LOB への移行」を参照してください。

用途

TO_LOB 演算子を使用して、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正常に実行されません。

```
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;
```

- 索引構成表の作成時に、CREATE TABLE AS SELECT 文で TO_LOB 演算子を使用して、LONG または LONG RAW 列を LOB 列に変換することはできません。

  かわりに、索引構成表の作成後、TO_LOB 演算子を使用して LONG または LONG RAW 列の INSERT AS SELECT を実行します。

- 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 ファンクション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:『Oracle9*i* SQL リファレンス』の第 6 章「ファンクション」の TO_LOB

使用例

次のアーカイブ・ソース表 adlibrary_tab が定義され、データを含んでいるとします。

```
Rem  This script is not needed if you use the (newer)
Rem  ALTER TABLE when migrating LONGs to LOBs
CREATE TABLE adlibrary_tab
(
   product_id    NUMBER(6),
   ad_id         NUMBER(6),
   photos        LONG RAW
);
```

次の例では、データを Print_media 表の LONG RAW 列（photos）から BLOB 列（ad_photo）にコピーし、SQL ファンクション TO_LOB を使用してこれを実行することを想定しています。

例

例を SQL で示します。この例は、すべてのプログラム環境に適用されます。

- 「SQL: TO_LOB 演算子を使用した LONG から LOB へのコピー」

SQL: TO_LOB 演算子を使用した LONG から LOB へのコピー

```
INSERT INTO Print_media (product_id, ad_id, ad_photo)
 SELECT product_id, TO_LOB(photos)
  FROM adlibrary_tab WHERE product_id =3106;
```

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 adlibrary_tab (
  product_id     NUMBER,
  ad_audience  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。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薦めします。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す。

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。

次の手順を実行して、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。

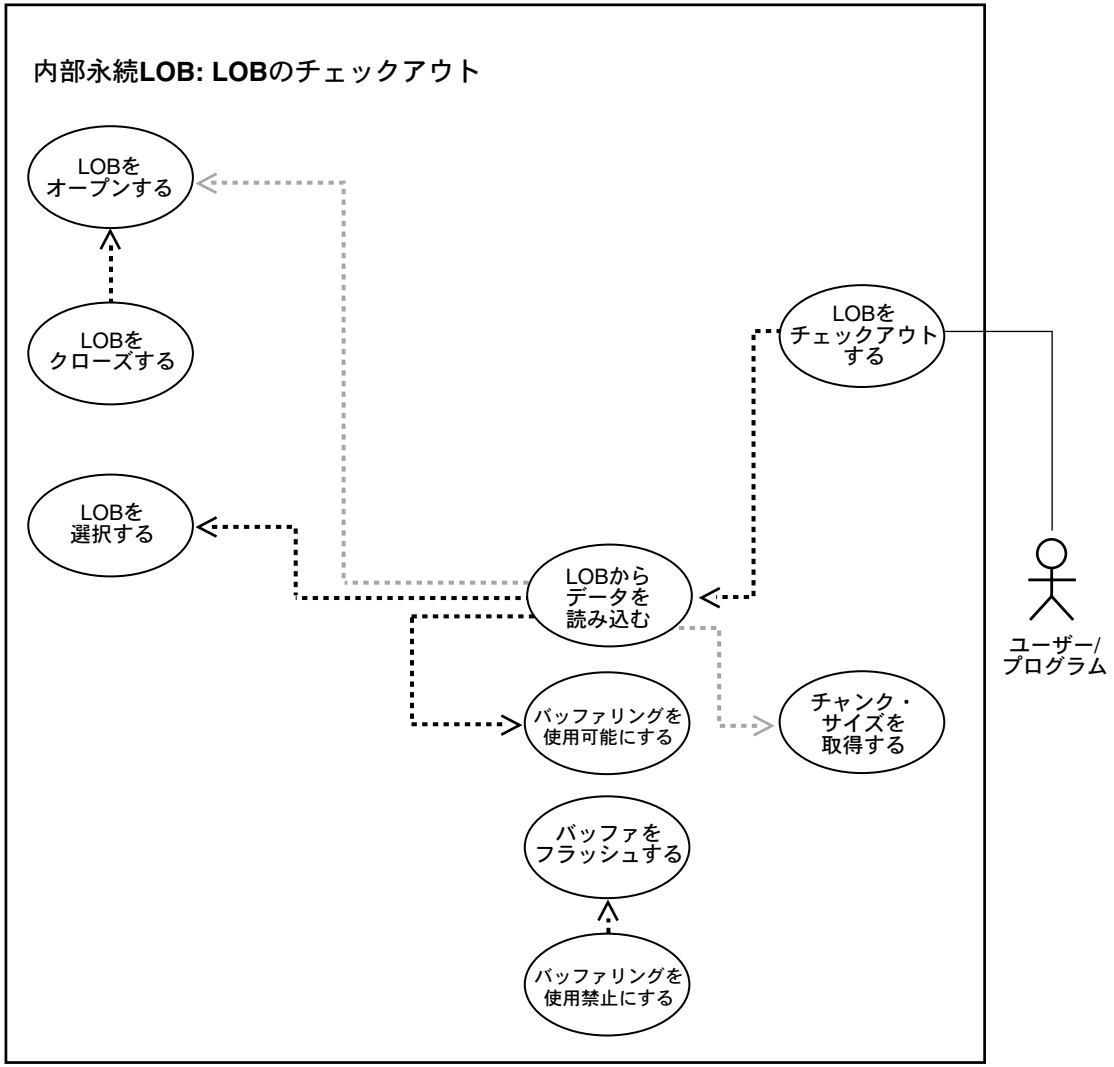
```
CREATE SYNONYM Long_tab FOR Lob_tab;
```

コピーの完了後、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 10-13 利用図 : LOB のチェックアウト



参照 : 10-2 ページの表 10-1「利用モデル : 内部永続 LOB」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には、ポーリングまたはコールバックによってストリーム・メカニズムを有効にした `OCILobRead()` を使用します。基礎となる読込み操作を行うには、ストリームを伴う OCI、OCCI 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関数またはファンクションのリストは、第 3 章「様々なプログラム環境での LOB のサポート」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）：『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』の第 23 章「DBMS_LOB」の「DBMS_LOB 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」および「CLOSE プロシージャ」
- C（OCI）：『Oracle Call Interface プログラマーズ・ガイド』の第 16 章「その他の OCI リレーショナル関数」の「LOB 関数」の OCILobOpen()、OCILobRead() および OCILobClose()
- C++（OCCI）：『Oracle C++ Call Interface プログラマーズ・ガイド』
- COBOL（Pro*COBOL）：『Pro*COBOL Precompiler プログラマーズ・ガイド』の LOB に関する詳細、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・ディレクティブ」の「LOB READ（実行可能埋込み SQL 拡張機能）」
- C/C++（Pro*C/C++）：『Pro*C/C++ Precompiler プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびディレクティブ」の「LOB OPEN（実行可能埋込み SQL 拡張機能）」および「LOB READ（実行可能埋込み SQL 拡張機能）」
- Visual Basic（OO4O オンライン・ヘルプ）：ヘルプの「目次」タブから、「OO4O オートメーション・サーバー・リファレンス」>「OO4O サーバーのオブジェクト」>「OraBLOB、OraCLOB」>「メソッド」>「read」、「copytofile」を選択
- Java（JDBC）：『Oracle9*i* JDBC 開発者ガイドおよびリファレンス』の第 8 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『Oracle9*i* 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「JDBC 2.0 LOB 型と Oracle 拡張型のサポート」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。使用例のチェックイン部分の詳細は、10-78 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」に示す例と似ています。次のプログラム環境での例を示します。

- PL/SQL（DBMS_LOB）: LOB のチェックアウト（10-69 ページ）
- C（OCI）: LOB のチェックアウト（10-70 ページ）
- C++（OCCI）: 今回のリリースでは例は提供されません。
- COBOL（Pro*COBOL）: LOB のチェックアウト（10-72 ページ）
- C/C++（Pro*C/C++）: LOB のチェックアウト（10-74 ページ）
- Visual Basic（OO4O）: LOB のチェックアウト（10-75 ページ）
- Java（JDBC）: LOB のチェックアウト（10-76 ページ）

PL/SQL（DBMS_LOB）: LOB のチェックアウト

```
/* Checking out a LOB. The procedure checkOutLOB_proc used here is not part of the
   DBMS_LOB package: */
CREATE OR REPLACE PROCEDURE checkOutLOB_proc IS
    Lob_loc     BLOB;
    Buffer      VARCHAR2(32767);
    Amount      BINARY_INTEGER := 32767;
    Position    INTEGER := 2147483647;
BEGIN
   /* Select the LOB: */
   SELECT Nesttab.formatted_doc INTO Lob_loc
       FROM TABLE(SELECT PMtab.Textdoc_ntab FROM Print_media PMtab
          WHERE PMtab.product_id = 3106 AND PMtab.ad_id = 13001) Nesttab
             WHERE Nesttab.document_typ = 'PDF';
   /* Opening the LOB is optional: */
   DBMS_LOB.OPEN (Lob_loc, DBMS_LOB.LOB_READONLY);
   LOOP
      DBMS_LOB.READ (Lob_loc, Amount, Position, Buffer);
      /* Process the buffer: */
      Position := Position + Amount;
   END LOOP;
   /* Closing the LOB is mandatory if you have opened it: */
```

```
   DBMS_LOB.CLOSE (Lob_loc);
EXCEPTION
   WHEN NO_DATA_FOUND THEN
      DBMS_OUTPUT.PUT_LINE('End of data');
END;
```

C（OCI）: LOB のチェックアウト

```
/* Checking out a lOB -- This example reads the entire contents of a BLOB
   piecewise into a buffer using a standard polling method, processing each
   buffer piece after every READ operation until the entire BLOB has been read: */

#define MAXBUFLEN 32767
/* Select the locator into a locator variable: */
sb4 select_formatteddoc_locator(Lob_loc, errhp, stmthp, svchp)
OCILobLocator *Lob_loc;
OCIError      *errhp;
OCISvcCtx     *svchp;
OCIStmt       *stmthp;
{
  text  *sqlstmt =
     (text *) "SELECT Intab.Transcript \
           FROM TABLE(SELECT pm.ad_textdoc_ntab FROM Print_media pm  \
              WHERE pm.product_id = 3060 AND pm.ad_id = 11001) ntab \
                 WHERE ntab.document_typ = 'PDF'";
 OCIDefine *defnp1, *defnp2;
 checkerr (errhp, OCIStmtPrepare(stmthp, errhp, sqlstmt,
                                 (ub4)strlen((char *)sqlstmt),
                                 (ub4) OCI_NTV_SYNTAX, (ub4) OCI_DEFAULT));
 checkerr (errhp, OCIDefineByPos(stmthp, &defnp1, errhp, (ub4) 1,
                                 (dvoid *)&Lob_loc, (sb4)0,
                                 (ub2) SQLT_CLOB,(dvoid *) 0,
                                 (ub2 *) 0, (ub2 *) 0, (ub4) OCI_DEFAULT)
                || OCIDefineByPos(stmthp, &defnp2, errhp, (ub4) 2,
                                 (dvoid *)&Lob_loc, (sb4)0,
                                 (ub2) SQLT_CLOB,(dvoid *) 0,
                                 (ub2 *) 0, (ub2 *) 0, (ub4) OCI_DEFAULT));
 /* Execute the select and fetch one row: */
 checkerr(errhp, OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                               (CONST OCISnapshot*) 0, (OCISnapshot*) 0,
                               (ub4) OCI_DEFAULT));
 return 0;
}

void checkoutLob(envhp, errhp, svchp, stmthp)
OCIEnv    *envhp;
```

```
OCIError  *errhp;
OCISvcCtx *svchp;
OCIStmt   *stmthp;
{
  OCILobLocator *Lob_loc;
  ub4 amt;
  ub4 offset;
  sword retval;
  boolean done;
  ub1 bufp[MAXBUFLEN];
  ub4 buflen;

  /* Allocate locators desriptors: */
  (void) OCIDescriptorAlloc((dvoid *)envhp, (dvoid **) &Lob_loc,
                            (ub4)OCI_DTYPE_LOB,(size_t) 0, (dvoid **) 0);
  /* Select the BLOB: */
  printf(" select the formatted_doc locator...\n");
  select_formatteddoc_locator(Lob_loc, errhp, stmthp, svchp);


  /* Open the CLOB: */
  printf (" open lob in checkOutLOB_proc\n");
  checkerr (errhp, (OCILobOpen(svchp, errhp, Lob_loc, OCI_LOB_READONLY)));

  /* Setting amt = 0 will read till the end of LOB: */
  amt = 0;
  buflen = sizeof(bufp);

 /* Process the data in pieces: */
  printf (" read lob in pieces\n");
 offset = 1;
 memset(bufp, '\0', MAXBUFLEN);
 done = FALSE;
 while (!done)
   {
     retval = OCILobRead(svchp, errhp, Lob_loc, &amt, offset, (dvoid *)bufp,
                         buflen, (dvoid *)0,(sb4 (*)(dvoid *, dvoid *, ub4,
                         ub1)) 0, (ub2) 0, (ub1) SQLCS_IMPLICIT);
     switch (retval)
       {
       case OCI_SUCCESS:              /* Only one piece or last piece */
    /* Process the data in bufp. amt will give the amount of data just read in
       bufp. This is in bytes for BLOBs and in characters for fixed
       width CLOBS and in bytes for variable width CLOBs */
    done = TRUE;
    break;
       case OCI_ERROR:
```

```
        checkerr (errhp, OCI_ERROR);
    done = TRUE;
    break;
       case OCI_NEED_DATA:           /* There are 2 or more pieces */
    /* Process the data in bufp. amt will give the amount of data just read in
       bufp. This is in bytes for BLOBs and in characters for fixed
       width CLOBS and in bytes for variable width CLOBs. */
    break;
       default:
         checkerr (errhp, retval);
    done = TRUE;
       break;
       } /* while */
   }
  /* Closing the CLOB is mandatory if you have opened it: */
  printf (" close lob in checkOutLOB_proc\n");
  checkerr (errhp, OCILobClose(svchp, errhp, Lob_loc));

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

  return;
}
```

COBOL（Pro*COBOL）: LOB のチェックアウト

```
*  Checking out a LOB
   IDENTIFICATION DIVISION.
   PROGRAM-ID. CHECKOUT.
   ENVIRONMENT DIVISION.
   DATA DIVISION.
   WORKING-STORAGE SECTION.

   01  USERID   PIC X(11) VALUES "PM/PM".
   01  CLOB1          SQL-CLOB.
   01  BUFFER         PIC X(5) VARYING.
   01  AMT            PIC S9(9) COMP.
   01  OFFSET         PIC S9(9) COMP VALUE 1.
   01  D-BUFFER-LEN   PIC 9.
   01  D-AMT          PIC 9.
       EXEC SQL INCLUDE SQLCA END-EXEC.

   PROCEDURE DIVISION.
   READ-CLOB.
       EXEC SQL WHENEVER SQLERROR DO PERFORM SQL-ERROR END-EXEC.
       EXEC SQL
```

```
      CONNECT :USERID
    END-EXEC.
* Allocate and initialize the CLOB locator:
    EXEC SQL ALLOCATE :CLOB1 END-EXEC.
    EXEC SQL WHENEVER NOT FOUND GOTO END-OF-CLOB END-EXEC.
    EXEC SQL
      SELECT AD_SOURCETEXT INTO :CLOB1 FROM PRINT_MEDIA
         WHERE PRODUCT_ID = 3060 AND AD_ID = 11001
    END-EXEC.

* Initiate polling read:
    MOVE 0 TO AMT.
* Read first piece of the CLOB into the buffer:
    EXEC SQL
      LOB READ :AMT FROM :CLOB1 AT :OFFSET INTO :BUFFER
    END-EXEC.
    DISPLAY "Reading a CLOB ...".
    DISPLAY " ".
    MOVE BUFFER-LEN TO D-BUFFER-LEN.
    DISPLAY "first read (", D-BUFFER-LEN, "): "
       BUFFER-ARR(1:BUFFER-LEN).

* Read subsequent pieces of the CLOB:
 READ-LOOP.
    MOVE "     " TO BUFFER-ARR.
    EXEC SQL
      LOB READ :AMT FROM :CLOB1 INTO :BUFFER
    END-EXEC.
    MOVE BUFFER-LEN TO D-BUFFER-LEN.
    DISPLAY "next read (", D-BUFFER-LEN, "): "
      BUFFER-ARR(1:BUFFER-LEN).

    GO TO READ-LOOP.

* Read the last piece of the CLOB:
 END-OF-CLOB.
    EXEC SQL WHENEVER NOT FOUND CONTINUE END-EXEC.
    EXEC SQL FREE :CLOB1 END-EXEC.
    MOVE BUFFER-LEN TO D-BUFFER-LEN.
    DISPLAY "last read (", D-BUFFER-LEN, "): "
      BUFFER-ARR(1:BUFFER-LEN).
    EXEC SQL
      ROLLBACK WORK RELEASE
    END-EXEC.
    STOP RUN.

 SQL-ERROR.
```

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

C/C++（Pro*C/C++）: LOB のチェックアウト

```
/* Checking out a LOB -
   This example reads the entire contents of a CLOB piecewise into a
   buffer using a standard polling method, processing each buffer piece
   after every READ operation until the entire CLOB has been read: */
#include <oci.h>
#include <stdio.h>
#include <sqlca.h>

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

#define BufferLength 256

void checkOutLOB_proc()
{
  OCIClobLocator *Lob_loc;
  int Amount;
  int Clip_ID, Segment;
  VARCHAR Buffer[BufferLength];

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  EXEC SQL ALLOCATE :Lob_loc;

  /* Use Dynamic SQL to retrieve the LOB: */
  EXEC SQL PREPARE S FROM
    'SELECT Intab. \
       FROM TABLE(SELECT PMtab.textdoc_ntab FROM Print_media PMtab \
```

```
          WHERE PMtab.product_id = :pid AND PMtab.ad_id = :aid) ntab \
             WHERE ntab.document_typ = :seg';
  EXEC SQL DECLARE C CURSOR FOR S;
  Clip_ID = Segment = 1;
  EXEC SQL OPEN C USING :PID, :AID, :Segment;
  EXEC SQL FETCH C INTO :Lob_loc;
  EXEC SQL CLOSE C;

  /* Open the LOB: */
  EXEC SQL LOB OPEN :Lob_loc READ ONLY;
  /* Setting Amount = 0 will initiate the polling method: */
  Amount = 0;
  /* Set the maximum size of the Buffer: */
  Buffer.len = BufferLength;
  EXEC SQL WHENEVER NOT FOUND DO break;
  while (TRUE)
    {
      /* Read a piece of the LOB into the Buffer: */
      EXEC SQL LOB READ :Amount FROM :Lob_loc INTO :Buffer;
      printf("Checkout %d characters\n", Buffer.len);
    }
  printf("Checkout %d characters\n", Amount);

  /* Closing the LOB is mandatory if you have opened it: */
  EXEC SQL LOB CLOSE :Lob_loc;
  EXEC SQL FREE :Lob_loc;
}

void main()
{
  char *pm = "pm/pm";
  EXEC SQL CONNECT :pm;
  checkOutLOB_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

Visual Basic（OO4O）: LOB のチェックアウト

```
'Checking out a lOB
'There are two ways of reading a lob using orablob.read or orablob.copytofile

'Using OraBlob.Read mechanism
Dim OraDyn as OraDynaset, OraPhoto as OraBlob, amount_read%, chunksize%, chunk

chunksize = 32767
set OraDyn = OraDb.CreateDynaset("select * from Print_media", ORADYN_DEFAULT)
```

```
set OraPhoto = OraDyn.Fields("ad_photo").Value
OraPhoto.PollingAmount = OraPhoto.Size 'Read entire BLOB contents
Do
    amount_read = OraPhoto.Read(chunk,chunksize) 'chunk returned is a variant of
type byte array
Loop Until OraPhoto.Status <> ORALOB_NEED_DATA

'Using OraBlob.CopyToFile mechanism
Set OraDyn = OraDb.CreateDynaset("select * from Print_media", ORADYN_DEFAULT)
Set OraPhoto = OraDyn.Fields("ad_photo").Value

'Read entire BLOB contents
OraPhoto.CopyToFile "c:\myphoto.jpg"
```

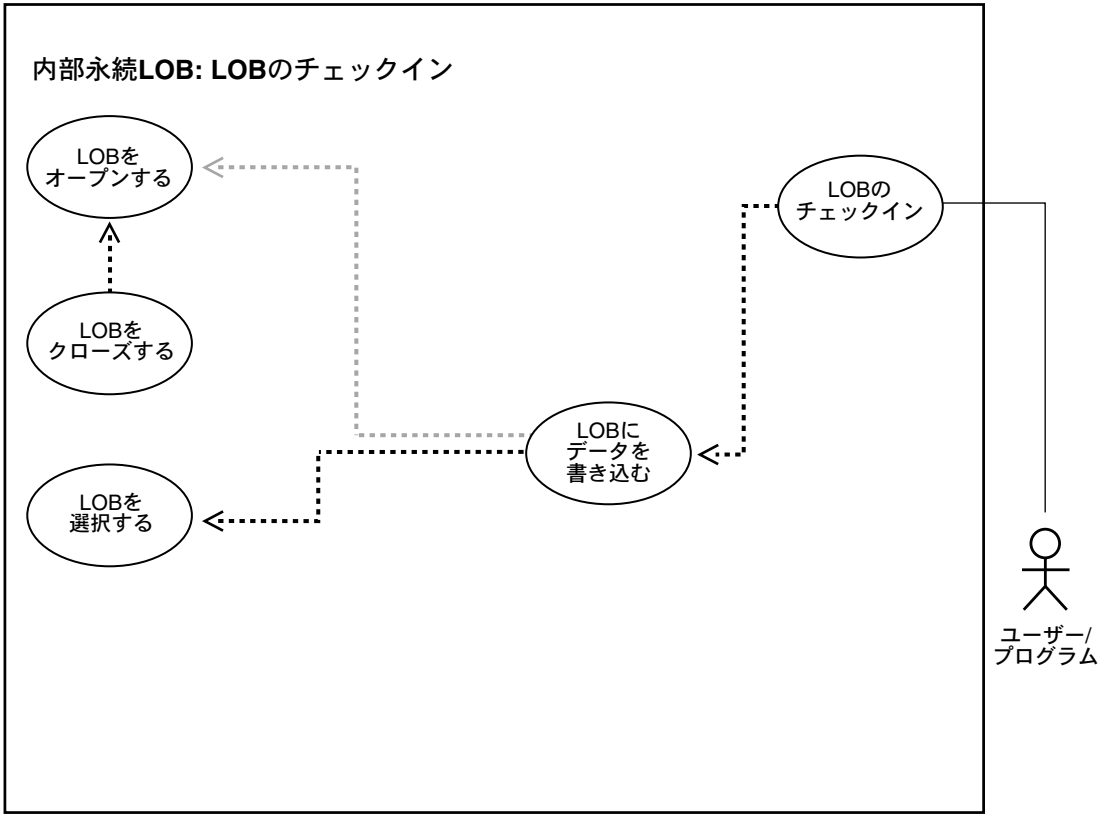
Java（JDBC）: LOB のチェックアウト

```
// Checking out a LOB
import java.io.InputStream;
import java.io.OutputStream;

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

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

public class Ex2_59
{

  static final int MAXBUFSIZE = 32767;
  public static void main (String args [])
     throws Exception
  {
    // Load the Oracle JDBC driver:
    DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
    // Connect to the database:
    Connection conn =
       DriverManager.getConnection ("jdbc:oracle:oci8:@", "samp", "samp");
    // It's faster when auto commit is off:
    conn.setAutoCommit (false);
```

```
    // Create a Statement:
    Statement stmt = conn.createStatement ();
    try
    {
       CLOB src_lob = null;
       InputStream in = null;
       byte buf[] = new byte[MAXBUFSIZE];

       ResultSet rset = stmt.executeQuery (
        "SELECT ntab.formatted_doc FROM TABLE( "
         +" SELECT pm.ad_textdoc_ntab FROM Print_media pm "
         +" WHERE pm.product_id=3060 AND ad_id = 11001) ntab
         +" WHERE ntab.document_typ='html'");
       if (rset.next())
       {
          src_lob = ((OracleResultSet)rset).getCLOB (1);
          in = src_lob.getAsciiStream();
   }

   int length = 0;
   int pos = 0;
   while ((in != null) && ((length = in.read(buf)) != -1))
   {
      pos += length;
      System.out.println(Integer.toString(pos));
      // Process the buffer:
   }

   in.close();
   rset.close();
   stmt.close();
   conn.commit();
   conn.close();
    }
    catch (SQLException e)
    {
       e.printStackTrace();
    }
  }
}
```

LOB のチェックイン

図 10-14 利用図 : LOB のチェックイン



参照 : 10-2 ページの表 10-1「利用モデル : 内部永続 LOB」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には、ポーリングまたはコールバックによってストリーム・メカニズムを有効にした `OCILobWrite()` を使用します。

構文

各プログラム環境で使用可能な関数またはファンクションのリストは、第 3 章「様々なプログラム環境での LOB のサポート」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）：『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』の第 23 章「DBMS_LOB」の「DBMS_LOB 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」および「CLOSE プロシージャ」
- C（OCI）：『Oracle Call Interface プログラマーズ・ガイド』の第 16 章「その他の OCI リレーショナル関数」の「LOB 関数」の OCILobOpen()、OCILobWrite() および OCILobClose()
- C++（OCCI）：『Oracle C++ Call Interface プログラマーズ・ガイド』
- COBOL（Pro*COBOL）：『Pro*COBOL Precompiler プログラマーズ・ガイド』の LOB に関する詳細、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・ディレクティブ」の「LOB WRITE（実行可能埋込み SQL 拡張機能）」
- C/C++（Pro*C/C++）：『Pro*C/C++ Precompiler プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびディレクティブ」の「LOB WRITE（実行可能埋込み SQL 拡張機能）」
- Visual Basic（OO4O オンライン・ヘルプ）：ヘルプの「目次」タブから、「OO4O オートメーション・サーバー・リファレンス」>「OO4O サーバーのオブジェクト」>「OraBLOB、OraCLOB」>「メソッド」>「write」、および「OO4O オートメーション・サーバー・リファレンス」>「OO4O サーバーのオブジェクト」>「OraDynaset」>「update」を選択
- Java（JDBC）：『Oracle9*i* JDBC 開発者ガイドおよびリファレンス』の第 8 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『Oracle9*i* 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「JDBC 2.0 LOB 型と Oracle 拡張型のサポート」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、10-67 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ネストした表 ad_textdoc_ntab の中の CLOB formatted_doc 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。

次の例では、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ます。

例

次のプログラム環境での例を示します。

- PL/SQL（DBMS_LOB）: LOB のチェックイン（10-80 ページ）
- C（OCI）: LOB のチェックイン（10-81 ページ）
- C++（OCCI）: 今回のリリースでは例は提供されません。
- COBOL（Pro*COBOL）: LOB のチェックイン（10-84 ページ）
- C/C++（Pro*C/C++）: LOB のチェックイン（10-87 ページ）
- Visual Basic（OO4O）: LOB のチェックイン（10-89 ページ）
- Java（JDBC）: LOB のチェックイン（10-90 ページ）

PL/SQL（DBMS_LOB）: LOB のチェックイン

```
/* Checking in a LOB. The example procedure checkInLOB_proc is not part of the
   DBMS_LOB package: */
CREATE OR REPLACE PROCEDURE checkInLOB_proc IS
    Lob_loc         BLOB;
    Buffer          VARCHAR2(32767);
    Amount          BINARY_INTEGER := 32767;
    Position        INTEGER := 2147483647;
    i               INTEGER;
BEGIN
    /* Select the LOB: */
    SELECT ntab.formatted_doc INTO Lob_loc
        FROM TABLE(SELECT PMtab.Textdoc_ntab FROM Print_media PMtab
                   WHERE PMtab.product_id = 3060 AND PMtab.ad_id = 11001) ntab
        WHERE ntab.document_typ = 'pdf' FOR UPDATE;
    /* Opening the LOB is optional: */
    DBMS_LOB.OPEN (Lob_loc, DBMS_LOB.LOB_READWRITE)
    FOR i IN 1..3 LOOP
        /* Fill the Buffer with data to be written. */
        /* Write data: */
        DBMS_LOB.WRITE (Lob_loc, Amount, Position, Buffer);
```

```
        Position := Position + Amount;
    END LOOP;
    /* Closing the LOB is mandatory if you have opened it: */
    DBMS_LOB.CLOSE (Lob_loc);

EXCEPTION
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('Operation failed');
END;
```

C（OCI）: LOB のチェックイン

```
/* Checking in a LOB. This example demonstrates how using OCI you can write
   arbitrary amounts of data to an Internal LOB in either a single piece
   of in multiple pieces using a streaming mechanism that utilizes standard
   polling. A statically allocated Buffer holds the data being
   written to the LOB. */

#define MAXBUFLEN 32767
/* Select the locator into a locator variable */
sb4 select_lock_formateddoc_locator(Lob_loc, errhp, stmthp,svchp)
OCILobLocator *Lob_loc;
OCIError      *errhp;
OCISvcCtx     *svchp;
OCIStmt       *stmthp;
{
  OCIDefine *defnp1, *defnp2;

  text  *sqlstmt =
     (text *) "SELECT ntab.formatted_doc \
           FROM TABLE(SELECT pm.ad_textdoc_ntab FROM Print_media pm \
              WHERE pm.product_id = 2268 AND pm.ad_id = 21001) ntab \
                 WHERE ntab.document_typ = 'PDF' FOR UPDATE";

  checkerr (errhp, OCIStmtPrepare(stmthp, errhp, sqlstmt,
                                  (ub4)strlen((char *)sqlstmt),
                                  (ub4) OCI_NTV_SYNTAX, (ub4) OCI_DEFAULT));
  checkerr (errhp, OCIDefineByPos(stmthp, &defnp1, errhp, (ub4) 1,
                                  (dvoid *)&Lob_loc, (sb4)0,
                                  (ub2) SQLT_CLOB,(dvoid *) 0,
                                  (ub2 *) 0, (ub2 *) 0, (ub4) OCI_DEFAULT)
           || OCIDefineByPos(stmthp, &defnp2, errhp, (ub4) 2,
                                  (dvoid *)&Lob_loc, (sb4)0,
                                  (ub2) SQLT_CLOB,(dvoid *) 0,
                                  (ub2 *) 0, (ub2 *) 0, (ub4) OCI_DEFAULT)
                                  );
```

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

     return OCI_SUCCESS;
}

void checkinLob(envhp, errhp, svchp, stmthp)
OCIEnv    *envhp;
OCIError  *errhp;
OCISvcCtx *svchp;
OCIStmt   *stmthp;
{
  OCIClobLocator *Lob_loc;
  ub4 Total = 2.5*MAXBUFLEN;
  ub4 amtp;
  ub4 offset;
  ub4 remainder;
  ub4 nbytes;
  boolean last;
  ub1 bufp[MAXBUFLEN];
  sb4 err;

  /* Allocate locators desriptors*/
  (void) OCIDescriptorAlloc((dvoid *)envhp, (dvoid **) &Lob_loc,
                            (ub4)OCI_DTYPE_LOB,(size_t) 0,(dvoid **) 0);
  /* Select the CLOB */
  printf(" select the formatted_doc locator...\n");
  select_lock_formatteddoc_locator(Lob_loc, errhp, stmthp, svchp);

  /* Open the CLOB */
  printf (" open the locator.\n");
  checkerr (errhp, (OCILobOpen(svchp, errhp, Lob_loc, OCI_LOB_READWRITE)));

  printf (" write the lob in pieces\n");
  if (Total > MAXBUFLEN)
    nbytes = MAXBUFLEN;   /* We will use streaming via standard polling */
  else
    nbytes = Total;                /* Only a single write is required */

  /* Fill the buffer with nbytes worth of data */
  remainder = Total - nbytes;
  /* Setting Amount to 0 streams the data until use specifies OCI_LAST_PIECE */
  amtp = 0;
  /* offset = <Starting position where to begin writing the data>; */
  offset = 1;
```

```
if (0 == remainder)
{
  amtp = nbytes;
  /* Here, (Total <= MAXBUFLEN ) so we can write in one piece */
    checkerr (errhp, OCILobWrite (svchp, errhp, Lob_loc, amtp,
                                  offset, bufp, nbytes,
                                  OCI_ONE_PIECE, (dvoid *) 0,
                                  (sb4 (*)(dvoid *,dvoid *,ub4 *,ub1 *)) 0,
                                  0, SQLCS_IMPLICIT));
}
else
{
  /* Here (Total > MAXBUFLEN ) so we use streaming via standard polling */
  /* write the first piece.  Specifying first initiates polling. */
  err = OCILobWrite (svchp, errhp, Lob_loc, &amtp, offset, bufp, nbytes,
                     OCI_FIRST_PIECE, (dvoid *) 0,
                     (sb4 (*)(dvoid *,dvoid *,ub4 *,ub1 *)) 0,
                     0, SQLCS_IMPLICIT);
  if (err != OCI_NEED_DATA)
    checkerr (errhp, err);
  last = FALSE;

  /* write the next (interim) and last pieces */
  do
  {
    if (remainder > MAXBUFLEN)
      nbytes = MAXBUFLEN;     /* Still have more pieces to go */
    else
    {
      nbytes = remainder;     /* Here, (remainder <= MAXBUFLEN) */
      last = TRUE;            /* This is going to be the Final piece */
    }

    /* Fill the buffer with nbytes worth of data */
    if (last)
    {
      /* Specifying LAST terminates polling */
      err = OCILobWrite (svchp, errhp, Lob_loc, &amtp,
                         offset, bufp, nbytes,
                         OCI_LAST_PIECE, (dvoid *) 0,
                         (sb4 (*)(dvoid*,dvoid*,ub4*,ub1 *))0,
                         0, SQLCS_IMPLICIT);
      if (err != OCI_SUCCESS)
        checkerr(errhp, err);
    }
    else
```

```
    {
      err = OCILobWrite (svchp, errhp, Lob_loc, &amtp,
                         offset, bufp, nbytes,
                         OCI_NEXT_PIECE, (dvoid *) 0,
                         (sb4 (*)(dvoid*,dvoid*,ub4*,ub1 *))0,
                         0, SQLCS_IMPLICIT);
      if (err != OCI_NEED_DATA)
        checkerr (errhp, err);
    }
    /* Determine how much is left to write */
    remainder = remainder - nbytes;
  } while (!last);
}

/* At this point, (remainder == 0) */

/* Closing the BLOB is mandatory if you have opened it */
checkerr (errhp, OCILobClose(svchp, errhp, Lob_loc));

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

}
```

COBOL（Pro*COBOL）: LOB のチェックイン

```
* Checking in a LOB
  IDENTIFICATION DIVISION.
  PROGRAM-ID. CHECKIN.
  ENVIRONMENT DIVISION.
  INPUT-OUTPUT SECTION.
  FILE-CONTROL.
     SELECT INFILE
        ASSIGN TO "datfile.dat"
        ORGANIZATION IS SEQUENTIAL.
  DATA DIVISION.
  FILE SECTION.

  FD INFILE
     RECORD CONTAINS 80 CHARACTERS.
  01 INREC      PIC X(80).

  WORKING-STORAGE SECTION.
  01  USERID   PIC X(11) VALUES "P/PM".
  01  CLOB1          SQL-CLOB.M
  01  BUFFER         PIC X(80) VARYING.
```

```
 01  AMT            PIC S9(9) COMP VALUE 0.
 01  OFFSET         PIC S9(9) COMP VALUE 1.
 01  END-OF-FILE    PIC X(1) VALUES "N".
 01  D-BUFFER-LEN   PIC 9.
 01  D-AMT          PIC 9.
     EXEC SQL INCLUDE SQLCA END-EXEC.

 PROCEDURE DIVISION.
 WRITE-CLOB.
     EXEC SQL WHENEVER SQLERROR DO PERFORM SQL-ERROR END-EXEC.
     EXEC SQL CONNECT :USERID END-EXEC.

* Allocate and initialize the CLOB locator:
     EXEC SQL ALLOCATE :CLOB1 END-EXEC.
     EXEC SQL
          SELECT AD_SOURCETEXT INTO :CLOB1 FROM PRINT_MEDIA
          WHERE PRODUCT_ID = 3060 AND AD_ID = 11001 FOR UPDATE
     END-EXEC.

* Open the input file for reading:
     OPEN INPUT INFILE.

* Either write entire record or write first piece.
* Read a data file here and populate BUFFER-ARR and BUFFER-LEN.
* END-OF-FILE will be set to "Y" when the entire file has been
* read.
     PERFORM READ-NEXT-RECORD.
     MOVE INREC TO BUFFER-ARR.
     MOVE 80 TO BUFFER-LEN.
     IF (END-OF-FILE = "Y")
        MOVE 80 TO AMT
        EXEC SQL
             LOB WRITE ONE :AMT FROM :BUFFER
             INTO :CLOB1 AT :OFFSET END-EXEC
     ELSE
        DISPLAY "LOB WRITE FIRST"
        DISPLAY  BUFFER-ARR
        MOVE 321 TO AMT
        EXEC SQL
           LOB WRITE FIRST :AMT FROM :BUFFER INTO :CLOB1
        END-EXEC
     END-IF.

* Continue reading from the input data file
* and writing to the CLOB:
     PERFORM READ-WRITE
        UNTIL END-OF-FILE = "Y".
```

```
    PERFORM SIGN-OFF.
    STOP RUN.

READ-WRITE.
    PERFORM READ-NEXT-RECORD.
    MOVE INREC TO BUFFER-ARR.
    DISPLAY "READ-WRITE".
    DISPLAY INREC.
    MOVE 80 TO BUFFER-LEN.
    IF END-OF-FILE = "Y"
        DISPLAY "LOB WRITE LAST: ", BUFFER-ARR
        MOVE 1 TO BUFFER-LEN
        EXEC SQL
            LOB WRITE LAST :AMT FROM :BUFFER INTO :CLOB1 END-EXEC
    ELSE
        DISPLAY "LOB WRITE NEXT: ", BUFFER-ARR
        MOVE 0 TO AMT
        EXEC SQL
            LOB WRITE NEXT :AMT FROM :BUFFER INTO :CLOB1 END-EXEC
    END-IF.

READ-NEXT-RECORD.
    MOVE SPACES TO INREC.
    READ INFILE NEXT RECORD
         AT END
          MOVE "Y" TO END-OF-FILE.

SIGN-OFF.
    CLOSE INFILE.
    EXEC SQL FREE :CLOB1 END-EXEC.
    EXEC SQL ROLLBACK WORK RELEASE END-EXEC.
    STOP RUN.

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

C/C++（Pro*C/C++）：LOB のチェックイン

```
/* Checking in a LOB.
   This example shows how you can use Pro*C/C++ to WRITE
   arbitrary amounts of data to an Internal LOB in either a single piece
   or in multiple pieces using a Streaming Mechanism that utilizes standard
   polling. A static Buffer holds the data being written: */

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

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

#define BufferLength 512

void checkInLOB_proc(multiple) int multiple;
{
  OCIClobLocator *Lob_loc;
  VARCHAR Buffer[BufferLength];
  unsigned int Total;
  unsigned int Amount;
  unsigned int remainder, nbytes;
  boolean last;

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  /* Allocate and Initialize the Locator: */
  EXEC SQL ALLOCATE :Lob_loc;
  EXEC SQL SELECT ad_sourcetext INTO :Lob_loc
            FROM Print_media WHERE product_id = 3060 AND ad_id = 11001 FOR UPDATE;
  /* Open the LOB: */
  EXEC SQL LOB OPEN :Lob_loc READ WRITE;
  Total = Amount = (multiple * BufferLength);
  if (Total > BufferLength)
    nbytes = BufferLength;   /* We will use streaming via standard polling */
  else
    nbytes = Total;                     /* Only a single WRITE is required */
  /* Fill the Buffer with nbytes worth of data: */
  memset((void *)Buffer.arr, 32, nbytes);
  Buffer.len = nbytes;       /* Set the Length */
  remainder = Total - nbytes;
```

```
if (0 == remainder)
  {
    /* Here, (Total <= BufferLength) so we can WRITE in ONE piece: */
    EXEC SQL LOB WRITE ONE :Amount FROM :Buffer INTO :Lob_loc;
    printf("Write ONE Total of %d characters\n", Amount);
  }
else
  {
    /* Here (Total > BufferLength) so use streaming via standard polling:
       WRITE the FIRST piece.  Specifying FIRST initiates polling: */
    EXEC SQL LOB WRITE FIRST :Amount FROM :Buffer INTO :Lob_loc;
    printf("Write FIRST %d characters\n", Buffer.len);
    last = FALSE;
    /* WRITE the NEXT (interim) and LAST pieces: */
    do
      {
        if (remainder > BufferLength)
          nbytes = BufferLength;         /* Still have more pieces to go */
        else
          {
            nbytes = remainder;
            last = TRUE;          /* This is going to be the Final piece */
          }
        /* Fill the Buffer with nbytes worth of data: */
        memset((void *)Buffer.arr, 32, nbytes);
        Buffer.len = nbytes;       /* Set the Length */
        if (last)
          {
            EXEC SQL WHENEVER SQLERROR DO Sample_Error();
            /* Specifying LAST terminates polling: */
            EXEC SQL LOB WRITE LAST :Amount FROM :Buffer INTO :Lob_loc;
            printf("Write LAST Total of %d characters\n", Amount);
          }
        else
          {
            EXEC SQL WHENEVER SQLERROR DO break;
            EXEC SQL LOB WRITE NEXT :Amount FROM :Buffer INTO :Lob_loc;
            printf("Write NEXT %d characters\n", Buffer.len);
          }
        /* Determine how much is left to WRITE: */
        remainder = remainder - nbytes;
      } while (!last);
  }
EXEC SQL WHENEVER SQLERROR DO Sample_Error();
/* At this point, (Amount == Total), the total amount that was written */
/* Close the LOB: */
EXEC SQL LOB CLOSE :Lob_loc;
EXEC SQL FREE :Lob_loc;
```

```
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  checkInLOB_proc(1);
  EXEC SQL ROLLBACK WORK;
  checkInLOB_proc(4);
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

Visual Basic（OO4O）: LOB のチェックイン

```
'Checking in a LOB
'There are two ways of writing a lob - using  orablob.write or
orablob.copyfromfile

'Using the OraBlob.Write mechanism
Dim MySession As OraSession
Dim OraDb As OraDatabase
Dim fnum As Integer
Dim OraDyn As OraDynaset, OraPhoto As OraBlob, amount_written%, chunksize%,
curchunk() As Byte

Set MySession = CreateObject("OracleInProcServer.XOraSession")
Set OraDb = MySession.OpenDatabase("exampledb", "pm/pm", 0&)
chunksize = 500
ReDim curchunk(chunksize)
Set OraDyn = OraDb.CreateDynaset("SELECT * FROM Print_media", ORADYN_DEFAULT)
Set OraPhoto = OraDyn.Fields("ad_photo").Value

fnum = FreeFile
Open "c:\tmp\keyboard_3016_13001.jpg" For Binary As #fnum
OraPhoto.offset = 1
OraPhoto.pollingAmount = LOF(fnum)
remainder = LOF(fnum)

 Dim piece As Byte
 Get #fnum, , curchunk
 OraDyn.Edit
 piece = ORALOB_FIRST_PIECE
 OraPhoto.Write curchunk, chunksize, ORALOB_FIRST_PIECE

 While OraPhoto.Status = ORALOB_NEED_DATA
    remainder = remainder - chunksize
    If remainder <= chunksize Then
```

```
        chunksize = remainder
        piece = ORALOB_LAST_PIECE
    Else
        piece = ORALOB_NEXT_PIECE
    End If

    Get #fnum, , curchunk
    OraPhoto.Write curchunk, chunksize, piece
 Wend

OraDyn.Update

'Using the OraBlob.CopyFromFile mechanism
Set OraDyn = OraDb.CreateDynaset("select * from Print_media order by product_
id, ad_id", ORADYN_DEFAULT)
Set OraPhoto = OraDyn.Fields("ad_photo").Value
OraDyn.Edit
OraPhoto.CopyFromFile "c:\tmp\keyboard3016_13001.jpg"
OraDyn.Update
```

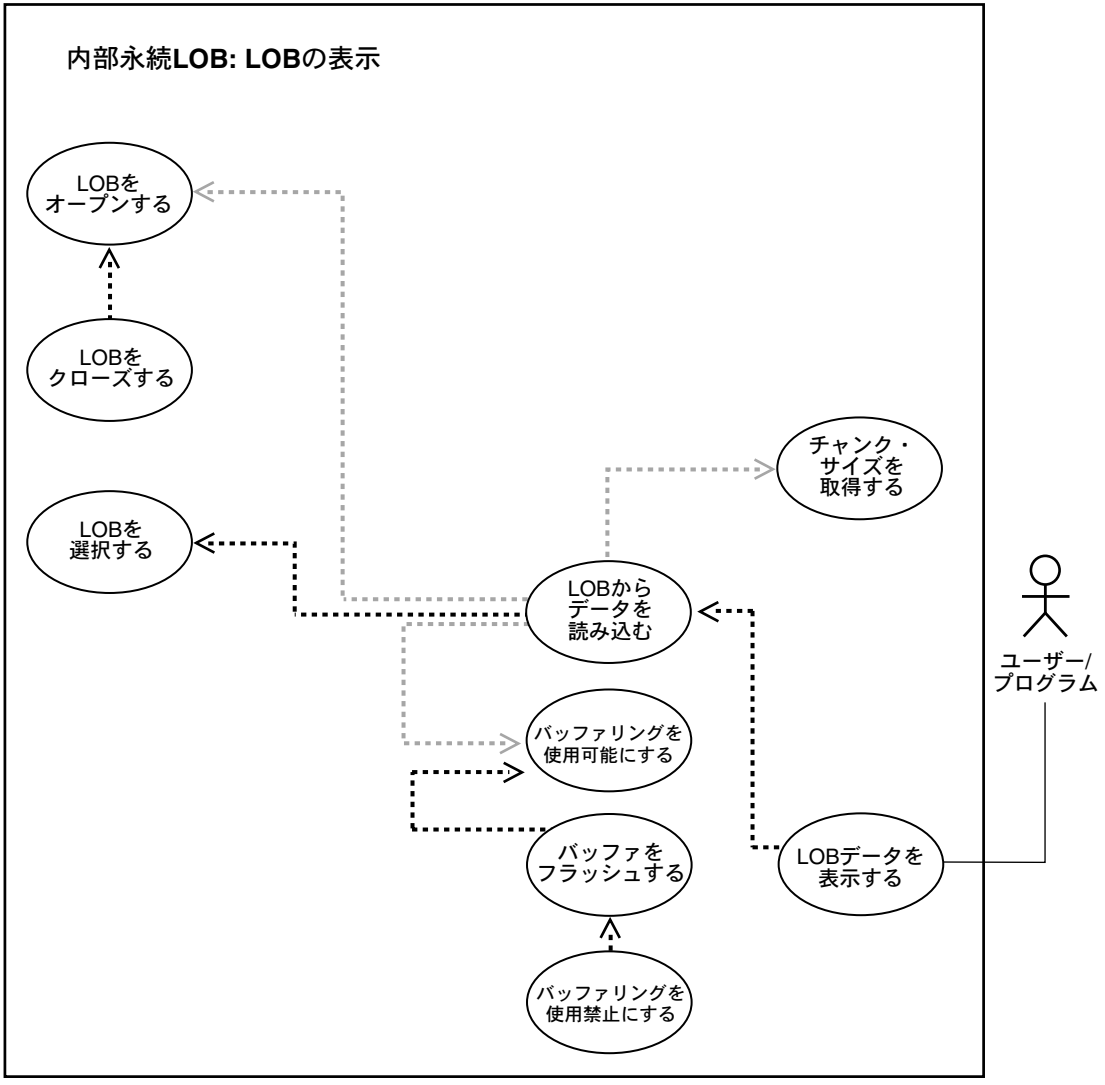
Java（JDBC）: LOB のチェックイン

```
// Checking in a LOB
import java.io.InputStream;
import java.io.OutputStream;

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

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

public class Ex2_66
{
  static final int MAXBUFSIZE = 32767;

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

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

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

    // Create a Statement:
    Statement stmt = conn.createStatement ();
    try
    {
     CLOB lob_loc = null;
     String buf = new String ("Some Text To Write for the Ad");
     ResultSet rset = stmt.executeQuery (
       "SELECT ad_sourcetext FROM Print_media WHERE product_id = 2268
       AND ad_id = 21001 FOR UPDATE");
   if (rset.next())
   {
     lob_loc = ((OracleResultSet)rset).getCLOB (1);
   }

   long pos = 0;       // Offset within the CLOB where the data is to be written
   long length = 0;    // This is the size of the buffer to be written

   // This loop writes the buffer three times consecutively:
   for (int i = 0; i < 3; i++)
   {
      pos = lob_loc.length();

      // an alternative is: lob_loc.putString(pos, buf);
      lob_loc.putString(pos, buf);

      // Some debug information:
      System.out.println(" putString(" + Long.toString(pos) +" buf);");
   }
   stmt.close();
   conn.commit();
   conn.close();
    }
    catch (SQLException e)
    {
       e.printStackTrace();
    }
  }
}
```

LOB データの表示

図 10-15 利用図 : LOB データの表示



参照： 10-2 ページの表 10-1「利用モデル : 内部永続 LOB」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には、ポーリングまたはコールバックによってストリーム・メカニズムを有効にした `OCILobRead()` を使用します。

構文

各プログラム環境で使用可能な関数またはファンクションのリストは、第 3 章「様々なプログラム環境での LOB のサポート」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）：『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』の第 23 章「DBMS_LOB」の「DBMS_LOB 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」および「CLOSE プロシージャ」
- C（OCI）：『Oracle Call Interface プログラマーズ・ガイド』の第 16 章「その他の OCI リレーショナル関数」の「LOB 関数」の OCILobOpen()、OCILobRead() および OCILobClose()
- C++（OCCI）：『Oracle C++ Call Interface プログラマーズ・ガイド』
- COBOL（Pro*COBOL）：『Pro*COBOL Precompiler プログラマーズ・ガイド』の LOB に関する詳細、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・ディレクティブ」の 「LOB READ（実行可能埋込み SQL 拡張機能）」
- C/C++（Pro*C/C++）：『Pro*C/C++ Precompiler プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびディレクティブ」の「LOB READ（実行可能埋込み SQL 拡張機能）」
- Visual Basic（OO4O オンライン・ヘルプ）: ヘルプの「目次」タブから、「OO4O オートメーション・サーバー・リファレンス」>「OO4O サーバーのオブジェクト」>「OraBLOB、OraCLOB」>「メソッド」>「read」、および「OO4O オートメーション・サーバー・リファレンス」「OO4O サーバーのオブジェクト」>「OraBLOB、OraCLOB」>「プロパティ」>「offset」を選択
- Java（JDBC）：『Oracle9*i* JDBC 開発者ガイドおよびリファレンス』の第 8 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『Oracle9*i* 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「JDBC 2.0 LOB 型と Oracle 拡張型のサポート」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 ad_header から、イメージ logo をクライアント側にストリーム読込みし、データを表示します。

例

次のプログラム環境での例を示します。

- PL/SQL（DBMS_LOB）: LOB データの表示（10-94 ページ）
- C（OCI）: LOB データの表示（10-95 ページ）
- C++（OCCI）: 今回のリリースでは例は提供されません。
- COBOL（Pro*COBOL）: LOB データの表示（10-97 ページ）
- C/C++（Pro*C/C++）: LOB データの表示（10-99 ページ）
- Visual Basic（OO4O）: LOB データの表示（10-100 ページ）
- Java（JDBC）: LOB データの表示（10-101 ページ）

PL/SQL（DBMS_LOB）: LOB データの表示

```
/* Displaying LOB data.The example procedure displayLOB_proc is not part of the
DBMS_LOB package: */
CREATE OR REPLACE PROCEDURE displayLOB_proc IS
Lob_loc  BLOB;
Buffer   RAW(1024);
Amount   BINARY_INTEGER := 1024;
Position INTEGER := 1;
BEGIN
   /* Select the LOB: */
   SELECT pm.ad_header.logo INTO Lob_loc
      FROM print_media pm WHERE pm.product_id = 3060 AND pm.ad_id = 11001;
   /* Opening the LOB is optional: */
   DBMS_LOB.OPEN (Lob_loc, DBMS_LOB.LOB_READONLY);
   LOOP
      DBMS_LOB.READ (Lob_loc, Amount, Position, Buffer);
      /* Display the buffer contents: */
      DBMS_OUTPUT.PUT_LINE(utl_raw.cast_to_varchar2(Buffer));
      Position := Position + Amount;
   END LOOP;
   /* Closing the LOB is mandatory if you have opened it: */
   DBMS_LOB.CLOSE (Lob_loc);
   EXCEPTION
      WHEN NO_DATA_FOUND THEN
         DBMS_OUTPUT.PUT_LINE('End of data');
END;
```

C（OCI）：LOB データの表示

```
/* Displaying LOB data. This example reads the entire contents of a BLOB
   piecewise into a buffer using the standard polling method, processing
   each buffer piece after every READ operation until the entire BLOB
   has been read. */

#define MAXBUFLEN 32767
/* Select the locator into a locator variable */
sb4 select_mapobjectdrawing_locator(Lob_loc, errhp, svchp, stmthp)
OCILobLocator *Lob_loc;
OCIError      *errhp;
OCISvcCtx     *svchp;
OCIStmt       *stmthp;
{
  OCIDefine *defnp1, *defnp2;
  text  *sqlstmt =
     (text *) "SELECT pm.Adheader_type.logo \
                 FROM Print_media pm WHERE pm.product_id = 3060 AND ad_id = 11001";
  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)
          || OCIDefineByPos(stmthp, &defnp2, errhp, (ub4) 2,
                                  (dvoid *)&Lob_loc, (sb4)0,
                                  (ub2) SQLT_BLOB,(dvoid *) 0,
                                  (ub2 *) 0, (ub2 *) 0, (ub4) OCI_DEFAULT)
                                  );

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

void displayLob(envhp, errhp, svchp, stmthp)
OCIEnv    *envhp;
OCIError  *errhp;
OCISvcCtx *svchp;
OCIStmt   *stmthp;
```

```
{
  OCIBlobLocator *Lob_loc;
  ub4 amt;
  ub4 offset;
  sword retval;
  boolean done;
  ub1 bufp[MAXBUFLEN];
  ub4 buflen;
  OCILobLocator *Lob_Loc;

  /* Allocate the Source (bfile) & destination (blob) locators desriptors*/
  (void) OCIDescriptorAlloc((dvoid *) envhp,
                            (dvoid **) &Lob_loc, (ub4)OCI_DTYPE_LOB,
                            (size_t) 0, (dvoid **) 0);
  /* Select the BLOB */
  printf(" select the adheaderlogo locator...\n");
  select_adheaderlogo_locator(Lob_loc, errhp, svchp, stmthp);

  /* Open the BLOB */
  printf(" open the lob\n");
  checkerr (errhp, (OCILobOpen(svchp, errhp, Lob_loc, OCI_LOB_READONLY)));

  /* Setting amt = 0 will read till the end of LOB*/
  amt = 0;
  buflen = sizeof(bufp);

 /* Process the data in pieces */
  printf(" Process the data in pieces\n");
  offset = 1;
  memset(bufp, '\0', MAXBUFLEN);
  done = FALSE;
  while (!done)
  {
    retval = OCILobRead(svchp, errhp, Lob_loc, &amt, offset, (dvoid *) bufp,
                        buflen, (dvoid *)0,
                        (sb4 (*)(dvoid *, dvoid *, ub4, ub1)) 0,
                        (ub2) 0, (ub1) SQLCS_IMPLICIT);
    switch (retval)
    {
    case OCI_SUCCESS:           /* Only one piece or last piece*/
      /* Process the data in bufp. amt will give the amount of data just read in
         bufp. This is in bytes for BLOBs and in characters for fixed
         width CLOBS and in bytes for variable width CLOBs
       */
      done = TRUE;
      break;
    case OCI_ERROR:
```

```
      checkerr (errhp, retval);
      done = TRUE;
      break;
     case OCI_NEED_DATA:         /* There are 2 or more pieces */
      /* Process the data in bufp. amt will give the amount of data just read in
         bufp. This is in bytes for BLOBs and in characters for fixed
         width CLOBS and in bytes for variable width CLOBs
       */
      break;
     default:
      checkerr (errhp, retval);
      done = TRUE;
      break;
     }
   } /* while */

   /* Closing the BLOB is mandatory if you have opened it */
    printf(" close the lob \n");
    checkerr (errhp, OCILobClose(svchp, errhp, Lob_loc));

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

}
```

COBOL（Pro*COBOL）：LOB データの表示

```
      * DISPLAYING LOB DATA
        IDENTIFICATION DIVISION.
        PROGRAM-ID. DISPLAY-BLOB.
        ENVIRONMENT DIVISION.
        DATA DIVISION.
        WORKING-STORAGE SECTION.

        01  USERID   PIC X(11) VALUES "SAMP/SAMP".
        01  BLOB1          SQL-BLOB.
        01  BUFFER2        PIC X(5) VARYING.
        01  AMT            PIC S9(9) COMP.
        01  OFFSET         PIC S9(9) COMP VALUE 1.
        01  D-AMT          PIC 9.

            EXEC SQL VAR BUFFER2 IS RAW(5) END-EXEC.
            EXEC SQL INCLUDE SQLCA END-EXEC.
```

```
        PROCEDURE DIVISION.
        DISPLAY-BLOB.
            EXEC SQL WHENEVER SQLERROR DO PERFORM SQL-ERROR END-EXEC.
            EXEC SQL CONNECT :USERID END-EXEC.
      * Allocate and initialize the BLOB locator:
            EXEC SQL ALLOCATE :BLOB1 END-EXEC.
             EXEC SQL WHENEVER NOT FOUND GOTO END-OF-BLOB END-EXEC.
              EXEC SQL SELECT PM.AD_PHOTO INTO :BLOB1
               FROM PRINT_MEDIA PM WHERE PM.PRODUCT_ID = 2268 AND AD_ID = 21001
END-EXEC.
            DISPLAY "Found column AD_PHOTO".
      * Initiate polling read:
            MOVE 0 TO AMT.

            EXEC SQL LOB READ :AMT FROM :BLOB1 AT :OFFSET
               INTO :BUFFER2 END-EXEC.
            DISPLAY " ".
            MOVE AMT TO D-AMT.
            DISPLAY "first read (", D-AMT, "): " BUFFER2.
         READ-BLOB-LOOP.
            MOVE "     " TO BUFFER2.
            EXEC SQL LOB READ :AMT FROM :BLOB1 INTO :BUFFER2 END-EXEC.
             MOVE AMT TO D-AMT.
             DISPLAY "next read (", D-AMT, "): " BUFFER2.
            GO TO READ-BLOB-LOOP.

        END-OF-BLOB.
            EXEC SQL WHENEVER NOT FOUND CONTINUE END-EXEC.
            EXEC SQL FREE :BLOB1 END-EXEC.
            MOVE AMT TO D-AMT.
            DISPLAY "last read (", D-AMT, "): " BUFFER2(1:AMT).
            EXEC SQL ROLLBACK WORK RELEASE END-EXEC.
            STOP RUN.

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

C/C++（Pro*C/C++）：LOB データの表示

```
/* Displaying LOB data. This example reads the entire contents of a BLOB
   piecewise into a buffer using a standard polling method, processing
   each buffer piece after every READ operation until the entire BLOB
   has been read: */

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

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

#define BufferLength 32767

void displayLOB_proc()
{
  OCIBlobLocator *Lob_loc;
  int Amount;
  struct {
    unsigned short Length;
    char Data[BufferLength];
  } Buffer;
  /* Datatype equivalencing is mandatory for this datatype: */
  EXEC SQL VAR Buffer IS VARRAW(BufferLength);

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  EXEC SQL ALLOCATE :Lob_loc;
  /* Select the BLOB: */
  EXEC SQL SELECT m.ad_header.header_text INTO Lob_loc
           FROM Print_media m WHERE m.product_id = 3060 AND ad_id = 11001;
  /* Open the BLOB: */
  EXEC SQL LOB OPEN :Lob_loc READ ONLY;
  /* Setting Amount = 0 will initiate the polling method: */
  Amount = 0;
  /* Set the maximum size of the Buffer: */
  Buffer.Length = BufferLength;
  EXEC SQL WHENEVER NOT FOUND DO break;
```

```
  while (TRUE)
    {
      /* Read a piece of the BLOB into the Buffer: */
      EXEC SQL LOB READ :Amount FROM :Lob_loc INTO :Buffer;
      /* Process (Buffer.Length == BufferLength) amount of Buffer.Data */
    }
  /* Process (Buffer.Length == Amount) amount of Buffer.Data */
  /* Closing the BLOB is mandatory if you have opened it: */
  EXEC SQL LOB CLOSE :Lob_loc;
  EXEC SQL FREE :Lob_loc;
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  displayLOB_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

Visual Basic（OO4O）: LOB データの表示

```
'Displaying LOB data
'Using the OraClob.Read mechanism
Dim MySession As OraSession
Dim OraDb As OraDatabase

Set MySession = CreateObject("OracleInProcServer.XOraSession")
Set OraDb = MySession.OpenDatabase("exampledb", "samp/samp", 0&)
Dim OraDyn as OraDynaset, OraAdSourceText as OraClob, amount_read%, chunksize%,
chunk
chunksize = 32767
Set OraDyn = OraDb.CreateDynaset("SELECT * FROM Print_media", ORADYN_DEFAULT)
Set OraAdSourceText = OraDyn.Fields("ad_sourcetext").Value
OraAdSourceText.PollingAmount = OraAdSourceText.Size 'Read entire CLOB contents
Do
   'chunk returned is a variant of type byte array:
    amount_read = OraAdSourceText.Read(chunk, chunksize)
   'Msgbox chnunk
Loop Until OraAdSourceText.Status <> ORALOB_NEED_DATA
```

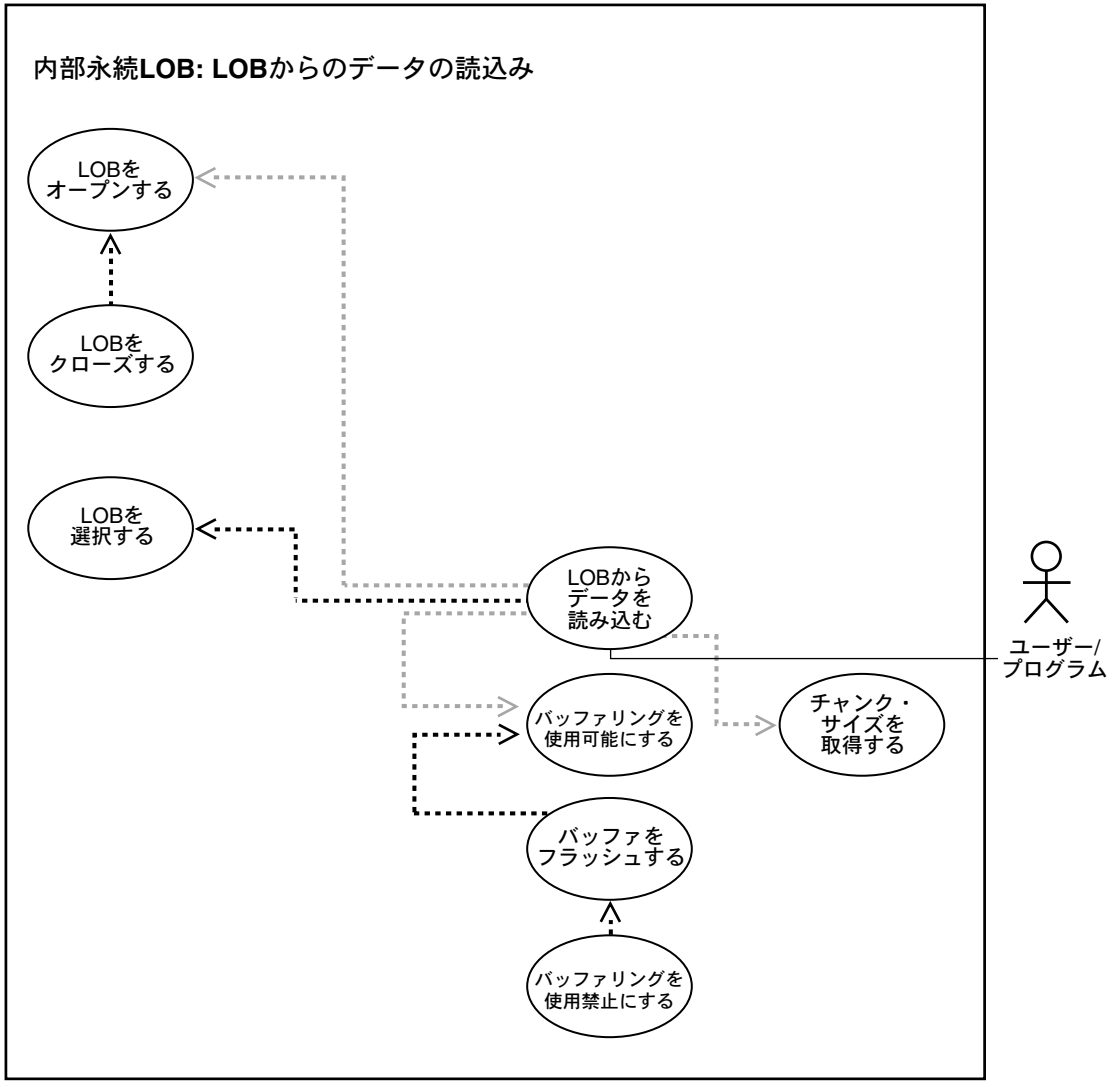
Java（JDBC）：LOB データの表示

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

// Oracle Specific JDBC classes:
import oracle.sql.*;
import oracle.jdbc.driver.*;
public class Ex2_72
{
  static final int MAXBUFSIZE = 32767;
  public static void main (String args [])
       throws Exception
  {
    // Load the Oracle JDBC driver:
    DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

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

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

    // Create a Statement:
    Statement stmt = conn.createStatement ();
    try
    {
     BLOB lob_loc = null;
     InputStream in = null;
     byte buf[] = new byte[MAXBUFSIZE];
     int pos = 0;
     int length = 0;
     ResultSet rset = stmt.executeQuery (
        "SELECT pm.ad_header.logo FROM Print_media pm
            WHERE pm.product_id = 2056 AND ad_id = 12001");
   if (rset.next())
   {
     lob_loc = ((OracleResultSet)rset).getBLOB (1);
   }
```

```
    // read this LOB through an InputStream:
    in = lob_loc.getBinaryStream();
    while ((length = in.read(buf)) != -1)
    {
       pos += length;
       System.out.println(Integer.toString(pos));
       // Process the contents of the buffer here.
    }
       in.close();
    stmt.close();
    conn.commit();
    conn.close();
     }
     catch (SQLException e)
     {
        e.printStackTrace();
     }
  }
}
```

LOB からのデータの読込み

図 10-16 利用図 : LOB からのデータの読込み



参照： 10-2 ページの表 10-1「利用モデル : 内部永続 LOB」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

ストリーム読込み 大量の LOB データを最も効率よく読み込むには、ポーリングまたはコールバックによってストリーム・メカニズムを有効にした OCILobRead() を使用します。

LOB 値を読み込む場合、LOB の終わり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。次に、すべてのパラメータの初期化を除いた OCI 読込みコールを示します。

```
#define MAX_LOB_SIZE 4294967295
ub4  amount =  MAX_LOB_SIZE;
ub4  offset = 1000;
OCILobRead(svchp, errhp, locp, &amount, offset, bufp, bufl, 0, 0, 0, 0)
```

注意：

- DBMS_LOB.READ では、量パラメータをデータのサイズより大きく指定できます。PL/SQL では、量パラメータをバッファ・サイズ以下に指定する必要があります。また、バッファ・サイズは **32KB** 以下に制限されます。
- OCILobRead では、量パラメータを 4GB -1 に指定し、LOB の終わりまで読み込むことができます。

- ポーリング・モードを使用すると、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と、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（『Oracle Call Interface 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` ファンクションは、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、ディスクからのデータの読込み時に、Oracle データベースで使用されているサイズと同じ単位を使用するためです。アプリケーションで問題が発生しないかぎり、同じ LOB チャンクに複数回の LOB 読込みコールを発行するのではなく、チャンク全体の取得に十分なサイズでバッチ読込みを行うと、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関数またはファンクションのリストは、第 3 章「様々なプログラム環境での LOB のサポート」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）：『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』の第 23 章「DBMS_LOB」の「DBMS_LOB 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」および「CLOSE プロシージャ」
- C（OCI）：『Oracle Call Interface プログラマーズ・ガイド』の第 16 章「その他の OCI リレーショナル関数」の「LOB 関数」の OCILobOpen()、OCILobRead() および OCILobClose()
- C++（OCCI）：『Oracle C++ Call Interface プログラマーズ・ガイド』
- COBOL（Pro*COBOL）：『Pro*COBOL Precompiler プログラマーズ・ガイド』の LOB に関する詳細、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・ディレクティブ」の「LOB READ（実行可能埋込み SQL 拡張機能）」
- C/C++（Pro*C/C++）：『Pro*C/C++ Precompiler プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびディレクティブ」の「LOB READ（実行可能埋込み SQL 拡張機能）」
- Visual Basic（OO4O オンライン・ヘルプ）：ヘルプの「目次」タブから、「OO4O オートメーション・サーバー・リファレンス」>「OO4O サーバーのオブジェクト」>「OraBLOB、OraCLOB」>「メソッド」>「read」を選択
- Java（JDBC）：『Oracle9*i* JDBC 開発者ガイドおよびリファレンス』の第 8 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『Oracle9*i* 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「JDBC 2.0 LOB 型と Oracle 拡張型のサポート」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 つのイメージから読み込みます。

例

次のプログラム環境での例を示します。

PL/SQL（DBMS_LOB）: LOB からのデータの読込み

```
/* Reading LOB data. The example procedure readLOB_proc is not part of the
   DBMS_LOB package: */
CREATE OR REPLACE PROCEDURE readLOB_proc IS
    Lob_loc          BLOB;
    Buffer           RAW(32767);
    Amount           BINARY_INTEGER := 32767;
    Position         INTEGER := 1000;
    Chunksize        INTEGER;
BEGIN
    /* Select the LOB: */
   SELECT ad_composite INTO Lob_loc
      FROM print_media WHERE product_id = 3060 AND ad_id = 11001;
   /* Find out the chunksize for this LOB column: */
   Chunksize := DBMS_LOB.GETCHUNKSIZE(Lob_loc);
   IF (Chunksize < 32767) THEN
      Amount := (32767 / Chunksize) * Chunksize;
   END IF;
   /* Opening the LOB is optional: */
   DBMS_LOB.OPEN (Lob_loc, DBMS_LOB.LOB_READONLY);
   /* Read data from the LOB: */
   DBMS_LOB.READ (Lob_loc, Amount, Position, Buffer);
   /* Closing the LOB is mandatory if you have opened it: */
   DBMS_LOB.CLOSE (Lob_loc);
END;
```

C（OCI）: LOB からのデータの読込み

```
/* Reading LOB data. This example reads the entire contents of a BLOB
   piecewise into a buffer using a standard polling method, processing
   each buffer piece after every READ operation until the entire BLOB
   has been read. */
#define MAXBUFLEN 1000

/* Select the locator into a locator variable */
sb4 select_frame_locator(Lob_loc, errhp, svchp, stmthp)
OCILobLocator *Lob_loc;
OCIError      *errhp;
OCISvcCtx     *svchp;
OCIStmt       *stmthp;
{
  OCIDefine *defnp1;

  text  *sqlstmt =
     (text *) "SELECT ad_composite \
                FROM Print_media pm \
                WHERE pm.product_id = 2268";

  printf("  prepare statement in select_adcomposite_locator\n");
  checkerr (errhp, OCIStmtPrepare(stmthp, errhp, sqlstmt,
                                  (ub4)strlen((char *)sqlstmt),
                                  (ub4) OCI_NTV_SYNTAX, (ub4) OCI_DEFAULT));
    printf("  OCIDefineByPos in select_adcomposite_locator\n");
  checkerr (errhp, OCIDefineByPos(stmthp, &defnp1, errhp, (ub4) 1,
                                  (dvoid *)&Lob_loc, (sb4)0,
                                  (ub2) SQLT_BLOB,(dvoid *) 0,
                                  (ub2 *) 0, (ub2 *) 0, (ub4) OCI_DEFAULT));
   /* Execute the select and fetch one row */
   printf("  OCIStmtExecute in select_adcomposite_locator\n");
   checkerr(errhp, OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                                  (CONST OCISnapshot*) 0, (OCISnapshot*) 0,
                                  (ub4) OCI_DEFAULT));
      return 0;
}

void readLOB_proc(envhp, errhp, svchp, stmthp)
OCIEnv    *envhp;
OCIError  *errhp;
OCISvcCtx *svchp;
OCIStmt   *stmthp;
{
  ub4 amt;
  ub4 offset;
```

```
sword retval;
ub1 bufp[MAXBUFLEN];
ub4 buflen;
boolean done;

OCILobLocator *Lob_loc;
OCILobLocator *blob;

/* Allocate the Source (bfile) & destination (blob) locators descriptors*/
(void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &Lob_loc,
                          (ub4)OCI_DTYPE_LOB, (size_t) 0, (dvoid **) 0);
/* Select the BLOB */
printf(" call select_ad4read_locator\n");
select_adcomposite_locator(Lob_loc, errhp, svchp, stmthp);

/* Open the BLOB */
printf(" call OCILobOpen\n");
checkerr (errhp, OCILobOpen(svchp, errhp, Lob_loc, OCI_LOB_READONLY));

/* Setting the amt to the buffer length.  Note here that amt is in bytes
   since we are using a BLOB  */
amt = 0;
buflen = sizeof(bufp);

/* Process the data in pieces */
printf(" process the data in pieces\n");
offset = 1;
memset(bufp, '\0', MAXBUFLEN);
done = FALSE;

while (!done)
{
retval = OCILobRead(svchp, errhp, Lob_loc, &amt, offset, (dvoid *) bufp,
                    buflen, (dvoid *)0,
                    (sb4 (*)(dvoid *, dvoid *, ub4, ub1)) 0,
                    (ub2) 0, (ub1) SQLCS_IMPLICIT);
switch (retval)
  {
  case OCI_SUCCESS:             /* Only one piece  since amtp == bufp */
    /* Process the data in bufp. amt will give the amount of data just read in
       bufp. This is in bytes for BLOBs and in characters for fixed
       width CLOBS and in bytes for variable width CLOBs */
    printf("[%.*s]\n", buflen, bufp);
    done = TRUE;
   break;
  case OCI_ERROR:
    /*   report_error();        this function is not shown here */
```

```
      done = TRUE;
      break;
    case OCI_NEED_DATA:
      printf("[%.*s]\n", buflen, bufp);
      break;
    default:
      (void) printf("Unexpected ERROR: OCILobRead() LOB.\n");
       done = TRUE;
       break;
    }
  }
  /* Closing the BLOB is mandatory if you have opened it */
     checkerr (errhp, OCILobClose(svchp, errhp, Lob_loc));

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

COBOL（Pro*COBOL）: LOB からのデータの読込み

```
      * READING LOB DATA
        IDENTIFICATION DIVISION.
        PROGRAM-ID. ONE-READ-BLOB.
        ENVIRONMENT DIVISION.
        DATA DIVISION.
        WORKING-STORAGE SECTION.

        01  BLOB1          SQL-BLOB.
        01  BUFFER2        PIC X(32767) VARYING.
        01  AMT            PIC S9(9) COMP.
        01  OFFSET         PIC S9(9) COMP VALUE 1.
        01  USERID   PIC X(11) VALUES "PM/PM".
            EXEC SQL INCLUDE SQLCA END-EXEC.
            EXEC SQL VAR BUFFER2 IS LONG RAW(32767) END-EXEC.
        PROCEDURE DIVISION.
        ONE-READ-BLOB.
            EXEC SQL WHENEVER SQLERROR DO PERFORM SQL-ERROR END-EXEC.
            EXEC SQL
               CONNECT :USERID
            END-EXEC.

       * Allocate and initialize the CLOB locator:
            EXEC SQL ALLOCATE :BLOB1 END-EXEC.
            EXEC SQL WHENEVER NOT FOUND GOTO END-OF-BLOB END-EXEC.
             EXEC SQL
               SELECT AD_COMPOSITE INTO :BLOB1
```

```
           FROM PRINT_MEDIA PM WHERE PM.PRODUCT_ID = 2268
           AND AD_ID = 21001 END-EXEC.
      EXEC SQL LOB OPEN :BLOB1 END-EXEC.

    * Perform a single read:
      MOVE 32767 TO AMT.
      EXEC SQL
         LOB READ :AMT FROM :BLOB1 INTO :BUFFER2 END-EXEC.
      EXEC SQL LOB CLOSE :BLOB1 END-EXEC.

     END-OF-BLOB.
      DISPLAY "BUFFER2: ", BUFFER2(1:AMT).
      EXEC SQL WHENEVER NOT FOUND CONTINUE END-EXEC.
      EXEC SQL FREE :BLOB1 END-EXEC.
      EXEC SQL ROLLBACK WORK RELEASE END-EXEC.
      STOP RUN.

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

C/C++（Pro*C/C++）: LOB からのデータの読込み

```
/* Reading LOB data
#include <oci.h>
#include <stdio.h>
#include <sqlca.h>

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

#define BufferLength 32767

void readLOB_proc()
{
  OCIBlobLocator *Lob_loc;
```

```
  int Amount = BufferLength;
  /* Here (Amount == BufferLength) so only one READ is needed: */
  char Buffer[BufferLength];
  /* Datatype equivalencing is mandatory for this datatype: */
  EXEC SQL VAR Buffer IS RAW(BufferLength);

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  EXEC SQL ALLOCATE :Lob_loc;
  EXEC SQL SELECT ad_composite INTO :Lob_loc
           FROM Print_media WHERE product_id = 3060 AND ad_id = 11001;
  /* Open the BLOB: */
  EXEC SQL LOB OPEN :Lob_loc READ ONLY;
  EXEC SQL WHENEVER NOT FOUND CONTINUE;
  /* Read the BLOB data into the Buffer: */
  EXEC SQL LOB READ :Amount FROM :Lob_loc INTO :Buffer;
  printf("Read %d bytes\n", Amount);
  /* Close the BLOB: */
  EXEC SQL LOB CLOSE :Lob_loc;
  EXEC SQL FREE :Lob_loc;
}

void main()
{
  char *samp = "pm/pm";
  EXEC SQL CONNECT :pm;
  readLOB_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

Visual Basic（OO4O）: LOB からのデータの読込み

```
'Reading LOB data using the OraClob.Read mechanism
Dim MySession As OraSession
Dim OraDb As OraDatabase

Set MySession = CreateObject("OracleInProcServer.XOraSession")
Set OraDb = MySession.OpenDatabase("exampledb", "samp/samp", 0&)
Dim OraDyn as OraDynaset, OraAdSourceText as OraClob, amount_read%, chunksize%,
chunk

chunksize = 32767
Set OraDyn = OraDb.CreateDynaset("SELECT * FROM Print_media", ORADYN_DEFAULT)
Set OraAdSourceText = OraDyn.Fields("ad_sourcetext").Value
OraAdSourceText.pollingAmount = OraAdSourceText.Size
'Read entire CLOB contents
Do
```

```
amount_read = OraAdSourceText.Read(chunk,chunksize)
'chunk returned is a variant of type byte array
Loop Until OraAdSourceText.Status <> ORALOB_NEED_DATA
```

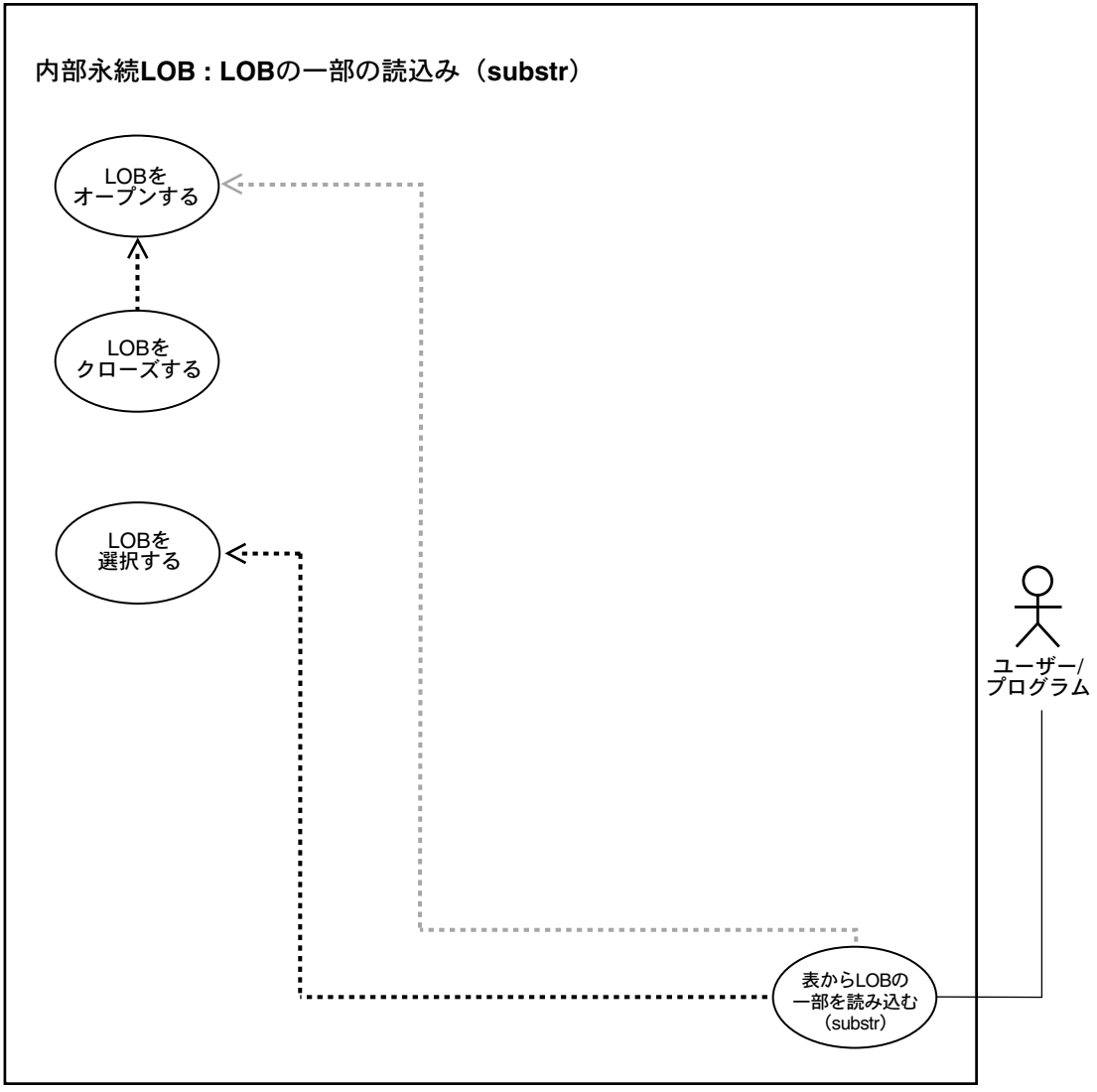
Java（JDBC）: LOB からのデータの読込み

```
// Reading LOB data
// Java IO classes:
import java.io.InputStream;
import java.io.OutputStream;

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

// Oracle Specific JDBC classes:
import oracle.sql.*;
import oracle.jdbc.driver.*;
public class Ex2_79
{
  static final int MAXBUFSIZE = 32767;
  public static void main (String args [])
       throws Exception
  {
    // Load the Oracle JDBC driver:
    DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

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

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

    // Create a Statement:
    Statement stmt = conn.createStatement ();
    try
    {
     BLOB lob_loc = null;
     byte buf[] = new byte[MAXBUFSIZE];
     ResultSet rset = stmt.executeQuery (
          "SELECT ad_composite FROM Print_media WHERE product_id = 2056 AND ad_id =
```

```
12001");
     if (rset.next())
     {
      lob_loc = ((OracleResultSet)rset).getBLOB (1);
     }
     // MAXBUFSIZE is the number of bytes to read and 1000 is the offset from
     // which to start reading
     buf = lob_loc.getBytes(1000, MAXBUFSIZE);

     // Display the contents of the buffer here:
     System.out.println(new String(buf));
     stmt.close();
     conn.commit();
    conn.close();
    }
    catch (SQLException e)
    {
       e.printStackTrace();
    }
  }
}
```

LOB の一部の読込み（substr）

図 10-17 利用図 : LOB の一部の読込み（substr）



参照： 10-2 ページの表 10-1「利用モデル : 内部永続 LOB」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関数またはファンクションのリストは、第 3 章「様々なプログラム環境での LOB のサポート」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）：『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』の第 23 章「DBMS_LOB」の「DBMS_LOB 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」および「CLOSE プロシージャ」
- C（OCI）：参照マニュアルはありません。
- C++（OCCI）：『Oracle C++ Call Interface プログラマーズ・ガイド』
- COBOL（Pro*COBOL）：『Pro*COBOL Precompiler プログラマーズ・ガイド』の LOB に関する詳細、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・ディレクティブ」の「ALLOCATE（実行可能埋込み SQL 拡張機能）」、「LOB OPEN（実行可能埋込み SQL 拡張機能）」、「LOB READ（実行可能埋込み SQL 拡張機能）」と「LOB CLOSE（実行可能埋込み SQL 拡張機能）」
- C/C++（Pro*C/C++）：『Pro*C/C++ Precompiler プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびディレクティブ」の「LOB READ（実行可能埋込み SQL 拡張機能）」、および『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』の第 23 章「DBMS_LOB」の「DBMS_LOB サブプログラムの要約」の「SUBSTR ファンクション」
- Visual Basic（OO4O オンライン・ヘルプ）：ヘルプの「目次」タブから、「OO4O オートメーション・サーバー・リファレンス」>「OO4O サーバーのオブジェクト」>「OraBLOB、OraCLOB」>「プロパティ」>「offset」、および「OO4O オートメーション・サーバー・リファレンス」>「OO4O サーバーのオブジェクト」>「OraBLOB、OraCLOB」>「メソッド」>「read」を選択
- Java（JDBC）：『Oracle9*i* JDBC 開発者ガイドおよびリファレンス』の第 8 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『Oracle9*i* 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「JDBC 2.0 LOB 型と Oracle 拡張型のサポート」の「BLOB、CLOB および BFILE のサポート」

使用例

次の例では、イメージ ad_photo から一部を読み込みます。

例

次のプログラム環境での例を示します。

- PL/SQL（DBMS_LOB）: LOB の一部の読込み（substr）（10-116 ページ）
- C（OCI）: 今回のリリースでは例は提供されません。
- C++（OCCI）: 今回のリリースでは例は提供されません。
- COBOL（Pro*COBOL）: LOB の一部の読込み（substr）（10-117 ページ）
- C/C++（Pro*C/C++）: LOB の一部の読込み（substr）（10-118 ページ）
- Visual Basic（OO4O）: LOB の一部の読込み（substr）（10-119 ページ）
- Java（JDBC）: LOB の一部の読込み（substr）（10-120 ページ）

PL/SQL（DBMS_LOB）: LOB の一部の読込み（substr）

```
/* Reading portion of the LOB data using substr.
   Example procedure substringLOB_proc is not part of the
   DBMS_LOB package: */
CREATE OR REPLACE PROCEDURE substringLOB_proc IS
    Lob_loc          BLOB;
    Amount           BINARY_INTEGER := 32767;
    Position         INTEGER := 1024;
    Buffer           RAW(32767);
BEGIN
    /* Select the LOB: */
    SELECT ad_photo INTO Lob_loc FROM Print_media
       WHERE product_id = 3060 AND ad_id = 11001;
    /* Opening the LOB is optional: */
    DBMS_LOB.OPEN (Lob_loc, DBMS_LOB.LOB_READONLY);
    Buffer := DBMS_LOB.SUBSTR(Lob_loc, Amount, Position);
    /* Process the data */
    /* Closing the LOB is mandatory if you have opened it: */
    DBMS_LOB.CLOSE (Lob_loc);
END;

/* In the following SQL statement, 255 is the amount to read
   and 1 is the starting offset from which to read: */
SELECT DBMS_LOB.SUBSTR(ad_photo, 255, 1) FROM Print_media WHERE product_id =
3060 AND ad_id = 11001;
```

COBOL（Pro*COBOL）: LOB の一部の読込み（substr）

```
* READING PORTION OF THE LOB DATA USING SUBSTR
  IDENTIFICATION DIVISION.
  PROGRAM-ID. BLOB-SUBSTR.
  ENVIRONMENT DIVISION.
  DATA DIVISION.
  WORKING-STORAGE SECTION.

  01  BLOB1          SQL-BLOB.
  01  BUFFER2        PIC X(32767) VARYING.
  01  AMT            PIC S9(9) COMP.
  01  POS            PIC S9(9) COMP VALUE 1.
  01  USERID   PIC X(11) VALUES "SAMP/SAMP".
      EXEC SQL INCLUDE SQLCA END-EXEC.
      EXEC SQL VAR BUFFER2 IS VARRAW(32767) END-EXEC.

  PROCEDURE DIVISION.
  BLOB-SUBSTR.
      EXEC SQL WHENEVER SQLERROR DO PERFORM SQL-ERROR END-EXEC.
      EXEC SQL
         CONNECT :USERID
      END-EXEC.

 * Allocate and initialize the CLOB locator:
      EXEC SQL ALLOCATE :BLOB1 END-EXEC.

      EXEC SQL WHENEVER NOT FOUND GOTO END-OF-BLOB END-EXEC.
        EXEC SQL
         SELECT AD_COMPOSITE INTO :BLOB1
         FROM PRINT_MEDIA PM WHERE PM.PRODUCT_ID = 2268
         AND AD_ID = 21001 END-EXEC.
      DISPLAY "Selected the BLOB".

 * Open the BLOB for READ ONLY:
      EXEC SQL LOB OPEN :BLOB1 READ ONLY END-EXEC.

 * Execute PL/SQL to get SUBSTR functionality:
      MOVE 5 TO AMT.
      EXEC SQL EXECUTE
        BEGIN
          :BUFFER2 := DBMS_LOB.SUBSTR(:BLOB1,:AMT,:POS); END; END-EXEC.
      EXEC SQL LOB CLOSE :BLOB1 END-EXEC.
      DISPLAY "Substr: ", BUFFER2-ARR(POS:AMT).
```

```
END-OF-BLOB.
    EXEC SQL WHENEVER NOT FOUND CONTINUE END-EXEC.
    EXEC SQL FREE :BLOB1 END-EXEC.
    EXEC SQL ROLLBACK WORK RELEASE END-EXEC.
    STOP RUN.

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

C/C++（Pro*C/C++）: LOB の一部の読込み（substr）

```
/* Reading portion of the LOB using (substr). Pro*C/C++ lacks an equivalent
   embedded SQL form for the DBMS_LOB.SUBSTR() function.
   However, Pro*C/C++ can interoperate with PL/SQL using anonymous
   PL/SQL blocks embedded in a Pro*C/C++ program as this example shows: */

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

#define BufferLength 32767

void substringLOB_proc()
{
  OCIBlobLocator *Lob_loc;
  int Position = 1;
  int Amount = BufferLength;
  struct {
    unsigned short Length;
    char Data[BufferLength];
  } Buffer;
```

```
  /* Datatype equivalencing is mandatory for this datatype: */
  EXEC SQL VAR Buffer IS VARRAW(BufferLength);

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  EXEC SQL ALLOCATE :Lob_loc;
  EXEC SQL SELECT ad_photo INTO Lob_loc
           FROM Print_media WHERE product_id = 3060 AND ad_id = 11001;
  /* Open the BLOB: */
  EXEC SQL LOB OPEN :Lob_loc READ ONLY;
  /* Invoke SUBSTR() from within an anonymous PL/SQL block: */
  EXEC SQL EXECUTE
    BEGIN
      :Buffer := DBMS_LOB.SUBSTR(:Lob_loc, :Amount, :Position);
    END;
  END-EXEC;
  /* Close the BLOB: */
  EXEC SQL LOB CLOSE :Lob_loc;
  /* Process the Data */
  /* Release resources used by the locator: */
  EXEC SQL FREE :Lob_loc;
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  substringLOB_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
  exit(0);
}
```

Visual Basic（OO4O）: LOB の一部の読込み（substr）

```
'Reading portion of a LOB (or BFILE). In OO4O this is accomplished by
'setting the OraBlob.Offset and OraBlob.chunksize properties.
'Using the OraClob.Read mechanism
Dim MySession As OraSession
Dim OraDb As OraDatabase
Dim OraDyn as OraDynaset, OraAdSourceText as OraClob, amount_read%, chunksize%,
chunk

Set MySession = CreateObject("OracleInProcServer.XOraSession")
Set OraDb = MySession.OpenDatabase("exampledb", "samp/samp", 0&)
```

```
Set OraDyn = OraDb.CreateDynaset("SELECT * FROM Print_media", ORADYN_DEFAULT)
Set OraAdSourceText = OraDyn.Fields("ad_sourcetext").Value

'Let's read 100 bytes from the 500th byte onwards:
OraAdSourceText.Offset = 500
OraAdSourceText.PollingAmount = OraAdSourceText.Size 'Read entire CLOB contents
amount_read = OraAdSourceText.Read(chunk, 100)
'chunk returned is a variant of type byte array
```

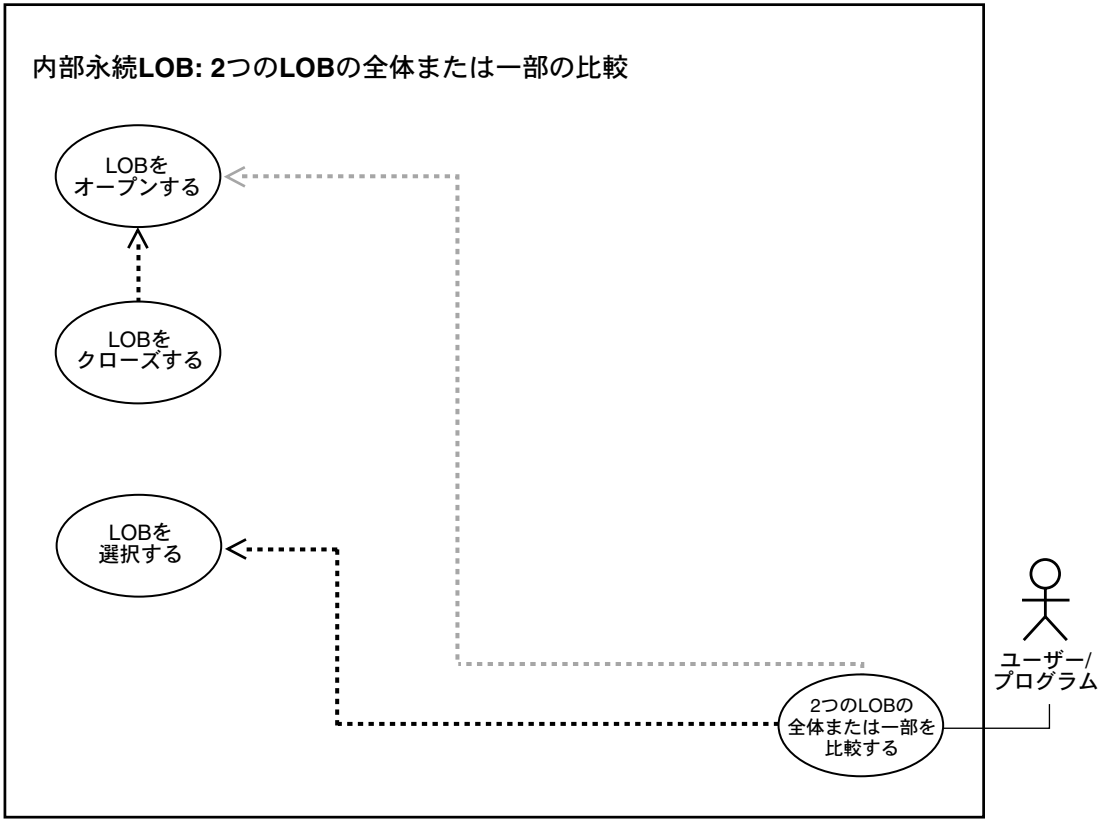
Java（JDBC）: LOB の一部の読込み（substr）

```
// Reading portion of a LOB using substr
import java.io.InputStream;
import java.io.OutputStream;

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

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

public class Ex2_79
{

  static final int MAXBUFSIZE = 32767;
  public static void main (String args [])
       throws Exception
  {
    // Load the Oracle JDBC driver:
    DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

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

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

```
    // Create a Statement:
    Statement stmt = conn.createStatement ();
    try
    {
       BLOB lob_loc = null;
       byte buf[] = new byte[MAXBUFSIZE];

       ResultSet rset = stmt.executeQuery (
          "SELECT ad_composite FROM Print_media
           WHERE product_id = 3106 AND ad_id = 13001");
       if (rset.next())
       {
          lob_loc = ((OracleResultSet)rset).getBLOB (1);
       }

       OracleCallableStatement cstmt = (OracleCallableStatement)
          conn.prepareCall ("BEGIN DBMS_LOB.OPEN(?, "
                            +"DBMS_LOB.LOB_READONLY); END;");
       cstmt.setBLOB(1, lob_loc);
       cstmt.execute();

       // MAXBUFSIZE is the number of bytes to read and 1000 is the offset from
       // which to start reading:
       buf = lob_loc.getBytes(1000, MAXBUFSIZE);
      // Display the contents of the buffer here.

      cstmt = (OracleCallableStatement)
         conn.prepareCall ("BEGIN DBMS_LOB.CLOSE(?); END;");
      cstmt.setBLOB(1, lob_loc);
      cstmt.execute();

      stmt.close();
      cstmt.close();
     conn.commit();
     conn.close();
    }
    catch (SQLException e)
    {
       e.printStackTrace();
    }
  }
}
```

2 つの LOB の全体または一部の比較

図 10-18 利用図 : 2 つの LOB の全体または一部の比較



参照 : 10-2 ページの表 10-1「利用モデル : 内部永続 LOB」を参照してください。

用途

2 つの 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関数またはファンクションのリストは、第 3 章「様々なプログラム環境での LOB のサポート」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）：『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』の第 23 章「DBMS_LOB」の「DBMS_LOB 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。
- C++（OCCI）：『Oracle C++ Call Interface プログラマーズ・ガイド』
- COBOL（Pro*COBOL）：『Pro*COBOL Precompiler プログラマーズ・ガイド』の LOB に関する詳細、LOB 文の使用上の注意と付録 F「埋込み SQL 文およびプリコンパイラ・ディレクティブ」の「EXECUTE（実行可能埋込み SQL）」、および『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』の第 23 章「DBMS_LOB」の「DBMS_LOB サブプログラムの要約」の「COMPARE ファンクション」
- C/C++（Pro*C/C++）：『Pro*C/C++ Precompiler プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびディレクティブ」の「EXECUTE（実行可能埋込み SQL）」、および『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』の第 23 章「DBMS_LOB」の「DBMS_LOB サブプログラムの要約」の「COMPARE ファンクション」
- Visual Basic（OO4O オンライン・ヘルプ）：ヘルプの「目次」タブから、「OO4O オートメーション・サーバー・リファレンス」>「OO4O サーバーのオブジェクト」>「OraDynaset」>「メソッド」>「movenext」を選択
- Java（JDBC）：『Oracle9*i* JDBC 開発者ガイドおよびリファレンス』の第 8 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『Oracle9*i* 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「JDBC 2.0 LOB 型と Oracle 拡張型のサポート」の「BLOB、CLOB および BFILE のサポート」

使用例

次の例では、`Print_media` 表の 2 つのイメージを比較して、異なっているかどうかを確認します。

例

次のプログラム環境での例を示します。

- PL/SQL（DBMS_LOB）：2 つの LOB の全体または一部の比較（10-124 ページ）
- C（OCI）：今回のリリースでは例は提供されません。
- C++（OCCI）：今回のリリースでは例は提供されません。

PL/SQL（DBMS_LOB）: 2 つの LOB の全体または一部の比較

```
/* Comparing all or part of two LOBs. The example procedure compareTwoLOBs_proc
   is not part of the DBMS_LOB package: */
CREATE OR REPLACE PROCEDURE compareTwoLOBs_proc IS
    Lob_loc1            BLOB;
    Lob_loc2            BLOB;
    Amount              INTEGER := 32767;
    Retval              INTEGER;
BEGIN
    /* Select the LOB: */
    SELECT ad_composite INTO Lob_loc1 FROM Print_media
        WHERE product_id = 3060 AND ad_id = 11001;
    SELECT ad_composite INTO Lob_loc2 FROM Print_media
        WHERE product_id = 2056 AND ad_id = 12001;
    /* Opening the LOB is optional: */
    DBMS_LOB.OPEN (Lob_loc1, DBMS_LOB.LOB_READONLY);
    DBMS_LOB.OPEN (Lob_loc2, DBMS_LOB.LOB_READONLY);
    /* Compare the two frames: */
    retval := DBMS_LOB.COMPARE(Lob_loc1, Lob_loc2, Amount, 1, 1);
    IF retval = 0 THEN
       DBMS_OUTPUT.PUT_LINE('Processing for equal frames');
    ELSE
       DBMS_OUTPUT.PUT_LINE('Processing for non-equal frames');
    END IF;
    /* Closing the LOB is mandatory if you have opened it: */
    DBMS_LOB.CLOSE (Lob_loc1);
    DBMS_LOB.CLOSE (Lob_loc2);
END;
```

COBOL（Pro*COBOL）: 2 つの LOB の全体または一部の比較

```
      * COMPARING ALL OR PART OF TWO LOBS
       IDENTIFICATION DIVISION.
       PROGRAM-ID. COMPARE.
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
```

```
 01  USERID   PIC X(11) VALUES "SAMP/SAMP".
 01  BLOB1          SQL-BLOB.
 01  BLOB2          SQL-BLOB.
 01  BUFFER2        PIC X(32767) VARYING.
 01  RET            PIC S9(9) COMP.
 01  AMT            PIC S9(9) COMP.
 01  POS            PIC S9(9) COMP VALUE 1024.
 01  OFFSET         PIC S9(9) COMP VALUE 1.

     EXEC SQL VAR BUFFER2 IS VARRAW(32767) END-EXEC.
     EXEC SQL INCLUDE SQLCA END-EXEC.

 PROCEDURE DIVISION.
 COMPARE-BLOB.
     EXEC SQL WHENEVER SQLERROR DO PERFORM SQL-ERROR END-EXEC.
     EXEC SQL
          CONNECT :USERID
     END-EXEC.
* Allocate and initialize the BLOB locators:
     EXEC SQL ALLOCATE :BLOB1 END-EXEC.
     EXEC SQL ALLOCATE :BLOB2 END-EXEC.
      EXEC SQL WHENEVER NOT FOUND GOTO END-OF-BLOB END-EXEC.
       EXEC SQL
          SELECT AD_COMPOSITE INTO :BLOB1
          FROM PRINT_MEDIA PM WHERE PM.PRODUCT_ID = 2268  AND AD_ID = 21001
       END-EXEC.
      EXEC SQL
          SELECT AD_COMPOSITE INTO :BLOB2
          FROM PRINT_MEDIA PM WHERE PM.PRODUCT_ID = 3060 AND AD_ID = 11001
          END-EXEC.

* Open the BLOBs for READ ONLY:
     EXEC SQL LOB OPEN :BLOB1 READ ONLY END-EXEC.
     EXEC SQL LOB OPEN :BLOB2 READ ONLY END-EXEC.

* Execute PL/SQL to get COMPARE functionality:
     MOVE 4 TO AMT.
     EXEC SQL EXECUTE
       BEGIN
         :RET := DBMS_LOB.COMPARE(:BLOB1,:BLOB2,:AMT,1,1); END; END-EXEC.

     IF RET = 0
*        Logic for equal BLOBs goes here
         DISPLAY "BLOBs are equal"
     ELSE
*        Logic for unequal BLOBs goes here
         DISPLAY "BLOBs are not equal"
```

```
            END-IF.
            EXEC SQL LOB CLOSE :BLOB1 END-EXEC.
            EXEC SQL LOB CLOSE :BLOB2 END-EXEC.

        END-OF-BLOB.
            EXEC SQL WHENEVER NOT FOUND CONTINUE END-EXEC.
            EXEC SQL FREE :BLOB1 END-EXEC.
            EXEC SQL FREE :BLOB2 END-EXEC.
            EXEC SQL ROLLBACK WORK RELEASE END-EXEC.
            STOP RUN.

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

C/C++（Pro*C/C++）: 2 つの LOB の全体または一部の比較

```
/* Comparing all or part of two LOBs
#include <oci.h>
#include <stdio.h>
#include <sqlca.h>

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

void compareTwoLobs_proc()
{
  OCIBlobLocator *Lob_loc1, *Lob_loc2;
  int Amount = 32767;
  int Retval;

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  /* Allocate the LOB locators: */
  EXEC SQL ALLOCATE :Lob_loc1;
  EXEC SQL ALLOCATE :Lob_loc2;
  /* Select the LOBs: */
```

```
  EXEC SQL SELECT ad_composite INTO :Lob_loc1
     FROM Print_media WHERE product_id = 3060 AND ad_id = 11001;
  EXEC SQL SELECT ad_composite INTO :Lob_loc2
     FROM Print_media WHERE product_id = 2056 AND ad_id = 12001;
  /* Opening the LOBs is Optional: */
  EXEC SQL LOB OPEN :Lob_loc1 READ ONLY;
  EXEC SQL LOB OPEN :Lob_loc2 READ ONLY;
  /* Compare the two Frames using DBMS_LOB.COMPARE() from within PL/SQL: */
  EXEC SQL EXECUTE
     BEGIN
        :Retval := DBMS_LOB.COMPARE(:Lob_loc1, :Lob_loc2, :Amount, 1, 1);
     END;
  END-EXEC;
  if (0 == Retval)
     printf("The frames are equal\n");
  else
     printf("The frames are not equal\n");
  /* Closing the LOBs is mandatory if you have opened them: */
  EXEC SQL LOB CLOSE :Lob_loc1;
  EXEC SQL LOB CLOSE :Lob_loc2;
  /* Release resources held by the locators: */
  EXEC SQL FREE :Lob_loc1;
  EXEC SQL FREE :Lob_loc2;
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  compareTwoLobs_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

Visual Basic（OO4O）: 2 つの LOB の全体または一部の比較

```
'Comparing all or part of two LOBs
Dim MySession As OraSession
Dim OraDb As OraDatabase

Set MySession = CreateObject("OracleInProcServer.XOraSession")
Set OraDb = MySession.OpenDatabase("exampledb", "samp/samp", 0&)
Dim OraDyn as OraDynaset, OraAdPhoto1 as OraBLOB, OraAdPhotoClone as OraBLOB

Set OraDyn = OraDb.CreateDynaset(
   "SELECT * FROM Print_media ORDER BY product_id, ad_id", ORADYN_DEFAULT)
Set OraAdPhoto1 = OraDyn.Fields("ad_photo").Value
```

```
'Clone it for future reference
Set OraAdPhotoClone = OraAdPhoto1.Clone

'Lets go to the next row and compare LOBs
OraDyn.MoveNext

MsgBox CBool(OraAdPhotot1.Compare(OraAdPhototClone, OraAdPhotoClone.size, 1, 1))
```

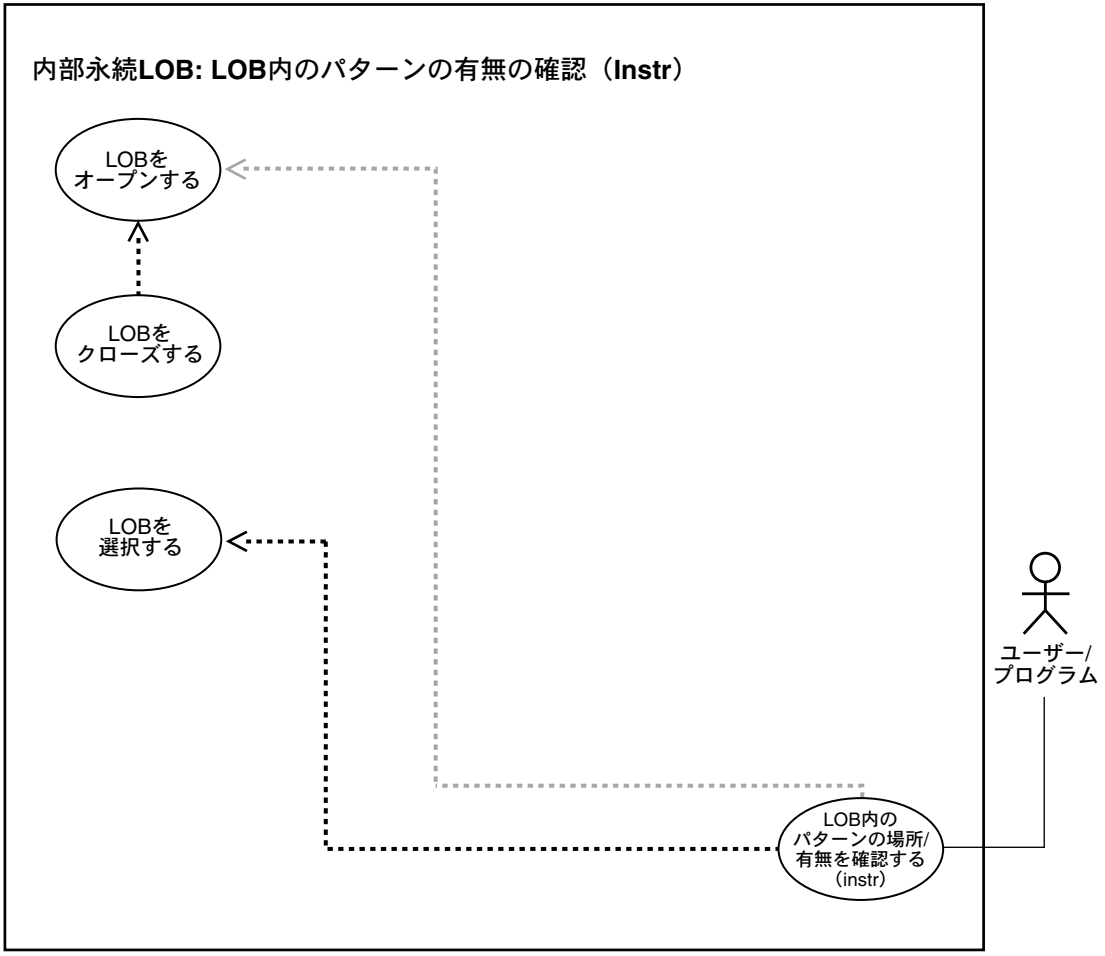
Java（JDBC）: 2 つの LOB の全体または一部の比較

```
// Comparing all or part of two LOBs
import java.io.InputStream;
import java.io.OutputStream;

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

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

public class Ex2_87
{
  static final int MAXBUFSIZE = 32767;
  public static void main (String args [])
       throws Exception
  {
    // Load the Oracle JDBC driver:
    DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

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

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

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

```
    {
     BLOB lob_loc1 = null;
     BLOB lob_loc2 = null;
       ResultSet rset = stmt.executeQuery (
          "SELECT ad_composite FROM Print_media
           WHERE product_id = 2056 AND ad_id = 12001");
   if (rset.next())
   {
     lob_loc1 = ((OracleResultSet)rset).getBLOB (1);
   }

       rset = stmt.executeQuery (
          "SELECT ad_composite FROM Print_media
           WHERE product_id = 3106 AND ad_id = 13001");
   if (rset.next())
   {
     lob_loc2 = ((OracleResultSet)rset).getBLOB (1);
   }

   if (lob_loc1.length() > lob_loc2.length())
     System.out.println ("Looking for LOB2 inside LOB1. result = "
       + Long.toString(lob_loc1.position(lob_loc2, 1)));
   else
     System.out.println("Looking for LOB1 inside LOB2.  result = "
     + Long.toString(lob_loc2.position(lob_loc1, 1)));

   stmt.close();
   conn.commit();
   conn.close();
    }
    catch (SQLException e)
    {
       e.printStackTrace();
    }
  }
}
```

パターン : LOB 内のパターンの有無の確認（instr）

図 10-19 利用図 : LOB 内のパターンの有無の確認（instr）



参照： 10-2 ページの表 10-1「利用モデル : 内部永続 LOB」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関数またはファンクションのリストは、第 3 章「様々なプログラム環境での LOB のサポート」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）：『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』の第 23 章「DBMS_LOB」の「DBMS_LOB 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。
- C++（OCCI）：『Oracle C++ Call Interface プログラマーズ・ガイド』
- COBOL（Pro*COBOL）：『Pro*COBOL Precompiler プログラマーズ・ガイド』の LOB に関する詳細、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・ディレクティブ」の「EXECUTE（実行可能埋込み SQL）」、および『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』の第 23 章「DBMS_LOB」の「DBMS_LOB サブプログラムの要約」の「INSTR ファンクション」
- C/C++（Pro*C/C++）：『Pro*C/C++ Precompiler プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびディレクティブ」の「EXECUTE（実行可能埋込み SQL）」、および『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』の第 23 章「DBMS_LOB」の「DBMS_LOB 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）：『Oracle9*i* JDBC 開発者ガイドおよびリファレンス』の第 8 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『Oracle9*i* 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「JDBC 2.0 LOB 型と Oracle 拡張型のサポート」の「BLOB、CLOB および BFILE のサポート」

使用例

次の例では、`ad_sourcetext` 列の広告テキストに「children」という文字列が存在するかどうかを確認します。

例

次のプログラム環境での例を示します。

PL/SQL（DBMS_LOB）: LOB 内のパターンの有無の確認（instr）

```
/* Seeing if a pattern exists in the LOB using instr.
   The example procedure instringLOB_proc is not part of the
   DBMS_LOB package: */
CREATE OR REPLACE PROCEDURE instringLOB_proc IS
   Lob_loc        CLOB;
   Pattern        VARCHAR2(30) := 'children';
   Position       INTEGER := 0;
   Offset         INTEGER := 1;
   Occurrence     INTEGER := 1;
BEGIN
   /* Select the LOB: */
   SELECT ad_sourcetext INTO Lob_loc
      FROM Print_media WHERE product_id = 2268 AND ad_id = 21001;
   /* Opening the LOB is optional: */
   DBMS_LOB.OPEN (Lob_loc, DBMS_LOB.LOB_READONLY);
   /* Seek for the pattern: */
   Position := DBMS_LOB.INSTR(Lob_loc, Pattern, Offset, Occurrence);
   IF Position = 0 THEN
      DBMS_OUTPUT.PUT_LINE('Pattern not found');
   ELSE
      DBMS_OUTPUT.PUT_LINE('The pattern occurs at '|| position);
   END IF;
   /* Closing the LOB is mandatory if you have opened it: */
   DBMS_LOB.CLOSE (Lob_loc);
END;
```

COBOL（Pro*COBOL）: LOB 内のパターンの有無の確認（instr）

```
* SEEING IF A PATTERN EXISTS IN THE LOB USING INSTR
  IDENTIFICATION DIVISION.
  PROGRAM-ID. CLOB-INSTR.
  ENVIRONMENT DIVISION.
  DATA DIVISION.
  WORKING-STORAGE SECTION.
  01  CLOB1          SQL-CLOB.
  01  PATTERN        PIC X(8) VALUE "children".
  01  POS            PIC S9(9) COMP.
  01  OFFSET         PIC S9(9) COMP VALUE 1.
  01  OCCURRENCE     PIC S9(9) COMP VALUE 1.
  01  USERID   PIC X(11) VALUES "SAMP/SAMP".
      EXEC SQL INCLUDE SQLCA END-EXEC.

  PROCEDURE DIVISION.
  CLOB-INSTR.
      EXEC SQL WHENEVER SQLERROR DO PERFORM SQL-ERROR END-EXEC.
      EXEC SQL
           CONNECT :USERID
      END-EXEC.

 * Allocate and initialize the CLOB locator:
      EXEC SQL ALLOCATE :CLOB1 END-EXEC.
       EXEC SQL WHENEVER NOT FOUND GOTO END-OF-CLOB END-EXEC.
        EXEC SQL SELECT AD_SOURCETEXT INTO :CLOB1
           FROM PRINT_MEDIA
           WHERE PRODUCT_ID = 2268 AND AD_ID = 21001 END-EXEC.

 * Open the CLOB for READ ONLY:
      EXEC SQL LOB OPEN :CLOB1 READ ONLY END-EXEC.

 * Execute PL/SQL to get INSTR functionality:
      EXEC SQL EXECUTE
        BEGIN
          :POS := DBMS_LOB.INSTR(:CLOB1, :PATTERN,:OFFSET,:OCCURRENCE);
          END; END-EXEC.

      IF POS = 0
 *        Logic for pattern not found here
          DISPLAY "Pattern not found."
      ELSE
 *        Pos contains position where pattern is found
          DISPLAY "Pattern found."
      END-IF.
      EXEC SQL LOB CLOSE :CLOB1 END-EXEC.
```

```
END-OF-CLOB.
    EXEC SQL WHENEVER NOT FOUND CONTINUE END-EXEC.
    EXEC SQL FREE :CLOB1 END-EXEC.
    EXEC SQL ROLLBACK WORK RELEASE END-EXEC.
    STOP RUN.

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

C/C++（Pro*C/C++）: LOB 内のパターンの有無の確認（instr）

```
/* Seeing if a pattern exists in the LOB using instr
#include <oci.h>
#include <stdio.h>
#include <sqlca.h>

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

void instringLOB_proc()
{
  OCIClobLocator *Lob_loc;
  char *Pattern = "The End";
  int Position = 0;
  int Offset = 1;
  int Occurrence = 1;

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  EXEC SQL ALLOCATE :Lob_loc;
  EXEC SQL SELECT ad_sourcetext INTO :Lob_loc
           FROM Print_media WHERE product_id = 3060 AND ad_id = 11001;
  /* Opening the LOB is Optional: */
  EXEC SQL LOB OPEN :Lob_loc;
  /* Seek the Pattern using DBMS_LOB.INSTR() in a PL/SQL block: */
```

```
  EXEC SQL EXECUTE
    BEGIN
      :Position := DBMS_LOB.INSTR(:Lob_loc, :Pattern, :Offset, :Occurrence);
    END;
  END-EXEC;
  if (0 == Position)
    printf("Pattern not found\n");
  else
    printf("The pattern occurs at %d\n", Position);
  /* Closing the LOB is mandatory if you have opened it: */
  EXEC SQL LOB CLOSE :Lob_loc;
  EXEC SQL FREE :Lob_loc;
}

void main()
{
  char *samp = "pm/pm";
  EXEC SQL CONNECT :pm;
  instringLOB_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

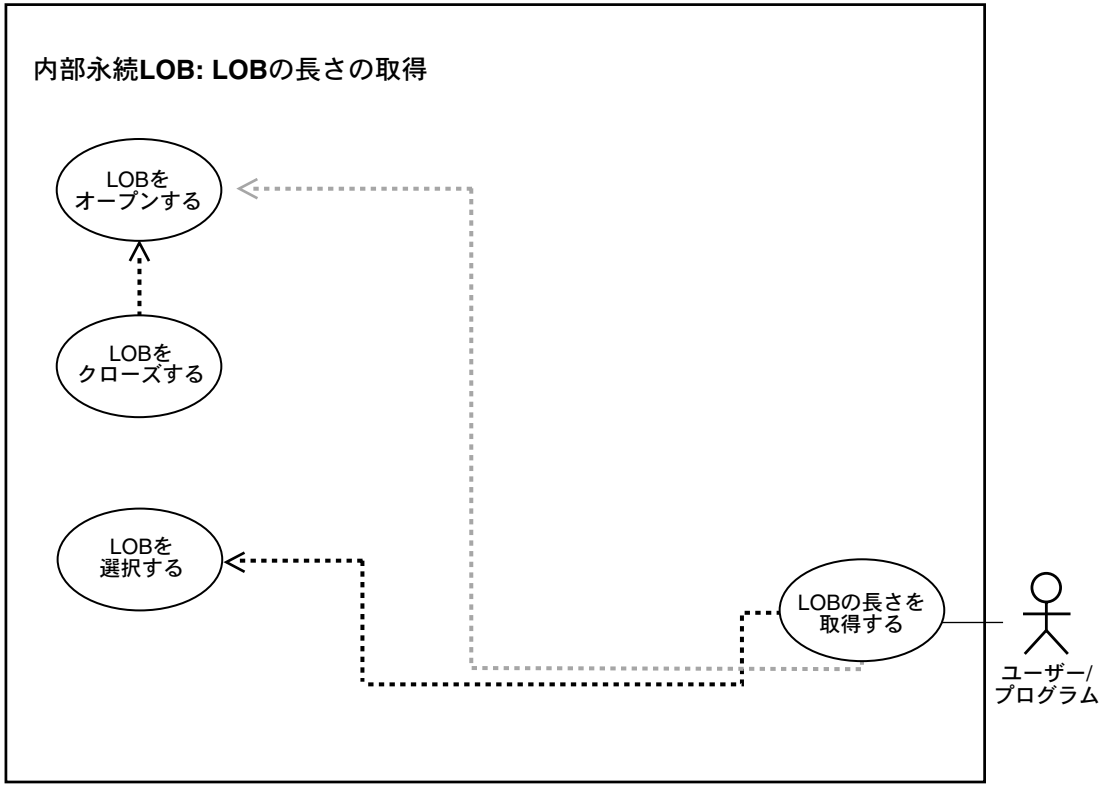
Java（JDBC）: LOB 内のパターンの有無の確認（instr）

```
// Seeing if a pattern exists in the LMOB using instr
import java.io.OutputStream;

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

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

public class Ex2_91
{
  static final int MAXBUFSIZE = 32767;
  public static void main (String args [])
       throws Exception
  {
```

```
    // Load the Oracle JDBC driver:
    DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

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

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

    // Create a Statement:
    Statement stmt = conn.createStatement ();
    try
    {
     final int offset = 1;       // Start looking at the first byte
     final int occurrence = 1;  // Start at the 1st occurrence of the pattern
within the CLOB

   CLOB lob_loc = null;
   String pattern = new String("Junk"); // Pattern to look for within the CLOB.

       ResultSet rset = stmt.executeQuery (
          "SELECT ad_sourcetext FROM Print_media
           WHERE product_id = 2268 AND ad_id = 21001");
   if (rset.next())
   {
     lob_loc = ((OracleResultSet)rset).getCLOB (1);
   }

   // Search for location of pattern string in the CLOB, starting at offset 1:
   long result = lob_loc.position(pattern, offset);
   System.out.println("Results of Pattern Comparison : " +
      Long.toString(result));

   stmt.close();
   conn.commit();
   conn.close();

    }
    catch (SQLException e)
    {
       e.printStackTrace();
    }
  }
}
```

長さ : LOB の長さの取得

図 10-20 利用図 : LOB の長さの取得



参照 : 10-2 ページの表 10-1「利用モデル : 内部永続 LOB」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関数またはファンクションのリストは、第 3 章「様々なプログラム環境での LOB のサポート」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）：『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』の第 23 章「DBMS_LOB」の「DBMS_LOB サブプログラムの要約」の「GETLENGTH ファンクション」
- C（OCI）：『Oracle Call Interface プログラマーズ・ガイド』の第 16 章「その他の OCI リレーショナル関数」の「LOB 関数」の OCILobGetLength()
- C++（OCCI）：『Oracle C++ Call Interface プログラマーズ・ガイド』
- COBOL（Pro*COBOL）：『Pro*COBOL Precompiler プログラマーズ・ガイド』の LOB に関する詳細、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・ディレクティブ」の「LOB DESCRIBE（実行可能埋込み SQL 拡張機能）」
- C/C++（Pro*C/C++）：『Pro*C/C++ Precompiler プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびディレクティブ」の「LOB DESCRIBE（実行可能埋込み SQL 拡張機能）」
- Visual Basic（OO4O オンライン・ヘルプ）: ヘルプの「目次」タブから、「OO4O オートメーション・サーバー・リファレンス」>「OO4O サーバーのオブジェクト」>「OraDynaset」>「プロパティ」>「fields」を選択
- Java（JDBC）：『Oracle9*i* JDBC 開発者ガイドおよびリファレンス』の第 8 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『Oracle9*i* 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「JDBC 2.0 LOB 型と Oracle 拡張型のサポート」の「BLOB、CLOB および BFILE のサポート」

使用例

次の例では、外国語のテキスト列（`ad_fltextn`）内の `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での例を示します。

- PL/SQL（DBMS_LOB）: LOB の長さの取得（10-139 ページ）
- C（OCI）: LOB の長さの取得（10-139 ページ）
- C++（OCCI）: 今回のリリースでは例は提供されません。
- COBOL（Pro*COBOL）: LOB の長さの取得（10-141 ページ）
- C/C++（Pro*C/C++）: LOB の長さの取得（10-142 ページ）

- Visual Basic（OO4O）: LOB の長さの取得（10-143 ページ）
- Java（JDBC）: LOB の長さの取得（10-143 ページ）

PL/SQL（DBMS_LOB）: LOB の長さの取得

```
/* Getting the length of a LOB.
   Example procedure getLengthLOB_proc is not part of the
   DBMS_LOB package: */
CREATE OR REPLACE PROCEDURE getLengthLOB_proc IS
   Lob_loc     NCLOB;
   Length      INTEGER;
BEGIN
   /* Select the LOB: */
   SELECT ad_fltextn INTO Lob_loc FROM Print_media
       WHERE product_id = 3106 AND ad_id = 13001;
   /* Opening the LOB is optional: */
   DBMS_LOB.OPEN (Lob_loc, DBMS_LOB.LOB_READONLY);
   /* Get the length of the LOB: */
   length := DBMS_LOB.GETLENGTH(Lob_loc);
   IF length IS NULL THEN
       DBMS_OUTPUT.PUT_LINE('LOB is null.');
   ELSE
       DBMS_OUTPUT.PUT_LINE('The length is '|| length);
   END IF;
   /* Closing the LOB is mandatory if you have opened it: */
   DBMS_LOB.CLOSE (Lob_loc);
END;
```

C（OCI）: LOB の長さの取得

```
/* Getting the length of a LOB
/* Select the locator into a locator variable */
sb4 select_adfltextn_locator(Lob_loc, errhp, svchp, stmthp)
OCILobLocator *Lob_loc;
OCIError      *errhp;
OCISvcCtx     *svchp;
OCIStmt       *stmthp;
{
  OCIDefine *defnp1;

  text  *sqlstmt =
    (text *)"SELECT ad_fltextn FROM Print_media
            WHERE product_id = 2268";
```

```
  checkerr (errhp, OCIStmtPrepare(stmthp, errhp, sqlstmt,
                                  (ub4)strlen((char *)sqlstmt),
                                  (ub4)OCI_NTV_SYNTAX, (ub4)OCI_DEFAULT));
  /* Define the column being selected */
  checkerr (errhp, OCIDefineByPos(stmthp, &defnp1, errhp, (ub4) 1,
                                  (dvoid *)&Lob_loc, (sb4)0,
                                  (ub2)SQLT_CLOB,(dvoid *)0, (ub2 *)0,
                                  (ub2 *)0, (ub4)OCI_DEFAULT));
   /* Execute and fetch one row */
  checkerr (errhp, OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                                  (CONST OCISnapshot*) 0, (OCISnapshot*) 0,
                                  (ub4) OCI_DEFAULT));
  return 0;
}

/* This function gets the length of the selected LOB */
void getLengthLob(envhp, errhp, svchp, stmthp)
OCIEnv    *envhp;
OCIError  *errhp;
OCISvcCtx *svchp;
OCIStmt   *stmthp;
{
  ub4 length;
  OCILobLocator *Lob_loc;
    /* Allocate Locator resources */
  (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &Lob_loc,
                            (ub4)OCI_DTYPE_LOB, (size_t) 0, (dvoid **) 0);

  /* Select a LOB locator from FLSub */
  printf(" select a adfltextn locator\n");
  select_adfltextn_locator(Lob_loc, errhp, svchp, stmthp);

  /* Opening the LOB is Optional */
  printf(" Open the locator (optional)\n");
  checkerr (errhp, (OCILobOpen(svchp, errhp, Lob_loc, OCI_LOB_READONLY)));

  printf(" get the length of ad_fltextn.\n");
  checkerr (errhp, OCILobGetLength(svchp, errhp, Lob_loc, &length));

  /* Length is undefined if the LOB is NULL or undefined */
  fprintf(stderr," Length of LOB is %d\n",length);

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

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

return;
}
```

COBOL（Pro*COBOL）: LOB の長さの取得

```
* GETTING THE LENGTH OF A LOB
  IDENTIFICATION DIVISION.
  PROGRAM-ID. LOB-LENGTH.
  ENVIRONMENT DIVISION.
  DATA DIVISION.
  WORKING-STORAGE SECTION.

  01  CLOB1          SQL-CLOB.
  01  LOB-ATTR-GRP.
      05 LEN         PIC S9(9) COMP.
  01  D-LEN          PIC 9(4).
  01  USERID   PIC X(11) VALUES "SAMP/SAMP".
      EXEC SQL INCLUDE SQLCA END-EXEC.
  PROCEDURE DIVISION.
  LOB-LENGTH.
      EXEC SQL WHENEVER SQLERROR DO PERFORM SQL-ERROR END-EXEC.
      EXEC SQL CONNECT :USERID END-EXEC.

 * Allocate and initialize the target CLOB:
      EXEC SQL ALLOCATE :CLOB1 END-EXEC.
      EXEC SQL WHENEVER NOT FOUND GOTO END-OF-CLOB END-EXEC.
      EXEC SQL
         SELECT AD_SOURCETEXT INTO :CLOB1
         FROM PRINT_MEDIA
         WHERE PRODUCT_ID = 3060 AND AD_ID = 11001 END-EXEC.

 * Obtain the length of the CLOB:
      EXEC SQL
         LOB DESCRIBE :CLOB1 GET LENGTH INTO :LEN END-EXEC.
      MOVE LEN TO D-LEN.
      DISPLAY "The length is ", D-LEN.

 * Free the resources used by the CLOB:
  END-OF-CLOB.
      EXEC SQL WHENEVER NOT FOUND CONTINUE END-EXEC.
      EXEC SQL FREE :CLOB1 END-EXEC.
      EXEC SQL ROLLBACK WORK RELEASE END-EXEC.
      STOP RUN.
```

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

C/C++（Pro*C/C++）: LOB の長さの取得

```
/* Getting the length of a LOB */
#include <oci.h>
#include <stdio.h>
#include <sqlca.h>

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

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  EXEC SQL ALLOCATE :Lob_loc;
  EXEC SQL SELECT ad_sourcetext INTO :Lob_loc
           FROM Print_media WHERE product_id = 3060 AND ad_id = 11001;
  /* Opening the LOB is Optional: */
  EXEC SQL LOB OPEN :Lob_loc READ ONLY;
  /* Get the Length: */
  EXEC SQL LOB DESCRIBE :Lob_loc GET LENGTH INTO :Length;
  /* If the LOB is NULL or unitialized, then Length is Undefined: */
  printf("Length is %d characters\n", Length);
  /* Closing the LOB is mandatory if you have Opened it: */
  EXEC SQL LOB CLOSE :Lob_loc;
  EXEC SQL FREE :Lob_loc;
}

void main()
{
```

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

Visual Basic（OO4O）: LOB の長さの取得

```
'Getting the length of a LOB
Dim MySession As OraSession
Dim OraDb As OraDatabase

Dim OraDyn As OraDynaset, OraAdPhoto1 As OraBlob, OraAdPhotoClone As OraBlob

Set MySession = CreateObject("OracleInProcServer.XOraSession")
Set OraDb = MySession.OpenDatabase("exampledb", "samp/samp", 0&)
Set OraDyn = OraDb.CreateDynaset(
    "SELECT * FROM Print_media ORDER BY product_id, ad_id", ORADYN_DEFAULT)
Set OraAdPhoto1 = OraDyn.Fields("ad_photo").Value

'Display out size of the lob:
MsgBox "Length of the lob is " & OraAdPhoto1.Size
```

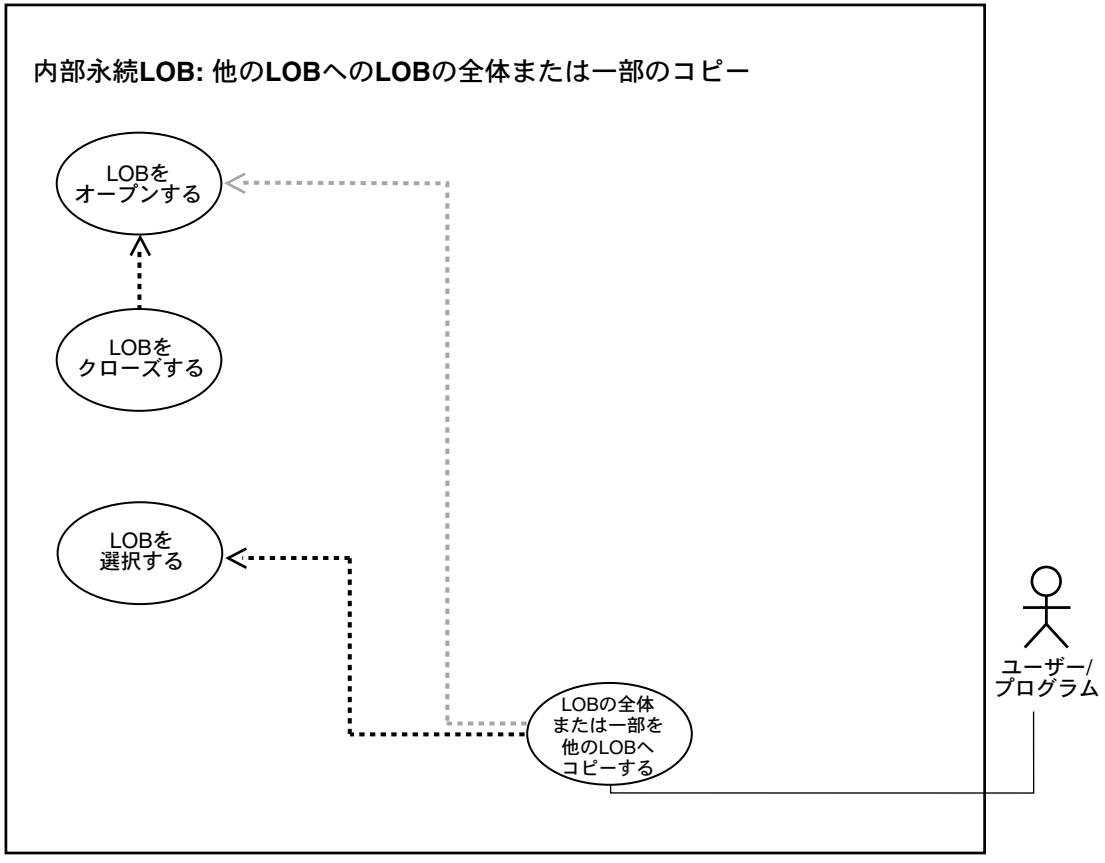
Java（JDBC）: LOB の長さの取得

```
//Getting the length of a LOB
import java.io.OutputStream;

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

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

public class Ex2_95
{
```

```
  static final int MAXBUFSIZE = 32767;
  public static void main (String args [])
       throws Exception
  {
    // Load the Oracle JDBC driver:
    DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

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

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

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

    try
    {
     CLOB lob_loc = null;
      ResultSet rset = stmt.executeQuery
          ("SELECT ad_sourcetext FROM Print_media WHERE product_id = 3106");
   if (rset.next())
   {
     lob_loc = ((OracleResultSet)rset).getCLOB (1);
   }

   System.out.println(
      "Length of this column is : " + Long.toString(lob_loc.length()));

   stmt.close();
   conn.commit();
   conn.close();
    }
    catch (SQLException e)
    {
       e.printStackTrace();
    }
  }
}
```

他の LOB への LOB の全体または一部のコピー

図 10-21 利用図 : 他の LOB への LOB の全体または一部のコピー



参照 : 10-2 ページの表 10-1「利用モデル : 内部永続 LOB」を参照してください。

用途

LOB の全体または一部を、他の 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、5-5 ページの「更新済ロケータを介した LOB の更新」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関数またはファンクションのリストは、第 3 章「様々なプログラム環境での LOB のサポート」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）：『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』の第 23 章「DBMS_LOB」の「DBMS_LOB サブプログラムの要約」の「COPY プロシージャ」
- C（OCI）：『Oracle Call Interface プログラマーズ・ガイド』の第 16 章「その他の OCI リレーショナル関数」の「LOB 関数」の OCILobCopy()
- C++（OCCI）：『Oracle C++ Call Interface プログラマーズ・ガイド』
- COBOL（Pro*COBOL）：『Pro*COBOL Precompiler プログラマーズ・ガイド』の LOB に関する詳細、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・ディレクティブ」の「LOB COPY（実行可能埋込み SQL 拡張機能）」、および『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』の第 23 章「DBMS_LOB」の「DBMS_LOB サブプログラムの要約」の「COPY プロシージャ」
- C/C++（Pro*C/C++）：『Pro*C/C++ Precompiler プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびディレクティブ」の「LOB COPY（実行可能埋込み SQL 拡張機能）」
- Visual Basic（OO4O オンライン・ヘルプ）：ヘルプの「目次」タブから、「OO4O オートメーション・サーバー・リファレンス」>「OO4O サーバーのオブジェクト」>「OraBLOB、OraCLOB」>「メソッド」>「copy」、および「OO4O オートメーション・サーバー・リファレンス」>「OO4O サーバーのオブジェクト」>「OraDynaset」>「メソッド」>「movenext」、「edit」を選択
- Java（JDBC）：『Oracle9*i* JDBC 開発者ガイドおよびリファレンス』の第 8 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『Oracle9*i* 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「JDBC 2.0 LOB 型と Oracle 拡張型のサポート」の「BLOB、CLOB および BFILE のサポート」

使用例

次の例では、ad_photo 列から ad_composite 列にイメージの一部をコピーします。

例

次のプログラム環境での例を示します。

- PL/SQL（DBMS_LOB）: 他の LOB への LOB の全体または一部のコピー（10-147 ページ）
- C（OCI）: 他の LOB への LOB の全体または一部のコピー（10-148 ページ）
- COBOL（Pro*COBOL）: 他の LOB への LOB の全体または一部のコピー（10-150 ページ）
- C/C++（Pro*C/C++）: 他の LOB への LOB の全体または一部のコピー（10-152 ページ）
- Visual Basic（OO4O）: 他の LOB への LOB の全体または一部のコピー（10-153 ページ）
- Java（JDBC）: 他の LOB への LOB の全体または一部のコピー（10-154 ページ）

PL/SQL（DBMS_LOB）: 他の LOB への LOB の全体または一部のコピー

```
/* Copying all or part of a LOB to another LOB.
   Example procedure copyLOB_proc is not part of DBMS_LOB package: */
CREATE OR REPLACE PROCEDURE copyLOB_proc IS
   Dest_loc     BLOB;
   Src_loc      BLOB;
   Amount       NUMBER;
   Dest_pos     NUMBER;
   Src_pos      NUMBER;
BEGIN
   SELECT ad_composite INTO Dest_loc FROM Print_media
      WHERE product_id = 3106 AND ad_id = 13001 FOR UPDATE;
   /* Select the LOB: */
   SELECT ad_photo INTO Src_loc FROM Print_media
      WHERE product_id = 2056 AND ad_id = 12001;
   /* Opening the LOBs is optional: */
   DBMS_LOB.OPEN(Dest_loc, DBMS_LOB.LOB_READWRITE);
   DBMS_LOB.OPEN(Src_loc, DBMS_LOB.LOB_READONLY);
   /* Copies the LOB from the source position to the destination position: */
   DBMS_LOB.COPY(Dest_loc, Src_loc, Amount, Dest_pos, Src_pos);
   /* Closing LOBs is mandatory if you have opened them: */
   DBMS_LOB.CLOSE(Dest_loc);
```

```
   DBMS_LOB.CLOSE(Src_loc);
   COMMIT;
EXCEPTION
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('Operation failed');
END;
```

C（OCI）: 他の LOB への LOB の全体または一部のコピー

```
/* Copying all or part of a LOB to another LOB */
/* Select the locator */
sb4 select_photo_locator_2(Lob_loc, dest_type, errhp, svchp, stmthp)
OCILobLocator *Lob_loc;
ub1           dest_type;                    /* whether destination locator */
OCIError      *errhp;
OCISvcCtx     *svchp;
OCIStmt       *stmthp;
{
  char        sqlstmt[150];
  OCIDefine   *defnp1;
  if (dest_type == TRUE)
  {
    strcpy (sqlstmt,
           (char *)"SELECT ad_photo FROM Print_media
              WHERE product_id=2268 FOR UPDATE");
    printf ("  select destination ad_photo locator\n");
  }
  else
  {
    strcpy(sqlstmt, (char *)"SELECT ad_photo FROM Print_media WHERE product_
id=3106");
    printf ("  select source ad_photo locator\n");
  }
  checkerr (errhp, OCIStmtPrepare(stmthp, errhp, (text *)sqlstmt,
                                  (ub4)strlen((char *)sqlstmt),
                                  (ub4) OCI_NTV_SYNTAX, (ub4) OCI_DEFAULT));
  checkerr (errhp, OCIDefineByPos(stmthp, &defnp1, errhp, (ub4) 1,
                                  (dvoid *)&Lob_loc, (sb4)0,
                                  (ub2) SQLT_BLOB,(dvoid *) 0,
                                   (ub2 *) 0, (ub2 *) 0, (ub4) OCI_DEFAULT));
  /* execute the select and fetch one row */
  checkerr(errhp, OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                                 (CONST OCISnapshot*) 0, (OCISnapshot*) 0,
                                 (ub4) OCI_DEFAULT));
```

```
  return 0;
}

/* This function copies part of the Source LOB into a specified position
   in the destination LOB
 */
void copyAllPartLob(envhp, errhp, svchp, stmthp)
OCIEnv    *envhp;
OCIError  *errhp;
OCISvcCtx *svchp;
OCIStmt   *stmthp;
{
  OCIBlobLocator *Dest_loc, *Src_loc;
  int Amount = 1000;                                  /* <Amount to Copy> */
  int Dest_pos = 100;                      /*<Position to start copying into> */
  int Src_pos = 1;                         /* <Position to start copying from> */

  /* Allocate the LOB locators */
  (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &Dest_loc,
                            (ub4)OCI_DTYPE_LOB, (size_t) 0, (dvoid **) 0);
  (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &Src_loc,
                            (ub4)OCI_DTYPE_LOB, (size_t) 0, (dvoid **) 0);

  /* Select the LOBs */
  printf(" select the destination and source locators\n");
  select_photo_locator_2(Dest_loc, TRUE, errhp, svchp, stmthp);
                                                     /* destination locator */
  select_photo_locator_2(Src_loc, FALSE, errhp, svchp, stmthp);
                                                         /* source locator */

  /* Opening the LOBs is Optional */
  printf (" open the destination locator (optional)\n");
  checkerr (errhp, OCILobOpen(svchp, errhp, Dest_loc, OCI_LOB_READWRITE));
  printf (" open the source locator (optional)\n");
  checkerr (errhp, OCILobOpen(svchp, errhp, Src_loc, OCI_LOB_READONLY));

  printf (" copy the lob (amount) from the source to destination\n");
  checkerr (errhp, OCILobCopy(svchp, errhp, Dest_loc, Src_loc,
                              Amount, Dest_pos, Src_pos));

  /* Closing the LOBs is Mandatory if they have been Opened */
  printf(" close the locators\n");
  checkerr (errhp, OCILobClose(svchp, errhp, Dest_loc));
  checkerr (errhp, OCILobClose(svchp, errhp, Src_loc));
```

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

return;
}
```

COBOL（Pro*COBOL）: 他の LOB への LOB の全体または一部のコピー

```
* COPYING ALL OR PART OF A LOB TO ANOTHER LOB
  IDENTIFICATION DIVISION.
  PROGRAM-ID. BLOB-COPY.
  ENVIRONMENT DIVISION.
  DATA DIVISION.
  WORKING-STORAGE SECTION.

  01  USERID   PIC X(11) VALUES "SAMP/SAMP".
  01  DEST           SQL-BLOB.
  01  SRC            SQL-BLOB.

 * Define the amount to copy.
 * This value has been chosen arbitrarily:
  01  AMT            PIC S9(9) COMP VALUE 10.
 * Define the source and destination position.
 * These values have been chosen arbitrarily:
  01  SRC-POS        PIC S9(9) COMP VALUE 1.
  01  DEST-POS       PIC S9(9) COMP VALUE 1.

 * The return value from PL/SQL function:
  01  RET            PIC S9(9) COMP.
      EXEC SQL INCLUDE SQLCA END-EXEC.
  PROCEDURE DIVISION.
  COPY-BLOB.
      EXEC SQL WHENEVER SQLERROR DO PERFORM SQL-ERROR END-EXEC.
      EXEC SQL
           CONNECT :USERID
      END-EXEC.

 * Allocate and initialize the BLOB locators:
      EXEC SQL ALLOCATE :DEST END-EXEC.
      EXEC SQL ALLOCATE :SRC END-EXEC.
      DISPLAY "Source and destination LOBs are open.".

      EXEC SQL WHENEVER NOT FOUND GOTO END-OF-BLOB END-EXEC.
      EXEC SQL
           SELECT AD_PHOTO INTO :SRC
```

```
                FROM PRINT_MEDIA PM
                WHERE PM.PRODUCT_ID = 3106 AND AD_ID = 13001 END-EXEC.
           DISPLAY "Source LOB populated.".
           EXEC SQL
                SELECT AD_PHOTO INTO :DEST
                FROM PRINT_MEDIA PM
                WHERE PM.PRODUCT_ID = 3060 AND AD_ID = 11001 FOR UPDATE
END-EXEC.
           DISPLAY "Destination LOB populated.".

      * Open the DESTination LOB read/write and SRC LOB read only
           EXEC SQL LOB OPEN :DEST READ WRITE END-EXEC.
           EXEC SQL LOB OPEN :SRC READ ONLY END-EXEC.
           DISPLAY "Source and destination LOBs are open.".

      * Copy the desired amount
           EXEC SQL
                LOB COPY :AMT FROM :SRC AT :SRC-POS
                TO :DEST AT :DEST-POS END-EXEC.
           DISPLAY "Src LOB copied to destination LOB.".

      * Execute PL/SQL to get COMPARE functionality
      * to make sure that the BLOBs are identical
           EXEC SQL EXECUTE
             BEGIN
               :RET := DBMS_LOB.COMPARE(:SRC,:DEST,:AMT,1,1); END; END-EXEC.

           IF RET = 0
      *        Logic for equal BLOBs goes here
               DISPLAY "BLOBs are equal"
           ELSE
      *        Logic for unequal BLOBs goes here
               DISPLAY "BLOBs are not equal"
           END-IF.

           EXEC SQL LOB CLOSE :DEST END-EXEC.
           EXEC SQL LOB CLOSE :SRC END-EXEC.

       END-OF-BLOB.
           EXEC SQL WHENEVER NOT FOUND CONTINUE END-EXEC.
           EXEC SQL FREE :DEST END-EXEC.
           EXEC SQL FREE :SRC END-EXEC.
           EXEC SQL ROLLBACK WORK RELEASE END-EXEC.
           STOP RUN.
```

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

C/C++（Pro*C/C++）: 他の LOB への LOB の全体または一部のコピー

```
/* Copying all or part of a LOB to another LOB */
#include <oci.h>
#include <stdio.h>
#include <sqlca.h>

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

void copyLOB_proc()
{
  OCIBlobLocator *Dest_loc, *Src_loc;
  int Amount = 5;
  int Dest_pos = 10;
  int Src_pos = 1;

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  /* Allocate the LOB locators: */
  EXEC SQL ALLOCATE :Dest_loc;
  EXEC SQL ALLOCATE :Src_loc;
  /* Select the LOBs: */
  EXEC SQL SELECT ad_photo INTO :Dest_loc
           FROM Print_media WHERE product_id = 2268 AND AD_ID = 21001 FOR UPDATE;
  EXEC SQL SELECT ad_photo INTO :Src_loc
           FROM Print_media WHERE product_id = 2056 AND ad_id = 12001;
  /* Opening the LOBs is Optional: */
  EXEC SQL LOB OPEN :Dest_loc READ WRITE;
  EXEC SQL LOB OPEN :Src_loc READ ONLY;
```

```
  /* Copies the specified Amount from the source position in the source
     LOB to the destination position in the destination LOB: */
  EXEC SQL LOB COPY :Amount
              FROM :Src_loc AT :Src_pos TO :Dest_loc AT :Dest_pos;
  /* Closing the LOBs is mandatory if they have been opened: */
  EXEC SQL LOB CLOSE :Dest_loc;
  EXEC SQL LOB CLOSE :Src_loc;
  /* Release resources held by the locators: */
  EXEC SQL FREE :Dest_loc;
  EXEC SQL FREE :Src_loc;
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  copyLOB_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

Visual Basic（OO4O）: 他の LOB への LOB の全体または一部のコピー

```
'Copying all or part of a LOB to another LOB
Dim MySession As OraSession
Dim OraDb As OraDatabase
Dim OraDyn As OraDynaset, OraAdPhoto1 As OraBlob, OraAdPhotoClone As OraBlob

Set MySession = CreateObject("OracleInProcServer.XOraSession")
Set OraDb = MySession.OpenDatabase("exampledb", "samp/samp", 0&)
Set OraDyn = OraDb.CreateDynaset(
    "SELECT * FROM Print_media ORDER BY product_id, ad_id", ORADYN_DEFAULT)
Set OraAdPhoto1 = OraDyn.Fields("ad_photo").Value

Set OraAdPhotoClone = OraAdPhoto1.Clone

'Go to next row and copy LOB

OraDyn.MoveNext

OraDyn.Edit
OraAdPhoto1.Copy OraAdPhotoClone, OraAdPhotoClone.Size, 1, 1
OraDyn.Update
```

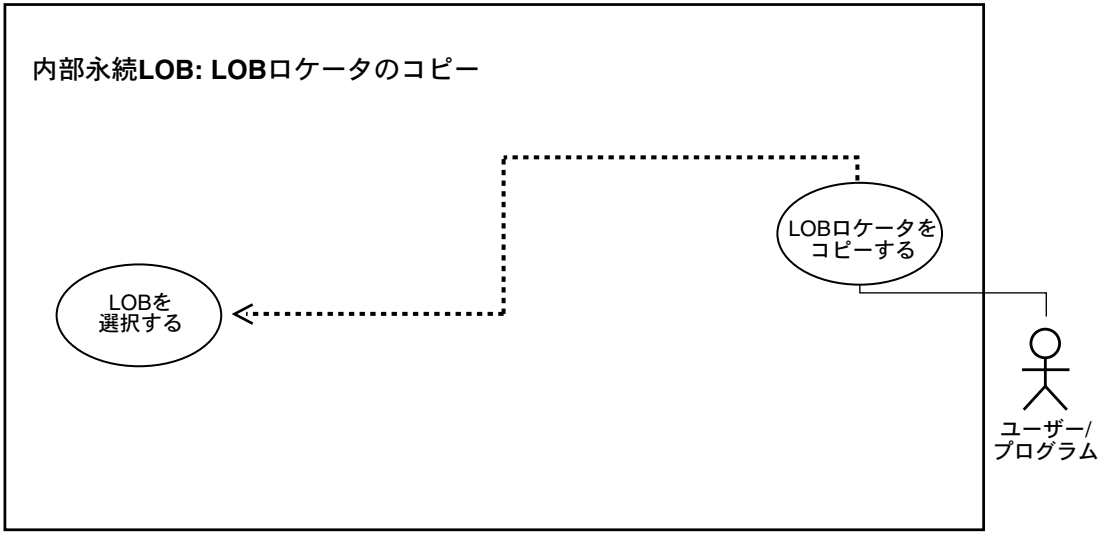
Java（JDBC）：他の LOB への LOB の全体または一部のコピー

```
// Copying all or part of a LOB to another LOB
 import java.io.OutputStream;

// Core JDBC classes:
import java.sql.DriverManager;
import java.sql.Connection;
import java.sql.Types;
import java.sql.Statement;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
// Oracle Specific JDBC classes:
import oracle.sql.*;
import oracle.jdbc.driver.*;

public class Ex2_100
{

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

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

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

    // Create a Statement:
    Statement stmt = conn.createStatement ();
    try
    {
     final int AMOUNT_TO_COPY = 2000;
     ResultSet rset = null;
     BLOB dest_loc = null;
     BLOB src_loc = null;
     InputStream in = null;
     OutputStream out = null;
     byte[] buf = new byte[AMOUNT_TO_COPY];
     rset = stmt.executeQuery (
          "SELECT ad_photo FROM Print_media
           WHERE product_id = 3060 AND ad_ad = 11001");
```

```
   if (rset.next())
   {
     src_loc = ((OracleResultSet)rset).getBLOB (1);
   }
   in = src_loc.getBinaryStream();

       rset = stmt.executeQuery (
          "SELECT ad_photo FROM Print_media
           WHERE product_id = 3106 AND ad_id = 13001 FOR UPDATE");
   if (rset.next())
   {
     dest_loc = ((OracleResultSet)rset).getBLOB (1);
   }
   out = dest_loc.getBinaryOutputStream();


   // read AMOUNT_TO_COPY bytes into buf from stream, starting from offset 0:
   in.read(buf, 0, AMOUNT_TO_COPY);

   // write AMOUNT_TO_COPY bytes from buf into output stream, starting at offset
0:
   out.write(buf, 0, AMOUNT_TO_COPY);

   // Close all streams and handles
   in.close();
   out.flush();
   out.close();
   stmt.close();
   conn.commit();
   conn.close();

    }
    catch (SQLException e)
    {
       e.printStackTrace();
    }
  }
}
```

LOB ロケータのコピー

図 10-22 利用図 : LOB ロケータのコピー



参照： 10-2 ページの表 10-1「利用モデル : 内部永続 LOB」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関数またはファンクションのリストは、第 3 章「様々なプログラム環境での LOB のサポート」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）: 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、第 5 章「ラージ・オブジェクト（LOB）: 詳細事項」の「読込み一貫性のあるロケータ」
- C（OCI）:『Oracle Call Interface プログラマーズ・ガイド』の第 16 章「その他の OCI リレーショナル関数」の「LOB 関数」の OCILobAssign() および OCILobIsEqual()

- C++（OCCI）：『Oracle C++ Call Interface プログラマーズ・ガイド』
- COBOL（Pro*COBOL）：『Pro*COBOL Precompiler プログラマーズ・ガイド』の LOB に関する詳細、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・ディレクティブ」の「ALLOCATE（実行可能埋込み SQL 拡張機能）」と「LOB ASSIGN（実行可能埋込み SQL 拡張機能）」
- C/C++（Pro*C/C++）：『Pro*C/C++ Precompiler プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびディレクティブ」の「ALLOCATE（実行可能埋込み SQL 拡張機能）」および「LOB ASSIGN（実行可能埋込み SQL 拡張機能）」
- Visual Basic（OO4O オンライン・ヘルプ）：ヘルプの「目次」タブから、「OO4O オートメーション・サーバー・リファレンス」>「OO4O サーバーのオブジェクト」>「OraBLOB、OraCLOB」>「メソッド」>「copy」を選択
- Java（JDBC）：『Oracle9*i* JDBC 開発者ガイドおよびリファレンス』の第 8 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『Oracle9*i* 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「JDBC 2.0 LOB 型と Oracle 拡張型のサポート」の「BLOB、CLOB および BFILE のサポート」

使用例

次の例では、ロケータを、イメージ（`ad_composite`）を含む他のロケータ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での例を示します。

- PL/SQL（DBMS_LOB）: LOB ロケータのコピー（10-158 ページ）
- C（OCI）: LOB ロケータのコピー（10-158 ページ）
- C++（OCCI）: 今回のリリースでは例は提供されません。
- COBOL（Pro*COBOL）: LOB ロケータのコピー（10-160 ページ）
- C/C++（Pro*C/C++）: LOB ロケータのコピー（10-161 ページ）
- Visual Basic（OO4O）: LOB ロケータのコピー（10-162 ページ）
- Java（JDBC）: LOB ロケータのコピー（10-162 ページ）

PL/SQL（DBMS_LOB）：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。この項目についての詳しい説明があります。詳細は、5-2 ページの「読込み一貫性のあるロケータ」を参照してください。

```
/* Copying a LOB locator.
   Example procedure lobAssign_proc is not part of DBMS_LOB package. */
CREATE OR REPLACE PROCEDURE lobAssign_proc IS
  Lob_loc1    blob;
  Lob_loc2    blob;
BEGIN
  SELECT ad_composite INTO Lob_loc1 FROM Print_media
     WHERE product_id = 3106 AND ad_id = 13001 FOR UPDATE;
   /* Assign Lob_loc1 to Lob_loc2 thereby saving a copy of the value of the lob
     at this point in time. */
  Lob_loc2 := Lob_loc1;
  /* When you write some data to the lob through Lob_loc1, Lob_loc2 will not see
     the newly written data whereas Lob_loc1 will see the new data. */
END;
```

C（OCI）：LOB ロケータのコピー

```
/* Copying a LOB locator */
/* Select the locator */
sb4 select_lock_adcomp_locator(Lob_loc, errhp, svchp, stmthp)
OCILobLocator *Lob_loc;
OCIError      *errhp;
OCISvcCtx     *svchp;
OCIStmt       *stmthp;
{
  text  *sqlstmt =
    (text *)"SELECT ad_composite FROM Print_media
             WHERE product_id=3106 AND ad_id = 13001 FOR UPDATE";
  OCIDefine *defnp1;
  checkerr (errhp, OCIStmtPrepare(stmthp, errhp, sqlstmt,
                                  (ub4)strlen((char *)sqlstmt),
                                  (ub4) OCI_NTV_SYNTAX, (ub4) OCI_DEFAULT));
  checkerr (errhp, OCIDefineByPos(stmthp, &defnp1, errhp, (ub4) 1,
                                  (dvoid *)&Lob_loc, (sb4)0,
                                  (ub2) SQLT_BLOB,(dvoid *) 0,
                                  (ub2 *) 0, (ub2 *) 0, (ub4) OCI_DEFAULT));
```

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

void assignLob(envhp, errhp, svchp, stmthp)
OCIEnv    *envhp;
OCIError  *errhp;
OCISvcCtx *svchp;
OCIStmt   *stmthp;
{
  OCILobLocator *dest_loc, *src_loc;
  boolean       isEqual;

  /* Allocate the LOB locators */
  (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &dest_loc,
                            (ub4)OCI_DTYPE_LOB, (size_t) 0, (dvoid **) 0);
  (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &src_loc,
                            (ub4)OCI_DTYPE_LOB, (size_t) 0, (dvoid **) 0);

  /* Select the LOBs */
  printf (" select and lock a frame locator\n");
  select_lock_adcomp_locator(src_loc, errhp, svchp, stmthp);/* source locator */

  /* Assign src_loc to dest_loc thereby saving a copy of the value of the LOB
     at this point in time.
   */
  printf(" assign the src locator to dest locator\n");
  checkerr (errhp, OCILobAssign(envhp, errhp, src_loc, &dest_loc));

  /* When you write some data to the LOB through Lob_loc1, Lob_loc2 will not
     see the newly written data whereas Lob_loc1 will see the new data.
   */

  /* Call OCI to see if the two locators are Equal */

  printf (" check if Lobs are Equal.\n");
  checkerr (errhp, OCILobIsEqual(envhp, src_loc, dest_loc, &isEqual));
  if (isEqual)
  {
    /* ... The LOB locators are Equal */
    printf(" Lob Locators are equal.\n");
  }
  else
  {
```

```
    /* ... The LOB locators are not Equal */
    printf(" Lob Locators are NOT Equal.\n");
  }

  /* Note that in this example, the LOB locators will be Equal */

  /* Free resources held by the locators*/
  (void) OCIDescriptorFree((dvoid *) dest_loc, (ub4) OCI_DTYPE_LOB);
  (void) OCIDescriptorFree((dvoid *) src_loc, (ub4) OCI_DTYPE_LOB);
  return;
}
```

COBOL（Pro*COBOL）: LOB ロケータのコピー

```
      * COPYING A LOB LOCATOR
        IDENTIFICATION DIVISION.
        PROGRAM-ID. COPY-LOCATOR.
        ENVIRONMENT DIVISION.
        DATA DIVISION.
        WORKING-STORAGE SECTION.

        01  USERID   PIC X(11) VALUES "SAMP/SAMP".
        01  DEST          SQL-BLOB.
        01  SRC           SQL-BLOB.
            EXEC SQL INCLUDE SQLCA END-EXEC.
        PROCEDURE DIVISION.
        COPY-BLOB-LOCATOR.
            EXEC SQL WHENEVER SQLERROR DO PERFORM SQL-ERROR END-EXEC.
            EXEC SQL
                 CONNECT :USERID
            END-EXEC.
       * Allocate and initialize the BLOB locators:
            EXEC SQL ALLOCATE :DEST END-EXEC.
            EXEC SQL ALLOCATE :SRC END-EXEC.
            EXEC SQL WHENEVER NOT FOUND GOTO END-OF-BLOB END-EXEC.
            EXEC SQL
                 SELECT AD_COMPOSITE INTO :SRC
                 FROM PRINT_MEDIA
                 WHERE PRODUCT_ID = 2268 AND AD_ID = 21001 FOR UPDATE
            END-EXEC.
             EXEC SQL LOB ASSIGN :SRC TO :DEST END-EXEC.

       * When you write data to the LOB through SRC, DEST will
       * not see the newly written data

        END-OF-BLOB.
```

```
            EXEC SQL WHENEVER NOT FOUND CONTINUE END-EXEC.
            EXEC SQL FREE :DEST END-EXEC.
            EXEC SQL FREE :SRC END-EXEC.
            EXEC SQL ROLLBACK WORK RELEASE END-EXEC.
            STOP RUN.

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

C/C++（Pro*C/C++）：LOB ロケータのコピー

```
/* Copying a LOB locator */
#include <oci.h>
#include <stdio.h>
#include <sqlca.h>

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

void lobAssign_proc()
{
  OCIBlobLocator *Lob_loc1, *Lob_loc2;

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  EXEC SQL ALLOCATE :Lob_loc1;
  EXEC SQL ALLOCATE :Lob_loc2;
  EXEC SQL SELECT ad_composite INTO :Lob_loc1
           FROM Print_media
           WHERE product_id = 3060 AND ad_id = 11001 FOR UPDATE;
  /* Assign Lob_loc1 to Lob_loc2 thereby saving a copy of the value of the
     LOB at this point in time: */
  EXEC SQL LOB ASSIGN :Lob_loc1 TO :Lob_loc2;
  /* When you write some data to the LOB through Lob_loc1, Lob_loc2 will not
     see the newly written data whereas Lob_loc1 will see the new data: */
}
```

```
void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  lobAssign_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

Visual Basic（OO4O）: LOB ロケータのコピー

```
'Copying a LOB locator
Dim MySession As OraSession
Dim OraDb As OraDatabase
Dim OraDyn As OraDynaset, OraAdPhoto1 As OraBlob, OraAdPhotoClone As OraBlob

Set MySession = CreateObject("OracleInProcServer.XOraSession")
Set OraDb = MySession.OpenDatabase("exampledb", "samp/samp", 0&)
Set OraDyn = OraDb.CreateDynaset(
   "SELECT * FROM Print_media ORDER BY product_id, ad_id ", ORADYN_DEFAULT)
Set OraAdPhoto1 = OraDyn.Fields("ad_photo").Value
Set OraAdPhotoClone = OraAdPhoto1.Clone

OraDyn.MoveNext

'Copy 1000 bytes of LOB values OraAdPhotoClone to OraAdPhotol at OraAdPhotol
'offset 100:
OraDyn.Edit
OraAdPhoto1.Copy OraAdPhotoClone, 1000, 100
OraDyn.Update
```

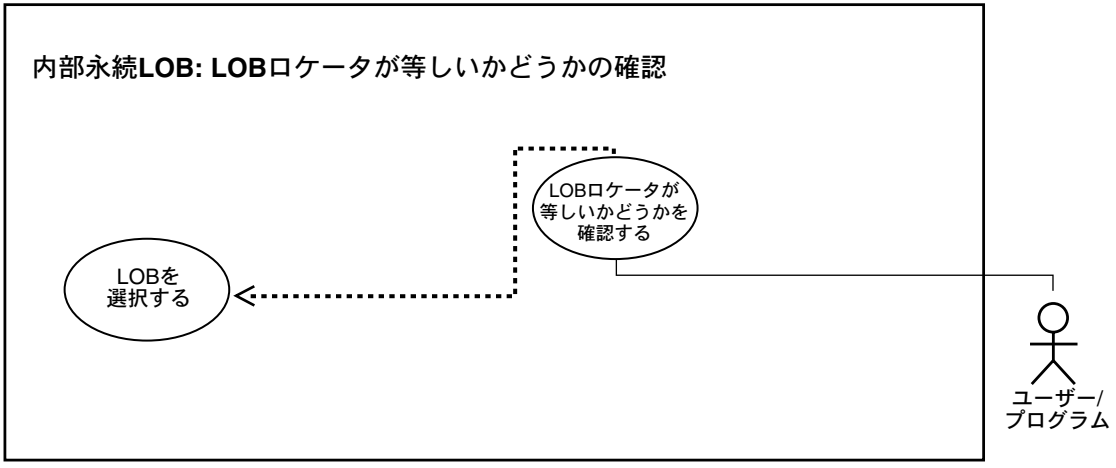
Java（JDBC）: LOB ロケータのコピー

```
// Copying a LOB locator
import java.sql.Connection;
import java.sql.Types;

import java.sql.Statement;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

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

```
import oracle.jdbc.driver.*;

public class Ex2_104
{
public static void main (String args [])
       throws Exception
  {
    // Load the Oracle JDBC driver:
    DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

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

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

    // Create a Statement:
    Statement stmt = conn.createStatement ();
  try
    {
     BLOB lob_loc1 = null;
     BLOB lob_loc2 = null;
     ResultSet rset = stmt.executeQuery (
         "SELECT ad_composite FROM Print_media
                 WHERE product_id = 2056 AND ad_id = 12001");
   if (rset.next())
   {
     lob_loc1 = ((OracleResultSet)rset).getBLOB (1);
   }

   // When you write data to LOB through lob_loc1,lob_loc2 will not see changes
   lob_loc2 = lob_loc1;
   stmt.close();
   conn.commit();
   conn.close();

    }
    catch (SQLException e)
    {
       e.printStackTrace();
    }
  }
}
```

2 つの LOB ロケータが等しいかどうかの確認

図 10-23 利用図 : 2 つの LOB ロケータが等しいかどうかの確認



参照 : 10-2 ページの表 10-1「利用モデル : 内部永続 LOB」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関数またはファンクションのリストは、第 3 章「様々なプログラム環境での LOB のサポート」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）:『Oracle Call Interface プログラマーズ・ガイド』の第 16 章「その他の OCI リレーショナル関数」の「LOB 関数」の OCILobAssign() および OCILobIsEqual()
- C++（OCCI）:『Oracle C++ Call Interface プログラマーズ・ガイド』

- COBOL（Pro*COBOL）: 参照マニュアルはありません。
- C/C++（Pro*C/C++）:『Pro*C/C++ Precompiler プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびディレクティブ」の「LOB ASSIGN（実行可能埋込み SQL 拡張機能）」
- Visual Basic（OO4O）: 参照マニュアルはありません。
- Java（JDBC）:『Oracle9*i* JDBC 開発者ガイドおよびリファレンス』の第 8 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『Oracle9*i* 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「JDBC 2.0 LOB 型と Oracle 拡張型のサポート」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「読込み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での例を示します。

- PL/SQL（DBMS_LOB）: 今回のリリースでは例は提供されません。
- C（OCI）: 2 つの LOB ロケータが等しいかどうかの確認（10-165 ページ）
- C++（OCCI）: 今回のリリースでは例は提供されません。
- COBOL（Pro*COBOL）: 今回のリリースでは例は提供されません。
- C/C++（Pro*C/C++）: 2 つの LOB ロケータが等しいかどうかの確認（10-167 ページ）
- Visual Basic（OO4O）: 今回のリリースでは例は提供されません。
- Java（JDBC）: 2 つの LOB ロケータが等しいかどうかの確認（10-168 ページ）

C（OCI）: 2 つの LOB ロケータが等しいかどうかの確認

```
/* Seeing if One LOB locator is Equal to Another */
/* Select the locator: */
sb4 select_lock_adcomp_locator(Lob_loc, errhp, svchp, stmthp)
OCILobLocator *Lob_loc;
OCIError      *errhp;
OCISvcCtx     *svchp;
OCIStmt       *stmthp;
{
  text  *sqlstmt =
```

```
    (text *)"SELECT ad_composite FROM Print_media
             WHERE product_id=2268 AND ad_id = 21001 FOR UPDATE";
  OCIDefine *defnp1;
  checkerr (errhp, OCIStmtPrepare(stmthp, errhp, sqlstmt,
                                  (ub4)strlen((char *)sqlstmt),
                                  (ub4) OCI_NTV_SYNTAX, (ub4) OCI_DEFAULT));
  checkerr (errhp, OCIDefineByPos(stmthp, &defnp1, errhp, (ub4) 1,
                                  (dvoid *)&Lob_loc, (sb4)0,
                                  (ub2) SQLT_BLOB,(dvoid *) 0,
                                  (ub2 *) 0, (ub2 *) 0, (ub4) OCI_DEFAULT));

  /* Execute the select and fetch one row: */
  checkerr(errhp, OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                                 (CONST OCISnapshot*) 0, (OCISnapshot*) 0,
                                 (ub4) OCI_DEFAULT));
  return (0);
}
void locatorIsEqual(envhp, errhp, svchp, stmthp)
OCIEnv    *envhp;
OCIError  *errhp;
OCISvcCtx *svchp;
OCIStmt   *stmthp;
{
  OCILobLocator *dest_loc, *src_loc;
  boolean       isEqual;

  /* Allocate the LOB locators: */
  (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &dest_loc,
                            (ub4)OCI_DTYPE_LOB, (size_t) 0, (dvoid **) 0);
  (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &src_loc,
                            (ub4)OCI_DTYPE_LOB, (size_t) 0, (dvoid **) 0);

  /* Select the LOBs: */
  printf (" select and lock an ad_composite locator\n");
  select_lock_adcomp_locator(src_loc, errhp, svchp, stmthp);/* source locator */

  /* Assign src_loc to dest_loc thereby saving a copy of the value of the LOB
     at this point in time: */
  printf(" assign the src locator to dest locator\n");
  checkerr (errhp, OCILobAssign(envhp, errhp, src_loc, &dest_loc));

  /* When you write some data to the LOB through Lob_loc1, Lob_loc2 will not
     see the newly written data whereas Lob_loc1 will see the new data: */

  /* Call OCI to see if the two locators are Equal: */

  printf (" check if Lobs are Equal.\n");
```

```
  checkerr (errhp, OCILobIsEqual(envhp, src_loc, dest_loc, &isEqual));

  if (isEqual)
  {
    /* ... The LOB locators are Equal: */
    printf(" Lob Locators are equal.\n");
  }
  else
  {
    /* ... The LOB locators are not Equal: */
    printf(" Lob Locators are NOT Equal.\n");
  }

  /* Note that in this example, the LOB locators will be Equal */

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

  return;
}
```

C/C++（Pro*C/C++）: 2 つの LOB ロケータが等しいかどうかの確認

```
/* Seeing if One LOB locator is equal to another */
/* Pro*C/C++ does not provide a mechanism to test the equality of two
   locators. But you can use OCI directly. Two locators can be
   compared to determine whether or not they are equal as this example
   demonstrates: */

#include <sql2oci.h>
#include <stdio.h>
#include <sqlca.h>

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

void LobLocatorIsEqual_proc()
{
  OCIBlobLocator *Lob_loc1, *Lob_loc2;
  OCIEnv *oeh;
```

```
  boolean isEqual;
  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  EXEC SQL ALLOCATE :Lob_loc1;
  EXEC SQL ALLOCATE :Lob_loc2;
  EXEC SQL SELECT ad_composite INTO Lob_loc1
           FROM Print_media
           where product_id = 3060 AND ad_id = 11001 FOR UPDATE;
  /* Assign Lob_loc1 to Lob_loc2 thereby saving a copy of the value of the
     LOB at this point in time: */
  EXEC SQL LOB ASSIGN :Lob_loc1 TO :Lob_loc2;
  /* When you write some data to the lob through Lob_loc1, Lob_loc2 will
     not see the newly written data whereas Lob_loc1 will see the new
     data. */
  /* Get the OCI Environment Handle using a SQLLIB Routine: */
  (void) SQLEnvGet(SQL_SINGLE_RCTX, &oeh);
  /* Call OCI to see if the two locators are Equal: */
  (void) OCILobIsEqual(oeh, Lob_loc1, Lob_loc2, &isEqual);
  if (isEqual)
    printf("The locators are equal\n");
  else
    printf("The locators are not equal\n");
  /* Note that in this example, the LOB locators will be Equal */
  EXEC SQL FREE :Lob_loc1;
  EXEC SQL FREE :Lob_loc2;
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  LobLocatorIsEqual_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

Java（JDBC）: 2 つの LOB ロケータが等しいかどうかの確認

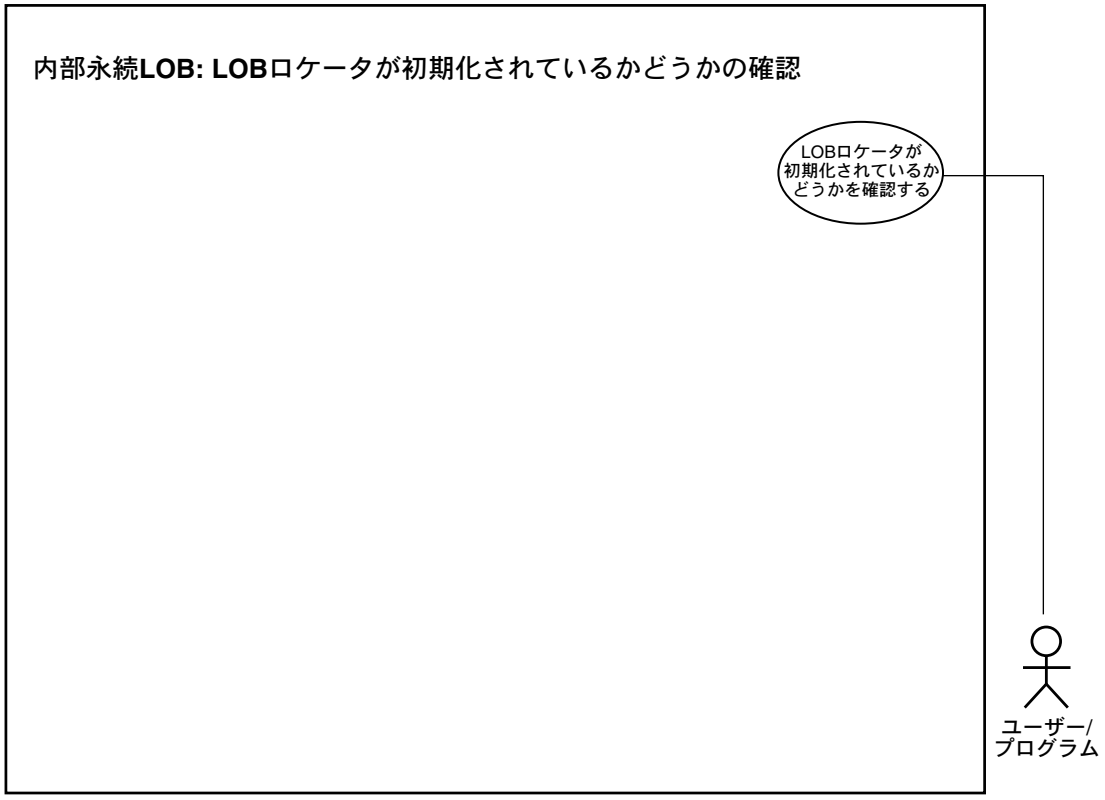
```
// Seeing if one LOB locator is equal to another
import java.sql.Connection;
import java.sql.Types;
import java.sql.Statement;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

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

```
import oracle.jdbc.driver.*;

public class Ex2_108
{
 public static void main (String args [])
       throws Exception
  {
    // Load the Oracle JDBC driver:
    DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

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

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

    // Create a Statement:
    Statement stmt = conn.createStatement ();
    try
    {
      BLOB lob_loc1 = null;
      BLOB lob_loc2 = null;
      ResultSet rset = stmt.executeQuery (
          "SELECT ad_photo FROM Print_media
            WHERE product_id = 3106 AND ad_id = 13001");
   if (rset.next())
   {
     lob_loc1 = ((OracleResultSet)rset).getBLOB (1);
   }

   // When you write data to LOB through lob_loc1,lob_loc2 will not see the
changes:
   lob_loc2 = lob_loc1;

   // Note that in this example, the Locators will be equal.
   if (lob_loc1.equals(lob_loc2))
   {
      // The Locators are equal:
      System.out.println("Locators are equal");
   }
   else
   {
      // The Locators are different:
      System.out.println("Locators are NOT equal");
   }
```

```
    stmt.close();
    conn.commit();
    conn.close();

     }
     catch (SQLException e)
     {
        e.printStackTrace();
     }
   }
}
```

ロケータの初期化 : LOB ロケータが初期化されているかどうかの確認

図 10-24 利用図 : LOB ロケータが初期化されているかどうかの確認



参照： 10-2 ページの表 10-1「利用モデル : 内部永続 LOB」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関数またはファンクションのリストは、第 3 章「様々なプログラム環境での LOB のサポート」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）:『Oracle Call Interface プログラマーズ・ガイド』の第 16 章「その他の OCI リレーショナル関数」の「LOB 関数」の OCILobLocatorIsInit()
- C++（OCCI）:『Oracle C++ Call Interface プログラマーズ・ガイド』
- COBOL（Pro*COBOL）: 参照マニュアルはありません。
- C/C++（Pro*C/C++）:『Pro*C/C++ Precompiler プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびディレクティブ」および『Oracle Call Interface プログラマーズ・ガイド』の第 16 章「その他の 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での例を示します。

- PL/SQL（DBMS_LOB）: 今回のリリースでは例は提供されません。
- C（OCI）: LOB ロケータが初期化されているかどうかの確認（10-173 ページ）
- C++（OCCI）: 今回のリリースでは例は提供されません。
- COBOL（Pro*COBOL）: 今回のリリースでは例は提供されません。

- 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（10-174 ページ）
- Visual Basic（OO4O）: 今回のリリースでは例は提供されません。
- Java（JDBC）: 今回のリリースでは例は提供されません。

C（OCI）: LOB ロケータが初期化されているかどうかの確認

```
/* Seeing if a LOB locator is initialized */
/* Select the locator: */
sb4 select_adcomp_locator(Lob_loc, errhp, svchp, stmthp)
OCILobLocator *Lob_loc;
OCIError      *errhp;
OCISvcCtx     *svchp;
OCIStmt       *stmthp;
{
  text      *sqlstmt =
         (text *)"SELECT ad_composite FROM Print_media
                  WHERE product_id=2268 AND ad_id = 21001";
  OCIDefine *defnp1;
  checkerr (errhp, OCIStmtPrepare(stmthp, errhp, sqlstmt,
                                  (ub4)strlen((char *)sqlstmt),
                                  (ub4) OCI_NTV_SYNTAX, (ub4) OCI_DEFAULT));
  checkerr (errhp, OCIDefineByPos(stmthp, &defnp1, errhp, (ub4) 1,
                                  (dvoid *)&Lob_loc, (sb4)0,
                                  (ub2) SQLT_BLOB,(dvoid *) 0,
                                  (ub2 *) 0, (ub2 *) 0, (ub4) OCI_DEFAULT));

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

void isInitializedLob(envhp, errhp, svchp, stmthp)
OCIEnv    *envhp;
OCIError  *errhp;
OCISvcCtx *svchp;
OCIStmt   *stmthp;
{
  OCILobLocator *Lob_loc1, *Lob_loc2;
  boolean       isInitialized;

  /* Allocate the LOB locators: */
  printf(" allocate locator 1 and 2\n");
```

```
  (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &Lob_loc1,
                          (ub4)OCI_DTYPE_LOB, (size_t) 0, (dvoid **) 0);
  (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &Lob_loc2,
                            (ub4)OCI_DTYPE_LOB, (size_t) 0, (dvoid **) 0);

  /* Select the LOBs: */
  printf (" select an ad_composite locator into locator 1\n");
  select_adcomp_locator(Lob_loc1, errhp, svchp, stmthp);         /* locator 1 */

  /* Determine if the locator 1 is Initialized -: */
  checkerr(errhp, OCILobLocatorIsInit(envhp, errhp, Lob_loc1, &isInitialized));
                                    /* IsInitialized should return TRUE here */
  printf(" for Locator 1, isInitialized = %d\n", isInitialized);

  /* Determine if the locator 2 is Initialized -: */
  checkerr(errhp, OCILobLocatorIsInit(envhp, errhp, Lob_loc2, &isInitialized));
                                    /* IsInitialized should return FALSE here */
  printf(" for Locator 2, isInitialized = %d\n", isInitialized);

  /* Free resources held by the locators: */
  (void) OCIDescriptorFree((dvoid *) Lob_loc1, (ub4) OCI_DTYPE_LOB);
  (void) OCIDescriptorFree((dvoid *) Lob_loc2, (ub4) OCI_DTYPE_LOB);
  return;
  }
```

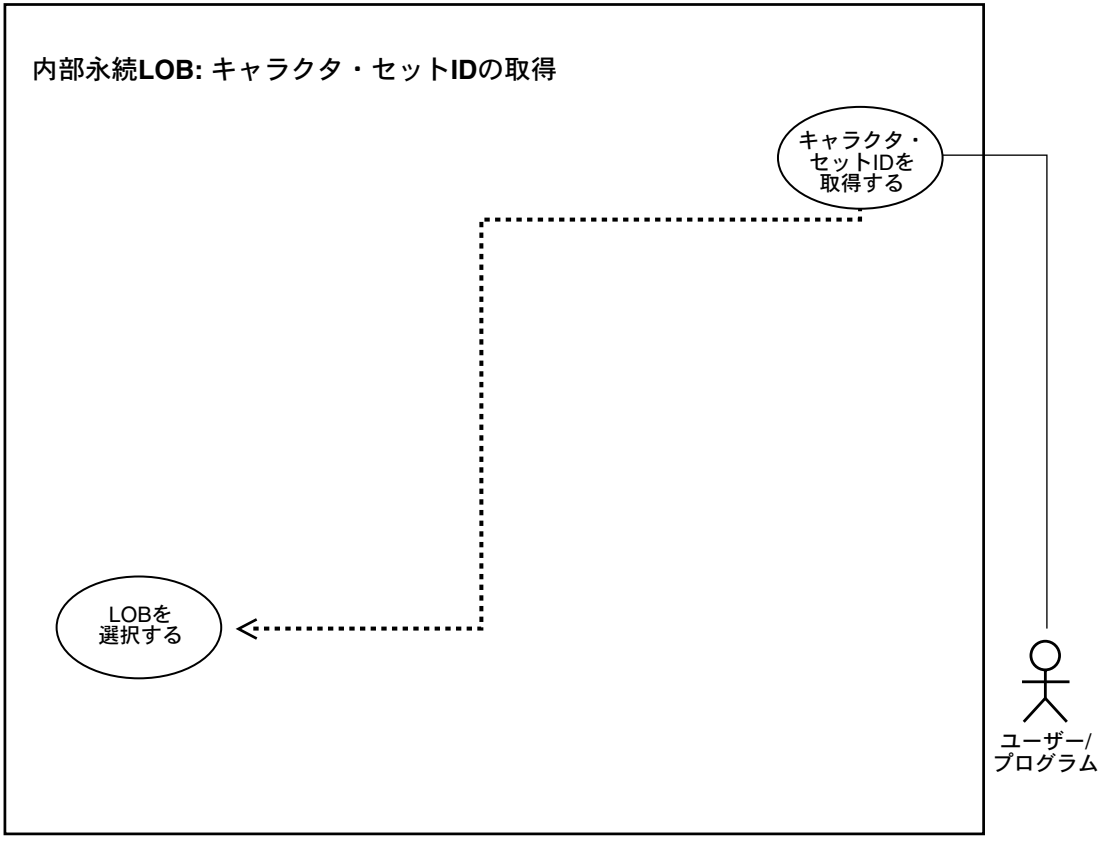
C/C++（Pro*C/C++）: LOB ロケータが初期化されているかどうかの確認

```
/* Seeing if a LOB locator is initialized */
/* Pro*C/C++ has no form of embedded SQL statement to determine if a LOB
   locator is initialized. Locators in Pro*C/C++ are initialized when they
   are allocated via the EXEC SQL ALLOCATE statement. An example
   can be written that uses embedded SQL and the OCI as is shown here: */

#include <sql2oci.h>
#include <stdio.h>
#include <sqlca.h>

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

void LobLocatorIsInit_proc()
```

```
{
  OCIBlobLocator *Lob_loc;
  OCIEnv *oeh;
  OCIError *err;
  boolean isInitialized;

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  EXEC SQL ALLOCATE :Lob_loc;
  EXEC SQL SELECT ad_composite INTO Lob_loc
           FROM Print_media where product_id = 2056 AND ad_id =  12001;
  /* Get the OCI Environment Handle using a SQLLIB Routine: */
  (void) SQLEnvGet(SQL_SINGLE_RCTX, &oeh);
  /* Allocate the OCI Error Handle: */
  (void) OCIHandleAlloc((dvoid *)oeh, (dvoid **)&err,
                        (ub4)OCI_HTYPE_ERROR, (ub4)0, (dvoid **)0);
  /* Use the OCI to determine if the locator is Initialized: */
  (void) OCILobLocatorIsInit(oeh, err, Lob_loc, &isInitialized);
  if (isInitialized)
    printf("The locator is initialized\n");
  else
    printf("The locator is not initialized\n");
  /* Note that in this example, the locator is initialized */
  /* Deallocate the OCI Error Handle: */
  (void) OCIHandleFree(err, OCI_HTYPE_ERROR);
  /* Release resources held by the locator: */
  EXEC SQL FREE :Lob_loc;
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  LobLocatorIsInit_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

キャラクタ・セット ID: キャラクタ・セット ID の取得

図 10-25 利用図 : キャラクタ・セット ID の取得



参照 : 10-2 ページの表 10-1「利用モデル : 内部永続 LOB」を参照してください。

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関数またはファンクションのリストは、第 3 章「様々なプログラム環境での LOB のサポート」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）:『Oracle Call Interface プログラマーズ・ガイド』の第 16 章「その他の OCI リレーショナル関数」の「LOB 関数」の OCILobCharSetId()
- C++（OCCI）:『Oracle C++ Call Interface プログラマーズ・ガイド』
- 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、外国語のテキスト（`ad_fltextn`）のキャラクタ・セット ID を取得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）: 今回のリリースでは例は提供されません。
- C（OCI）: キャラクタ・セット ID の取得（10-177 ページ）
- C++（OCCI）: 今回のリリースでは例は提供されません。
- COBOL（Pro*COBOL）: 今回のリリースでは例は提供されません。
- C/C++（Pro*C/C++）: 今回のリリースでは例は提供されません。
- Visual Basic（OO4O）: 今回のリリースでは例は提供されません。
- Java（JDBC）: 今回のリリースでは例は提供されません。

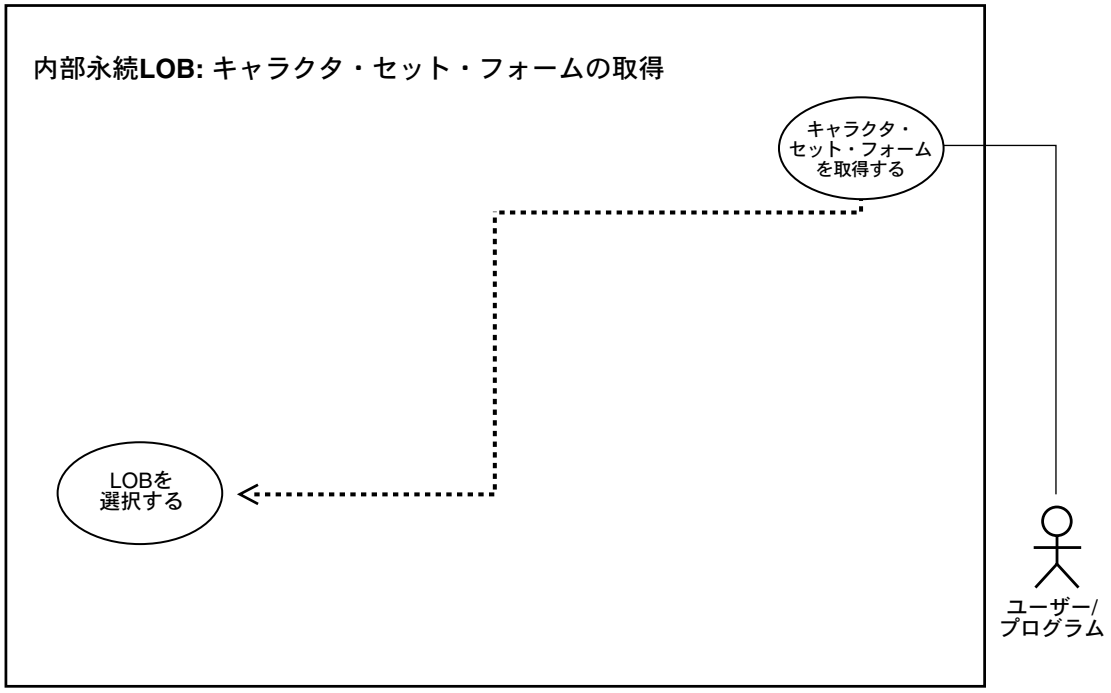
C（OCI）: キャラクタ・セット ID の取得

```
/* Getting character set id */
/* This function takes a valid LOB locator and prints the character set id of
the LOB. */
/* Select the locator */
sb4 select_adfltextn_locator(Lob_loc, errhp, svchp, stmthp)
OCILobLocator *Lob_loc;
OCIError      *errhp;
```

```
OCISvcCtx     *svchp;
OCIStmt       *stmthp;
{
  OCIDefine *defnp1;
  text  *sqlstmt =
    (text *)"SELECT ad_fltextn FROM Print_media
             WHERE product_id = 2268 AND ad_id = 21001";
  checkerr (errhp, OCIStmtPrepare(stmthp, errhp, sqlstmt,
                                  (ub4)strlen((char *)sqlstmt),
                                  (ub4)OCI_NTV_SYNTAX, (ub4)OCI_DEFAULT));
  /* Define the column being selected */
  checkerr (errhp, OCIDefineByPos(stmthp, &defnp1, errhp, (ub4) 1,
                                  (dvoid *)&Lob_loc, (sb4)0,
                                  (ub2)SQLT_CLOB,(dvoid *)0, (ub2 *)0,
                                  (ub2 *)0, (ub4)OCI_DEFAULT));
 /* Execute and fetch one row */
  checkerr (errhp, OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                                  (CONST OCISnapshot*) 0, (OCISnapshot*) 0,
                                  (ub4) OCI_DEFAULT));
  return 0;
}
sb4 getcsidLob (envhp, errhp, svchp, stmthp)
OCIEnv    *envhp;
OCIError  *errhp;
OCISvcCtx *svchp;
OCIStmt   *stmthp;
{
  OCILobLocator *Lob_loc;
  ub2 charsetid =0 ;
  (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &Lob_loc,
                            (ub4)OCI_DTYPE_LOB, (size_t) 0, (dvoid **) 0);
  printf (" select a ad_fltextn locator\n");
  select_adfltextn_locator(Lob_loc, errhp, svchp, stmthp);
  printf (" get the character set id of adfltextn_locator\n");

  /* Get the charactersid ID of the LOB*/
  checkerr (errhp, OCILobCharSetId(envhp, errhp, Lob_loc, &charsetid));
  printf(" character Set ID of ad_fltextn is : %d\n", charsetid);

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

  return;
}
```

キャラクタ・セットフォーム : キャラクタ・セット・フォームの取得

図 10-26 利用図 : キャラクタ・セット・フォームの取得



参照 : 10-2 ページの表 10-1「利用モデル : 内部永続 LOB」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関数またはファンクションのリストは、第 3 章「様々なプログラム環境での LOB のサポート」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）:『Oracle Call Interface プログラマーズ・ガイド』の第 16 章「その他の OCI リレーショナル関数」の「LOB 関数」の OCILobCharSetForm()
- C++（OCCI）:『Oracle C++ Call Interface プログラマーズ・ガイド』
- 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、外国語のテキスト（`ad_fltextn`）のキャラクタ・セット・フォームを取得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）: 今回のリリースでは例は提供されません。
- C（OCI）: キャラクタ・セット・フォームの取得（10-181 ページ）
- C++（OCCI）: 今回のリリースでは例は提供されません。
- COBOL（Pro*COBOL）: 今回のリリースでは例は提供されません。
- C/C++（Pro*C/C++）: 今回のリリースでは例は提供されません。
- Visual Basic（OO4O）: 今回のリリースでは例は提供されません。
- Java（JDBC）: 今回のリリースでは例は提供されません。

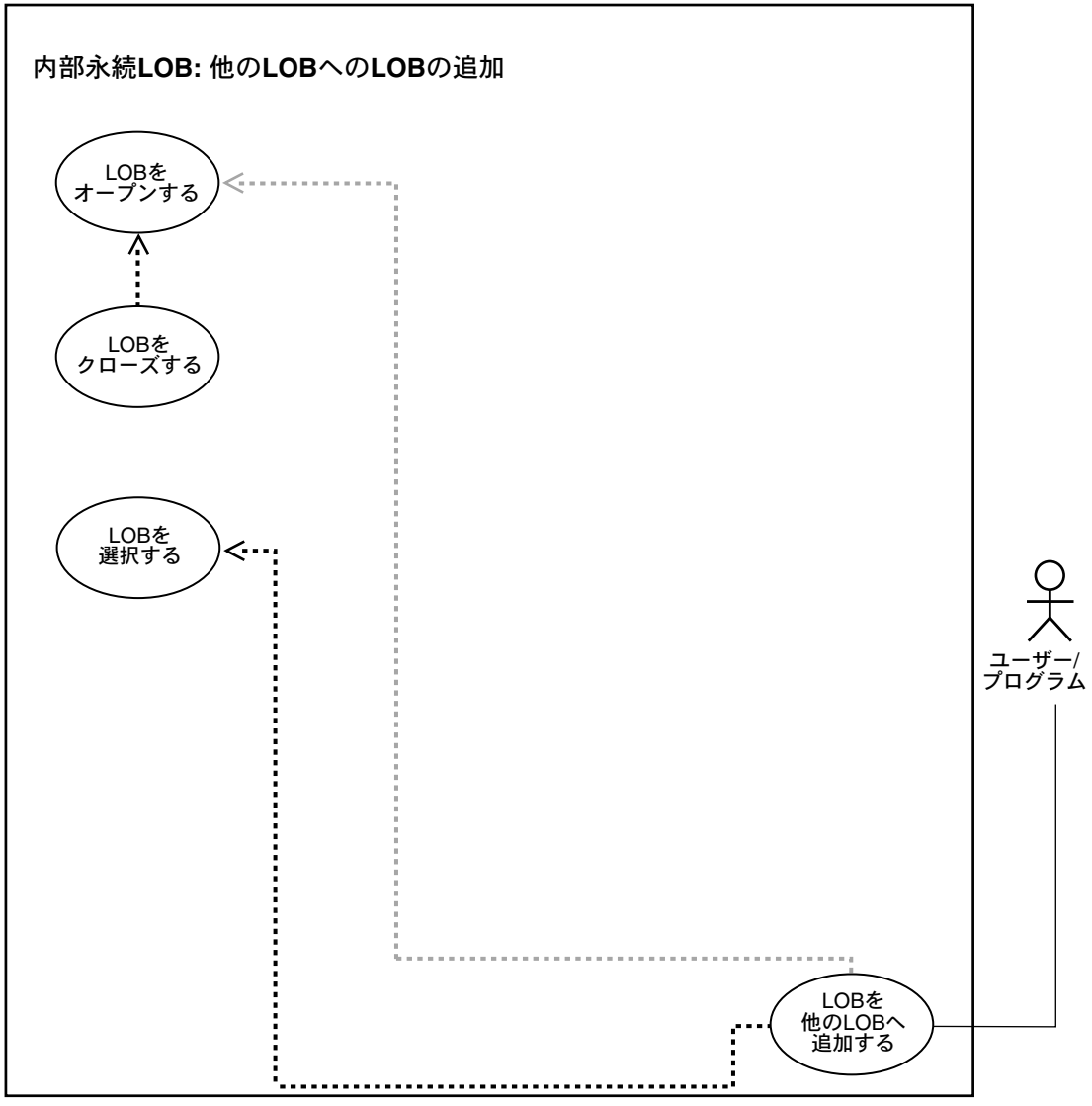
C（OCI）：キャラクタ・セット・フォームの取得

```
/* Getting character set form of the foreign language ad text, ad_fltextn */
/* Select the locator */
sb4 select_adfltextn_locator(Lob_loc, errhp, svchp, stmthp)
OCILobLocator *Lob_loc;
OCIError      *errhp;
OCISvcCtx     *svchp;
OCIStmt       *stmthp;
{
  OCIDefine *defnp1;
  text  *sqlstmt =
    (text *)"SELECT ad_fltextn FROM Print_media
             WHERE product_id = 2268 AND ad_id = 21001";
  checkerr (errhp, OCIStmtPrepare(stmthp, errhp, sqlstmt,
                                  (ub4)strlen((char *)sqlstmt),
                                  (ub4)OCI_NTV_SYNTAX, (ub4)OCI_DEFAULT));

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

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

  return 0;
}

/* This function takes a valid LOB locator and prints the character set form
   of the LOB.
 */
sb4 getcsformLob(envhp, errhp, svchp, stmthp)
OCIEnv    *envhp;
OCIError  *errhp;
OCISvcCtx *svchp;
OCIStmt   *stmthp;
{
  OCILobLocator *Lob_loc;
  ub1 charset_form = 0 ;

  (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &Lob_loc,
                            (ub4)OCI_DTYPE_LOB, (size_t) 0, (dvoid **) 0);
```

```
  printf (" select an ad_fltextn locator\n");
  select_adfltextn_locator(Lob_loc, errhp, svchp, stmthp);
  printf (" get the character set form of ad_fltextn\n");

  /* Get the charactersid form of the LOB*/
  checkerr (errhp, OCILobCharSetForm(envhp, errhp, Lob_loc, &charset_form));
  printf(" character Set Form of ad_fltextn is : %d\n", charset_form);

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

  return;
}
```

他の LOB への LOB の追加

図 10-27 利用図 : 他の LOB への LOB の追加



参照： 10-2 ページの表 10-1「利用モデル : 内部永続 LOB」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 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、5-5 ページの「更新済ロケータを介した LOB の更新」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関数またはファンクションのリストは、第 3 章「様々なプログラム環境での LOB のサポート」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）：『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』の第 23 章「DBMS_LOB」の「DBMS_LOB サブプログラムの要約」の「APPEND プロシージャ」
- C（OCI）：『Oracle Call Interface プログラマーズ・ガイド』の第 16 章「その他の OCI リレーショナル関数」の「LOB 関数」の OCILobAppend()
- C++（OCCI）：『Oracle C++ Call Interface プログラマーズ・ガイド』
- COBOL（Pro*COBOL）：『Pro*COBOL Precompiler プログラマーズ・ガイド』の LOB に関する詳細、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・ディレクティブ」の「LOB APPEND（実行可能埋込み SQL 拡張機能）」
- C/C++（Pro*C/C++）：『Pro*C/C++ Precompiler プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびディレクティブ」の「LOB APPEND（実行可能埋込み SQL 拡張機能）」
- Visual Basic（OO4O）：（Oracle Objects for OLE（OO4O）オンライン・ヘルプ）: ヘルプの「目次」タブから、「OO4O オートメーション・サーバー・リファレンス」>「OO4O サーバーのオブジェクト」>「OraBLOB、OraCLOB」>「メソッド」>「append」を選択

- Java（JDBC）：『Oracle9*i* JDBC 開発者ガイドおよびリファレンス』の第 8 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『Oracle9*i* 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「JDBC 2.0 LOB 型と Oracle 拡張型のサポート」の「BLOB、CLOB および BFILE のサポート」

使用例

この例では、イメージを他のイメージに追加します。

例

次のプログラム環境での例を示します。

- PL/SQL（DBMS_LOB）：他の LOB への LOB の追加（10-185 ページ）
- C（OCI）：他の LOB への LOB の追加（10-186 ページ）
- C++（OCCI）：今回のリリースでは例は提供されません。
- COBOL（Pro*COBOL）：他の LOB への LOB の追加（10-188 ページ）
- C/C++（Pro*C/C++）：他の LOB への LOB の追加（10-189 ページ）
- Visual Basic（OO4O）：他の LOB への LOB の追加（10-190 ページ）
- Java（JDBC）：他の LOB への LOB の追加（10-191 ページ）

PL/SQL（DBMS_LOB）：他の LOB への LOB の追加

```
/* Appending one LOB to another */
/* Note that the example procedure appendLOB_proc is not part of the
   DBMS_LOB package: */
CREATE OR REPLACE PROCEDURE appendLOB_proc IS
   Dest_loc        BLOB;
   Src_loc         BLOB;
BEGIN
   /* Select the LOB, get the destination LOB locator: */
   SELECT ad_photo INTO Dest_loc FROM Print_media
      WHERE product_id = 2268 AND ad_id = 21001 FOR UPDATE;
   /* Select the LOB, get the destination LOB locator: */
   SELECT ad_photo INTO Src_loc FROM Print_media
      WHERE product_id = 2056 AND ad_id = 12001;
   /* Opening the LOB is optional: */
   DBMS_LOB.OPEN (Dest_loc, DBMS_LOB.LOB_READWRITE);
   DBMS_LOB.OPEN (Src_loc, DBMS_LOB.LOB_READONLY);
   DBMS_LOB.APPEND(Dest_loc, Src_loc);
   /* Closing the LOB is mandatory if you have opened it: */
   DBMS_LOB.CLOSE (Dest_loc);
```

```
   DBMS_LOB.CLOSE (Src_loc);
COMMIT;

EXCEPTION
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('Operation failed');
END;
```

C（OCI）: 他の LOB への LOB の追加

```
/* Appending one LOB to another. */
/* This function appends the Source LOB to the end of the Destination LOB */
/* Select the locator */
sb4 select_lock_adphoto_locator_2(Lob_loc, dest_type, errhp, svchp, stmthp)
OCILobLocator *Lob_loc;
ub1            dest_type;                    /* whether destination locator */
OCIError      *errhp;
OCISvcCtx     *svchp;
OCIStmt       *stmthp;
{
  char        sqlstmt[150];
  OCIDefine   *defnp1;
if (dest_type == TRUE)
  {
   strcpy (sqlstmt,
  (char *)"SELECT ad_photo FROM Print_media
          WHERE product_id=2268 AND ad_id=21001 FOR UPDATE");
    printf ("  select destination ad_photo locator\n");
  }
  else
  {
   strcpy(sqlstmt, (char *)"SELECT ad_photo FROM Print_media
            WHERE product_id=3106 and ad_id=13001");
    printf ("  select source ad_photo locator\n");
  }
  checkerr (errhp, OCIStmtPrepare(stmthp, errhp, (text *)sqlstmt,
                                  (ub4)strlen((char *)sqlstmt),
                                  (ub4) OCI_NTV_SYNTAX, (ub4) OCI_DEFAULT));

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

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

```
                    (ub4) OCI_DEFAULT));

  return 0;
}
void appendLob(envhp, errhp, svchp, stmthp)
OCIEnv    *envhp;
OCIError  *errhp;
OCISvcCtx *svchp;
OCIStmt   *stmthp;
{
  OCILobLocator *Dest_loc, *Src_loc;

  /* Allocate the LOB locators */
  (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &Dest_loc,
                            (ub4)OCI_DTYPE_LOB, (size_t) 0, (dvoid **) 0);
  (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &Src_loc,
                            (ub4)OCI_DTYPE_LOB, (size_t) 0, (dvoid **) 0);

  /* Select the LOBs */
  printf(" select source and destination Lobs\n");
  select_lock_adphoto_locator_2(Dest_loc, TRUE, errhp, svchp, stmthp);
                                                      /* destination locator */
  select_lock_adphoto_locator_2(Src_loc, FALSE, errhp, svchp, stmthp);
                                                          /* source locator */
  /* Opening the LOBs is Optional */
  checkerr (errhp, OCILobOpen(svchp, errhp, Dest_loc, OCI_LOB_READWRITE));
  checkerr (errhp, OCILobOpen(svchp, errhp, Src_loc, OCI_LOB_READONLY));

  /* Append Source LOB to the end of the Destination LOB. */
  printf(" append the source Lob to the destination Lob\n");
  checkerr(errhp, OCILobAppend(svchp, errhp, Dest_loc, Src_loc));

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

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

  return;
}
```

COBOL（Pro*COBOL）: 他の LOB への LOB の追加

```
* APPENDING ONE LOB TO ANOTHER
  IDENTIFICATION DIVISION.
  PROGRAM-ID. LOB-APPEND.
  ENVIRONMENT DIVISION.
  DATA DIVISION.
  WORKING-STORAGE SECTION.
  01  USERID        PIC X(11) VALUES "SAMP/SAMP".
  01  DEST          SQL-BLOB.
  01  SRC           SQL-BLOB.
      EXEC SQL INCLUDE SQLCA END-EXEC.

  PROCEDURE DIVISION.
  APPEND-BLOB.
      EXEC SQL WHENEVER SQLERROR DO PERFORM SQL-ERROR END-EXEC.
      EXEC SQL CONNECT :USERID END-EXEC.

 * Allocate and initialize the BLOB locators:
      EXEC SQL ALLOCATE :DEST END-EXEC.
      EXEC SQL ALLOCATE :SRC END-EXEC.
      EXEC SQL WHENEVER NOT FOUND GOTO END-OF-BLOB END-EXEC.
      EXEC SQL SELECT AD_PHOTO INTO :DEST
           FROM PRINT_MEDIA WHERE PRODUCT_ID = 2268
           AND AD_ID = 21001 FOR UPDATE END-EXEC.
      EXEC SQL SELECT AD_PHOTO INTO :SRC
           FROM PRINT_MEDIA WHERE PRODUCT_ID = 3060
           AND AD_ID = 11001 END-EXEC.

 * Open the DESTination LOB read/write and SRC LOB read only:
      EXEC SQL LOB OPEN :DEST READ WRITE END-EXEC.
      EXEC SQL LOB OPEN :SRC READ ONLY END-EXEC.

 * Append the source LOB to the destination LOB:
      EXEC SQL LOB APPEND :SRC TO :DEST END-EXEC.
      EXEC SQL LOB CLOSE :DEST END-EXEC.
      EXEC SQL LOB CLOSE :SRC END-EXEC.

  END-OF-BLOB.
      EXEC SQL WHENEVER NOT FOUND CONTINUE END-EXEC.
      EXEC SQL FREE :DEST END-EXEC.
      EXEC SQL FREE :SRC END-EXEC.
      EXEC SQL ROLLBACK WORK RELEASE END-EXEC.
      STOP RUN.

  SQL-ERROR.
      EXEC SQL WHENEVER SQLERROR CONTINUE END-EXEC.
```

```
DISPLAY " ".
DISPLAY "ORACLE ERROR DETECTED:".
DISPLAY " ".
DISPLAY SQLERRMC.
EXEC SQL ROLLBACK WORK RELEASE END-EXEC.
STOP RUN.
```

C/C++（Pro*C/C++）：他の LOB への LOB の追加

```
/* Appending one LOB to another */
#include <oci.h>
#include <stdio.h>
#include <sqlca.h>

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

void appendLOB_proc()
{
  OCIBlobLocator *Dest_loc, *Src_loc;
  EXEC SQL WHENEVER SQLERROR DO Sample_Error();

  /* Allocate the locators: */
  EXEC SQL ALLOCATE :Dest_loc;
  EXEC SQL ALLOCATE :Src_loc;

  /* Select the destination locator: */
  EXEC SQL SELECT Sound INTO :Dest_loc
           FROM Print_media WHERE product_id = 2268 AND
           ad_id = 21001 FOR UPDATE;

  /* Select the source locator: */
  EXEC SQL SELECT Sound INTO :Src_loc
           FROM Print_media WHERE product_id = 3060 AND
           ad_id = 11001;

  /* Opening the LOBs is Optional: */
  EXEC SQL LOB OPEN :Dest_loc READ WRITE;
  EXEC SQL LOB OPEN :Src_loc READ ONLY;
```

```
  /* Append the source LOB to the end of the destination LOB: */
  EXEC SQL LOB APPEND :Src_loc TO :Dest_loc;

  /* Closing the LOBs is mandatory if they have been opened: */
  EXEC SQL LOB CLOSE :Dest_loc;
  EXEC SQL LOB CLOSE :Src_loc;

  /* Release resources held by the locators: */
  EXEC SQL FREE :Dest_loc;
  EXEC SQL FREE :Src_loc;
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  appendLOB_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

Visual Basic（OO4O）: 他の LOB への LOB の追加

```
'Appending one LOB to another
Dim MySession As OraSession
Dim OraDb As OraDatabase
Dim OraDyn As OraDynaset, OraAdPhoto1 As OraBlob, OraAdPhotoClone As OraBlob

Set MySession = CreateObject("OracleInProcServer.XOraSession")
Set OraDb = MySession.OpenDatabase("exampledb", "samp/samp", 0&)
Set OraDyn = OraDb.CreateDynaset(
   "SELECT * FROM Print_media ORDER BY product_id, ad_id", ORADYN_DEFAULT)
Set OraAdPhoto1 = OraDyn.Fields("ad_photo").Value
Set OraAdPhotoClone = OraAdPhoto1

OraDyn.MoveNext
OraDyn.Edit
OraAdPhoto1.Append OraAdPhotoClone
OraDyn.Update
```

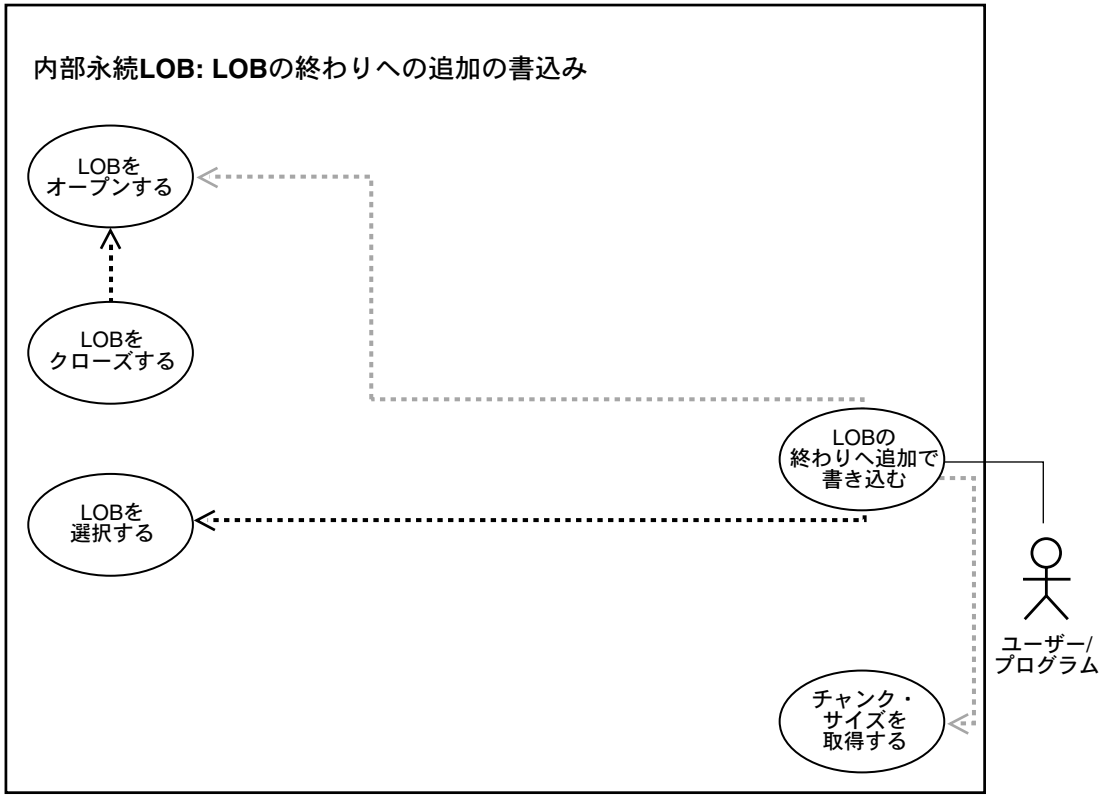
Java（JDBC）：他の LOB への LOB の追加

```
// Appending one LOB to another
import java.io.InputStream;
import java.io.OutputStream;

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

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

public class Ex2_121
{
  static final int MAXBUFSIZE = 32767;
  public static void main (String args [])
       throws Exception
  {
    // Load the Oracle JDBC driver:
    DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

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

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

    // Create a Statement:
    Statement stmt = conn.createStatement ();
    try
    {
     ResultSet rset = null;
     BLOB dest_loc = null;
     BLOB src_loc = null;
   InputStream in = null;
   byte[] buf = new byte[MAXBUFSIZE];
   int length = 0;
   long pos = 0;
      rset = stmt.executeQuery (
```

```
          "SELECT ad_photo FROM Print_media
           WHERE product_id = 2268 AND ad_id = 21001");
    if (rset.next())
    {
      src_loc = ((OracleResultSet)rset).getBLOB (1);
    }
    in = src_loc.getBinaryStream();

        rset = stmt.executeQuery (
           "SELECT ad_photo FROM Print_media
            WHERE product_id = 3060 AND ad_id = 11001 FOR UPDATE");
    if (rset.next())
    {
      dest_loc = ((OracleResultSet)rset).getBLOB (1);
    }
    // Start writing at the end of the LOB.  ie. append:
    pos = dest_loc.length();
    while ((length = in.read(buf)) != -1)
    {
       // Write the contents of the buffer into position pos of the output LOB:
       dest_loc.putBytes(pos, buf);
       pos += length;
    }

    // Close all streams and handles:
    in.close();
    stmt.close();
    conn.commit();
    conn.close();

     }
     catch (SQLException e)
     {
        e.printStackTrace();
     }
  }
}
```

LOB の終わりへの追加の書込み

図 10-28 利用図 : LOB の終わりへの追加の書込み



参照 : 10-2 ページの表 10-1「利用モデル : 内部永続 LOB」を参照してください。

用途

LOB の終わりへ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 の終わりに書き込まれます。

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。

- コールバック関数またはファンクションが 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またはファンクション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。
- コールバック関数またはファンクション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、5-5 ページの「更新済ロケータを介した LOB の更新」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関数またはファンクションのリストは、第 3 章「様々なプログラム環境での LOB のサポート」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）：『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』の第 23 章「DBMS_LOB」の「DBMS_LOB サブプログラムの要約」の「WRITEAPPEND プロシージャ」
- C（OCI）：『Oracle Call Interface プログラマーズ・ガイド』の第 16 章「その他の OCI リレーショナル関数」の「LOB 関数」の OCILobWriteAppend()
- C++（OCCI）：『Oracle C++ Call Interface プログラマーズ・ガイド』

- COBOL（Pro*COBOL）：『Pro*COBOL Precompiler プログラマーズ・ガイド』の LOB に関する詳細、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・ディレクティブ」の「LOB WRITE（実行可能埋込み SQL 拡張機能）」
- C/C++（Pro*C/C++）：『Pro*C/C++ Precompiler プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびディレクティブ」の「LOB WRITE（実行可能埋込み SQL 拡張機能）」、「LOB APPEND（実行可能埋込み SQL 拡張機能）」
- Visual Basic（OO4O）：参照マニュアルはありません。
- Java（JDBC）：『Oracle9*i* JDBC 開発者ガイドおよびリファレンス』の第 8 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『Oracle9*i* 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「JDBC 2.0 LOB 型と Oracle 拡張型のサポート」の「BLOB、CLOB および BFILE のサポート」

使用例

次の例では、イメージの終わりに書込みを行います。

例

次のプログラム環境での例を示します。

- PL/SQL（DBMS_LOB）：LOB の終わりへの追加の書込み（10-195 ページ）
- C（OCI）：LOB の終わりへの追加の書込み（10-196 ページ）
- C++（OCCI）：今回のリリースでは例は提供されません。
- COBOL（Pro*COBOL）：LOB の終わりへの追加の書込み（10-198 ページ）
- C/C++（Pro*C/C++）：LOB の終わりへの追加の書込み（10-199 ページ）
- Visual Basic（OO4O）：今回のリリースでは例は提供されません。
- Java（JDBC）：LOB の終わりへの追加の書込み（10-200 ページ）

PL/SQL（DBMS_LOB）：LOB の終わりへの追加の書込み

```
/* Write-appending to a LOB
/* Example procedure lobWriteAppend_proc is not part of DBMS_LOB package: */
CREATE OR REPLACE PROCEDURE lobWriteAppend_proc IS
   Lob_loc    BLOB;
   Buffer     RAW(32767);
   Amount     Binary_integer := 32767;
BEGIN
   SELECT ad_composite INTO Lob_loc FROM Print_media
   WHERE product_id = 2056 AND ad_id =  12001 FOR UPDATE;
   /* Fill the buffer with data... */
```

```
   /* Opening the LOB is optional: */
   DBMS_LOB.OPEN (Lob_loc, DBMS_LOB.LOB_READWRITE);
   /* Append the data from the buffer to the end of the LOB: */
   DBMS_LOB.WRITEAPPEND(Lob_loc, Amount, Buffer);
   /* Closing the LOB is mandatory if you have opened it: */
   DBMS_LOB.CLOSE(Lob_loc);
END;
```

C（OCI）: LOB の終わりへの追加の書込み

```
/* Write-appending to a LOB */
/* Select the locator into a locator variable: */
sb4 select_lock_adcomp_locator(Lob_loc, errhp, svchp, stmthp)
OCILobLocator *Lob_loc;
OCIError      *errhp;
OCISvcCtx     *svchp;
OCIStmt       *stmthp;
{
  text  *sqlstmt =
    (text *)"SELECT ad_composite FROM Print_media
             WHERE product_id=2268 AND ad_id = 21001 FOR UPDATE";
  OCIDefine *defnp1;
  checkerr (errhp, OCIStmtPrepare(stmthp, errhp, sqlstmt,
                                  (ub4)strlen((char *)sqlstmt),
                                  (ub4) OCI_NTV_SYNTAX, (ub4) OCI_DEFAULT));
  checkerr (errhp, OCIDefineByPos(stmthp, &defnp1, errhp, (ub4) 1,
                                  (dvoid *)&Lob_loc, (sb4)0,
                                  (ub2) SQLT_BLOB,(dvoid *) 0,
                                  (ub2 *) 0, (ub2 *) 0, (ub4) OCI_DEFAULT));

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

  return (0);
}

#define MAXBUFLEN 32767

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

```
  OCIBlobLocator *Lob_loc;
  ub4 amt;
  ub4 offset;
  sword retval;
  ub1 bufp[MAXBUFLEN];
  ub4 buflen;

  /* Allocate the locator: */
  (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &Lob_loc,
                        (ub4)OCI_DTYPE_LOB, (size_t) 0, (dvoid **) 0);

  /* Select the BLOB: */
  printf(" select and lock an ad-composite locator\n");
  select_lock_adcomp_locator(Lob_loc, errhp, svchp, stmthp);

  /* Open the BLOB: */
  checkerr (errhp, (OCILobOpen(svchp, errhp, Lob_loc, OCI_LOB_READWRITE)));

  /* Setting the amt to the buffer length.  Note here that amt is in bytes
     since we are using a BLOB: */
  amt    = sizeof(bufp);
  buflen = sizeof(bufp);

  /* Fill bufp with data: */
  /* Write the data from the buffer at the end of the LOB: */
  printf(" write-append data to the frame Lob\n");
  checkerr (errhp, OCILobWriteAppend (svchp, errhp, Lob_loc, &amt,
                            bufp, buflen,
                            OCI_ONE_PIECE, (dvoid *)0,
                            (sb4 (*)(dvoid *, dvoid *, ub4 *, ub1 *))0,
                            0, SQLCS_IMPLICIT));

  /* Closing the BLOB is mandatory if you have opened it: */
  checkerr (errhp, OCILobClose(svchp, errhp, Lob_loc));
  /* Free resources held by the locators: */
  (void) OCIDescriptorFree((dvoid *) Lob_loc, (ub4) OCI_DTYPE_LOB);

  return;
}
```

COBOL（Pro*COBOL）: LOB の終わりへの追加の書込み

```
* WRITE-APPENDING TO A LOB
 IDENTIFICATION DIVISION.
 PROGRAM-ID. WRITE-APPEND-BLOB.
 ENVIRONMENT DIVISION.
 DATA DIVISION.
 WORKING-STORAGE SECTION.

 01  BLOB1         SQL-BLOB.
 01  AMT           PIC S9(9) COMP.
 01  BUFFER        PIC X(32767) VARYING.
     EXEC SQL VAR BUFFER IS LONG RAW (32767) END-EXEC.
 01  USERID   PIC X(11) VALUES "SAMP/SAMP".
     EXEC SQL INCLUDE SQLCA END-EXEC.

 PROCEDURE DIVISION.
 WRITE-APPEND-BLOB.

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

* Allocate and initialize the BLOB locators:
     EXEC SQL ALLOCATE :BLOB1 END-EXEC.
     EXEC SQL WHENEVER NOT FOUND GOTO END-OF-BLOB END-EXEC.
     EXEC SQL SELECT AD_COMPOSITE INTO :BLOB1
          FROM PRINT_MEDIA
          WHERE PRODUCT_ID = 3106 AND AD_ID = 13001 FOR UPDATE END-EXEC.

* Open the target LOB:
     EXEC SQL LOB OPEN :BLOB1 READ WRITE END-EXEC.

*    Populate AMT here:
     MOVE 5 TO AMT.
     MOVE "2424242424" to BUFFER.

* Append the source LOB to the destination LOB:
     EXEC SQL LOB WRITE APPEND :AMT FROM :BUFFER INTO :BLOB1 END-EXEC.
     EXEC SQL LOB CLOSE :BLOB1 END-EXEC.

 END-OF-BLOB.
     EXEC SQL WHENEVER NOT FOUND CONTINUE END-EXEC.
     EXEC SQL FREE :BLOB1 END-EXEC.
     EXEC SQL ROLLBACK WORK RELEASE END-EXEC.
     STOP RUN.

 SQL-ERROR.
```

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

C/C++（Pro*C/C++）: LOB の終わりへの追加の書込み

```
/* Write-appending to a LOB */
#include <oci.h>
#include <stdio.h>
#include <sqlca.h>

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

#define BufferLength 128

void LobWriteAppend_proc()
{
  OCIBlobLocator *Lob_loc;
  int Amount = BufferLength;
  /* Amount == BufferLength so only a single WRITE is needed: */
  char Buffer[BufferLength];
  /* Datatype equivalencing is mandatory for this datatype: */
  EXEC SQL VAR Buffer IS RAW(BufferLength);
  EXEC SQL ALLOCATE :Lob_loc;
  EXEC SQL SELECT ad_composite INTO :Lob_loc
           FROM Print_media
           WHERE product_id = 3060 AND ad_id = 11001 FOR UPDATE;
  /* Opening the LOB is Optional: */
  EXEC SQL LOB OPEN :Lob_loc;
  memset((void *)Buffer, 1, BufferLength);
  /* Write the data from the buffer at the end of the LOB: */
  EXEC SQL LOB WRITE APPEND :Amount FROM :Buffer INTO :Lob_loc;
  /* Closing the LOB is mandatory if it has been opened: */
  EXEC SQL LOB CLOSE :Lob_loc;
  EXEC SQL FREE :Lob_loc;
}
```

```
void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  LobWriteAppend_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

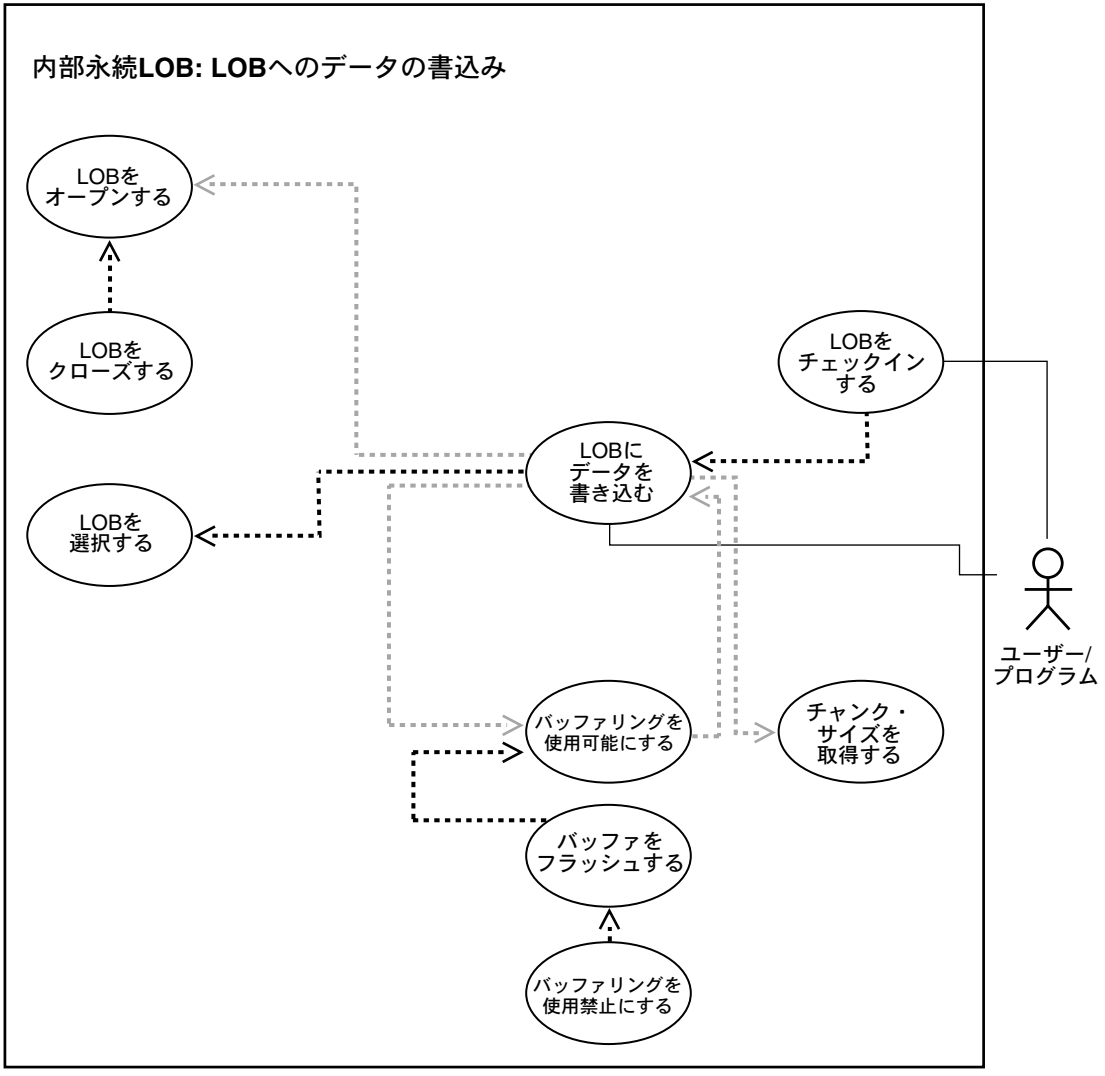
Java（JDBC）: LOB の終わりへの追加の書込み

```
// Write-appending to a LOB
import java.io.OutputStream;

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

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

public class Ex2_126
{
  static final int MAXBUFSIZE = 32767;
  public static void main (String args [])
       throws Exception
  {
    // Load the Oracle JDBC driver:
    DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

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

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

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

```
    {
     BLOB dest_loc = null;
     byte[] buf = new byte[MAXBUFSIZE];
     long pos = 0;
     ResultSet rset = stmt.executeQuery (
        "SELECT ad_composite FROM Print_media
         WHERE product_id = 2056 AND ad_id = 12001 FOR UPDATE");
   if (rset.next())
   {
     dest_loc = ((OracleResultSet)rset).getBLOB (1);
   }

   // Start writing at the end of the LOB. ie. append:
   pos = dest_loc.length();

   // fill buf with contents to be written:
   buf = (new String("Hello World")).getBytes();

   // Write the contents of the buffer into position pos of the output LOB:
   dest_loc.putBytes(pos, buf);

   // Close all streams and handles:
   stmt.close();
   conn.commit();
   conn.close();

    }
    catch (SQLException e)
    {
       e.printStackTrace();
    }
  }
}
```

LOB へのデータの書込み

図 10-29 利用図 : LOB へのデータの書込み



参照： 10-2 ページの表 10-1「利用モデル : 内部永続 LOB」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

ストリーム書込み 大量の LOB データを最も効率よく書き込むには、ポーリングまたはコールバックによってストリーム・メカニズムを有効にした OCILobWrite() を使用します。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、その後の操作で読み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 ファンクションは、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。

書込みパフォーマンスを改善するためのチャンク・サイズの複数倍の指定 このチャンク・サイズの複数倍を指定して書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 チャンクが書込み要求ごとにバージョン化されるためです。すべての書込みを 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、5-5 ページの「更新済ロケータを介した LOB の更新」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。

- 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関数またはファンクションのリストは、第 3 章「様々なプログラム環境での LOB のサポート」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）：『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』の第 23 章「DBMS_LOB」の「DBMS_LOB サブプログラムの要約」の「WRITE プロシージャ」
- C（OCI）：『Oracle Call Interface プログラマーズ・ガイド』の第 16 章「その他の OCI リレーショナル関数」の「LOB 関数」の OCILobWrite()
- C++（OCCI）：『Oracle C++ Call Interface プログラマーズ・ガイド』

- COBOL（Pro*COBOL）：『Pro*COBOL Precompiler プログラマーズ・ガイド』の LOB に関する詳細、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・ディレクティブ」の「LOB WRITE（実行可能埋込み SQL 拡張機能）」
- C/C++（Pro*C/C++）：『Pro*C/C++ Precompiler プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびディレクティブ」の「LOB WRITE（実行可能埋込み SQL 拡張機能）」
- Visual Basic（OO4O オンライン・ヘルプ）：ヘルプの「目次」タブから、「OO4O オートメーション・サーバー・リファレンス」>「OO4O サーバーのオブジェクト」>「OraBLOB、OraCLOB」>「メソッド」>「write」、「copyfromfile」を選択
- Java（JDBC）：『Oracle9*i* JDBC 開発者ガイドおよびリファレンス』の第 8 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『Oracle9*i* 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「JDBC 2.0 LOB 型と Oracle 拡張型のサポート」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 ad_sourcetext データ（広告用のテキスト）を更新できます。

例

次のプログラム環境での例を示します。

- PL/SQL（DBMS_LOB）：LOB へのデータの書込み（10-205 ページ）
- C（OCI）：LOB へのデータの書込み（10-207 ページ）
- COBOL（Pro*COBOL）：LOB へのデータの書込み（10-210 ページ）
- C/C++（Pro*C/C++）：LOB へのデータの書込み（10-212 ページ）
- Visual Basic（OO4O）：LOB へのデータの書込み（10-215 ページ）
- Java（JDBC）：LOB へのデータの書込み（10-216 ページ）

PL/SQL（DBMS_LOB）：LOB へのデータの書込み

```
/* Writing data to a LOB */
/* Example procedure writeDataToLOB_proc is not part of DBMS_LOB package. */
CREATE or REPLACE PROCEDURE writeDataToLOB_proc IS
   Lob_loc          CLOB;
   Buffer           VARCHAR2(32767);
   Amount           BINARY_INTEGER := 32767;
   Position         INTEGER := 1;
   i                INTEGER;
BEGIN
```

```
   /* Select a LOB: */
   SELECT ad_sourcetext INTO Lob_loc
        FROM Print_media
           WHERE product_id = 2056 AND ad_id = 12001 FOR UPDATE;
   /* Opening the LOB is optional: */
   DBMS_LOB.OPEN (Lob_loc, DBMS_LOB.LOB_READWRITE);
   /* Fill the buffer with data to write to the LOB: */
   FOR i IN 1..3 LOOP
      DBMS_LOB.WRITE (Lob_loc, Amount, Position, Buffer);
      /* Fill the buffer with more data to write to the LOB: */
      Position := Position + Amount;
   END LOOP;
   /* Closing the LOB is mandatory if you have opened it: */
   DBMS_LOB.CLOSE (Lob_loc);
END;

/* We add a second example to show a case in which the buffer size and amount
   differs from the first example: */
CREATE or REPLACE PROCEDURE writeDataToLOB_proc IS
   Lob_loc         CLOB;
   Buffer          VARCHAR2(32767);
   Amount          BINARY_INTEGER := 32767;
   Position        INTEGER;
   i               INTEGER;
   Chunk_size      INTEGER;
BEGIN
  SELECT ad_sourcetext INTO Lob_loc
        FROM Print_media
           WHERE product_id = 2056 AND ad_id = 12001 FOR UPDATE;
     /* Opening the LOB is optional: */
    DBMS_LOB.OPEN (Lob_loc, DBMS_LOB.LOB_READWRITE);
    Chunk_size := DBMS_LOB.GETCHUNKSIZE(Lob_loc);

    /* Fill the buffer with 'Chunk_size' worth of data to write to
       the LOB. Use the chunk size (or a multiple of chunk size) when writing
       data to the LOB.  Make sure that you write within a chunk boundary and
       don't overlap different chunks within a single call to DBMS_LOB.WRITE. */

    Amount := Chunk_size;

    /* Write data starting at the beginning of the second chunk: */
    Position := Chunk_size + 1;
    FOR i IN 1..3 LOOP
        DBMS_LOB.WRITE (Lob_loc, Amount, Position, Buffer);
```

```
        /* Fill the buffer with more data (of size Chunk_size) to  write to
           the LOB: */
        Position := Position + Amount;
    END LOOP;
    /* Closing the LOB is mandatory if you have opened it: */
    DBMS_LOB.CLOSE (Lob_loc);
END;
```

C（OCI）: LOB へのデータの書込み

```
/* Writing data to a LOB.
   Using OCI you can write arbitrary amounts of data
   to an Internal LOB in either a single piece or in multiple pieces using
   streaming with standard polling. A dynamically allocated Buffer
   holds the data being written to the LOB. */

/* Select the locator into a locator variable */
sb4 select_lock_adsourcetext_locator(Lob_loc, errhp, svchp, stmthp)
OCILobLocator *Lob_loc;
OCIError      *errhp;
OCISvcCtx     *svchp;
OCIStmt       *stmthp;
{
  text  *sqlstmt =
    (text *) "SELECT ad_sourcetext FROM Print_media pm  \
                WHERE pm.product_id = 2268 ANd ad_id = 21001 FOR UPDATE";
  OCIDefine *defnp1;
  checkerr (errhp, OCIStmtPrepare(stmthp, errhp, sqlstmt,
                                   (ub4)strlen((char *)sqlstmt),
                                   (ub4) OCI_NTV_SYNTAX, (ub4) OCI_DEFAULT));
  checkerr (errhp, OCIDefineByPos(stmthp, &defnp1, errhp, (ub4) 1,
                                   (dvoid *)&Lob_loc, (sb4)0,
                                   (ub2) SQLT_CLOB,(dvoid *) 0,
                                   (ub2 *) 0, (ub2 *) 0, (ub4) OCI_DEFAULT));
 /* Execute the select and fetch one row */
  checkerr(errhp, OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                                  (CONST OCISnapshot*) 0, (OCISnapshot*) 0,
                                  (ub4) OCI_DEFAULT));
  return (0);
}

#define MAXBUFLEN 32767
void writeDataToLob(envhp, errhp, svchp, stmthp)
OCIEnv    *envhp;
OCIError  *errhp;
OCISvcCtx *svchp;
```

```
OCIStmt   *stmthp;
{
  OCIClobLocator *Lob_loc;
  ub4 Total = 2.5*MAXBUFLEN;
                    /* <total amount of data to write to the CLOB in bytes> */
  unsigned int amt;
  unsigned int offset;
  unsigned int remainder, nbytes;
  boolean last;
  ub1 bufp[MAXBUFLEN];
  sb4 err;

  /* Allocate the locators desriptors*/
  (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &Lob_loc,
                            (ub4)OCI_DTYPE_LOB, (size_t) 0, (dvoid **) 0);

  /* Select the CLOB */
  printf (" select an ad_source_text Lob\n");
  select_lock_adsourcetext_locator(Lob_loc, errhp, svchp, stmthp);

  /* Open the CLOB */
  checkerr (errhp, (OCILobOpen(svchp, errhp, Lob_loc, OCI_LOB_READWRITE)));
  if (Total > MAXBUFLEN)
    nbytes = MAXBUFLEN;   /* We will use streaming via standard polling */
  else
    nbytes = Total;                     /* Only a single write is required */

  /* Fill the buffer with nbytes worth of data */
  remainder = Total - nbytes;

  /* Setting Amount to 0 streams the data until use specifies OCI_LAST_PIECE */
  amt = 0;
  offset = 1;

  printf(" write the Lob data in pieces\n");
  if (0 == remainder)
  {
    amt = nbytes;
    /* Here, (Total <= MAXBUFLEN ) so we can write in one piece */
    checkerr (errhp, OCILobWrite (svchp, errhp, Lob_loc, &amt,
                                  offset, bufp, nbytes,
                                  OCI_ONE_PIECE, (dvoid *)0,
                                  (sb4 (*)(dvoid*,dvoid*,ub4*,ub1 *))0,
                                  0, SQLCS_IMPLICIT));
  }
  else
  {
```

```
  /* Here (Total > MAXBUFLEN ) so we use streaming via standard polling */
  /* write the first piece.  Specifying first initiates polling. */
  err =  OCILobWrite (svchp, errhp, Lob_loc, &amt,
                      offset, bufp, nbytes,
                      OCI_FIRST_PIECE, (dvoid *)0,
                      (sb4 (*)(dvoid*,dvoid*,ub4*,ub1 *))0,
                      0, SQLCS_IMPLICIT);
  if (err != OCI_NEED_DATA)
    checkerr (errhp, err);
  last = FALSE;
  /* Write the next (interim) and last pieces */
  do
  {
    if (remainder > MAXBUFLEN)
      nbytes = MAXBUFLEN;            /* Still have more pieces to go */
    else
    {
      nbytes = remainder;      /* Here, (remainder <= MAXBUFLEN) */
      last = TRUE;             /* This is going to be the final piece */
    }

    /* Fill the Buffer with nbytes worth of data */
    if (last)
    {
      /* Specifying LAST terminates polling */
      err = OCILobWrite (svchp, errhp, Lob_loc, &amt,
                         offset, bufp, nbytes,
                         OCI_LAST_PIECE, (dvoid *)0,
                         (sb4 (*)(dvoid*,dvoid*,ub4*,ub1 *))0,
                         0, SQLCS_IMPLICIT);
      if (err != OCI_SUCCESS)
        checkerr(errhp, err);
    }
    else
    {
      err = OCILobWrite (svchp, errhp, Lob_loc, &amt,
                         offset, bufp, nbytes,
                         OCI_NEXT_PIECE, (dvoid *)0,
                         (sb4 (*)(dvoid*,dvoid*,ub4*,ub1 *))0,
                         0, SQLCS_IMPLICIT);
      if (err != OCI_NEED_DATA)
        checkerr (errhp, err);
    }
    /* Determine how much is left to write */
    remainder = remainder - nbytes;
  } while (!last);
}
```

```
 /* At this point, (remainder == 0) */

 /* Closing the LOB is mandatory if you have opened it */
 checkerr (errhp, OCILobClose(svchp, errhp, Lob_loc));

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

 return;
}
```

COBOL（Pro*COBOL）: LOB へのデータの書込み

```
* WRITING DATA TO A LOB
  IDENTIFICATION DIVISION.
  PROGRAM-ID. WRITE-CLOB.
  ENVIRONMENT DIVISION.
  INPUT-OUTPUT SECTION.
  FILE-CONTROL.
     SELECT INFILE
        ASSIGN TO "datfile.dat"
        ORGANIZATION IS SEQUENTIAL.
  DATA DIVISION.
  FILE SECTION.

  FD INFILE
     RECORD CONTAINS 5 CHARACTERS.
  01 INREC      PIC X(5).

  WORKING-STORAGE SECTION.
  01  CLOB1          SQL-CLOB.
  01  BUFFER         PIC X(5) VARYING.
  01  AMT            PIC S9(9) COMP VALUES 321.
  01  OFFSET         PIC S9(9) COMP VALUE 1.
  01  END-OF-FILE    PIC X(1) VALUES "N".
  01  D-BUFFER-LEN   PIC 9.
  01  D-AMT          PIC 9.
  01  USERID   PIC X(11) VALUES "SAMP/SAMP".

      EXEC SQL INCLUDE SQLCA END-EXEC.

  PROCEDURE DIVISION.
  WRITE-CLOB.
      EXEC SQL WHENEVER SQLERROR GOTO SQL-ERROR END-EXEC.
      EXEC SQL
```

```
          CONNECT :USERID
     END-EXEC.

* Open the input file:
     OPEN INPUT INFILE.
* Allocate and initialize the CLOB locator:
     EXEC SQL ALLOCATE :CLOB1 END-EXEC.
     EXEC SQL
          SELECT AD_SOURCETEXT INTO :CLOB1 FROM PRINT_MEDIA
          WHERE PRODUCT_ID = 3106 AND AD_ID = 13001 FOR UPDATE
     END-EXEC.

* Either write entire record or write first piece
* Read a data file here and populate BUFFER-ARR and BUFFER-LEN
* END-OF-FILE will be set to "Y" when the entire file has been
* read.
     PERFORM READ-NEXT-RECORD.
     MOVE INREC TO BUFFER-ARR.
     MOVE 5 TO BUFFER-LEN.
     IF (END-OF-FILE = "Y")
        EXEC SQL
             LOB WRITE ONE :AMT FROM :BUFFER
             INTO :CLOB1 AT :OFFSET
        END-EXEC
     ELSE
        DISPLAY "LOB WRITE FIRST: ", BUFFER-ARR
        EXEC SQL
           LOB WRITE FIRST :AMT FROM :BUFFER
           INTO :CLOB1 AT :OFFSET
        END-EXEC.

* Continue reading from the input data file
* and writing to the CLOB:
     PERFORM READ-NEXT-RECORD.
     PERFORM WRITE-TO-CLOB
        UNTIL END-OF-FILE = "Y".
     MOVE INREC TO BUFFER-ARR.
     MOVE 1 TO BUFFER-LEN.
     DISPLAY "LOB WRITE LAST: ", BUFFER-ARR(1:BUFFER-LEN).
     EXEC SQL
          LOB WRITE LAST :AMT FROM :BUFFER INTO :CLOB1
     END-EXEC.
     EXEC SQL
          ROLLBACK WORK RELEASE
     END-EXEC.
     STOP RUN.
```

```
 WRITE-TO-CLOB.
   IF ( END-OF-FILE = "N")
      MOVE INREC TO BUFFER-ARR.
      MOVE 5 TO BUFFER-LEN.
      DISPLAY "LOB WRITE NEXT: ", BUFFER-ARR(1:BUFFER-LEN).
      EXEC SQL
          LOB WRITE NEXT :AMT FROM :BUFFER INTO :CLOB1
      END-EXEC.
      PERFORM READ-NEXT-RECORD.

 READ-NEXT-RECORD.
    MOVE SPACES TO INREC.
    READ INFILE NEXT RECORD
         AT END
          MOVE "Y" TO END-OF-FILE.
    DISPLAY "END-OF-FILE IS " END-OF-FILE.

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

C/C++（Pro*C/C++）: LOB へのデータの書込み

```
/* Writing data to a LOB */
/* This example shows how you can use Pro*C/C++ to write
   arbitrary amounts of data to an Internal LOB in either a single piece
   of in multiple pieces using a Streaming Mechanism that utilizes standard
   polling.  A dynamically allocated Buffer holds the data being
   written to the LOB: */
#include <oci.h>
#include <stdio.h>
#include <string.h>
#include <sqlca.h>

void Sample_Error()
{
  EXEC SQL WHENEVER SQLERROR CONTINUE;
```

```
  printf("%.*s\n", sqlca.sqlerrm.sqlerrml, sqlca.sqlerrm.sqlerrmc);
  EXEC SQL ROLLBACK WORK RELEASE;
  exit(1);
}

#define BufferLength 1024

void writeDataToLOB_proc(multiple) int multiple;
{
  OCIClobLocator *Lob_loc;
  varchar Buffer[BufferLength];
  unsigned int Total;
  unsigned int Amount;
  unsigned int remainder, nbytes;
  boolean last;

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  /* Allocate and Initialize the Locator: */
  EXEC SQL ALLOCATE :Lob_loc;
  EXEC SQL SELECT ad_sourcetext INTO Lob_loc
           FROM Print_media WHERE product_id = 3060 AND ad_id = 11001 FOR UPDATE;
  /* Open the CLOB: */
  EXEC SQL LOB OPEN :Lob_loc READ WRITE;
  Total = Amount = (multiple * BufferLength);
  if (Total > BufferLength)
    nbytes = BufferLength;   /* We will use streaming via standard polling */
  else
    nbytes = Total;                     /* Only a single write is required */
  /* Fill the buffer with nbytes worth of data: */
  memset((void *)Buffer.arr, 32, nbytes);
  Buffer.len = nbytes;       /* Set the Length */
  remainder = Total - nbytes;
  if (0 == remainder)
    {
      /* Here, (Total <= BufferLength) so we can write in one piece: */
      EXEC SQL LOB WRITE ONE :Amount FROM :Buffer INTO :Lob_loc;
      printf("Write ONE Total of %d characters\n", Amount);
    }
  else
    {
      /* Here (Total > BufferLength) so we streaming via standard polling */
      /* write the first piece.  Specifying first initiates polling: */
      EXEC SQL LOB WRITE FIRST :Amount FROM :Buffer INTO :Lob_loc;
      printf("Write first %d characters\n", Buffer.len);
      last = FALSE;
      /* Write the next (interim) and last pieces: */
      do
```

```
        {
          if (remainder > BufferLength)
            nbytes = BufferLength;        /* Still have more pieces to go */
          else
            {
              nbytes = remainder;    /* Here, (remainder <= BufferLength) */
              last = TRUE;         /* This is going to be the Final piece */
            }
          /* Fill the buffer with nbytes worth of data: */
          memset((void *)Buffer.arr, 32, nbytes);
          Buffer.len = nbytes;       /* Set the Length */
          if (last)
            {
              EXEC SQL WHENEVER SQLERROR DO Sample_Error();
              /* Specifying LAST terminates polling: */
              EXEC SQL LOB WRITE LAST :Amount FROM :Buffer INTO :Lob_loc;
              printf("Write LAST Total of %d characters\n", Amount);
            }
          else
            {
              EXEC SQL WHENEVER SQLERROR DO break;
              EXEC SQL LOB WRITE NEXT :Amount FROM :Buffer INTO :Lob_loc;
              printf("Write NEXT %d characters\n", Buffer.len);
            }
          /* Determine how much is left to write: */
          remainder = remainder - nbytes;
        } while (!last);
    }
  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  /* At this point, (Amount == Total), the total amount that was written */
  /* Close the CLOB: */
  EXEC SQL LOB CLOSE :Lob_loc;
  /* Free resources held by the Locator: */
  EXEC SQL FREE :Lob_loc;
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  writeDataToLOB_proc(1);
  EXEC SQL ROLLBACK WORK;
  writeDataToLOB_proc(4);
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

Visual Basic（OO4O）: LOB へのデータの書込み

```
'Writing data to a LOB
'There are two ways of writing a lob, with orablob.write or
orablob.copyfromfile

'Using the OraBlob.Write mechanism
Dim OraDyn As OraDynaset, OraAdPhoto As OraBlob, amount_written%, chunksize%,
curchunk() As Byte

chunksize = 32767
Set OraDyn = OraDb.CreateDynaset("SELECT * FROM Print_media", ORADYN_DEFAULT)
Set OraAdPhoto = OraDyn.Fields("ad_photo").Value

fnum = FreeFile
Open "c:\tmp\keyboard_3106_13001" For Binary As #fnum

OraAdPhoto.offset = 1
OraAdPhoto.pollingAmount = LOF(fnum)
remainder = LOF(fnum)

Dim piece As Byte
Get #fnum, , curchunk

OraDyn.Edit

piece = ORALOB_FIRST_PIECE
OraAdPhoto.Write curchunk, chunksize, ORALOB_FIRST_PIECE

While OraAdPhoto.Status = ORALOB_NEED_DATA
   remainder = remainder - chunksize
   If remainder  <= chunksize Then
      chunksize = remainder
      piece = ORALOB_LAST_PIECE
   Else
      piece = ORALOB_NEXT_PIECE
   End If

   Get #fnum, , curchunk
   OraAdPhoto.Write curchunk, chunksize, piece

Wend

OraDyn.Update

'Using the OraBlob.CopyFromFile mechanism
```

```
Set OraDyn = OraDb.CreateDynaset("select * from Print_media", ORADYN_DEFAULT)
Set OraAdPhoto = OraDyn.Fields("ad_photo").Value

Oradyn.Edit
OraAdPhoto.CopyFromFile "c:\keyboardphoto3106.jpg"
Oradyn.Update
```

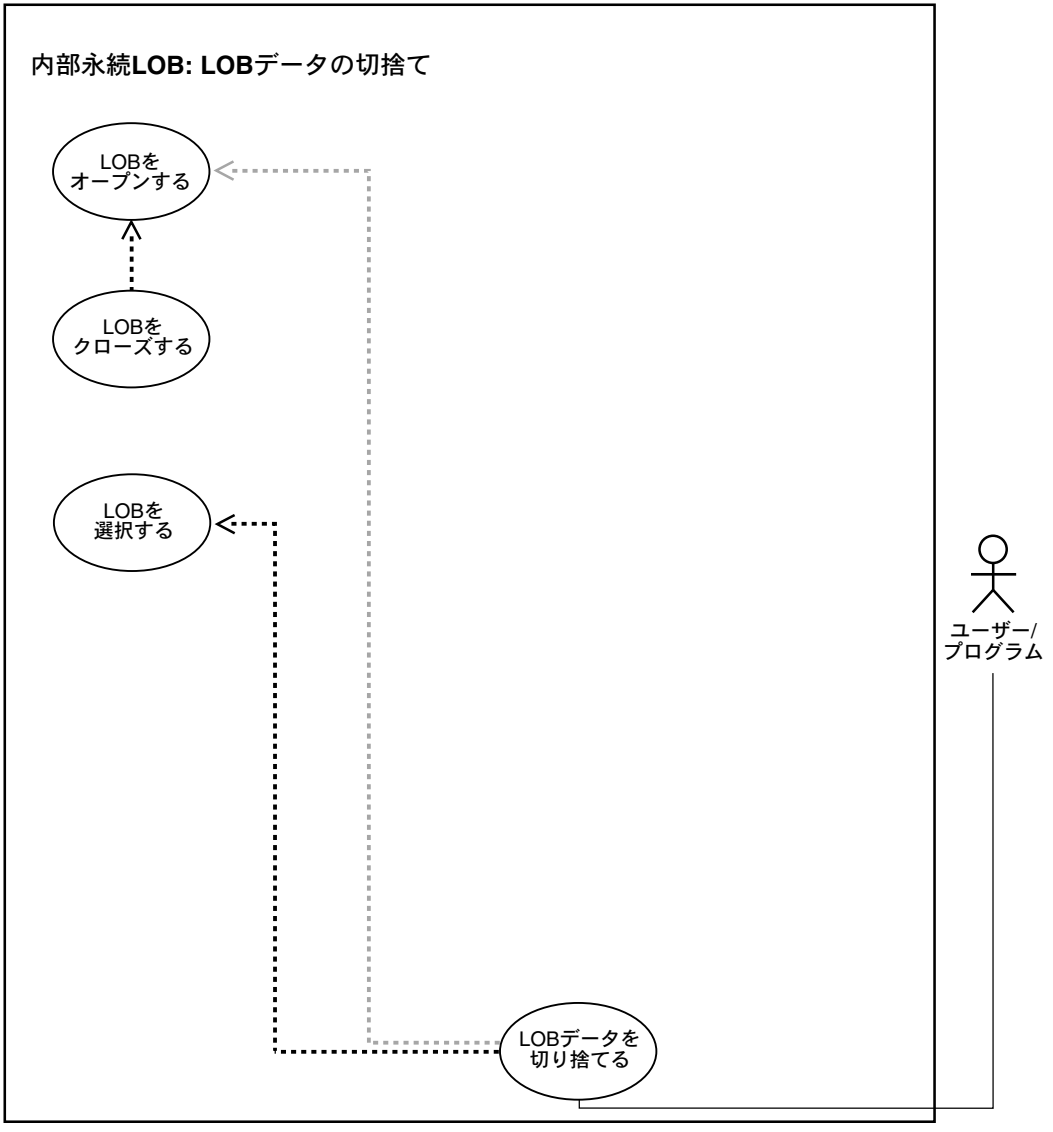
Java（JDBC）: LOB へのデータの書込み

```
//Writing data to a LOB
import java.io.OutputStream;

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

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

public class Ex2_126
{
  static final int MAXBUFSIZE = 32767;
  public static void main (String args [])
       throws Exception
  {
    // Load the Oracle JDBC driver:
    DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

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

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

    // Create a Statement:
    Statement stmt = conn.createStatement ();
    try
    {
     BLOB dest_loc = null;
```

```
      byte[] buf = new byte[MAXBUFSIZE];
      long pos = 0;
      ResultSet rset = stmt.executeQuery (
         "SELECT ad_composite FROM Print_media
          WHERE product_id = 2056 AND ad_id = 12001 FOR UPDATE");
    if (rset.next())
    {
      dest_loc = ((OracleResultSet)rset).getBLOB (1);
    }

    // Start writing at the end of the LOB. ie. append:
    pos = dest_loc.length();

    // fill buf with contents to be written:
    buf = (new String("Hello World")).getBytes();

    // Write the contents of the buffer into position pos of the output LOB:
    dest_loc.putBytes(pos, buf);

    // Close all streams and handles:
    stmt.close();
    conn.commit();
    conn.close();

     }
     catch (SQLException e)
     {
        e.printStackTrace();
     }
   }
}
```

LOB データの切捨て

図 10-30 利用図 : LOB データの切捨て



参照： 10-2 ページの表 10-1「利用モデル : 内部永続 LOB」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、次の方法を使用すると明示的にロックを行うことができます。

- SQL プログラムおよび PL/SQL プログラムでは SELECT FOR UPDATE 文
- OCI プログラムでは OCI pin または lock 関数

更新後のロケータの状態の詳細は、5-5 ページの「更新済ロケータを介した LOB の更新」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関数またはファンクションのリストは、第 3 章「様々なプログラム環境での LOB のサポート」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）：『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』の第 23 章「DBMS_LOB」の「DBMS_LOB サブプログラムの要約」の「TRIM プロシージャ」
- C（OCI）：『Oracle Call Interface プログラマーズ・ガイド』の第 16 章「その他の OCI リレーショナル関数」の「LOB 関数」の OCILobTrim()
- C++（OCCI）：『Oracle C++ Call Interface プログラマーズ・ガイド』
- COBOL（Pro*COBOL）：『Pro*COBOL Precompiler プログラマーズ・ガイド』の LOB に関する詳細、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・ディレクティブ」の「LOB TRIM（実行可能埋込み SQL 拡張機能）」
- C/C++（Pro*C/C++）：『Pro*C/C++ Precompiler プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびディレクティブ」の「LOB TRIM（実行可能埋込み SQL 拡張機能）」
- Visual Basic（OO4O オンライン・ヘルプ）：ヘルプの「目次」タブから、「OO4O オートメーション・サーバー・リファレンス」>「OO4O サーバーのオブジェクト」>「OraBLOB、OraCLOB」>「メソッド」>「trim」を選択

- Java（JDBC）：『Oracle9*i* JDBC 開発者ガイドおよびリファレンス』の第 8 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『Oracle9*i* 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「JDBC 2.0 LOB 型と Oracle 拡張型のサポート」の「BLOB、CLOB および BFILE のサポート」

使用例

次の例では、`Adheader_typ` 表の `ad_finaltext` 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での例を示します。

- PL/SQL（DBMS_LOB）: LOB データの切捨て（10-220 ページ）
- C（OCI）: LOB データの切捨て（10-221 ページ）
- C++（OCCI）: 今回のリリースでは例は提供されません。
- COBOL（Pro*COBOL）: LOB データの切捨て（10-222 ページ）
- C/C++（Pro*C/C++）: LOB データの切捨て（10-223 ページ）
- Visual Basic（OO4O）: LOB データの切捨て（10-225 ページ）
- Java（JDBC）: LOB データの切捨て（10-225 ページ）

PL/SQL（DBMS_LOB）: LOB データの切捨て

```
/* Trimming LOB data */
/* Example procedure trimLOB_proc is not part of DBMS_LOB package: */
CREATE OR REPLACE PROCEDURE trimLOB_proc IS
   Lob_loc        CLOB;
BEGIN
   /* Select the LOB, get the LOB locator: */
   SELECT pm.Adheader_typ.ad_finaltext INTO Lob_loc FROM Print_media pm
      WHERE pm.product_id = 2056 AND pm.ad_id = 12001 FOR UPDATE;
   /* Opening the LOB is optional: */
   DBMS_LOB.OPEN (Lob_loc, DBMS_LOB.LOB_READWRITE);
   /* Trim the LOB data: */
   DBMS_LOB.TRIM(Lob_loc,100);
   /* Closing the LOB is mandatory if you have opened it: */
   DBMS_LOB.CLOSE (Lob_loc);
COMMIT;
```

```
/* Exception handling: */
EXCEPTION
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('Operation failed');
END;
```

C（OCI）: LOB データの切捨て

```
/* Trimming LOB data
/* Select the locator into a locator variable */
sb4 select_lock_adfinaltext_locator(Lob_loc, errhp, svchp, stmthp)
OCILobLocator *Lob_loc;
OCIError      *errhp;
OCISvcCtx     *svchp;
OCIStmt       *stmthp;
{
  text  *sqlstmt =
      (text *) "SELECT pm.ad_finaltext \
           FROM Print_media pm WHERE pm.product_id = 2268
           AND ad_id = 21001 FOR UPDATE";
  OCIDefine *defnp1;
  checkerr (errhp, OCIStmtPrepare(stmthp, errhp, sqlstmt,
                                  (ub4)strlen((char *)sqlstmt),
                                  (ub4) OCI_NTV_SYNTAX, (ub4) OCI_DEFAULT));
    checkerr (errhp, OCIDefineByPos(stmthp, &defnp1, errhp, (ub4) 1,
                                  (dvoid *)&Lob_loc, (sb4)0,
                                  (ub2) SQLT_CLOB,(dvoid *) 0,
                                  (ub2 *) 0, (ub2 *) 0, (ub4) OCI_DEFAULT));

  /* Execute the select and fetch one row */
  checkerr(errhp, OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                                 (CONST OCISnapshot*) 0, (OCISnapshot*) 0,
                                 (ub4) OCI_DEFAULT));
  return 0;
}
void trimLob(envhp, errhp, svchp, stmthp)
OCIEnv    *envhp;
OCIError  *errhp;
OCISvcCtx *svchp;
OCIStmt   *stmthp;
{
  OCILobLocator *Lob_loc;
  unsigned int trimLength;
  (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &Lob_loc,
                            (ub4)OCI_DTYPE_LOB, (size_t) 0, (dvoid **) 0);
```

```
/* Select the CLOB */
printf( " select an ad_finaltext LOB\n");
select_lock_adfinaltext_locator(Lob_loc, errhp, svchp, stmthp);

/* Open the CLOB */
checkerr (errhp, (OCILobOpen(svchp, errhp, Lob_loc, OCI_LOB_READWRITE)));

/* Trim the LOB to its new length */
trimLength = 100;                       /* <New truncated length of the LOB>*/

printf (" trim the lob to %d bytes\n", trimLength);
checkerr (errhp, OCILobTrim (svchp, errhp, Lob_loc, trimLength ));

/* Closing the CLOB is mandatory if you have opened it */
checkerr (errhp, OCILobClose(svchp, errhp, Lob_loc));

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

COBOL（Pro*COBOL）: LOB データの切捨て

```
* Trimming LOB data
  IDENTIFICATION DIVISION.
  PROGRAM-ID. TRIM-CLOB.
  ENVIRONMENT DIVISION.
  DATA DIVISION.
  WORKING-STORAGE SECTION.

  01  CLOB1          SQL-CLOB.
  01  NEW-LEN        PIC S9(9) COMP.
 * Define the source and destination position and location:
  01  SRC-POS        PIC S9(9) COMP.
  01  DEST-POS       PIC S9(9) COMP.
  01  SRC-LOC        PIC S9(9) COMP.
  01  DEST-LOC       PIC S9(9) COMP.
  01  USERID   PIC X(11) VALUES "SAMP/SAMP".
      EXEC SQL INCLUDE SQLCA END-EXEC.

  PROCEDURE DIVISION.
  TRIM-CLOB.
      EXEC SQL WHENEVER SQLERROR DO PERFORM SQL-ERROR END-EXEC.
      EXEC SQL CONNECT :USERID END-EXEC.

 * Allocate and initialize the CLOB locators:
      EXEC SQL ALLOCATE :CLOB1 END-EXEC.
```

```
           EXEC SQL WHENEVER NOT FOUND GOTO END-OF-CLOB END-EXEC.
           EXEC SQL
                SELECT PM.AD_SOURCETEXT INTO :CLOB1
                FROM PRINT_MEDIA PM
                WHERE PM.PRODUCT_ID = 3060
                AND AD_ID = 11001 FOR UPDATE END-EXEC.

      * Open the CLOB:
           EXEC SQL LOB OPEN :CLOB1 READ WRITE END-EXEC.

      * Move some value to  NEW-LEN:
           MOVE 3 TO NEW-LEN.
           EXEC SQL
                LOB TRIM :CLOB1 TO :NEW-LEN END-EXEC.

           EXEC SQL LOB CLOSE :CLOB1 END-EXEC.
       END-OF-CLOB.
           EXEC SQL WHENEVER NOT FOUND CONTINUE END-EXEC.
           EXEC SQL FREE :CLOB1 END-EXEC.
           EXEC SQL ROLLBACK WORK RELEASE END-EXEC.
           STOP RUN.

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

C/C++（Pro*C/C++）：LOB データの切捨て

```
/* Trimming LOB data  */
#include "pers_trim.h"
#include <stdio.h>
#include <sqlca.h>
void Sample_Error()
{
  EXEC SQL WHENEVER SQLERROR CONTINUE;
  printf("sqlcode = %ld\n", sqlca.sqlcode);
  printf("%.*s\n", sqlca.sqlerrm.sqlerrml, sqlca.sqlerrm.sqlerrmc);
  EXEC SQL ROLLBACK WORK RELEASE;
  exit(1);
}
```

```
void trimLOB_proc()
{
  voiced_typ_ref *vt_ref;
  voiced_typ *vt_typ;
  OCIClobLocator *Lob_loc;
  unsigned int Length, trimLength;

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  EXEC SQL ALLOCATE :Lob_loc;
  EXEC SQL ALLOCATE :vt_ref;
  EXEC SQL ALLOCATE :vt_typ;

  /* Retrieve the REF using Associative SQL */
           EXEC SQL SELECT PMtab.ad_sourctext INTO :vt_ref
           FROM Print_media PMtab
           WHERE PMtab.product_id = 3060 AND ad_id = 11001 FOR UPDATE;

  /* Dereference the Object using the Navigational Interface */
  EXEC SQL OBJECT DEREF :vt_ref INTO :vt_typ FOR UPDATE;
  Lob_loc = vt_typ->script;

  /* Opening the LOB is Optional */
  EXEC SQL LOB OPEN :Lob_loc READ WRITE;
  EXEC SQL LOB DESCRIBE :Lob_loc GET LENGTH INTO :Length;
  printf("Old length was %d\n", Length);
  trimLength = (unsigned int)(Length / 2);

  /* Trim the LOB to its new length */
  EXEC SQL LOB TRIM :Lob_loc TO :trimLength;

  /* Closing the LOB is mandatory if it has been opened */
  EXEC SQL LOB CLOSE :Lob_loc;

  /* Mark the Object as Modified (Dirty) */
  EXEC SQL OBJECT UPDATE :vt_typ;

  /* Flush the changes to the LOB in the Object Cache */
  EXEC SQL OBJECT FLUSH :vt_typ;

  /* Display the new (modified) length */
  EXEC SQL SELECT Mtab.Voiced_ref.Script INTO :Lob_loc
           FROM Multimedia_tab Mtab WHERE Mtab.Clip_ID = 2;
  EXEC SQL LOB DESCRIBE :Lob_loc GET LENGTH INTO :Length;
  printf("New length is now %d\n", Length);

  /* Free the Objects and the LOB Locator */
  EXEC SQL FREE :vt_ref;
```

```
  EXEC SQL FREE :vt_typ;
  EXEC SQL FREE :Lob_loc;
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  trimLOB_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

Visual Basic（OO4O）: LOB データの切捨て

```
'Trimming LOB data
Dim MySession As OraSession
Dim OraDb As OraDatabase
Dim OraDyn As OraDynaset, OraAdPhoto1 As OraBlob, OraAdPhotoClone As OraBlob

Set MySession = CreateObject("OracleInProcServer.XOraSession")
Set OraDb = MySession.OpenDatabase("exampledb", "samp/samp", 0&)
Set OraDyn = OraDb.CreateDynaset(
   "SELECT * FROM Print_media ORDER BY product_id, ad_id", ORADYN_DEFAULT)
Set OraAdPhoto1 = OraDyn.Fields("ad_photo").Value

OraDyn.Edit
OraAdPhoto1.Trim 10
OraDyn.Update
```

Java（JDBC）: LOB データの切捨て

```
// Trimming BLOBs and CLOBs.
// You need to import the java.sql package to use JDBC
import java.sql.*;

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

class TrimLob
{
  public static void main (String args [])
       throws SQLException
  {
    // Load the Oracle JDBC driver
```

```
    DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

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

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

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

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

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

    // Populate the table
    stmt.execute ("insert into basic_lob_table values ('one',
'0101010101010101010101010101', 'onetwothreefour')");

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

      // Show the original lob length
```

```
      System.out.println ("Open the lobs");
      System.out.println ("blob.length()="+blob.length());
      System.out.println ("clob.length()="+clob.length());

      // Trim the lobs
      System.out.println ("Trim the lob to length = 6");
      blob.trim (6);
      clob.trim (6);

      // Show the lob length after trim()
      System.out.println ("Open the lobs");
      System.out.println ("blob.length()="+blob.length());
      System.out.println ("clob.length()="+clob.length());
    }

    // Close the ResultSet
    rset.close ();

    // Close the Statement
    stmt.close ();

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

次の例は、以前の方法です。DBMS_LOB.trim を使用して LOB データを切り捨てます。

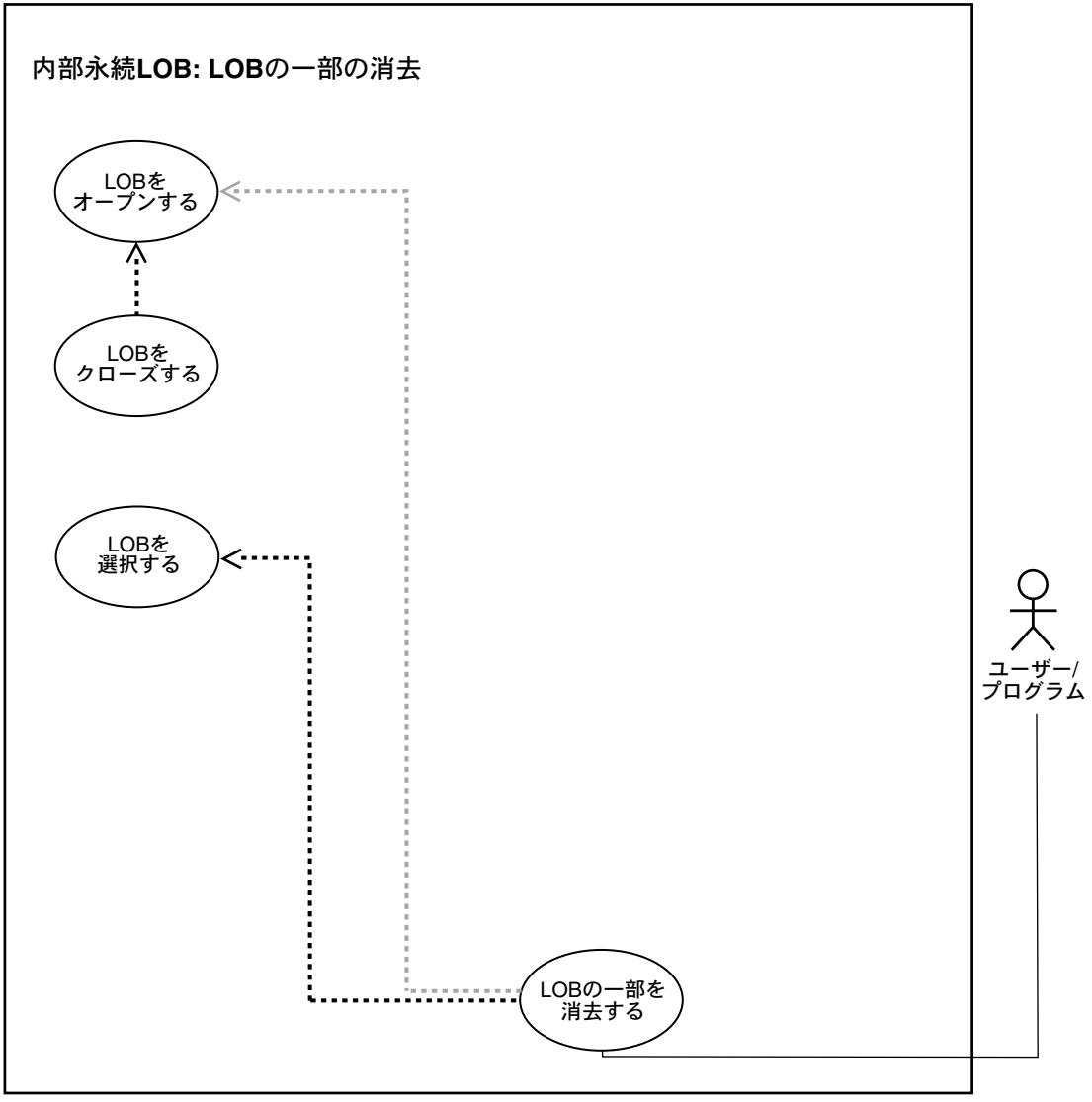
```
// Trimming LOB data
// Java IO classes:
import java.io.InputStream;
import java.io.OutputStream;

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

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

public class Ex2_141
```

```
{
  static final int MAXBUFSIZE = 32767;
  public static void main (String args [])
       throws Exception
  {
    // Load the Oracle JDBC driver:
    DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

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

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

    // Create a Statement:
    Statement stmt = conn.createStatement ();
    try
    {
     CLOB lob_loc = null;

     ResultSet rset = stmt.executeQuery (
        "SELECT pm.ad_finaltext FROM Print_media pm
            WHERE pm.product_id = 2056 AND ad_id = 12001 FOR UPDATE");
   if (rset.next())
   {
     lob_loc = ((OracleResultSet)rset).getCLOB (1);
   }

   // Open the LOB for READWRITE:
       OracleCallableStatement cstmt = (OracleCallableStatement)
          conn.prepareCall ("BEGIN DBMS_LOB.OPEN(?,DBMS_LOB.LOB_READWRITE);
          END;");
   cstmt.setCLOB(1, lob_loc);
   cstmt.execute();

   // Trim the LOB to length of 400:
   cstmt = (OracleCallableStatement)
      conn.prepareCall ("BEGIN DBMS_LOB.TRIM(?, 400); END;");
   cstmt.setCLOB(1, lob_loc);
   cstmt.execute();

   // Close the LOB:
   cstmt = (OracleCallableStatement) conn.prepareCall (
      "BEGIN DBMS_LOB.CLOSE(?); END;");
   cstmt.setCLOB(1, lob_loc);
   cstmt.execute();
```

```
   stmt.close();
   cstmt.close();
   conn.commit();
   conn.close();
    }
    catch (SQLException e)
    {
       e.printStackTrace();
    }
  }
}
```

LOB の一部の消去

図 10-31 利用図 : LOB の一部の消去



参照： 10-2 ページの表 10-1「利用モデル : 内部永続 LOB」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、5-5 ページの「更新済ロケータを介した LOB の更新」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関数またはファンクションのリストは、第 3 章「様々なプログラム環境での LOB のサポート」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）：『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』の第 23 章「DBMS_LOB」の「DBMS_LOB サブプログラムの要約」の「ERASE プロシージャ」
- C（OCI）：『Oracle Call Interface プログラマーズ・ガイド』の第 16 章「その他の OCI リレーショナル関数」の「LOB 関数」の OCILobErase()
- C++（OCCI）：『Oracle C++ Call Interface プログラマーズ・ガイド』
- COBOL（Pro*COBOL）：『Pro*COBOL Precompiler プログラマーズ・ガイド』の LOB に関する詳細、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・ディレクティブ」の「LOB ERASE（実行可能埋込み SQL 拡張機能）」
- C/C++（Pro*C/C++）：『Pro*C/C++ Precompiler プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびディレクティブ」の「LOB ERASE（実行可能埋込み SQL 拡張機能）」
- Visual Basic（OO4O オンライン・ヘルプ）：ヘルプの「目次」タブから、「OO4O オートメーション・サーバー・リファレンス」>「OO4O サーバーのオブジェクト」>「OraBLOB、OraCLOB」>「メソッド」>「erase」を選択

- Java（JDBC）：『Oracle9*i* JDBC 開発者ガイドおよびリファレンス』の第 8 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『Oracle9*i* 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「JDBC 2.0 LOB 型と Oracle 拡張型のサポート」の「BLOB、CLOB および BFILE のサポート」

使用例

次の例では、イメージ（ad_photo）の一部を消去します。

例

次のプログラム環境での例を示します。

- PL/SQL（DBMS_LOB）: LOB の一部の消去（10-232 ページ）
- C（OCI）: LOB の一部の消去（10-233 ページ）
- C++（OCCI）: 今回のリリースでは例は提供されません。
- COBOL（Pro*COBOL）: LOB の一部の消去（10-234 ページ）
- C/C++（Pro*C/C++）: LOB の一部の消去（10-236 ページ）
- Visual Basic（OO4O）: LOB の一部の消去（10-237 ページ）
- Java（JDBC）: LOB の一部の消去（10-237 ページ）

PL/SQL（DBMS_LOB）: LOB の一部の消去

```
/* Erasing part of a LOB.
   Example procedure eraseLOB_proc is not part of DBMS_LOB package: */
CREATE OR REPLACE PROCEDURE eraseLOB_proc IS
   Lob_loc        BLOB;
   Amount         INTEGER := 3000;
BEGIN
   /* Select the LOB, get the LOB locator: */
   SELECT ad_photo INTO lob_loc FROM Print_media
      WHERE product_id = 3106 AND ad_id = 13001 FOR UPDATE;
   /* Opening the LOB is optional: */
   DBMS_LOB.OPEN (Lob_loc, DBMS_LOB.LOB_READWRITE);
   /* Erase the data: */
   DBMS_LOB.ERASE(Lob_loc, Amount, 2000);
   /* Closing the LOB is mandatory if you have opened it: */
   DBMS_LOB.CLOSE (Lob_loc);
COMMIT;
```

```
/* Exception handling: */
EXCEPTION
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('Operation failed');
END;
```

C（OCI）: LOB の一部の消去

```
/* Erasing part of a LOB (persistent LOBs)
   Select the locator into a locator variable: */
sb4 select_lock_adphoto_locator(Lob_loc, errhp, svchp, stmthp)
OCILobLocator *Lob_loc;
OCIError      *errhp;
OCISvcCtx     *svchp;
OCIStmt       *stmthp;
{
  text  *sqlstmt =
     (text *)"SELECT ad_photo FROM Print_media
              WHERE product_id=3060 AND ad_id = 11001 FOR UPDATE";
  OCIDefine *defnp1, *defnp2;

  checkerr (errhp, 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)
           || OCIDefineByPos(stmthp, &defnp2, errhp, (ub4) 2,
                                  (dvoid *)&Lob_loc, (sb4)0,
                                  (ub2) SQLT_BLOB,(dvoid *) 0,
                                  (ub2 *) 0, (ub2 *) 0, (ub4) OCI_DEFAULT)
                                  );

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

void eraseLob(envhp, errhp, svchp, stmthp)
OCIEnv    *envhp;
```

```
OCIError  *errhp;
OCISvcCtx *svchp;
OCIStmt   *stmthp;
{
  OCILobLocator *Lob_loc;
  ub4 amount = 3000;
  ub4 offset = 2000;

  OCILobLocator *Lob_Loc;

  (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &Lob_loc,
                            (ub4)OCI_DTYPE_LOB, (size_t) 0, (dvoid **) 0);

  /* Select the CLOB: */
  printf( " select and lock an ad_photo LOB\n");
  select_lock_adphoto_locator(Lob_loc, errhp, svchp, stmthp);

  /* Open the BLOB: */
  checkerr (errhp, (OCILobOpen(svchp, errhp, Lob_loc, OCI_LOB_READWRITE)));

  /* Erase the data starting at the specified Offset: */
  printf(" erase %d bytes from the ad_photo Lob\n", amount);
  checkerr (errhp, OCILobErase (svchp, errhp, Lob_loc, &amount, offset ));

  /* Closing the BLOB is mandatory if you have opened it: */
  checkerr (errhp, OCILobClose(svchp, errhp, Lob_loc));

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

COBOL（Pro*COBOL）: LOB の一部の消去

```
* ERASING PART OF A LOB
  IDENTIFICATION DIVISION.
  PROGRAM-ID. ERASE-BLOB.
  ENVIRONMENT DIVISION.
  DATA DIVISION.
  WORKING-STORAGE SECTION.

  01  USERID   PIC X(11) VALUES "SAMP/SAMP".
  01  BLOB1          SQL-BLOB.
  01  AMT            PIC S9(9) COMP.
  01  OFFSET         PIC S9(9) COMP.
      EXEC SQL INCLUDE SQLCA END-EXEC.
```

```
 PROCEDURE DIVISION.
 ERASE-BLOB.

     EXEC SQL WHENEVER SQLERROR DO PERFORM SQL-ERROR END-EXEC.
     EXEC SQL
          CONNECT :USERID
     END-EXEC.
* Allocate and initialize the BLOB locators:
     EXEC SQL ALLOCATE :BLOB1 END-EXEC.
     EXEC SQL WHENEVER NOT FOUND GOTO END-OF-BLOB END-EXEC.
     EXEC SQL
          SELECT AD_PHOTO INTO :BLOB1
          FROM PRINT_MEDIA PM
          WHERE PM.PRODUCT_ID = 2268 AND AD_ID = 21001 FOR UPDATE
     END-EXEC.

* Open the BLOB:
     EXEC SQL LOB OPEN :BLOB1 READ WRITE END-EXEC.

* Move some value to AMT and OFFSET:
     MOVE 2 TO AMT.
     MOVE 1 TO OFFSET.
     EXEC SQL
          LOB ERASE :AMT FROM :BLOB1 AT :OFFSET END-EXEC.
     EXEC SQL LOB CLOSE :BLOB1 END-EXEC.
 END-OF-BLOB.
     EXEC SQL WHENEVER NOT FOUND CONTINUE END-EXEC.
     EXEC SQL FREE :BLOB1 END-EXEC.
     EXEC SQL ROLLBACK WORK RELEASE END-EXEC.
     STOP RUN.

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

C/C++（Pro*C/C++）: LOB の一部の消去

```
/* Erasing part of a LOB */
#include <oci.h>
#include <stdio.h>
#include <sqlca.h>

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

void eraseLob_proc()
{
  OCIBlobLocator *Lob_loc;
  int Amount = 5;
  int Offset = 5;

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  EXEC SQL ALLOCATE :Lob_loc;
  EXEC SQL SELECT ad_composite INTO :Lob_loc
           FROM Print_media WHERE product_id = 3060 AND ad_id = 11001 FOR UPDATE;
  /* Opening the LOB is Optional: */
  EXEC SQL LOB OPEN :Lob_loc READ WRITE;
  /* Erase the data starting at the specified Offset: */
  EXEC SQL LOB ERASE :Amount FROM :Lob_loc AT :Offset;
  printf("Erased %d bytes\n", Amount);
  /* Closing the LOB is mandatory if it has been opened: */
  EXEC SQL LOB CLOSE :Lob_loc;
  EXEC SQL FREE :Lob_loc;
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  eraseLob_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

Visual Basic（OO4O）: LOB の一部の消去

```
'Erasing part of a LOB
Dim MySession As OraSession
Dim OraDb As OraDatabase
Dim OraDyn As OraDynaset, OraAdPhoto1 As OraBlob, OraAdPhotoClone As OraBlob

Set MySession = CreateObject("OracleInProcServer.XOraSession")
Set OraDb = MySession.OpenDatabase("exampledb", "samp/samp", 0&)

Set OraDyn = OraDb.CreateDynaset("SELECT * FROM Print_media ORDER BY product_
id", ORADYN_DEFAULT)
Set OraAdPhoto1 = OraDyn.Fields("ad_photo").Value
'Erase 10 bytes begining from the 100th byte:
OraDyn.Edit
OraAdPhoto1.Erase 10, 100
OraDyn.Update
```

Java（JDBC）: LOB の一部の消去

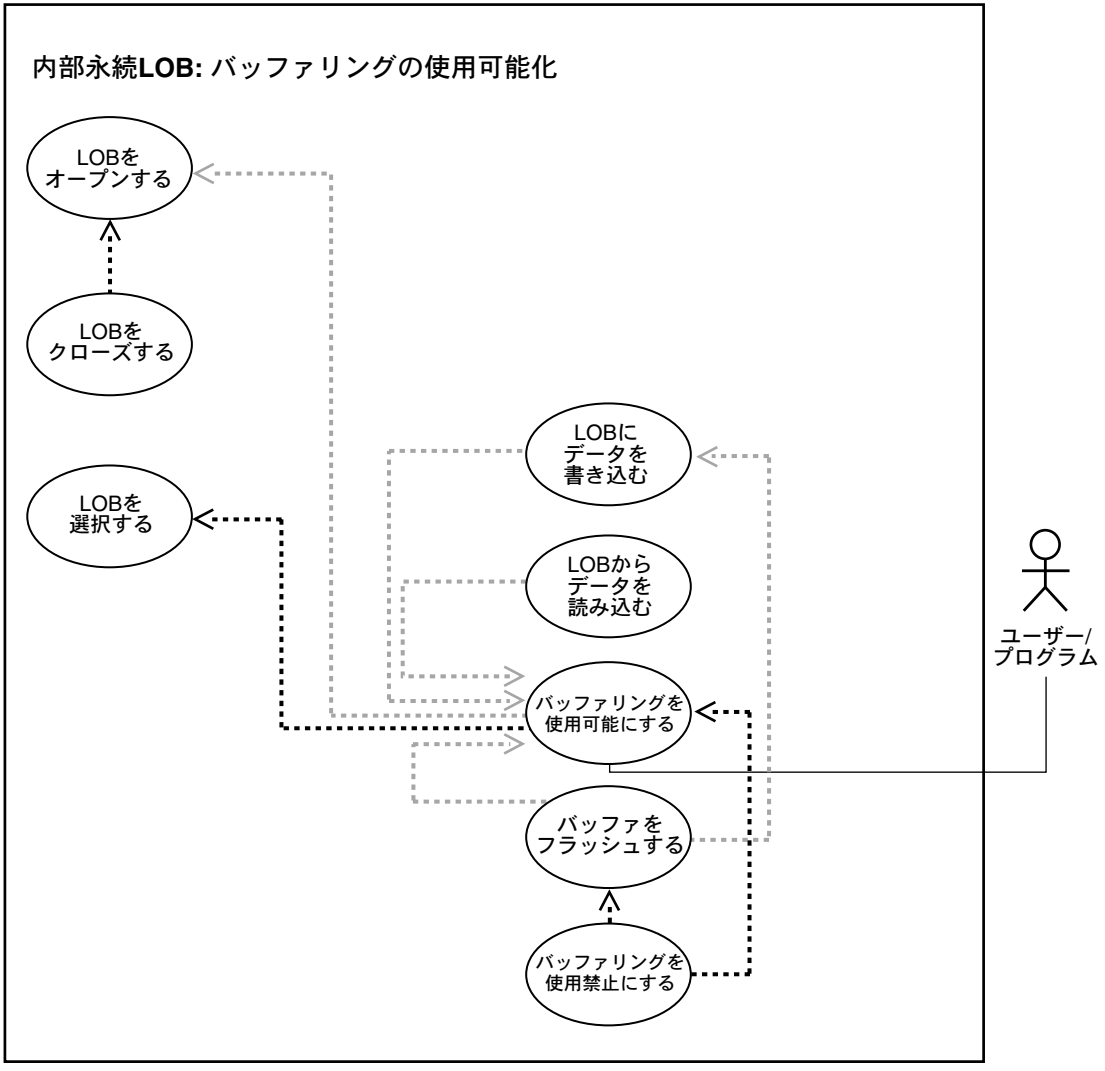
```
// Erasing part of a LOB
import java.io.InputStream;
import java.io.OutputStream;

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

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

public class Ex2_145
{
  static final int MAXBUFSIZE = 32767;
  public static void main (String args [])
       throws Exception
  {
    // Load the Oracle JDBC driver:
    DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
```

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

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

 // Create a Statement:
 Statement stmt = conn.createStatement ();
 try
 {
  BLOB lob_loc = null;
  int eraseAmount = 30;
  ResultSet rset = stmt.executeQuery (
     "SELECT ad_photo FROM Print_media
      WHERE product_id = 2056 AND ad_id = 12001 FOR UPDATE");
if (rset.next())
{
  lob_loc = ((OracleResultSet)rset).getBLOB (1);
}

// Open the LOB for READWRITE:
    OracleCallableStatement cstmt = (OracleCallableStatement)
       conn.prepareCall ("BEGIN DBMS_LOB.OPEN(?, "
                         +"DBMS_LOB.LOB_READWRITE); END;");
cstmt.setBLOB(1, lob_loc);
cstmt.execute();

// Erase eraseAmount bytes starting at offset 2000:
cstmt = (OracleCallableStatement)
   conn.prepareCall ("BEGIN DBMS_LOB.ERASE(?, ?, 1); END;");
cstmt.registerOutParameter (1, OracleTypes.BLOB);
cstmt.registerOutParameter (2, Types.INTEGER);
cstmt.setBLOB(1, lob_loc);
cstmt.setInt(2, eraseAmount);
cstmt.execute();
lob_loc = cstmt.getBLOB(1);
eraseAmount = cstmt.getInt(2);

// Close the LOB:
cstmt = (OracleCallableStatement) conn.prepareCall (
   "BEGIN DBMS_LOB.CLOSE(?); END;");
cstmt.setBLOB(1, lob_loc);
cstmt.execute();
```

```
    conn.commit();
    stmt.close();
    cstmt.close();
    conn.commit();
    conn.close();

     }
     catch (SQLException e)
     {
        e.printStackTrace();
     }
   }
}
```

LOB バッファリングの使用可能化

図 10-32 利用図 : LOB バッファリングの使用可能化



参照 : 10-2 ページの表 10-1「利用モデル : 内部永続 LOB」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

少量のデータの読込みおよび書込みを実行する場合は、バッファリングを使用可能にします。これらの作業の完了後、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。
- 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。

LOB バッファリング・サブシステムの詳細は、5-18 ページの「LOB バッファリング・サブシステム（LBS）」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関数またはファンクションのリストは、第 3 章「様々なプログラム環境での LOB のサポート」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）:『Oracle Call Interface プログラマーズ・ガイド』の第 16 章「その他の OCI リレーショナル関数」の「LOB 関数」の OCILobEnableBuffering()、OCILobDisableBuffering() および OCILobFlushBuffer()
- C++（OCCI）:『Oracle C++ Call Interface プログラマーズ・ガイド』
- COBOL（Pro*COBOL）:『Pro*COBOL Precompiler プログラマーズ・ガイド』の LOB に関する詳細、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・ディレクティブ」の「LOB ENABLE BUFFERING（実行可能埋込み SQL 拡張機能）」
- C/C++（Pro*C/C++）:『Pro*C/C++ Precompiler プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびディレクティブ」の「LOB ENABLE BUFFERING（実行可能埋込み SQL 拡張機能）」
- Visual Basic（OO4O オンライン・ヘルプ）: ヘルプの「目次」タブから、「OO4O オートメーション・サーバー・リファレンス」>「OO4O サーバーのオブジェクト」>「OraBLOB、OraCLOB」>「メソッド」>「EnableBuffering」を選択
- 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作成された ad_photo についてのバッファリング管理の一部です。

例

次のプログラム環境での例を示します。

- PL/SQL（DBMS_LOB）: 今回のリリースでは例は提供されません。
- C（OCI）: 今回のリリースでは例は提供されません。
- C++（OCCI）: 今回のリリースでは例は提供されません。
- COBOL（Pro*COBOL）: LOB の一部の消去（10-234 ページ）
- C/C++（Pro*C/C++）: LOB の一部の消去（10-236 ページ）
- Visual Basic（OO4O）: LOB の一部の消去（10-237 ページ）

C（OCI）: LOB バッファリングの使用可能化

参照： 10-252 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

```
* ENABLING LOB BUFFERING
  IDENTIFICATION DIVISION.
  PROGRAM-ID. LOB-BUFFERING.
  ENVIRONMENT DIVISION.
  DATA DIVISION.
  WORKING-STORAGE SECTION.
  01  USERID   PIC X(11) VALUES "SAMP/SAMP".
  01  BLOB1          SQL-BLOB.
  01  BUFFER         PIC X(10).
  01  AMT            PIC S9(9) COMP.
      EXEC SQL VAR BUFFER IS RAW(10) END-EXEC.
      EXEC SQL INCLUDE SQLCA END-EXEC.

  PROCEDURE DIVISION.
  LOB-BUFFERING.

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

```
* Allocate and initialize the BLOB locator:
     EXEC SQL ALLOCATE :BLOB1 END-EXEC.
     EXEC SQL WHENEVER NOT FOUND GOTO END-OF-BLOB END-EXEC.
     EXEC SQL
          SELECT ad_photo INTO :BLOB1
          FROM PRINT_MEDIA
          WHERE PRODUCT_ID = 3060 AND AD_ID = 11001
          FOR UPDATE END-EXEC.

* Open the BLOB and enable buffering:
     EXEC SQL LOB OPEN :BLOB1 READ WRITE END-EXEC.
     EXEC SQL
          LOB ENABLE BUFFERING :BLOB1 END-EXEC.

* Write some data to the BLOB:
     MOVE "242424" TO BUFFER.
     MOVE 3 TO AMT.
     EXEC SQL
          LOB WRITE :AMT FROM :BUFFER INTO :BLOB1 END-EXEC.

     MOVE "212121" TO BUFFER.
     MOVE 3 TO AMT.
     EXEC SQL
          LOB WRITE :AMT FROM :BUFFER INTO :BLOB1 END-EXEC.

* Now flush the buffered writes:
     EXEC SQL LOB FLUSH BUFFER :BLOB1 END-EXEC.
     EXEC SQL LOB DISABLE BUFFERING :BLOB1 END-EXEC.
     EXEC SQL LOB CLOSE :BLOB1 END-EXEC.

 END-OF-BLOB.
     EXEC SQL WHENEVER NOT FOUND CONTINUE END-EXEC.
     EXEC SQL FREE :BLOB1 END-EXEC.
     EXEC SQL ROLLBACK WORK RELEASE END-EXEC.
     STOP RUN.

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

C/C++（Pro*C/C++）: LOB バッファリングの使用可能化

```
/* Enabling LOB buffering
#include <oci.h>
#include <stdio.h>
#include <string.h>
#include <sqlca.h>

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

#define BufferLength 256

void enableBufferingLOB_proc()
{
  OCIBlobLocator *Lob_loc;
  int Amount = BufferLength;
  int multiple, Position = 1;
  /* Datatype equivalencing is mandatory for this datatype: */
  char Buffer[BufferLength];
  EXEC SQL VAR Buffer is RAW(BufferLength);

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  /* Allocate and Initialize the LOB: */
  EXEC SQL ALLOCATE :Lob_loc;
  EXEC SQL SELECT ad_composite INTO :Lob_loc
             FROM Print_media
             WHERE product_id = 3060 AND ad_id = 11001 FOR UPDATE;

  /* Enable use of the LOB Buffering Subsystem: */
  EXEC SQL LOB ENABLE BUFFERING :Lob_loc;
  memset((void *)Buffer, 0, BufferLength);
  for (multiple = 0; multiple < 8; multiple++)
    {
      /* Write data to the LOB: */
      EXEC SQL LOB WRITE ONE :Amount
                    FROM :Buffer INTO :Lob_loc AT :Position;
      Position += BufferLength;
    }
```

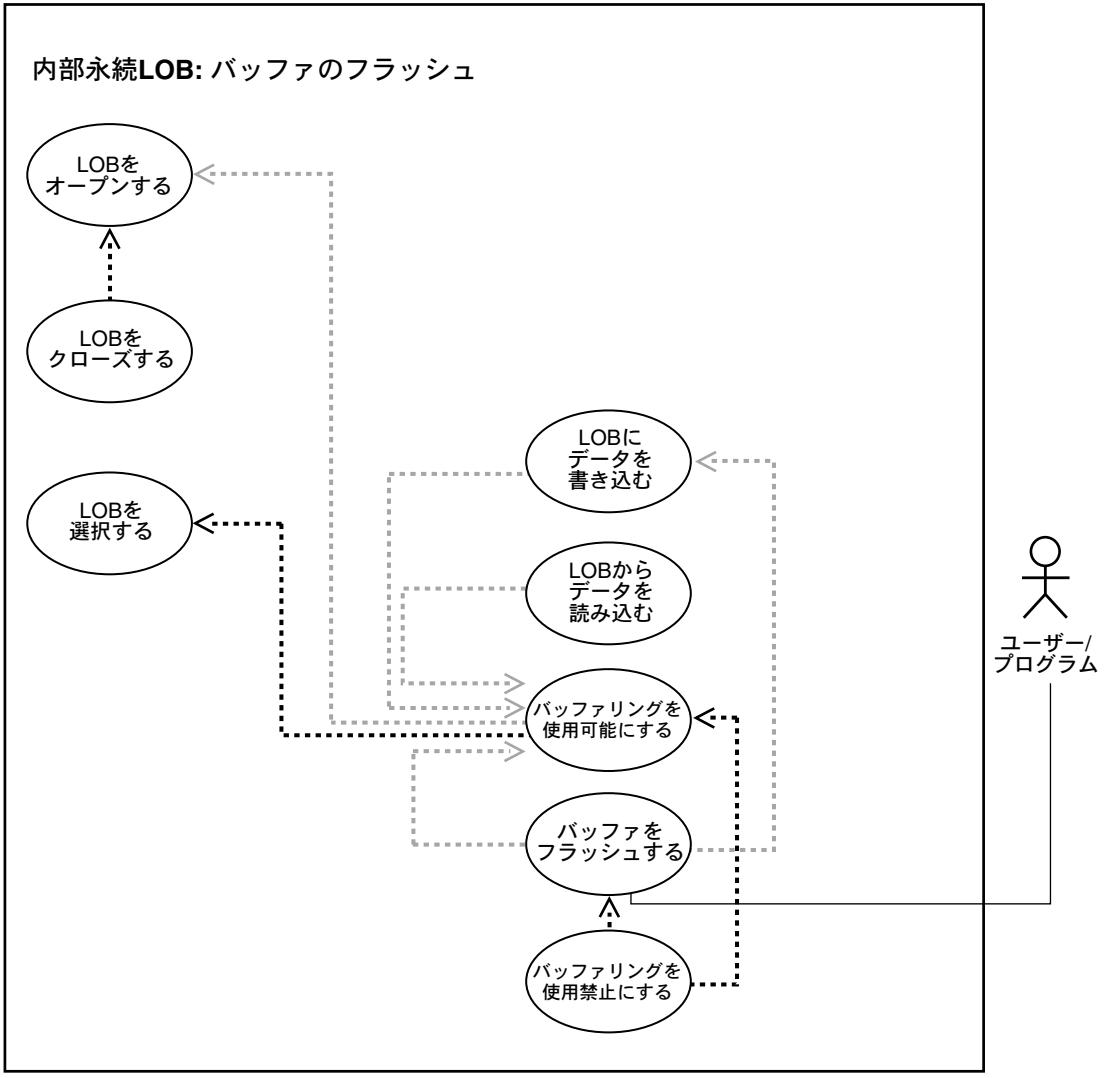
```
  /* Flush the contents of the buffers and Free their resources: */
  EXEC SQL LOB FLUSH BUFFER :Lob_loc FREE;
  /* Turn off use of the LOB Buffering Subsystem: */
  EXEC SQL LOB DISABLE BUFFERING :Lob_loc;
  /* Release resources held by the Locator: */
  EXEC SQL FREE :Lob_loc;
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  enableBufferingLOB_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

Visual Basic（OO4O）: LOB バッファリングの使用可能化

```
'Enabling LOB buffering (persistent LOBs)
Dim MySession As OraSession
Dim OraDb As OraDatabase
Dim OraDyn As OraDynaset, OraAdPhoto1 As OraBlob, OraAdPhotoClone As OraBlob

Set MySession = CreateObject("OracleInProcServer.XOraSession")
Set OraDb = MySession.OpenDatabase("exampledb", "samp/samp", 0&)
Set OraDyn = OraDb.CreateDynaset(
   "SELECT * FROM Print_media ORDER BY product_id", ORADYN_DEFAULT)
Set OraAdPhoto1 = OraDyn.Fields("ad_photo").Value

'Enable buffering:
OraAdPhoto1.EnableBuffering
```

バッファのフラッシュ

図 10-33 利用図 : バッファのフラッシュ



参照： 10-2 ページの表 10-1「利用モデル : 内部永続 LOB」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

少量のデータの読込みおよび書込みを実行する場合は、バッファリングを使用可能にします。これらの作業の完了後、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。
- 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込みおよび書込みを実行する場合は、バッファリングを使用可能にしないでください。

LOB バッファリング・サブシステムの詳細は、5-18 ページの「LOB バッファリング・サブシステム（LBS）」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関数またはファンクションのリストは、第 3 章「様々なプログラム環境での LOB のサポート」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）:『Oracle Call Interface プログラマーズ・ガイド』の第 16 章「その他の OCI リレーショナル関数」の「LOB 関数」の OCILobEnableBuffering()、OCILobDisableBuffering() および OCILobFlushBuffer()
- C++（OCCI）:『Oracle C++ Call Interface プログラマーズ・ガイド』
- COBOL（Pro*COBOL）:『Pro*COBOL Precompiler プログラマーズ・ガイド』の LOB に関する詳細、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・ディレクティブ」の「LOB FLUSH BUFFER（実行可能埋込み SQL 拡張機能）」
- C/C++（Pro*C/C++）:『Pro*C/C++ Precompiler プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびディレクティブ」の「LOB FLUSH BUFFER（実行可能埋込み SQL 拡張機能）」

- Visual Basic（OO4O オンライン・ヘルプ）：ヘルプの「目次」タブから、「OO4O オートメーション・サーバー・リファレンス」>「OO4O サーバーのオブジェクト」>「OraBLOB、OraCLOB」>「メソッド」>「FlushBuffer」を選択
- 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作成された ad_photo についてのバッファリング管理の一部です。

例

次のプログラム環境での例を示します。

- PL/SQL（DBMS_LOB）：今回のリリースでは例は提供されません。
- C（OCI）：バッファのフラッシュ（10-248 ページ）
- C++（OCCI）：今回のリリースでは例は提供されません。
- COBOL（Pro*COBOL）：バッファのフラッシュ（10-248 ページ）
- C/C++（Pro*C/C++）：バッファのフラッシュ（10-250 ページ）
- Visual Basic（OO4O）：今回のリリースでは例は提供されません。
- Java（JDBC）：今回のリリースでは例は提供されません。

C（OCI）：バッファのフラッシュ

参照： 10-252 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

```
* Flushing the LOB buffer (persistent LOBs)
  IDENTIFICATION DIVISION.
  PROGRAM-ID. LOB-BUFFERING.
  ENVIRONMENT DIVISION.
  DATA DIVISION.
  WORKING-STORAGE SECTION.

  01  USERID   PIC X(11) VALUES "SAMP/SAMP".
  01  BLOB1          SQL-BLOB.
  01  BUFFER         PIC X(10).
  01  AMT            PIC S9(9) COMP.
      EXEC SQL VAR BUFFER IS RAW(10) END-EXEC.
      EXEC SQL INCLUDE SQLCA END-EXEC.
```

```
 PROCEDURE DIVISION.
 LOB-BUFFERING.

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

* Allocate and initialize the BLOB locator:
     EXEC SQL ALLOCATE :BLOB1 END-EXEC.
     EXEC SQL WHENEVER NOT FOUND GOTO END-OF-BLOB END-EXEC.
     EXEC SQL
          SELECT AD_PHOTO INTO :BLOB1
          FROM PRINT_MEDIA
          WHERE PRODUCT_ID = 2056 AND AD_ID = 12001 FOR UPDATE
     END-EXEC.

* Open the BLOB and enable buffering:
     EXEC SQL LOB OPEN :BLOB1 READ WRITE END-EXEC.
     EXEC SQL LOB ENABLE BUFFERING :BLOB1 END-EXEC.

* Write some data to the BLOB:
     MOVE "242424" TO BUFFER.
     MOVE 3 TO AMT.
     EXEC SQL
          LOB WRITE :AMT FROM :BUFFER INTO :BLOB1
     END-EXEC.

     MOVE "212121" TO BUFFER.
     MOVE 3 TO AMT.
     EXEC SQL
          LOB WRITE :AMT FROM :BUFFER INTO :BLOB1
     END-EXEC.

* Now flush the buffered writes:
     EXEC SQL LOB FLUSH BUFFER :BLOB1 END-EXEC.
     EXEC SQL LOB DISABLE BUFFERING :BLOB1 END-EXEC.
     EXEC SQL LOB CLOSE :BLOB1 END-EXEC.
 END-OF-BLOB.
     EXEC SQL WHENEVER NOT FOUND CONTINUE END-EXEC.
     EXEC SQL FREE :BLOB1 END-EXEC.
     EXEC SQL ROLLBACK WORK RELEASE END-EXEC.
     STOP RUN.

 SQL-ERROR.
     EXEC SQL WHENEVER SQLERROR CONTINUE END-EXEC.
```

```
            DISPLAY " ".
            DISPLAY "ORACLE ERROR DETECTED:".
            DISPLAY " ".
            DISPLAY SQLERRMC.
            EXEC SQL ROLLBACK WORK RELEASE END-EXEC.
            STOP RUN.
```

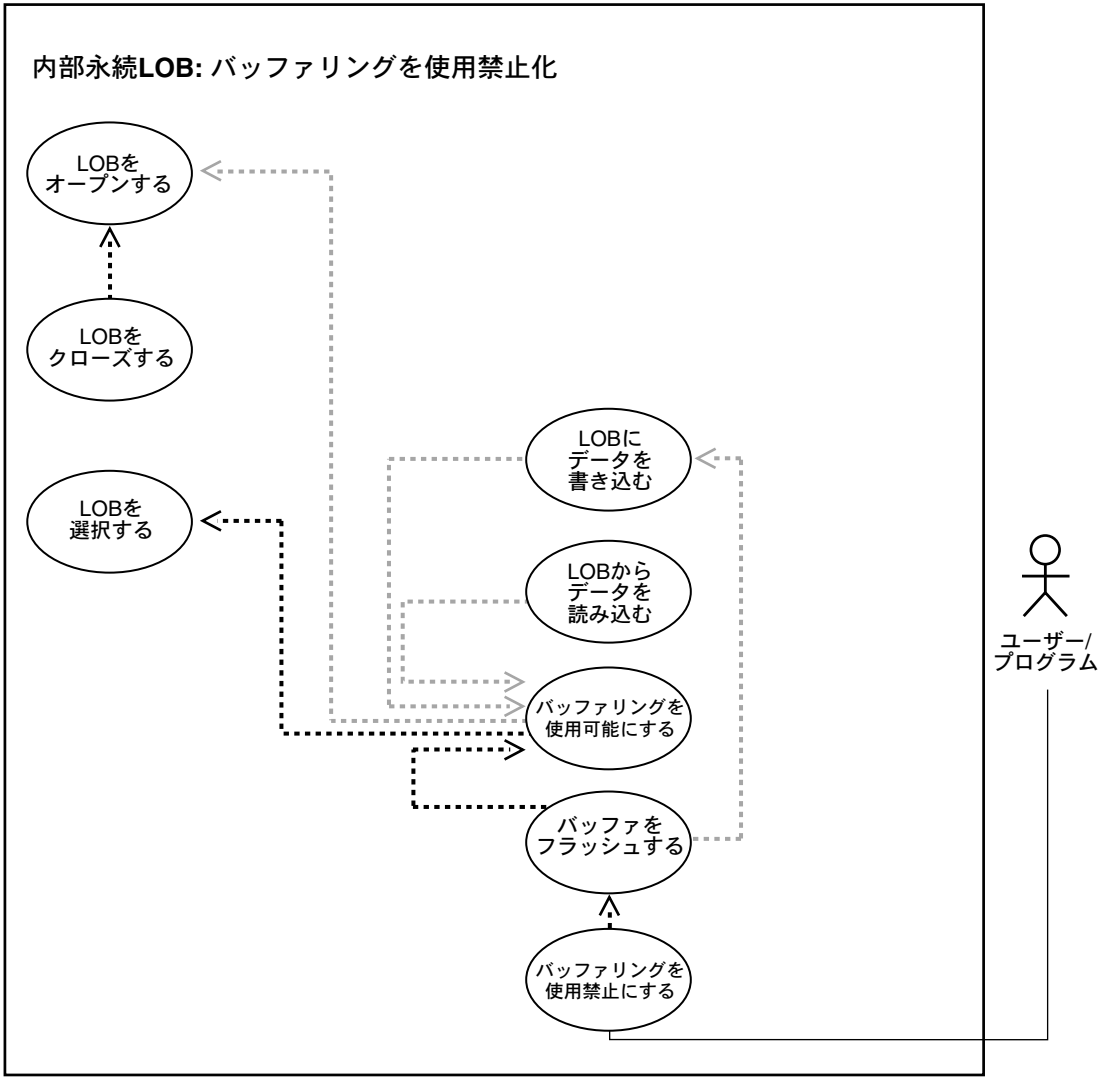
C/C++（Pro*C/C++）：バッファのフラッシュ

```
/* Flushing the LOB Buffer (persistent LOBs)
#include <oci.h>
#include <stdio.h>
#include <string.h>
#include <sqlca.h>

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

#define BufferLength 256

void flushBufferingLOB_proc()
{
  OCIBlobLocator *Lob_loc;
  int Amount = BufferLength;
  int multiple, Position = 1;

  /* Datatype equivalencing is mandatory for this datatype: */
  char Buffer[BufferLength];
  EXEC SQL VAR Buffer is RAW(BufferLength);
  EXEC SQL WHENEVER SQLERROR DO Sample_Error();

  /* Allocate and Initialize the LOB: */
  EXEC SQL ALLOCATE :Lob_loc;
  EXEC SQL SELECT Sound INTO :Lob_loc
             FROM Multimedia_tab WHERE Clip_ID = 1 FOR UPDATE;

  /* Enable use of the LOB Buffering Subsystem: */
  EXEC SQL LOB ENABLE BUFFERING :Lob_loc;
  memset((void *)Buffer, 0, BufferLength);
  for (multiple = 0; multiple < 8; multiple++)
    {
```

```
      /* Write data to the LOB: */
      EXEC SQL LOB WRITE ONE :Amount
                    FROM :Buffer INTO :Lob_loc AT :Position;
      Position += BufferLength;
    }
  /* Flush the contents of the buffers and Free their resources: */
  EXEC SQL LOB FLUSH BUFFER :Lob_loc FREE;
  /* Turn off use of the LOB Buffering Subsystem: */
  EXEC SQL LOB DISABLE BUFFERING :Lob_loc;
  /* Release resources held by the Locator: */
  EXEC SQL FREE :Lob_loc;
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  flushBufferingLOB_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

LOB バッファリングの使用禁止化

図 10-34 利用図 : LOB バッファリングの使用禁止化



参照： 10-2 ページの表 10-1「利用モデル : 内部永続 LOB」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

少量のデータの読込みおよび書込みを実行する場合は、バッファリングを使用可能にします。これらの作業の完了後、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。
- 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込みおよび書込みを実行する場合は、バッファリングを使用可能にしないでください。

LOB バッファリング・サブシステムの詳細は、5-18 ページの「LOB バッファリング・サブシステム（LBS）」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関数またはファンクションのリストは、第 3 章「様々なプログラム環境での LOB のサポート」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）:『Oracle Call Interface プログラマーズ・ガイド』の第 16 章「その他の OCI リレーショナル関数」の「LOB 関数」の OCILobEnableBuffering()、OCILobDisableBuffering() および OCILobFlushBuffer()
- C++（OCCI）:『Oracle C++ Call Interface プログラマーズ・ガイド』
- COBOL（Pro*COBOL）:『Pro*COBOL Precompiler プログラマーズ・ガイド』の LOB に関する詳細、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・ディレクティブ」の「LOB DISABLE BUFFERING（実行可能埋込み SQL 拡張機能）」
- C/C++（Pro*C/C++）:『Pro*C/C++ Precompiler プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびディレクティブ」の「LOB DISABLE BUFFERING（実行可能埋込み SQL 拡張機能）」

- Visual Basic（OO4O オンライン・ヘルプ）: ヘルプの「目次」タブから、「OO4O オートメーション・サーバー・リファレンス」>「OO4O サーバーのオブジェクト」>「OraBLOB、OraCLOB」>「メソッド」>「DisableBuffering」を選択
- 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作成された ad_photo についてのバッファリング管理の一部です。

例

次のプログラム環境での例を示します。

- PL/SQL（DBMS_LOB）: 今回のリリースでは例は提供されません。
- C（OCI）: LOB バッファリングの使用禁止化（10-254 ページ）
- C++（OCCI）: 今回のリリースでは例は提供されません。
- COBOL（Pro*COBOL）: LOB バッファリングの使用禁止化（10-256 ページ）
- C/C++（Pro*C/C++）: LOB バッファリングの使用禁止化（10-258 ページ）
- Visual Basic（OO4O）: LOB バッファリングの使用禁止化（10-259 ページ）
- Java（JDBC）: 今回のリリースでは例は提供されません。

C（OCI）: LOB バッファリングの使用禁止化

```
/* Disabling LOB buffering (persistent LOBs) */
/* Select the locator into a locator variable: */
sb4 select_lock_adphoto_locator(Lob_loc, errhp, svchp, stmthp)
OCILobLocator *Lob_loc;
OCIError      *errhp;
OCISvcCtx     *svchp;
OCIStmt       *stmthp;
{
  text  *sqlstmt =
     (text *)"SELECT ad_photo FROM Print_media
              WHERE product_id=3060 FOR UPDATE";
  OCIDefine *defnp1;
  checkerr (errhp, OCIStmtPrepare(stmthp, errhp, sqlstmt,
                                  (ub4)strlen((char *)sqlstmt),
                                  (ub4) OCI_NTV_SYNTAX, (ub4) OCI_DEFAULT));
  checkerr (errhp, OCIDefineByPos(stmthp, &defnp1, errhp, (ub4) 1,
                                  (dvoid *)&Lob_loc, (sb4)0,
                                  (ub2) SQLT_BLOB,(dvoid *) 0,
                                  (ub2 *) 0, (ub2 *) 0, (ub4) OCI_DEFAULT));
```

```
  /* Execute the select and fetch one row: */
  checkerr(errhp, OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                                 (CONST OCISnapshot*) 0, (OCISnapshot*) 0,
                                 (ub4) OCI_DEFAULT));
    return 0;
}
#define MAXBUFLEN 32767
void lobBuffering (envhp, errhp, svchp, stmthp)
OCIEnv    *envhp;
OCIError  *errhp;
OCISvcCtx *svchp;
OCIStmt   *stmthp;
{
  OCILobLocator *Lob_loc;
  ub4 amt;
  ub4 offset;
  sword retval;
  ub1 bufp[MAXBUFLEN];
  ub4 buflen;

  /* Allocate the locator desriptor: */
  (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &Lob_loc,
                            (ub4)OCI_DTYPE_LOB, (size_t) 0, (dvoid **) 0);

  /* Select the BLOB: */
  printf (" select an ad_photo Lob\n");
  select_lock_adphoto_locator(Lob_loc, errhp, svchp, stmthp);

  /* Open the BLOB: */
  checkerr (errhp, (OCILobOpen(svchp, errhp, Lob_loc, OCI_LOB_READWRITE)));

  /* Enable LOB Buffering: */
  printf (" enable LOB buffering\n");
  checkerr (errhp, OCILobEnableBuffering(svchp, errhp, Lob_loc));

  printf (" write data to LOB\n");

  /* Write data into the LOB: */
  amt    = sizeof(bufp);
  buflen = sizeof(bufp);
  offset = 1;

  checkerr (errhp, OCILobWrite (svchp, errhp, Lob_loc, &amt,
                                offset, bufp, buflen,
                                OCI_ONE_PIECE, (dvoid *)0,
                                (sb4 (*)(dvoid*,dvoid*,ub4*,ub1 *))0,
```

```
                                    0, SQLCS_IMPLICIT));

  /* Flush the buffer: */
  printf(" flush the LOB buffers\n");
  checkerr (errhp, OCILobFlushBuffer(svchp, errhp, Lob_loc,
                                     (ub4)OCI_LOB_BUFFER_FREE));

  /* Disable Buffering: */
  printf (" disable LOB buffering\n");
  checkerr (errhp, OCILobDisableBuffering(svchp, errhp, Lob_loc));

  /* Subsequent LOB WRITEs will not use the LOB Buffering Subsystem: */

  /* Closing the BLOB is mandatory if you have opened it: */
  checkerr (errhp, OCILobClose(svchp, errhp, Lob_loc));

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

  return;

}
```

COBOL（Pro*COBOL）: LOB バッファリングの使用禁止化

```
      * DISABLING LOB BUFFERING (PERSISTENT LOBS)
        IDENTIFICATION DIVISION.
        PROGRAM-ID. LOB-BUFFERING.
        ENVIRONMENT DIVISION.
        DATA DIVISION.
        WORKING-STORAGE SECTION.

        01  USERID   PIC X(11) VALUES "SAMP/SAMP".
        01  BLOB1          SQL-BLOB.
        01  BUFFER         PIC X(10).
        01  AMT            PIC S9(9) COMP.
            EXEC SQL VAR BUFFER IS RAW(10) END-EXEC.
            EXEC SQL INCLUDE SQLCA END-EXEC.

        PROCEDURE DIVISION.
        LOB-BUFFERING.

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

```
* Allocate and initialize the BLOB locator:
     EXEC SQL ALLOCATE :BLOB1 END-EXEC.

     EXEC SQL WHENEVER NOT FOUND GOTO END-OF-BLOB END-EXEC.
     EXEC SQL
          SELECT SOUND INTO :BLOB1
          FROM MULTIMEDIA_TAB
          WHERE CLIP_ID = 1 FOR UPDATE
     END-EXEC.

* Open the BLOB and enable buffering:
     EXEC SQL LOB OPEN :BLOB1 READ WRITE END-EXEC.
     EXEC SQL
          LOB ENABLE BUFFERING :BLOB1
     END-EXEC.

* Write some data to the BLOB:
     MOVE "242424" TO BUFFER.
     MOVE 3 TO AMT.
     EXEC SQL LOB WRITE :AMT FROM :BUFFER INTO :BLOB1 END-EXEC.

     MOVE "212121" TO BUFFER.
     MOVE 3 TO AMT.
     EXEC SQL LOB WRITE :AMT FROM :BUFFER INTO :BLOB1 END-EXEC.

* Now flush the buffered writes:
     EXEC SQL LOB FLUSH BUFFER :BLOB1 END-EXEC.
     EXEC SQL LOB DISABLE BUFFERING :BLOB1 END-EXEC.
     EXEC SQL LOB CLOSE :BLOB1 END-EXEC.

 END-OF-BLOB.
     EXEC SQL WHENEVER NOT FOUND CONTINUE END-EXEC.
     EXEC SQL FREE :BLOB1 END-EXEC.
     EXEC SQL ROLLBACK WORK RELEASE END-EXEC.
     STOP RUN.

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

C/C++（Pro*C/C++）: LOB バッファリングの使用禁止化

```
/* Disabling LOB buffering (persistent LOBs) */
#include <oci.h>
#include <stdio.h>
#include <string.h>
#include <sqlca.h>

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

#define BufferLength 256

void disableBufferingLOB_proc()
{
  OCIBlobLocator *Lob_loc;
  int Amount = BufferLength;
  int multiple, Position = 1;
  /* Datatype equivalencing is mandatory for this datatype: */
  char Buffer[BufferLength];
  EXEC SQL VAR Buffer is RAW(BufferLength);
  EXEC SQL WHENEVER SQLERROR DO Sample_Error();

/* Allocate and Initialize the LOB: */
  EXEC SQL ALLOCATE :Lob_loc;
  EXEC SQL SELECT ad_photo INTO :Lob_loc
             FROM Print_media
             WHERE product_id = 3060 AND ad_id = 11001 FOR UPDATE;
  /* Enable use of the LOB Buffering Subsystem: */
  EXEC SQL LOB ENABLE BUFFERING :Lob_loc;
  memset((void *)Buffer, 0, BufferLength);
  for (multiple = 0; multiple < 7; multiple++)
    {
      /* Write data to the LOB: */
      EXEC SQL LOB WRITE ONE :Amount
                    FROM :Buffer INTO :Lob_loc AT :Position;
      Position += BufferLength;
    }
  /* Flush the contents of the buffers and Free their resources: */
  EXEC SQL LOB FLUSH BUFFER :Lob_loc FREE;
  /* Turn off use of the LOB Buffering Subsystem: */
  EXEC SQL LOB DISABLE BUFFERING :Lob_loc;
```

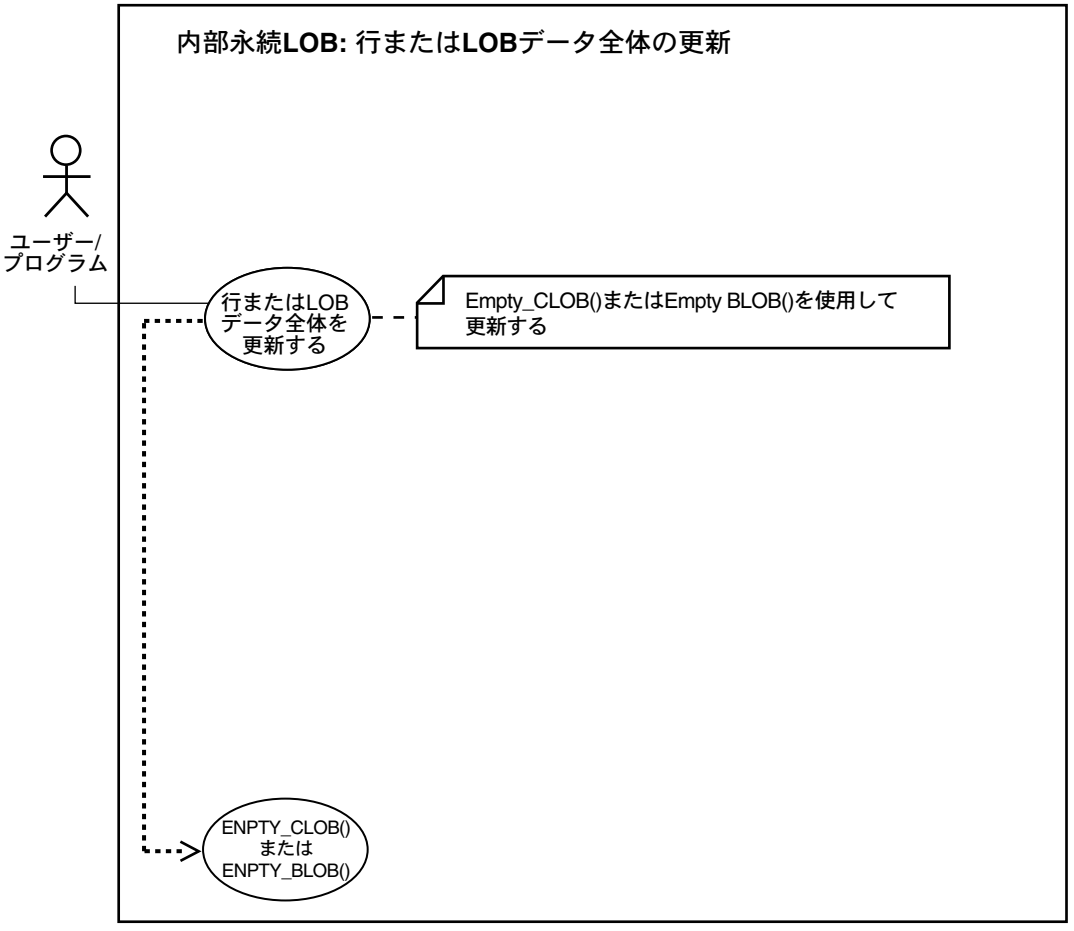
```
  /* Write APPEND can only be done when Buffering is Disabled: */
  EXEC SQL LOB WRITE APPEND ONE :Amount FROM :Buffer INTO :Lob_loc;
  /* Release resources held by the Locator: */
  EXEC SQL FREE :Lob_loc;
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  disableBufferingLOB_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

Visual Basic（OO4O）: LOB バッファリングの使用禁止化

```
'Disabling LOB buffering (persistent LOBs)
Dim MySession As OraSession
Dim OraDb As OraDatabase
Dim OraDyn As OraDynaset, OraAdPhoto1 As OraBlob, OraAdPhotoClone As OraBlob

Set MySession = CreateObject("OracleInProcServer.XOraSession")
Set OraDb = MySession.OpenDatabase("exampledb", "samp/samp", 0&)
Set OraDyn = OraDb.CreateDynaset(
   "SELECT * FROM Print_media ORDER BY product_id, ad_id", ORADYN_DEFAULT)

Set OraAdPhoto1 = OraDyn.Fields("ad_photo").Value
'Disable buffering:
OraAdPhoto1.DisableBuffering
```

EMPTY_CLOB() または EMPTY_BLOB() を使用した LOB の更新

図 10-35 利用図 : EMPTY_CLOB() または EMPTY_BLOB() を使用した LOB の更新



参照：

- 10-2 ページの表 10-1「利用モデル : 内部永続 LOB」を参照してください。
- 10-264 ページの「別の表からの LOB の選択による行の更新」を参照してください。
- 10-266 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」を参照してください。

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。

- 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)
- 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）(7-15 ページ)
- 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)
- 例 : PL/SQL - 4,000 バイトを超えるバインド（挿入は未サポート）(7-18 ページ)
- 例 : PL/SQL - 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（7-19 ページ)
- 例 : C（OCI）- INSERT および UPDATE での 4,000 バイトを超えるバインドの使用（7-19 ページ)

注意： `EMPTY_CLOB()` または `EMPTY_BLOB()` のかわりに実際の値で LOB を更新するとパフォーマンスが改善されます。

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

LOB 列を NULL 以外にする　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()` 関数またはファンクションを使用して、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 Print_media
          SET ad_sourcetext = 'This is a One Line Story'
          WHERE product_id = 2268;
```

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:『Oracle9*i* SQL リファレンス』の第 18 章「SQL 文 : SAVEPOINT ～ UPDATE」の「UPDATE」

使用例

次の例では、EMPTY_CLOB を使用して、様々なデータ型に LOB を更新する場合の一連の手続きを示します。

例

例を SQL で示します。この例は、すべてのプログラム環境に適用されます。

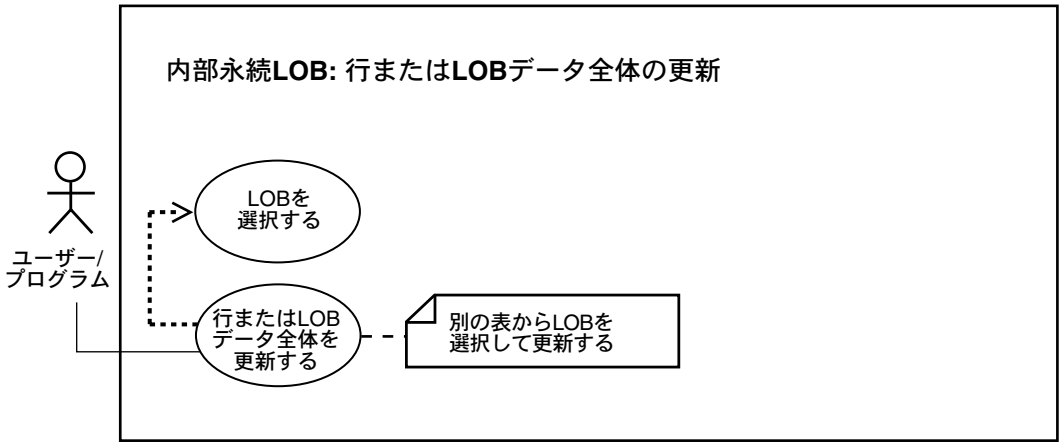
- SQL: EMPTY_CLOB() または EMPTY_BLOB() を使用した LOB の更新

SQL: EMPTY_CLOB() または EMPTY_BLOB() を使用した LOB の更新

```
UPDATE Print_media SET ad_sourcetext = EMPTY_CLOB()
       WHERE product_id = 3060 AND ad_id = 11001;

UPDATE Print_media SET ad_fltextn = EMPTY_CLOB()
       WHERE product_id = 3060 AND ad_id = 11001;

UPDATE Print_media SET ad_photo = EMPTY_BLOB()
       WHERE product_id = 3060 AND ad_id = 11001;
```

別の表からの LOB の選択による行の更新

図 10-36 利用図 : 別の表からの LOB の選択による行の更新



参照 :

- 10-2 ページの表 10-1「利用モデル : 内部永続 LOB」を参照してください。
- 10-262 ページの「EMPTY_CLOB() または EMPTY_BLOB() を使用した LOB の更新」を参照してください。
- 10-265 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」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:『Oracle9*i* SQL リファレンス』の第 18 章「SQL 文 : SAVEPOINT ～ UPDATE」の「UPDATE」

使用例

次の例では、参照を使用して online_media のデータを更新します。

例

この SQL は、すべてのプログラム環境に適用されます。

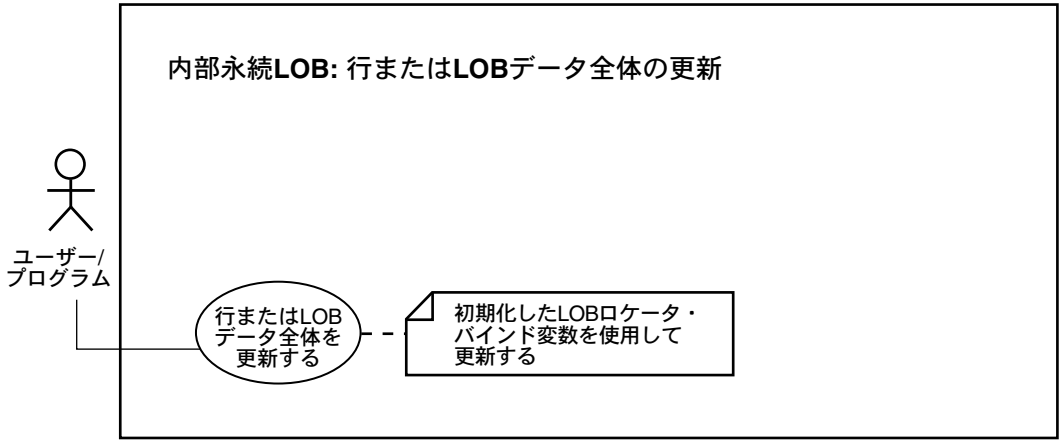
- SQL: 別の表からの LOB の選択による行の更新

SQL: 別の表からの LOB の選択による行の更新

```
Rem Updating a row by selecting a LOB from another table (persistent LOBs)

UPDATE Print_media SET ad_sourcetext =
   (SELECT * product_text FROM online_media WHERE product_id = 3060);
    WHERE product_id = 3060 AND ad_id = 11001;
```

初期化した LOB ロケータ・バインド変数を使用した更新

図 10-37 利用図 : 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 :

- 10-2 ページの表 10-1「利用モデル : 内部永続 LOB」を参照してください。
- 10-262 ページの「EMPTY_CLOB() または EMPTY_BLOB() を使用した LOB の更新」を参照してください。
- 10-263 ページの「別の表からの LOB の選択による行の更新」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関数またはファンクションのリストは、第 3 章「様々なプログラム環境での LOB のサポート」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:『Oracle9*i* SQL リファレンス』の第 18 章「SQL 文 : SAVEPOINT ～ UPDATE」の「UPDATE」
- C（OCI）:『Oracle Call Interface プログラマーズ・ガイド』の第 16 章「その他の OCI リレーショナル関数」の「LOB 関数」
- C++（OCCI）:『Oracle C++ Call Interface プログラマーズ・ガイド』
- COBOL（Pro*COBOL）:『Pro*COBOL Precompiler プログラマーズ・ガイド』の LOB に関する詳細、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・ディレクティブ」の「ALLOCATE（実行可能埋込み SQL 拡張機能）」
- C/C++（Pro*C/C++）:『Pro*C/C++ Precompiler プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびディレクティブ」の「ALLOCATE（実行可能埋込み SQL 拡張機能）」
- Visual Basic（OO4O オンライン・ヘルプ）: ヘルプの「目次」タブから、「OO4O オートメーション・サーバー・リファレンス」>「OO4O サーバーのオブジェクト」>「OraDatabase」>「メソッド」>「ExecuteSQL」を選択
- Java（JDBC）:『Oracle9*i* JDBC 開発者ガイドおよびリファレンス』の第 8 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『Oracle9*i* 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「JDBC 2.0 LOB 型と Oracle 拡張型のサポート」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 `ad_photo` データを更新します。

例

次のプログラム環境での例を示します。

- PL/SQL（DBMS_LOB）: 初期化した LOB ロケータ・バインド変数を使用した更新（10-267 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（10-267 ページ）
- C++（OCCI）: 今回のリリースでは例は提供されません。
- 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（10-269 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（10-270 ページ）

PL/SQL（DBMS_LOB）：初期化した LOB ロケータ・バインド変数を使用した更新

```
/* Updating a LOB by initializing a LOB locator bind variable */
/* Example procedure updateUseBindVariable_proc is not part of
   DBMS_LOB package: */
CREATE OR REPLACE PROCEDURE updateUseBindVariable_proc (Lob_loc BLOB) IS
BEGIN
   UPDATE Print_media SET ad_photo = lob_loc
       WHERE product_id = 3060 AND ad_id = 11001;
END;

DECLARE
   Lob_loc     BLOB;
BEGIN
   /* Select the LOB: */
   SELECT ad_photo INTO Lob_loc
      FROM Print_media
         WHERE product-id = 3060 AND ad_id = 11001;
   updateUseBindVariable_proc (Lob_loc);
   COMMIT;
END;
```

C（OCI）：初期化した LOB ロケータ・バインド変数を使用した更新

```
/* Updating a LOB by initializing a LOB locator bind variable (persistent LOBs) */
/* Select the locator into a locator variable: */
sb4 select_adphoto_locator(Lob_loc, errhp, svchp, stmthp)
OCILobLocator *Lob_loc;
OCIError      *errhp;
OCISvcCtx     *svchp;
OCIStmt       *stmthp;
{
  text  *sqlstmt =
     (text *)"SELECT ad_photo FROM Print_media
              WHERE product_id=2268";
  OCIDefine *defnp1;
```

```
  checkerr (errhp, OCIStmtPrepare(stmthp, errhp, sqlstmt,
                                  (ub4)strlen((char *)sqlstmt),
                                  (ub4) OCI_NTV_SYNTAX, (ub4) OCI_DEFAULT));
  checkerr (errhp, OCIDefineByPos(stmthp, &defnp1, errhp, (ub4) 1,
                                   (dvoid *)&Lob_loc, (sb4)0,
                                   (ub2) SQLT_BLOB,(dvoid *) 0,
                                   (ub2 *) 0, (ub2 *) 0, (ub4) OCI_DEFAULT));
 /* Execute the select and fetch one row: */
  checkerr(errhp, OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                                 (CONST OCISnapshot*) 0, (OCISnapshot*) 0,
                                 (ub4) OCI_DEFAULT));
  return 0;
}

/* Update the LOB  in the selected row in the table: */
void updateLobUsingBind (envhp, errhp, svchp, stmthp)
OCIEnv    *envhp;
OCIError  *errhp;
OCISvcCtx *svchp;
OCIStmt   *stmthp;
{
  text  *updstmt =
    (text *) "UPDATE Print_media SET ad_photo = :1 WHERE product_id = 3106";
  OCILobLocator *Lob_loc;
  OCIBind       *bndhp1;

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

  /* Select the locator: */
  printf(" select an ad_photo locator\n");
  (void)select_adphoto_locator(Lob_loc, errhp, svchp, stmthp);

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

  /* Binds the bind positions: */
  printf(" bind locator to bind position\n");

  checkerr (errhp, OCIBindByPos(stmthp, &bndhp1, errhp, (ub4) 1,
                                (dvoid *) &Lob_loc, (sb4)0, SQLT_BLOB,
                                (dvoid *) 0, (ub2 *)0, (ub2 *)0,
                                (ub4) 0, (ub4 *) 0, (ub4) OCI_DEFAULT));
```

```
  /* Execute the SQL statement: */
  printf ("update LOB column in another row using this locator\n");
  checkerr (errhp, OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                                  (CONST OCISnapshot*) 0, (OCISnapshot*) 0,
                                  (ub4) OCI_DEFAULT));

  return;
}
```

COBOL（Pro*COBOL）：初期化した LOB ロケータ・バインド変数を使用した更新

```
     * Updating a LOB by initializing a LOB locator bind variable
     * [Example script: 3806.pco]
       IDENTIFICATION DIVISION.
       PROGRAM-ID. UPDATE-BLOB.
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.

       01  BLOB1          SQL-BLOB.
       01  NEW-LEN        PIC S9(9) COMP.
       01  AMT            PIC S9(9) COMP.
       01  OFFSET         PIC S9(9) COMP.

      * Define the source and destination position and location:
       01  SRC-POS        PIC S9(9) COMP.
       01  DEST-POS       PIC S9(9) COMP.
       01  SRC-LOC        PIC S9(9) COMP.
       01  DEST-LOC       PIC S9(9) COMP.
       01  USERID   PIC X(11) VALUES "SAMP/SAMP".
           EXEC SQL INCLUDE SQLCA END-EXEC.

       PROCEDURE DIVISION.
       UPDATE-BLOB.

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

```
* Allocate and initialize the BLOB locators:
     EXEC SQL ALLOCATE :BLOB1 END-EXEC.
     EXEC SQL WHENEVER NOT FOUND GOTO END-OF-BLOB END-EXEC.
     EXEC SQL
          SELECT AD_PHOTO INTO :BLOB1
          FROM PRINT_MEDIA
          WHERE PRODUCT_ID = 2056 AND AD_ID = 12001
     END-EXEC.

     EXEC SQL
          UPDATE PRINT_MEDIA
          SET AD_PHOTO = :BLOB1
          WHERE PRODUCT_ID = 2268 AND AD_ID = 21001
     END-EXEC.

 END-OF-BLOB.
     EXEC SQL WHENEVER NOT FOUND CONTINUE END-EXEC.
     EXEC SQL FREE :BLOB1 END-EXEC.
     EXEC SQL ROLLBACK WORK RELEASE END-EXEC.
     STOP RUN.

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

C/C++（Pro*C/C++）: 初期化した LOB ロケータ・バインド変数を使用した更新

```
/* Updating a LOB by initializing a LOB locator bind variable (persistent LOBs)*/
#include <oci.h>
#include <stdio.h>
#include <sqlca.h>

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

```
void updateUseBindVariable_proc(Lob_loc)
  OCIBlobLocator *Lob_loc;
{
  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  EXEC SQL UPDATE Print_media SET ad_photo = :Lob_loc WHERE product_id = 2268;
}

void updateLOB_proc()
{
  OCIBlobLocator *Lob_loc;

  EXEC SQL ALLOCATE :Lob_loc;
  EXEC SQL SELECT ad_photo INTO :Lob_loc
           FROM Print_media WHERE product_id = 3060;
  updateUseBindVariable_proc(Lob_loc);
  EXEC SQL FREE :Lob_loc;
  EXEC SQL COMMIT WORK;
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  updateLOB_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

Visual Basic（OO4O）：初期化した LOB ロケータ・バインド変数を使用した更新

```
'Updating a LOB by initializing a LOB locator bind variable (persistent LOBs)
Dim OraDyn As OraDynaset, OraAdPhoto as OraBlob

'Select a column with product_id  = 3106:
Set OraDyn = OraDb.CreateDynaset("SELECT * FROM Print_media WHERE
    product_id= 3106", ORADYN_DEFAULT)

'Get the OraBlob object from the field:
Set OraAdPhoto = OraDyn.Fields("AD_PHOTO").Value

'Create a parameter for OraBlob object:
OraDb.Parameters.Add "AD_PHOTO",Null,ORAPARM_INPUT,ORATYPE_BLOB
```

```
'Set the value of ad_photo parameter to OraAdPhoto:
OraDb.Parameters("AD_PHOTO").Value = OraAdPhoto

'Update table Print_media with OraAdPhoto for product_id = 2268:
OraDb.ExecuteSQL("Update  Print_media SET ad_photo = :AD_PHOTO
   WHERE product_id = 2268")
```

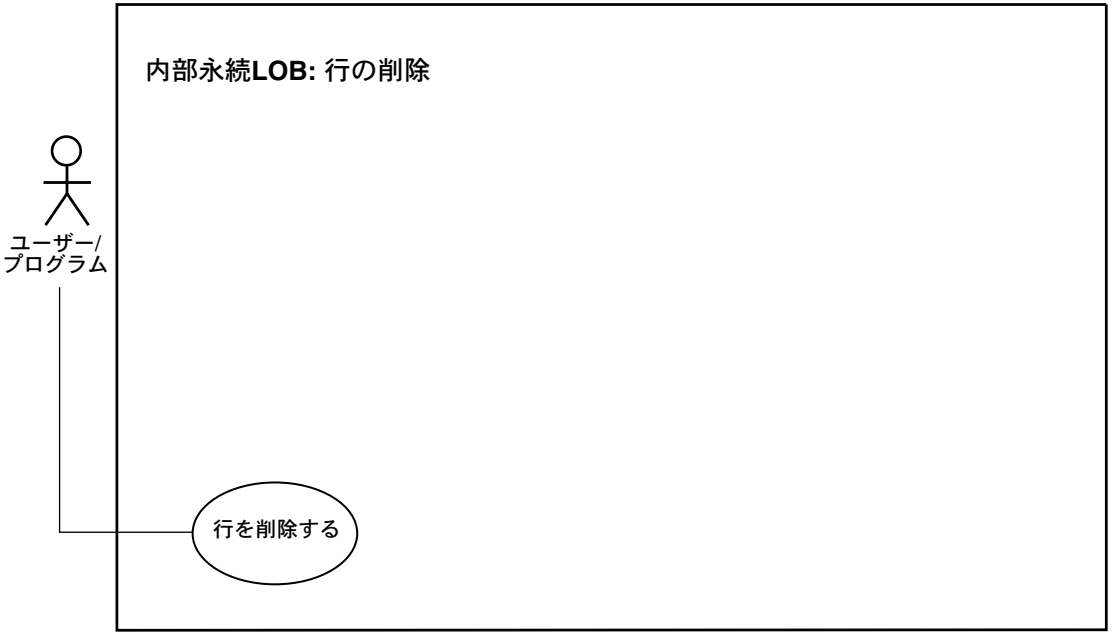
Java（JDBC）：初期化した LOB ロケータ・バインド変数を使用した更新

```
// Updating a LOB by initializing a LOB locator bind variable (persistent LOBs)
import java.sql.Connection;
import java.sql.Statement;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

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

public class Ex2_163
{
 public static void main (String args [])
       throws Exception
  {
    // Load the Oracle JDBC driver:
      DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

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

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

    // Create a Statement:
    Statement stmt = conn.createStatement ();
    try
    {
     ResultSet rset = stmt.executeQuery (
        "SELECT ad_photo FROM Print_media
                WHERE product_id = 3106 AND ad_id = 13001");
```

```
      if (rset.next())
      {
       // retrieve the LOB locator from the ResultSet:
       BLOB photo_blob = ((OracleResultSet)rset).getBLOB (1);

        OraclePreparedStatement ops =
          (OraclePreparedStatement) conn.prepareStatement(
             "UPDATE Print_media SET ad_photo = ?
                     WHERE product_id = 2056 AND ad_id = 12001");
          ops.setBlob(1, photo_blob);
          ops.execute();
          rset.close();
          stmt.close();
     conn.commit();
     conn.close();
   }
    }
    catch (SQLException e)
    {
       e.printStackTrace();
    }
  }
}
```

LOB を含む表の行の削除

図 10-38 利用図 : LOB を含む表の行の削除



参照： 10-2 ページの表 10-1「利用モデル : 内部永続 LOB」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 列または内部 LOB 属性を含む行を削除します。

- SQL データ操作言語（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、読込み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。この項目についての詳しい説明があります。詳細は、5-2 ページの「読込み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:『Oracle9*i* SQL リファレンス』の第 16 章「SQL 文 : CREATE TYPE ～ DROP ROLLBACK SEGMENT」の「DELETE」、第 17 章「SQL 文 : DROP SEQUENCE ～ ROLLBACK」の「DROP TABLE」、第 18 章「SQL 文 : SAVEPOINT ～ UPDATE」の「TRUNCATE」

使用例

次の例では、product_id = 3060 AND ad_id 11001 の製品に関連するすべてのデータを削除します。

例

この SQL は、すべてのプログラム環境に適用されます。

- SQL: LOB の削除（10-275 ページ）

SQL: LOB の削除

```
DELETE FROM Print_media WHERE product_id = 3060 AND ad_id = 11001;
TRUNCATE TABLE Print_media;
DROP TABLE Print_media;
```

11

一時 LOB

利用モデル

この章では、一時 LOB に対する操作（「一時 LOB への BFILE データのロード」など）を、利用方法の点から説明します。表 11-1「利用モデル : 内部一時 LOB」に、すべての利用方法を示します。

個々の利用方法

一時 LOB の個々の利用方法を、次のように説明します。

- **利用図** : 利用方法を表す図。図の見方については、付録 A「Unified Modeling Language 図」を参照してください。
- **用途** : LOB に関するこの利用方法の用途。
- **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、利用方法に対する LOB 関連操作の基礎となる構文。
- **使用例** : 利用方法の実装例について、サンプル・スキーマの点から説明した例。サンプル・スキーマの詳細は、『Oracle9*i* サンプル・スキーマ』を参照してください。
- **例** : 各利用方法の適用法。サンプル・スキーマを基にしています。

利用モデル : 内部一時 LOB

表 11-1「利用モデル : 内部一時 LOB」の「+」は、特定の利用方法で、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「様々なプログラム環境での LOB のサポート」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 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 を使用した Visual Basic
- **J** － JDBC を使用した Java

表 11-1 利用モデル : 内部一時 LOB

利用方法およびページ	P	O	B	C	V*	J
11-146 ページの「他の LOB への一時 LOB の追加」	+	+	+	+	-	-
11-22 ページの「LOB が一時 LOB であるかどうかの確認」	+	+	+	+	-	+
11-89 ページの「2 つの一時 LOB の全体または一部の比較」	+	-	+	+	-	-
11-122 ページの「一時 LOB の LOB ロケータのコピー」	+	+	+	+	-	-
11-112 ページの「他の LOB への一時 LOB の全体または一部のコピー」	+	+	+	+	-	-
11-13 ページの「一時 LOB の作成」	+	+	+	+	-	+
11-136 ページの「一時 LOB の LOB ロケータが初期化されているかどうかの確認」	-	+	-	+	-	-
11-96 ページの「一時 LOB 内のパターンの有無の確認（instr）」	+	-	+	+	-	-
11-38 ページの「一時 LOB への BFILE データのロード」	+	+	+	+	-	-
11-46 ページの「一時 BLOB への BFILE のバイナリ・データのロード」	+	-	-	-	-	-
11-50 ページの「一時 CLOB/NCLOB へのファイルのキャラクタ・データのロード」	+	-	-	-	-	-
11-207 ページの「一時 LOB に対する LOB バッファリングの使用禁止化」	-	+	+	+	-	-
11-61 ページの「一時 LOB データの表示」	+	+	+	+	-	-

表 11-1 利用モデル : 内部一時 LOB（続き）

利用方法およびページ	P	O	B	C	V*	J
11-193 ページの「一時 LOB に対する LOB バッファリングの使用可能化」	-	+	+	+	-	-
11-184 ページの「一時 LOB の一部の消去」	+	+	+	+	-	-
11-143 ページの「一時 LOB のキャラクタ・セット・フォームの取得」	-	+	-	-	-	-
11-140 ページの「一時 LOB のキャラクタ・セット ID の取得」	-	+	-	-	-	-
11-103 ページの「一時 LOB の長さの取得」	+	+	+	+	-	-
11-200 ページの「一時 LOB に対するバッファのフラッシュ」	-	+	+	+	-	-
11-30 ページの「一時 LOB の解放」	+	+	+	+	-	+
11-131 ページの「2 つの一時 LOB ロケータが等しいかどうかの確認」	-	+	-	+	-	-
11-38 ページの「一時 LOB への BFILE データのロード」	+	+	+	+	-	-
11-46 ページの「一時 BLOB への BFILE のバイナリ・データのロード」	+	-	-	-	-	-
11-50 ページの「一時 CLOB/NCLOB へのファイルのキャラクタ・データのロード」	+	-	-	-	-	-
11-71 ページの「一時 LOB からのデータの読込み」	+	+	+	+	-	-
11-82 ページの「一時 LOB の一部の読込み（substr）」	+	-	+	+	-	-
11-175 ページの「一時 LOB データの切捨て」	+	+	+	+	-	-
11-156 ページの「一時 LOB への追加の書込み」	+	+	+	+	-	-
11-164 ページの「一時 LOB へのデータの書込み」	+	+	+	+	-	-

* 内部一時 LOB の利用方法には、Visual Basic の例はありません。

プログラム環境

Oracle9*i* では、次のプログラム環境または「インタフェース」での一時 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
- JDBC を使用した Java

ロケータ

前述の「インタフェース」は、一時 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。一時 LOB はどの表にも含まれないため、一時 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一時 LOB ロケータ

一時 LOB に対する SQL は、IN 値として使用できる一時 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： 一時 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 をコピーして一時 LOB に置くことはありません。

一時 LOB および内部永続 LOB に使用できる関数またはファンクション

内部永続 LOB および一時 LOB では、次のファンクションを使用できます。

- 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（OCILobLocatorAssign、OCILobLocatorIsInit など）。

さらに、ISTEMPORARY ファンクションを使用して、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： 一時 LOB では、トランザクションおよび読込み一貫性をサポートしていません。

ヒントとしての DBMS_LOB.createtemporary() パラメータ

DBMS_LOB.createtemporary() のコールには、存続期間パラメータを使用します。このパラメータは単なるヒントであり、厳密には施行されません。

一時表領域への一時 LOB データの格納

一時 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一時 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。

一時 LOB は表スキーマに対応付けられていないため、一時 LOB については「インライン」および「アウトライン」という言葉は意味を持ちません。

注意： すべての一時 LOB はサーバーに格納されます。クライアント側では一時 LOB はサポートされていません。

一時 LOB の存続期間

一時 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。

一時 LOB の作成インタフェースに、一時 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、すべての一時 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のインスタンスが終了した場合は、すべての一時 LOB が削除されます。

OCI による論理バケットへの一時 LOB のグループ化

OCI ユーザーは、一時 LOB を論理バケットにグループ化できます。

OCIDuration は、一時 LOB に対する存続期間を表します。ユーザーが特定の存続期間を指定しない場合は、一時 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、一時 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。各一時 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一時 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、一時 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。

一時 LOB 領域の明示的な解放による再利用

一時 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。一時 LOB を明示的に解放することによって、セッション内の一時 LOB 領域を再利用できます。

- セッションが完了するとき : 1 つ以上の一時 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戻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利用可能な状態で保持されます。かわりに、セッションで再利用可能な状態で保持されます。
- 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、一時 LOB が削除されるとともに、一時 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、領域は全般的な再利用のために一時表領域に戻されます。

一時 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かぎり、`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、一時 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一時 LOB を明示的に解放します。

CR およびロールバックは一時 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、一時 LOB を解放してから再開する必要があります。

参照 : 第 7 章「モデリングおよび設計」、7-4 ページの「LOB 記憶域」および第 9 章「LOB: 効果的な使用方法」を参照してください。

ロケータおよびセマンティクス

ユーザーが一時 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。一時 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、一時 LOB ロケータには固有の機能があります。

一時 LOB 固有の機能

次の機能は、一時 LOB に固有のものです。

- **永続 LOB ロケータによる一時 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、上書きされた一時 LOB を指す temporary_lob のロケータのコピーを保持しないかぎり、この一時 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一時 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、一時 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一時 LOB に対して生成されないため、複数のロケータを同じ一時 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、意味的には一時 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 OCILobLocatorAssign または PL/SQL の同等の代入を使用するたびに、データベースは一時 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 つのロケータが一時 LOB の作成に使用され、OCILobLocatorAssign を使用してその一時 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一時 LOB をコピーして 2 つ目のロケータが元の一時 LOB ではなくコピーを指すようにします。

  - **一時 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。一時 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、同じ一時 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、モジュール間で一時 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、各一時 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一時 LOB のローカル・コピーを作成することを回避できます。

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。

* **一時 LOB の他の一時 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 が参照していた一時 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**

一時 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、更新された一時 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。一時 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一時 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、オブジェクト型が一時 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：

- 『Oracle9*i* データベース概要』を参照してください。
- 『Oracle9*i* 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。

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薦めします。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一時 LOB を使用しないでください。

* SELECT 文では、LOB 値を INTO 句で一時 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一時 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**: 存続期間内の一時 LOB を持ち、その存続期間で OCIDurationEnd をコールし、その後にその一時 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一時 LOB を解放しようとします。また、そのようなロケータを使用して一時 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一時 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 OCIDuration 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一時 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。

一時 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。

- 一時 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一時 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一時 LOB にアクセスできません。
- 一時 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、『PL/SQL ユーザーズ・ガイドおよびリファレンス』の第 7 章「PL/SQL のサブプログラム」の「NOCOPY コンパイラ・ヒントを使用した大型データ構造の受渡し」を参照してください。

一時 LOB の管理

Oracle は一時 LOB をセッションごとに追跡しており、v$temporary_lobs という v$ ビューを提供します。アプリケーションは、どのユーザーが一時 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、および一時 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。

JDBC および一時 LOB の使用

Oracle9*i* JDBC ドライバには、一時 LOB を作成およびクローズするための API が含まれます。これらの API によって、以前使用していた dbms_lob PL/SQL パッケージの dbms_lob.createTemporary()、dbms_lob.isTemporary() および dbms_lob.freeTemporary() を使用する必要がなくなります。

JDBC および一時 LOB の使用

oracle.sql.BLOB クラスは、標準 JDBC java.sql.Blob インタフェースの Oracle JDBC ドライバの実装です。oracle.sql.BLOB インスタンスは、データベース内の BLOB オブジェクトを表します。表 11-2 に、一時 BLOB にアクセスするための、oracle.sql.BLOB にある新しい Oracle 拡張 API を示します。

表 11-2 JDBC: 一時 BLOB API

メソッド	説明
`public static BLOB createTemporary( Connection conn, boolean cache, int duration) throws SQLException`	一時 BLOB を作成します。
`public static boolean isTemporary( BLOB blob) throws SQLException`	指定の BLOB ロケータが一時 BLOB を参照するかどうかを確認します。
`public boolean isTemporary() throws SQLException`	現行の BLOB ロケータが一時 BLOB を参照するかどうかを確認します。
`public static void freeTemporary( BLOB temp_blob) throws SQLException`	指定の一時 BLOB を解放します。
`public void freeTemporary() throws SQLException`	一時 BLOB を解放します。

JDBC および一時 CLOB の使用

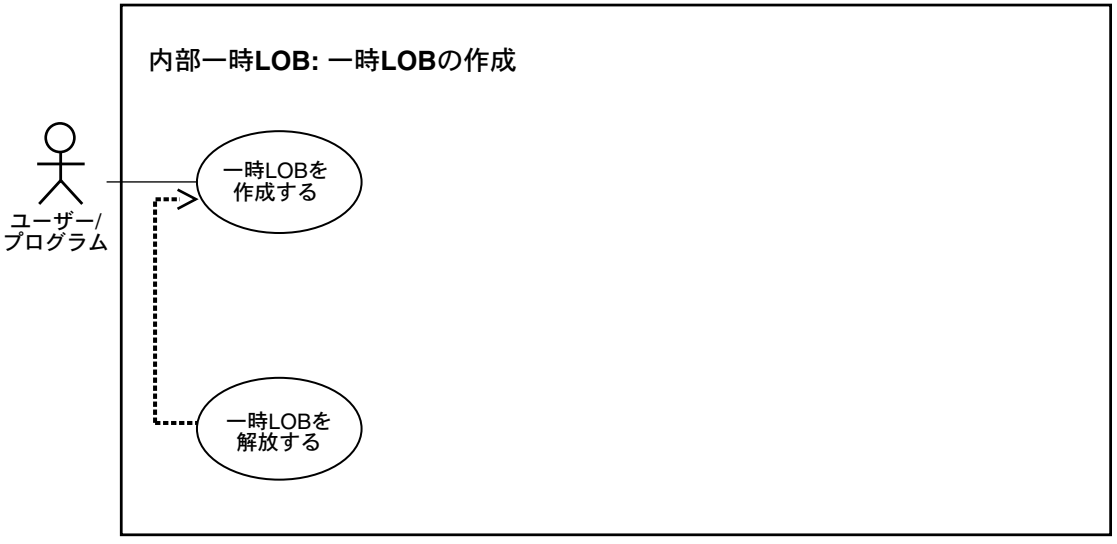
oracle.sql.CLOB クラスは、標準 JDBC java.sql.Clob インタフェースの Oracle JDBC ドライバの実装です。表 11-3 に、oracle.sql.CLOB にある、一時 CLOB にアクセスするための新しい Oracle 拡張 API を示します。

表 11-3 JDBC: 一時 CLOB API

メソッド	説明
`public static CLOB createTemporary( Connection conn, boolean cache, int duration) throws SQLException`	一時 CLOB を作成します。
`public static boolean isTemporary( CLOB clob) throws SQLException`	指定の CLOB ロケータが一時 CLOB を参照するかどうかを確認します。
`public boolean isTemporary() throws SQLException`	現行の CLOB ロケータが一時 CLOB を参照するかどうかを確認します。
`public static void freeTemporary( CLOB temp_clob) throws SQLException`	指定の一時 CLOB を解放します。
`public void freeTemporary() throws SQLException`	一時 CLOB を解放します。

一時 LOB の作成

図 11-1 利用図 : 一時 LOB の作成



参照 : 11-2 ページの表 11-1「利用モデル : 内部一時 LOB」を参照してください。

用途

一時 LOB を作成します。

使用上の注意

一時 LOB は、作成された時点では空です。

一時 LOB は、永続 LOB でサポートされている `EMPTY_BLOB()` ファンクションまたは `EMPTY_CLOB()` ファンクションをサポートしません。`EMPTY_BLOB()` ファンクションは、LOB の初期化はされて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関数またはファンクションのリストは、第 3 章「様々なプログラム環境での LOB のサポート」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）：『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』の第 23 章「DBMS_LOB」の「DBMS_LOB サブプログラムの要約」の「CREATETEMPORARY プロシージャ」および「FREETEMPORARY プロシージャ」
- C（OCI）：『Oracle Call Interface プログラマーズ・ガイド』の第 16 章「その他の OCI リレーショナル関数」の「LOB 関数」の「OCILobCreateTemporary()」
- COBOL（Pro*COBOL）：『Pro*COBOL Precompiler プログラマーズ・ガイド』の LOB に関する詳細、LOB 文の使用上の注意、付録 F「埋込み SQL およびプリコンパイラ・ディレクティブ」の「LOB DESCRIBE（実行可能埋込み SQL 拡張機能）」および「LOB COPY（実行可能埋込み SQL 拡張機能）」
- C/C++（Pro*C/C++）：『Pro*C/C++ Precompiler プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびディレクティブ」の「LOB DESCRIBE（実行可能埋込み SQL 拡張機能）」および「LOB COPY（実行可能埋込み SQL 拡張機能）」
- Visual Basic（OO4O）：参照マニュアルはありません。
- Java（JDBC）：『Oracle9*i* JDBC 開発者ガイドおよびリファレンス』の第 8 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『Oracle9*i* 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「JDBC 2.0 LOB 型と Oracle 拡張型のサポート」の「BLOB、CLOB および BFILE のサポート」

使用例

次の例では、一時 LOB を作成し、Print_media 表の ad_composite イメージからその一時 LOB に、オブジェクトのコンテンツをコピーします。この技術は、たとえば、イメージをあるグラフィック・フォーマットから他のグラフィック・フォーマットに変換する必要がある場合に有効です。作成された一時 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での例を示します。

- PL/SQL（DBMS_LOB）：一時 LOB の作成（11-15 ページ）
- C（OCI）：一時 LOB の作成（11-16 ページ）
- COBOL（Pro*COBOL）：一時 LOB の作成（11-18 ページ）
- C/C++（Pro*C/C++）：一時 LOB の作成（11-19 ページ）

- Visual Basic（OO4O）: 今回のリリースでは例は提供されません。
- Java（JDBC）: 一時 BLOB の作成（11-20 ページ）

PL/SQL（DBMS_LOB）: 一時 LOB の作成

注意： この章に示す例を使用するには、次のデータ構造を設定する必要があります。

```
Rem  Set up script for working with Temporary LOBs

DROP TABLE long_raw_tab;
CREATE TABLE long_raw_tab (id number, long_raw_col long raw);
INSERT INTO long_raw_tab VALUES (1, HEXTORAW('7D'));
INSERT INTO Print_media (product_id,ad_composite) SELECT
   id,TO_LOB(long_raw_col) FROM long_raw_tab;
```

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 ad_composite INTO Src_loc FROM Print_media
       WHERE product_id = 3060 AND ad_id = 11001;

    /* Create a temporary LOB: */
   DBMS_LOB.CREATETEMPORARY(Dest_loc,TRUE);

   /* Copy the entire object from the Src_loc to the Temporary Lob: */
   DBMS_LOB.COPY(Dest_loc,Src_loc,DBMS_LOB.GETLENGTH(Src_loc),1,1);
   DBMS_LOB.FREETEMPORARY(Dest_loc);
END;
```

C（OCI）：一時 LOB の作成

```
/* Creating a temporary LOB using C(OCI). [Example script: 3820.c]
  This function reads in one of the composite ads, ad_composite,
  from table Print_media.It creates a temporary LOB so that you can use the
  temporary LOB to convert the image from one format to another, say JPG to GIFF.
  The Temporary LOB created is read through the CACHE, and is automatically cleaned
  up after the your session, if it is not explicitly freed sooner.
  This function returns 0 if it completes successfully, and -1 if it fails: */

sb4 select_and_createtemp (OCILobLocator *lob_loc,
                           OCIError      *errhp,
                           OCISvcCtx     *svchp,
                           OCIStmt       *stmthp,
                           OCIEnv        *envhp)
{
  OCIDefine     *defnp1, *defnp2;
  OCIBind       *bndhp;
  text          *sqlstmt;
  int rowind =1;
  ub4 loblen = 0;
  OCILobLocator *tblob;
  printf ("in select_and_createtemp \n");
  if(OCIDescriptorAlloc((dvoid*)envhp, (dvoid **)&tblob,
                        (ub4)OCI_DTYPE_LOB, (size_t)0, (dvoid**)0))
  {
    printf("failed in OCIDescriptor Alloc in select_and_createtemp \n");
    return -1;
  }

  /* Arbitrarily select where product_id =3060: */
  sqlstmt = (text *)
     "SELECT ad_composite FROM Print_media
         WHERE product_id = 3060 AND ad_id = 11001 FOR UPDATE";

  if (OCIStmtPrepare(stmthp, errhp, sqlstmt,
                     (ub4) strlen((char *)sqlstmt),
                     (ub4) OCI_NTV_SYNTAX, (ub4) OCI_DEFAULT))
  {
      (void) printf("FAILED: OCIStmtPrepare() sqlstmt\n");
      return -1;
  }

  /* Define for BLOB: */
  if (OCIDefineByPos(stmthp,
             &defnp1, errhp, (ub4) 1, (dvoid *) &lob_loc, (sb4)0,
             (ub2) SQLT_BLOB, (dvoid *) 0, (ub2 *) 0,
```

```
            (ub2 *) 0, (ub4) OCI_DEFAULT)
    || &defnp2, errhp, (ub4) 2, (dvoid *) &lob_loc, (sb4)0,
            (ub2) SQLT_BLOB, (dvoid *) 0, (ub2 *) 0,
            (ub2 *) 0, (ub4) OCI_DEFAULT))
{
  (void) printf("FAILED: Select locator: OCIDefineByPos()\n");
  return -1;
}
/* Execute the select and fetch one row: */
if (OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                   (CONST OCISnapshot*) 0, (OCISnapshot*) 0,
                   (ub4) OCI_DEFAULT))
{
  (void) printf("FAILED: OCIStmtExecute() sqlstmt\n");
  return -1;
}
if(OCILobCreateTemporary(svchp,
                         errhp, tblob, (ub2)0, SQLCS_IMPLICIT,
                         OCI_TEMP_BLOB, OCI_ATTR_NOCACHE,
                         OCI_DURATION_SESSION))
{
  (void) printf("FAILED: CreateTemporary() \n");
  return -1;
}

if (OCILobGetLength(svchp, errhp, lob_loc, &loblen) != 0)
{
  printf("OCILobGetLength FAILED\n");
  return -1;
}
if (OCILobCopy(svchp, errhp, tblob,lob_loc,(ub4)loblen, (ub4) 1, (ub4) 1))
{
  printf( "OCILobCopy FAILED \n");
}
if(OCILobFreeTemporary(svchp,errhp,tblob))
{
  printf ("FAILED: OCILobFreeTemporary call \n");
  return -1;
}

return 0;
}
```

COBOL（Pro*COBOL）：一時 LOB の作成

```
* Creating a Temporary LOB  [example script: 3821.pco]
  IDENTIFICATION DIVISION.
  PROGRAM-ID. CREATE-TEMPORARY.
  ENVIRONMENT DIVISION.
  DATA DIVISION.
  WORKING-STORAGE SECTION.

  01  USERID   PIC X(11) VALUES "SAMP/SAMP".
  01  BLOB1          SQL-BLOB.
  01  TEMP-BLOB      SQL-BLOB.
  01  LEN            PIC S9(9) COMP.
  01  D-LEN          PIC 9(9).
  01  ORASLNRD       PIC 9(4).

      EXEC SQL INCLUDE SQLCA END-EXEC.
      EXEC ORACLE OPTION (ORACA=YES) END-EXEC.
      EXEC SQL INCLUDE ORACA END-EXEC.
  PROCEDURE DIVISION.
  CREATE-TEMPORARY.
      EXEC SQL WHENEVER SQLERROR DO PERFORM SQL-ERROR END-EXEC.
      EXEC SQL
           CONNECT :USERID
      END-EXEC.

 * Allocate and initialize the CLOB locators:
      EXEC SQL ALLOCATE :BLOB1 END-EXEC.
      EXEC SQL ALLOCATE :TEMP-BLOB END-EXEC.
      EXEC SQL
           LOB CREATE TEMPORARY :TEMP-BLOB
      END-EXEC.

      EXEC SQL WHENEVER NOT FOUND GOTO END-OF-BLOB END-EXEC.
      EXEC ORACLE OPTION (SELECT_ERROR=NO) END-EXEC.
      EXEC SQL
           SELECT AD_COMPOSITE INTO :BLOB1
           FROM PRINT_MEDIA
           WHERE PRODUCT_ID = 3060 AND AD_ID = 11001
      END-EXEC.

 * Get the length of the persistent BLOB:
      EXEC SQL
           LOB DESCRIBE :BLOB1
           GET LENGTH INTO :LEN
      END-EXEC.
```

```
* Copy the entire length from persistent to temporary:
     EXEC SQL
          LOB COPY :LEN FROM :BLOB1 TO :TEMP-BLOB
     END-EXEC.

* Free the temporary LOB:
     EXEC SQL
          LOB FREE TEMPORARY :TEMP-BLOB
     END-EXEC.

 END-OF-BLOB.
     EXEC SQL WHENEVER NOT FOUND CONTINUE END-EXEC.
     EXEC SQL FREE :BLOB1 END-EXEC.
     EXEC SQL FREE :TEMP-BLOB END-EXEC.
     STOP RUN.

 SQL-ERROR.
     EXEC SQL WHENEVER 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++）：一時 LOB の作成

```
/* Creating a temporary LOB  [example script #: 3822.c] */
#include <oci.h>
#include <stdio.h>
#include <sqlca.h>

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

void createTempLOB_proc()
{
  OCIBlobLocator *Lob_loc, *Temp_loc;
  int Amount;

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
```

```
  /* Allocate the LOB Locators: */
  EXEC SQL ALLOCATE :Lob_loc;
  EXEC SQL ALLOCATE :Temp_loc;

  /* Create the Temporary LOB: */
  EXEC SQL LOB CREATE TEMPORARY :Temp_loc;
  EXEC SQL SELECT ad_composite INTO :Lob_loc FROM Print_media
       WHERE product_ID = 3060 AND ad_id = 111001;

  /* Copy the full length of the source LOB into the Temporary LOB: */
  EXEC SQL LOB DESCRIBE :Lob_loc GET LENGTH INTO :Amount;
  EXEC SQL LOB COPY :Amount FROM :Lob_loc TO :Temp_loc;

  /* Free the Temporary LOB: */
  EXEC SQL LOB FREE TEMPORARY :Temp_loc;

  /* Release resources held by the Locators: */
  EXEC SQL FREE :Lob_loc;
  EXEC SQL FREE :Temp_loc;
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  createTempLOB_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

Java（JDBC）: 一時 BLOB の作成

JDBC アプリケーションは、oracle.sql.BLOB クラスに定義されている createTemporary static メソッドを使用して一時 BLOB を作成できます。createTemporary static メソッドの定義は次のとおりです。

```
/**
 * Create a temporary blob.
 *
 * @param cache Specifies if LOB should be read into buffer cache or not.
 * @param duration The duration of the temporary LOB. The following are
 *                          valid values: DURATION_SESSION, DURATION_CALL.
 * @return A temporary blob.
 * @since 8.2.0
 */
public static BLOB createTemporary (Connection conn, boolean cache, int duration)
throws SQLException
```

存続期間パラメータに指定可能な値は次のとおりです。

public static final int DURATION_SESSION

public static final int DURATION_CALL

createTemporary をコールするたびに一時 BLOB が戻ります。次に例を示します。

```
// Make a JDBC connection
   Connection conn = ...

// Create a temporary BLOB
   BLOB temporaryBlob = BLOB.createTemporary (conn, true,
        BLOB.DURATION_SESSION);
```

この新しい API によって、以前の DBMS_LOB.createTemporary() を使用する必要がなくなります。

Java（JDBC）：一時 CLOB の作成

JDBC アプリケーションは、oracle.sql.CLOB クラスに定義されている createTemporary static メソッドを使用して一時 CLOB を作成できます。createTemporary static メソッドの定義は次のとおりです。

```
/**
* Create a temporary clob.
*
* @param cache Specifies if LOB should be read into buffer cache or not.
* @param duration The duration of the temporary LOB. The following are
*                              valid values: DURATION_SESSION, DURATION_CALL.
* @return A temporary clob.
*/
public static CLOB createTemporary (Connection conn, boolean cache, int duration)
throws SQLException
```

存続期間パラメータに使用可能な値は次のとおりです。

```
public static final int DURATION_SESSION
public static final int DURATION_CALL
```

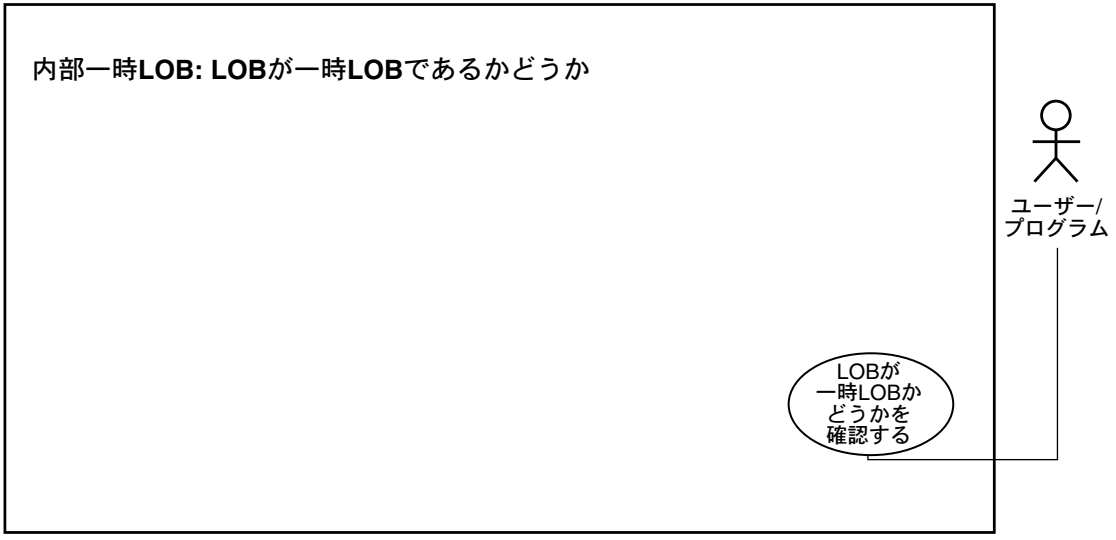
createTemporary をコールするたびに一時 CLOB が戻ります。次に例を示します。

```
// Make a JDBC connection
   Connection conn = ...
// Create a temporary CLOB
   CLOB temporaryClob = CLOB.createTemporary (conn, true, CLOB.DURATION_SESSION);
```

この新しい API によって、以前の DBMS_LOB.createTemporary() を使用する必要がなくなります。

LOB が一時 LOB であるかどうかの確認

図 11-2 利用図 : LOB が一時 LOB であるかどうかの確認



参照 : 11-2 ページの表 11-1「利用モデル : 内部一時 LOB」を参照してください。

用途

LOB が一時 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関数またはファンクションのリストは、第 3 章「様々なプログラム環境での LOB のサポート」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）:『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』の第 23 章「DBMS_LOB」の「DBMS_LOB サブプログラムの要約」の「ISTEMPORARY ファンクション」および「FREETEMPORARY プロシージャ」

- C（OCI）：『Oracle Call Interface プログラマーズ・ガイド』の第 16 章「その他の OCI リレーショナル関数」の「LOB 関数」の「OCILobIsTemporary()」
- COBOL（Pro*COBOL）：『Pro*COBOL Precompiler プログラマーズ・ガイド』の LOB に関する詳細、LOB 文の使用上の注意、付録 F「埋込み SQL およびプリコンパイラ・ディレクティブ」の「LOB DESCRIBE（実行可能埋込み SQL 拡張機能）」
- C/C++（Pro*C/C++）：『Pro*C/C++ Precompiler プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびディレクティブ」の「DESCRIBE（実行可能埋込み SQL 拡張機能）」
- Visual Basic（OO4O）：参照マニュアルはありません。
- Java（JDBC）：『Oracle9*i* JDBC 開発者ガイドおよびリファレンス』の第 8 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『Oracle9*i* 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「JDBC 2.0 LOB 型と Oracle 拡張型のサポート」の「BLOB、CLOB および BFILE のサポート」

使用例

次の例は、ロケータが一時 LOB に対応付けられているかどうかを問い合せます。

例

次のプログラム環境での例を示します。

- PL/SQL（DBMS_LOB）：LOB が一時 LOB であるかどうかの確認（11-24 ページ）
- C（OCI）：LOB が一時 LOB であるかどうかの確認（11-24 ページ）
- COBOL（Pro*COBOL）：LOB が一時 LOB であるかどうかの確認（11-25 ページ）
- C/C++（Pro*C/C++）：LOB が一時 LOB であるかどうかの確認（11-27 ページ）

- Visual Basic（OO4O）：今回のリリースでは例は提供されません。

- Java（JDBC）：BLOB が一時 BLOB であるかどうかの確認（11-28 ページ）
- Java（JDBC）：CLOB が一時的であるかどうかの確認（11-29 ページ）

PL/SQL（DBMS_LOB）: LOB が一時 LOB であるかどうかの確認

```
/* Checking if a LOB is temporary. [Example script: 3828.sql]
   This is an example of freeing a temporary LOB. First test to make
   sure that the LOB locator points to a temporary LOB, then free it.
   Otherwise, issue an error: */

CREATE or REPLACE PROCEDURE freeTempLob_proc(Lob_loc IN OUT BLOB) IS
BEGIN
   /* Free the temporary LOB locator passed in. */
   /* First check to make sure that the locator is pointing to a temporary
      LOB:*/
    IF DBMS_LOB.ISTEMPORARY(Lob_loc) = 1 THEN
       /* Free the temporary LOB locator: */
        DBMS_LOB.FREETEMPORARY(Lob_loc);
        DBMS_OUTPUT.PUT_LINE(' temporary LOB was freed');
    ELSE
        /* Print an error: */
        DBMS_OUTPUT.PUT_LINE(
           'Locator passed in was not a temporary LOB locator');
    END IF;
END;
```

C（OCI）: LOB が一時 LOB であるかどうかの確認

```
/* Checking if a LOB is temporary.  [Example script: 3829.c]
   This function frees a temporary LOB. It takes a locator as an argument,
   checks to see if it is a temporary LOB. If it is, the function frees
   the temporary LOB. Otherwise, it prints out a message saying the locator
   was not a temporary LOB locator. This function returns 0 if it
   completes successfully, -1 otherwise: */

sb4 check_and_free_temp(OCILobLocator *tblob,
                        OCIError      *errhp,
                        OCISvcCtx     *svchp,
                        OCIStmt       *stmthp,
                        OCIEnv        *envhp)
{
  boolean is_temp;
  is_temp = FALSE;
```

```
if (OCILobIsTemporary(envhp, errhp, tblob, &is_temp))
{
  printf ("FAILED: OCILobIsTemporary call\n");
  return -1;
}
if(is_temp)
{
    if(OCILobFreeTemporary(svchp, errhp, tblob))
    {
        printf ("FAILED: OCILobFreeTemporary call\n");
        return -1;

    }else
    {
        printf("Temporary LOB freed\n");
    }
}else
{
    printf("locator is not a temporary LOB locator\n");
}
return 0;
}
```

COBOL（Pro*COBOL）: LOB が一時 LOB であるかどうかの確認

```
* Checking if a LOB is temporary  [Example script: 3830.pco]
  IDENTIFICATION DIVISION.
  PROGRAM-ID. TEMP-LOB-ISTEMP.
  ENVIRONMENT DIVISION.
  DATA DIVISION.
  WORKING-STORAGE SECTION.

  01  USERID   PIC X(11) VALUES "SAMP/SAMP".
  01  TEMP-BLOB      SQL-BLOB.
  01  IS-TEMP        PIC S9(9) COMP.
  01  ORASLNRD       PIC 9(4).

      EXEC SQL INCLUDE SQLCA END-EXEC.
      EXEC ORACLE OPTION (ORACA=YES) END-EXEC.
      EXEC SQL INCLUDE ORACA END-EXEC.
```

```
 PROCEDURE DIVISION.
 CREATE-TEMPORARY.
     EXEC SQL WHENEVER SQLERROR DO PERFORM SQL-ERROR END-EXEC.
     EXEC SQL
          CONNECT :USERID
     END-EXEC.

* Allocate and initialize the BLOB locators:
     EXEC SQL ALLOCATE :TEMP-BLOB END-EXEC.
     EXEC SQL
          LOB CREATE TEMPORARY :TEMP-BLOB
     END-EXEC.

* Check if the LOB is temporary:
     EXEC SQL
          LOB DESCRIBE :TEMP-BLOB
          GET ISTEMPORARY INTO :IS-TEMP
     END-EXEC.

     IF IS-TEMP = 1
*      Logic for a temporary LOB goes here
       DISPLAY "LOB is temporary."
     ELSE
*      Logic for a persistent LOB goes here.
       DISPLAY "LOB is persistent."
     END-IF.

     EXEC SQL
          LOB FREE TEMPORARY :TEMP-BLOB
     END-EXEC.
     EXEC SQL FREE :TEMP-BLOB END-EXEC.
     STOP RUN.

 SQL-ERROR.
     EXEC SQL WHENEVER 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 が一時 LOB であるかどうかの確認

```
/* Checking if a LOB is temporary  [Example script: 3831.pc]
#include <oci.h>
#include <stdio.h>
#include <sqlca.h>

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

void lobIsTemp_proc()
{
  OCIBlobLocator *Temp_loc;
  int isTemporary = 0;

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  /* Allocate and Create the Temporary LOB: */
  EXEC SQL ALLOCATE :Temp_loc;
  EXEC SQL LOB CREATE TEMPORARY :Temp_loc;
  /* Determine if the Locator is a Temporary LOB Locator: */
  EXEC SQL LOB DESCRIBE :Temp_loc GET ISTEMPORARY INTO :isTemporary;

  /* Note that in this example, isTemporary should be 1 (TRUE) */
  if (isTemporary)
    printf("Locator is a Temporary LOB locator\n");
  /* Free the Temporary LOB: */
  EXEC SQL LOB FREE TEMPORARY :Temp_loc;
  /* Release resources held by the Locator: */
  EXEC SQL FREE :Temp_loc;
else
    printf("Locator is not a Temporary LOB locator \n");
   }

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  lobIsTemp_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

Java（JDBC）: BLOB が一時 BLOB であるかどうかの確認

JDBC アプリケーションは、isTemporary インスタンス・メソッドを使用して現在の BLOB オブジェクトが一時的かどうかを確認するか、または BLOB オブジェクトを isTemporary static メソッドに渡して指定された BLOB オブジェクトが一時的であるかどうかを確認します。これらの 2 つのメソッドの定義は次のとおりです。

```
/**
* Checking if a BLOB is temporary. [Example script: 3833.java]
* Returns true if LOB locator points to a temporary BLOB, False if not.
* @param lob the BLOB to test.
* @returns true if LOB locator points to a temporary BLOB, False if not.
*/
  public static boolean isTemporary (BLOB lob) throws SQLException

/**
 * Returns true if LOB locator points to a temporary BLOB, False if not.
 * @returns true if LOB locator points to a temporary BLOB, False if not.
 */
   public boolean isTemporary () throws SQLException

  //The usage example is--

  BLOB blob = ...

  // See if the BLOB is temporary
  boolean isTemporary = blob.isTemporary ();

  // See if the specified BLOB is temporary
  boolean isTemporary2 = BLOB.isTemporary(blob);
```

この JDBC API によって、以前の DBMS_LOB.isTemporary() を使用する必要がなくなります。

Java（JDBC）: CLOB が一時的であるかどうかの確認

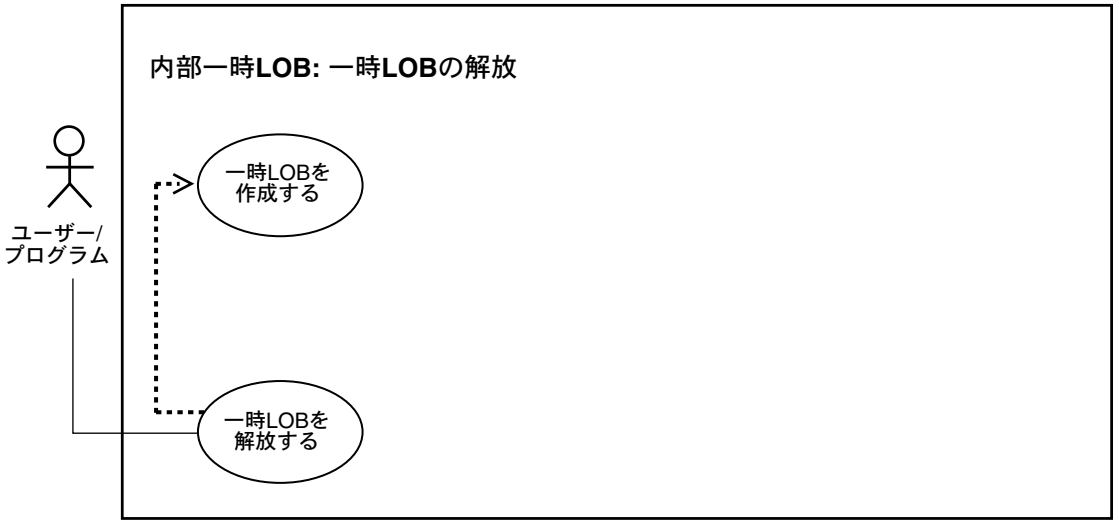
JDBC アプリケーションは、isTemporary インスタンス・メソッドを使用して現在の CLOB オブジェクトが一時的かどうかを確認するか、または CLOB オブジェクトを isTemporary static メソッドに渡して指定された CLOB オブジェクトが一時的であるかどうかを確認します。これらの 2 つのメソッドの定義は次のとおりです。

```
/**
 * Checking if LOB is temporary  [Example script: 3834.java]
 * Return true if the LOB locator points to a temporary CLOB, False if it
 * does not.
 *
 * @param lob the BLOB to test.
 * @return true if the LOB locator points to a temporary CLOB, False if it
 *         does not.
 */
 public static boolean isTemporary (CLOB lob) throws SQLException

 /**
  * Return true if the LOB locator points to a temporary CLOB, False if it
  * does not.
  *
  * @return true if the LOB locator points to a temporary CLOB, False if it
  *         does not.
  */
   public boolean isTemporary () throws SQLException

 //The usage example is--

 CLOB clob = ...

 // See if the CLOB is temporary
 boolean isTemporary = clob.isTemporary ();

 // See if the specified CLOB is temporary
 boolean isTemporary2 = CLOB.isTemporary(clob);
```

この API によって、以前の DBMS_LOB.isTemporary() を使用する必要がなくなります。

一時 LOB の解放

図 11-3 利用図 : 一時 LOB の解放



参照： 11-2 ページの表 11-1「利用モデル : 内部一時 LOB」を参照してください。

用途

一時 LOB を解放します。

使用上の注意

一時 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。

一時 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、一時 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関数またはファンクションのリストは、第 3 章「様々なプログラム環境での LOB のサポート」を参照してください。

- PL/SQL（DBMS_LOB）：『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』の第 23 章「DBMS_LOB」の「DBMS_LOB サブプログラムの要約」の「FREETEMPORARY プロシージャ」
- C（OCI）：『Oracle Call Interface プログラマーズ・ガイド』の第 16 章「その他の OCI リレーショナル関数」の「LOB 関数」の「OCILobFreeTemporary()」
- COBOL（Pro*COBOL）：『Pro*COBOL Precompiler プログラマーズ・ガイド』の LOB に関する詳細、LOB 文の使用上の注意および付録 F「埋込み SQL およびプリコンパイラ・ディレクティブ」の「LOB FREE TEMPORARY（実行可能埋込み SQL 拡張機能）」
- C/C++（Pro*C/C++）：『Pro*C/C++ Precompiler プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびディレクティブ」の「LOB FREE TEMPORARY（実行可能埋込み SQL 拡張機能）」
- Visual Basic（OO4O）：参照マニュアルはありません。
- Java（JDBC）：第 3 章「様々なプログラム環境での LOB のサポート」の表 3-50「JDBC: 一時 BLOB API」

  注意： temporaryClob.java クラスは、JDBC 標準では必要ないため、使用しないでください。

使用例

ありません。

例

次のプログラム環境での例を示します。

- PL/SQL（DBMS_LOB）：一時 LOB の解放（11-32 ページ）
- C（OCI）：一時 LOB の解放（11-32 ページ）
- COBOL（Pro*COBOL）：一時 LOB の解放（11-33 ページ）
- C/C++（Pro*C/C++）：一時 LOB の解放（11-34 ページ）
- Visual Basic（OO4O）：今回のリリースでは例は提供されません。
- Java（JDBC）：一時 BLOB の解放および Java（JDBC）：一時 CLOB の解放
- Java（JDBC）：TemporaryClob.java を使用した一時 CLOB の作成および解放（使用できない）

PL/SQL（DBMS_LOB）：一時 LOB の解放

```
/*  Freeing a temporary LOB  [Example script: 3836.sql]
DECLARE
   Dest_loc       BLOB;
   Src_loc        BFILE := BFILENAME('ADPHOTO_DIR', 'monitor_3060_11001');
   Amount         INTEGER := 4000;
BEGIN
   DBMS_LOB.CREATETEMPORARY(Dest_loc,TRUE);
   /* Opening the BFILE is mandatory: */
   DBMS_LOB.OPEN(Src_loc, DBMS_LOB.LOB_READONLY);

   /* Opening the LOB is optional: */
   DBMS_LOB.OPEN(Dest_loc,DBMS_LOB.LOB_READWRITE);
   DBMS_LOB.LOADFROMFILE(Dest_loc, Src_loc, Amount);

   /* Closing the LOB is mandatory if you have opened it: */
   DBMS_LOB.CLOSE(Src_loc);
   DBMS_LOB.CLOSE(Dest_loc);
   /* Free the temporary LOB: */
   DBMS_LOB.FREETEMPORARY(Dest_loc);
END;
```

C（OCI）：一時 LOB の解放

```
/* Freeing a temporary LOB.  [Example script: 3837.c]
   This function creates a temporary LOB and then frees it:
   This function returns 0 if it completes successfully, -1 otherwise:  */

sb4 freeTempLob(OCIError      *errhp,
                OCISvcCtx     *svchp,
                OCIStmt       *stmthp,
                OCIEnv        *envhp)
{
  OCILobLocator *tblob;
  checkerr (errhp,OCIDescriptorAlloc((dvoid*)envhp, (dvoid **)&tblob,
                                     (ub4)OCI_DTYPE_LOB, (size_t)0,
                                     (dvoid**)0));
  if(OCILobCreateTemporary(svchp,errhp,tblob,(ub2)0,SQLCS_IMPLICIT,
                           OCI_TEMP_BLOB, OCI_ATTR_NOCACHE,
                           OCI_DURATION_SESSION))
  {
      (void) printf("FAILED:CreateTemporary():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）：一時 LOB の解放

```
* Freeing a temporary LOB  [Example script: 3838.pco]
  IDENTIFICATION DIVISION.
  PROGRAM-ID. FREE-TEMPORARY.
  ENVIRONMENT DIVISION.
  DATA DIVISION.
  WORKING-STORAGE SECTION.

  01  USERID   PIC X(11) VALUES "SAMP/SAMP".

  01  TEMP-BLOB      SQL-BLOB.
  01  IS-TEMP        PIC S9(9) COMP.
  01  ORASLNRD        PIC 9(4).
      EXEC SQL INCLUDE SQLCA END-EXEC.
      EXEC ORACLE OPTION (ORACA=YES) END-EXEC.
      EXEC SQL INCLUDE ORACA END-EXEC.

  PROCEDURE DIVISION.
  FREE-TEMPORARY.

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

  * Allocate and initialize the BLOB locators:
      EXEC SQL ALLOCATE :TEMP-BLOB END-EXEC.
      EXEC SQL
           LOB CREATE TEMPORARY :TEMP-BLOB
      END-EXEC.

  * Do something with the temporary LOB here:
```

```
* Free the temporary LOB:
     EXEC SQL
          LOB FREE TEMPORARY :TEMP-BLOB
     END-EXEC.
     EXEC SQL FREE :TEMP-BLOB END-EXEC.
     STOP RUN.

 SQL-ERROR.
     EXEC SQL WHENEVER SQLERROR CONTINUE END-EXEC.
     MOVE ORASLNR TO ORASLNRD.
     DISPLAY " ".
     DISPLAY "ORACLE ERROR DETECTED ON LINE ", ORASLNRD, ":".
     DISPLAY " ".
     DISPLAY SQLERRMC.
     EXEC SQL ROLLBACK WORK RELEASE END-EXEC.
     STOP RUN.
```

C/C++（Pro*C/C++）: 一時 LOB の解放

```
/* Freeing a temporary LOB. [Example script: 3839.pc]
#include <oci.h>
#include <stdio.h>
#include <sqlca.h>

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

void freeTempLob_proc()
{
  OCIBlobLocator *Temp_loc;

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  EXEC SQL ALLOCATE :Temp_loc;
  EXEC SQL LOB CREATE TEMPORARY :Temp_loc;
  /* Do something with the Temporary LOB: */
  EXEC SQL LOB FREE TEMPORARY :Temp_loc;
  EXEC SQL FREE :Temp_loc;
}

void main()
{
```

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

Java（JDBC）：一時 BLOB の解放

JDBC アプリケーションは、freeTemporary インスタンス・メソッドを使用して現在の BLOB オブジェクトを解放するか、または解放する一時 BLOB を freeTemporary static メソッドに渡して指定された一時 BLOB を解放します。これらの 2 つのメソッドの定義は次のとおりです。

```
/**
* Freeing a temporary BLOB.  [Example script:  3840.java]
* This example frees the contents and locator of a temporary BLOB.
* @param temp_lob A temporary BLOB to be freed.
* @exception SQLException if temp_lob is a permanent LOB or temp_lob has already
* been freed.
*/

public static void freeTemporary (BLOB temp_lob) throws SQLException

/**
* Free the contents and the locator of the temporary BLOB.
* @exception SQLException if self is a permanent LOB or self has already been
freed.
*/

public void freeTemporary() throws SQLException

/**
* The usage example example is --
* BLOB tempBlob1 = ...
* BLOB tempBlob2 = ...
* // free the temporary BLOB
* tempBlob1.freeTemporary ();
* // free the specified temporary BLOB
* BLOB.freeTemporary(tempBlob2);
*
* The newer freeTemporary APIs should replace previous workaround of
* using dbms_lob.freeTemporary() in dbms_lob PL/SQL package.
*/
```

この freeTemporaryAPI によって、以前の dbms_lob.freeTemporary() を使用する必要がなくなります。

Java（JDBC）：一時 CLOB の解放

JDBC アプリケーションは、freeTemporary インスタンス・メソッドを使用して現在の CLOB オブジェクトを解放するか、または解放する一時 CLOB を freeTemporary static メソッドに渡して指定された一時 CLOB を解放します。これらの 2 つのメソッドの定義は次のとおりです。

```
/**
* Freeing a temporary CLOB.  [Example script: 3841.java]
* Free the contents and the locator of the temporary blob.
* @param temp_lob A temporary CLOB to be freed.
* @since 8.2.0
* @exception SQLException if temp_lob is a permanent LOB or temp_lob has
*            already been freed.
*/

public static void freeTemporary (CLOB temp_lob) throws SQLException

/**
* Free the contents and the locator of the temporary CLOB.
*
* @since 8.2.0
* @exception SQLException if self is a permanent lob or self has
*            already been freed.
*/
public void freeTemporary() throws SQLException


/**
* Use the free temporary CLOB API as follows:
* CLOB tempClob1 = ...
* CLOB tempClob2 = ...
* // free the temporary CLOB
* tempClob1.freeTemporary ();
* // free the specified temporary CLOB
* CLOB.freeTemporary(tempClob2);
*
*The freeTemporary API replaces previous workarounds that use
DBMS_LOB.freetemporary().
*/
```

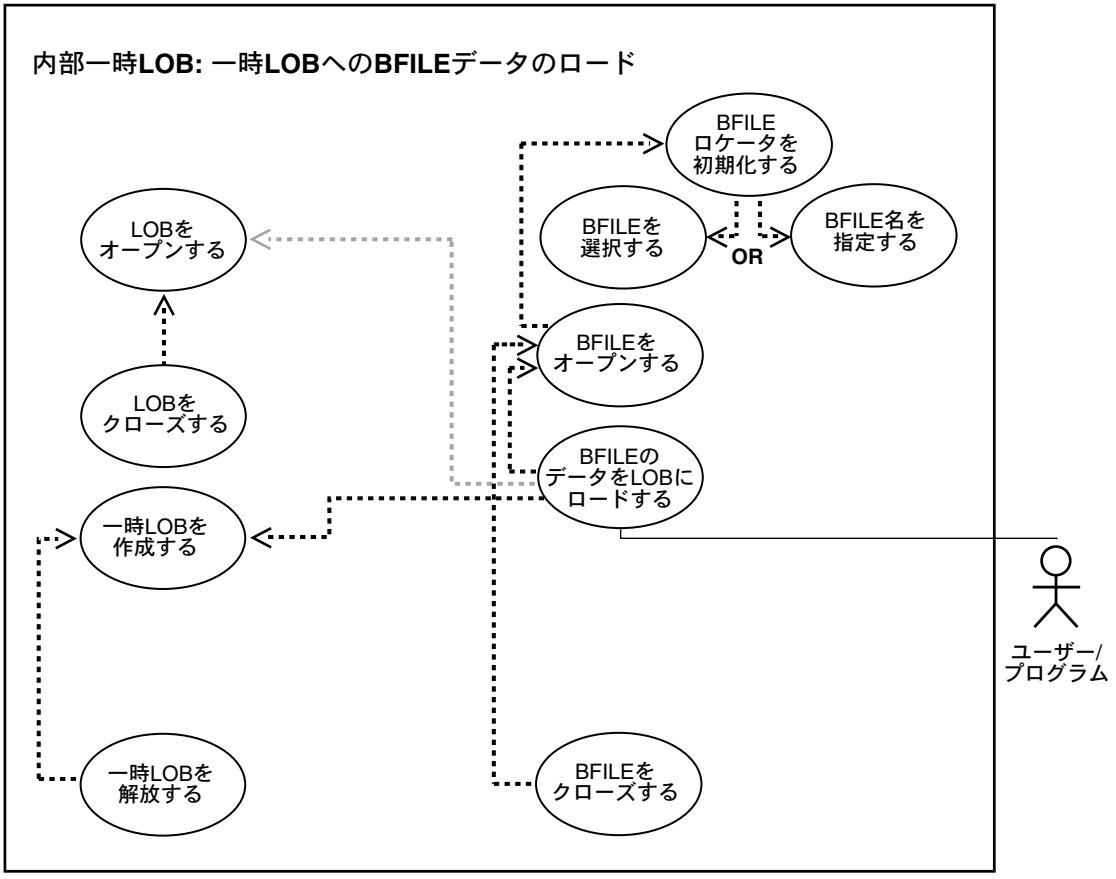
この新しい API によって、以前の DBMS_LOB.freeTemporary() を使用する必要がなくなります。

Java（JDBC）: TemporaryClob.java を使用した一時 CLOB の作成および解放

注意： temporaryClob.java クラスおよび temporaryBlob.java クラスは使用できません。JDBC 標準では、これらのクラスは必要ありません。

一時 LOB への BFILE データのロード

図 11-4 利用図 : LOB への BFILE データのロード



参照：

- 11-2 ページの表 11-1「利用モデル : 内部一時 LOB」を参照してください。
- 11-46 ページの「一時 BLOB への BFILE のバイナリ・データのロード」を参照してください。
- 11-50 ページの「一時 CLOB/NCLOB へのファイルのキャラクタ・データのロード」を参照してください。

用途

BFILE のデータを一時 LOB にロードします。

使用上の注意

注意： LOADBLOBFROMFILE および LOADCLOBFROMFILE プロシージャでは、この項のプロシージャの機能が実装され、バイナリ・データおよび文字データをロードする機能が改善されています。改善されたプロシージャは、PL/SQL 環境のみで使用可能です。可能なかぎり、改善されたプロシージャのいずれかを使用することをお薦めします。詳細は、11-46 ページの「一時 BLOB への BFILE のバイナリ・データのロード」および 11-50 ページの「一時 CLOB/NCLOB へのファイルのキャラクタ・データのロード」を参照してください。

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に、暗黙的に実行されます。DBMS_LOB.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。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。このため、テキストをロードする場合は LOADCLOBFROMFILE プロシージャを使用します。（11-50 ページの「一時 CLOB/NCLOB へのファイルのキャラクタ・データのロード」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関数またはファンクションのリストは、第 3 章「様々なプログラム環境での LOB のサポート」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）：『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』の第 23 章「DBMS_LOB」の「DBMS_LOB サブプログラムの要約」の「CREATETEMPORARY プロシージャ」および「FREETEMPORARY プロシージャ」

- C（OCI）：『Oracle Call Interface プログラマーズ・ガイド』の第 16 章「その他の OCI リレーショナル関数」の「LOB 関数」の「OCILobLoadFromFile()」
- COBOL（Pro*COBOL）：『Pro*COBOL Precompiler プログラマーズ・ガイド』の LOB に関する詳細、LOB 文の使用上の注意および付録 F「埋込み SQL およびプリコンパイラ・ディレクティブ」の「LOB LOAD（実行可能埋込み SQL 拡張機能）」
- C/C++（Pro*C/C++）：『Pro*C/C++ Precompiler プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびディレクティブ」の「LOB LOAD（実行可能埋込み SQL 拡張機能）」
- Visual Basic（OO4O）：参照マニュアルはありません。
- Java（JDBC）：10-40 ページの「Java（JDBC）：LOB への BFILE データのロード」

使用例

次のプロシージャの例では、ターゲットとなる LOB にロードする LOB データを含む OS のソース・ディレクトリ（ADPHOTO_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での例を示します。

- PL/SQL（DBMS_LOB）：一時 LOB への BFILE データのロード（11-40 ページ）
- C（OCI）：一時 LOB への BFILE データのロード（11-41 ページ）
- COBOL（Pro*COBOL）：一時 LOB への BFILE データのロード（11-43 ページ）
- C/C++（Pro*C/C++）：一時 LOB への BFILE データのロード（11-44 ページ）
- Visual Basic（OO4O）：今回のリリースでは例は提供されません。
- Java（JDBC）：今回のリリースでは例は提供されません。

PL/SQL（DBMS_LOB）：一時 LOB への BFILE データのロード

```
/* Loading a temporary LOB with data from a BFILE.
   Example procedure freeTempLob_proc is not part of DBMS_LOB package: */

CREATE or REPLACE PROCEDURE freeTempLob_proc(Lob_loc IN OUT BLOB) IS
BEGIN
   DBMS_LOB.CREATETEMPORARY(Lob_loc,TRUE);
   /* Use the temporary LOB locator here, then free it.*/
   /* Free the temporary LOB locator: */
   DBMS_LOB.FREETEMPORARY(Lob_loc);
   DBMS_OUTPUT.PUT_LINE('Temporary LOB was freed');
END;
```

C（OCI）：一時 LOB への BFILE データのロード

```
/* Loading a temporary LOB with data from a BFILE.
   This section of code shows you how to create a temporary LOB, and load
   the contents of a BFILE into that temporary LOB: */

sb4 load_temp(OCIError *errhp,
              OCISvcCtx *svchp,
              OCIStmt   *stmthp,
              OCIEnv    *envhp)
{
  OCILobLocator *bfile;
  int amount =100;
  OCILobLocator *tblob;

  printf("in load_temp\n");
  if(OCIDescriptorAlloc((dvoid*)envhp, (dvoid **)&tblob,
                        (ub4)OCI_DTYPE_LOB, (size_t)0, (dvoid**)0))
  {
    printf("OCIDescriptorAlloc failed in load_temp\n");
    return -1;
  }
  if(OCIDescriptorAlloc((dvoid*)envhp, (dvoid **)&bfile,
                        (ub4)OCI_DTYPE_FILE, (size_t)0, (dvoid**)0))
  {
    printf("OCIDescriptorAlloc failed in load_temp\n");
    return -1;
  }

  /* Create a temporary LOB: */
  if(OCILobCreateTemporary(svchp, errhp, tblob, (ub2)0,
                           SQLCS_IMPLICIT, OCI_TEMP_BLOB,
                           OCI_ATTR_NOCACHE, OCI_DURATION_SESSION))
  {
    (void) printf("FAILED: CreateTemporary() \n");
    return -1;
  }

  if(OCILobFileSetName(envhp, errhp, &bfile, (text *)"ADPHOTO_DIR",
                       (ub2)strlen("ADPHOTO_DIR"), (text *)"monitor_3060_11001",
                       (ub2)strlen("monitor_3060_11001")))
  {
    printf("OCILobFileSetName FAILED in load_temp\n");
    return -1;
  }

   /* Opening the BFILE is mandatory: */
```

```
 if (OCILobFileOpen(svchp, errhp, (OCILobLocator *) bfile, OCI_LOB_READONLY))
{
  printf( "OCILobFileOpen FAILED for the bfile load_temp \n");
  return -1;
}

 /* Opening the LOB is optional: */
 if (OCILobOpen(svchp, errhp, (OCILobLocator *) tblob, OCI_LOB_READWRITE))
{
  printf( "OCILobOpen FAILED for temp LOB \n");
  return -1;
}

if(OCILobLoadFromFile(svchp,
       errhp,
       tblob,
       (OCILobLocator*)bfile,
       (ub4)amount,
       (ub4)1,(ub4)1))
{
  printf( "OCILobLoadFromFile FAILED\n");
  return -1;
}

/* Close the lobs: */
if (OCILobFileClose(svchp, errhp, (OCILobLocator *) bfile))
{
  printf( "OCILobClose FAILED for bfile \n");
  return -1;
}

checkerr(errhp,(OCILobClose(svchp, errhp, (OCILobLocator *) tblob)));


/* Free the temporary LOB now that we are done using it */
if(OCILobFreeTemporary(svchp, errhp, tblob))
{
  printf("OCILobFreeTemporary FAILED \n");
  return -1;
}
}
```

COBOL（Pro*COBOL）：一時 LOB への BFILE データのロード

```
* Loading a temporary LOB with data from a BFILE.
  IDENTIFICATION DIVISION.
  PROGRAM-ID. LOAD-TEMPORARY.
  ENVIRONMENT DIVISION.
  DATA DIVISION.
  WORKING-STORAGE SECTION.

  01  USERID   PIC X(11) VALUES "USER1/USER1".
  01  TEMP-BLOB      SQL-BLOB.
  01  SRC-BFILE      SQL-BFILE.
  01  DIR-ALIAS      PIC X(30) VARYING.
  01  FNAME          PIC X(20) VARYING.
  01  DIR-IND        PIC S9(4) COMP.
  01  FNAME-IND      PIC S9(4) COMP.
  01  AMT            PIC S9(9) COMP VALUE 10.
  01  ORASLNRD        PIC 9(4).

      EXEC SQL INCLUDE SQLCA END-EXEC.
      EXEC ORACLE OPTION (ORACA=YES) END-EXEC.
      EXEC SQL INCLUDE ORACA END-EXEC.

  PROCEDURE DIVISION.
  LOAD-TEMPORARY.

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

 * Allocate and initialize the BFILE and BLOB locators:
      EXEC SQL ALLOCATE :SRC-BFILE END-EXEC.
      EXEC SQL ALLOCATE :TEMP-BLOB END-EXEC.
      EXEC SQL
           LOB CREATE TEMPORARY :TEMP-BLOB
      END-EXEC.

 * Set up the directory and file information:
      MOVE "ADPHOTO_DIR" TO DIR-ALIAS-ARR.
      MOVE 9 TO DIR-ALIAS-LEN.
      MOVE "keyboard_photo3106_13001" TO FNAME-ARR.
      MOVE 16 TO FNAME-LEN.

      EXEC SQL
         LOB FILE SET :SRC-BFILE DIRECTORY = :DIR-ALIAS,
         FILENAME = :FNAME
```

```
           END-EXEC.

      * Open source BFILE and destination temporary BLOB:
           EXEC SQL LOB OPEN :TEMP-BLOB READ WRITE END-EXEC.
           EXEC SQL LOB OPEN :SRC-BFILE READ ONLY END-EXEC.

           EXEC SQL
                LOB LOAD :AMT FROM FILE :SRC-BFILE INTO :TEMP-BLOB
           END-EXEC.

      * Close the LOBs:
           EXEC SQL LOB CLOSE :SRC-BFILE END-EXEC.
           EXEC SQL LOB CLOSE :TEMP-BLOB END-EXEC.

      * Free the temporary LOB:
           EXEC SQL
                LOB FREE TEMPORARY :TEMP-BLOB
           END-EXEC.

      * And free the LOB locators:
           EXEC SQL FREE :TEMP-BLOB END-EXEC.
           EXEC SQL FREE :SRC-BFILE END-EXEC.
           STOP RUN.

       SQL-ERROR.
           EXEC SQL
               WHENEVER SQLERROR CONTINUE
           END-EXEC.
           MOVE ORASLNR TO ORASLNRD.
           DISPLAY " ".
           DISPLAY "ORACLE ERROR DETECTED ON LINE ", ORASLNRD, ":".
           DISPLAY " ".
           DISPLAY SQLERRMC.
           EXEC SQL
               ROLLBACK WORK RELEASE
           END-EXEC.
           STOP RUN.
```

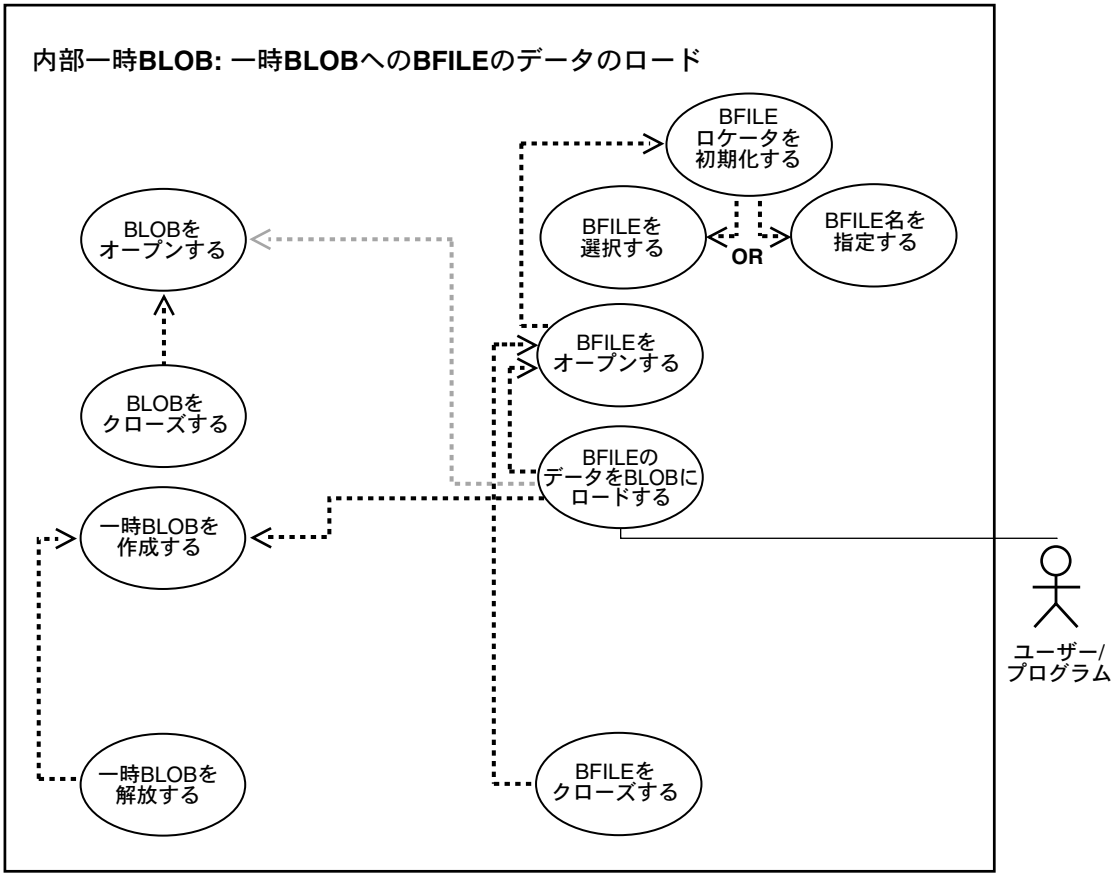
C/C++（Pro*C/C++）: 一時 LOB への BFILE データのロード

```
/* Loading a temporary LOB with data from a BFILE. */
#include <oci.h>
#include <stdio.h>
#include <sqlca.h>

void Sample_Error()
```

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

void loadTempLobFromBFILE_proc()
{
  OCIBlobLocator *Temp_loc;
  OCIBFileLocator *Lob_loc;
  char *Dir = "ADPHOTO_DIR", *Name = "monitor_photo_3060_11001";
  int Amount = 4096;

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  /* Allocate and Create the Temporary LOB: */
  EXEC SQL ALLOCATE :Temp_loc;
  EXEC SQL LOB CREATE TEMPORARY :Temp_loc;
  /* Allocate and Initialize the BFILE Locator: */
  EXEC SQL ALLOCATE :Lob_loc;
  EXEC SQL LOB FILE SET :Lob_loc DIRECTORY = :Dir, FILENAME = :Name;

  /* Opening the BFILE is mandatory; */
  /* Opening the LOB is optional: */
  EXEC SQL LOB OPEN :Lob_loc READ ONLY;
  EXEC SQL LOB OPEN :Temp_loc READ WRITE;
  /* Load the data from the BFILE into the Temporary LOB: */
  EXEC SQL LOB LOAD :Amount FROM FILE :Lob_loc INTO :Temp_loc;
  /* Closing the LOBs is Mandatory if they have been Opened: */
  EXEC SQL LOB CLOSE :Temp_loc;
  EXEC SQL LOB CLOSE :Lob_loc;
  /* Free the Temporary LOB: */
  EXEC SQL LOB FREE TEMPORARY :Temp_loc;
  /* Release resources held by the Locators: */
  EXEC SQL FREE :Temp_loc;
  EXEC SQL FREE :Lob_loc;
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  loadTempLobFromBFILE_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
)
```

一時 BLOB への BFILE のバイナリ・データのロード

図 11-5 利用図 : 一時 BLOB への BFILE データのロード



参照：

- 11-2 ページの表 11-1「利用モデル : 内部一時 LOB」を参照してください。
- 11-50 ページの「一時 CLOB/NCLOB へのファイルのキャラクタ・データのロード」を参照してください。
- 11-38 ページの「一時 LOB への BFILE データのロード」を参照してください。

用途

BFILE のバイナリ・データを一時 BLOB にロードします。

使用上の注意

バイナリ・データをロードする場合は LOADBLOBFROMFILE を使用し、テキストをロードする場合は LOADCLOBFROMFILE を使用します。結果は、LOADFROMFILE を使用した場合と同じになります。また、ユーザーに新しいオフセットが戻されます。LOADCLOBFROMFILE API を使用すると、BFILE のキャラクタ・セット ID を指定できるため、BFILE データのキャラクタ・セットから宛先 CLOB/NCLOB のキャラクタ・セットへの変換が適切に実行されます。

構文

各プログラム環境で使用可能な関数またはファンクションのリストは、第 3 章「様々なプログラム環境での LOB のサポート」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）：『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』の第 23 章「DBMS_LOB」の「DBMS_LOB サブプログラムの要約」の「LOADBLOBFROMFILE プロシージャ」

使用例

この項のプロシージャの例では、製品メディアのサンプル・スキーマの Print_media 表を使用します。また、ターゲットとなる BLOB にロードするバイナリ LOB データを含む OS のソース・ディレクトリが存在することを想定しています。

例

「PL/SQL（DBMS_LOB）: 一時 BLOB への BFILE データのロード」の例では、PL/SQL プログラム環境での LOADBLOBFROMFILE の使用方法を示します。（他のプログラム環境はサポートされていません。）

PL/SQL（DBMS_LOB）：一時 BLOB への BFILE データのロード

次の項目の例を示します。

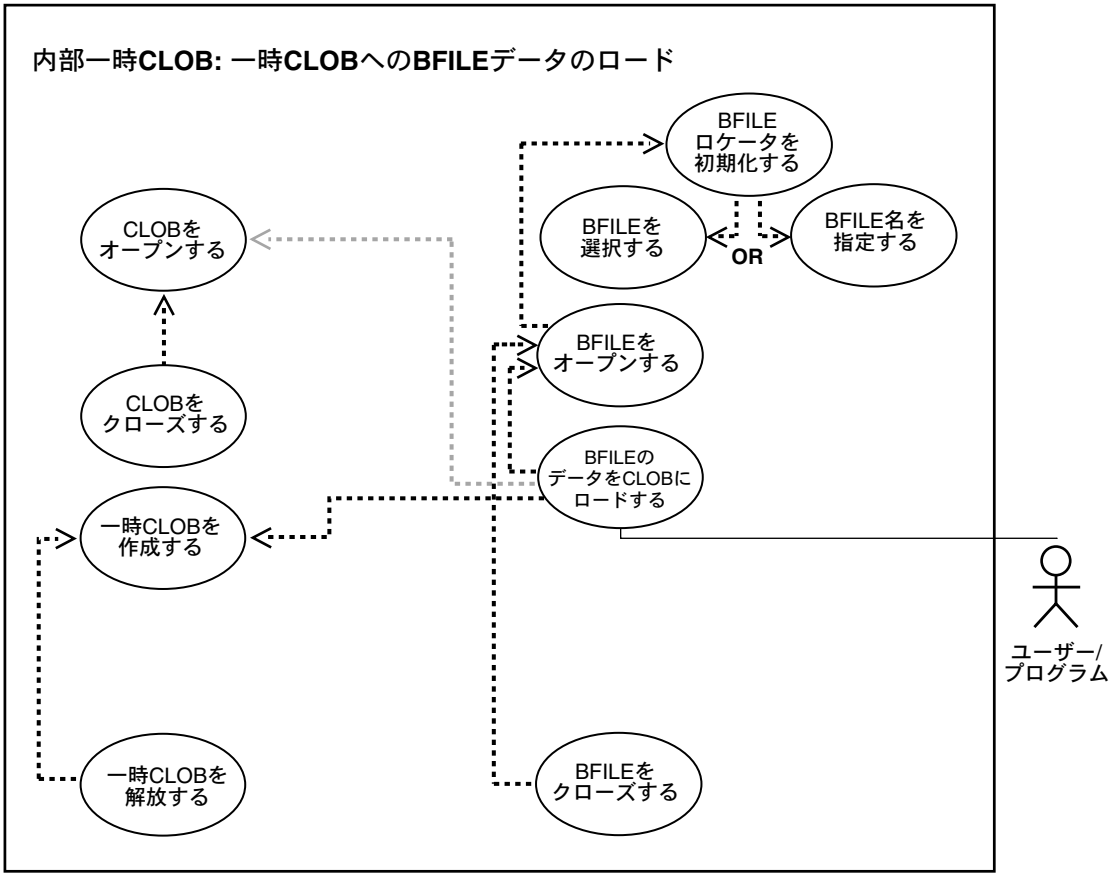
- BFILE のバイナリ・データを一時 BLOB にロードする方法。
- 最初に BFILE の長さを取得することなく BLOB への BFILE データをロードする方法。
- LOADBLOBFROMFILE によって返されたソース・オフセットおよび宛先オフセット値を使用して BLOB のサイズを決定する方法。

```
DECLARE
  src_loc     BFILE := bfilename('ADPHOTO_DIR','monitor_3060_11001');
  dst_loc     BLOB;
  src_offset  NUMBER := 1;
  dst_offset  NUMBER := 1;
  src_osin    NUMBER;
  dst_osin    NUMBER;
  bytes_rd    NUMBER;
  bytes_wt    NUMBER;
BEGIN
  dbms_lob.createtemporary(dst_loc, TRUE);

  /* Opening the source BFILE is mandatory. */
  dbms_lob.fileopen(src_loc, dbms_lob.file_readonly);

  /* Opening the LOB is optional. */
  dbms_lob.open(dst_loc, dbms_lob.lob_readwrite);
  /* Save the input source/destination offsets. */
  src_osin := src_offset;
  dst_osin := dst_offset;

  /* Use LOBMAXSIZE to indicate loading the entire BFILE. */
  dbms_lob.LOADBLOBFROMFILE(dst_loc, src_loc, dbms_lob.lobmaxsize, src_offset,
      dst_offset) ;

  /* Closing the LOB is mandatory if you have opened it */
  dbms_lob.close(dst_loc);
  dbms_lob.filecloseall();

  /* Use the src_offset returned to calculate the actual
   * amount read from the BFILE.
   */
  bytes_rd := src_offset - src_osin;
  dbms_output.put_line(' Number of bytes read from the BFILE ' || bytes_rd ) ;

  /* Use the dst_offset returned to calculate the actual
   * amount written to the BLOB.
   */
```

```
  bytes_wt := dst_offset - dst_osin;
  dbms_output.put_line(' Number of bytes written to the BLOB ' || bytes_wt ) ;

  /* If there is no exception the number of bytes read
   * should equal to the number of bytes written.
   */

  /* Free the temporary LOB. */
  dbms_lob.freetemporary(dst_loc);

END ;
```

一時 CLOB/NCLOB へのファイルのキャラクタ・データのロード

図 11-6 利用図 : CLOB または NCLOB への BFILE データのロード



参照：

- 11-2 ページの表 11-1「利用モデル : 内部一時 LOB」を参照してください。
- 11-46 ページの「一時 BLOB への BFILE のバイナリ・データのロード」を参照してください。
- 11-38 ページの「一時 LOB への BFILE データのロード」を参照してください。

用途

BFILE のキャラクタ・データを一時 CLOB または NCLOB にロードします。

使用上の注意

バイナリ・データをロードする場合は LOADBLOBFROMFILE を使用し、テキストをロードする場合は LOADCLOBFROMFILE を使用します。後者のメソッドでは BFILE のキャラクタ・セット ID を指定できます。LOADCLOBFROMFILE API を使用すると、BFILE のキャラクタ・セット ID を指定できるため、BFILE データ・キャラクタ・セットから宛先 CLOB/NCLOB のキャラクタ・セットへの変換が適切に実行されます。

構文

各プログラム環境で使用可能な関数またはファンクションのリストは、第 3 章「様々なプログラム環境での LOB のサポート」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）：『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』の第 23 章「DBMS_LOB」の「DBMS_LOB サブプログラムの要約」の「LOADCLOBFROMFILE プロシージャ」

使用例

この項のプロシージャの例では、製品メディアのサンプル・スキーマの Print_media 表を使用します。また、ターゲットとなる CLOB または NCLOB にロードするキャラクタ LOB データを含む OS のソース・ディレクトリが存在することを想定しています。

例

「PL/SQL（DBMS_LOB）: 一時 CLOB/NCLOB への BFILE データのロード」の例では、PL/SQL プログラム環境での LOADCLOBFROMFILE の使用方法を示します。（他のプログラム環境はサポートされていません。）

PL/SQL（DBMS_LOB）：一時 CLOB/NCLOB への BFILE データのロード

例

次の項目の例を示します。

- 一時 CLOB または NCLOB への BFILE のキャラクタ・データのロード方法。
- BFILE の getlength を呼び出さずにファイル全体をロードする方法。
- 返されたオフセットを使用して実際のロード量を求める方法。

この例では、ad_source が UTF8 キャラクタ・セット形式の BFILE であり、データベース・キャラクタ・セットが UTF8 であることを想定しています。

```
DECLARE
  src_loc     bfile := bfilename('ADVERT_DIR','ad_source_1000');
  dst_loc     clob  ;
  amt         number := dbms_lob.lobmaxsize;
  src_offset  number := 1 ;
  dst_offset  number := 1 ;
  lang_ctx    number := dbms_lob.default_lang_ctx;
  warning     number;
BEGIN
  dbms_lob.createtemporary(dst_loc, TRUE);
  dbms_lob.fileopen(src_loc, dbms_lob.file_readonly);

  /* The default_csid can be used when the BFILE encoding is in the same charset
   * as the destination CLOB/NCLOB charset
   */

  dbms_lob.LOADCLOBFROMFILE(dst_loc, src_loc, amt, dst_offset, src_offset,
      dbms_lob.default_csid, lang_ctx,warning) ;
  commit ;

  dbms_output.put_line(' Amount specified ' || amt ) ;
  dbms_output.put_line(' Number of bytes read from source: ' ||
      (src_offset-1));
  dbms_output.put_line(' Number of characters written to destination: ' ||
      (dst_offset-1) );
  if (warning = dbms_lob.warn_inconvertible_char)
  then
    dbms_output.put_line('Warning: Inconvertible character');
  end if;

  dbms_lob.filecloseall() ;
  dbms_lob.freetemporary(dst_loc);
END ;
```

例

次の項目の例を示します。

- NLS_CHARSET_ID ファンクションを使用してキャラクタ・セット名からキャラクタセット ID を取得する方法。
- 返されたオフセット値および言語のコンテキスト lang_ctx を使用して単一の BFILE から異なる LOB にデータ・ストリームをロードする方法。
- 警告メッセージを読み込む方法。

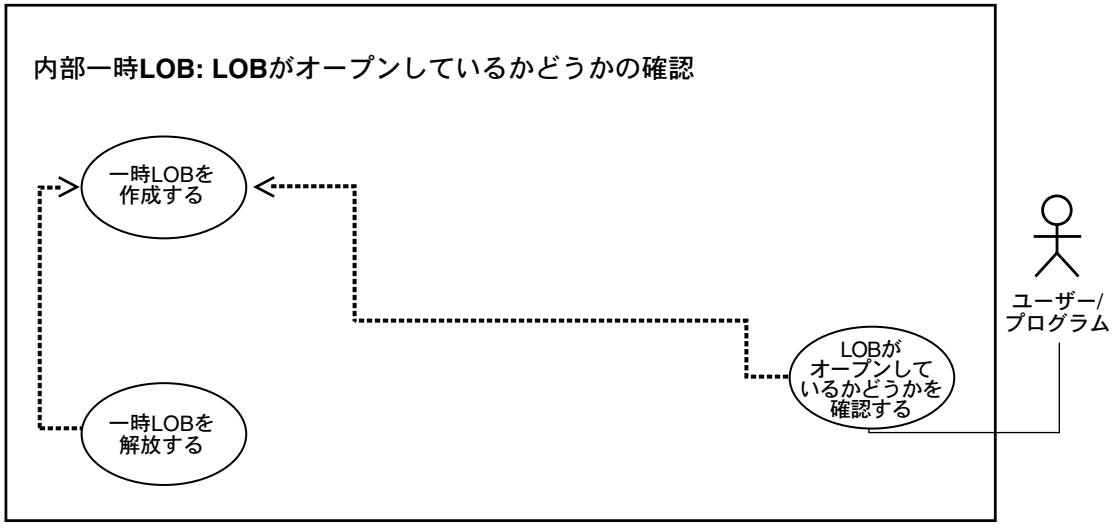
この例では、ad_file_ext_01 が JA16TSTSET 形式の BFILE であり、データベースの各国語キャラクタセットが AL16UTF16 であることを想定しています。

```
DECLARE
  src_loc     bfile := bfilename('ADVERT_DIR','ad_file_ext_01') ;
  dst_loc1    nclob;
  dst_loc2    nclob;
  amt         number := 1000;
  src_offset  number := 1;
  dst_offset  number := 1;
  src_osin    number;
  cs_id       number := NLS_CHARSET_ID('JA16TSTSET'); /* 998 */
  lang_ctx    number := dbms_lob.default_lang_ctx;
  warning     number;
BEGIN
  dbms_lob.fileopen(src_loc, dbms_lob.file_readonly);
  dbms_output.put_line(' BFILE csid is ' || cs_id) ;

  /* Load the first 1KB of the BFILE into dst_loc1 */
  dbms_output.put_line(' ----------------------------' ) ;
  dbms_output.put_line('   First load  ' ) ;
  dbms_output.put_line(' ----------------------------' ) ;

  dbms_lob.createtemporary(dst_loc1, TRUE);

  dbms_lob.LOADCLOBFROMFILE(dst_loc1, src_loc, amt, dst_offset, src_offset,
      cs_id, lang_ctx,warning);
  commit;

  /* The number bytes read may or may not be 1k */
  dbms_output.put_line(' Amount specified ' || amt ) ;
  dbms_output.put_line(' Number of bytes read from source: ' ||
      (src_offset-1) );
  dbms_output.put_line(' Number of characters written to destination: ' ||
      (dst_offset-1) );
  IF (warning = dbms_lob.warn_inconvertible_char)
  then
```

```
    dbms_output.put_line('Warning: Inconvertible character');
  END IF;

  /* load the next 1KB of the BFILE into the dst_loc2 */
  dbms_output.put_line(' ----------------------------' ) ;
  dbms_output.put_line('   Second load  ' ) ;
  dbms_output.put_line(' ----------------------------' ) ;

  dbms_lob.createtemporary(dst_loc2, TRUE);
  /* Notice we are using the src_offset and lang_ctx returned from the previous
   * load. We do not use value 1001 as the src_offset here because sometimes the
   * actual amount read may not be the same as the amount specified.
   */
  src_osin := src_offset;
  dst_offset := 1;
  dbms_lob.LOADCLOBFROMFILE(dst_loc2, src_loc, amt, dst_offset, src_offset,
      cs_id, lang_ctx, warning);
  commit;
  dbms_output.put_line(' Number of bytes read from source: ' ||
      (src_offset-src_osin) );
  dbms_output.put_line(' Number of characters written to destination: ' ||
      (dst_offset-1) );
  if (warning = dbms_lob.warn_inconvertible_char)
  then
    dbms_output.put_line('Warning: Inconvertible character');
  END IF;

  dbms_lob.filecloseall() ;
  dbms_lob.freetemporary(dst_loc1);
  dbms_lob.freetemporary(src_loc2);

END ;
```

一時 LOB がオープンしているかどうかの確認

図 11-7 利用図 : 一時 LOB がオープンしているかどうかの確認



参照： 11-2 ページの表 11-1「利用モデル : 内部一時 LOB」を参照してください。

用途

一時 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関数またはファンクションのリストは、第 3 章「様々なプログラム環境での LOB のサポート」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）:『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』の第 23 章「DBMS_LOB」の「DBMS_LOB サブプログラムの要約」の「ISOPEN ファンクション」

- C（OCI）：『Oracle Call Interface プログラマーズ・ガイド』の第 16 章「その他の OCI リレーショナル関数」の「LOB 関数」の「OCILobIsOpen()」
- COBOL（Pro*COBOL）：『Pro*COBOL Precompiler プログラマーズ・ガイド』の LOB に関する詳細、LOB 文の使用上の注意および付録 F「埋込み SQL およびプリコンパイラ・ディレクティブ」の「LOB DESCRIBE（実行可能埋込み SQL 拡張機能）」
- C/C++（Pro*C/C++）：『Pro*C/C++ Precompiler プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびディレクティブ」の「LOB DESCRIBE（実行可能埋込み SQL 拡張機能）」
- Visual Basic（OO4O）：参照マニュアルはありません。
- Java（JDBC）：10-57 ページの「Java（JDBC）：LOB がオープンしているかどうかの確認」

使用例

次の例では、ロケータを入力として取り、一時 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での例を示します。

- PL/SQL（DBMS_LOB）：一時 LOB がオープンしているかどうかの確認（11-56 ページ）
- C（OCI）：一時 LOB がオープンしているかどうかの確認（11-57 ページ）
- COBOL（Pro*COBOL）：一時 LOB がオープンしているかどうかの確認（11-58 ページ）
- C/C++（Pro*C/C++）：一時 LOB がオープンしているかどうかの確認（11-59 ページ）
- Visual Basic（OO4O）：今回のリリースでは例は提供されません。
- Java（JDBC）：今回のリリースでは例は提供されません。

PL/SQL（DBMS_LOB）：一時 LOB がオープンしているかどうかの確認

```
/* Determining if a temporary LOB is open.
   Example procedure seeTempLOBIsOpen_proc is not part of
   DBMS_LOB package. This procedure takes a locator as input, creates a
   temporary LOB, opens it and tests if the LOB is open. */

CREATE OR REPLACE PROCEDURE seeTempLOBIsOpen_proc(Lob_loc IN OUT BLOB,
                          Retval OUT INTEGER) IS
BEGIN
   /* Create the temporary LOB: */
   DBMS_LOB.CREATETEMPORARY(Lob_loc,TRUE);
   /* See If the LOB is open: */
   Retval := DBMS_LOB.ISOPEN(Lob_loc);
```

```
  /* The value of Retval will be 1  if  the LOB is open. */
   /* Free the temporary LOB: */
   DBMS_LOB.FREETEMPORARY(Lob_loc);
END;
```

C（OCI）：一時 LOB がオープンしているかどうかの確認

```
/* Determining if a temporary LOB is open.
   This function takes a locator and returns 0 if the function
   completes successfully. The function prints out "Temporary LOB is open" or
  "Temporary LOB is closed". It does not check whether or not the locator is
   actually pointing to a temporary LOB or not, but the open or close test will
   work either way. The function returns 0 if it completes
   successfully, and -1 if it fails. */

sb4 seeTempLOBIsOpen (OCILobLocator *lob_loc,
                      OCIError      *errhp,
                      OCISvcCtx     *svchp,
                      OCIStmt       *stmthp,
                      OCIEnv        *envhp)
{
    boolean is_open = FALSE;

    printf("in seeTempLOBIsOpen \n");

    if(OCILobCreateTemporary(svchp,
              errhp,
              lob_loc,
              (ub2)0,
              SQLCS_IMPLICIT,
              OCI_TEMP_BLOB,
              OCI_ATTR_NOCACHE,
              OCI_DURATION_SESSION))
    {
        (void) printf("FAILED: CreateTemporary() \n");
        return -1;
    }

    if(OCILobIsOpen(svchp, errhp, lob_loc, &is_open))
    {
        printf("OCILobIsOpen FAILED\n");
        return -1;
    }
    if(is_open)
    {
        printf("Temporary LOB is open\n");
```

```
}else
{
    printf("Temporary LOB is closed\n");

}

if(OCILobFreeTemporary(svchp,errhp,lob_loc))
{
    printf("OCILobFreeTemporary FAILED \n");
    return -1;
}
return 0;
}
```

COBOL（Pro*COBOL）: 一時 LOB がオープンしているかどうかの確認

```
 * Determining if a temporary LOB is open. [Example script: 3850.pco]
   IDENTIFICATION DIVISION.
   PROGRAM-ID. TEMP-LOB-ISOPEN.
   ENVIRONMENT DIVISION.
   DATA DIVISION.
   WORKING-STORAGE SECTION.

   01  USERID   PIC X(11) VALUES "SAMP/SAMP".
   01  TEMP-BLOB      SQL-BLOB.
   01  SRC-BFILE      SQL-BFILE.
   01  DIR-ALIAS      PIC X(30) VARYING.
   01  FNAME          PIC X(20) VARYING.
   01  DIR-IND        PIC S9(4) COMP.
   01  FNAME-IND      PIC S9(4) COMP.
   01  AMT            PIC S9(9) COMP.
   01  IS-OPEN        PIC S9(9) COMP.
   01  ORASLNRD        PIC 9(4).

       EXEC SQL INCLUDE SQLCA END-EXEC.
       EXEC ORACLE OPTION (ORACA=YES) END-EXEC.
       EXEC SQL INCLUDE ORACA END-EXEC.
   PROCEDURE DIVISION.
   TEMP-LOB-ISOPEN.
       EXEC SQL WHENEVER SQLERROR DO PERFORM SQL-ERROR END-EXEC.
       EXEC SQL
            CONNECT :USERID
       END-EXEC.

  * Allocate and initialize the BLOB locators:
```

```
           EXEC SQL ALLOCATE :TEMP-BLOB END-EXEC.
           EXEC SQL
                LOB CREATE TEMPORARY :TEMP-BLOB
           END-EXEC.

       * Open temporary LOB:
           EXEC SQL LOB OPEN :TEMP-BLOB READ ONLY END-EXEC.
           EXEC SQL
                LOB DESCRIBE :TEMP-BLOB GET ISOPEN INTO :IS-OPEN
           END-EXEC.

           IF IS-OPEN = 1
       *      Logic for an open temporary LOB goes here:
              DISPLAY "Temporary LOB is OPEN."
           ELSE
       *      Logic for a closed temporary LOB goes here:
              DISPLAY "Temporary LOB is CLOSED."
           END-IF.
           EXEC SQL
                ROLLBACK WORK RELEASE
           END-EXEC.
           STOP RUN.

        SQL-ERROR.
           EXEC SQL
               WHENEVER SQLERROR CONTINUE
           END-EXEC.
           MOVE ORASLNR TO ORASLNRD.
           DISPLAY " ".
           DISPLAY "ORACLE ERROR DETECTED ON LINE ", ORASLNRD, ":".
           DISPLAY " ".
           DISPLAY SQLERRMC.
           EXEC SQL
               ROLLBACK WORK RELEASE
           END-EXEC.
           STOP RUN.
```

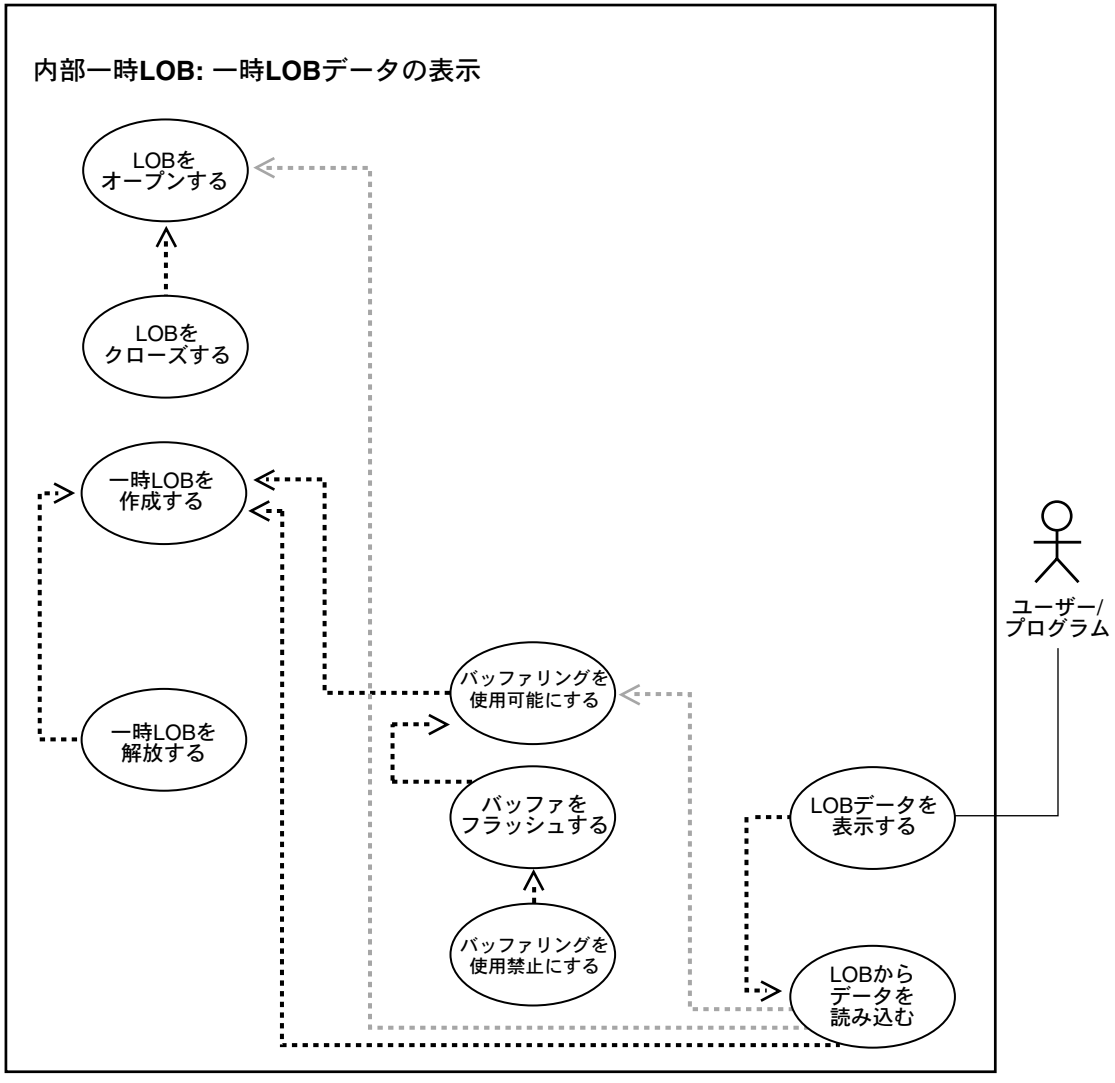
C/C++（Pro*C/C++）：一時 LOB がオープンしているかどうかの確認

```
/* Determining if a temporary LOB is open. [Example script: 3851.pc] */
#include <oci.h>
#include <stdio.h>
#include <sqlca.h>

void Sample_Error()
{
```

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

void tempLobIsOpen_proc()
{
  OCIBlobLocator *Temp_loc;
  int isOpen = 0;

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  /* Allocate and Create the Temporary LOB */
  EXEC SQL ALLOCATE :Temp_loc;
  EXEC SQL LOB CREATE TEMPORARY :Temp_loc;
  /* Open the Temporary LOB */
  EXEC SQL LOB OPEN :Temp_loc READ ONLY;
  /* Determine if the LOB is Open */
  EXEC SQL LOB DESCRIBE :Temp_loc GET ISOPEN INTO :isOpen;
  if (isOpen)
    printf("Temporary LOB is open\n");
  else
    printf("Temporary LOB is not open\n");
  /* Note that in this example, the LOB is Open so isOpen == 1 (TRUE) */
  /* Close the LOB */
  EXEC SQL LOB CLOSE :Temp_loc;
  /* Free the Temporary LOB */
  EXEC SQL LOB FREE TEMPORARY :Temp_loc;
  /* Release resources held by the Locator */
  EXEC SQL FREE :Temp_loc;
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  tempLobIsOpen_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

一時 LOB データの表示

図 11-8 利用図 : 一時 LOB データの表示



参照： 11-2 ページの表 11-1「利用モデル : 内部一時 LOB」を参照してください。

用途

一時 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関数またはファンクションのリストは、第 3 章「様々なプログラム環境での LOB のサポート」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）：『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』の第 23 章「DBMS_LOB」の「DBMS_LOB 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 43 章「DBMS_OUTPUT」の「DBMS_OUTPUT 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）：『Oracle Call Interface プログラマーズ・ガイド』の第 16 章「その他の OCI リレーショナル関数」の「LOB 関数」の「OCILobRead()」
- COBOL（Pro*COBOL）：『Pro*COBOL Precompiler プログラマーズ・ガイド』の LOB に関する詳細、LOB 文の使用上の注意および付録 F「埋込み SQL およびプリコンパイラ・ディレクティブ」の「LOB READ（実行可能埋込み SQL 拡張機能）」
- C/C++（Pro*C/C++）：『Pro*C/C++ Precompiler プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびディレクティブ」の「LOB READ（実行可能埋込み SQL 拡張機能）」
- Visual Basic（OO4O）: 参照マニュアルはありません。
- Java（JDBC）: 10-101 ページの「Java（JDBC）: LOB データの表示」

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 `Adheader_typ` からクライアント側へのイメージ `monitor_photo` のストリーム読込みを行います。

例

次のプログラム環境での例を示します。

- PL/SQL（DBMS_LOB）：一時 LOB データの表示（11-63 ページ）
- C（OCI）：一時 LOB データの表示（11-64 ページ）
- COBOL（Pro*COBOL）：一時 LOB データの表示（11-67 ページ）
- C/C++（Pro*C/C++）：一時 LOB データの表示（11-69 ページ）
- Visual Basic（OO4O）：今回のリリースでは例は提供されません。
- Java（JDBC）：今回のリリースでは例は提供されません。

PL/SQL（DBMS_LOB）：一時 LOB データの表示

```
/* DIsplaying temporary LOB data. [Example script: 3853.sql]
   The following function accesses the monitor_photo file, creates a temporary
   LOB, loads some data from the file, and then reads it back and
   displays it. */

DECLARE
   Dest_loc       BLOB;
   Src_loc        BFILE := BFILENAME('ADPHOTO_DIR', 'monitor_photo_3060_11001');
   Amount         INTEGER := 128;
   Bbuf           RAW(128);
   Position       INTEGER :=1;
BEGIN
   DBMS_LOB.CREATETEMPORARY(Dest_loc,TRUE);
   /* Opening the FILE  is mandatory: */
   DBMS_LOB.OPEN(Src_loc, DBMS_LOB.LOB_READONLY);
   /* Opening the LOB is optional: */
   DBMS_LOB.OPEN(Dest_loc,DBMS_LOB.LOB_READWRITE);
   DBMS_LOB.LOADFROMFILE(Dest_loc,Src_loc,Amount);

   LOOP
      DBMS_LOB.READ (Dest_loc, Amount, Position, Bbuf);
      /* Display the buffer contents: */
      DBMS_OUTPUT.PUT_LINE('Result :'|| utl_raw.cast_to_varchar2(Bbuf));
      Position := Position + Amount;
   END LOOP;
EXCEPTION
   WHEN NO_DATA_FOUND THEN
      DBMS_OUTPUT.PUT_LINE('End of data loaded into temp LOB');
```

```
    DBMS_LOB.CLOSE(Dest_loc);
    DBMS_LOB.FREETEMPORARY(Dest_loc);
    /* Closing the file is mandatory unless you close the files later: */
    DBMS_LOB.CLOSE(Src_loc);
END;
```

C（OCI）：一時 LOB データの表示

```
/* Displaying temporary LOB data.  [Example script: 3854.c]
   This function accesses the monitor_photo file for product 3060 with ad_id
   11001. It creates a temporary LOB, loads some data from the file, then reads
   it back and displays it. The reading is done in a streaming fashion. This
   function assumes that the file specified is kept in the directory known by
   the directory alias "ADPHOTO_DIR". It also assumes that the file is at least
   14000 bytes long, which is the amount to be read and loaded. These amounts
   are arbitrary for this example. This function uses fprintf() to display the
   contents of the file. This works well for text data, but you may wish to
   change the method for binary data. For audio  data, you could, for instance,
   call an audio function. The function returns 0 if it completes successfully,
   and -1 if it fails. */

#define MAXBUFLEN 32767
sb4 display_file_to_lob( OCIError      *errhp,
                         OCISvcCtx     *svchp,
                         OCIStmt       *stmthp,
                         OCIEnv        *envhp)
{
  int rowind;
  char *binfile;
  OCILobLocator *tblob;
  OCILobLocator *bfile;

  ub4 amount = 14000;
  ub4 offset = 0;
  ub4 loblen = 0;
  ub4 amtp   = 0;
  sword retval;
  ub4 piece  = 1;
  ub4 remainder= 0;
  ub1 bufp[MAXBUFLEN];
  sb4 return_code = 0;

  (void) printf("\n===> Testing loading files into lobs and displaying
  them\n\n");

  if(OCIDescriptorAlloc((dvoid *)envhp, (dvoid **) &tblob,
```

```
                      (ub4) OCI_DTYPE_LOB,
                      (size_t) 0, (dvoid **) 0))
{
   printf("OCIDescriptor Alloc FAILED in print_length\n");
   return -1;
}

if(OCIDescriptorAlloc((dvoid *)envhp, (dvoid **) &bfile,
                      (ub4) OCI_DTYPE_FILE,
                      (size_t) 0, (dvoid **) 0))
{
   printf("OCIDescriptor Alloc FAILED in print_length\n");
   return -1;
}

/* Create a temporary LOB: */
if(OCILobCreateTemporary(svchp, errhp, tblob,(ub2)0, SQLCS_IMPLICIT,
                         OCI_TEMP_BLOB, OCI_ATTR_NOCACHE,
                         OCI_DURATION_SESSION))
{
    (void) printf("FAILED: CreateTemporary() \n");
    return -1;
}

if(OCILobFileSetName(envhp, errhp, &bfile, (text*)"ADPHOTO_DIR",
                     (ub2)
                     strlen("ADPHOTO_DIR"),(text*)"monitor_photo_3060_11001",
                     (ub2)strlen("monitor_photo_3060_11001")))
{
    printf("OCILobFileSetName FAILED\n");
    return_code = -1;
}

/* Open the BFILE: */
if(OCILobFileOpen(svchp, errhp, (OCILobLocator *) bfile, OCI_FILE_READONLY))
{
   printf( "OCILobFileOpen FAILED \n");
   return_code = -1;
}

if(OCILobLoadFromFile(svchp,errhp,tblob,(OCILobLocator*)bfile,(ub4)amount,
                      (ub4)1,(ub4)1))
{
    printf( "OCILobLoadFromFile FAILED\n");
    return_code = -1;
}
```

```
offset = 1;
memset(bufp, '\0', MAXBUFLEN);

retval = OCILobRead(svchp, errhp, tblob, &amtp, offset,
                    (dvoid *) bufp, (amount < MAXBUFLEN ? amount : MAXBUFLEN),
                    (dvoid *)0, (sb4 (*)(dvoid *, dvoid *, ub4, ub1)) 0,
                    (ub2) 0, (ub1) SQLCS_IMPLICIT);

printf("1st piece read from file is %s\n",bufp);

switch (retval)
{
  case OCI_SUCCESS:             /* Only one piece */
    (void) printf("stream read piece # %d \n", ++piece);
    (void)printf("piece read was %s\n",bufp);
    break;
  case OCI_ERROR:
    /*  report_error();  function not shown here */
    break;
  case OCI_NEED_DATA:           /* There are 2 or more pieces */
    remainder = amount;
    printf("remainder is %d \n",remainder);
    do
    {
         memset(bufp, '\0', MAXBUFLEN);
         amtp = 0;
         remainder -= MAXBUFLEN;
         printf("remainder is %d \n",remainder);
         retval = OCILobRead(svchp, errhp, tblob,  &amtp, offset,
                             (dvoid *) bufp, (ub4) MAXBUFLEN, (dvoid *)0,
                             (sb4 (*)(dvoid *, dvoid *, ub4, ub1)) 0,
                             (ub2) 0, (ub1) SQLCS_IMPLICIT);

      /* The amount read returned is undefined for FIRST, NEXT pieces: */
          (void)fprintf(stderr,"stream read %d th piece, amtp = %d\n",
                        ++piece, amtp);
          (void)fprintf(stderr,"piece  of length read was %d\n",
              strlen((const char*)bufp));
          (void)fprintf(stderr,"piece read was %s\n",bufp);
     } while (retval == OCI_NEED_DATA);
     break;
  default:
    (void) printf("Unexpected ERROR: OCILobRead() LOB.\n");
     break;
}

 /* Close the audio file: */
```

```
if (OCILobFileClose(svchp, errhp, (OCILobLocator *) bfile))
{
   printf( "OCILobFileClose FAILED\n");
   return_code =  -1;
}
/* clean up the temp LOB now that we are done with it */

if(check_and_free_temp(tblob, errhp,  svchp,stmthp, envhp))
{
    printf("check and free failed in load test\n");
    return_code = -1;
}
return return_code;
}
```

COBOL（Pro*COBOL）：一時 LOB データの表示

```
* Displaying temporary LOB data. [Example script: 3855.pco]
  IDENTIFICATION DIVISION.
  PROGRAM-ID. ONE-READ-BLOB.
  ENVIRONMENT DIVISION.
  DATA DIVISION.
  WORKING-STORAGE SECTION.
  01  USERID         PIC X(9) VALUES "SAMP/SAMP".
  01  TEMP-BLOB      SQL-BLOB.
  01  SRC-BFILE      SQL-BFILE.
  01  DIR-ALIAS      PIC X(30) VARYING.
  01  FNAME          PIC X(20) VARYING.
  01  BUFFER2        PIC X(32767) VARYING.
  01  AMT            PIC S9(9) COMP.
  01  OFFSET         PIC S9(9) COMP VALUE 1.
  01  ORASLNRD       PIC 9(4).
  01  ISTEMP         PIC S9(9) COMP.

      EXEC SQL INCLUDE SQLCA END-EXEC.
      EXEC ORACLE OPTION (ORACA=YES) END-EXEC.
      EXEC SQL INCLUDE ORACA END-EXEC.
      EXEC SQL VAR BUFFER2 IS LONG RAW(32767) END-EXEC.
  PROCEDURE DIVISION.
  ONE-READ-BLOB.
      EXEC SQL WHENEVER SQLERROR DO PERFORM SQL-ERROR END-EXEC.
      EXEC SQL
           CONNECT :USERID
      END-EXEC.

   * Allocate and initialize the BLOB locator:
```

```
     EXEC SQL ALLOCATE :SRC-BFILE END-EXEC.
     EXEC SQL ALLOCATE :TEMP-BLOB END-EXEC.
     EXEC SQL LOB CREATE TEMPORARY :TEMP-BLOB END-EXEC.
     EXEC SQL WHENEVER NOT FOUND GOTO END-OF-BLOB END-EXEC.

* Set up the directory and file information:
     MOVE "ADPHOTO_DIR" TO DIR-ALIAS-ARR.
     MOVE 9 TO DIR-ALIAS-LEN.
     MOVE "keyboard_photo_3106_13001" TO FNAME-ARR.
     MOVE 16 TO FNAME-LEN.

     EXEC SQL
        LOB FILE SET :SRC-BFILE DIRECTORY = :DIR-ALIAS,
        FILENAME = :FNAME
     END-EXEC.

     EXEC SQL
        LOB DESCRIBE :SRC-BFILE GET LENGTH INTO :AMT
     END-EXEC.

* Open source BFILE and destination temporary BLOB:
     EXEC SQL LOB OPEN :SRC-BFILE READ ONLY END-EXEC.
     EXEC SQL LOB OPEN :TEMP-BLOB READ WRITE END-EXEC.

     EXEC SQL
          LOB LOAD :AMT FROM FILE :SRC-BFILE INTO :TEMP-BLOB
     END-EXEC.

* Perform a single read:
     EXEC SQL
          LOB READ :AMT FROM :TEMP-BLOB INTO :BUFFER2
     END-EXEC.

     DISPLAY "Read ", BUFFER2, " from TEMP-BLOB".

 END-OF-BLOB.
     EXEC SQL LOB CLOSE :TEMP-BLOB END-EXEC.
     EXEC SQL LOB CLOSE :SRC-BFILE END-EXEC.
     EXEC SQL LOB FREE TEMPORARY :TEMP-BLOB END-EXEC.
     EXEC SQL FREE :TEMP-BLOB END-EXEC.
     EXEC SQL FREE :SRC-BFILE END-EXEC.
     STOP RUN.

 SQL-ERROR.
     EXEC SQL
         WHENEVER SQLERROR CONTINUE
     END-EXEC.
```

```
MOVE ORASLNR TO ORASLNRD.
DISPLAY " ".
DISPLAY "ORACLE ERROR DETECTED ON LINE ", ORASLNRD, ":".
DISPLAY " ".
DISPLAY SQLERRMC.
EXEC SQL
    ROLLBACK WORK RELEASE
END-EXEC.
STOP RUN.
```

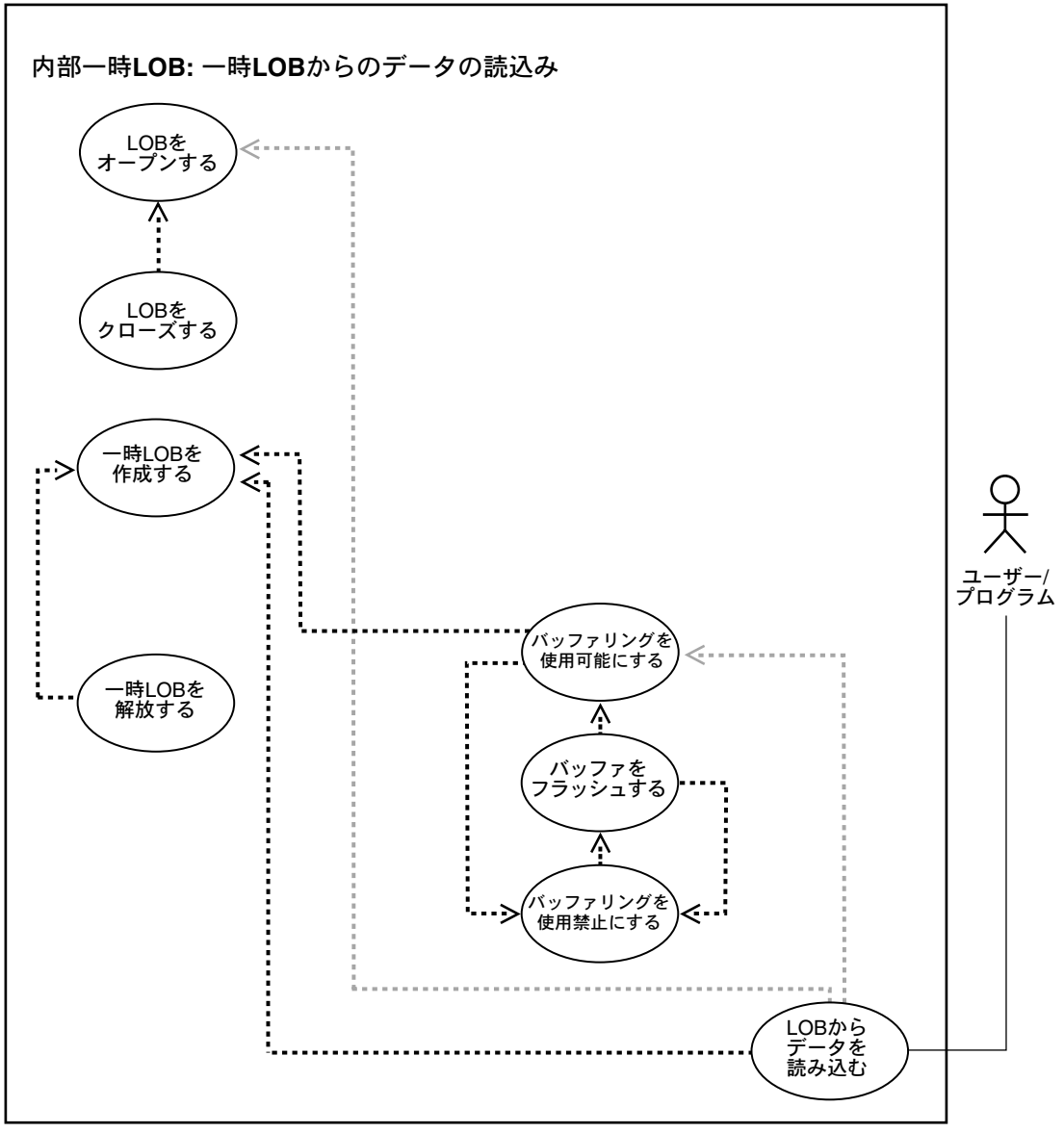
C/C++（Pro*C/C++）：一時 LOB データの表示

```
/* Displaying temporary LOB data. */
#include <oci.h>
#include <stdio.h>
#include <sqlca.h>

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

#define BufferLength 1024

void displayTempLOB_proc()
{
  OCIBlobLocator *Temp_loc;
  OCIBFileLocator *Lob_loc;
  char *Dir = "ADPHOTO_DIR", *Name = "monitor_photo_3060_11001";
  int Amount;
  struct {
    unsigned short Length;
    char Data[BufferLength];
  } Buffer;
  int Position = 1;
  /* Datatype Equivalencing is Mandatory for this Datatype */
  EXEC SQL VAR Buffer IS VARRAW(BufferLength);

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  /* Allocate and Initialize the LOB Locators */
  EXEC SQL ALLOCATE :Lob_loc;
  EXEC SQL ALLOCATE :Temp_loc;
  EXEC SQL LOB CREATE TEMPORARY :Temp_loc;
  EXEC SQL LOB FILE SET :Lob_loc DIRECTORY = :Dir, FILENAME = :Name;
```

```
  /* Opening the LOBs is Optional */
  EXEC SQL LOB OPEN :Lob_loc READ ONLY;
  EXEC SQL LOB OPEN :Temp_loc READ WRITE;
  /* Load a specified amount from the BFILE into the Temporary LOB */
  EXEC SQL LOB DESCRIBE :Lob_loc GET LENGTH INTO :Amount;
  EXEC SQL LOB LOAD :Amount FROM FILE :Lob_loc AT :Position INTO :Temp_loc;
  /* Setting Amount = 0 will initiate the polling method */
  Amount = 0;
  /* Set the maximum size of the Buffer */
  Buffer.Length = BufferLength;
  EXEC SQL WHENEVER NOT FOUND DO break;
  while (TRUE)
    {
      /* Read a piece of the BLOB into the Buffer */
      EXEC SQL LOB READ :Amount FROM :Temp_loc INTO :Buffer;
      printf("Display %d bytes\n", Buffer.Length);
    }
  printf("Display %d bytes\n", Amount);
  /* Closing the LOBs is mandatory if you have opened them */
  EXEC SQL LOB CLOSE :Lob_loc;
  EXEC SQL LOB CLOSE :Temp_loc;
  /* Free the Temporary LOB */
  EXEC SQL LOB FREE TEMPORARY :Temp_loc;
  /* Release resources held by the Locator */
  EXEC SQL FREE :Temp_loc;
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  displayTempLOB_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

一時 LOB からのデータの読込み

図 11-9 利用図 : 一時 LOB からのデータの読込み



参照： 11-2 ページの表 11-1「利用モデル : 内部一時 LOB」を参照してください。

用途

一時 LOB からデータを読み込みます。

使用上の注意

ストリーム読込み 大量の LOB データを最も効率よく読み込むには、ポーリングまたはコールバックによってストリーム・メカニズムを有効にした OCILobRead() を使用します。

LOB 値を読み込む場合、LOB の終わり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() コールを繰り返し、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。次に、パラメータの初期化を除いた OCI 読込みコールを示します。

```
#define MAX_LOB_SIZE 4294967295
ub4  amount =  MAX_LOB_SIZE;
ub4  offset = 1000;
OCILobRead(svchp, errhp, locp, &amount, offset, bufp, bufl, 0, 0, 0, 0)
```

ポーリング・モードを使用すると、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と、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（『Oracle Call Interface 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関数またはファンクションのリストは、第 3 章「様々なプログラム環境での LOB のサポート」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）：『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』の第 23 章「DBMS_LOB」の「DBMS_LOB サブプログラムの要約」の「READ プロシージャ」

- C（OCI）：『Oracle Call Interface プログラマーズ・ガイド』の第 16 章「その他の OCI リレーショナル関数」の「LOB 関数」の「OCILobRead()」
- COBOL（Pro*COBOL）：『Pro*COBOL Precompiler プログラマーズ・ガイド』の LOB に関する詳細、LOB 文の使用上の注意および付録 F「埋込み SQL およびプリコンパイラ・ディレクティブ」の「LOB READ（実行可能埋込み SQL 拡張機能）」
- C/C++（Pro*C/C++）：『Pro*C/C++ Precompiler プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびディレクティブ」の「LOB READ（実行可能埋込み SQL 拡張機能）」
- Visual Basic（OO4O）：参照マニュアルはありません。
- Java（JDBC）：10-112 ページの「Java（JDBC）：LOB からのデータの読込み」

使用例

次の例では、AD_PHOTO LOB 列から LOB データを読み込みます。

例

次のプログラム環境での例を示します。

- PL/SQL（DBMS_LOB）：一時 LOB からのデータの読込み（11-73 ページ）
- C（OCI）：一時 LOB からのデータの読込み（11-74 ページ）
- COBOL（Pro*COBOL）：一時 LOB からのデータの読込み（11-77 ページ）
- C/C++（Pro*C/C++）：一時 LOB からのデータの読込み（11-79 ページ）
- Visual Basic（OO4O）：今回のリリースでは例は提供されません。
- Java（JDBC）：今回のリリースでは例は提供されません。

PL/SQL（DBMS_LOB）：一時 LOB からのデータの読込み

```
/* Reading temporary LOB data. [Example script: 3859.sql]
   PL/SQL does not support streaming reads. See the OCI example for an
   illustration of streaming reads: */
DECLARE
  Dest_loc       BLOB;
  Src_loc        BFILE := BFILENAME('ADPHOTO_DIR', 'keyboard_photo_3106_13001');
  Amount         INTEGER := 4000;
  Bbuf           RAW(32767);
  Position       INTEGER :=1;
BEGIN
    DBMS_LOB.CREATETEMPORARY(Dest_loc,TRUE);
    /* Opening the FILE is mandatory: */
```

```
    DBMS_LOB.OPEN(Src_loc, DBMS_LOB.LOB_READONLY);
    /* Opening the LOB is optional: */
    DBMS_LOB.LOADFROMFILE(Dest_loc, Src_loc, Amount);
    DBMS_LOB.READ (Dest_loc, Amount, Position, Bbuf);
    /* Closing the LOB is mandatory if you have opened it: */
    DBMS_LOB.CLOSE(Src_loc);
END;
```

C（OCI）：一時 LOB からのデータの読込み

```
/* Reading temporary LOB data.  [Example script: 3861.c]
   This is the same example as for reading and displaying data from a
   temporary LOB. This function takes the 'monitor_photo for product 3060,
   ad_id 11001' file, opens that file as a BFILE as input, loads that file
   data into a temporary LOB and then reads the data from the temporary
   LOB, 5000 or less bytes at a time. 5000 bytes was an arbitrary maximum buffer
   length chosen for this example. The function returns 0 if it completes
   successfully, -1 if it fails. */

#define MAXBUFLEN 32767

sb4 test_file_to_lob (OCILobLocator *lob_loc,
                      OCIError      *errhp,
                      OCISvcCtx     *svchp,
                      OCIStmt       *stmthp,
                      OCIEnv        *envhp)
{
  int rowind;
  OCILobLocator *bfile;

  ub4 amount = 14000;
  ub4 offset =0;
  ub4   loblen = 0;
  ub4   amtp = 0;
  sword retval;
  ub4   piece = 1;
  ub4   remainder=0;
  ub1 bufp[MAXBUFLEN];

  (void) printf(
             "\n===> Testing loading files into lobs and displaying them\n\n");

  if (OCIDescriptorAlloc((dvoid **)&bfile,
                         (ub4)OCI_DTYPE_LOB, (size_t)0,
                         (dvoid**)0))
```

```
/* Create a temporary LOB: */
if(OCILobCreateTemporary(svchp, errhp, lob_loc, (ub2)0, SQLCS_IMPLICIT,
                         OCI_TEMP_BLOB, OCI_ATTR_NOCACHE,
                         OCI_DURATION_SESSION))
{
    (void) printf("FAILED: CreateTemporary() \n");
    return -1;
}
if(OCILobFileSetName(envhp, errhp, &bfile,(text*)"ADPHOTO_DIR",
                     (ub2)strlen("ADPHOTO_DIR"),
                     (text*)"monitor_photo_3060_11001",
                     (ub2)strlen("monitor_photo_3060_11001")))
 {
     printf("OCILobFileSetName FAILED\n");
     return -1;
 }
 if (OCILobFileOpen(svchp, errhp, (OCILobLocator *) bfile, OCI_FILE_READONLY))
{
    printf( "OCILobFileOpen FAILED \n");
    return -1;
}
if(OCILobLoadFromFile(svchp,errhp,lob_loc,(OCILobLocator*)bfile,(ub4)amount,
                      (ub4)1,(ub4)1))
{
    printf( "OCILobLoadFromFile FAILED\n");
    return -1;
}

offset = 1;
memset(bufp, '\0', 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:              /* Only one piece */
     (void) printf("stream read piece # %d \n", ++piece);
     (void)printf("piece read was %s\n",bufp);
     break;
   case OCI_ERROR:
     /*  report_error();  function not shown here */
     break;
   case OCI_NEED_DATA:            /* There are 2 or more pieces */
```

```
        remainder = amount;
        fprintf(stderr,"remainder is %d \n",remainder);
        do
        {
             memset(bufp, '\0', MAXBUFLEN);
             amtp = 0;
             remainder -= MAXBUFLEN;
             fprintf(stderr,"remainder is %d \n",remainder);

              retval = OCILobRead(svchp, errhp, lob_loc, &amtp, offset,
                                  (dvoid *) bufp,(ub4) MAXBUFLEN, (dvoid *)0,
                                  (sb4 (*)(dvoid *, dvoid *, ub4, ub1)) 0,
                                  (ub2) 0, (ub1) SQLCS_IMPLICIT);

            /* The amount read returned is undefined for FIRST, NEXT pieces: */
              (void)fprintf(stderr,"stream read %d th piece, amtp = %d\n",
                            ++piece, amtp);
              (void)fprintf(stderr,
                  "piece  of length read was %d\n",strlen((const  char *)bufp));
              (void)fprintf(stderr,"piece read was %s\n",bufp);
            } while (retval == OCI_NEED_DATA);
            break;
       default:
          (void) printf("Unexpected ERROR: OCILobRead() LOB.\n");
           break;
     }

      /* Close the audio file: */
     if (OCILobFileClose(svchp, errhp, (OCILobLocator *) bfile))
     {
        printf( "OCILobFileClose FAILED\n");
        return -1;
     }
     if (OCIDescriptorFree ((dvoid*) lob_loc, (ub4) OCI_DTYPE_LOB))
     {
        printf ("failed in OCIDescriptor Free\n");
        return -1;
     }

     /* Clean up the temp LOB now that we are done with it: */
     if(check_and_free_temp(lob_loc, errhp, svchp,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,
                        OCIStmt       *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）：一時 LOB からのデータの読込み

```
     * Reading temporary LOB data. [Example script: 3862.pco]
       IDENTIFICATION DIVISION.
       PROGRAM-ID. ONE-READ-BLOB.
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01  USERID          PIC X(9) VALUES "SAMP/SAMP".
       01  TEMP-BLOB       SQL-BLOB.
       01  SRC-BFILE       SQL-BFILE.
       01  DIR-ALIAS       PIC X(30) VARYING.
       01  FNAME           PIC X(20) VARYING.
       01  BUFFER2         PIC X(32767) VARYING.
       01  AMT             PIC S9(9) COMP.
       01  OFFSET          PIC S9(9) COMP VALUE 1.
```

```
 01  ORASLNRD       PIC 9(4).
 01  ISTEMP         PIC S9(9) COMP.

     EXEC SQL INCLUDE SQLCA END-EXEC.
     EXEC ORACLE OPTION (ORACA=YES) END-EXEC.
     EXEC SQL INCLUDE ORACA END-EXEC.
     EXEC SQL VAR BUFFER2 IS LONG RAW(32767) END-EXEC.
 PROCEDURE DIVISION.
 ONE-READ-BLOB.
     EXEC SQL WHENEVER SQLERROR DO PERFORM SQL-ERROR END-EXEC.
     EXEC SQL
          CONNECT :USERID
     END-EXEC.

* Allocate and initialize the BLOB locator:
     EXEC SQL ALLOCATE :SRC-BFILE END-EXEC.
     EXEC SQL ALLOCATE :TEMP-BLOB END-EXEC.
     EXEC SQL LOB CREATE TEMPORARY :TEMP-BLOB END-EXEC.
     EXEC SQL WHENEVER NOT FOUND GOTO END-OF-BLOB END-EXEC.

* Set up the directory and file information:
     MOVE "ADPHOTO_DIR" TO DIR-ALIAS-ARR.
     MOVE 9 TO DIR-ALIAS-LEN.
     MOVE "keyboard_photo_3106_13001" TO FNAME-ARR.
     MOVE 16 TO FNAME-LEN.

     EXEC SQL
        LOB FILE SET :SRC-BFILE DIRECTORY = :DIR-ALIAS,
        FILENAME = :FNAME
     END-EXEC.

     EXEC SQL
        LOB DESCRIBE :SRC-BFILE GET LENGTH INTO :AMT
     END-EXEC.

* Open source BFILE and destination temporary BLOB:
     EXEC SQL LOB OPEN :SRC-BFILE READ ONLY END-EXEC.
     EXEC SQL LOB OPEN :TEMP-BLOB READ WRITE END-EXEC.

     EXEC SQL
          LOB LOAD :AMT FROM FILE :SRC-BFILE INTO :TEMP-BLOB
     END-EXEC.

* Perform a single read:

     EXEC SQL
          LOB READ :AMT FROM :TEMP-BLOB INTO :BUFFER2
```

```
            END-EXEC.

            DISPLAY "Read ", BUFFER2, " from TEMP-BLOB".

        END-OF-BLOB.
            EXEC SQL LOB CLOSE :TEMP-BLOB END-EXEC.
            EXEC SQL LOB CLOSE :SRC-BFILE END-EXEC.
            EXEC SQL LOB FREE TEMPORARY :TEMP-BLOB END-EXEC.
            EXEC SQL FREE :TEMP-BLOB END-EXEC.
            EXEC SQL FREE :SRC-BFILE END-EXEC.
            STOP RUN.

        SQL-ERROR.
            EXEC SQL
                WHENEVER SQLERROR CONTINUE
            END-EXEC.
            MOVE ORASLNR TO ORASLNRD.
            DISPLAY " ".
            DISPLAY "ORACLE ERROR DETECTED ON LINE ", ORASLNRD, ":".
            DISPLAY " ".
            DISPLAY SQLERRMC.
            EXEC SQL
                ROLLBACK WORK RELEASE
            END-EXEC.
            STOP RUN.
```

C/C++（Pro*C/C++）：一時 LOB からのデータの読込み

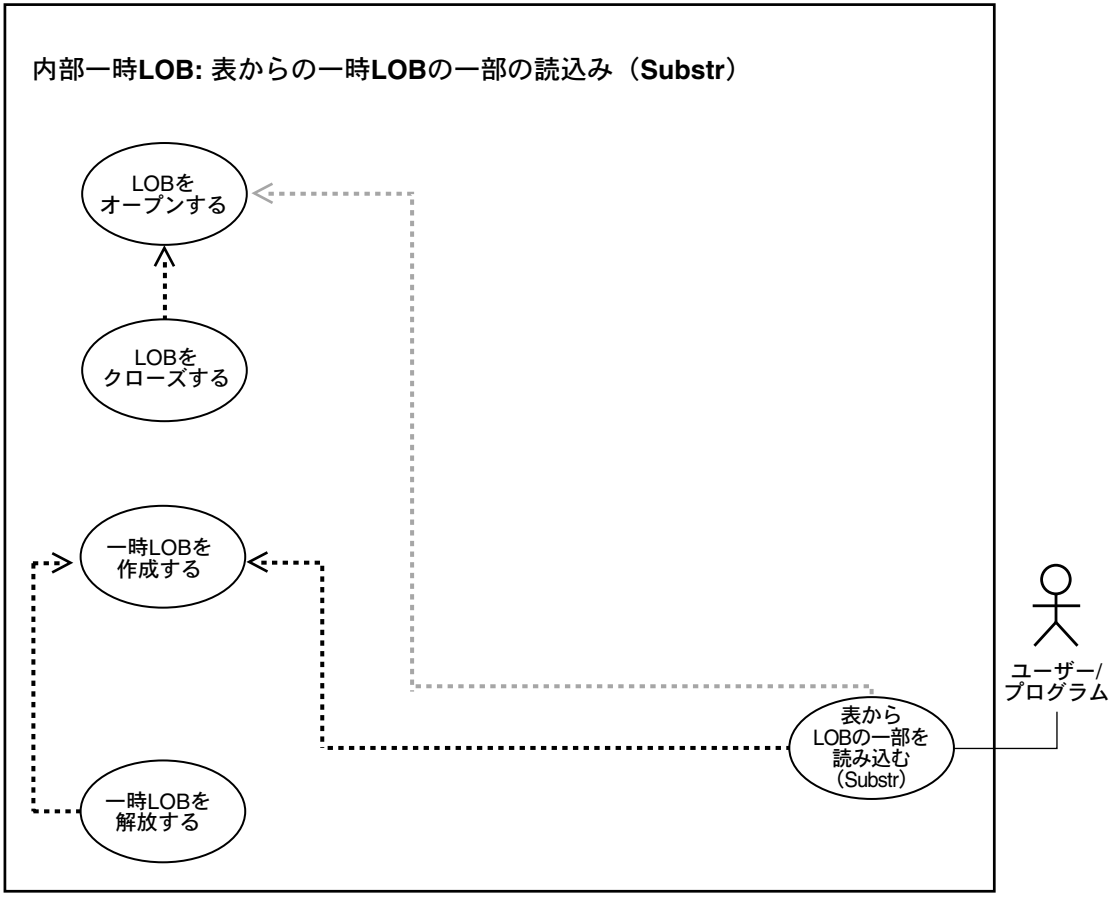
```
/*  Reading temporary LOB data.  [Example script: 3863.pc] */
#include <oci.h>
#include <stdio.h>
#include <sqlca.h>

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

#define BufferLength 1024

void readTempLOB_proc()
{
  OCIBlobLocator *Temp_loc;
```

```
OCIBFileLocator *Lob_loc;
char *Dir = "ADPHOTO_DIR", *Name = "monitor_photo_3060_11001";
int Length, Amount;
struct {
  unsigned short Length;
  char Data[BufferLength];
} Buffer;

/* Datatype Equivalencing is Mandatory for this Datatype */
EXEC SQL VAR Buffer IS VARRAW(BufferLength);
EXEC SQL WHENEVER SQLERROR DO Sample_Error();

/* Allocate and Initialize the BFILE Locator */
EXEC SQL ALLOCATE :Lob_loc;
EXEC SQL LOB FILE SET :Lob_loc DIRECTORY = :Dir, FILENAME = :Name;

/* Determine the Length of the BFILE */
EXEC SQL LOB DESCRIBE :Lob_loc GET LENGTH INTO :Length;

/* Allocate and Create the Temporary LOB */
EXEC SQL ALLOCATE :Temp_loc;
EXEC SQL LOB CREATE TEMPORARY :Temp_loc;

/* Open the BFILE for Reading */
EXEC SQL LOB OPEN :Lob_loc READ ONLY;

/* Load the BFILE into the Temporary LOB */
Amount = Length;
EXEC SQL LOB LOAD :Amount FROM FILE :Lob_loc INTO :Temp_loc;

/* Close the BFILE */
EXEC SQL LOB CLOSE :Lob_loc;
Buffer.Length = BufferLength;
EXEC SQL WHENEVER NOT FOUND DO break;
while (TRUE)
  {
    /* Read a piece of the Temporary LOB into the Buffer */
    EXEC SQL LOB READ :Amount FROM :Temp_loc INTO :Buffer;
    printf("Read %d bytes\n", Buffer.Length);
  }
printf("Read %d bytes\n", Amount);

/* Free the Temporary LOB */
EXEC SQL LOB FREE TEMPORARY :Temp_loc;

/* Release resources held by the Locators */
EXEC SQL FREE :Temp_loc;
```

```
  EXEC SQL FREE :Lob_loc;
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  readTempLOB_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

一時 LOB の一部の読込み（substr）

図 11-10 利用図 : 表からの一時 LOB の一部の読込み（substr）



参照： 11-2 ページの表 11-1「利用モデル : 内部一時 LOB」を参照してください。

用途

一時 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関数またはファンクションのリストは、第 3 章「様々なプログラム環境での LOB のサポート」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）：『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』の第 23 章「DBMS_LOB」の「DBMS_LOB 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）：『Pro*COBOL Precompiler プログラマーズ・ガイド』の LOB に関する詳細、LOB 文の使用上の注意および付録 F「埋込み SQL およびプリコンパイラ・ディレクティブ」の「LOB READ（実行可能埋込み SQL 拡張機能）」
- C/C++（Pro*C/C++）：『Pro*C/C++ Precompiler プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびディレクティブ」の「LOB LOAD（実行可能埋込み SQL 拡張機能）」、および『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』の第 23 章「DBMS_LOB」の「DBMS_LOB 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）：10-120 ページの「Java（JDBC）：LOB の一部の読込み（substr）」

使用例

次の例では、AD_PHOTO 列のイメージの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での例を示します。

- PL/SQL（DBMS_LOB）：一時 LOB の一部の読込み（substr）（11-84 ページ）
- C（OCI）：今回のリリースでは例は提供されません。
- COBOL（Pro*COBOL）：一時 LOB の一部の読込み（substr）（11-84 ページ）

- C/C++（Pro*C/C++）：一時 LOB の一部の読込み（substr）（11-86 ページ）
- Visual Basic（OO4O）：今回のリリースでは例は提供されません。
- Java（JDBC）：今回のリリースでは例は提供されません。

PL/SQL（DBMS_LOB）：一時 LOB の一部の読込み（substr）

```
/* Reading portion of a temporary LOB using substr. [Example script: 3864.sql]
   Example procedure substringTempLOB_proc is not part of DBMS_LOB package.
   This example assumes you have a 'monitor_photo_3060_11001' file in a
   directory which has a ADPHOTO alias */
CREATE or REPLACE PROCEDURE substringTempLOB_proc IS
   Dest_loc       BLOB;
   Src_loc        BFILE := BFILENAME('ADPHOTO_DIR', 'monitor_photo_3060_11001');
   Amount         INTEGER := 32767;
   Bbuf           RAW(32767);
   Position       INTEGER :=128;
BEGIN
   DBMS_LOB.CREATETEMPORARY(Dest_loc,TRUE);
   /* Opening the FILE  is mandatory: */
   DBMS_LOB.OPEN(Src_loc, DBMS_LOB.LOB_READONLY);
   /* Opening the LOB is optional */
   DBMS_LOB.OPEN(Dest_loc,DBMS_LOB.LOB_READWRITE);
   DBMS_LOB.LOADFROMFILE(Dest_loc, Src_loc, Amount);
   Bbuf := DBMS_LOB.SUBSTR(Dest_loc, Amount, Position);
   /* Closing the LOB is mandatory if you have opened it: */
   DBMS_LOB.CLOSE(Src_loc);
   DBMS_LOB.CLOSE(Dest_loc);
END;
```

COBOL（Pro*COBOL）：一時 LOB の一部の読込み（substr）

```
      * Reading portion of a temporary LOB using substr
      * [Example script: 3865.pco]
        IDENTIFICATION DIVISION.
        PROGRAM-ID. ONE-READ-BLOB.
        ENVIRONMENT DIVISION.
        DATA DIVISION.
        WORKING-STORAGE SECTION.

        01  USERID          PIC X(9) VALUES "SAMP/SAMP".
        01  TEMP-BLOB       SQL-BLOB.
        01  SRC-BFILE       SQL-BFILE.
        01  DIR-ALIAS       PIC X(30) VARYING.
        01  FNAME           PIC X(20) VARYING.
```

```
 01  BUFFER2        PIC X(32767) VARYING.
 01  AMT            PIC S9(9) COMP.
 01  OFFSET         PIC S9(9) COMP VALUE 1.
 01  ORASLNRD       PIC 9(4).
 01  ISTEMP         PIC S9(9) COMP.

     EXEC SQL INCLUDE SQLCA END-EXEC.
     EXEC ORACLE OPTION (ORACA=YES) END-EXEC.
     EXEC SQL INCLUDE ORACA END-EXEC.
     EXEC SQL VAR BUFFER2 IS LONG RAW(32767) END-EXEC.

 PROCEDURE DIVISION.
 ONE-READ-BLOB.

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

* Allocate and initialize the BLOB locator
     EXEC SQL ALLOCATE :SRC-BFILE END-EXEC.
     EXEC SQL ALLOCATE :TEMP-BLOB END-EXEC.
     EXEC SQL LOB CREATE TEMPORARY :TEMP-BLOB END-EXEC.
     EXEC SQL WHENEVER NOT FOUND GOTO END-OF-BLOB END-EXEC.

* Set up the directory and file information
     MOVE "ADPHOTO_DIR" TO DIR-ALIAS-ARR.
     MOVE 9 TO DIR-ALIAS-LEN.
     MOVE "keyboard_photo_3106_13001" TO FNAME-ARR.
     MOVE 16 TO FNAME-LEN.

     EXEC SQL
        LOB FILE SET :SRC-BFILE DIRECTORY = :DIR-ALIAS,
        FILENAME = :FNAME
     END-EXEC.

     EXEC SQL
        LOB DESCRIBE :SRC-BFILE GET LENGTH INTO :AMT
     END-EXEC.

* Open source BFILE and destination temporary BLOB.
     EXEC SQL LOB OPEN :SRC-BFILE READ ONLY END-EXEC.
     EXEC SQL LOB OPEN :TEMP-BLOB READ WRITE END-EXEC.

     EXEC SQL
          LOB LOAD :AMT FROM FILE :SRC-BFILE INTO :TEMP-BLOB
     END-EXEC.
```

```
      * Perform a single read

           EXEC SQL
                LOB READ :AMT FROM :TEMP-BLOB INTO :BUFFER2
           END-EXEC.

           DISPLAY "Read ", BUFFER2, " from TEMP-BLOB".

       END-OF-BLOB.
           EXEC SQL LOB CLOSE :TEMP-BLOB END-EXEC.
           EXEC SQL LOB CLOSE :SRC-BFILE END-EXEC.
           EXEC SQL LOB FREE TEMPORARY :TEMP-BLOB END-EXEC.
           EXEC SQL FREE :TEMP-BLOB END-EXEC.
           EXEC SQL FREE :SRC-BFILE END-EXEC.
           STOP RUN.

       SQL-ERROR.
           EXEC SQL
               WHENEVER SQLERROR CONTINUE
           END-EXEC.
           MOVE ORASLNR TO ORASLNRD.
           DISPLAY " ".
           DISPLAY "ORACLE ERROR DETECTED ON LINE ", ORASLNRD, ":".
           DISPLAY " ".
           DISPLAY SQLERRMC.
           EXEC SQL
               ROLLBACK WORK RELEASE
           END-EXEC.
           STOP RUN.
```

C/C++（Pro*C/C++）: 一時 LOB の一部の読込み（substr）

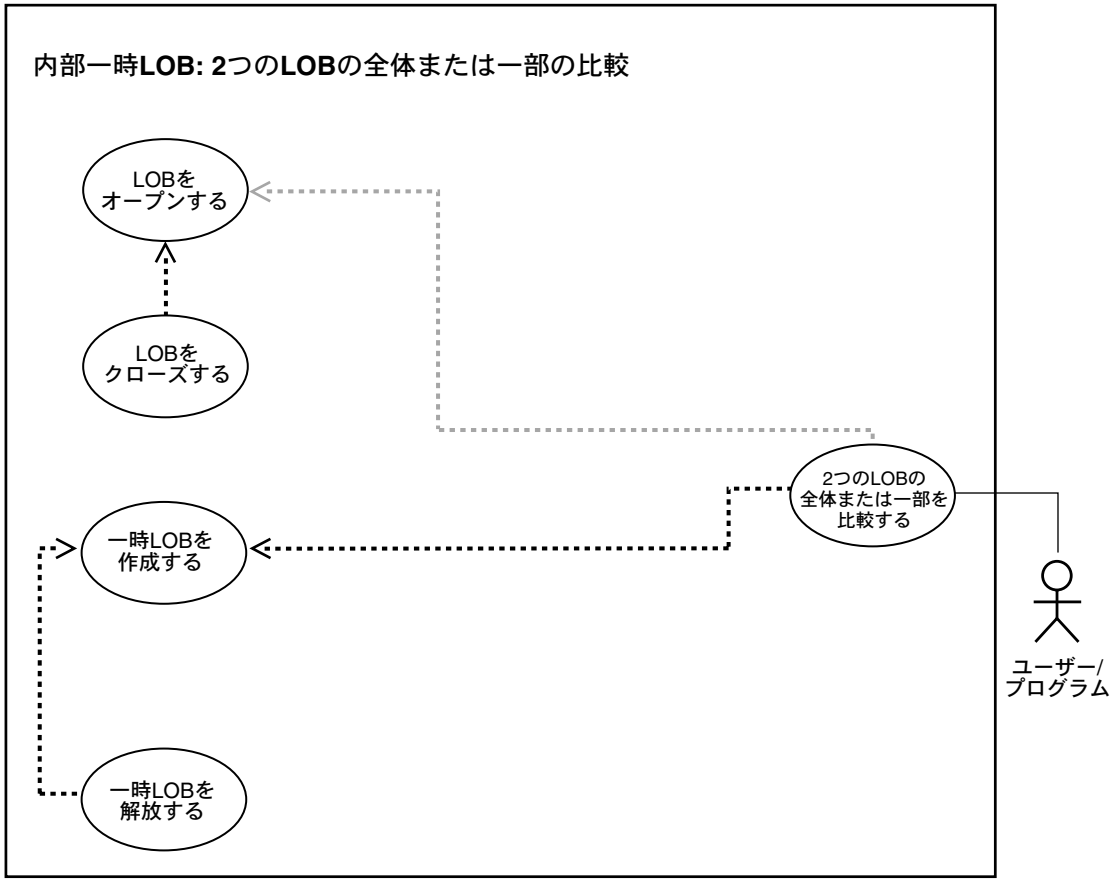
```
/* Reading a portion of a temporary LOB.  [Example script: 3866.pc]
   Pro*C/C++ lacks an equivalent embedded SQL form for the DBMS_LOB.SUBSTR()
   function. However, Pro*C/C++ can interoperate with PL/SQL using
   anonymous PL/SQL blocks embedded in a Pro*C/C++ program, as this example
   shows. */

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

void Sample_Error()
{
  EXEC SQL WHENEVER SQLERROR CONTINUE;
```

```
  printf("%.*s\n", sqlca.sqlerrm.sqlerrml, sqlca.sqlerrm.sqlerrmc);
  EXEC SQL ROLLBACK WORK RELEASE;
  exit(1);
}

#define BufferLength 4096

void substringTempLOB_proc()
{
  OCIBlobLocator *Temp_loc;
  OCIBFileLocator *Lob_loc;
  char *Dir = "ADPHOTO_DIR", *Name = "monitor_photo_3060_11001";
  int Position = 1024;
  unsigned int Length;
  int Amount = BufferLength;
  struct {
    unsigned short Length;
    char Data[BufferLength];
  } Buffer;
  /* Datatype Equivalencing is Mandatory for this Datatype: */
  EXEC SQL VAR Buffer IS VARRAW(BufferLength);

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  /* Allocate and Create the Temporary LOB: */
  EXEC SQL ALLOCATE :Temp_loc;
  EXEC SQL LOB CREATE TEMPORARY :Temp_loc;
  /* Allocate and Initialize the BFILE Locator: */
  EXEC SQL ALLOCATE :Lob_loc;
  EXEC SQL LOB FILE SET :Lob_loc DIRECTORY = :Dir, FILENAME = :Name;
  /* Open the LOBs: */
  EXEC SQL LOB OPEN :Lob_loc READ ONLY;
  EXEC SQL LOB OPEN :Temp_loc READ WRITE;
  /* Determine the length of the BFILE and load it into the Temporary LOB: */
  EXEC SQL LOB DESCRIBE :Lob_loc GET LENGTH INTO :Length;
  EXEC SQL LOB LOAD :Length FROM FILE :Lob_loc INTO :Temp_loc;
  /* Invoke SUBSTR() on the Temporary LOB inside a PL/SQL block: */
  EXEC SQL EXECUTE
    BEGIN
      :Buffer := DBMS_LOB.SUBSTR(:Temp_loc, :Amount, :Position);
    END;
  END-EXEC;
  /* Process the Data in the Buffer. */
  /* Closing the LOBs is Mandatory if they have been Opened: */
  EXEC SQL LOB CLOSE :Lob_loc;
  EXEC SQL LOB CLOSE :Temp_loc;
  /* Free the Temporary LOB: */
  EXEC SQL LOB FREE TEMPORARY :Temp_loc;
```

```
  /* Release resources used by the locators: */
  EXEC SQL FREE :Lob_loc;
  EXEC SQL FREE :Temp_loc;
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  substringTempLOB_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

2 つの一時 LOB の全体または一部の比較

図 11-11 利用図 : 2 つの一時 LOB の全体または一部の比較



参照 : 11-2 ページの表 11-1「利用モデル : 内部一時 LOB」を参照してください。

用途

2 つの一時 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関数またはファンクションのリストは、第 3 章「様々なプログラム環境での LOB のサポート」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）：『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』の第 23 章「DBMS_LOB」の「DBMS_LOB 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）：『Pro*COBOL Precompiler プログラマーズ・ガイド』の LOB に関する詳細、LOB 文の使用上の注意、付録 F「埋込み SQL およびプリコンパイラ・ディレクティブ」の「LOB COPY（実行可能埋込み SQL 拡張機能）」、および『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』の第 23 章「DBMS_LOB」の「DBMS_LOB サブプログラムの要約」の「COMPARE ファンクション」
- C/C++（Pro*C/C++）：『Pro*C/C++ Precompiler プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびディレクティブ」の「LOB COPY（実行可能埋込み SQL 拡張機能）」、および『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』の第 23 章「DBMS_LOB」の「DBMS_LOB 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）：10-128 ページの「Java（JDBC）：2 つの LOB の全体または一部の比較」

使用例

次の例では、2 つの複合広告を比較して違いを確認します。比較の結果に応じて、コンポジットを表に挿入します。

例

次のプログラム環境での例を示します。

- PL/SQL（DBMS_LOB）: 2 つの一時 LOB の全体または一部の比較（11-91 ページ）
- C（OCI）: 今回のリリースでは例は提供されません。
- COBOL（Pro*COBOL）: 2 つの一時 LOB の全体または一部の比較（11-92 ページ）
- C/C++（Pro*C/C++）: 2 つの一時 LOB の全体または一部の比較（11-94 ページ）
- Visual Basic（OO4O）: 今回のリリースでは例は提供されません。
- Java（JDBC）: 今回のリリースでは例は提供されません。

PL/SQL（DBMS_LOB）: 2 つの一時 LOB の全体または一部の比較

```
/* Comparing all or part of two temporary LOBs [Example script: 3868.sql]
   The procedure compareTwoTemporPersistLOBs_proc is not part
   of DBMS_LOB package. */

CREATE OR REPLACE PROCEDURE compareTwoTmpPerLOBs_proc IS
    Lob_loc1 BLOB;
    Lob_loc2 BLOB;
    Temp_loc BLOB;
    Amount   INTEGER := 32767;
    Retval   INTEGER;
BEGIN
    /* Select the LOB: */
    SELECT ad_composite INTO Lob_loc1 FROM Print_media
        WHERE product_ID = 3060 AND ad_id = 11001;
    SELECT ad_composite INTO Lob_loc2 FROM Print_media
        WHERE product_ID = 2268 AND ad_id = 21001;

    /* Copy one of the composite ad files into a temp LOB and convert
       it to a different format before comparing the ads : */
    DBMS_LOB.CREATETEMPORARY(Temp_loc, TRUE);
    DBMS_LOB.OPEN(Temp_loc, DBMS_LOB.LOB_READWRITE);
    DBMS_LOB.OPEN(Lob_loc1, DBMS_LOB.LOB_READONLY);
    DBMS_LOB.OPEN(Lob_loc2, DBMS_LOB.LOB_READONLY);
    /* Copy the persistent LOB into the temp LOB: */
    DBMS_LOB.COPY(Temp_loc,Lob_loc2,DBMS_LOB.GETLENGTH(Lob_loc2),1,1);

    /* Perform some conversion function on the temp LOB before comparing it*/
    /* ...some_conversion_format_function(Temp_loc); */
    retval := DBMS_LOB.COMPARE(Lob_loc1, Temp_loc, Amount, 1, 1);
    IF retval = 0 THEN
       DBMS_OUTPUT.PUT_LINE('Processing for equal frames');
```

```
ELSE
   DBMS_OUTPUT.PUT_LINE('Processing for non-equal frames');
END IF;
DBMS_LOB.CLOSE(Temp_loc);
DBMS_LOB.CLOSE(Lob_loc1);
DBMS_LOB.CLOSE(Lob_loc2);
/* Free the temporary LOB now that you are done using it: */
DBMS_LOB.FREETEMPORARY(Temp_loc);
END;
```

COBOL（Pro*COBOL）: 2 つの一時 LOB の全体または一部の比較

```
* Comparing all or part of two temporary LOBs [Example script: 3869.pco]
  IDENTIFICATION DIVISION.
  PROGRAM-ID. BLOB-COMPARE.
  ENVIRONMENT DIVISION.
  DATA DIVISION.
  WORKING-STORAGE SECTION.

  01  USERID   PIC X(11) VALUES "SAMP/SAMP".
  01  BLOB1          SQL-BLOB.
  01  BLOB2          SQL-BLOB.
  01  TEMP-BLOB      SQL-BLOB.
  01  RET            PIC S9(9) COMP.
  01  AMT            PIC S9(9) COMP VALUE 5.
  01 ORASLNRD        PIC 9(4).

      EXEC SQL INCLUDE SQLCA END-EXEC.
      EXEC ORACLE OPTION (ORACA=YES) END-EXEC.
      EXEC SQL INCLUDE ORACA END-EXEC.
  PROCEDURE DIVISION.
  BLOB-COMPARE.

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

 * Allocate and initialize the BLOB locators:
      EXEC SQL ALLOCATE :BLOB1 END-EXEC.
      EXEC SQL ALLOCATE :BLOB2 END-EXEC.
      EXEC SQL WHENEVER NOT FOUND GOTO END-OF-BLOB END-EXEC.
      EXEC SQL
           SELECT AD_COMPOSITE INTO :BLOB1
           FROM PRINT_MEDIA M WHERE M.PRODUCT_ID = 3106 AND AD_ID = 13001
      END-EXEC.
```

```
     EXEC SQL
          SELECT AD_COMPOSITE INTO :BLOB2
          FROM PRINT_MEDIA M WHERE M.PRODUCT_ID = 2268 AND AD_ID = 21001
     END-EXEC.

* Allocate and create a temporary LOB:
     EXEC SQL ALLOCATE :TEMP-BLOB END-EXEC.
     EXEC SQL
          LOB CREATE TEMPORARY :TEMP-BLOB
     END-EXEC.

* Open the BLOBs for READ ONLY, Open temp LOB READ/WRITE:
     EXEC SQL LOB OPEN :BLOB1 READ ONLY END-EXEC.
     EXEC SQL LOB OPEN :BLOB2 READ ONLY END-EXEC.
     EXEC SQL LOB OPEN :TEMP-BLOB READ WRITE END-EXEC.

* Copy data from BLOB2 to the temporary BLOB:
     EXEC SQL
          LOB COPY :AMT FROM :BLOB2 TO :TEMP-BLOB
     END-EXEC.

* Execute PL/SQL to use its COMPARE functionality:
     MOVE 5 TO AMT.
     EXEC SQL EXECUTE
       BEGIN
         :RET := DBMS_LOB.COMPARE(:BLOB1,:TEMP-BLOB,:AMT,1,1);
       END;
     END-EXEC.

     IF RET = 0
*        Logic for equal BLOBs goes here
         DISPLAY "BLOBs are equal"
     ELSE
*        Logic for unequal BLOBs goes here
         DISPLAY "BLOBs are not equal"
     END-IF.

     EXEC SQL LOB CLOSE :BLOB1 END-EXEC.
     EXEC SQL LOB CLOSE :BLOB2 END-EXEC.
     EXEC SQL LOB CLOSE :TEMP-BLOB END-EXEC.
     EXEC SQL LOB FREE TEMPORARY :TEMP-BLOB END-EXEC.

     EXEC SQL FREE :TEMP-BLOB END-EXEC.

 END-OF-BLOB.
     EXEC SQL WHENEVER NOT FOUND CONTINUE END-EXEC.
```

```
            EXEC SQL FREE :BLOB1 END-EXEC.
            EXEC SQL FREE :BLOB2 END-EXEC.
            STOP RUN.

        SQL-ERROR.
            EXEC SQL
                WHENEVER SQLERROR CONTINUE
            END-EXEC.
            MOVE ORASLNR TO ORASLNRD.
            DISPLAY " ".
            DISPLAY "ORACLE ERROR DETECTED ON LINE ", ORASLNRD, ":".
            DISPLAY " ".
            DISPLAY SQLERRMC.
            EXEC SQL
                ROLLBACK WORK RELEASE
            END-EXEC.
            STOP RUN.
```

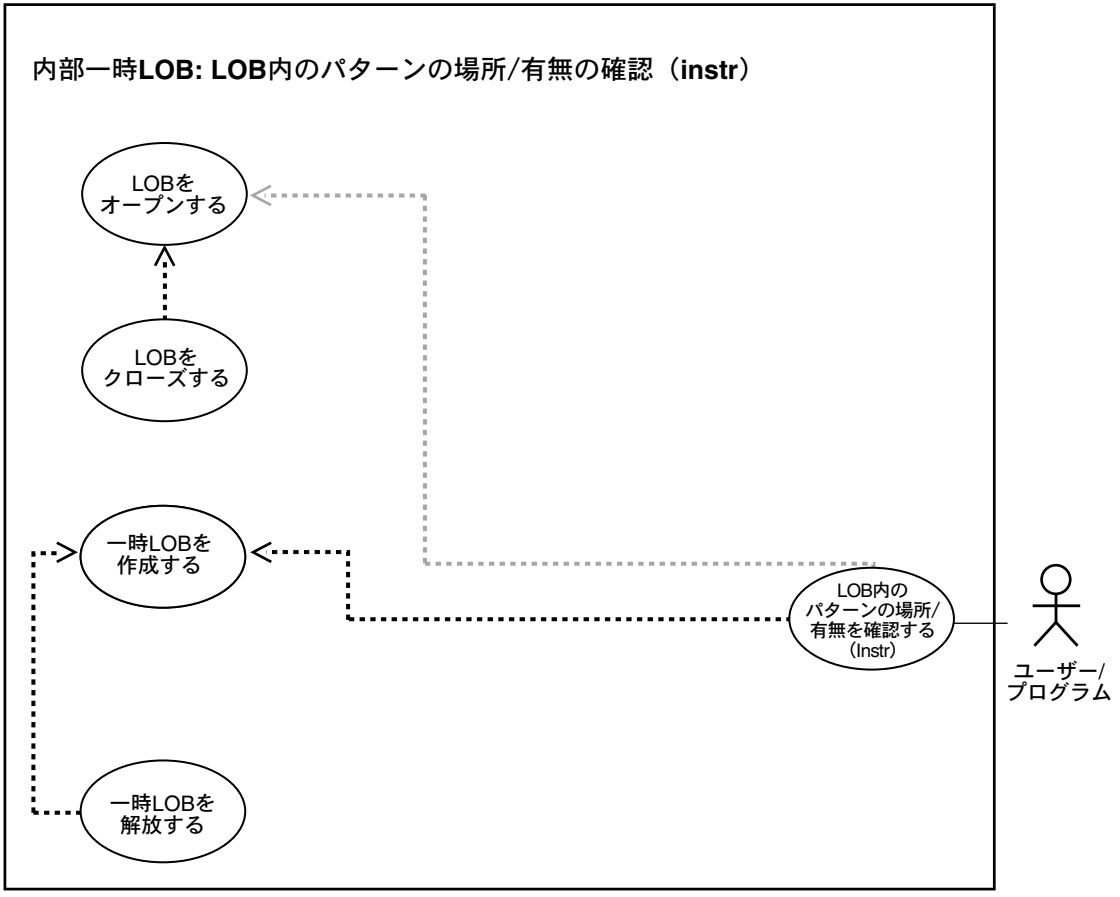
C/C++（Pro*C/C++）: 2 つの一時 LOB の全体または一部の比較

```
/* Comparing all or part of two temporary LOBs. [Example script: 3870.pc]
#include <oci.h>
#include <stdio.h>
#include <sqlca.h>

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

void compareTwoTempOrPersistLOBs_proc()
{
  OCIBlobLocator *Lob_loc1, *Lob_loc2, *Temp_loc;
  int Amount = 128;
  int Retval;

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  /* Allocate the LOB locators: */
  EXEC SQL ALLOCATE :Lob_loc1;
  EXEC SQL ALLOCATE :Lob_loc2;
  /* Select the LOBs: */
  EXEC SQL SELECT ad_composite INTO :Lob_loc1
     FROM Print_media WHERE Product_ID = 3060 AND ad_id = 11001;
```

```
  EXEC SQL SELECT ad_composite INTO :Lob_loc2
     FROM Print_media WHERE Product_ID = 2268 AND ad_id = 21001;
  /* Allocate and Create the Temporary LOB: */
  EXEC SQL ALLOCATE :Temp_loc;
  EXEC SQL LOB CREATE TEMPORARY :Temp_loc;

  /* Opening the LOBs is Optional: */
  EXEC SQL LOB OPEN :Lob_loc1 READ ONLY;
  EXEC SQL LOB OPEN :Lob_loc2 READ ONLY;
  EXEC SQL LOB OPEN :Temp_loc READ WRITE;
  /* Copy the Persistent LOB into the Temporary LOB: */
  EXEC SQL LOB COPY :Amount FROM :Lob_loc2 TO :Temp_loc;

  /* Compare the two Frames using DBMS_LOB.COMPARE() from within PL/SQL: */
  EXEC SQL EXECUTE
     BEGIN
       :Retval := DBMS_LOB.COMPARE(:Lob_loc1, :Temp_loc, :Amount, 1, 1);
     END;
  END-EXEC;
  if (0 == Retval)
    printf("Frames are equal\n");
  else
    printf("Frames are not equal\n");
  /* Closing the LOBs is mandatory if you have opened them: */
  EXEC SQL LOB CLOSE :Lob_loc1;
  EXEC SQL LOB CLOSE :Lob_loc2;
  EXEC SQL LOB CLOSE :Temp_loc;
  /* Free the Temporary LOB: */
  EXEC SQL LOB FREE TEMPORARY :Temp_loc;

  /* Release resources held by the locators: */
  EXEC SQL FREE :Lob_loc1;
  EXEC SQL FREE :Lob_loc2;
  EXEC SQL FREE :Temp_loc;
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  compareTwoTempOrPersistLOBs_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

一時 LOB 内のパターンの有無の確認（instr）

図 11-12 利用図 : 一時 LOB 内のパターンの有無の確認（instr）



参照： 11-2 ページの表 11-1「利用モデル : 内部一時 LOB」を参照してください。

用途

一時 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関数またはファンクションのリストは、第 3 章「様々なプログラム環境での LOB のサポート」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）：『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』の第 23 章「DBMS_LOB」の「DBMS_LOB 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）：『Pro*COBOL Precompiler プログラマーズ・ガイド』の LOB に関する詳細、LOB 文の使用上の注意、付録 F「埋込み SQL およびプリコンパイラ・ディレクティブ」の「LOB COPY（実行可能埋込み SQL 拡張機能）」、および『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』の第 23 章「DBMS_LOB」の「DBMS_LOB サブプログラムの要約」の「INSTR ファンクション」
- C/C++（Pro*C/C++）：『Pro*C/C++ Precompiler プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびディレクティブ」の「LOB COPY（実行可能埋込み SQL 拡張機能）」、および『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』の第 23 章「DBMS_LOB」の「DBMS_LOB 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）：10-135 ページの「Java（JDBC）：LOB 内のパターンの有無の確認（instr）」

使用例

次の例では、広告テキストを調べ、「children」という文字列が存在するかどうかを確認します。

例

次のプログラム環境での例を示します。

- PL/SQL（DBMS_LOB）：一時 LOB 内のパターンの有無の確認（11-98 ページ）
- C（OCI）：今回のリリースでは例は提供されません。
- COBOL（Pro*COBOL）：一時 LOB 内のパターンの有無の確認（instr）（11-99 ページ）
- 表「C/C++（Pro*C/C++）：一時 LOB 内のパターンの有無の確認（instr）」（11-101 ページ）
- Visual Basic（OO4O）：今回のリリースでは例は提供されません。
- Java（JDBC）：今回のリリースでは例は提供されません。

PL/SQL（DBMS_LOB）：一時 LOB 内のパターンの有無の確認

```
/* Determining if a pattern exists in a temporary LOB. [Example script: 3871.sql]
   Procedure instringTempLOB_proc is not part of DBMS_LOB package. */

CREATE OR REPLACE PROCEDURE instringTempLOB_proc IS
   Lob_loc        CLOB;
   Temp_clob      CLOB;
   Pattern        VARCHAR2(30) := 'children';    Position       INTEGER := 0;
   Offset         INTEGER := 1;
   Occurrence     INTEGER := 1;
BEGIN
   /* Create the temp LOB and copy a CLOB into it: */
   DBMS_LOB.CREATETEMPORARY(Temp_clob,TRUE);
   SELECT ad_sourcetext INTO Lob_loc FROM Print_media
      WHERE Product_ID = 3060 AND ad_id = 11001;

   DBMS_LOB.OPEN(Temp_clob,DBMS_LOB.LOB_READWRITE);
   DBMS_LOB.OPEN(Lob_loc,DBMS_LOB.LOB_READONLY);
   /* Copy the CLOB into the temp CLOB: */
   DBMS_LOB.COPY(Temp_clob,Lob_loc,DBMS_LOB.GETLENGTH(Lob_loc),1,1);
   /* Seek the pattern in the temp CLOB: */
   Position := DBMS_LOB.INSTR(Temp_clob, Pattern, Offset, Occurrence);
   IF Position = 0 THEN
      DBMS_OUTPUT.PUT_LINE('Pattern not found');
   ELSE
      DBMS_OUTPUT.PUT_LINE('The pattern occurs at '|| position);
   END IF;
   DBMS_LOB.CLOSE(Lob_loc);
   DBMS_LOB.CLOSE(Temp_clob);
   /* Free the temporary LOB: */
   DBMS_LOB.FREETEMPORARY(Temp_clob);
END;
```

COBOL（Pro*COBOL）：一時 LOB 内のパターンの有無の確認（instr）

```
* Determining if a pattern exists in a temporary LOB. [Example script: 3872.pco]
      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.

     * Allocate and initialize the CLOB locator:
          EXEC SQL ALLOCATE :CLOB1 END-EXEC.

          EXEC SQL WHENEVER NOT FOUND GOTO END-OF-CLOB END-EXEC.

          EXEC SQL WHENEVER NOT FOUND GOTO END-OF-CLOB END-EXEC.
          EXEC ORACLE OPTION (SELECT_ERROR=NO) END-EXEC.
          EXEC SQL
               SELECT AD_SOURCETEXT INTO :CLOB1
               FROM PRINT_MEDIA WHERE PRODUCT_ID = 3106 AND AD_ID = 11001
          END-EXEC.
          EXEC SQL ALLOCATE :TEMP-CLOB END-EXEC.
          EXEC SQL
               LOB CREATE TEMPORARY :TEMP-CLOB
```

```
     END-EXEC.

* Open the CLOB for READ ONLY:
     EXEC SQL LOB OPEN :CLOB1 READ ONLY END-EXEC.

* Use LOB describe to get the length of CLOB1:
     EXEC SQL
          LOB DESCRIBE :CLOB1 GET LENGTH INTO :LEN
     END-EXEC.
     EXEC SQL
          LOB COPY :LEN FROM :CLOB1 TO :TEMP-CLOB
     END-EXEC.

* Execute PL/SQL to get INSTR functionality:
     EXEC SQL EXECUTE
       BEGIN
         :POS := DBMS_LOB.INSTR(:TEMP-CLOB,:PATTERN,
                                :OFFSET, :OCCURRENCE);
       END;
     END-EXEC.

     IF POS = 0
*        Logic for pattern not found here
         DISPLAY "Pattern was not found"
     ELSE
*        Pos contains position where pattern is found
         DISPLAY "Pattern was found"
     END-IF.

* Close and free the LOBs:
     EXEC SQL LOB CLOSE :CLOB1 END-EXEC.
     EXEC SQL FREE :TEMP-CLOB END-EXEC.
     EXEC SQL
          LOB FREE TEMPORARY :TEMP-CLOB
     END-EXEC.
     EXEC SQL FREE :TEMP-CLOB END-EXEC.

 END-OF-CLOB.
     EXEC SQL WHENEVER NOT FOUND CONTINUE END-EXEC.
     EXEC SQL FREE :CLOB1 END-EXEC.
     STOP RUN.

 SQL-ERROR.
     EXEC SQL
         WHENEVER SQLERROR CONTINUE
     END-EXEC.
     MOVE ORASLNR TO ORASLNRD.
```

```
DISPLAY " ".
DISPLAY "ORACLE ERROR DETECTED ON LINE ", ORASLNRD, ":".
DISPLAY " ".
DISPLAY SQLERRMC.
EXEC SQL
    ROLLBACK WORK RELEASE
END-EXEC.
STOP RUN.
```

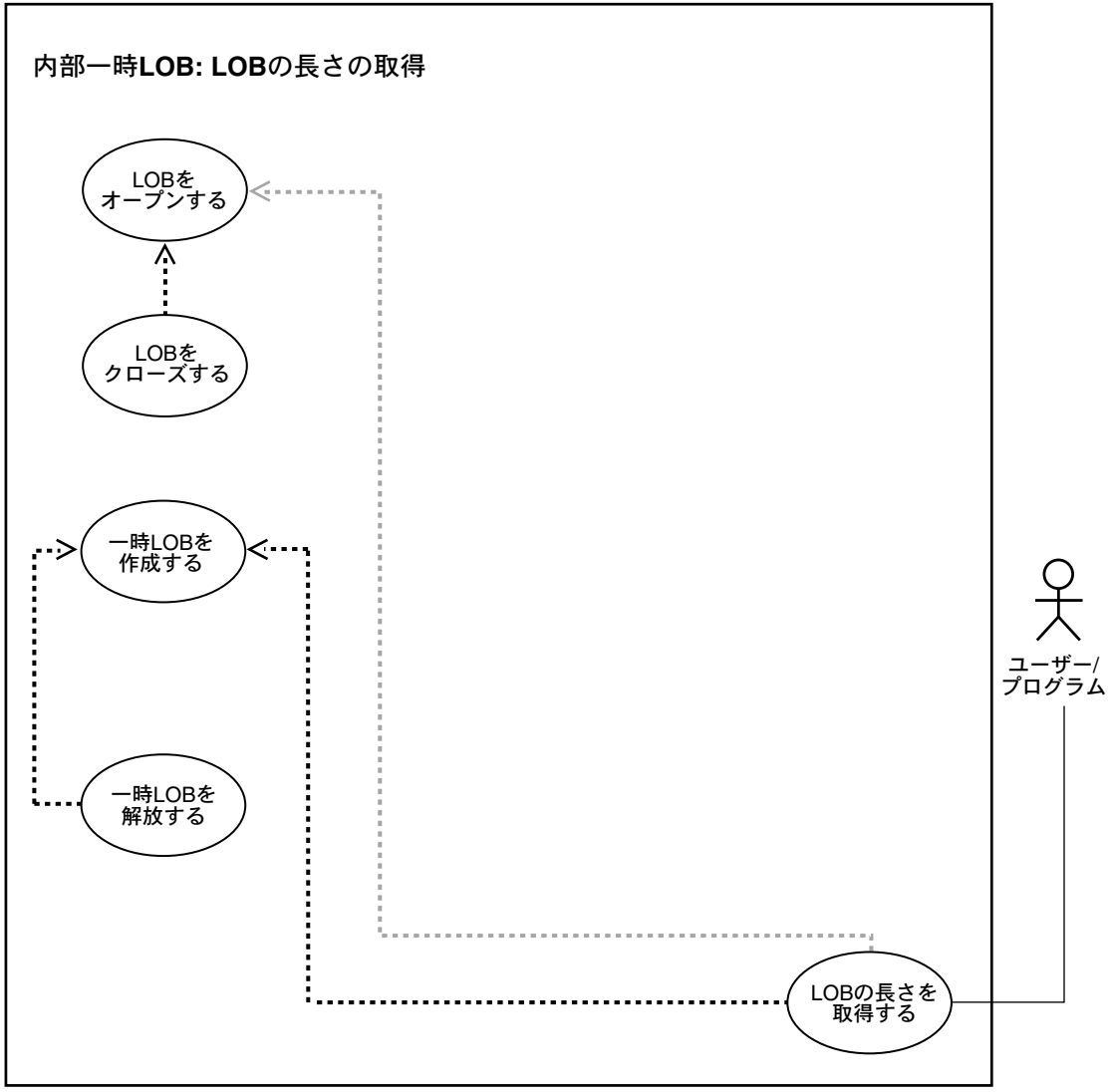
C/C++（Pro*C/C++）：一時 LOB 内のパターンの有無の確認（instr）

```
/* Determining if a pattern exists in a temporary LOB using instr
   [Example script: 3873.pc] */
#include <oci.h>
#include <stdio.h>
#include <sqlca.h>

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

void instringTempLOB_proc()
{
  OCIClobLocator *Lob_loc, *Temp_loc;
  char *Pattern = "The End";
  unsigned int Length;
  int Position = 0;
  int Offset = 1;
  int Occurrence = 1;

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  /* Allocate and Initialize the Persistent LOB: */
  EXEC SQL ALLOCATE :Lob_loc;
  EXEC SQL SELECT ad_sourcetext INTO :Lob_loc
     FROM print_media WHERE product_ID = 3060 AND ad_id = 11001;
  /* Allocate and Create the Temporary LOB: */
  EXEC SQL ALLOCATE :Temp_loc;
  EXEC SQL LOB CREATE TEMPORARY :Temp_loc;
  /* Opening the LOBs is Optional: */
  EXEC SQL LOB OPEN :Lob_loc READ ONLY;
  EXEC SQL LOB OPEN :Temp_loc READ WRITE;
  /* Determine the Length of the Persistent LOB: */
```

```
  EXEC SQL LOB DESCRIBE :Lob_loc GET LENGTH into :Length;
  /* Copy the Persistent LOB into the Temporary LOB: */
  EXEC SQL LOB COPY :Length FROM :Lob_loc TO :Temp_loc;
  /* Seek the Pattern using DBMS_LOB.INSTR() in a PL/SQL block: */
  EXEC SQL EXECUTE
     BEGIN
        :Position :=
            DBMS_LOB.INSTR(:Temp_loc, :Pattern, :Offset, :Occurrence);
     END;
  END-EXEC;
  if (0 == Position)
    printf("Pattern not found\n");
  else
    printf("The pattern occurs at %d\n", Position);
  /* Closing the LOBs is mandatory if you have opened them: */
  EXEC SQL LOB CLOSE :Lob_loc;
  EXEC SQL LOB CLOSE :Temp_loc;
  /* Free the Temporary LOB: */
  EXEC SQL LOB FREE TEMPORARY :Temp_loc;
  /* Release resources held by the Locators: */
  EXEC SQL FREE :Lob_loc;
  EXEC SQL FREE :Temp_loc;
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  instringTempLOB_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

一時 LOB の長さの取得

図 11-13 利用図 : 一時 LOB の長さの取得



参照： 11-2 ページの表 11-1「利用モデル : 内部一時 LOB」を参照してください。

用途

一時 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関数またはファンクションのリストは、第 3 章「様々なプログラム環境での LOB のサポート」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）：『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』の第 23 章「DBMS_LOB」の「DBMS_LOB サブプログラムの要約」の「GETLENGTH ファンクション」
- C（OCI）：『Oracle Call Interface プログラマーズ・ガイド』の第 16 章「その他の OCI リレーショナル関数」の「LOB 関数」の「OCILobGetLength()」
- COBOL（Pro*COBOL）：『Pro*COBOL Precompiler プログラマーズ・ガイド』の LOB に関する詳細、LOB 文の使用上の注意および付録 F「埋込み SQL およびプリコンパイラ・ディレクティブ」の「LOB DESCRIBE（実行可能埋込み SQL 拡張機能）」
- C/C++（Pro*C/C++）：『Pro*C/C++ Precompiler プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびディレクティブ」の「LOB DESCRIBE（実行可能埋込み SQL 拡張機能）」、および『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』の第 23 章「DBMS_LOB」の「DBMS_LOB サブプログラムの要約」の「GETLENGTH 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）: 10-143 ページの「Java（JDBC）: LOB の長さの取得」

使用例

次の例では、LOB 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での例を示します。

- PL/SQL（DBMS_LOB）：一時 LOB の長さの取得（11-105 ページ）
- C（OCI）：一時 LOB の長さの取得（11-106 ページ）
- COBOL（Pro*COBOL）：一時 LOB の長さの取得（11-108 ページ）
- C/C++（Pro*C/C++）：一時 LOB の長さの取得（11-110 ページ）
- Visual Basic（OO4O）：今回のリリースでは例は提供されません。
- Java（JDBC）：今回のリリースでは例は提供されません。

PL/SQL（DBMS_LOB）：一時 LOB の長さの取得

```
/* Finding the length of a temporary LOB.  [Example script: 3874.sql]
   Procedure getLengthTempCLOB_proc is not part of DBMS_LOB package. */

CREATE OR REPLACE PROCEDURE getLengthTempCLOB_proc IS
   Length      INTEGER;
   tlob        CLOB;
   bufc        VARCHAR2(8);
   Amount      NUMBER;
   pos         NUMBER;
   Src_loc     BFILE := BFILENAME('ADPHOTO_DIR', 'monitor_photo_3060_11001');
BEGIN
    DBMS_LOB.CREATETEMPORARY(tlob,TRUE);
    /* Opening the LOB is optional: */
    DBMS_LOB.OPEN(tlob,DBMS_LOB.LOB_READWRITE);
    /* Opening the file is mandatory: */
    DBMS_LOB.OPEN(Src_loc, DBMS_LOB.LOB_READONLY);
    Amount := 32767;
    DBMS_LOB.LOADFROMFILE(tlob, Src_loc, Amount);
    /* Get the length of the LOB: */
    length := DBMS_LOB.GETLENGTH(tlob);
    IF length = 0 THEN
        DBMS_OUTPUT.PUT_LINE('LOB is empty.');
    ELSE
        DBMS_OUTPUT.PUT_LINE('The length is ' || length);
    END IF;
     /* Must close any lobs that were opened: */
    DBMS_LOB.CLOSE(tlob);
    DBMS_LOB.CLOSE(Src_loc);
    /* Free the temporary LOB now that we are done with it: */
    DBMS_LOB.FREETEMPORARY(tlob);
END;
```

C（OCI）：一時 LOB の長さの取得

```
/* Finding the length of a temporary LOB. [Example script: 3875.c]
   This function takes a temporary LOB locator 'amount' as argument, and
   prints out the length of the corresponding LOB. The function returns
   0 if it completes successfully, -1 if it fails.*/
sb4 print_length( OCIError      *errhp,
                  OCISvcCtx     *svchp,
                  OCIStmt       *stmthp,
                  OCIEnv        *envhp)
{
  ub4 length=0;
  ub4 amount = 4;
  ub4 pos = 1;
  OCILobLocator *bfile;
  OCILobLocator *tblob;
  sb4 return_code = 0;

  printf("in print_length\n");
  if(OCIDescriptorAlloc((dvoid *)envhp, (dvoid **) &tblob,
                        (ub4) OCI_DTYPE_LOB,
                        (size_t) 0, (dvoid **) 0))
  {
     printf("OCIDescriptor Alloc FAILED in print_length\n");
     return -1;
  }

  if(OCIDescriptorAlloc((dvoid *)envhp, (dvoid **) &bfile,
                        (ub4) OCI_DTYPE_FILE,
                        (size_t) 0, (dvoid **) 0))
  {
     printf("OCIDescriptor Alloc FAILED in print_length\n");
     return -1;
  }

  if(OCILobFileSetName(envhp, errhp, &bfile, (text *)"ADPHOTO_DIR",
                       (ub2)strlen("ADPHOTO_DIR"),
                       (text *)"monitor_photo_3060_11001",
                       (ub2)strlen("monitor_photo_3060_11001")))
  {
     printf("OCILobFileSetName FAILED\n");
     return_code = -1;
  }
  checkerr(errhp,(OCILobFileOpen(svchp, errhp,
                                 (OCILobLocator *) bfile,
                                  OCI_LOB_READONLY)));
  /* Create a temporary BLOB: */
```

```
 if(OCILobCreateTemporary(svchp, errhp, tblob, (ub2)0, SQLCS_IMPLICIT,
                          OCI_TEMP_BLOB, OCI_ATTR_NOCACHE,
                          OCI_DURATION_SESSION))
 {
     (void) printf("FAILED: CreateTemporary() \n");
     return_code = -1 ;
 }

 if(OCILobOpen(svchp, errhp, (OCILobLocator *) tblob, OCI_LOB_READWRITE))
 {
     (void) printf("FAILED: Open Temporary \n");
     return_code = -1;
 }

if(OCILobLoadFromFile(svchp, errhp, tblob,(OCILobLocator*)bfile,
                      (ub4)amount, (ub4)1,(ub4)1))
 {
     (void) printf("FAILED: Open Temporary \n");
     return_code = -1;
 }

 if (OCILobGetLength(svchp, errhp, tblob,&length))
 {
     printf ("FAILED: OCILobGetLength in print_length\n");
     return_code = -1;
 }

 /* Close the bfile and the temp LOB */
 checkerr(errhp,OCILobFileClose(svchp, errhp, (OCILobLocator *) bfile));

 checkerr(errhp,OCILobClose(svchp, errhp, (OCILobLocator *) tblob));

 /* Free the temporary LOB now that we are done using it: */
 if(OCILobFreeTemporary(svchp, errhp, tblob))
 {
   printf("OCILobFreeTemporary FAILED \n");
   return_code = -1;
 }
 fprintf(stderr,"Length of LOB is %d\n",length);
 return return_code;
 }
```

COBOL（Pro*COBOL）: 一時 LOB の長さの取得

```
* Finding the length of a temporary LOB. [Example script: 3876.pco]
  IDENTIFICATION DIVISION.
  PROGRAM-ID. TEMP-LOB-LENGTH.
  ENVIRONMENT DIVISION.
  DATA DIVISION.
  WORKING-STORAGE SECTION.

  01  USERID   PIC X(11) VALUES "SAMP/SAMP".
  01  TEMP-BLOB      SQL-BLOB.
  01  SRC-BFILE      SQL-BFILE.
  01  DIR-ALIAS      PIC X(30) VARYING.
  01  FNAME          PIC X(20) VARYING.
  01  DIR-IND        PIC S9(4) COMP.
  01  FNAME-IND      PIC S9(4) COMP.
  01  AMT            PIC S9(9) COMP VALUE 10.
  01  LEN            PIC S9(9) COMP.
  01  LEN-D          PIC 9(4).
  01  ORASLNRD       PIC 9(4).

      EXEC SQL INCLUDE SQLCA END-EXEC.
      EXEC ORACLE OPTION (ORACA=YES) END-EXEC.
      EXEC SQL INCLUDE ORACA END-EXEC.
  PROCEDURE DIVISION.
  TEMP-LOB-LENGTH.
      EXEC SQL WHENEVER SQLERROR DO PERFORM SQL-ERROR END-EXEC.
      EXEC SQL
           CONNECT :USERID
      END-EXEC.

 * Allocate and initialize the BFILE and BLOB locators:
      EXEC SQL ALLOCATE :TEMP-BLOB END-EXEC.
      EXEC SQL ALLOCATE :SRC-BFILE END-EXEC.
      EXEC SQL
           LOB CREATE TEMPORARY :TEMP-BLOB
      END-EXEC.

 * Set up the directory and file information:
      MOVE "ADPHOTO_DIR" TO DIR-ALIAS-ARR.
      MOVE 9 TO DIR-ALIAS-LEN.
      MOVE "keyboard_photo_3106_13001" TO FNAME-ARR.
      MOVE 16 TO FNAME-LEN.

      EXEC SQL
         LOB FILE SET :SRC-BFILE DIRECTORY = :DIR-ALIAS,
         FILENAME = :FNAME
```

```
     END-EXEC.

* Open source BFILE and destination temporary BLOB:
     EXEC SQL LOB OPEN :TEMP-BLOB READ WRITE END-EXEC.
     EXEC SQL LOB OPEN :SRC-BFILE READ ONLY END-EXEC.
     EXEC SQL
          LOB LOAD :AMT FROM FILE :SRC-BFILE INTO :TEMP-BLOB
     END-EXEC.

* Get the length of the temporary LOB:
     EXEC SQL
          LOB DESCRIBE :TEMP-BLOB GET LENGTH INTO :LEN
     END-EXEC.
     MOVE LEN TO LEN-D.
     DISPLAY "Length of TEMPORARY LOB is ", LEN-D.
* Close the LOBs:
     EXEC SQL LOB CLOSE :SRC-BFILE END-EXEC.
     EXEC SQL LOB CLOSE :TEMP-BLOB END-EXEC.

* Free the temporary LOB:
     EXEC SQL
          LOB FREE TEMPORARY :TEMP-BLOB
     END-EXEC.

* And free the LOB locators:
     EXEC SQL FREE :TEMP-BLOB END-EXEC.
     EXEC SQL FREE :SRC-BFILE END-EXEC.
     STOP RUN.

 SQL-ERROR.
     EXEC SQL WHENEVER SQLERROR CONTINUE END-EXEC.
     MOVE ORASLNR TO ORASLNRD.
     DISPLAY " ".
     DISPLAY "ORACLE ERROR DETECTED ON LINE ", ORASLNRD, ":".
     DISPLAY " ".
     DISPLAY SQLERRMC.
     EXEC SQL ROLLBACK WORK RELEASE END-EXEC.
     STOP RUN.
```

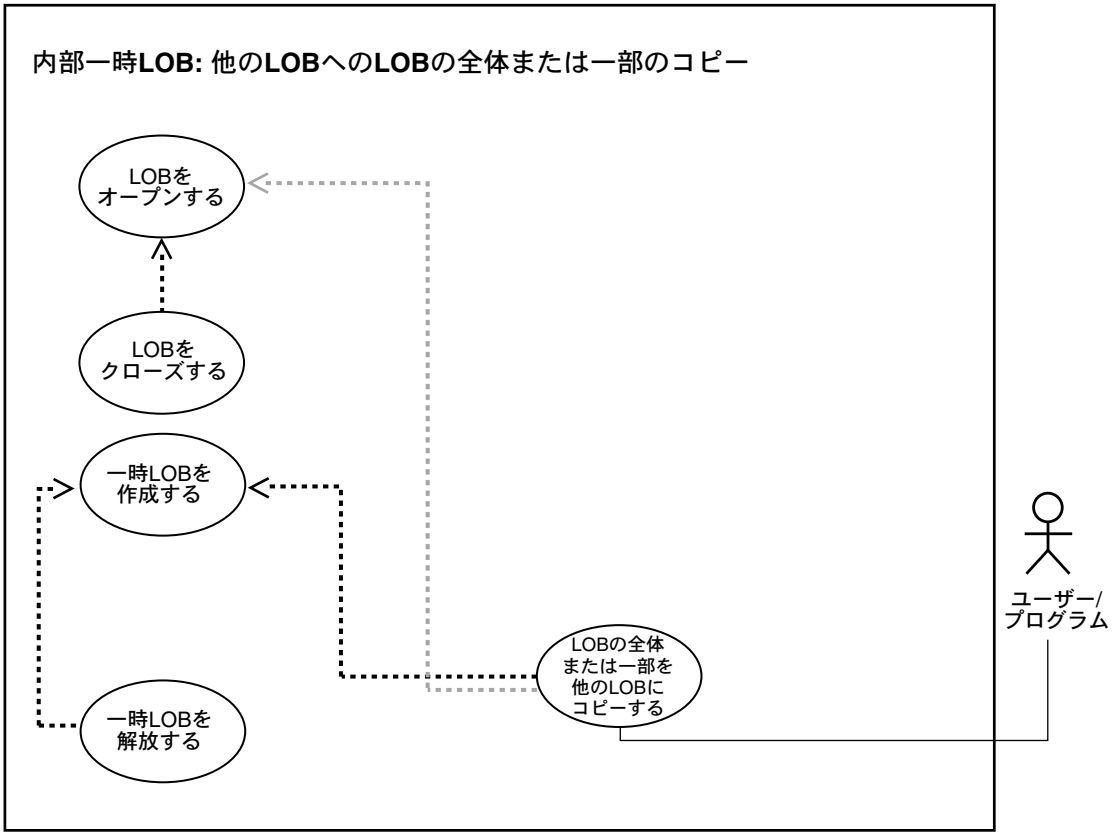
C/C++（Pro*C/C++）：一時 LOB の長さの取得

```
/* Finding the length of a temporary LOB.  [Example script: 3877.pc]  */
#include <oci.h>
#include <stdio.h>
#include <sqlca.h>

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

void getLengthTempLOB_proc()
{
  OCIBlobLocator *Temp_loc;
  OCIBFileLocator *Lob_loc;
  char *Dir = "ADPHOTO_DIR", *Name = "modem_photo_2268_21001";
  int Length, Amount;

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();

  /* Allocate and Create the Temporary LOB */
  EXEC SQL ALLOCATE :Temp_loc;
  EXEC SQL LOB CREATE TEMPORARY :Temp_loc;

  /* Allocate and Initialize the BFILE Locator: */
  EXEC SQL ALLOCATE :Lob_loc;
  EXEC SQL LOB FILE SET :Lob_loc DIRECTORY = :Dir, FILENAME = :Name;

  /* Opening the LOBs is Optional: */
  EXEC SQL LOB OPEN :Lob_loc READ ONLY;
  EXEC SQL LOB OPEN :Temp_loc READ WRITE;

  /* Load a specified amount from the BFILE into the Temporary LOB */
  Amount = 4096;
  EXEC SQL LOB LOAD :Amount FROM FILE :Lob_loc INTO :Temp_loc;

  /* Get the length of the Temporary LOB: */
  EXEC SQL LOB DESCRIBE :Temp_loc GET LENGTH INTO :Length;

  /* Note that in this example, Length == Amount == 4096: */
  printf("Length is %d bytes\n", Length);

  /* Closing the LOBs is Mandatory if they have been Opened: */
```

```
  EXEC SQL LOB CLOSE :Lob_loc;
  EXEC SQL LOB CLOSE :Temp_loc;

  /* Free the Temporary LOB: */
  EXEC SQL LOB FREE TEMPORARY :Temp_loc;
  /* Release resources held by the Locators: */
  EXEC SQL FREE :Lob_loc;
  EXEC SQL FREE :Temp_loc;
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  getLengthTempLOB_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

他の LOB への一時 LOB の全体または一部のコピー

図 11-14 利用図 : 他の LOB への一時 LOB の全体または一部のコピー



参照： 11-2 ページの表 11-1「利用モデル : 内部一時 LOB」を参照してください。

用途

一時 LOB の全体または一部を他の LOB にコピーします。

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関数またはファンクションのリストは、第 3 章「様々なプログラム環境での LOB のサポート」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）：『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』の第 23 章「DBMS_LOB」の「DBMS_LOB サブプログラムの要約」の「COPY プロシージャ」
- C（OCI）：『Oracle Call Interface プログラマーズ・ガイド』の第 16 章「その他の OCI リレーショナル関数」の「LOB 関数」の「OCILobCopy()」
- COBOL（Pro*COBOL）：『Pro*COBOL Precompiler プログラマーズ・ガイド』の LOB に関する詳細、LOB 文の使用上の注意および付録 F「埋込み SQL およびプリコンパイラ・ディレクティブ」の「LOB COPY（実行可能埋込み SQL 拡張機能）」
- C/C++（Pro*C/C++）：『Pro*C/C++ Precompiler プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびディレクティブ」の「LOB COPY（実行可能埋込み SQL 拡張機能）」
- Visual Basic（OO4O）：参照マニュアルはありません。
- Java（JDBC）：10-154 ページの「Java（JDBC）：他の LOB への LOB の全体または一部のコピー」

使用例

次の表を想定しています。

```
CREATE TABLE ad_library_tab of adheader_typ;

INSERT INTO adheader_tab
    (SELECT * FROM adheaderlib_tab Vtab1
       WHERE T2.creation_date = '08/16/2001');
```

次の例では、adheader_tab 表内に新しい LOB ロケータを作成し、adheader_tab 表に挿入された新しいロケータによって指される位置に Vtab1 からの LOB データをコピーします。

例

次のプログラム環境での例を示します。

- PL/SQL（DBMS_LOB）: 一時 LOB の全体または一部のコピー（11-114 ページ）
- C（OCI）: 他の LOB への一時 LOB の全体または一部のコピー（11-115 ページ）
- COBOL（Pro*COBOL）: 他の LOB への一時 LOB の全体または一部のコピー（11-118 ページ）
- C/C++（Pro*C/C++）: 他の LOB への一時 LOB の全体または一部のコピー（11-120 ページ）
- Visual Basic（OO4O）: 今回のリリースでは例は提供されません。
- Java（JDBC）: 今回のリリースでは例は提供されません。

PL/SQL（DBMS_LOB）: 一時 LOB の全体または一部のコピー

```
/* Copying all or part of one temporary LOB to another. [Example script: 3880.sql]
   Procedure copyTempLOB_proc is not part of DBMS_LOB package. */

CREATE OR REPLACE PROCEDURE copyTempLOB_proc IS
   Dest_pos     NUMBER;
   Src_pos      NUMBER;
   Dest_loc     BLOB;
   Dest_loc2    BLOB;
   Src_loc      BFILE := BFILENAME('ADPHOTO_DIR', 'monitor_photo_3060_11001');
   Amount       INTEGER := 32767;
BEGIN
   DBMS_LOB.CREATETEMPORARY(Dest_loc2,TRUE);
   DBMS_LOB.CREATETEMPORARY(Dest_loc,TRUE);
   /* Opening the FILE  is mandatory: */
   DBMS_LOB.OPEN(Src_loc, DBMS_LOB.LOB_READONLY);
   /* Opening the temporary  LOBs is optional: */
   DBMS_LOB.OPEN(Dest_loc,DBMS_LOB.LOB_READWRITE);
   DBMS_LOB.OPEN(Dest_loc2,DBMS_LOB.LOB_READWRITE);

   DBMS_LOB.LOADFROMFILE(Dest_loc, Src_loc, Amount);
      /* Set Dest_pos to the position at which we should start writing in the
          target temp LOB */
      /* Copies the LOB from the source position to the destination
          position:*/
      /* Set amount to the amount you want copied */
          Amount := 328;
          Dest_pos := 1000;
          Src_pos := 1000;
```

```
      /* Set Src_pos to the position from which we should start copying data
          from  tclob_src: */
          DBMS_LOB.COPY(Dest_loc2,Dest_loc, Amount, Dest_pos, Src_pos);
   COMMIT;
EXCEPTION
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('Operation failed');
      DBMS_LOB.CLOSE(Dest_loc);
      DBMS_LOB.CLOSE(Dest_loc2);
      DBMS_LOB.CLOSE(Src_loc);
      DBMS_LOB.FREETEMPORARY(Dest_loc);
      DBMS_LOB.FREETEMPORARY(Dest_loc2);
END;
```

C（OCI）：他の LOB への一時 LOB の全体または一部のコピー

```
/* Copying all or part of one temporary LOB to another. [Example script: 3881.c]
   This function copies 4000 bytes from one temporary LOB to another. It reads
   the source LOB starting at offset 1, and writes to the destination at
   offset 2.
   The function returns 0 if it completes successfully, -1 otherwise. */
   sb4 copy_temp_lobs (OCIError      *errhp,
                       OCISvcCtx     *svchp,
                       OCIStmt       *stmthp,
                       OCIEnv        *envhp)
{
  OCIDefine *defnp1;
  OCILobLocator *tblob;
  OCILobLocator *tblob2;
  OCILobLocator *bfile;
  int rowind =1;
  ub4 amount=4000;
  ub4 src_offset=1;
  ub4 dest_offset=2;
  sb4 return_code = 0;

  printf("in copy_temp_lobs \n");

  if(OCIDescriptorAlloc((dvoid*)envhp, (dvoid **)&tblob,
                        (ub4)OCI_DTYPE_LOB, (size_t)0, (dvoid**)0))
  {
    printf("OCIDescriptorAlloc failed in copy_temp_lobs\n");
    return -1;
  }
```

```
if(OCIDescriptorAlloc((dvoid*)envhp, (dvoid **)&bfile,
                      (ub4)OCI_DTYPE_FILE, (size_t)0, (dvoid**)0))
{
  printf("OCIDescriptorAlloc failed in copy_temp_lobs\n");
  return -1;
}

if(OCILobCreateTemporary(svchp, errhp, tblob, (ub2)0, SQLCS_IMPLICIT,
                         OCI_TEMP_BLOB, OCI_ATTR_NOCACHE,
                         OCI_DURATION_SESSION))
{
   (void) printf("FAILED: CreateTemporary() \n");
   return -1;
}

if(OCIDescriptorAlloc((dvoid*)envhp, (dvoid **)&tblob2,
                      (ub4)OCI_DTYPE_LOB, (size_t)0, (dvoid**)0))
{
  printf("OCIDescriptorAlloc failed in copy_temp_lobs\n");
  return_code =  -1;
}

if(OCILobCreateTemporary(svchp, errhp, tblob2, (ub2)0,  SQLCS_IMPLICIT,
                         OCI_TEMP_BLOB, OCI_ATTR_NOCACHE,
                         OCI_DURATION_SESSION))
{
   (void) printf("FAILED: CreateTemporary() \n");
   return_code = -1;
}

if(OCILobFileSetName(envhp, errhp, &bfile, (text *)"ADPHOTO_DIR",
                     (ub2)strlen("ADPHOTO_DIR"),
                     (text *)"monitor_photo_3060_11001",
                     (ub2)strlen("monitor_photo_3060_11001")))
{
  printf("OCILobFileSetName FAILED\n");
  return_code = -1;
}

if(OCILobFileOpen(svchp, errhp, (OCILobLocator *) bfile, OCI_LOB_READONLY))
{
  printf( "OCILobFileOpen FAILED  for the bfile\n");
  return_code = -1;
}
```

```
 if (OCILobOpen(svchp, errhp, (OCILobLocator *) tblob, OCI_LOB_READWRITE))
 {
   printf( "OCILobOpen FAILED for temp LOB \n");
   return_code =  -1;
 }
 if (OCILobOpen(svchp, errhp, (OCILobLocator *) tblob2, OCI_LOB_READWRITE ))
 {
   printf( "OCILobOpen FAILED for temp LOB \n");
   return_code =  -1;
 }

 if(OCILobLoadFromFile(svchp, errhp, tblob, (OCILobLocator*)bfile,
                       (ub4)amount, (ub4)1,(ub4)1))
 {
   printf( "OCILobLoadFromFile FAILED\n");
   return_code = -1;
 }

if (OCILobCopy(svchp, errhp, tblob2, tblob,  amount, dest_offset,
              src_offset))
 {
   printf ("FAILED: OCILobCopy in copy_temp_lobs\n");
   return -1;
 }
 /* Close LOBs here */

 if (OCILobFileClose(svchp, errhp, (OCILobLocator *) bfile))
 {
   printf( "OCILobFileClose FAILED for bfile \n");
   return_code =  -1;
 }
 if (OCILobClose(svchp, errhp, (OCILobLocator *) tblob))
 {
   printf( "OCILobClose FAILED for temporary LOB \n");
   return_code =  -1;
 }
 if (OCILobClose(svchp, errhp, (OCILobLocator *) tblob2))
 {
   printf( "OCILobClose FAILED for temporary LOB \n");
   return_code =  -1;
 }
 /* free the temporary lobs now that we are done using them */
 if(OCILobFreeTemporary(svchp, errhp, tblob))
 {
   printf("OCILobFreeTemporary FAILED \n");
   return_code = -1;
 }
```

```
  if(OCILobFreeTemporary(svchp, errhp, tblob2))
  {
    printf("OCILobFreeTemporary FAILED \n");
    return_code = -1;
  }
  return return_code;
}
```

COBOL（Pro*COBOL）: 他の LOB への一時 LOB の全体または一部のコピー

```
      * Copying all or part of one temporary LOB to another
      * [Example script: 3882.pco]
        IDENTIFICATION DIVISION.
        PROGRAM-ID. TEMP-BLOB-COPY.
        ENVIRONMENT DIVISION.
        DATA DIVISION.
        WORKING-STORAGE SECTION.

        01  USERID   PIC X(11) VALUES "SAMP/SAMP".
        01  TEMP-DEST      SQL-BLOB.
        01  TEMP-SRC       SQL-BLOB.
        01  SRC-BFILE      SQL-BFILE.
        01  DIR-ALIAS      PIC X(30) VARYING.
        01  FNAME          PIC X(30) VARYING.
        01  AMT            PIC S9(9) COMP.

       * Define the source and destination position and location:
        01  SRC-POS        PIC S9(9) COMP VALUE 1.
        01  DEST-POS       PIC S9(9) COMP VALUE 1.
        01 ORASLNRD        PIC 9(4).

            EXEC SQL INCLUDE SQLCA END-EXEC.
            EXEC ORACLE OPTION (ORACA=YES) END-EXEC.
            EXEC SQL INCLUDE ORACA END-EXEC.
        PROCEDURE DIVISION.
        TEMP-BLOB-COPY.
            EXEC SQL WHENEVER SQLERROR DO PERFORM SQL-ERROR END-EXEC.
            EXEC SQL
                 CONNECT :USERID
            END-EXEC.

       * Allocate and initialize the BLOB locators:
            EXEC SQL ALLOCATE :TEMP-DEST END-EXEC.
            EXEC SQL ALLOCATE :TEMP-SRC END-EXEC.
            EXEC SQL ALLOCATE :SRC-BFILE END-EXEC.
            EXEC SQL
```

```
           LOB CREATE TEMPORARY :TEMP-DEST
      END-EXEC.
      EXEC SQL
           LOB CREATE TEMPORARY :TEMP-SRC
      END-EXEC.

* Set up the directory and file information:
      MOVE "ADPHOTO_DIR" TO DIR-ALIAS-ARR.
      MOVE 9 TO DIR-ALIAS-LEN.
      MOVE "keyboard_photo_3106_13001" TO FNAME-ARR.
      MOVE 16 TO FNAME-LEN.

      EXEC SQL
         LOB FILE SET :SRC-BFILE DIRECTORY = :DIR-ALIAS,
         FILENAME = :FNAME
      END-EXEC.

* Open source BFILE and destination temporary BLOB:
      EXEC SQL LOB OPEN :TEMP-SRC READ WRITE END-EXEC.
      EXEC SQL LOB OPEN :TEMP-DEST READ WRITE END-EXEC.
      EXEC SQL LOB OPEN :SRC-BFILE READ ONLY END-EXEC.

* MOVE the desired amount to copy to AMT:
      MOVE 5 TO AMT.
      EXEC SQL
           LOB LOAD :AMT FROM FILE :SRC-BFILE INTO :TEMP-SRC
      END-EXEC.

* Copy data from BFILE to temporary LOB:
      EXEC SQL
           LOB COPY :AMT FROM :TEMP-SRC AT :SRC-POS
           TO :TEMP-DEST AT :DEST-POS
      END-EXEC.

      EXEC SQL LOB CLOSE :TEMP-SRC END-EXEC.
      EXEC SQL LOB CLOSE :TEMP-DEST END-EXEC.
      EXEC SQL LOB CLOSE :SRC-BFILE END-EXEC.
      EXEC SQL
           LOB FREE TEMPORARY :TEMP-SRC
      END-EXEC.
      EXEC SQL
           LOB FREE TEMPORARY :TEMP-DEST
      END-EXEC.
      EXEC SQL FREE :TEMP-SRC END-EXEC.
      EXEC SQL FREE :TEMP-DEST END-EXEC.
      EXEC SQL FREE :SRC-BFILE END-EXEC.
      STOP RUN.
```

```
SQL-ERROR.
    EXEC SQL
        WHENEVER SQLERROR CONTINUE
    END-EXEC.
    MOVE ORASLNR TO ORASLNRD.
    DISPLAY " ".
    DISPLAY "ORACLE ERROR DETECTED ON LINE ", ORASLNRD, ":".
    DISPLAY " ".
    DISPLAY SQLERRMC.
    EXEC SQL ROLLBACK WORK RELEASE END-EXEC.
    STOP RUN.
```

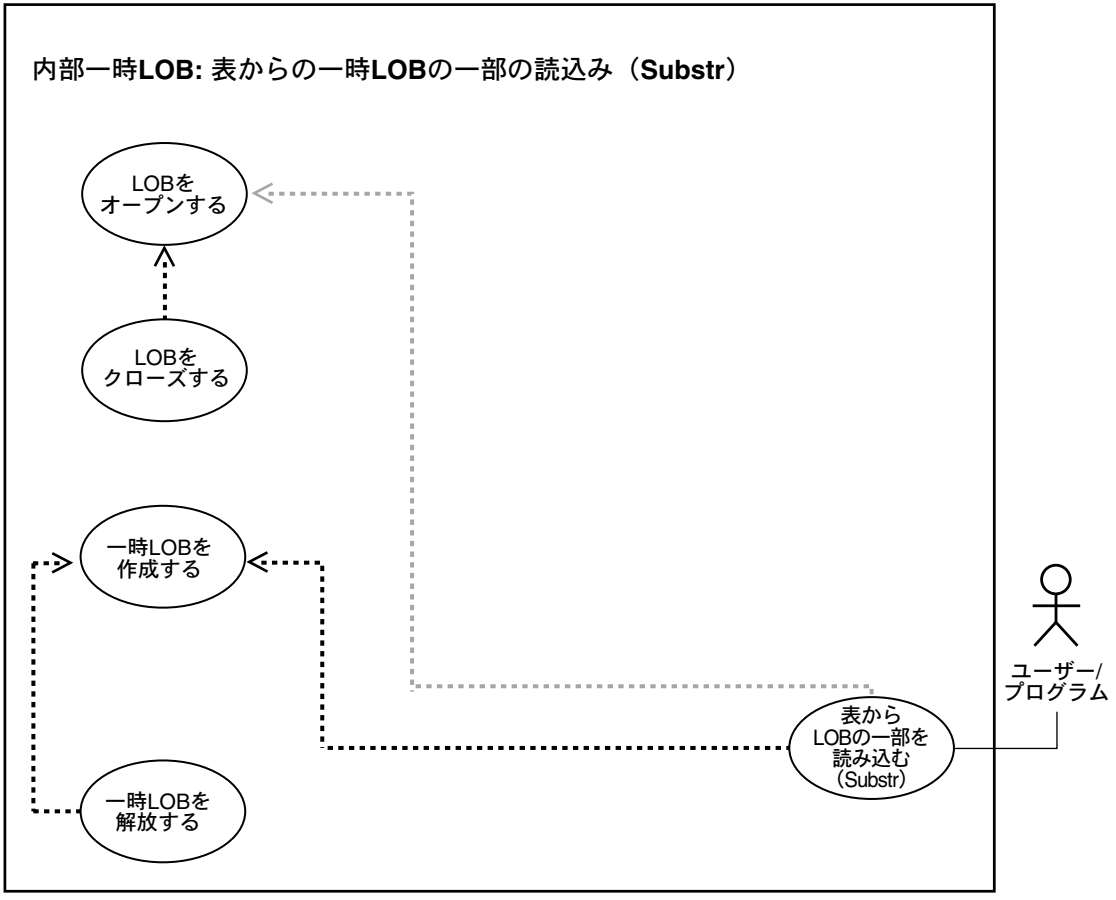
C/C++（Pro*C/C++）: 他の LOB への一時 LOB の全体または一部のコピー

```
/* Copying all or part of one temporary LOB to another. [Example script: 3883.pc] */
#include <oci.h>
#include <stdio.h>
#include <sqlca.h>

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

void copyTempLOB_proc()
{
  OCIBlobLocator *Temp_loc1, *Temp_loc2;
  OCIBFileLocator *Lob_loc;
  char *Dir = "ADPHOTO_DIR", *Name = "monitor_photo_3060_11001";
  int Amount;

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  /* Allocate and Create the Temporary LOBs: */
  EXEC SQL ALLOCATE :Temp_loc1;
  EXEC SQL ALLOCATE :Temp_loc2;
  EXEC SQL LOB CREATE TEMPORARY :Temp_loc1;
  EXEC SQL LOB CREATE TEMPORARY :Temp_loc2;
  /* Allocate and Initialize the BFILE Locator: */
  EXEC SQL ALLOCATE :Lob_loc;
  EXEC SQL LOB FILE SET :Lob_loc DIRECTORY = :Dir, FILENAME = :Name;
  /* Opening the LOBs is Optional: */
  EXEC SQL LOB OPEN :Lob_loc READ ONLY;
```

```
  EXEC SQL LOB OPEN :Temp_loc1 READ WRITE;
  EXEC SQL LOB OPEN :Temp_loc2 READ WRITE;
  /* Load a specified amount from the BFILE into one of the
     Temporary LOBs: */
  Amount = 4096;
  EXEC SQL LOB LOAD :Amount FROM FILE :Lob_loc INTO :Temp_loc1;
  /* Copy a specified amount from one Temporary LOB to another: */
  EXEC SQL LOB COPY :Amount FROM :Temp_loc1 TO :Temp_loc2;
  /* Closing the LOBs is Mandatory if they have been Opened: */
  EXEC SQL LOB CLOSE :Temp_loc1;
  EXEC SQL LOB CLOSE :Temp_loc2;
  EXEC SQL LOB CLOSE :Lob_loc;
  /* Free the Temporary LOBs: */
  EXEC SQL LOB FREE TEMPORARY :Temp_loc1;
  EXEC SQL LOB FREE TEMPORARY :Temp_loc2;
  /* Release resources held by the Locators: */
  EXEC SQL FREE :Temp_loc1;
  EXEC SQL FREE :Temp_loc2;
  EXEC SQL FREE :Lob_loc;
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  copyTempLOB_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

一時 LOB の LOB ロケータのコピー

図 11-15 利用図 : 一時 LOB の LOB ロケータのコピー



参照： 11-2 ページの表 11-1「利用モデル : 内部一時 LOB」を参照してください。

用途

一時 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

各プログラム環境で使用可能な関数またはファンクションのリストは、第 3 章「様々なプログラム環境での LOB のサポート」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）：『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』の第 23 章「DBMS_LOB」の「DBMS_LOB 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」および「FREETEMPORARY プロシージャ」
- C（OCI）：『Oracle Call Interface プログラマーズ・ガイド』の第 16 章「その他の OCI リレーショナル関数」の「LOB 関数」の「OCILobLocatorIsInit()」
- COBOL（Pro*COBOL）：『Pro*COBOL Precompiler プログラマーズ・ガイド』の LOB に関する詳細、LOB 文の使用上の注意および付録 F「埋込み SQL およびプリコンパイラ・ディレクティブ」の「LOB ASSIGN（実行可能埋込み SQL 拡張機能）」
- C/C++（Pro*C/C++）：『Pro*C/C++ Precompiler プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびディレクティブ」の「LOB ASSIGN（実行可能埋込み SQL 拡張機能）」
- Visual Basic（OO4O）：参照マニュアルはありません。
- Java（JDBC）：10-162 ページの「Java（JDBC）：LOB ロケータのコピー」

使用例

次の例では、一時 LOB ロケータを別の一時 LOB へコピーします。

例

次のプログラム環境での例を示します。

- PL/SQL（DBMS_LOB）：一時 LOB の LOB ロケータのコピー（11-124 ページ）
- C（OCI）：一時 LOB の LOB ロケータのコピー（11-125 ページ）
- COBOL（Pro*COBOL）：一時 LOB の LOB ロケータのコピー（11-127 ページ）
- C/C++（Pro*C/C++）：一時 LOB の LOB ロケータのコピー（11-129 ページ）
- Visual Basic（OO4O）：今回のリリースでは例は提供されません。

- Java（JDBC）: 今回のリリースでは例は提供されません。

PL/SQL（DBMS_LOB）: 一時 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、第 5 章「ラージ・オブジェクト（LOB）: 詳細事項」の「読込み一貫性のあるロケータ」を参照してください。

```
/* Copying a LOB locator for a temporary LOB.
   Procedure copyTempLOBLocator_proc is not part of DBMS_LOB package. */

CREATE OR REPLACE PROCEDURE copyTempLOBLocator_proc(
   Lob_loc1 IN OUT CLOB, Lob_loc2 IN OUT CLOB) IS

   bufp     VARCHAR2(4);
   Amount   NUMBER  := 32767;
   Src_loc  BFILE := BFILENAME('ADPHOTO_DIR', 'monitor_photo_3060_11001');
BEGIN
    DBMS_LOB.CREATETEMPORARY(Lob_loc1,TRUE);
    DBMS_LOB.CREATETEMPORARY(Lob_loc2,TRUE);
    /* Populate the first temporary LOB with some data. */
    /* Opening file is mandatory: */
    DBMS_LOB.OPEN(Src_loc,DBMS_LOB.LOB_READONLY);
    /* Opening LOB is optional: */
    DBMS_LOB.OPEN(Lob_loc1,DBMS_LOB.LOB_READWRITE);
    DBMS_LOB.OPEN(Lob_loc2,DBMS_LOB.LOB_READWRITE);
    DBMS_LOB.LOADFROMFILE(Lob_loc1,Src_loc,Amount);

   /* Assign Lob_loc1 to Lob_loc2 thereby creating  a copy of the value of
      the temporary LOB referenced by Lob_loc1 at this point in time: */
   Lob_loc2 := Lob_loc1;

   /* When you write some data to the LOB through Lob_loc1, Lob_loc2
      will not see the newly written data whereas Lob_loc1 will see
      the new data: */
   /*Closing LOBs is mandatory if they were opened: */
    DBMS_LOB.CLOSE (Src_loc);
    DBMS_LOB.CLOSE (Lob_loc1);
    DBMS_LOB.CLOSE (Lob_loc2);
    DBMS_LOB.FREETEMPORARY(Lob_loc1);
    DBMS_LOB.FREETEMPORARY(Lob_loc2);
END;
```

C（OCI）：一時 LOB の LOB ロケータのコピー

```
* Copying a LOB locator for a temporary LOB.  [Example script: 3886.c]
  This function creates two temporary lobs. It populates one and
  then copies the locator of that one to the other temporary
  LOB locator: */

sb4 copy_locators( OCIError    *errhp,
                   OCISvcCtx   *svchp,
                   OCIEnv      *envhp)
{
  sb4 return_code = 0;
  OCILobLocator *tblob;
  OCILobLocator *tblob2;
  OCILobLocator *bfile;
  ub4 amount = 4000;

  checkerr(errhp, OCIDescriptorAlloc((dvoid *)envhp, (dvoid **) &tblob,
                                     (ub4) OCI_DTYPE_LOB,
                                     (size_t) 0, (dvoid **) 0));

  checkerr(errhp, OCIDescriptorAlloc((dvoid *)envhp, (dvoid **) &tblob2,
                                     (ub4) OCI_DTYPE_LOB,
                                     (size_t) 0, (dvoid **) 0));

  checkerr(errhp, OCIDescriptorAlloc((dvoid *)envhp, (dvoid **) &bfile,
                                     (ub4) OCI_DTYPE_FILE,
                                     (size_t) 0, (dvoid **) 0));

  if(OCILobFileSetName(envhp, errhp, &bfile, (text *)"PHOTO_DIR",
                       (ub2)strlen("ADPHOTO_DIR"),
                       (text *)"monitor_photo_3060_11001",
                       (ub2)strlen("monitor_photo_3060_11001")))
  {
    printf("OCILobFileSetName FAILED in load_temp\n");
    return -1;
  }

  if (OCILobFileOpen(svchp, errhp, (OCILobLocator *) bfile, OCI_FILE_READONLY))
  {
    printf( "OCILobFileOpen FAILED for the bfile load_temp \n");
    return -1;
  }


  if(OCILobCreateTemporary(svchp,errhp, tblob,(ub2)0, SQLCS_IMPLICIT,
                           OCI_TEMP_BLOB, OCI_ATTR_NOCACHE,
```

```
                          OCI_DURATION_SESSION))
{
  (void) printf("FAILED: CreateTemporary() \n");
  return -1;
}

if(OCILobCreateTemporary(svchp,errhp, tblob2,(ub2)0, SQLCS_IMPLICIT,
                          OCI_TEMP_BLOB, OCI_ATTR_NOCACHE,
                          OCI_DURATION_SESSION))
{
  (void) printf("FAILED: CreateTemporary() \n");
  return -1;
}

if (OCILobOpen(svchp, errhp, (OCILobLocator *) tblob, OCI_LOB_READWRITE))
{
  printf( "OCILobOpen FAILED for temp LOB \n");
  return -1;
}

if (OCILobOpen(svchp, errhp, (OCILobLocator *) tblob2, OCI_LOB_READWRITE))
{
  printf( "OCILobOpen FAILED for temp LOB \n");
  return -1;
}

if(OCILobLoadFromFile(svchp, errhp, tblob, (OCILobLocator*)bfile,
                      (ub4)amount, (ub4)1,(ub4)1))
{
  printf("OCILobLoadFromFile failed \n");
  return_code = -1;
}

if(OCILobLocatorAssign(svchp,errhp, (CONST OCILobLocator *)tblob,&tblob2))
{

  printf("OCILobLocatorAssign failed \n");
  return_code = -1;
}

/* Close the lobs */
if (OCILobFileClose(svchp, errhp, (OCILobLocator *) bfile))
{
  printf( "OCILobClose FAILED for bfile \n");
  return -1;
}
```

```
  checkerr(errhp,(OCILobClose(svchp, errhp, (OCILobLocator *) tblob)));
  checkerr(errhp,(OCILobClose(svchp, errhp, (OCILobLocator *) tblob2)));


  /* Free the temporary lobs now that we are done using it */
  if(OCILobFreeTemporary(svchp, errhp, tblob))
  {
    printf("OCILobFreeTemporary FAILED \n");
    return -1;
  }

  if(OCILobFreeTemporary(svchp, errhp, tblob2))
  {
    printf("OCILobFreeTemporary FAILED \n");
    return -1;
  }
}
```

COBOL（Pro*COBOL）：一時 LOB の LOB ロケータのコピー

```
     * Copying a LOB locator for a temporary LOB
     * [Example script: 3887.pco]
       IDENTIFICATION DIVISION.
       PROGRAM-ID. TEMP-BLOB-COPY-LOCATOR.
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.

       01  USERID   PIC X(11) VALUES "SAMP/SAMP".

       01  TEMP-DEST      SQL-BLOB.
       01  TEMP-SRC       SQL-BLOB.
       01  SRC-BFILE      SQL-BFILE.
       01  DIR-ALIAS      PIC X(30) VARYING.
       01  FNAME          PIC X(30) VARYING.
       01  AMT            PIC S9(9) COMP.

       01 ORASLNRD        PIC 9(4).

           EXEC SQL INCLUDE SQLCA END-EXEC.
           EXEC ORACLE OPTION (ORACA=YES) END-EXEC.
           EXEC SQL INCLUDE ORACA END-EXEC.

       PROCEDURE DIVISION.
       TEMP-BLOB-COPY-LOCATOR.
```

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

* Allocate and initialize the BLOB locators:
     EXEC SQL ALLOCATE :TEMP-DEST END-EXEC.
     EXEC SQL ALLOCATE :TEMP-SRC END-EXEC.
     EXEC SQL ALLOCATE :SRC-BFILE END-EXEC.
     EXEC SQL
          LOB CREATE TEMPORARY :TEMP-DEST
     END-EXEC.
     EXEC SQL
          LOB CREATE TEMPORARY :TEMP-SRC
     END-EXEC.

* Set up the directory and file information:
     MOVE "ADPHOTO_DIR" TO DIR-ALIAS-ARR.
     MOVE 9 TO DIR-ALIAS-LEN.
     MOVE "keyboard_photo_3106_13001" TO FNAME-ARR.
     MOVE 16 TO FNAME-LEN.

     EXEC SQL
        LOB FILE SET :SRC-BFILE DIRECTORY = :DIR-ALIAS,
        FILENAME = :FNAME
     END-EXEC.

* Open source BFILE and destination temporary BLOB:
     EXEC SQL LOB OPEN :TEMP-SRC READ WRITE END-EXEC.
     EXEC SQL LOB OPEN :TEMP-DEST READ WRITE END-EXEC.
     EXEC SQL LOB OPEN :SRC-BFILE READ ONLY END-EXEC.

* MOVE the desired amount to copy to AMT:
     MOVE 5 TO AMT.
     EXEC SQL
          LOB LOAD :AMT FROM FILE :SRC-BFILE INTO :TEMP-SRC
     END-EXEC.

* Assign source BLOB locator to destination BLOB locator:
     EXEC SQL
          LOB ASSIGN :TEMP-SRC TO :TEMP-DEST
     END-EXEC.

     EXEC SQL LOB CLOSE :TEMP-SRC END-EXEC.
     EXEC SQL LOB CLOSE :TEMP-DEST END-EXEC.
     EXEC SQL LOB CLOSE :SRC-BFILE END-EXEC.
     EXEC SQL
```

```
            LOB FREE TEMPORARY :TEMP-SRC
        END-EXEC.
        EXEC SQL
            LOB FREE TEMPORARY :TEMP-DEST
        END-EXEC.
        EXEC SQL FREE :TEMP-SRC END-EXEC.
        EXEC SQL FREE :TEMP-DEST END-EXEC.
        EXEC SQL FREE :SRC-BFILE END-EXEC.
        STOP RUN.

    SQL-ERROR.
        EXEC SQL
            WHENEVER SQLERROR CONTINUE
        END-EXEC.
        MOVE ORASLNR TO ORASLNRD.
        DISPLAY " ".
        DISPLAY "ORACLE ERROR DETECTED ON LINE ", ORASLNRD, ":".
        DISPLAY " ".
        DISPLAY SQLERRMC.
        EXEC SQL ROLLBACK WORK RELEASE END-EXEC.
        STOP RUN.
```

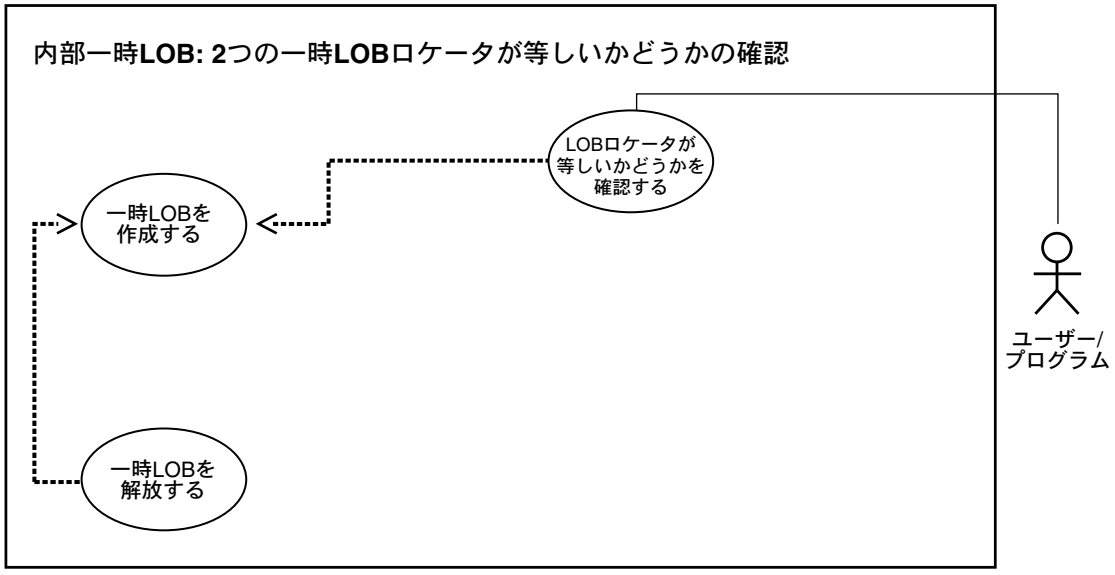
C/C++（Pro*C/C++）：一時 LOB の LOB ロケータのコピー

```
/* copying a lOB locator for a temporary LOB.  [Example script: 3888.pc]
#include <oci.h>
#include <stdio.h>
#include <sqlca.h>

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

void copyTempLobLocator_proc()
{
  OCIBlobLocator *Temp_loc1, *Temp_loc2;
  OCIBFileLocator *Lob_loc;
  char *Dir = "ADPHOTO_DIR", *Name = "monitor_photo_3060_11001";
  int Amount = 4096;

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
```

```
  /* Allocate and Create the Temporary LOBs: */
  EXEC SQL ALLOCATE :Temp_loc1;
  EXEC SQL ALLOCATE :Temp_loc2;
  EXEC SQL LOB CREATE TEMPORARY :Temp_loc1;
  EXEC SQL LOB CREATE TEMPORARY :Temp_loc2;

  /* Allocate and Initialize the BFILE Locator: */
  EXEC SQL ALLOCATE :Lob_loc;
  EXEC SQL LOB FILE SET :Lob_loc DIRECTORY = :Dir, FILENAME = :Name;

  /* Opening the LOBs is Optional: */
  EXEC SQL LOB OPEN :Lob_loc READ ONLY;
  EXEC SQL LOB OPEN :Temp_loc1 READ WRITE;
  EXEC SQL LOB OPEN :Temp_loc2 READ WRITE;

  /* Load a specified amount from the BFILE into the Temporary LOB: */
  EXEC SQL LOB LOAD :Amount FROM FILE :Lob_loc INTO :Temp_loc1;
  /* Assign Temp_loc1 to Temp_loc2 thereby creating a copy of the value of
     the Temporary LOB referenced by Temp_loc1 at this point in time: */
  EXEC SQL LOB ASSIGN :Temp_loc1 TO :Temp_loc2;

  /* Closing the LOBs is Mandatory if they have been Opened: */
  EXEC SQL LOB CLOSE :Lob_loc;
  EXEC SQL LOB CLOSE :Temp_loc1;
  EXEC SQL LOB CLOSE :Temp_loc2;

  /* Free the Temporary LOBs: */
  EXEC SQL LOB FREE TEMPORARY :Temp_loc1;
  EXEC SQL LOB FREE TEMPORARY :Temp_loc2;

  /* Release resources held by the Locators: */
  EXEC SQL FREE :Lob_loc;
  EXEC SQL FREE :Temp_loc1;
  EXEC SQL FREE :Temp_loc2;
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  copyTempLobLocator_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

2 つの一時 LOB ロケータが等しいかどうかの確認

図 11-16 利用図 : 2 つの一時 LOB ロケータが等しいかどうかの確認



参照 : 11-2 ページの表 11-1「利用モデル : 内部一時 LOB」を参照してください。

用途

一時 LOB の LOB ロケータが他の一時 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（第 5 章「ラージ・オブジェクト（LOB）: 詳細事項」の「読込み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関数またはファンクションのリストは、第 3 章「様々なプログラム環境での LOB のサポート」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）:『Oracle Call Interface プログラマーズ・ガイド』の第 16 章「その他の 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++）:『Pro*C/C++ Precompiler プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびディレクティブ」の「LOB ASSIGN（実行可能埋込み SQL 拡張機能）」、および『Oracle Call Interface プログラマーズ・ガイド』の第 16 章「その他の 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）: 10-168 ページの「Java（JDBC）: 2 つの LOB ロケータが等しいかどうかの確認」

使用例

ありません。

例

次のプログラム環境での例を示します。

- PL/SQL（DBMS_LOB）: 今回のリリースでは例は提供されません。
- C（OCI）: 2 つの一時 LOB ロケータが等しいかどうかの確認（11-133 ページ）
- COBOL（Pro*COBOL）: 今回のリリースでは例は提供されません。
- C/C++（Pro*C/C++）: 2 つの一時 LOB ロケータが等しいかどうかの確認（11-134 ページ）
- Visual Basic（OO4O）: 今回のリリースでは例は提供されません。
- Java（JDBC）: 今回のリリースでは例は提供されません。

C（OCI）: 2 つの一時 LOB ロケータが等しいかどうかの確認

```
/* Equality - Is one temporary LOB locator equal to another?  [Example script:
3889.c] */

sb4 ck_isequal (OCIError      *errhp,
                OCISvcCtx     *svchp,
                OCIStmt       *stmthp,
                OCIEnv        *envhp)
{
  OCILobLocator *loc1;
  OCILobLocator *loc2;
  boolean is_equal;
  is_equal= FALSE;
  if(OCILobCreateTemporary(svchp, errhp, loc1, (ub2)0, SQLCS_IMPLICIT,
                           OCI_TEMP_BLOB, OCI_ATTR_NOCACHE,
                           OCI_DURATION_SESSION))
  {
     (void) printf("FAILED: CreateTemporary() \n");
     return -1;
  }
   if(OCILobCreateTemporary(svchp, errhp, loc2, (ub2)0, SQLCS_IMPLICIT,
                            OCI_TEMP_BLOB, OCI_ATTR_NOCACHE,
                            OCI_DURATION_SESSION))
  {
     (void) printf("FAILED: CreateTemporary() \n");
     return -1;
  }

  if (OCILobIsEqual(envhp,loc1,loc2, &is_equal))
  {
    printf ("FAILED: OCILobLocatorIsEqual call\n");
    return -1;
  }
  if(is_equal)
  {
      fprintf (stderr,"LOB loators are equal \n");
      return -1;

   }else
   {
      fprintf(stderr,"LOB locators are not equal \n");
   }
   if(OCILobFreeTemporary(svchp,errhp,loc1))
   {
        printf("FAILED: OCILobFreeTemporary for temp LOB #1\n");
        return -1;
```

```
   }
  if(OCILobFreeTemporary(svchp,errhp,loc2))
   {
        printf("FAILED: OCILobFreeTemporary for temp LOB #2\n");
        return -1;
   }
OCILobDescriptor free????
   return 0;
  }
```

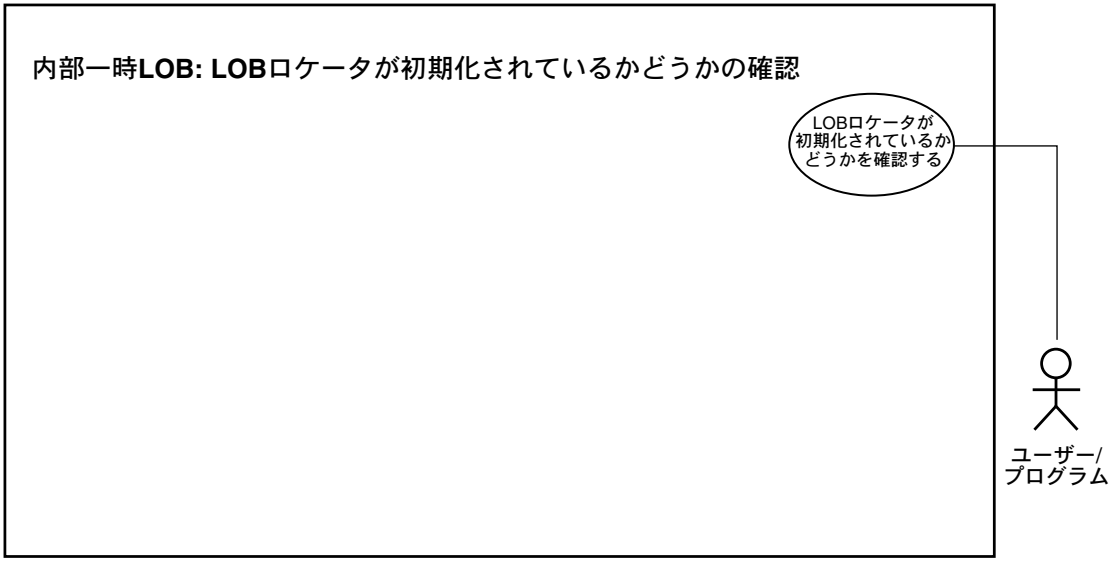
C/C++（Pro*C/C++）: 2 つの一時 LOB ロケータが等しいかどうかの確認

```
/* Equality - Is one LOB locator for a temporary LOB equal to another? */
/* [Example script: 3890.pc] */

#include <sql2oci.h>
#include <stdio.h>
#include <sqlca.h>

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

void seeTempLobLocatorsAreEqual_proc()
{
  OCIBlobLocator *Temp_loc1, *Temp_loc2;
  OCIBFileLocator *Lob_loc;
  char *Dir = "ADPHOTO_DIR", *Name = "monitor_photo_3060_11001";
  int Amount = 4096;
  OCIEnv *oeh;
  int isEqual = 0;

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  /* Allocate and Create the Temporary LOBs: */
  EXEC SQL ALLOCATE :Temp_loc1;
  EXEC SQL ALLOCATE :Temp_loc2;
  EXEC SQL LOB CREATE TEMPORARY :Temp_loc1;
  EXEC SQL LOB CREATE TEMPORARY :Temp_loc2;
  /* Allocate and Initialize the BFILE Locator: */
  EXEC SQL ALLOCATE :Lob_loc;
  EXEC SQL LOB FILE SET :Lob_loc DIRECTORY = :Dir, FILENAME = :Name;
```

```
  /* Opening the LOBs is Optional: */
  EXEC SQL LOB OPEN :Lob_loc READ ONLY;
  EXEC SQL LOB OPEN :Temp_loc1 READ WRITE;
  EXEC SQL LOB OPEN :Temp_loc2 READ WRITE;

  /* Load a specified amount from the BFILE into one of the Temporary LOBs: */
  EXEC SQL LOB LOAD :Amount FROM FILE :Lob_loc INTO :Temp_loc1;
  /* Retrieve the OCI Environment Handle: */
  (void) SQLEnvGet(SQL_SINGLE_RCTX, &oeh);

  /* Now assign Temp_loc1 to Temp_loc2 using Embedded SQL: */
  EXEC SQL LOB ASSIGN :Temp_loc1 TO :Temp_loc2;

  /* Determine if the Temporary LOBs are Equal: */
  (void) OCILobIsEqual(oeh, Temp_loc1, Temp_loc2, &isEqual);

  /* This time, isEqual should be 0 (FALSE): */
  printf("Locators %s equal\n", isEqual ? "are" : "are not");

  /* Assign Temp_loc1 to Temp_loc2 using C pointer assignment: */
  Temp_loc2 = Temp_loc1;

  /* Determine if the Temporary LOBs are Equal again: */
  (void) OCILobIsEqual(oeh, Temp_loc1, Temp_loc2, &isEqual);

  /* The value of isEqual should be 1 (TRUE) in this case: */
  printf("Locators %s equal\n", isEqual ? "are" : "are not");

  /* Closing the LOBs is Mandatory if they have been Opened: */
  EXEC SQL LOB CLOSE :Lob_loc;
  /* Note that because Temp_loc1 and Temp_loc2 are now equal, closing
     and freeing one will implicitely do the same to the other: */
  EXEC SQL LOB CLOSE :Temp_loc1;
  EXEC SQL LOB FREE TEMPORARY :Temp_loc1;
  /* Release resources held by the Locators: */
  EXEC SQL FREE :Lob_loc;
  EXEC SQL FREE :Temp_loc1;
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  seeTempLobLocatorsAreEqual_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

一時 LOB の LOB ロケータが初期化されているかどうかの確認

図 11-17 利用図 : 一時 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 : 11-2 ページの表 11-1「利用モデル : 内部一時 LOB」を参照してください。

用途

一時 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関数またはファンクションのリストは、第 3 章「様々なプログラム環境での LOB のサポート」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）:『Oracle Call Interface プログラマーズ・ガイド』の第 16 章「その他の 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++）:『Pro*C/C++ Precompiler プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびディレクティブ」の「LOB CREATE TEMPORARY（実行可能埋込み SQL 拡張機能）」、および『Oracle Call Interface プログラマーズ・ガイド』の第 16 章「その他の 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）: 第 10 章「内部永続 LOB」の「C（OCI）: LOB ロケータが初期化されているかどうかの確認」

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確認します。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での例を示します。

- PL/SQL（DBMS_LOB）: 今回のリリースでは例は提供されません。
- C（OCI）: 一時 LOB の LOB ロケータが初期化されているかどうかの確認（11-137 ページ）
- COBOL（Pro*COBOL）: 今回のリリースでは例は提供されません。
- 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（11-138 ページ）
- Visual Basic（OO4O）: 今回のリリースでは例は提供されません。
- Java（JDBC）: 今回のリリースでは例は提供されません。

C（OCI）: 一時 LOB の LOB ロケータが初期化されているかどうかの確認

```
/* Is a LOB locator for a temporary LOB is initialized? [Example script:3892.c]
   This function takes a LOB locator and checks if it is initialized. If it is
   initialized, it prints out a message, "LOB is initialized".
   Otherwise, it says "LOB is not initialized". This function returns
   0 if it completes successfully, -1 if it doesn't. */

sb4 ck_isinit (OCILobLocator *lob_loc,
               OCIError      *errhp,
               OCISvcCtx     *svchp,
               OCIStmt       *stmthp,
               OCIEnv        *envhp)
{
```

```
  boolean is_init;
  is_init= FALSE;
  if (OCILobLocatorIsInit(envhp,errhp, lob_loc, &is_init))
  {
    printf ("FAILED: OCILobLocatorIsInit call\n");
    return -1;
  }
  if(is_init)
  {
      printf ("LOB is initialized\n");
  }else
  {
      printf("LOB is not initialized\n");
   }
   return 0;
   }
```

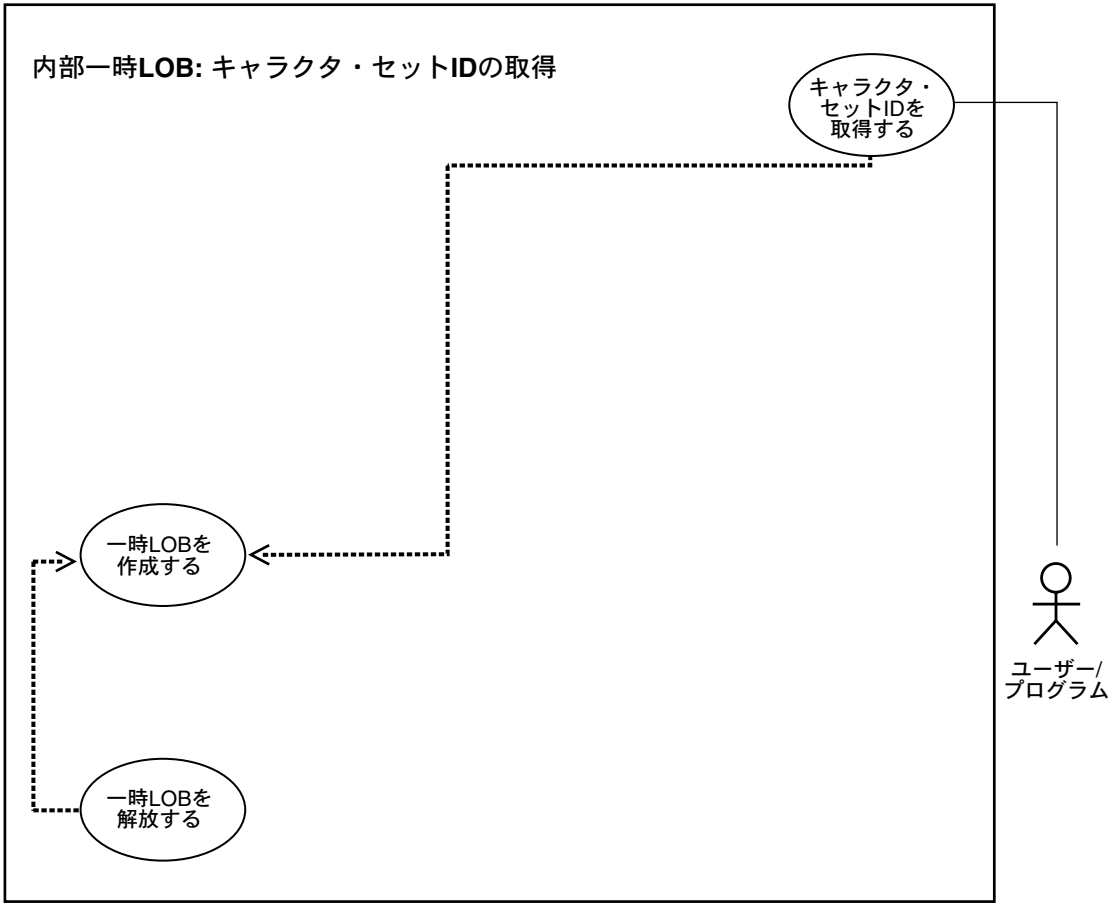
C/C++（Pro*C/C++）: LOB ロケータが初期化されているかどうかの確認

```
/* Is a LOB locator for a temporary LOB initialized? [Example script: 3893.pc] */

#include <sql2oci.h>
#include <stdio.h>
#include <sqlca.h>

void Sample_Error()
{
  EXEC SQL WHENEVER SQLERROR CONTINUE;
  printf("%.*s\n", sqlca.sqlerrm.sqlerrml, sqlca.sqlerrm.sqlerrmc);
  EXEC SQL ROLLBACK WORK RELEASE;
  exit(1);
}
void tempLobLocatorIsInit_proc()
{
  OCIBlobLocator *Temp_loc;
  OCIEnv *oeh;
  OCIError *err;
  boolean isInitialized = 0;

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  EXEC SQL ALLOCATE :Temp_loc;
  EXEC SQL LOB CREATE TEMPORARY :Temp_loc;
  /* Get the OCI Environment Handle using a SQLLIB Routine: */
  (void) SQLEnvGet(SQL_SINGLE_RCTX, &oeh);
  /* Allocate the OCI Error Handle: */
  (void) OCIHandleAlloc((dvoid *)oeh, (dvoid **)&err,
```

```
                        (ub4)OCI_HTYPE_ERROR, (ub4)0, (dvoid **)0);
  /* Use the OCI to determine if the locator is Initialized */
  (void) OCILobLocatorIsInit(oeh, err, Temp_loc, &isInitialized);
  if (isInitialized)
    printf("Locator is initialized\n");
  else
    printf("Locator is not initialized\n");
  /* Note that in this example, the locator is initialized. */
  /* Deallocate the OCI Error Handle: */
  (void) OCIHandleFree(err, OCI_HTYPE_ERROR);
  /* Free the Temporary LOB */
  EXEC SQL LOB FREE TEMPORARY :Temp_loc;
  /* Release resources held by the locator: */
  EXEC SQL FREE :Temp_loc;
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  tempLobLocatorIsInit_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

一時 LOB のキャラクタ・セット ID の取得

図 11-18 利用図 : 一時 LOB のキャラクタ・セット ID の取得



参照 : 11-2 ページの表 11-1「利用モデル : 内部一時 LOB」を参照してください。

用途

一時 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関数またはファンクションのリストは、第 3 章「様々なプログラム環境での LOB のサポート」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）:『Oracle Call Interface プログラマーズ・ガイド』の第 16 章「その他の 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）: 10-177 ページの「C（OCI）: キャラクタ・セット ID の取得」

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での例を示します。

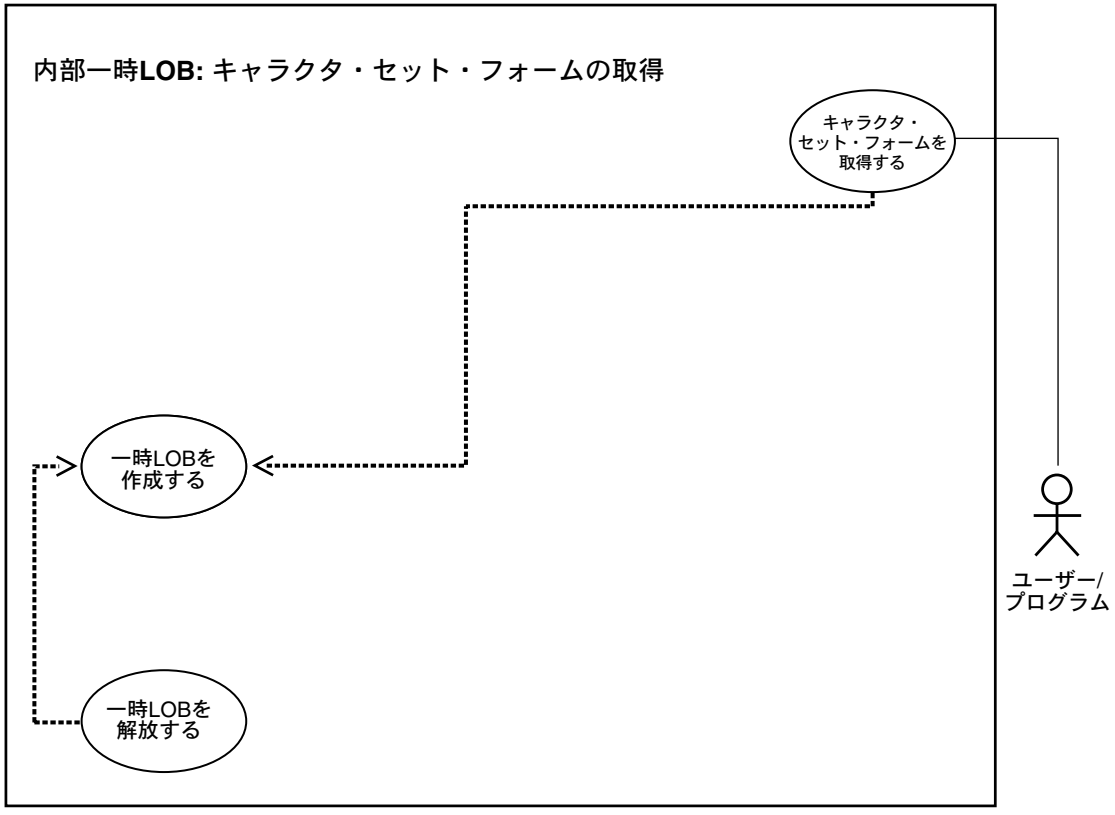
- PL/SQL（DBMS_LOB）: 今回のリリースでは例は提供されません。
- C（OCI）: 一時 LOB のキャラクタ・セット ID の取得（11-142 ページ）
- COBOL（Pro*COBOL）: 今回のリリースでは例は提供されません。
- C/C++（Pro*C/C++）: 今回のリリースでは例は提供されません。
- Visual Basic（OO4O）: 今回のリリースでは例は提供されません。
- Java（JDBC）: 今回のリリースでは例は提供されません。

C（OCI）：一時 LOB のキャラクタ・セット ID の取得

```
/* Finding the character set id of a temporary LOB. [Example script: 3894.c]
   This function takes a LOB locator and prints the character set id of the LOB.
   The function returns 0 if it completes successfully, -1 if it doesn't. */

sb4 get_charsetid (OCILobLocator *lob_loc,
                   OCIError      *errhp,
                   OCISvcCtx     *svchp,
                   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;

}
```

一時 LOB のキャラクタ・セット・フォームの取得

図 11-19 利用図 : 一時 LOB のキャラクタ・セット・フォームの取得



参照： 11-2 ページの表 11-1「利用モデル : 内部一時 LOB」を参照してください。

用途

一時 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関数またはファンクションのリストは、第 3 章「様々なプログラム環境での LOB のサポート」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）:『Oracle Call Interface プログラマーズ・ガイド』の第 16 章「その他の 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）: 10-181 ページの「C（OCI）: キャラクタ・セット・フォームの取得」

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での例を示します。

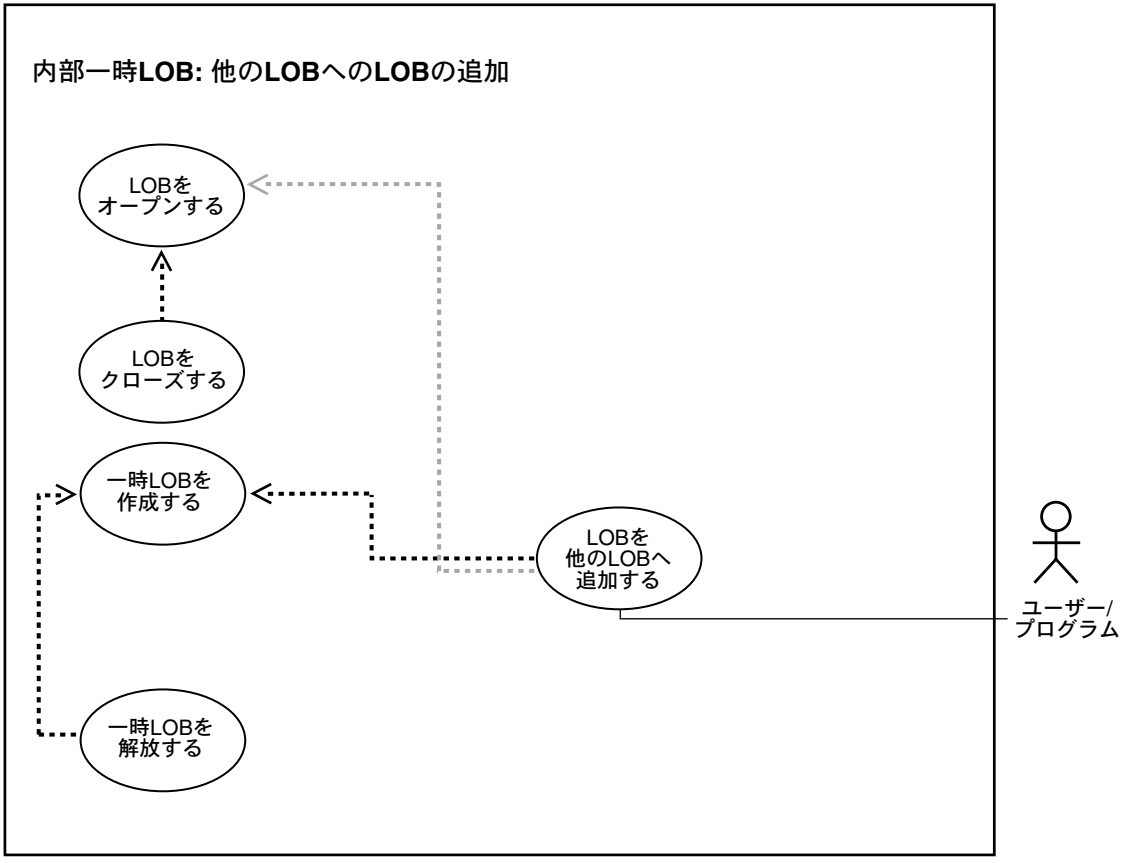
- PL/SQL（DBMS_LOB）: 今回のリリースでは例は提供されません。
- C（OCI）: 一時 LOB のキャラクタ・セット・フォームの取得（11-145 ページ）
- COBOL（Pro*COBOL）: 今回のリリースでは例は提供されません。
- C/C++（Pro*C/C++）: 今回のリリースでは例は提供されません。
- Visual Basic（OO4O）: 今回のリリースでは例は提供されません。
- Java（JDBC）: 今回のリリースでは例は提供されません。

C（OCI）: 一時 LOB のキャラクタ・セット・フォームの取得

```
/* Finding the character set form of a temporary LOB [Example script: 3895.c]
   This function takes a LOB locator and prints out the character set form for
   the LOB. It returns 0 if it completes successfully, -1 if it doesn't. */

sb4 get_charsetform (OCILobLocator *lob_loc,
                     OCIError      *errhp,
                     OCISvcCtx     *svchp,
                     OCIStmt       *stmthp,
                     OCIEnv        *envhp)
{
  ub1 charsetform =0;
  if(OCILobCreateTemporary(svchp,errhp,lob_loc,(ub2)0,
                           SQLCS_IMPLICIT, OCI_TEMP_CLOB, OCI_ATTR_NOCACHE,
                           OCI_DURATION_SESSION))
  {
     (void) printf("FAILED: CreateTemporary() \n");
     return -1;
  }

  if (OCILobCharSetForm(envhp,errhp, lob_loc, &charsetform))
  {
    printf ("FAILED: OCILobCharSetForm call\n");
    return -1;
  }
  fprintf (stderr,"LOB charsetform is %d\n",charsetform);

  if(OCILobFreeTemporary(svchp,errhp,lob_loc))
  {
        printf("FAILED: OCILobFreeTemporary \n");
        return -1;
  }
  return 0;
  }
```

他の LOB への一時 LOB の追加

図 11-20 利用図 : 他の LOB への一時 LOB の追加



参照 : 11-2 ページの表 11-1「利用モデル : 内部一時 LOB」を参照してください。

用途

一時 LOB を他の LOB に追加します。

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関数またはファンクションのリストは、第 3 章「様々なプログラム環境での LOB のサポート」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）：『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』の第 23 章「DBMS_LOB」の「DBMS_LOB サブプログラムの要約」の「APPEND プロシージャ」
- C（OCI）：『Oracle Call Interface プログラマーズ・ガイド』の第 16 章「その他の OCI リレーショナル関数」の「LOB 関数」の「OCILobAppend()」
- COBOL（Pro*COBOL）：『Pro*COBOL Precompiler プログラマーズ・ガイド』の LOB に関する詳細、LOB 文の使用上の注意および付録 F「埋込み SQL およびプリコンパイラ・ディレクティブ」の「LOB APPEND（実行可能埋込み SQL 拡張機能）」
- C/C++（Pro*C/C++）：『Pro*C/C++ Precompiler プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびディレクティブ」の「LOB APPEND（実行可能埋込み SQL 拡張機能）」
- Visual Basic（OO4O）：参照マニュアルはありません。
- Java（JDBC）：10-191 ページの「Java（JDBC）：他の LOB への LOB の追加」

使用例

次の例では、サウンドの 1 セグメントを別のセグメントへ追加する作業を行います。

例

次のプログラム環境での例を示します。

- PL/SQL（DBMS_LOB）：他の LOB への一時 LOB の追加（11-148 ページ）
- C（OCI）：他の LOB への一時 LOB の追加（11-148 ページ）
- COBOL（Pro*COBOL）：他の LOB への一時 LOB の追加（11-151 ページ）
- C/C++（Pro*C/C++）：他の LOB への一時 LOB の追加（11-154 ページ）

- Visual Basic（OO4O）: 今回のリリースでは例は提供されません。
- Java（JDBC）: 今回のリリースでは例は提供されません。

PL/SQL（DBMS_LOB）: 他の LOB への一時 LOB の追加

```
/* Appending one temporary LOB to another [Example script: 3896.sql]
   Procedure appendTempLOB_proc is not part of DBMS_LOB package. */

CREATE OR REPLACE PROCEDURE appendTempLOB_proc IS
   Dest_loc2 CLOB;
   Dest_loc  CLOB;
   Amount    NUMBER;
   Src_loc   BFILE := BFILENAME('ADPHOTO_DIR', 'monitor_photo_3060_11001');
BEGIN
   DBMS_LOB.CREATETEMPORARY(Dest_loc,TRUE);
   DBMS_LOB.CREATETEMPORARY(Dest_loc2,TRUE);
   DBMS_LOB.OPEN(Dest_loc,DBMS_LOB.LOB_READWRITE);
   DBMS_LOB.OPEN(Dest_loc2,DBMS_LOB.LOB_READWRITE);
   DBMS_LOB.OPEN(Src_loc,DBMS_LOB.LOB_READWRITE);
   Amount := 32767;
   DBMS_LOB.LOADFROMFILE(Dest_loc, Src_loc, Amount);
   DBMS_LOB.LOADFROMFILE(Dest_loc2, Src_loc, Amount);
   DBMS_LOB.APPEND(Dest_loc, Dest_loc2);
   /* Close the temporary lobs and then free them: */
   DBMS_LOB.CLOSE(Dest_loc);
   DBMS_LOB.CLOSE(Dest_loc2);
   DBMS_LOB.CLOSE(Src_loc);
   DBMS_LOB.FREETEMPORARY(Dest_loc);
   DBMS_LOB.FREETEMPORARY(Dest_loc2);
END;
```

C（OCI）: 他の LOB への一時 LOB の追加

```
/* Appending one temporary LOB to another  [Example script: 3897.c]
   This function takes two temporary LOB locators and appends the second
   LOB to the first one. It returns 0 if it completes successfully, -1,
   otherwise.*/

sb4 append_temp_lobs (OCIError      *errhp,
                      OCISvcCtx     *svchp,
                      OCIStmt       *stmthp,
                      OCIEnv        *envhp)
{
  OCILobLocator *tblob;
  OCILobLocator *tblob2;
```

```
OCILobLocator *bfile;
ub4 amt = 4000;
sb4 return_code = 0;

printf("in append \n");
if(OCIDescriptorAlloc((dvoid *)envhp, (dvoid **) &tblob,
                      (ub4) OCI_DTYPE_LOB,
                      (size_t) 0, (dvoid **) 0))
{
   printf("OCIDescriptor Alloc FAILED in print_length\n");
   return -1;
}
if(OCIDescriptorAlloc((dvoid *)envhp, (dvoid **) &tblob2,
                      (ub4) OCI_DTYPE_LOB,
                      (size_t) 0, (dvoid **) 0))
{
   printf("OCIDescriptor Alloc FAILED in print_length\n");
   return -1;
}

if(OCIDescriptorAlloc((dvoid *)envhp, (dvoid **) &bfile,
                      (ub4) OCI_DTYPE_FILE,
                      (size_t) 0, (dvoid **) 0))
{
   printf("OCIDescriptor Alloc FAILED in print_length\n");
   return -1;
}

/* Set the BFILE to point to the monitor_photo file */
if(OCILobFileSetName(envhp, errhp, &bfile, (text *)"PHOTO_DIR",
                     (ub2)strlen("ADPHOTO_DIR"),
                     (text *)"monitor_photo_3060_11001",
                     (ub2)strlen("monitor_photo_3060_11001")))
{
  printf("OCILobFileSetName FAILED\n");
  return -1;
}

if (OCILobFileOpen(svchp, errhp, (OCILobLocator *) bfile, OCI_LOB_READONLY))
{
  printf( "OCILobFileOpen FAILED  for the bfile\n");
  return_code = -1;
}
```

```
if(OCILobCreateTemporary(svchp,errhp,tblob,(ub2)0, SQLCS_IMPLICIT,
                         OCI_TEMP_CLOB, OCI_ATTR_NOCACHE,
                         OCI_DURATION_SESSION))
{
   (void) printf("FAILED: CreateTemporary() \n");
   return_code = -1;
}

if(OCILobCreateTemporary(svchp,errhp,tblob2,(ub2)0, SQLCS_IMPLICIT,
                         OCI_TEMP_CLOB, OCI_ATTR_NOCACHE,
                         OCI_DURATION_SESSION))
{
   (void) printf("FAILED: CreateTemporary() \n");
   return_code = -1;
}

/* Open the lobs: */
if (OCILobOpen(svchp, errhp, (OCILobLocator *) tblob, OCI_LOB_READWRITE))
{
  printf( "OCILobOpen FAILED for temp LOB tblob \n");
  return_code =  -1;
}

if (OCILobOpen(svchp, errhp, (OCILobLocator *) tblob2, OCI_LOB_READWRITE))
{
  printf( "OCILobOpen FAILED for temp LOB, tblob2 \n");
  return_code =  -1;
}

/* Populate the source temporary LOB with some data: */

if(OCILobLoadFromFile(svchp, errhp, tblob,(OCILobLocator*)bfile,
                    (ub4)amt, (ub4)1,(ub4)1))
{
  printf( "OCILobLoadFromFile FAILED\n");
  return_code = -1;
}

/* Append the source LOB to the dest temp LOB: */
if (OCILobAppend(svchp, errhp,tblob2,tblob))
{
  printf ("FAILED: OCILobAppend in append_temp_lobs\n");
  return_code = -1;
}else
```

```
 {
   printf("Append succeeded\n");
 }

 if(OCILobFreeTemporary(svchp,errhp,tblob))
 {
       printf("FAILED: OCILobFreeTemporary \n");
       return_code = -1;
 }
if(OCILobFreeTemporary(svchp,errhp,tblob2))
{
     printf("FAILED: OCIlobFreeTemporary\n");
     return_code = -1;
}
return return_code;
}
```

COBOL（Pro*COBOL）：他の LOB への一時 LOB の追加

```
* Appending one temporary LOB to another  [Example script: 3898.pco]
  IDENTIFICATION DIVISION.
  PROGRAM-ID. APPEND-TEMP-BLOB.
  ENVIRONMENT DIVISION.
  DATA DIVISION.
  WORKING-STORAGE SECTION.

 * Define the username and password:
  01  USERID   PIC X(11) VALUES "SAMP/SAMP".

 * Define the temporary LOBs and the source BFILE:
  01  TEMP-BLOB1     SQL-BLOB.
  01  TEMP-BLOB2     SQL-BLOB.
  01  SRC-BFILE      SQL-BFILE.
  01  AMT            PIC S9(9) COMP.
  01  DIR-ALIAS      PIC X(30) VARYING.
  01  FNAME          PIC X(30) VARYING.

 * Define the source position in BFILE:
  01  SRC-POS        PIC S9(9) COMP.

 * Define the line number in case of error:
  01  ORASLNRD       PIC 9(4).
```

```
     EXEC SQL INCLUDE SQLCA END-EXEC.
     EXEC ORACLE OPTION (ORACA=YES) END-EXEC.
     EXEC SQL INCLUDE ORACA END-EXEC.
 PROCEDURE DIVISION.
 APPEND-TEMP-BLOB.
     EXEC SQL WHENEVER SQLERROR DO PERFORM SQL-ERROR END-EXEC.
     EXEC SQL
          CONNECT :USERID
     END-EXEC.

* Allocate and initialize the BLOB locators:
     EXEC SQL ALLOCATE :TEMP-BLOB1 END-EXEC.
     EXEC SQL ALLOCATE :TEMP-BLOB2 END-EXEC.
     EXEC SQL ALLOCATE :SRC-BFILE END-EXEC.
     EXEC SQL
          LOB CREATE TEMPORARY :TEMP-BLOB1
     END-EXEC.
     EXEC SQL
          LOB CREATE TEMPORARY :TEMP-BLOB2
     END-EXEC.

* Set up the directory and file information:
     MOVE "ADPHOTO_DIR" TO DIR-ALIAS-ARR.
     MOVE 9 TO DIR-ALIAS-LEN.
     MOVE "keyboard_photo_3106_11001" TO FNAME-ARR.
     MOVE 16 TO FNAME-LEN.

     EXEC SQL
        LOB FILE SET :SRC-BFILE DIRECTORY = :DIR-ALIAS,
        FILENAME = :FNAME
     END-EXEC.

* Open source BFILE and destination temporary BLOB:
     EXEC SQL LOB OPEN :TEMP-BLOB2 READ WRITE END-EXEC.
     EXEC SQL LOB OPEN :TEMP-BLOB1 READ WRITE END-EXEC.
     EXEC SQL LOB OPEN :SRC-BFILE READ ONLY END-EXEC.
     DISPLAY "LOBs opened.".
```

```
* Move the desired amount to copy to AMT:
     MOVE 5 TO AMT.
     MOVE 1 TO SRC-POS.
     EXEC SQL
          LOB LOAD :AMT FROM FILE :SRC-BFILE
          AT :SRC-POS INTO :TEMP-BLOB1
     END-EXEC.

     ADD 1 TO AMT GIVING SRC-POS.
     EXEC SQL
          LOB LOAD :AMT FROM FILE :SRC-BFILE
          AT :SRC-POS INTO :TEMP-BLOB2
     END-EXEC.
     DISPLAY "Temporary LOBs loaded".

     EXEC SQL
          LOB APPEND :TEMP-BLOB2 TO :TEMP-BLOB1
     END-EXEC.
     DISPLAY "LOB APPEND complete.".

     EXEC SQL
          LOB FREE TEMPORARY :TEMP-BLOB1
     END-EXEC.
     EXEC SQL
          LOB FREE TEMPORARY :TEMP-BLOB2
     END-EXEC.
     EXEC SQL FREE :TEMP-BLOB1 END-EXEC.
     EXEC SQL FREE :TEMP-BLOB2 END-EXEC.
     EXEC SQL FREE :SRC-BFILE END-EXEC.
     STOP RUN.

 SQL-ERROR.
     EXEC SQL
         WHENEVER SQLERROR CONTINUE
     END-EXEC.
     MOVE ORASLNR TO ORASLNRD
     DISPLAY " ".
     DISPLAY "ORACLE ERROR DETECTED ON LINE ", ORASLNRD, ":".
     DISPLAY " ".
     DISPLAY SQLERRMC.
     EXEC SQL ROLLBACK WORK RELEASE END-EXEC.
     STOP RUN.
```

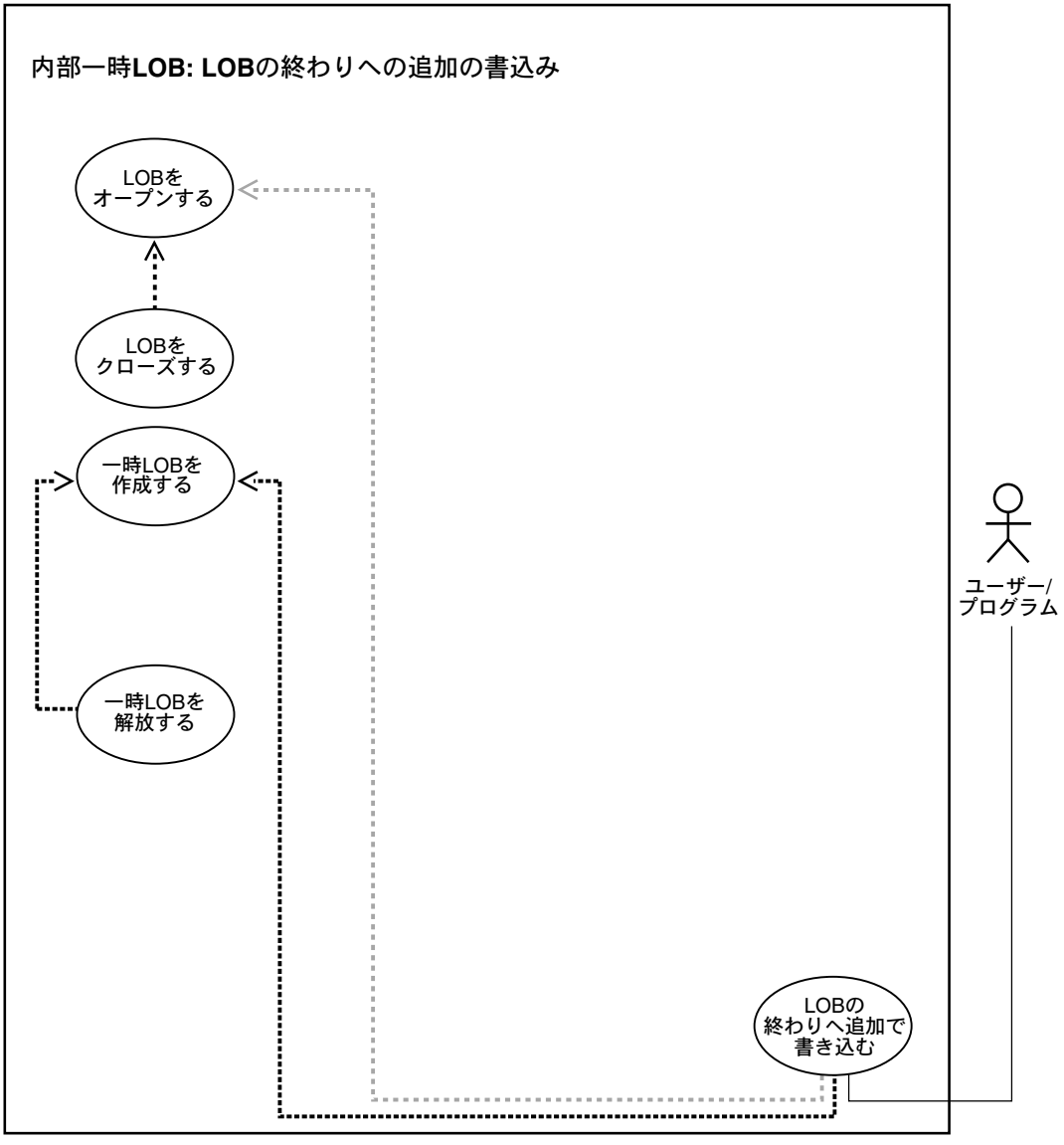
C/C++（Pro*C/C++）：他の LOB への一時 LOB の追加

```
/* Appending one temporary LOB to another.  [Example script: 3899.pc] */

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

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

void appendTempLOB_proc()
{
  OCIBlobLocator *Temp_loc1, *Temp_loc2;
  OCIBFileLocator *Lob_loc;
  char *Dir = "ADPHOTO_DIR", *Name = "monitor_photo_3060_11001";
  int Amount = 2048;
  int Position = 1;

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  /* Allocate and Create the Temporary LOBs: */
  EXEC SQL ALLOCATE :Temp_loc1;
  EXEC SQL ALLOCATE :Temp_loc2;
  EXEC SQL LOB CREATE TEMPORARY :Temp_loc1;
  EXEC SQL LOB CREATE TEMPORARY :Temp_loc2;
  /* Allocate and Initialize the BFILE Locator: */
  EXEC SQL ALLOCATE :Lob_loc;
  EXEC SQL LOB FILE SET :Lob_loc DIRECTORY = :Dir, FILENAME = :Name;

  /* Opening the LOBs is Optional: */
  EXEC SQL LOB OPEN :Lob_loc READ ONLY;
  EXEC SQL LOB OPEN :Temp_loc1 READ WRITE;
  EXEC SQL LOB OPEN :Temp_loc2 READ WRITE;

  /* Load a specified amount from the BFILE into the first Temporary LOB: */
  EXEC SQL LOB LOAD :Amount FROM FILE :Lob_loc AT :Position INTO :Temp_loc1;

  /* Set the Position for the next load from the same BFILE: */
  Position = Amount + 1;
```

```
  /* Load a second amount from the BFILE into the second Temporary LOB: */
  EXEC SQL LOB LOAD :Amount FROM FILE :Lob_loc AT :Position INTO :Temp_loc2;

  /* Append the second Temporary LOB to the end of the first one: */
  EXEC SQL LOB APPEND :Temp_loc2 TO :Temp_loc1;

  /* Closing the LOBs is Mandatory if they have been Opened: */
  EXEC SQL LOB CLOSE :Lob_loc;
  EXEC SQL LOB CLOSE :Temp_loc1;
  EXEC SQL LOB CLOSE :Temp_loc2;

  /* Free the Temporary LOBs: */
  EXEC SQL LOB FREE TEMPORARY :Temp_loc1;
  EXEC SQL LOB FREE TEMPORARY :Temp_loc2;

  /* Release resources held by the Locators: */
  EXEC SQL FREE :Lob_loc;
  EXEC SQL FREE :Temp_loc1;
  EXEC SQL FREE :Temp_loc2;
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  appendTempLOB_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

一時 LOB への追加の書込み

図 11-21 利用図 : 一時 LOB への追加の書込み



参照： 11-2 ページの表 11-1「利用モデル : 内部一時 LOB」を参照してください。

用途

一時 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関数またはファンクションのリストは、第 3 章「様々なプログラム環境での LOB のサポート」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）：『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』の第 23 章「DBMS_LOB」の「DBMS_LOB サブプログラムの要約」の「WRITEAPPEND プロシージャ」
- C（OCI）：『Oracle Call Interface プログラマーズ・ガイド』の第 16 章「その他の OCI リレーショナル関数」の「LOB 関数」の「OCILobWriteAppend()」
- COBOL（Pro*COBOL）：『Pro*COBOL Precompiler プログラマーズ・ガイド』の LOB に関する詳細、LOB 文の使用上の注意および付録 F「埋込み SQL およびプリコンパイラ・ディレクティブ」の「LOB WRITE（実行可能埋込み SQL 拡張機能）」
- C/C++（Pro*C/C++）：『Pro*C/C++ Precompiler プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびディレクティブ」の「LOB WRITE（実行可能埋込み SQL 拡張機能）」
- Visual Basic（OO4O）：参照マニュアルはありません。
- Java（JDBC）：10-200 ページの「Java（JDBC）：LOB の終わりへの追加の書込み」

使用例

次の例では、イメージ・ファイルから、32,767 バイトのデータを読み込み、一時 LOB に追加します。

例

次のプログラム環境での例を示します。

- PL/SQL（DBMS_LOB）: 一時 LOB への追加の書込み（11-158 ページ）
- C（OCI）: 一時 LOB への追加の書込み（11-159 ページ）
- COBOL（Pro*COBOL）: 一時 LOB への追加の書込み（11-160 ページ）
- C/C++（Pro*C/C++）: 一時 LOB への追加の書込み（11-162 ページ）
- Visual Basic（OO4O）: 今回のリリースでは例は提供されません。
- Java（JDBC）: 今回のリリースでは例は提供されません。

PL/SQL（DBMS_LOB）: 一時 LOB への追加の書込み

```
/* Write-appending to a temporary LOB.  [Example script: 3900.sql]
   Procedure writeAppendTempLOB_proc is not part of DBMS_LOB package.
   This procedure reads in 32767 bytes of data from the monitor_photo_3060_11001
   file and appends it to a temporary LOB. */

CREATE OR REPLACE PROCEDURE writeAppendTempLOB_proc IS
   Lob_loc    BLOB;
   Buffer     RAW(32767);
   Src_loc    BFILE := BFILENAME('ADPHOTO_DIR', 'monitor_photo_3060_11001');
   Amount     Binary_integer := 32767;
   Position   Binary_integer := 128;
BEGIN
   DBMS_LOB.CREATETEMPORARY(Lob_loc,TRUE);
   /* Opening the temporary LOB is optional: */
   DBMS_LOB.OPEN(Lob_loc,DBMS_LOB.LOB_READWRITE);
   /* Opening the FILE is mandatory: */
   DBMS_LOB.OPEN(Src_loc, DBMS_LOB.LOB_READONLY);
   /* Fill the buffer with data: */
   DBMS_LOB.LOADFROMFILE (Lob_loc,Src_loc, Amount);

   /* Append the data from the buffer to the end of the LOB: */
   DBMS_LOB.WRITEAPPEND(Lob_loc, Amount, Buffer);
   DBMS_LOB.CLOSE(Src_loc);
   DBMS_LOB.CLOSE(Lob_loc);
   DBMS_LOB.FREETEMPORARY(Lob_loc);
END;
```

C（OCI）：一時 LOB への追加の書込み

```
/* Write-appending to a temporary LOB  [Example script: 3901.c] */

#define MAXBUFLEN 32767
sb4 write_append_temp_lobs (OCIError      *errhp,
                            OCISvcCtx     *svchp,
                            OCIStmt       *stmthp,
                            OCIEnv        *envhp)
{
  OCIClobLocator *tclob;
  unsigned int Total = 40000;
  unsigned int amtp;
  unsigned int  nbytes;
  ub1 bufp[MAXBUFLEN];

  /* Allocate the locators desriptors: */
  (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &tclob ,
                            (ub4)OCI_DTYPE_LOB, (size_t) 0, (dvoid **) 0);

  if(OCILobCreateTemporary(svchp, errhp, tclob, (ub2)0,  SQLCS_IMPLICIT,
           OCI_TEMP_CLOB, OCI_ATTR_NOCACHE, OCI_DURATION_SESSION))
  {
     (void) printf("FAILED: CreateTemporary() \n");
     return -1;
  }

  /* Open the CLOB */
  printf("calling open \n");
  checkerr (errhp, (OCILobOpen(svchp, errhp, tclob, OCI_LOB_READWRITE)));

  nbytes = MAXBUFLEN;   /* We will use Streaming via Standard Polling */

  /* Fill the Buffer with nbytes worth of Data */
  memset(bufp,'a',32767);

  amtp = sizeof(bufp);
  /* Setting Amount to 0 streams the data until use specifies OCI_LAST_PIECE */

  printf("calling write append \n");
  checkerr (errhp, OCILobWriteAppend (svchp, errhp, tclob, &amtp,
                                      bufp, nbytes, OCI_ONE_PIECE, (dvoid *)0,
                                      (sb4 (*)(dvoid*,dvoid*,ub4*,ub1 *))0,
                                       0, SQLCS_IMPLICIT));
```

```
printf("calling close \n");
/* Closing the LOB is mandatory if you have opened it: */
checkerr (errhp, OCILobClose(svchp, errhp, tclob));

/* Free the temporary LOB: */
printf("calling free\n");
checkerr(errhp,OCILobFreeTemporary(svchp,errhp,tclob));

/* Free resources held by the locators: */
(void) OCIDescriptorFree((dvoid *) tclob, (ub4) OCI_DTYPE_LOB);
}
```

COBOL（Pro*COBOL）: 一時 LOB への追加の書込み

```
* Write-appending to a temporary LOB. [Example script: 3902.pco]
  IDENTIFICATION DIVISION.
  PROGRAM-ID. WRITE-APPEND-TEMP.
  ENVIRONMENT DIVISION.
  DATA DIVISION.
  WORKING-STORAGE SECTION.

  01  USERID   PIC X(11) VALUES "SAMP/SAMP".
  01  TEMP-BLOB      SQL-BLOB.
  01  SRC-BFILE      SQL-BFILE.
  01  BUFFER         PIC X(2048).
  01  DIR-ALIAS      PIC X(30) VARYING.
  01  FNAME          PIC X(20) VARYING.
  01  DIR-IND        PIC S9(4) COMP.
  01  FNAME-IND      PIC S9(4) COMP.
  01  AMT            PIC S9(9) COMP VALUE 10.
      EXEC SQL VAR BUFFER IS RAW(2048) END-EXEC.
  01  ORASLNRD        PIC 9(4).

      EXEC SQL INCLUDE SQLCA END-EXEC.
      EXEC ORACLE OPTION (ORACA=YES) END-EXEC.
      EXEC SQL INCLUDE ORACA END-EXEC.

  PROCEDURE DIVISION.
  WRITE-APPEND-TEMP.

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

```
* Allocate and initialize the BFILE and BLOB locators:
     EXEC SQL ALLOCATE :TEMP-BLOB END-EXEC.
     EXEC SQL ALLOCATE :SRC-BFILE END-EXEC.
     EXEC SQL
          LOB CREATE TEMPORARY :TEMP-BLOB
     END-EXEC.

* Set up the directory and file information:
     MOVE "ADPHOTO_DIR" TO DIR-ALIAS-ARR.
     MOVE 9 TO DIR-ALIAS-LEN.
     MOVE "keyboard_photo_3106_11001" TO FNAME-ARR.
     MOVE 16 TO FNAME-LEN.

     EXEC SQL
        LOB FILE SET :SRC-BFILE DIRECTORY = :DIR-ALIAS,
        FILENAME = :FNAME
     END-EXEC.

* Open source BFILE and destination temporary BLOB:
     EXEC SQL LOB OPEN :TEMP-BLOB READ WRITE END-EXEC.
     EXEC SQL LOB OPEN :SRC-BFILE READ ONLY END-EXEC.
     EXEC SQL
          LOB LOAD :AMT FROM FILE :SRC-BFILE INTO :TEMP-BLOB
     END-EXEC.

  MOVE "262626" TO BUFFER.
  MOVE 3 TO AMT.
* Append the data in BUFFER to TEMP-BLOB:
     EXEC SQL
          LOB WRITE APPEND :AMT FROM :BUFFER INTO :TEMP-BLOB
     END-EXEC.
* Close the LOBs:
     EXEC SQL LOB CLOSE :SRC-BFILE END-EXEC.
     EXEC SQL LOB CLOSE :TEMP-BLOB END-EXEC.

* Free the temporary LOB:
     EXEC SQL
          LOB FREE TEMPORARY :TEMP-BLOB
     END-EXEC.

* And free the LOB locators:
     EXEC SQL FREE :TEMP-BLOB END-EXEC.
     EXEC SQL FREE :SRC-BFILE END-EXEC.
     STOP RUN.
```

```
SQL-ERROR.
    EXEC SQL
        WHENEVER SQLERROR CONTINUE
    END-EXEC.
    MOVE ORASLNR TO ORASLNRD.
    DISPLAY " ".
    DISPLAY "ORACLE ERROR DETECTED ON LINE ", ORASLNRD, ":".
    DISPLAY " ".
    DISPLAY SQLERRMC.
    EXEC SQL
        ROLLBACK WORK RELEASE
    END-EXEC.
    STOP RUN.
```

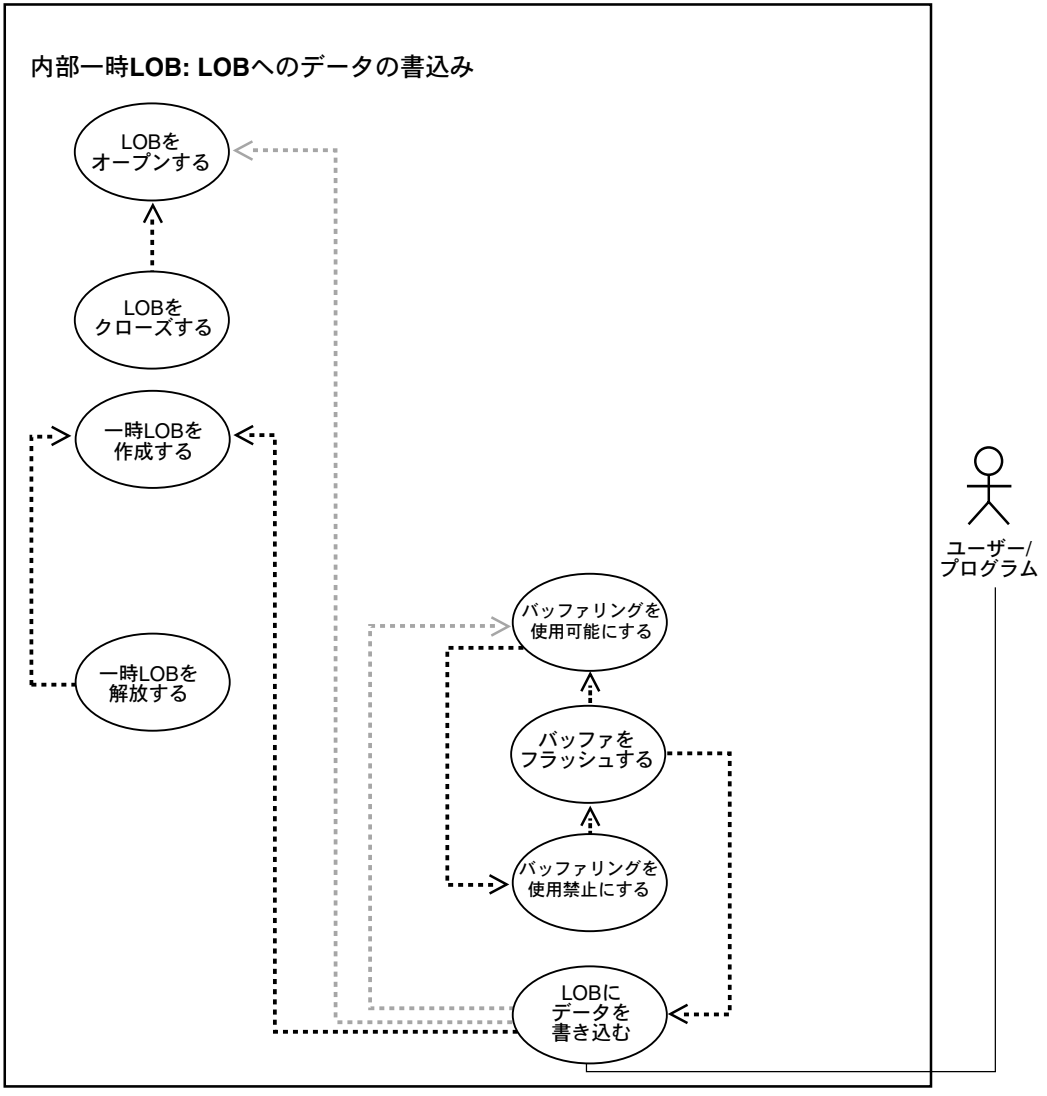
C/C++（Pro*C/C++）：一時 LOB への追加の書込み

```
/* Write-appending to a temporary LOB.  [Example script: 3903.pc]  */

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

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

#define BufferLength 256

void writeAppendTempLOB_proc()
{
  OCIBlobLocator *Temp_loc;
  OCIBFileLocator *Lob_loc;
  char *Dir = "ADPHOTO_DIR", *Name = "monitor_photo_3060_11001";
  int Amount;
  struct {
    unsigned short Length;
    char Data[BufferLength];
  } Buffer;
  EXEC SQL VAR Buffer IS VARRAW(BufferLength);
  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
```

```
  /* Allocate and Create the Temporary LOB: */
  EXEC SQL ALLOCATE :Temp_loc;
  EXEC SQL LOB CREATE TEMPORARY :Temp_loc;

  /* Allocate and Initialize the BFILE Locator: */
  EXEC SQL ALLOCATE :Lob_loc;
  EXEC SQL LOB FILE SET :Lob_loc DIRECTORY = :Dir, FILENAME = :Name;

  /* Opening the LOBs is Optional: */
  EXEC SQL LOB OPEN :Lob_loc READ ONLY;
  EXEC SQL LOB OPEN :Temp_loc READ WRITE;

  /* Load a specified amount from the BFILE into the Temporary LOB: */
  Amount = 2048;
  EXEC SQL LOB LOAD :Amount FROM FILE :Lob_loc INTO :Temp_loc;
  strcpy((char *)Buffer.Data, "afafafafafaf");
  Buffer.Length = 6;

  /* Write the contents of the Buffer to the end of the Temporary LOB: */
  Amount = Buffer.Length;
  EXEC SQL LOB WRITE APPEND :Amount FROM :Buffer INTO :Temp_loc;

  /* Closing the LOBs is Mandatory if they have been Opened: */
  EXEC SQL LOB CLOSE :Lob_loc;
  EXEC SQL LOB CLOSE :Temp_loc;

  /* Free the Temporary LOB */
  EXEC SQL LOB FREE TEMPORARY :Temp_loc;
  /* Release resources held by the Locators: */
  EXEC SQL FREE :Lob_loc;
  EXEC SQL FREE :Temp_loc;
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  writeAppendTempLOB_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

一時 LOB へのデータの書込み

図 11-22 利用図 : 一時 LOB へのデータの書込み



参照： 11-2 ページの表 11-1「利用モデル : 内部一時 LOB」を参照してください。

用途

データを一時 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には、ポーリングまたはコールバックによってストリーム・メカニズムを有効にした OCILobWrite() を使用します。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一時 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。

```
/* Writing data to a temporary LOB.  [Example script: 3905.sql]

declare
   blob_loc  BLOB;
   rawbuf RAW(10);
   an_offset INTEGER := 1;
   an_amount BINARY_INTEGER := 10;
begin
   select blob_col into blob_loc from a_table
where id = 1;
   rawbuf := '1234567890123456789';
   dbms_lob.write(blob_loc, an_amount, an_offset,
rawbuf);
   commit;
end;
```

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関数またはファンクションのリストは、第 3 章「様々なプログラム環境での LOB のサポート」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）：『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』の第 23 章「DBMS_LOB」の「DBMS_LOB サブプログラムの要約」の「WRITE プロシージャ」
- C（OCI）：『Oracle Call Interface プログラマーズ・ガイド』の第 16 章「その他の OCI リレーショナル関数」の「LOB 関数」の「OCILobWrite()」
- COBOL（Pro*COBOL）：『Pro*COBOL Precompiler プログラマーズ・ガイド』の LOB に関する詳細、LOB 文の使用上の注意および付録 F「埋込み SQL およびプリコンパイラ・ディレクティブ」の「LOB WRITE（実行可能埋込み SQL 拡張機能）」
- C/C++（Pro*C/C++）：『Pro*C/C++ Precompiler プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびディレクティブ」の「LOB WRITE（実行可能埋込み SQL 拡張機能）」
- Visual Basic（OO4O）：参照マニュアルはありません。
- Java（JDBC）：10-216 ページの「Java（JDBC）：LOB へのデータの書込み」

使用例

次のプロシージャの例では、LOB にデータを書き込むことによって `ad_sourcetext` データ（広告のテキスト）を更新できます。

例

次のプログラム環境での例を示します。

- PL/SQL（DBMS_LOB）：一時 LOB へのデータの書込み（11-167 ページ）
- C（OCI）：一時 LOB へのデータの書込み（11-167 ページ）
- COBOL（Pro*COBOL）：一時 LOB へのデータの書込み（11-170 ページ）

- C/C++（Pro*C/C++）：一時 LOB へのデータの書込み（11-172 ページ）
- Visual Basic（OO4O）：今回のリリースでは例は提供されません。
- Java（JDBC）：今回のリリースでは例は提供されません。

PL/SQL（DBMS_LOB）：一時 LOB へのデータの書込み

```
/* Writing data to a temporary LOB  [Example script: 3909.sql]
   Procedure writeDataToTempLOB_proc is not part of DBMS_LOB package. */

CREATE or REPLACE PROCEDURE writeDataToTempLOB_proc IS
   Lob_loc         CLOB;
   Buffer          VARCHAR2(26);
   Amount          BINARY_INTEGER := 26;
   Position        INTEGER := 1;
   i               INTEGER;
BEGIN
    DBMS_LOB.CREATETEMPORARY(Lob_loc,TRUE);
   /* Opening the LOB is optional: */
   DBMS_LOB.OPEN (Lob_loc, DBMS_LOB.LOB_READWRITE);
   /* Fill the buffer with data to write to the LOB: */
   Buffer := 'abcdefghijklmnopqrstuvwxyz';

   FOR i IN 1..3 LOOP
      DBMS_LOB.WRITE (Lob_loc, Amount, Position, Buffer);
      /* Fill the buffer with more data to write to the LOB: */
      Position := Position + Amount;
   END LOOP;
   /* Closing the LOB is mandatory if you have opened it: */
   DBMS_LOB.CLOSE (Lob_loc);
   DBMS_LOB.FREETEMPORARY(Lob_loc);
END;
```

C（OCI）：一時 LOB へのデータの書込み

```
/* Writing data to a temporary LOB.  [Example script: 3910] */
/* This example illustrates streaming writes with polling */

#define MAXBUFLEN 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[MAXBUFLEN];
sb4     err;

/* Allocate the locators descriptors: */
(void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &tclob ,
                          (ub4)OCI_DTYPE_LOB, (size_t) 0, (dvoid **) 0);

if(OCILobCreateTemporary(svchp,
         errhp,
         tclob,
         (ub2)0,
         SQLCS_IMPLICIT,
         OCI_TEMP_CLOB,
         OCI_ATTR_NOCACHE,
         OCI_DURATION_SESSION))
{
   (void) printf("FAILED: CreateTemporary() \n");
   return -1;
}

/* Open the CLOB: */
checkerr (errhp, (OCILobOpen(svchp, errhp, tclob, OCI_LOB_READWRITE)));

if (Total > MAXBUFLEN)
  nbytes = MAXBUFLEN;   /* We will use Streaming via Standard Polling */
else
  nbytes = Total;       /* Only a single WRITE is required */

/* Fill the Buffer with nbytes worth of Data: */
memset(bufp,'a',32767);

remainder = Total - nbytes;
amtp = 0;
offset = 1;
/* Setting Amount to 0 streams the data until use specifies OCI_LAST_PIECE: */

if (0 == remainder)
{
  amtp = nbytes;
  /* Here, (Total <= MAXBUFLEN ) so we can WRITE in ONE piece: */
  checkerr (errhp, OCILobWrite (svchp, errhp, tclob, &amtp,
                                offset, bufp, nbytes,
```

```
                                OCI_ONE_PIECE, (dvoid *)0,
                                (sb4 (*)(dvoid*,dvoid*,ub4*,ub1 *))0,
                                0, SQLCS_IMPLICIT));
}
else
{
    /* Here (Total > MAXBUFLEN ) so we use Streaming via Standard Polling: */
    /* WRITE the FIRST piece.  Specifying FIRST initiates Polling: */
    err = OCILobWrite (svchp, errhp, tclob, &amtp,
                       offset, bufp, nbytes,
                       OCI_FIRST_PIECE, (dvoid *)0,
                       (sb4 (*)(dvoid*,dvoid*,ub4*,ub1 *))0,
                       0, SQLCS_IMPLICIT);

    if (err != OCI_NEED_DATA)
      checkerr (errhp, err);

    last = FALSE;
    /* WRITE the NEXT (interim) and LAST pieces: */
    do
      {
        if (remainder > MAXBUFLEN)
          nbytes = MAXBUFLEN;            /* Still have more pieces to go */
        else
        {
          nbytes = remainder;        /* Here, (remainder <= MAXBUFLEN) */
          last = TRUE;             /* This is going to be the Final piece */
        }

        /* Fill the Buffer with nbytes worth of Data */

        if (last)
        {
          /* Specifying LAST terminates Polling */
          err = OCILobWrite (svchp, errhp, tclob, &amtp,
                             offset, bufp, nbytes,
                             OCI_LAST_PIECE, (dvoid *)0,
                             (sb4 (*)(dvoid*,dvoid*,ub4*,ub1 *))0,
                             0, SQLCS_IMPLICIT);

          if (err != 0)
            checkerr (errhp, err);

        } else

        {
          err =  OCILobWrite (svchp, errhp, tclob, &amtp,
```

```
                                    offset, bufp, nbytes,
                                    OCI_NEXT_PIECE, (dvoid *)0,
                                    (sb4 (*)(dvoid*,dvoid*,ub4*,ub1 *))0,
                                    0, SQLCS_IMPLICIT);

            if (err != OCI_NEED_DATA)
              checkerr (errhp, err);

          }
          /* Determine how much is left to WRITE: */
          remainder = remainder - nbytes;
        } while (!last);
    }
   /* At this point, (remainder == 0) */

 /* Closing the LOB is mandatory if you have opened it: */
    checkerr (errhp, OCILobClose(svchp, errhp, tclob));

  /* Free the temporary LOB: */
  checkerr(errhp,OCILobFreeTemporary(svchp,errhp,tclob));

  /* Free resources held by the locators: */
  (void) OCIDescriptorFree((dvoid *) tclob, (ub4) OCI_DTYPE_LOB);
}
```

COBOL（Pro*COBOL）: 一時 LOB へのデータの書込み

```
      * Writing data to a temporary LOB   [Example script: 3911.pco]
        IDENTIFICATION DIVISION.
        PROGRAM-ID. WRITE-TEMP.
        ENVIRONMENT DIVISION.
        DATA DIVISION.
        WORKING-STORAGE SECTION.

        01  USERID   PIC X(11) VALUES "SAMP/SAMP".
        01  TEMP-CLOB      SQL-CLOB.
        01  BUFFER         PIC X(20) VARYING.
        01  DIR-ALIAS      PIC X(30) VARYING.
        01  FNAME          PIC X(20) VARYING.
        01  DIR-IND        PIC S9(4) COMP.
        01  FNAME-IND      PIC S9(4) COMP.
        01  AMT            PIC S9(9) COMP VALUE 10.
        01  ORASLNRD        PIC 9(4).

            EXEC SQL INCLUDE SQLCA END-EXEC.
            EXEC ORACLE OPTION (ORACA=YES) END-EXEC.
```

```
    EXEC SQL INCLUDE ORACA END-EXEC.

 PROCEDURE DIVISION.
 WRITE-TEMP.

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

* Allocate and initialize the BFILE and BLOB locators:
    EXEC SQL ALLOCATE :TEMP-CLOB END-EXEC.
    EXEC SQL
         LOB CREATE TEMPORARY :TEMP-CLOB
    END-EXEC.
    EXEC SQL LOB OPEN :TEMP-CLOB READ WRITE END-EXEC.

  MOVE "ABCDE12345ABCDE12345" TO BUFFER-ARR.
  MOVE 20 TO BUFFER-LEN.
  MOVE 20 TO AMT.
* Append the data in BUFFER to TEMP-CLOB:
    EXEC SQL LOB WRITE :AMT FROM :BUFFER INTO :TEMP-CLOB END-EXEC.

* Close the LOBs:
    EXEC SQL LOB CLOSE :TEMP-CLOB END-EXEC.

* Free the temporary LOB:
    EXEC SQL LOB FREE TEMPORARY :TEMP-CLOB END-EXEC.

* And free the LOB locators:
    EXEC SQL FREE :TEMP-CLOB END-EXEC.
    STOP RUN.

 SQL-ERROR.
    EXEC SQL WHENEVER SQLERROR CONTINUE END-EXEC.
    MOVE ORASLNR TO ORASLNRD.
    DISPLAY " ".
    DISPLAY "ORACLE ERROR DETECTED ON LINE ", ORASLNRD, ":".
    DISPLAY " ".
    DISPLAY SQLERRMC.
    EXEC SQL ROLLBACK WORK RELEASE END-EXEC.
    STOP RUN.
```

C/C++（Pro*C/C++）：一時 LOB へのデータの書込み

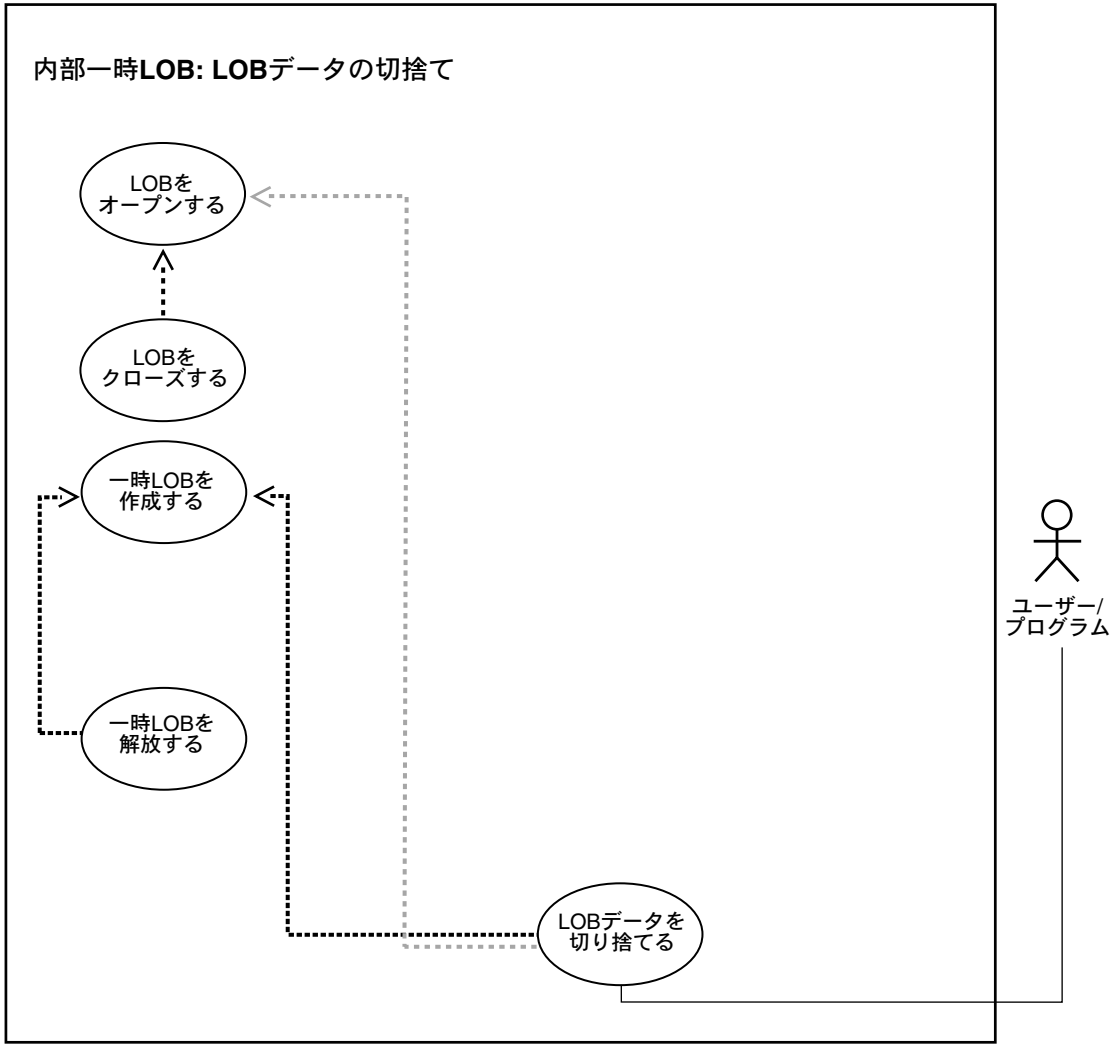
```
/* Writing data to a temporary LOB. [Example script: 3912] */

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

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

#define BufferLength 1024

void writeDataToTempLOB_proc(multiple) int multiple;
{
  OCIClobLocator *Temp_loc;
  varchar Buffer[BufferLength];
  unsigned int Total;
  unsigned int Amount;
  unsigned int remainder, nbytes;
  boolean last;

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  /* Allocate and Initialize the Temporary LOB: */
  EXEC SQL ALLOCATE :Temp_loc;
  EXEC SQL LOB CREATE TEMPORARY :Temp_loc;
  /* Open the Temporary LOB: */
  EXEC SQL LOB OPEN :Temp_loc READ WRITE;
  Total = Amount = (multiple * BufferLength);
  if (Total > BufferLength)
    nbytes = BufferLength;   /* We will use Streaming via Standard Polling */
  else
    nbytes = Total;          /* Only a single WRITE is required */
  /* Fill the Buffer with nbytes worth of Data: */
  memset((void *)Buffer.arr, 32, nbytes);
  Buffer.len = nbytes;       /* Set the Length */
  remainder = Total - nbytes;
  if (0 == remainder)
    {
      /* Here, (Total <= BufferLength) so we can WRITE in ONE piece: */
      EXEC SQL LOB WRITE ONE :Amount FROM :Buffer INTO :Temp_loc;
```

```
    printf("Write ONE Total of %d characters\n", Amount);
  }
else
  {
    /* Here (Total > BufferLength) so use Streaming via Standard Polling */
    /* WRITE the FIRST piece.  Specifying FIRST initiates Polling: */
    EXEC SQL LOB WRITE FIRST :Amount FROM :Buffer INTO :Temp_loc;
    printf("Write FIRST %d characters\n", Buffer.len);
    last = FALSE;
    /* WRITE the NEXT (interim) and LAST pieces: */
    do
      {
        if (remainder > BufferLength)
          nbytes = BufferLength;         /* Still have more pieces to go */
        else
          {
            nbytes = remainder;     /* Here, (remainder <= BufferLength) */
            last = TRUE;          /* This is going to be the Final piece */
          }
        /* Fill the Buffer with nbytes worth of Data: */
        memset((void *)Buffer.arr, 32, nbytes);
        Buffer.len = nbytes;       /* Set the Length */
        if (last)
          {
            EXEC SQL WHENEVER SQLERROR DO Sample_Error();
            /* Specifying LAST terminates Polling: */
            EXEC SQL LOB WRITE LAST :Amount FROM :Buffer INTO :Temp_loc;
            printf("Write LAST Total of %d characters\n", Amount);
          }
        else
          {
            EXEC SQL WHENEVER SQLERROR DO break;
            EXEC SQL LOB WRITE NEXT :Amount FROM :Buffer INTO :Temp_loc;
            printf("Write NEXT %d characters\n", Buffer.len);
          }
        /* Determine how much is left to WRITE: */
        remainder = remainder - nbytes;
      } while (!last);
  }
EXEC SQL WHENEVER SQLERROR DO Sample_Error();
/* At this point, (Amount == Total), the total amount that was written. */
/* Close the Temporary LOB: */
EXEC SQL LOB CLOSE :Temp_loc;
/* Free the Temporary LOB: */
EXEC SQL LOB FREE TEMPORARY :Temp_loc;
/* Free resources held by the Locator: */
EXEC SQL FREE :Temp_loc;
```

```
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  writeDataToTempLOB_proc(1);                          /* Write One Piece */
  writeDataToTempLOB_proc(4);     /* Write Multiple Pieces using Polling */
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

一時 LOB データの切捨て

図 11-23 利用図 : 一時 LOB データの切捨て



参照： 11-2 ページの表 11-1「利用モデル : 内部一時 LOB」を参照してください。

用途

一時 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関数またはファンクションのリストは、第 3 章「様々なプログラム環境での LOB のサポート」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）：『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』の第 23 章「DBMS_LOB」の「DBMS_LOB サブプログラムの要約」の「TRIM プロシージャ」
- C（OCI）：『Oracle Call Interface プログラマーズ・ガイド』の第 16 章「その他の OCI リレーショナル関数」の「LOB 関数」の「OCILobTrim()」
- COBOL（Pro*COBOL）：『Pro*COBOL Precompiler プログラマーズ・ガイド』の LOB に関する詳細、LOB 文の使用上の注意および付録 F「埋込み SQL およびプリコンパイラ・ディレクティブ」の「LOB TRIM（実行可能埋込み SQL 拡張機能）」
- C/C++（Pro*C/C++）：『Pro*C/C++ Precompiler プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびディレクティブ」の「LOB TRIM（実行可能埋込み SQL 拡張機能）」
- Visual Basic（OO4O）: 参照マニュアルはありません。
- Java（JDBC）: 10-225 ページの「Java（JDBC）: LOB データの切捨て」

使用例

次の例では、`adheader_tab` 表の `ad_finaltext` 列で参照されるテキスト（CLOB データ）にアクセスして切り捨てます。

例

次のプログラム環境での例を示します。

- PL/SQL（DBMS_LOB）：一時 LOB データの切捨て（11-177 ページ）
- C（OCI）：一時 LOB データの切捨て（11-178 ページ）
- COBOL（Pro*COBOL）：一時 LOB データの切捨て（11-180 ページ）
- C/C++（Pro*C/C++）：一時 LOB データの切捨て（11-182 ページ）
- Visual Basic（OO4O）：今回のリリースでは例は提供されません。
- Java（JDBC）：今回のリリースでは例は提供されません。

PL/SQL（DBMS_LOB）：一時 LOB データの切捨て

```
/* Trimming temporary LOB data.  [Example script: 3914.sql]
   Procedure trimTempLOB_proc is not part of DBMS_LOB package. */

CREATE OR REPLACE PROCEDURE trimTempLOB_proc IS
   Lob_loc         CLOB;
   Amount          number;
   Src_loc         BFILE := BFILENAME('ADPHOTO_DIR', 'monitor_photo_3060_11001');
   TrimAmount      number := 100;
BEGIN
   /* Create a temporary LOB: */
   DBMS_LOB.CREATETEMPORARY(Lob_loc,TRUE);
   /* Opening the LOB is optional: */
   DBMS_LOB.OPEN (Lob_loc, DBMS_LOB.LOB_READWRITE);
   /* Opening the file is mandatory: */
   DBMS_LOB.OPEN(Src_loc, DBMS_LOB.LOB_READONLY);
   /* Populate the temporary LOB with some data: */
   Amount := 32767;
   DBMS_LOB.LOADFROMFILE(Lob_loc, Src_loc, Amount);
   DBMS_LOB.TRIM(Lob_loc,TrimAmount);
   /* Closing the LOB is mandatory if you have opened it: */
   DBMS_LOB.CLOSE (Lob_loc);
   DBMS_LOB.CLOSE(Src_loc);
   DBMS_LOB.FREETEMPORARY(Lob_loc);
COMMIT;
EXCEPTION
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('Operation failed');
END;
```

C（OCI）：一時 LOB データの切捨て

```
/* Trimming temporary LOB data.  [Example script: 3915.c]

sb4 trim_temp_lobs (  OCIError      *errhp,
                      OCISvcCtx     *svchp,
                      OCIStmt       *stmthp,
                      OCIEnv        *envhp)
{
  OCILobLocator *tblob;
  OCILobLocator *bfile;
  ub4 amt = 4000;
  ub4 trim_size = 2;
  sb4 return_code = 0;

  printf("in trim\n");
  if(OCIDescriptorAlloc((dvoid *)envhp, (dvoid **) &tblob,
                        (ub4) OCI_DTYPE_LOB,
                        (size_t) 0, (dvoid **) 0))
  {
     printf("OCIDescriptor Alloc FAILED in trim\n");
     return -1;
  }

  if(OCIDescriptorAlloc((dvoid *)envhp, (dvoid **) &bfile,
                        (ub4) OCI_DTYPE_FILE,
                        (size_t) 0, (dvoid **) 0))
  {
     printf("OCIDescriptor Alloc FAILED in trim\n");
     return -1;
  }

  /* Set the BFILE to point to the monitor_photo file: */
  if(OCILobFileSetName(envhp, errhp, &bfile, (text *)"ADPHOTO_DIR",
                       (ub2)strlen("ADPHOTO_DIR"),
                       (text *)"monitor_photo_3060_11001",
                       (ub2)strlen("monitor_photo_3060_11001")))
  {
    printf("OCILobFileSetName FAILED\n");
    return -1;
  }

  if (OCILobFileOpen(svchp, errhp, (OCILobLocator *) bfile, OCI_LOB_READONLY))
  {
    printf( "OCILobFileOpen FAILED  for the bfile\n");
    return_code = -1;
  }
```

```
if(OCILobCreateTemporary(svchp,errhp,tblob,(ub2)0, SQLCS_IMPLICIT,
                         OCI_TEMP_BLOB, OCI_ATTR_NOCACHE,
                         OCI_DURATION_SESSION))
{
   (void) printf("FAILED: CreateTemporary() \n");
   return -1;
}

if (OCILobOpen(svchp, errhp, (OCILobLocator *) tblob, OCI_LOB_READWRITE))
{
  printf( "OCILobOpen FAILED for temp LOB \n");
  return_code =  -1;
}

/* populate the temp LOB with 4000 bytes of data */
if(OCILobLoadFromFile(svchp, errhp, tblob, (OCILobLocator*)bfile,
                      (ub4)amt,(ub4)1,(ub4)1))
{
  printf( "OCILobLoadFromFile FAILED\n");
  return_code = -1;
}

if (OCILobTrim(svchp, errhp, (OCILobLocator *) tblob, trim_size))
{
    printf( "OCILobTrim FAILED for temp LOB \n");
    return_code = -1;
} else
{
    printf( "OCILobTrim succeeded for temp LOB \n");
}

if (OCILobClose(svchp, errhp, (OCILobLocator *) bfile))
{
  printf( "OCILobClose FAILED for bfile \n");
  return_code =  -1;
}

if (OCILobClose(svchp, errhp, (OCILobLocator *) tblob))
{
  printf( "OCILobClose FAILED for temporary LOB \n");
  return_code =  -1;
}
/* Free the temporary LOB now that we are done using it: */
if(OCILobFreeTemporary(svchp, errhp, tblob))
{
  printf("OCILobFreeTemporary FAILED \n");
```

```
  return_code = -1;
}
return return_code;
}
```

COBOL（Pro*COBOL）: 一時 LOB データの切捨て

```
* Trimming temporary LOB data.  [Example script: 3916.pco]
  IDENTIFICATION DIVISION.
  PROGRAM-ID. TEMP-LOB-TRIM.
  ENVIRONMENT DIVISION.
  DATA DIVISION.
  WORKING-STORAGE SECTION.

  01  USERID   PIC X(11) VALUES "SAMP/SAMP".
  01  TEMP-BLOB      SQL-BLOB.
  01  SRC-BFILE      SQL-BFILE.
  01  DIR-ALIAS      PIC X(30) VARYING.
  01  FNAME          PIC X(20) VARYING.
  01  DIR-IND        PIC S9(4) COMP.
  01  FNAME-IND      PIC S9(4) COMP.
  01  AMT            PIC S9(9) COMP VALUE 10.
  01  ORASLNRD        PIC 9(4).

      EXEC SQL INCLUDE SQLCA END-EXEC.
      EXEC ORACLE OPTION (ORACA=YES) END-EXEC.
      EXEC SQL INCLUDE ORACA END-EXEC.

  PROCEDURE DIVISION.
  TEMP-LOB-TRIM.
      EXEC SQL WHENEVER SQLERROR DO PERFORM SQL-ERROR END-EXEC.
      EXEC SQL
           CONNECT :USERID
      END-EXEC.

 * Allocate and initialize the BFILE and BLOB locators:
      EXEC SQL ALLOCATE :TEMP-BLOB END-EXEC.
      EXEC SQL ALLOCATE :SRC-BFILE END-EXEC.
      EXEC SQL
           LOB CREATE TEMPORARY :TEMP-BLOB
      END-EXEC.

 * Set up the directory and file information:
      MOVE "ADPHOTO_DIR" TO DIR-ALIAS-ARR.
      MOVE 9 TO DIR-ALIAS-LEN.
      MOVE "keyboard_photo_3106_13001" TO FNAME-ARR.
```

```
     MOVE 16 TO FNAME-LEN.

     EXEC SQL
        LOB FILE SET :SRC-BFILE DIRECTORY = :DIR-ALIAS,
        FILENAME = :FNAME
     END-EXEC.

* Open source BFILE and destination temporary BLOB:
     EXEC SQL LOB OPEN :TEMP-BLOB READ WRITE END-EXEC.
     EXEC SQL LOB OPEN :SRC-BFILE READ ONLY END-EXEC.
     EXEC SQL
          LOB LOAD :AMT FROM FILE :SRC-BFILE INTO :TEMP-BLOB
     END-EXEC.

* Trim the last half of the data:
     MOVE 5 TO AMT.
     EXEC SQL LOB TRIM :TEMP-BLOB TO :AMT END-EXEC.

* Close the LOBs:
     EXEC SQL LOB CLOSE :SRC-BFILE END-EXEC.
     EXEC SQL LOB CLOSE :TEMP-BLOB END-EXEC.

* Free the temporary LOB:
     EXEC SQL LOB FREE TEMPORARY :TEMP-BLOB END-EXEC.

* And free the LOB locators:
     EXEC SQL FREE :TEMP-BLOB END-EXEC.
     EXEC SQL FREE :SRC-BFILE END-EXEC.
     STOP RUN.

 SQL-ERROR.
     EXEC SQL WHENEVER SQLERROR CONTINUE END-EXEC.
     MOVE ORASLNR TO ORASLNRD.
     DISPLAY " ".
     DISPLAY "ORACLE ERROR DETECTED ON LINE ", ORASLNRD, ":".
     DISPLAY " ".
     DISPLAY SQLERRMC.
     EXEC SQL ROLLBACK WORK RELEASE END-EXEC.
     STOP RUN.
```

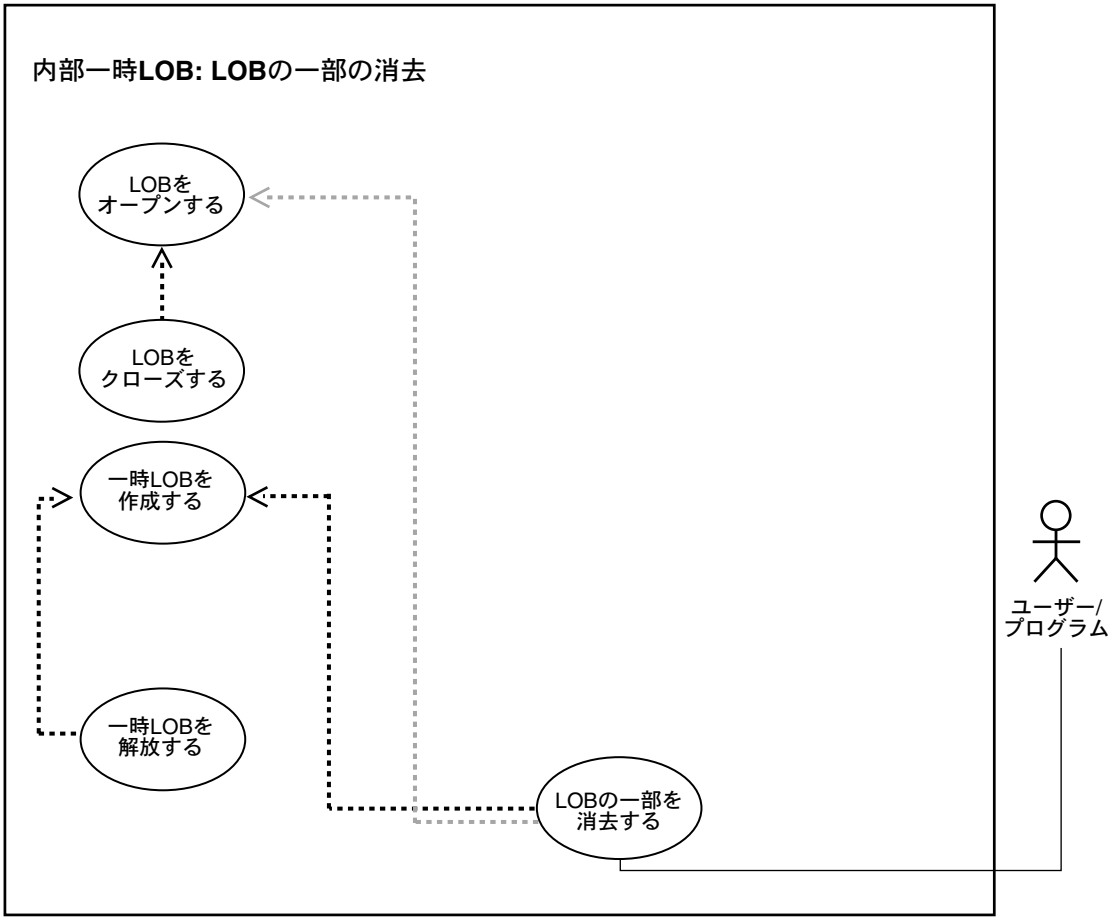
C/C++（Pro*C/C++）：一時 LOB データの切捨て

```
/* Trimming temporary LOB data.  [Example script:  3917.pc]  */

void trimTempLOB_proc()
#include <oci.h>
#include <stdio.h>
#include <sqlca.h>

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

void trimTempLOB_proc()
{
  OCIBlobLocator *Temp_loc;
  OCIBFileLocator *Lob_loc;
  char *Dir = "ADPHOTO_DIR", *Name = "monitor_photo_3060_11001";
  int Amount = 4096;
  int trimLength;

  /* Allocate and Create the Temporary LOB: */
  EXEC SQL ALLOCATE :Temp_loc;
  EXEC SQL LOB CREATE TEMPORARY :Temp_loc;

  /* Allocate and Initialize the BFILE Locator: */
  EXEC SQL ALLOCATE :Lob_loc;
  EXEC SQL LOB FILE SET :Lob_loc DIRECTORY = :Dir, FILENAME = :Name;

  /* Opening the LOBs is Optional: */
  EXEC SQL LOB OPEN :Lob_loc READ ONLY;
  EXEC SQL LOB OPEN :Temp_loc READ WRITE;

  /* Load the specified amount from the BFILE into the Temporary LOB: */
  EXEC SQL LOB LOAD :Amount FROM FILE :Lob_loc INTO :Temp_loc;

  /* Set the new length of the Temporary LOB: */
  trimLength = (int) (Amount / 2);

  /* Trim the Temporary LOB to its new length: */
  EXEC SQL LOB TRIM :Temp_loc TO :trimLength;

  /* Closing the LOBs is Mandatory if they have been Opened: */
```

```
  EXEC SQL LOB CLOSE :Lob_loc;
  EXEC SQL LOB CLOSE :Temp_loc;

  /* Free the Temporary LOB: */
  EXEC SQL LOB FREE TEMPORARY :Temp_loc;

  /* Release resources held by the Locators: */
  EXEC SQL FREE :Lob_loc;
  EXEC SQL FREE :Temp_loc;
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  trimTempLOB_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

一時 LOB の一部の消去

図 11-24 利用図 : 一時 LOB の一部の消去



参照 : 11-2 ページの表 11-1「利用モデル : 内部一時 LOB」を参照してください。

用途

一時 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関数またはファンクションのリストは、第 3 章「様々なプログラム環境での LOB のサポート」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）：『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』の第 23 章「DBMS_LOB」の「DBMS_LOB サブプログラムの要約」の「ERASE プロシージャ」
- C（OCI）：『Oracle Call Interface プログラマーズ・ガイド』の第 16 章「その他の OCI リレーショナル関数」の「LOB 関数」の「OCILobErase()」
- COBOL（Pro*COBOL）：『Pro*COBOL Precompiler プログラマーズ・ガイド』の LOB に関する詳細、LOB 文の使用上の注意および付録 F「埋込み SQL およびプリコンパイラ・ディレクティブ」の「LOB ERASE（実行可能埋込み SQL 拡張機能）」
- C/C++（Pro*C/C++）：『Pro*C/C++ Precompiler プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびディレクティブ」の「LOB ERASE（実行可能埋込み SQL 拡張機能）」
- Visual Basic（OO4O）：参照マニュアルはありません。
- Java（JDBC）：10-237 ページの「Java（JDBC）：LOB の一部の消去」

使用例

ありません。

例

次のプログラム環境での例を示します。

- PL/SQL（DBMS_LOB）：一時 LOB の一部の消去（11-186 ページ）
- C（OCI）：一時 LOB の一部の消去（11-187 ページ）
- COBOL（Pro*COBOL）：一時 LOB の一部の消去（11-189 ページ）
- C/C++（Pro*C/C++）：一時 LOB の一部の消去（11-191 ページ）
- Visual Basic（OO4O）：今回のリリースでは例は提供されません。
- Java（JDBC）：今回のリリースでは例は提供されません。

PL/SQL（DBMS_LOB）：一時 LOB の一部の消去

```
/* Erasing part of a temporary LOB.  [Example script: 3918.sql]
   Procedure eraseTempLOB_proc is not part of DBMS_LOB package: */

CREATE OR REPLACE PROCEDURE trimTempLOB_proc IS
   Lob_loc        CLOB;
   amt            number;
   Src_loc        BFILE := BFILENAME('ADPHOTO_DIR', 'monitor_photo_3060_11001');
   Amount         INTEGER := 32767;
BEGIN

   /* Create a temporary LOB: */
   DBMS_LOB.CREATETEMPORARY(Lob_loc,TRUE);

   /* Opening the LOB is optional: */
   DBMS_LOB.OPEN (Lob_loc, DBMS_LOB.LOB_READWRITE);

   /* Populate the temporary LOB with some data: */
   Amount := 32767;
   DBMS_LOB.LOADFROMFILE(Lob_loc, Src_loc, Amount);
   /* Erase  the LOB data: */
   amt := 3000;
   DBMS_LOB.ERASE(Lob_loc, amt, 2);

    /* Closing the LOB is mandatory if you have opened it: */
   DBMS_LOB.CLOSE (Lob_loc);
   DBMS_LOB.CLOSE(Src_loc);
   DBMS_LOB.FREETEMPORARY(Lob_loc);
COMMIT;
EXCEPTION
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('Operation failed');
END;
```

C（OCI）：一時 LOB の一部の消去

```
/* Erasing part of a temporary LOB.  [Example script: 3919.c]
   This example erases 2 bytes at offset 100 in a temporary LOB: */

sb4 erase_temp_lobs ( OCIError      *errhp,
                      OCISvcCtx     *svchp,
                      OCIStmt       *stmthp,
                      OCIEnv        *envhp)
{
  OCILobLocator *tblob;
  OCILobLocator *bfile;
  ub4 amt = 4000;
  ub4 erase_size = 2;
  ub4 erase_offset = 100;
  sb4 return_code = 0;

  printf("in erase\n");
  if(OCIDescriptorAlloc((dvoid *)envhp, (dvoid **) &tblob,
                        (ub4) OCI_DTYPE_LOB,
                        (size_t) 0, (dvoid **) 0))
  {
     printf("OCIDescriptor Alloc FAILED \n");
     return -1;
  }

  if(OCIDescriptorAlloc((dvoid *)envhp, (dvoid **) &bfile,
                        (ub4) OCI_DTYPE_FILE,
                        (size_t) 0, (dvoid **) 0))
  {
     printf("OCIDescriptor Alloc FAILED \n");
     return -1;
  }

  /* Set the BFILE to point to the monitor_photo_3060_11001 file: */
  if(OCILobFileSetName(envhp, errhp, &bfile,
                       (text *)"ADPHOTO_DIR",
                       (ub2)strlen("ADPHOTO_DIR"),
                       (text *)"monitor_photo_3060_11001",
                       (ub2)strlen("monitor_photo_3060_11001")))
  {
    printf("OCILobFileSetName FAILED\n");
    return -1;
  }
```

```
if (OCILobFileOpen(svchp, errhp, (OCILobLocator *) bfile, OCI_LOB_READONLY))
{
  printf( "OCILobFileOpen FAILED  for the bfile\n");
  return_code = -1;
}

if(OCILobCreateTemporary(svchp,errhp,tblob,(ub2)0, SQLCS_IMPLICIT,
    OCI_TEMP_BLOB, OCI_ATTR_NOCACHE, OCI_DURATION_SESSION))
{
   (void) printf("FAILED: CreateTemporary() \n");
   return -1;
}

if (OCILobOpen(svchp, errhp, (OCILobLocator *) tblob, OCI_LOB_READWRITE))
{
  printf( "OCILobOpen FAILED for temp LOB \n");
  return_code =  -1;
}

/* Populate the temp LOB with 4000 bytes of data: */
if(OCILobLoadFromFile(svchp,
                      errhp,
                      tblob,
                      (OCILobLocator*)bfile,
                      (ub4)amt,
                      (ub4)1,(ub4)1))
{
  printf( "OCILobLoadFromFile FAILED\n");
  return_code = -1;

}

if (OCILobErase(svchp, errhp, (OCILobLocator *) tblob, &erase_size,
               erase_offset))
{
    printf( "OCILobErase FAILED for temp LOB \n");
    return_code = -1;
} else
{
    printf( "OCILobErase succeeded for temp LOB \n");
}
```

```
if (OCILobClose(svchp, errhp, (OCILobLocator *) bfile))
{
  printf( "OCILobClose FAILED for bfile \n");
  return_code =  -1;
}

if (OCILobClose(svchp, errhp, (OCILobLocator *) tblob))
{
  printf( "OCILobClose FAILED for temporary LOB \n");
  return_code =  -1;
}
/* free the temporary LOB now that we are done using it */
if(OCILobFreeTemporary(svchp, errhp, tblob))
{
  printf("OCILobFreeTemporary FAILED \n");
  return_code = -1;
}
return return_code;
}
```

COBOL（Pro*COBOL）：一時 LOB の一部の消去

```
* Erasing part of a temporary LOB.  [Example script: 3920.pco]
  IDENTIFICATION DIVISION.
  PROGRAM-ID. TEMP-BLOB-ERASE.
  ENVIRONMENT DIVISION.
  DATA DIVISION.
  WORKING-STORAGE SECTION.

  01  USERID   PIC X(11) VALUES "SAMP/SAMP".
  01  TEMP-BLOB      SQL-BLOB.
  01  SRC-BFILE      SQL-BFILE.
  01  DIR-ALIAS      PIC X(30) VARYING.
  01  FNAME          PIC X(20) VARYING.
  01  DIR-IND        PIC S9(4) COMP.
  01  FNAME-IND      PIC S9(4) COMP.
  01  AMT            PIC S9(9) COMP VALUE 10.
  01  POS            PIC S9(9) COMP VALUE 1.
  01  ORASLNRD        PIC 9(4).

      EXEC SQL INCLUDE SQLCA END-EXEC.
      EXEC ORACLE OPTION (ORACA=YES) END-EXEC.
      EXEC SQL INCLUDE ORACA END-EXEC.
```

```
 PROCEDURE DIVISION.
 TEMP-BLOB-ERASE.
     EXEC SQL WHENEVER SQLERROR DO PERFORM SQL-ERROR END-EXEC.
     EXEC SQL
          CONNECT :USERID
     END-EXEC.

* Allocate and initialize the BLOB locator:
     EXEC SQL ALLOCATE :TEMP-BLOB END-EXEC.
     EXEC SQL ALLOCATE :SRC-BFILE END-EXEC.
     EXEC SQL LOB CREATE TEMPORARY :TEMP-BLOB END-EXEC.

* Set up the directory and file information:
     MOVE "ADPHOTO_DIR" TO DIR-ALIAS-ARR.
     MOVE 9 TO DIR-ALIAS-LEN.
     MOVE "keyboard_photo_3106_13001" TO FNAME-ARR.
     MOVE 16 TO FNAME-LEN.

     EXEC SQL
        LOB FILE SET :SRC-BFILE DIRECTORY = :DIR-ALIAS,
        FILENAME = :FNAME
     END-EXEC.

* Open source BFILE and destination temporary BLOB:
     EXEC SQL LOB OPEN :TEMP-BLOB READ WRITE END-EXEC.
     EXEC SQL LOB OPEN :SRC-BFILE READ ONLY END-EXEC.
     EXEC SQL
          LOB LOAD :AMT FROM FILE :SRC-BFILE INTO :TEMP-BLOB
     END-EXEC.

* Erase some of the LOB data:
     EXEC SQL
          LOB ERASE :AMT FROM :TEMP-BLOB AT :POS
     END-EXEC.

* Close the LOBs
     EXEC SQL LOB CLOSE :SRC-BFILE END-EXEC.
     EXEC SQL LOB CLOSE :TEMP-BLOB END-EXEC.
* Free the temporary LOB:
     EXEC SQL
          LOB FREE TEMPORARY :TEMP-BLOB
     END-EXEC.
```

```
* And free the LOB locators:
     EXEC SQL FREE :TEMP-BLOB END-EXEC.
     EXEC SQL FREE :SRC-BFILE END-EXEC.
     STOP RUN.

 SQL-ERROR.
     EXEC SQL
         WHENEVER SQLERROR CONTINUE
     END-EXEC.
     MOVE ORASLNR TO ORASLNRD.
     DISPLAY " ".
     DISPLAY "ORACLE ERROR DETECTED ON LINE ", ORASLNRD, ":".
     DISPLAY " ".
     DISPLAY SQLERRMC.
     EXEC SQL ROLLBACK WORK RELEASE END-EXEC.
     STOP RUN.
```

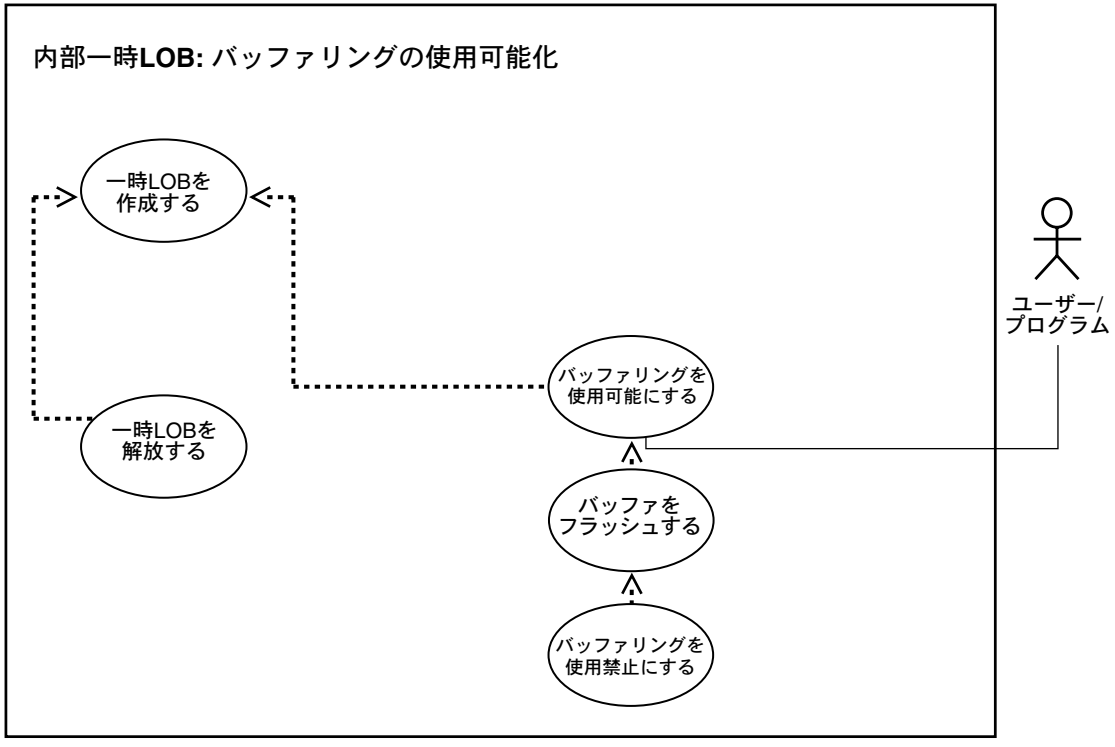
C/C++（Pro*C/C++）：一時 LOB の一部の消去

```
/* Erasing part of a temporary LOB.  [Example script: 3921.pc] */

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

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

void eraseTempLOB_proc()
{
  OCIBlobLocator *Temp_loc;
  OCIBFileLocator *Lob_loc;
  char *Dir = "ADPHOTO_DIR", *Name = "monitor_photo_3060_11001";
  int Amount;
  int Position = 1024;

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
```

```
  /* Allocate and Create the Temporary LOB: */
  EXEC SQL ALLOCATE :Temp_loc;
  EXEC SQL LOB CREATE TEMPORARY :Temp_loc;

  /* Allocate and Initialize the BFILE Locator: */
  EXEC SQL ALLOCATE :Lob_loc;
  EXEC SQL LOB FILE SET :Lob_loc DIRECTORY = :Dir, FILENAME = :Name;

  /* Opening the LOBs is Optional: */
  EXEC SQL LOB OPEN :Lob_loc READ ONLY;
  EXEC SQL LOB OPEN :Temp_loc READ WRITE;

  /* Load a specified amount from the BFILE into the Temporary LOB: */
  Amount = 4096;
  EXEC SQL LOB LOAD :Amount FROM FILE :Lob_loc INTO :Temp_loc;

  /* Erase a specified amount from the Temporary LOB at a given position: */
  Amount = 2048;
  EXEC SQL LOB ERASE :Amount FROM :Temp_loc AT :Position;

  /* Closing the LOBs is Mandatory if they have been Opened: */
  EXEC SQL LOB CLOSE :Lob_loc;
  EXEC SQL LOB CLOSE :Temp_loc;

  /* Free the Temporary LOB: */
  EXEC SQL LOB FREE TEMPORARY :Temp_loc;

  /* Release resources held by the Locators: */
  EXEC SQL FREE :Lob_loc;
  EXEC SQL FREE :Temp_loc;
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  eraseTempLOB_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

一時 LOB に対する LOB バッファリングの使用可能化

図 11-25 利用図 : 一時 LOB に対する LOB バッファリングの使用可能化



参照 : 11-2 ページの表 11-1「利用モデル : 内部一時 LOB」を参照してください。

用途

一時 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

少量のデータの読込みおよび書込みを行う場合は、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込みおよびストリーム書込みを行う場合は、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関数またはファンクションのリストは、第 3 章「様々なプログラム環境での LOB のサポート」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）:『Oracle Call Interface プログラマーズ・ガイド』の第 16 章「その他の OCI リレーショナル関数」の「LOB 関数」の「OCILobEnableBuffering()」、「OCILobDisbleBuffering()」および「OCILobFlushBuffer()」
- COBOL（Pro*COBOL）:『Pro*COBOL Precompiler プログラマーズ・ガイド』の LOB に関する詳細、LOB 文の使用上の注意および付録 F「埋込み SQL およびプリコンパイラ・ディレクティブ」の「LOB ENABLE BUFFERING（実行可能埋込み SQL 拡張機能）」
- C/C++（Pro*C/C++）:『Pro*C/C++ Precompiler プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびディレクティブ」の「LOB ENABLE BUFFERING（実行可能埋込み SQL 拡張機能）」
- Visual Basic（OO4O）: 参照マニュアルはありません。
- Java（JDBC）: 10-240 ページの「LOB バッファリングの使用可能化」

使用例

ありません。

例

次のプログラム環境での例を示します。

- PL/SQL（DBMS_LOB）: 今回のリリースでは例は提供されません。
- C（OCI）: 一時 LOB に対する LOB バッファリングの使用可能化（11-195 ページ）
- COBOL（Pro*COBOL）: 一時 LOB に対する LOB バッファリングの使用可能化（11-197 ページ）
- C/C++（Pro*C/C++）: 一時 LOB に対する LOB バッファリングの使用可能化（11-198 ページ）

- Visual Basic（OO4O）: 今回のリリースでは例は提供されません。
- Java（JDBC）: 今回のリリースでは例は提供されません。

C（OCI）: 一時 LOB に対する LOB バッファリングの使用可能化

```
/* Enabling LOB buffering for a temporary LOB.  [Example script: 3922.c] */

#define MAXBUFLEN 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[MAXBUFLEN];
  ub4 buflen;

  /* Allocate the descriptor for the lob locator: */
  (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &tblob,
                        (ub4)OCI_DTYPE_LOB, (size_t) 0, (dvoid **) 0);

  /* Select the BLOB: */
  printf (" create a temporary Lob\n");
  /* Create a temporary LOB: */
  if(OCILobCreateTemporary(svchp, errhp, tblob, (ub2)0, SQLCS_IMPLICIT,
                           OCI_TEMP_BLOB,
                           OCI_ATTR_NOCACHE,
                           OCI_DURATION_SESSION))
  {
    (void) printf("FAILED: CreateTemporary() \n");
    return -1;
  }

  /* Open the BLOB: */
  if (OCILobOpen(svchp, errhp, (OCILobLocator *) tblob, OCI_LOB_READWRITE))
  {
    printf( "OCILobOpen FAILED for temp LOB \n");
    return -1;
  }

  /* Enable LOB Buffering: */
  printf (" enable LOB buffering\n");
```

```
  checkerr (errhp, OCILobEnableBuffering(svchp, errhp, tblob));

  printf (" write data to LOB\n");

  /* Write data into the LOB: */
  amt    = sizeof(bufp);
  buflen = sizeof(bufp);
  offset = 1;
  checkerr (errhp, OCILobWrite (svchp, errhp, tblob, &amt,
                                offset, bufp, buflen,
                                OCI_ONE_PIECE, (dvoid *)0,
                                (sb4 (*)(dvoid*,dvoid*,ub4*,ub1 *))0,
                                0, SQLCS_IMPLICIT));

  /* Flush the buffer: */
  printf(" flush the LOB buffers\n");
  checkerr (errhp, OCILobFlushBuffer(svchp, errhp, tblob,
                                     (ub4)OCI_LOB_BUFFER_FREE));

  /* Disable Buffering: */
  printf (" disable LOB buffering\n");
  checkerr (errhp, OCILobDisableBuffering(svchp, errhp, tblob));

  /* Subsequent LOB WRITEs will not use the LOB Buffering Subsystem */

  /* Closing the BLOB is mandatory if you have opened it: */
  checkerr (errhp, OCILobClose(svchp, errhp, tblob));

  /* Free the temporary LOB now that we are done using it: */
  if(OCILobFreeTemporary(svchp, errhp, tblob))
  {
    printf("OCILobFreeTemporary FAILED \n");
    return -1;
  }

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

  return;

}
```

COBOL（Pro*COBOL）：一時 LOB に対する LOB バッファリングの使用可能化

```
* Enabling LOB buffering for a temporary LOB.  [Example script: 3923.pco]
  IDENTIFICATION DIVISION.
  PROGRAM-ID. TEMP-LOB-BUFFERING.
  ENVIRONMENT DIVISION.
  DATA DIVISION.
  WORKING-STORAGE SECTION.

  01  USERID   PIC X(11) VALUES "SAMP/SAMP".
  01  TEMP-BLOB          SQL-BLOB.
  01  BUFFER             PIC X(80).
  01  AMT                PIC S9(9) COMP VALUE 10.
  01  ORASLNRD        PIC 9(4).

      EXEC SQL VAR BUFFER IS RAW(80) END-EXEC.
      EXEC SQL INCLUDE SQLCA END-EXEC.
      EXEC ORACLE OPTION (ORACA=YES) END-EXEC.
      EXEC SQL INCLUDE ORACA END-EXEC.

  PROCEDURE DIVISION.
  TEMP-LOB-BUFFERING.

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

 * Allocate and initialize the CLOB locators:
      EXEC SQL ALLOCATE :TEMP-BLOB END-EXEC.
      EXEC SQL
           LOB CREATE TEMPORARY :TEMP-BLOB
      END-EXEC.

 * Enable buffering for the temporary LOB:
      EXEC SQL LOB ENABLE BUFFERING :TEMP-BLOB END-EXEC.

 * Write some data to the temporary LOB here:
      MOVE '252525262626252525' TO BUFFER.
      EXEC SQL
           LOB WRITE ONE :AMT FROM :BUFFER
           INTO :TEMP-BLOB END-EXEC

 * Flush the buffered writes:
      EXEC SQL
           LOB FLUSH BUFFER :TEMP-BLOB FREE END-EXEC.
```

```
      * Disable buffering for the temporary LOB:
           EXEC SQL
                LOB DISABLE BUFFERING :TEMP-BLOB
           END-EXEC.
           EXEC SQL
                LOB FREE TEMPORARY :TEMP-BLOB
           END-EXEC.
           EXEC SQL FREE :TEMP-BLOB END-EXEC.
           STOP RUN.

       SQL-ERROR.
           EXEC SQL WHENEVER SQLERROR CONTINUE END-EXEC.
           MOVE ORASLNR TO ORASLNRD.
           DISPLAY " ".
           DISPLAY "ORACLE ERROR DETECTED ON LINE ", ORASLNRD, ":".
           DISPLAY " ".
           DISPLAY SQLERRMC.
           EXEC SQL ROLLBACK WORK RELEASE END-EXEC.
           STOP RUN.
```

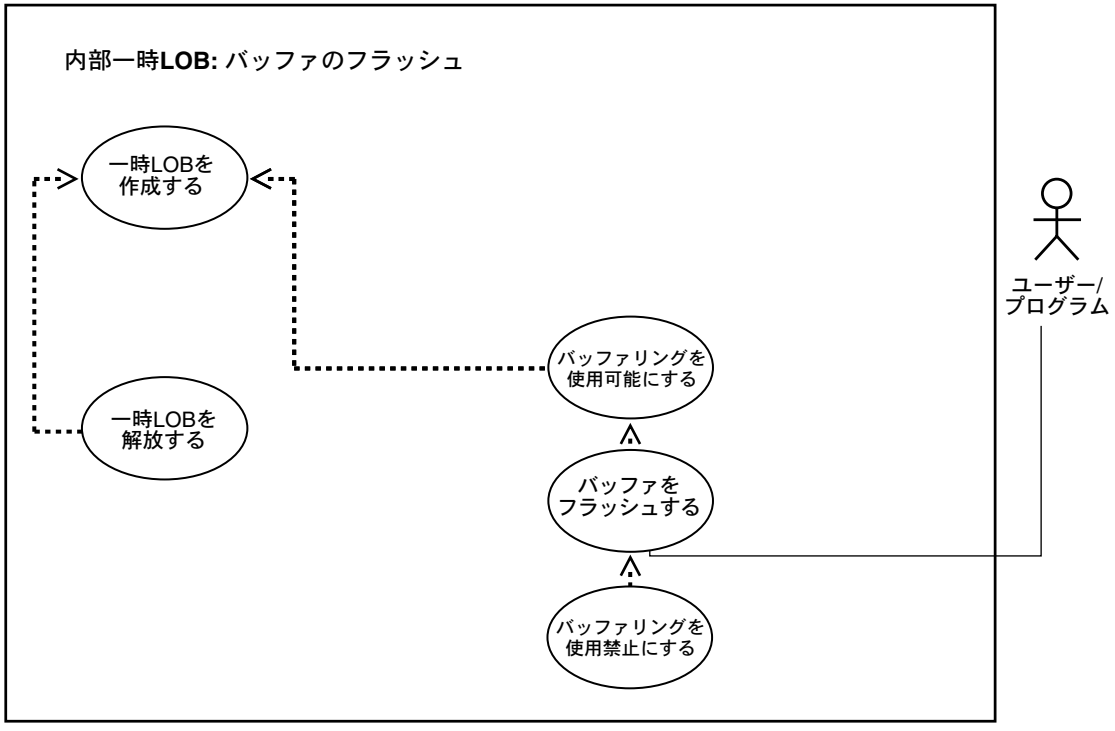
C/C++（Pro*C/C++）：一時 LOB に対する LOB バッファリングの使用可能化

```
/* Enabling LOB buffering for a temporary LOB.  [Example script: 3924.pc] */

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

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

#define BufferLength 1024

void enableBufferingTempLOB_proc()
{
  OCIClobLocator *Temp_loc;
  varchar Buffer[BufferLength];
  int Amount = BufferLength;
  int multiple, Length = 0, Position = 1;

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
```

```
  /* Allocate and Create the Temporary LOB: */
  EXEC SQL ALLOCATE :Temp_loc;
  EXEC SQL LOB CREATE TEMPORARY :Temp_loc;

  /* Enable use of the LOB Buffering Subsystem: */
  EXEC SQL LOB ENABLE BUFFERING :Temp_loc;
  memset((void *)Buffer.arr, 42, BufferLength);
  Buffer.len = BufferLength;
  for (multiple = 0; multiple < 8; multiple++)
    {
      /* Write Data to the Temporary LOB: */
      EXEC SQL LOB WRITE ONE :Amount
                    FROM :Buffer INTO :Temp_loc AT :Position;
      Position += BufferLength;
    }

  /* Flush the contents of the buffers and Free their resources: */
  EXEC SQL LOB FLUSH BUFFER :Temp_loc FREE;
  /* Turn off use of the LOB Buffering Subsystem: */
  EXEC SQL LOB DISABLE BUFFERING :Temp_loc;
  EXEC SQL LOB DESCRIBE :Temp_loc GET LENGTH INTO :Length;
  printf("Wrote %d characters using the Buffering Subsystem\n", Length);

  /* Free the Temporary LOB: */
  EXEC SQL LOB FREE TEMPORARY :Temp_loc;
  /* Release resources held by the Locator: */
  EXEC SQL FREE :Temp_loc;
}
void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  enableBufferingTempLOB_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

一時 LOB に対するバッファのフラッシュ

図 11-26 利用図 : 一時 LOB に対するバッファのフラッシュ



参照 : 11-2 ページの表 11-1「利用モデル : 内部一時 LOB」を参照してください。

用途

一時 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関数またはファンクションのリストは、第 3 章「様々なプログラム環境での LOB のサポート」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）:『Oracle Call Interface プログラマーズ・ガイド』の第 16 章「その他の OCI リレーショナル関数」の「LOB 関数」の「OCILobFlushBuffer()」
- COBOL（Pro*COBOL）:『Pro*COBOL Precompiler プログラマーズ・ガイド』の LOB に関する詳細、LOB 文の使用上の注意および付録 F「埋込み SQL およびプリコンパイラ・ディレクティブ」の「LOB FLUSH BUFFER（実行可能埋込み SQL 拡張機能）」
- C/C++（Pro*C/C++）:『Pro*C/C++ Precompiler プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびディレクティブ」の「LOB FLUSH BUFFER（実行可能埋込み SQL 拡張機能）」
- Visual Basic（OO4O）: 参照マニュアルはありません。
- Java（JDBC）: 10-246 ページの「バッファのフラッシュ」

使用例

ありません。

例

次のプログラム環境での例を示します。

- PL/SQL（DBMS_LOB）: 今回のリリースでは例は提供されません。
- C（OCI）: 一時 LOB に対するバッファのフラッシュ（11-202 ページ）
- COBOL（Pro*COBOL）: 一時 LOB に対するバッファのフラッシュ（11-203 ページ）
- C/C++（Pro*C/C++）: 一時 LOB に対するバッファのフラッシュ（11-205 ページ）
- Visual Basic（OO4O）: 今回のリリースでは例は提供されません。
- Java（JDBC）: 今回のリリースでは例は提供されません。

C（OCI）：一時 LOB に対するバッファのフラッシュ

```
/* Flushing the temporary LOB buffer. [Exmaple script: 3926]  */

sb4 lobBuffering (envhp, errhp, svchp, stmthp)
OCIEnv    *envhp;
OCIError  *errhp;
OCISvcCtx *svchp;
OCIStmt   *stmthp;
{
  OCILobLocator *tblob;
  ub4 amt;
  ub4 offset;
  sword retval;
  ub1 bufp[MAXBUFLEN];
  ub4 buflen;

  /* Allocate the descriptor for the lob locator: */
  (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &tblob,
                            (ub4)OCI_DTYPE_LOB, (size_t) 0, (dvoid **) 0);

  /* Select the BLOB: */
  printf (" create a temporary Lob\n");
  /* Create a temporary lob :*/
  if(OCILobCreateTemporary(svchp, errhp, tblob, (ub2)0,
                           SQLCS_IMPLICIT, OCI_TEMP_BLOB,
                           OCI_ATTR_NOCACHE, OCI_DURATION_SESSION))
  {
    (void) printf("FAILED: CreateTemporary() \n");
    return -1;
  }

  /* Open the BLOB: */
  if (OCILobOpen(svchp, errhp, (OCILobLocator *) tblob, OCI_LOB_READWRITE))
  {
    printf( "OCILobOpen FAILED for temp lob \n");
    return -1;
  }

  /* Enable LOB Buffering: */
  printf (" enable LOB buffering\n");
  checkerr (errhp, OCILobEnableBuffering(svchp, errhp, tblob));

  printf (" write data to LOB\n");

  /* Write data into the LOB: */
  amt    = sizeof(bufp);
```

```
  buflen = sizeof(bufp);
  offset = 1;
  checkerr (errhp, OCILobWrite (svchp, errhp, tblob, &amt,
                                offset, bufp, buflen,
                                OCI_ONE_PIECE, (dvoid *)0,
                                (sb4 (*)(dvoid*,dvoid*,ub4*,ub1 *))0,
                                0, SQLCS_IMPLICIT));
  /* Flush the buffer: */
  printf(" flush the LOB buffers\n");
  checkerr (errhp, OCILobFlushBuffer(svchp, errhp, tblob,
                                     (ub4)OCI_LOB_BUFFER_FREE));
  /* Disable Buffering: */
  printf (" disable LOB buffering\n");
  checkerr (errhp, OCILobDisableBuffering(svchp, errhp, tblob));

  /* Subsequent LOB WRITEs will not use the LOB Buffering Subsystem */

  /* Closing the BLOB is mandatory if you have opened it: */
  checkerr (errhp, OCILobClose(svchp, errhp, tblob));

  /* Free the temporary lob now that we are done using it: */
  if(OCILobFreeTemporary(svchp, errhp, tblob))
  {
    printf("OCILobFreeTemporary FAILED \n");
    return -1;
  }

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

  return;

}
```

COBOL（Pro*COBOL）：一時 LOB に対するバッファのフラッシュ

```
      * Flushing a temporary LOB buffer.  [Example script: 3927.pco]
        IDENTIFICATION DIVISION.
        PROGRAM-ID. FREE-TEMPORARY.
        ENVIRONMENT DIVISION.
        DATA DIVISION.
        WORKING-STORAGE SECTION.

        01  USERID   PIC X(11) VALUES "SAMP/SAMP".

        01  TEMP-BLOB      SQL-BLOB.
```

```
 01  IS-TEMP        PIC S9(9) COMP.
 01  ORASLNRD        PIC 9(4).

     EXEC SQL INCLUDE SQLCA END-EXEC.
     EXEC ORACLE OPTION (ORACA=YES) END-EXEC.
     EXEC SQL INCLUDE ORACA END-EXEC.

 PROCEDURE DIVISION.
 FREE-TEMPORARY.

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

* Allocate and initialize the BLOB locators:
     EXEC SQL ALLOCATE :TEMP-BLOB END-EXEC.
     EXEC SQL
          LOB CREATE TEMPORARY :TEMP-BLOB
     END-EXEC.

* Do something with the temporary LOB here:

* Free the temporary LOB:
     EXEC SQL
          LOB FREE TEMPORARY :TEMP-BLOB
     END-EXEC.
     EXEC SQL FREE :TEMP-BLOB END-EXEC.
     STOP RUN.

 SQL-ERROR.
     EXEC SQL
         WHENEVER SQLERROR CONTINUE
     END-EXEC.
     MOVE ORASLNR TO ORASLNRD.
     DISPLAY " ".
     DISPLAY "ORACLE ERROR DETECTED ON LINE ", ORASLNRD, ":".
     DISPLAY " ".
     DISPLAY SQLERRMC.
     EXEC SQL ROLLBACK WORK RELEASE END-EXEC.
     STOP RUN.
```

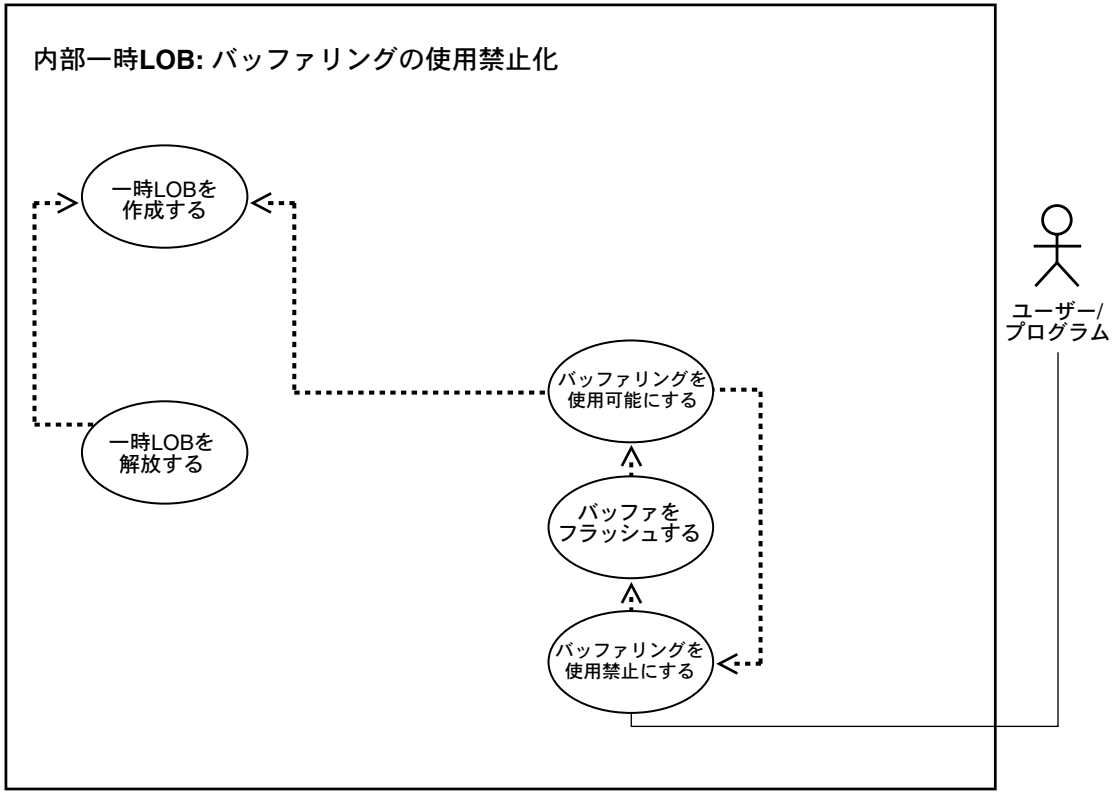
C/C++（Pro*C/C++）：一時 LOB に対するバッファのフラッシュ

```
/* Flushing a temporary LOB buffer.  [Example script: 3928.pc]  */

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

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

#define BufferLength 1024

void flushBufferingTempLOB_proc()
{
  OCIClobLocator *Temp_loc;
  varchar Buffer[BufferLength];
  int Amount = BufferLength;
  int multiple, Length = 0, Position = 1;

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  /* Allocate and Create the Temporary LOB: */
  EXEC SQL ALLOCATE :Temp_loc;
  EXEC SQL LOB CREATE TEMPORARY :Temp_loc;
  /* Enable use of the LOB Buffering Subsystem: */
  EXEC SQL LOB ENABLE BUFFERING :Temp_loc;
  memset((void *)Buffer.arr, 42, BufferLength);
  Buffer.len = BufferLength;
  for (multiple = 0; multiple < 8; multiple++)
    {
      /* Write Data to the Temporary LOB: */
      EXEC SQL LOB WRITE ONE :Amount
         FROM :Buffer INTO :Temp_loc AT :Position;
      Position += BufferLength;
    }
  /* Flush the contents of the buffers and Free their resources: */
  EXEC SQL LOB FLUSH BUFFER :Temp_loc FREE;

  /* Turn off use of the LOB Buffering Subsystem: */
  EXEC SQL LOB DISABLE BUFFERING :Temp_loc;
  EXEC SQL LOB DESCRIBE :Temp_loc GET LENGTH INTO :Length;
  printf("Wrote %d characters using the Buffering Subsystem\n", Length);
```

```
  /* Free the Temporary LOB */
  EXEC SQL LOB FREE TEMPORARY :Temp_loc;
  /* Release resources held by the Locator: */
  EXEC SQL FREE :Temp_loc;
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  flushBufferingTempLOB_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

一時 LOB に対する LOB バッファリングの使用禁止化

図 11-27 利用図 : 一時 LOB に対する LOB バッファリングの使用禁止化



参照 : 11-2 ページの表 11-1「利用モデル : 内部一時 LOB」を参照してください。

用途

一時 LOB バッファリングを使用禁止にします。

使用上の注意

少量のデータの読込みおよび書込みを行う場合は、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込みおよびストリーム書込みを行う場合は、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関数またはファンクションのリストは、第 3 章「様々なプログラム環境での LOB のサポート」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）:『Oracle Call Interface プログラマーズ・ガイド』の第 16 章「その他の OCI リレーショナル関数」の「LOB 関数」の「OCILobDisableBuffering()」
- COBOL（Pro*COBOL）:『Pro*COBOL Precompiler プログラマーズ・ガイド』の LOB に関する詳細、LOB 文の使用上の注意および付録 F「埋込み SQL およびプリコンパイラ・ディレクティブ」の「LOB DISABLE BUFFERING（実行可能埋込み SQL 拡張機能）」
- C/C++（Pro*C/C++）:『Pro*C/C++ Precompiler プログラマーズ・ガイド』の付録 F「埋込み SQL 文およびディレクティブ」の「LOB DISABLE BUFFERING（実行可能埋込み SQL 拡張機能）」
- Visual Basic（OO4O）: 参照マニュアルはありません。
- Java（JDBC）: 10-254 ページの「C（OCI）: LOB バッファリングの使用禁止化」

使用例

ありません。

例

次のプログラム環境での例を示します。

- PL/SQL（DBMS_LOB）：今回のリリースでは例は提供されません。
- C（OCI）：一時 LOB に対する LOB バッファリングの使用禁止化（11-209 ページ）
- COBOL（Pro*COBOL）：一時 LOB に対する LOB バッファリングの使用禁止化（11-211 ページ）
- C/C++（Pro*C/C++）：一時 LOB に対する LOB バッファリングの使用禁止化（11-212 ページ）
- Visual Basic（OO4O）：今回のリリースでは例は提供されません。
- Java（JDBC）：今回のリリースでは例は提供されません。

C（OCI）：一時 LOB に対する LOB バッファリングの使用禁止化

```
/* Disabling LOB buffering for temporary LOBs. [Example script: 3929.c] */

sb4 lobBuffering (envhp, errhp, svchp, stmthp)
OCIEnv    *envhp;
OCIError  *errhp;
OCISvcCtx *svchp;
OCIStmt   *stmthp;
{
  OCILobLocator *tblob;
  ub4 amt;
  ub4 offset;
  sword retval;
  ub1 bufp[MAXBUFLEN];
  ub4 buflen;

  /* Allocate the descriptor for the lob locator: */
  (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &tblob,
                            (ub4)OCI_DTYPE_LOB, (size_t) 0, (dvoid **) 0);

  /* Select the BLOB: */
  printf (" create a temporary Lob\n");
  /* Create a temporary LOB: */
  if(OCILobCreateTemporary(svchp,errhp, tblob, (ub2)0, SQLCS_IMPLICIT,
                           OCI_TEMP_BLOB,
                           OCI_ATTR_NOCACHE,
                           OCI_DURATION_SESSION))
  {
    (void) printf("FAILED: CreateTemporary() \n");
    return -1;
```

```
}

/* Open the BLOB: */
if (OCILobOpen(svchp, errhp, (OCILobLocator *) tblob, OCI_LOB_READWRITE))
{
  printf( "OCILobOpen FAILED for temp LOB \n");
  return -1;
}

/* Enable LOB Buffering: */
printf (" enable LOB buffering\n");
checkerr (errhp, OCILobEnableBuffering(svchp, errhp, tblob));

printf (" write data to LOB\n");

/* Write data into the LOB: */
amt    = sizeof(bufp);
buflen = sizeof(bufp);
offset = 1;
checkerr (errhp, OCILobWrite (svchp, errhp, tblob, &amt,
                              offset, bufp, buflen,
                              OCI_ONE_PIECE, (dvoid *)0,
                              (sb4 (*)(dvoid*,dvoid*,ub4*,ub1 *))0,
                              0, SQLCS_IMPLICIT));

/* Flush the buffer: */
printf(" flush the LOB buffers\n");
checkerr (errhp, OCILobFlushBuffer(svchp, errhp, tblob,
                                (ub4)OCI_LOB_BUFFER_FREE));

/* Disable Buffering: */
printf (" disable LOB buffering\n");
checkerr (errhp, OCILobDisableBuffering(svchp, errhp, tblob));

/* Subsequent LOB WRITEs will not use the LOB Buffering Subsystem */

/* Closing the BLOB is mandatory if you have opened it: */
checkerr (errhp, OCILobClose(svchp, errhp, tblob));

/* Free the temporary LOB now that we are done using it: */
if(OCILobFreeTemporary(svchp, errhp, tblob))
{
  printf("OCILobFreeTemporary FAILED \n");
  return -1;
}

/* Free resources held by the locators: */
```

```
  (void) OCIDescriptorFree((dvoid *) tblob, (ub4) OCI_DTYPE_LOB);

  return;

}
```

COBOL（Pro*COBOL）：一時 LOB に対する LOB バッファリングの使用禁止化

```
* Disabling LOB buffering for a temporary LOB.  [Example script: 3930.pco]
  IDENTIFICATION DIVISION.
  PROGRAM-ID. TEMP-LOB-BUFFERING.
  ENVIRONMENT DIVISION.
  DATA DIVISION.
  WORKING-STORAGE SECTION.

  01  USERID   PIC X(11) VALUES "SAMP/SAMP".
  01  TEMP-BLOB          SQL-BLOB.
  01  BUFFER             PIC X(80).
  01  AMT                PIC S9(9) COMP VALUE 10.
  01  ORASLNRD        PIC 9(4).

      EXEC SQL VAR BUFFER IS RAW(80) END-EXEC.
      EXEC SQL INCLUDE SQLCA END-EXEC.
      EXEC ORACLE OPTION (ORACA=YES) END-EXEC.
      EXEC SQL INCLUDE ORACA END-EXEC.

  PROCEDURE DIVISION.
  TEMP-LOB-BUFFERING.

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

 * Allocate and initialize the CLOB locators:
      EXEC SQL ALLOCATE :TEMP-BLOB END-EXEC.
      EXEC SQL
           LOB CREATE TEMPORARY :TEMP-BLOB
      END-EXEC.

 * Enable buffering for the temporary LOB:
      EXEC SQL
           LOB ENABLE BUFFERING :TEMP-BLOB
      END-EXEC.

 * Write some data to the temporary LOB here:
```

```
           MOVE '252525262626252525' TO BUFFER.
           EXEC SQL
                LOB WRITE ONE :AMT FROM :BUFFER
                INTO :TEMP-BLOB
           END-EXEC

      * Flush the buffered writes:
           EXEC SQL
                LOB FLUSH BUFFER :TEMP-BLOB FREE
           END-EXEC.

      * Disable buffering for the temporary LOB:
           EXEC SQL
                LOB DISABLE BUFFERING :TEMP-BLOB
           END-EXEC.
           EXEC SQL
                LOB FREE TEMPORARY :TEMP-BLOB
           END-EXEC.

           EXEC SQL FREE :TEMP-BLOB END-EXEC.
           STOP RUN.

       SQL-ERROR.
           EXEC SQL WHENEVER SQLERROR CONTINUE END-EXEC.
           MOVE ORASLNR TO ORASLNRD.
           DISPLAY " ".
           DISPLAY "ORACLE ERROR DETECTED ON LINE ", ORASLNRD, ":".
           DISPLAY " ".
           DISPLAY SQLERRMC.
           EXEC SQL ROLLBACK WORK RELEASE END-EXEC.
           STOP RUN.
```

C/C++（Pro*C/C++）：一時 LOB に対する LOB バッファリングの使用禁止化

```
/* Disabling LOB buffering for a temporary LOB.  [Example script: 3931.pc] */

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

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

```
  exit(1);
}

#define BufferLength 1024

void disableBufferingTempLOB_proc()
{
  OCIClobLocator *Temp_loc;
  varchar Buffer[BufferLength];
  int Amount = BufferLength;
  int multiple, Length = 0, Position = 1;

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();

  /* Allocate and Create the Temporary LOB: */
  EXEC SQL ALLOCATE :Temp_loc;
  EXEC SQL LOB CREATE TEMPORARY :Temp_loc;

  /* Enable use of the LOB Buffering Subsystem: */
  EXEC SQL LOB ENABLE BUFFERING :Temp_loc;
  memset((void *)Buffer.arr, 42, BufferLength);
  Buffer.len = BufferLength;
  for (multiple = 0; multiple < 7; multiple++)
    {

      /* Write Data to the Temporary LOB: */
      EXEC SQL LOB WRITE ONE :Amount
                    FROM :Buffer INTO :Temp_loc AT :Position;
      Position += BufferLength;
    }

  /* Flush the contents of the buffers and Free their resources: */
  EXEC SQL LOB FLUSH BUFFER :Temp_loc FREE;

  /* Turn off use of the LOB Buffering Subsystem:  */
  EXEC SQL LOB DISABLE BUFFERING :Temp_loc;

  /* Write APPEND can only be done when Buffering is Disabled: */
  EXEC SQL LOB WRITE APPEND ONE :Amount FROM :Buffer INTO :Temp_loc;
  EXEC SQL LOB DESCRIBE :Temp_loc GET LENGTH INTO :Length;

  printf("Wrote a total of %d characters\n", Length);

  /* Free the Temporary LOB: */
  EXEC SQL LOB FREE TEMPORARY :Temp_loc;

  /* Release resources held by the Locator: */
```

```
  EXEC SQL FREE :Temp_loc;
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  disableBufferingTempLOB_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

12

外部 LOB（BFILE）

利用モデル

この章では、外部 LOB に対する操作（「BFILE からのデータの読込み」など）を、利用方法の点から説明します。表 12-1「利用モデル図 : 外部 LOB（BFILE）」に、すべての利用方法を示します。

個々の利用方法

外部 LOB（BFILE）の個々の利用方法を、次のように説明します。

- **利用図** : 利用方法を表す図。これらの UML に基づいた図の見方については、付録 A「Unified Modeling Language 図」を参照してください。
- **用途** : LOB に関するこの利用方法の用途。
- **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、利用方法に対する LOB 関連操作の基礎となる構文。
- **使用例** : 利用方法の実装例について、サンプル・スキーマの点から説明した例。サンプル・スキーマの詳細は、『Oracle9*i* サンプル・スキーマ』を参照してください。
- **例** : 各利用方法の適用法。サンプル・スキーマを基にしています。

利用モデル : 外部 LOB（BFILE）

表 12-1「利用モデル図 : 外部 LOB（BFILE）」の「+」は、特定の利用方法で、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「様々なプログラム環境での LOB のサポート」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 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 を使用した Visual Basic
- **J** − JDBC を使用した Java
- **S** − SQL

表 12-1 利用モデル図 : 外部 LOB（BFILE）

利用方法およびページ	P	O	B	C	V	J
12-13 ページの「1 つ以上の BFILE 列を含む表の作成」	S	S	S	S	S	S
12-17 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
12-20 ページの「BFILE を含むネストした表を持つ表の作成」	S	S	S	S	S	S
12-22 ページの「BFILENAME() を使用した行の挿入」	S	+	+	+	+	+
12-31 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
12-33 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」	+	+	+	+	+	+
12-42 ページの「外部 LOB（BFILE）へのデータのロード」	S	S	S	S	S	S
12-46 ページの「LOB への BFILE データのロード」	+	+	+	+	+	+
12-55 ページの「BLOB への BFILE データのロード」	+	-	-	-	-	-
12-59 ページの「CLOB への BFILE データのロード」	+	-	-	-	-	-
12-63 ページの「BFILE をオープンする方法」	-	-	-	-	-	-
12-65 ページの「FILEOPEN を使用した BFILE のオープン」	+	+	-	-	-	+
12-70 ページの「OPEN を使用した BFILE のオープン」	+	+	+	+	+	+
12-78 ページの「BFILE がオープンしているかどうかを確認する方法」	-	-	-	-	-	-

表 12-1 利用モデル図 : 外部 LOB（BFILE）（続き）

利用方法およびページ	P	O	B	C	V	J
12-80 ページの「FILEISOPEN を使用した、BFILE がオープンしているかどうかの確認」	+	+	-	-	-	+
12-86 ページの「ISOPEN を使用した、BFILE がオープンしているかどうかの確認」	+	+	+	+	+	+
12-96 ページの「BFILE データの表示」	+	+	+	+	+	+
12-107 ページの「BFILE からのデータの読込み」	+	+	+	+	+	+
12-117 ページの「BFILE データの一部の読込み（substr）」	+	-	+	+	+	+
12-125 ページの「2 つの BFILE の全体または一部の比較」	+	-	+	+	+	+
12-135 ページの「BFILE 内のパターンの有無の確認（instr）」	+	-	+	+	-	+
12-143 ページの「BFILE が存在するかどうかの確認」	+	+	+	+	+	+
12-153 ページの「BFILE の長さの取得」	+	+	+	+	+	+
12-162 ページの「BFILE の LOB ロケータのコピー」	+	+	+	+	-	+
12-170 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」	-	+	-	+	-	-
12-175 ページの「BFILE の 2 つの LOB ロケータが等しいかどうかの確認」	-	+	-	+	-	+
12-182 ページの「ディレクトリ別名およびファイル名の取得」	+	+	+	+	+	+
12-191 ページの「BFILENAME() を使用した BFILE の更新」	S	S	S	S	S	S
12-194 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
12-196 ページの「初期化した BFILE ロケータを使用した BFILE の更新」	+	+	+	+	+	+
12-205 ページの「FILECLOSE を使用した BFILE のクローズ」	+	+	-	-	+	+
12-210 ページの「CLOSE を使用した BFILE のクローズ」	+	+	+	+	+	+
12-218 ページの「FILECLOSEALL を使用したオープン中のすべての BFILE のクローズ」	+	+	+	+	+	+
12-228 ページの「BFILE を含む表の行の削除」	S	S	S	S	S	S

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いずれかを使用します。

- Pro*C/C++ や Pro*COBOL などのプリコンパイラ
- OCI
- PL/SQL（DBMS_LOB パッケージ）
- Java（JDBC）
- 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「様々なプログラム環境での LOB のサポート」を参照してください。

ディレクトリ・オブジェクト

ディレクトリ・オブジェクトによって、Oracle サーバーにおける BFILE のアクセスおよび使用の管理が簡単になります（『Oracle9*i* SQL リファレンス』の第 13 章「SQL 文 : CREATE CLUSER ～ CREATE JAVA」の「CREATE DIRECTORY」を参照）。ディレクトリ・オブジェクトは、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示します。サーバーのファイル・システム内にあるファイルへは、ディレクトリ・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かぎり、アクセスできます。

BFILE ロケータの初期化

ディレクトリ・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しなくても済みます。ディレクトリ別名は、BFILE ロケータの初期化の際に、`BFILENAME()` ファンクション（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 OS 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。OS 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`*** でもかまいません）。

データベース・レコードへの OS ファイルの対応付け

OS ファイルを BFILE に対応付けるために、まず OS ファイルへのフルパス名の別名であるディレクトリ・オブジェクトを作成する必要があります。

既存の OS 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 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() ファンクションを使用して、BFILE を NULL に初期化してから、後で OS 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.gif および image2.gif が格納される物理ディレクトリを示すディレクトリ・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込みファンクションです。

SQL DML を使用して物理ファイルがレコードに対応付けられると、BFILE に対するその後の読込み操作は、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。このファンクションは、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、操作中のモジュール内にかぎり必要な場合、データベース内の列に対応付けすることなく、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、列の値を初期化した後、SELECT を使用してその列の値を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の必要がなくなります。

詳細は、DBMS_LOB.LOADFROMFILE の例を参照してください（12-46 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

ディレクトリ・オブジェクトのネーミング規則は、表および索引のネーミング規則と同じです。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は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、ディレクトリ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、ディレクトリ・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 / 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 データ定義言語（DDL）: ディレクトリ・オブジェクトの CREATE、REPLACE および ALTER
- SQL データ操作言語（DML）: ディレクトリ・オブジェクトに対する READ システム権限およびオブジェクト権限の GRANT および REVOKE

所有権および権限

ディレクトリ・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、『Oracle9*i* SQL リファレンス』を参照してください。Oracle9*i* 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込み権限

ディレクトリ・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。ディレクトリ・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ディレクトリ・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。

ディレクトリ・オブジェクトが表す物理ディレクトリには、Oracle サーバー・プロセスに対する適切な OS 権限（この場合は読込み）が含まれてい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 サーバー・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込み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込み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 サーバー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、『Oracle9*i* 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、『Oracle9*i* 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・オブジェクトのカタログ・ビュー

カタログ・ビューは、ディレクトリ・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）

  このビューには、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）

  このビューには、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。

ディレクトリ・オブジェクト使用のガイドライン

ディレクトリ機能の主な目的は、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、既存アプリケーションの変更が不要で、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 がディレクトリ・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。

- **ディレクトリ・オブジェクトは、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。**ディレクトリ・オブジェクトは、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 OS 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。ディレクトリ・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、ディレクトリ・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、ディレクトリ・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。ディレクトリ・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正常に実行されません。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停止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正常に実行されません。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・アクセスの管理にディレクトリ・オブジェクトを使用することは、OS 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 権限とともにディレクトリ・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。

共有サーバー（マルチスレッド・サーバー : MTS）・モードの BFILE

Oracle9*i* では、共有サーバー（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、共有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 操作が含まれる共有サーバーのセッションは、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、次回のリリースではなくなります。

外部 LOB（BFILE）ロケータ

BFILE では、値はサーバー側の OS 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、インラインに格納されます。

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、メンバー・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薦めします。
- **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。つまり、`OCIObjectNew()` の後でかつ `OCIObjectFlush()` の前に、`OCILobFileSetName()` 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 を挿入または更新すると、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。

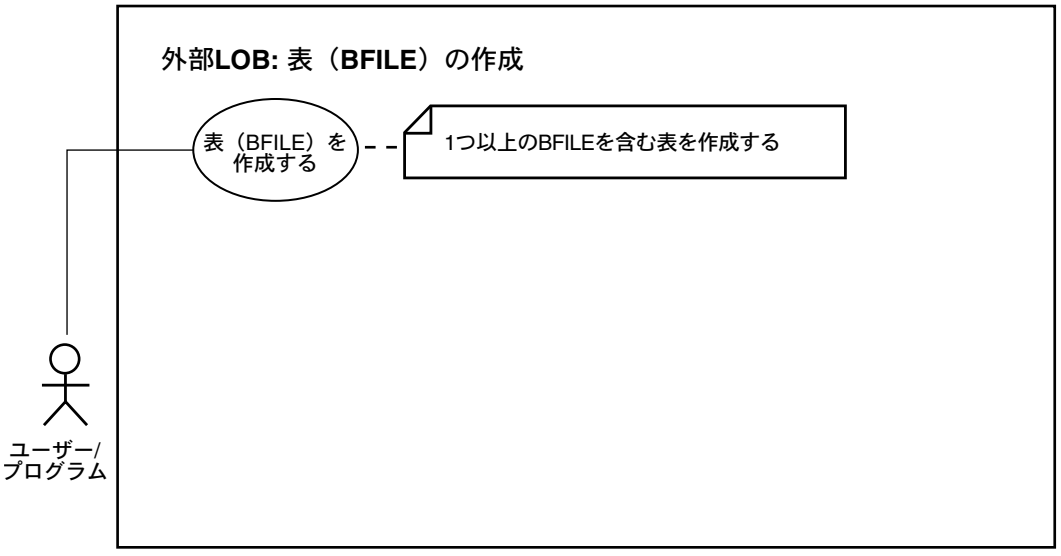
一般的な規則

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

1 つ以上の BFILE 列を含む表の作成

図 12-1 利用図 : 1 つ以上の BFILE 列を含む表の作成



参照 :

- 12-2 ページの表 12-1「利用モデル図 : 外部 LOB（BFILE）」
- 12-17 ページの「BFILE 属性を持つオブジェクト型の表の作成」
- 12-20 ページの「BFILE を含むネストした表を持つ表の作成」

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

SQL のデータ定義言語（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。

構文

次のマニュアルの項を参照してください。

- SQL:『Oracle9*i* SQL リファレンス』の第 15 章「SQL 文 : CREATE SYNONYM ～ CREATE TRIGGER」の「CREATE TABLE」

使用例

仮想アプリケーションの中心は、Print_media 表です。この表の列を構成する様々な型によって、印刷されたメディアに使用される何種類もの要素を 1 つにまとめることができます。

例

例を SQL で示します。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（12-14 ページ）

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場合もあります。

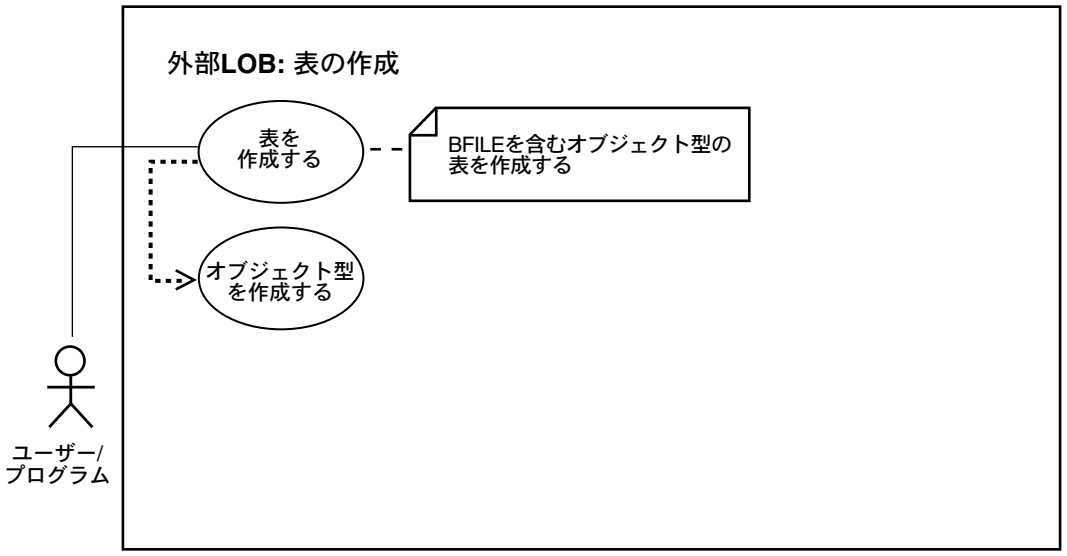
```
/*  Setup script for creating Print_media, Online_media and associated structures */

Rem  The HR and OE Schema need to be created before you create the PM Schema
Rem  For a detailed listing of the pm_drop.sql and pm_main.sql scripts see
Rem  the manual, Oracle9i Sample Schemas.

DROP USER pm CASCADE;
DROP DIRECTORY ADPHOTO_DIR;
DROP DIRECTORY ADCOMPOSITE_DIR;
DROP DIRECTORY ADGRAPHIC_DIR;
DROP INDEX onlinemedia CASCADE CONSTRAINTS;
DROP INDEX printmedia CASCADE CONSTRAINTS;
DROP TABLE online_media CASCADE CONSTRAINTS;
DROP TABLE print_media CASCADE CONSTRAINTS;
DROP TYPE textdoc_typ;
DROP TYPE textdoc_tab;
DROP TYPE adheader_typ;
DROP TABLE adheader_typ;
CREATE USER pm;
GRANT CONNECT, RESOURCE to pm;

CREATE DIRECTORY ADPHOTO_DIR AS '/tmp/';
CREATE DIRECTORY ADCOMPOSITE_DIR AS '/tmp/';
CREATE DIRECTORY ADGRAPHIC_DIR AS '/tmp/';
CREATE DIRECTORY media_dir AS '/tmp/';
GRANT READ ON DIRECTORY ADPHOTO_DIR to pm;
```

```
GRANT READ ON DIRECTORY ADCOMPOSITE_DIR to pm;
GRANT READ ON DIRECTORY ADGRAPHIC_DIR to pm;
GRANT READ ON DIRECTORY media_dir to pm;

CONNECT pm/pm  (or &pass);
COMMIT;

CREATE TABLE a_table (blob_col BLOB);

CREATE TYPE adheader_typ AS OBJECT (
   header_name    VARCHAR2(256),
   creation_date  DATE,
   header_text    VARCHAR(1024),
   logo           BLOB );

CREATE TYPE textdoc_typ AS OBJECT (
   document_typ   VARCHAR2(32),
   formatted_doc  BLOB);

CREATE TYPE Textdoc_ntab AS TABLE of textdoc_typ;

CREATE TABLE Adheader_tab of adheader_typ (
   header_name    VARCHAR2(256) CONSTRAINT hname CHECK (hname IS NOT NULL),
   creation_date  DATE DEFAULT NULL,
   logo           DEFAULT EMPTY_BLOB()
   );

CREATE TABLE online_media
( product_id  NUMBER(6),
product_photo ORDSYS.ORDImage,
product_photo_signature ORDSYS.ORDImageSignature,
product_thumbnail ORDSYS.ORDImage,
product_video ORDSYS.ORDVideo,
product_audio ORDSYS.ORDAudio,
product_text CLOB,
product_testimonials ORDSYS.ORDDoc);

CREATE UNIQUE INDEX onlinemedia_pk
  ON online_media (product_id);

ALTER TABLE online_media
ADD (CONSTRAINT onlinemedia_pk
PRIMARY KEY (product_id), CONSTRAINT loc_c_id_fk
FOREIGN KEY (product_id)  REFERENCES oe.product_information(product_id)
);

CREATE TABLE print_media
```

```
(product_id NUMBER(6),
ad_id NUMBER(6),
ad_composite BLOB,
ad_sourcetext CLOB,
ad_finaltext CLOB,
ad_fktextn NCLOB,
ad_testdocs_ntab textdoc_tab,
ad_photo BLOB,
ad_graphic BFILE,
ad_header adheader_typ,
press_release LONG) NESTED TABLE ad_textdocs_ntab STORE AS textdocs_nestedtab;

CREATE UNIQUE INDEX printmedia_pk
  ON print_media (product_id, ad_id);

ALTER TABLE print_media
ADD (CONSTRAINT printmedia_pk
PRIMARY KEY (product_id, ad_id),
CONSTRAINT printmedia_fk FOREIGN KEY (product_id)
REFERENCES oe.product_information(product_id)
);
```

BFILE 属性を持つオブジェクト型の表の作成

図 12-2 利用図 : BFILE を含む表の作成



参照 :

- 12-2 ページの表 12-1「利用モデル図 : 外部 LOB（BFILE）」を参照してください。
- 12-13 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。
- 12-20 ページの「BFILE を含むネストした表を持つ表の作成」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。SQL のデータ定義言語（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。

構文

次のマニュアルの項を参照してください。

- SQL:『Oracle9*i* SQL リファレンス』の第 15 章「SQL 文 : CREATE SYNONYM ～ CREATE TRIGGER」の「CREATE TABLE」および第 16 章「SQL 文 : CREATE TYPE ～ DROP ROLLBACK SEGMENT」の「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

例を SQL で示します。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（12-18 ページ）

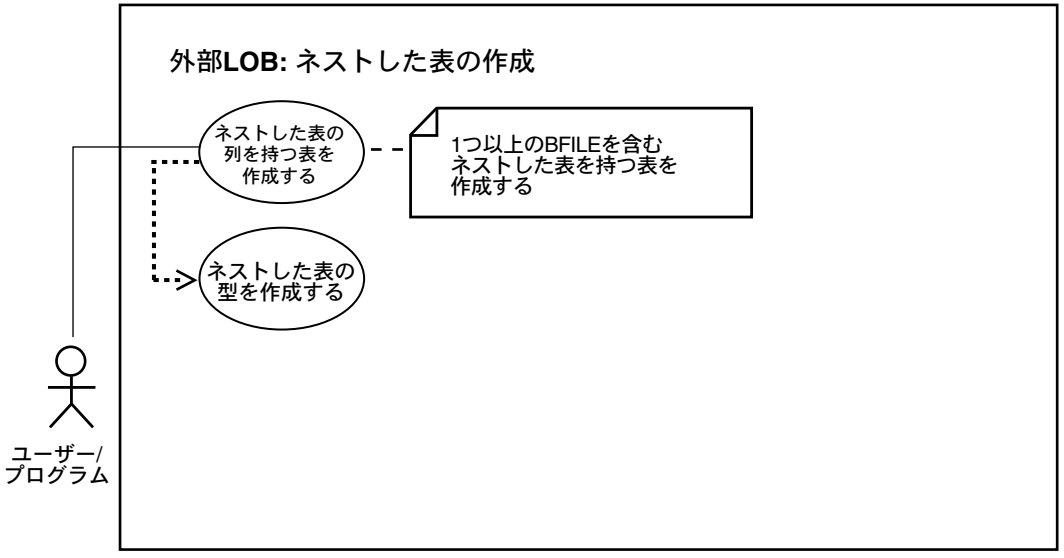
SQL: BFILE 属性を持つオブジェクト型の表の作成

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

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

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

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

BFILE を含むネストした表を持つ表の作成

図 12-3 利用図 : BFILE を含むネストした表を持つ表の作成



参照 :

- 12-2 ページの表 12-1「利用モデル図 : 外部 LOB（BFILE)」を参照してください。
- 12-13 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。
- 12-17 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。

用途

BFILE を含むネストした表を持つ表を作成します。

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ネストした表を作成する必要があります。SQL のデータ定義言語（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。

構文

次のマニュアルの項を参照してください。

- SQL:『Oracle9*i* SQL リファレンス』の第 15 章「SQL 文 : CREATE SYNONYM ～ CREATE TRIGGER」の「CREATE TABLE」および第 16 章「SQL 文 : CREATE TYPE ～ DROP ROLLBACK SEGMENT」の「CREATE TYPE」

使用例

次の例の Print_media 表は、textdoc_typ 型を含むネストした表 ad_textdoc_ntab を含んでいます。この型は 2 種類の LOB データ型を使用しています。BFILE は製品のグラフィック・イメージに使用し、BLOB は書式化されたドキュメントに使用します。

BFILE 列を持つ表を作成する方法については、12-13 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

例を SQL で示します。この例は、すべてのプログラム環境に適用されます。

- SQL: BFILE を含むネストした表を持つ表の作成（12-21 ページ）

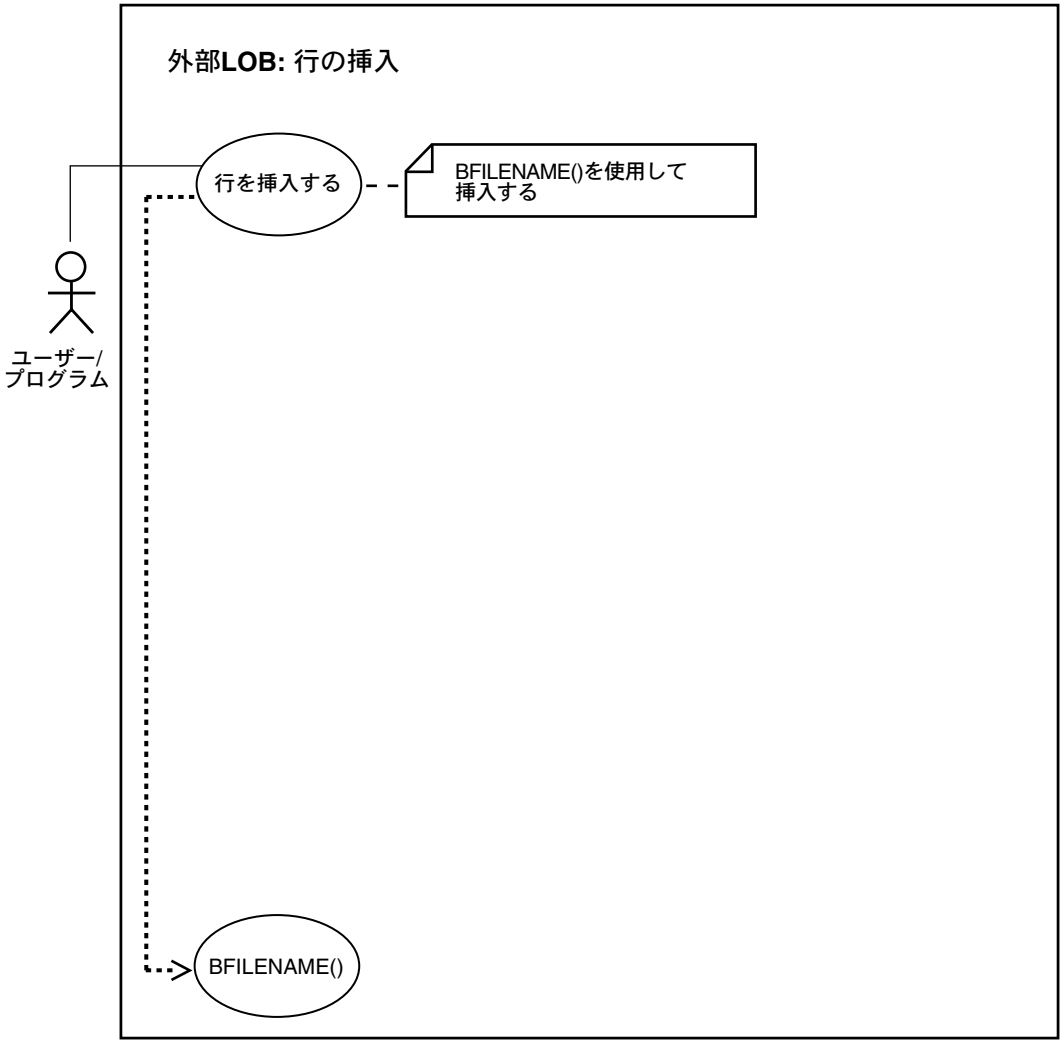
SQL: BFILE を含むネストした表を持つ表の作成

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 textdoc_typ AS OBJECT (
   document_typ    VARCHAR2(32),
   formatted_doc   BLOB);

CREATE TYPE Textdoc_ntab AS TABLE of textdoc_typ;

/* Embedding the nested table is accomplished when the structure
   of the containing table is defined. Using the PM sample schema,
   this is done by adding the following clause to the end of the CREATE
   Print_media statement: */

NESTED TABLE ad_textdocs_ntab STORE AS textdocs_nestedtab;
```

BFILENAME() を使用した行の挿入

図 12-4 利用図 : BILENAME() を使用した行の挿入



参照：

- 12-2 ページの表 12-1「利用モデル図 : 外部 LOB（BFILE）」を参照してください。
- 12-31 ページの「別の表からの BFILE の選択による BFILE 行の挿入」を参照してください。
- 12-33 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」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() ファンクションは、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ディレクトリ・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、ディレクトリ・オブジェクトおよび OS 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、このディレクトリ・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、ディレクトリ・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確認されません。このような確認は、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・アクセス時にかぎり実行されます。

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 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() ファンクションの 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： 12-7 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関数またはファンクションのリストは、第 3 章「様々なプログラム環境での LOB のサポート」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:『Oracle9*i* SQL リファレンス』の第 17 章「SQL 文 : DROP SEQUENCE ～ ROLLBACK」の「INSERT」
- C（OCI）:『Oracle Call Interface 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 16 章「その他の OCI リレーショナル関数」の「LOB 関数」
- COBOL（Pro*COBOL）:『Pro*COBOL Precompiler プログラマーズ・ガイド』の LOB に関する詳細、LOB 文の使用上の注意、付録 F「埋込み SQL およびプリコンパイラ・ディレクティブ」、および『Oracle9*i* SQL リファレンス』の第 17 章「SQL 文 : DROP SEQUENCE ～ ROLLBACK」の「INSERT」
- C/C++（Pro*C/C++）:『Pro*C/C++ Precompiler 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」、付録 F「埋込み SQL 文およびディレクティブ」、および『Oracle9*i* SQL リファレンス』の第 17 章「SQL 文 : DROP SEQUENCE ～ ROLLBACK」の「INSERT」
- Visual Basic（OO4O オンライン・ヘルプ）: ヘルプの「目次」タブから、「OO4O オートメーション・サーバー・リファレンス」>「OO4O サーバーのオブジェクト」>「OraDynaset」>「メソッド」>「AddNew」を選択
- Java（JDBC）:『Oracle9*i* JDBC 開発者ガイドおよびリファレンス』の第 8 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『Oracle9*i* 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「JDBC 2.0 LOB 型と Oracle 拡張型のサポート」の「BLOB、CLOB および BFILE のサポート」

例

次のプログラム環境での例を示します。

SQL: BFILENAME() を使用した行の挿入

```
/* Inserting a row using BFILENAME().  [Example script: 3945.sql]
   Note that this is the same INSERT statement as applied to internal
   persistent LOBs but with the BFILENAME() function added to initialize
   the BFILE columns: */

INSERT INTO Print_media VALUES (3106, 13001, EMPTY_BLOB(),
    EMPTY_CLOB(), EMPTY_CLOB(), EMPTY_CLOB(), NULL,
    EMPTY_BLOB(), BFILENAME('AD_GRAPHIC_DIR', '3106_keyboard'),
    NULL, "Your press release text goes here");
```

C（OCI）: BFILENAME() を使用した行の挿入

```
/* Inserting a row using BFILENAME.  [Example script: 3946.c] */

void insertUsingBfilename(svchp, stmthp, errhp)
OCISvcCtx *svchp;
OCIStmt *stmthp;
OCIError *errhp;
{
  text  *insstmt =
      (text *) "INSERT INTO Print_media VALUES (3060, 11001, EMPTY_BLOB(), \
         EMPTY_CLOB(), EMPTY_CLOB(), EMPTY_CLOB(), \
        (SELECT REF(ad) FROM Textdoc_ntab ad WHERE document_typ = 'PDF'), \
         EMPTY_BLOB(), BFILENAME ('ADGRAPHIC_DIR','monitor_3060_11001'), \
        (SELECT REF(adhead) FROM Adheader_typ Adhead \
           WHERE creation_date = '1-20-2001'), \
          "PRESS RELEASE \
Date of Press Release: January 11, 2001 \
Contact Information:  Shelley and Co., Oracle Corporation, 500 Oracle Parkway, \
Redwood City, CA 94065 \"
```

```
Disclaimer: This product, product name, and information is fictitious and has \
been composed to illustrate the functionality of Oracle products. \
Any similarity to existing products or product names is coincidental. \
TIGER2 3060 Monitor ..... an Exceptional Visual Experience! \
Oracle announces its return to manufacturing hardware and computer peripherals! \
The first model to have completed rigorous usability and stress tests is the \
TIGER2 +3060 17-Inch CRT MONITOR with its cousin the TIGER2 3060a 17-inch \
Flatscreen." \
)";

  /* Prepare the SQL statement */
  checkerr (errhp, OCIStmtPrepare(stmthp, errhp, insstmt, (ub4)
                                  strlen((char *) insstmt),
                                  (ub4) OCI_NTV_SYNTAX, (ub4)OCI_DEFAULT));
  /* Execute the SQL statement */
  checkerr (errhp, OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                                  (CONST OCISnapshot*) 0, (OCISnapshot*) 0,
                                  (ub4) OCI_DEFAULT));
}
```

COBOL（Pro*COBOL）: BFILENAME() を使用した行の挿入

```
* Inserting a row using BFILENAME()  [Example script: 3947.pco]
  IDENTIFICATION DIVISION.
  PROGRAM-ID. BFILE-INSERT.
  ENVIRONMENT DIVISION.
  DATA DIVISION.
  WORKING-STORAGE SECTION.

  01  USERID         PIC X(11) VALUES "SAMP/SAMP".
  01  ORASLNRD       PIC 9(4).

      EXEC SQL INCLUDE SQLCA END-EXEC.
      EXEC ORACLE OPTION (ORACA=YES) END-EXEC.
      EXEC SQL INCLUDE ORACA END-EXEC.

  PROCEDURE DIVISION.
  BFILE-INSERT.

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

      EXEC SQL
           INSERT INTO PRINT_MEDIA (PRODUCT_ID, AD_GRAPHIC)
```

```
            VALUES (1, BFILENAME('ADGRAPHIC_DIR', 'KEYBOARD_310_13001'))
    END-EXEC.
    EXEC SQL
         ROLLBACK WORK RELEASE
    END-EXEC.
    STOP RUN.

SQL-ERROR.
    EXEC SQL
        WHENEVER SQLERROR CONTINUE
    END-EXEC.
    MOVE ORASLNR TO ORASLNRD.
    DISPLAY " ".
    DISPLAY "ORACLE ERROR DETECTED ON LINE ", ORASLNRD, ":".
    DISPLAY " ".
    DISPLAY SQLERRMC.
    EXEC SQL
        ROLLBACK WORK RELEASE
    END-EXEC.
    STOP RUN.
```

C/C++（Pro*C/C++）: BFILENAME() を使用した行の挿入

```
/* Inserting a row using BFILENAME().  [Example script: 3948.pc] */

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

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

void BFILENAMEInsert_proc()
{
  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  EXEC SQL WHENEVER NOT FOUND CONTINUE;

  /* Delete any existing row: */
  EXEC SQL DELETE FROM Print_media WHERE product_id = 2056 AND ad_id = 12001;

  /* Insert a new row using the BFILENAME() function for BFILEs: */
```

```
  EXEC SQL INSERT INTO Print_media
    VALUES (2056, 12001, EMPTY_BLOB(), EMPTY_CLOB(), EMPTY_CLOB(), EMPTY_CLOB(),
       ad_textdocs(textdoc_typ(PDF, EMPTY_BLOB())),
       EMPTY_BLOB(),
       BFILENAME('ADGRAPHIC_DIR', 'mousepad_2056_12001'),
       NULL,
       'You Can't Beat this Mousepad for Ergonomic Value!!')
    ;
  printf("Inserted %d row\n", sqlca.sqlerrd[2]);
}

void main()
{
   char *samp = "samp/samp";
   EXEC SQL CONNECT :samp;
   BFILENAMEInsert_proc();
   EXEC SQL ROLLBACK WORK RELEASE;
}
```

Visual Basic（OO4O）: BFILENAME() を使用した行の挿入

```
'Inserting a row using BFILENAME(). [Example script: 3949.txt]

Dim OraDyn as OraDynaset, OraAdGraphic as OraBFile

Set OraDyn = OraDb.CreateDynaset("select * from Print_media", ORADYN_DEFAULT)
Set OraAdGraphic = OraDyn.Fields("ad_graphic").Value
OraDyn.AddNew
OraDyn.Fields("product_id").value = 3060
OraDyn.Fields("ad_sourcetext").value = Empty 'This is equivalent to EMPTY_CLOB() in
SQL
OraDyn.Fields("fltextn").value = Empty
'Initialize BFile Data:
OraAdGraphic.DirectoryName = "ADGRAPHIC_DIR"
OraAdGraphic.FileName = "monitor_graphic_3060_11001"
OraDyn.Fields("ad_composite").Value = Empty
OraDyn.Fields("ad_photo").Value = Empty
OraDyn.Update
'Add the row to the table
```

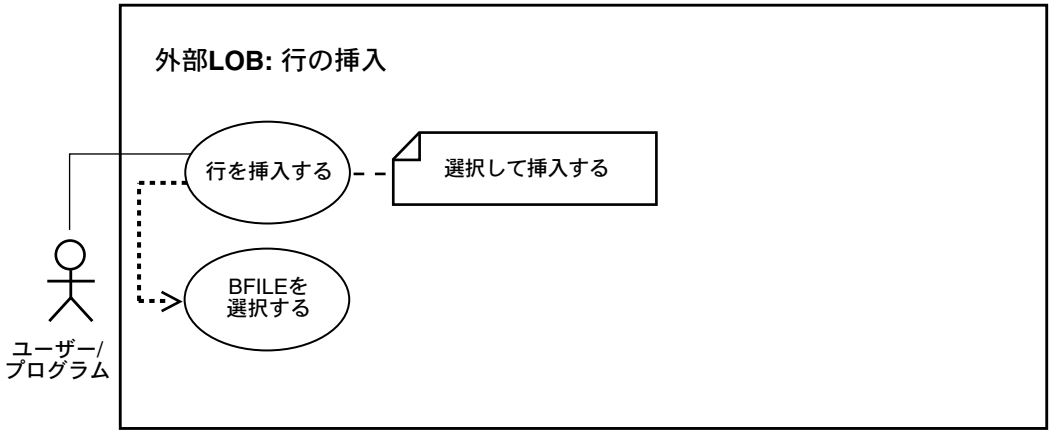
Java（JDBC）: BFILENAME() を使用した行の挿入

```
// Inserting a row using BFILENAME().  [Example script: 3951.java]

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

// Oracle Specific JDBC classes:
import oracle.sql.*;
import oracle.jdbc.driver.*;
public class Ex4_21
{
  public static void main (String args [])
       throws Exception
  {
    // Load the Oracle JDBC driver:
    DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

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

    // Create a Statement:
    Statement stmt = conn.createStatement ();
    try
    {
       stmt.execute("INSERT INTO Print_media "
          +"VALUES (3060, 11001, EMPTY_BLOB(), EMPTY_CLOB(), "
          +" EMPTY_CLOB(), EMPTY_CLOB(),"
          +"(SELECT REF(ad) FROM Textdoc_ntab ad"
          +"    WHERE document_typ = 'PDF'),"
          +"EMPTY_BLOB(), BFILENAME ('AD_GRAPHIC','monitor_3060'), "
          +"(SELECT REF(adhead) FROM Adheader_typ Adhead"
          +"    WHERE creation_date = '1-20-2001'), "
          +"PRESS RELEASE \"
+"Date of Press Release: January 11, 2001 \"
+"Contact Information:  Any name,Oracle Corporation, 500 Oracle Parkway," +"Redwood
City, CA 94065 \"
+"Disclaimer: This product, product name, and information is fictitious and has"
```

```
been" +"composed to illustrate the functionality of Oracle products. \"
+"Any similarity to existing products or product names is coincidental. \"
+"TIGER2 3060 Monitor ..... an Exceptional Visual Experience! \"
+"Oracle announces its return to manufacturing hardware and computer peripherals! \"
+"The first model to have completed rigorous usability and stress tests is the"
+"TIGER2 +3060 17-Inch CRT MONITOR with its cousin the TIGER2 3060a 17-inch"
+"Flatscreen. \"
+"Its initial offering is for $150 and its suggested retail value is $299. \"
);
   // Commit the transaction:
   conn.commit();
   stmt.close();
   conn.close();
    }
    catch (SQLException e)
    {
   e.printStackTrace();
    }
  }
}
```

別の表からの BFILE の選択による BFILE 行の挿入

図 12-5 利用図 : 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 :

- 12-2 ページの表 12-1「利用モデル図 : 外部 LOB（BFILE）」を参照してください。
- 12-22 ページの「BFILENAME() を使用した行の挿入」を参照してください。
- 12-33 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」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・アプローチを使用すると、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。

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。

構文

次のマニュアルの項を参照してください。

- SQL:『Oracle9*i* SQL リファレンス』の第 17 章「SQL 文 : DROP SEQUENCE ～ ROLLBACK」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

例を SQL で示します。この例は、すべてのプログラム環境に適用されます。

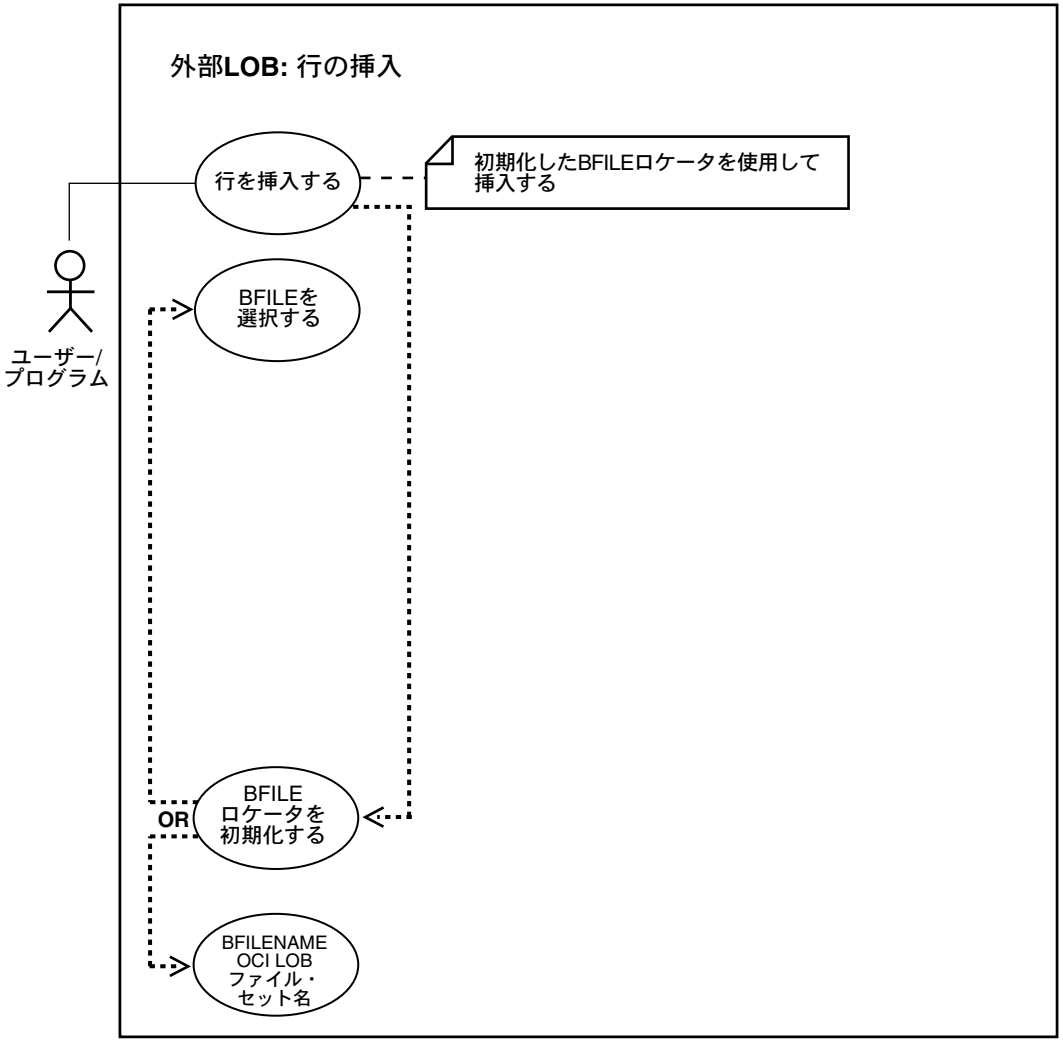
- SQL: 別の表からの BFILE の選択による BFILE を含む行の挿入（12-32 ページ）

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 12-6 利用図 : 初期化した BFILE ロケータを使用した BFILE を含む行の挿入



参照：

- 12-2 ページの表 12-1「利用モデル図 : 外部 LOB（BFILE）」を参照してください。
- 12-22 ページの「BFILENAME() を使用した行の挿入」を参照してください。
- 12-31 ページの「別の表からの BFILE の選択による BFILE 行の挿入」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。

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。

構文

各プログラム環境で使用可能な関数またはファンクションのリストは、第 3 章「様々なプログラム環境での LOB のサポート」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:『Oracle9*i* SQL リファレンス』の第 17 章「SQL 文 : DROP SEQUENCE 〜 ROLLBACK」の「INSERT」
- C（OCI）:『Oracle Call Interface 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 16 章「その他の OCI リレーショナル関数」の「LOB 関数」
- COBOL（Pro*COBOL）:『Pro*COBOL Precompiler プログラマーズ・ガイド』の LOB に関する詳細、LOB 文の使用上の注意、付録 F「埋込み SQL およびプリコンパイラ・ディレクティブ」、および『Oracle9*i* SQL リファレンス』の第 17 章「SQL 文 : DROP SEQUENCE 〜 ROLLBACK」の「INSERT」

- C/C++（Pro*C/C++）：『Pro*C/C++ Precompiler 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」、付録 F「埋込み SQL 文およびディレクティブ」の「LOB FILE SET（実行可能埋込み SQL 拡張機能）」、および『Oracle9*i* SQL リファレンス』の第 17 章「SQL 文 : DROP SEQUENCE ～ ROLLBACK」の「INSERT」
- Visual Basic（OO4O オンライン・ヘルプ）：ヘルプの「目次」タブから、「OO4O オートメーション・サーバー・リファレンス」>「OO4O サーバーのオブジェクト」>「OraBFILE」>「プロパティ」>「DirectoryName」、「FileName」、および「OO4O オートメーション・サーバー・リファレンス」>「OO4O サーバーのオブジェクト」>「OraDynaset」>「メソッド」>「Update」を選択
- Java（JDBC）：『Oracle9*i* JDBC 開発者ガイドおよびリファレンス』の第 8 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『Oracle9*i* 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「JDBC 2.0 LOB 型と Oracle 拡張型のサポート」の「BLOB、CLOB および BFILE のサポート」

使用例

次の例では、OS のソース・ファイル（`ADGRAPHIC_DIR`）から `ad_graphic` を挿入します。

例

次のプログラム環境での例を示します。

- PL/SQL（DBMS_LOB）：初期化した BFILE ロケータを使用した BFILE を含む行の挿入（12-36 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（12-36 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（12-37 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（12-38 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（12-39 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（12-40 ページ）

PL/SQL（DBMS_LOB）：初期化した BFILE ロケータを使用した BFILE を含む行の挿入

```
/* Inserting  row containing a BFILE by initializing a BFILE locator
   [Example script: 3953.sql]  */

DECLARE
  /* Initialize the BFILE locator: */
  Lob_loc  BFILE := BFILENAME('ADGRAPHIC_DIR', 'keyboard_graphic_3106_13001');
BEGIN
    INSERT INTO Print_media
    (product_id, ad_id, ad_graphic) VALUES (3106, 13001, Lob_loc);
    COMMIT;
END;
```

C（OCI）：初期化した BFILE ロケータを使用した BFILE を含む行の挿入

```
/* Inserting a row by initializing a BFILE Locator. [Example script: 3954.c] */

void insertUsingBfileLocator(envhp, svchp, stmthp, errhp)
OCIEnv *envhp;
OCISvcCtx *svchp;
OCIStmt *stmthp;
OCIError *errhp;
{
  text  *insstmt =
     (text *) "INSERT INTO Print_media (product_id, ad_graphic) \
         VALUES (2056, :Lob_loc)";
  OCIBind *bndhp;
  OCILobLocator *Lob_loc;
  OraText *Dir = (OraText *)"ADGRAPHIC_DIR", *Name = (OraText
*)"mousepad_2056_12001";

  /* Prepare the SQL statement: */
  checkerr (errhp, OCIStmtPrepare(stmthp, errhp, insstmt,  (ub4)
                                  strlen((char *) insstmt),
                                  (ub4) OCI_NTV_SYNTAX, (ub4)OCI_DEFAULT));
  /* Allocate Locator resources: */
  (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &Lob_loc,
                            (ub4)OCI_DTYPE_FILE, (size_t) 0, (dvoid **) 0);
  checkerr (errhp, OCILobFileSetName(envhp, errhp, &Lob_loc,
                                     Dir, (ub2)strlen((char *)Dir),
                                     Name,(ub2)strlen((char *)Name)));
  checkerr (errhp, OCIBindByPos(stmthp, &bndhp, errhp, (ub4) 1,
                                (dvoid *) &Lob_loc, (sb4) 0,  SQLT_BFILE,
                                (dvoid *) 0, (ub2 *)0, (ub2 *)0,
```

```
                                (ub4) 0, (ub4 *) 0, (ub4) OCI_DEFAULT));
  /* Execute the SQL statement: */
  checkerr (errhp, OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                                  (CONST OCISnapshot*) 0, (OCISnapshot*) 0,
                                  (ub4) OCI_DEFAULT));
  /* Free LOB resources: */
  OCIDescriptorFree((dvoid *) Lob_loc, (ub4) OCI_DTYPE_FILE);
}
```

COBOL（Pro*COBOL）：初期化した BFILE ロケータを使用した BFILE を含む行の挿入

```
* Inserting a row containing a BFILE by initializing a BFILE
* [Example script: 3955.pco]
  IDENTIFICATION DIVISION.
  PROGRAM-ID. BFILE-INSERT-INIT.
  ENVIRONMENT DIVISION.
  DATA DIVISION.
  WORKING-STORAGE SECTION.

  01  USERID         PIC X(11) VALUES "SAMP/SAMP".
  01  TEMP-BLOB      SQL-BLOB.
  01  SRC-BFILE      SQL-BFILE.
  01  DIR-ALIAS      PIC X(30) VARYING.
  01  FNAME          PIC X(20) VARYING.
  01  DIR-IND        PIC S9(4) COMP.
  01  FNAME-IND      PIC S9(4) COMP.
  01  AMT            PIC S9(9) COMP.
  01  ORASLNRD       PIC 9(4).

      EXEC SQL INCLUDE SQLCA END-EXEC.
      EXEC ORACLE OPTION (ORACA=YES) END-EXEC.
      EXEC SQL INCLUDE ORACA END-EXEC.

  PROCEDURE DIVISION.
  BFILE-INSERT-INIT.
      EXEC SQL WHENEVER SQLERROR DO PERFORM SQL-ERROR END-EXEC.
      EXEC SQL CONNECT :USERID END-EXEC.

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

 * Set up the directory and file information:
      MOVE "ADGRAPHIC_DIR" TO DIR-ALIAS-ARR.
      MOVE 9 TO DIR-ALIAS-LEN.
```

```
           MOVE "keyboard_graphic_3106_13001" TO FNAME-ARR.
           MOVE 16 TO FNAME-LEN.

       * Set the directory alias and filename in locator:
           EXEC SQL
              LOB FILE SET :SRC-BFILE DIRECTORY = :DIR-ALIAS,
              FILENAME = :FNAME END-EXEC.

           EXEC SQL
                INSERT INTO PRINT_MEDIA (PRODUCT_ID, AD_GRAPHIC)
                VALUES (3106, :SRC-BFILE)END-EXEC.
           EXEC SQL ROLLBACK WORK END-EXEC.
           EXEC SQL FREE :SRC-BFILE END-EXEC.
           STOP RUN.

        SQL-ERROR.
           EXEC SQL WHENEVER SQLERROR CONTINUE END-EXEC.
           MOVE ORASLNR TO ORASLNRD.
           DISPLAY " ".
           DISPLAY "ORACLE ERROR DETECTED ON LINE ", ORASLNRD, ":".
           DISPLAY " ".
           DISPLAY SQLERRMC.
           EXEC SQL ROLLBACK WORK RELEASE END-EXEC.
           STOP RUN.
```

C/C++（Pro*C/C++）: 初期化した BFILE ロケータを使用した BFILE を含む行の挿入

```
/* Inserting a row containing a BFILE by initializing a BFILE */
/* [Example script: 3958.pc] */
#include <oci.h>
#include <stdio.h>
#include <sqlca.h>
void Sample_Error()
{
  EXEC SQL WHENEVER SQLERROR CONTINUE;
  printf("%.*s\n", sqlca.sqlerrm.sqlerrml, sqlca.sqlerrm.sqlerrmc);
  EXEC SQL ROLLBACK WORK RELEASE;
  exit(1);
}

void insertBFILELocator_proc()
{
  OCIBFileLocator *Lob_loc;
  char *Dir = "ADGRAPHIC_DIR", *Name = "mousepad_graphic_2056_12001";
```

```
  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  /* Allocate the input Locator: */
  EXEC SQL ALLOCATE :Lob_loc;
  /* Set the Directory and Filename in the Allocated (Initialized) Locator: */
  EXEC SQL LOB FILE SET :Lob_loc DIRECTORY = :Dir, FILENAME = :Name;
  EXEC SQL INSERT INTO Print_media (Product_ID, ad_graphic) VALUES (2056, :Lob_loc);
  /* Release resources held by the Locator: */
  EXEC SQL FREE :Lob_loc;
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  insertBFILELocator_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
              }
}
```

Visual Basic（OO4O）: 初期化した BFILE ロケータを使用した BFILE を含む行の挿入

```
' Inserting a row containing a BFILE by initializing a BFILE.
' [Example script: 3959.txt]

Dim OraDyn as OraDynaset, OraPhoto as OraBFile, OraMusic as OraBFile
Set OraDyn = OraDb.CreateDynaset("select * from Print_media", ORADYN_DEFAULT)
Set OraMusic = OraDyn.Fields("ad_graphic").Value

'Edit the first row and initiliaze the "ad_graphic" column:
OraDyn.Edit
OraPhoto.DirectoryName = "ADGRAPHIC_DIR"
OraPhoto.Filename = "mousepad_graphic_2056_12001"
OraDyn.Update
```

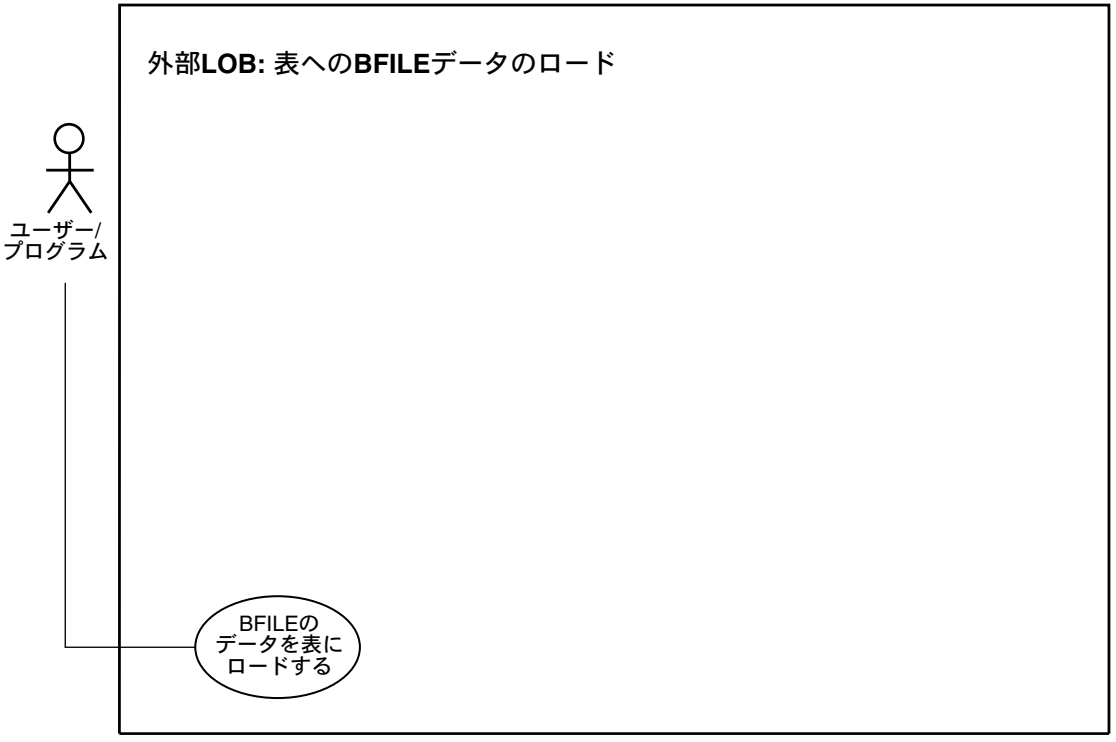
Java（JDBC）：初期化した BFILE ロケータを使用した BFILE を含む行の挿入

```
// Inserting a row containing a BFILE by initializing a BFILE.
// [Example script: 3960.java]

// Java IO classes:
import java.io.InputStream;
import java.io.OutputStream;

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

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

public class Ex4_26
{
  public static void main (String args [])
       throws Exception
  {
    // Load the Oracle JDBC driver:
    DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

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

    // Create a Statement:
    Statement stmt = conn.createStatement ();
    try
    {
       BFILE src_lob = null;
       ResultSet rset = null;
       OracleCallableStatement cstmt = null;
       rset = stmt.executeQuery (
          "SELECT BFILENAME('ADGRAPHIC_DIR','monitor_graphic_3060_11001') FROM
DUAL");
       if (rset.next())
       {
          src_lob = ((OracleResultSet)rset).getBFILE (1);
```

```
        }

        // Prepare a CallableStatement to OPEN the LOB for READWRITE:
        cstmt = (OracleCallableStatement) conn.prepareCall (
           "INSERT INTO Print_media (product_id, ad_graphic) VALUES (3060, ?)");
        cstmt.setBFILE(1, src_lob);
        cstmt.execute();

    //Close the statements and commit the transaction:
    stmt.close();
    cstmt.close();
    conn.commit();
    conn.close();
     }
     catch (SQLException e)
     {
        e.printStackTrace();
     }
   }
}
```

外部 LOB（BFILE）へのデータのロード

図 12-7 利用図 : 外部 LOB（BFILE）への初期データのロード



参照： 12-2 ページの表 12-1「利用モデル図 : 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 OS 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、必要なディレクトリ・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、『Oracle9*i* 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 ディレクティブで構成されます。

BFILE ディレクティブは、ディレクトリ・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。ディレクトリ・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。

- 『Oracle9*i* データベース・ユーティリティ』の「SQL*Loader」
- 第 4 章「LOB の管理」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 およびディレクトリ・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 adgraphic_photo as '/tmp';
CREATE OR REPLACE DIRECTORY adgraphic_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 : ファイル名およびディレクトリ・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 Print_media
FIELDS TERMINATED BY ','
(product_id  INTEGER EXTERNAL(6),
 FileName    FILLER CHAR(30),
 ad_graphic  BFILE(CONSTANT "modem_graphic_2268_21001", FileName))
```

データ・ファイル（sample9.dat）

```
007, modem_2268.jpg,
008, monitor_3060.jpg,
009, keyboard_2056.jpg,
```

注意： サイズが指定されない場合、product_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。ADGRAPHIC_PHOTO は、すべてのファイルが格納されるディレクトリです。ADGRAPHIC_DIR は、事前に作成されたディレクトリ・オブジェクトです。

BFILE へのデータのロード : ファイル名およびディレクトリ・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 Print_media
FIELDS TERMINATED BY ','
(
 product_id INTEGER EXTERNAL(6),
 ad_graphic BFILE (DirName, FileName),
 FileName  FILLER CHAR(30),
 DirName   FILLER CHAR(30)
)
```

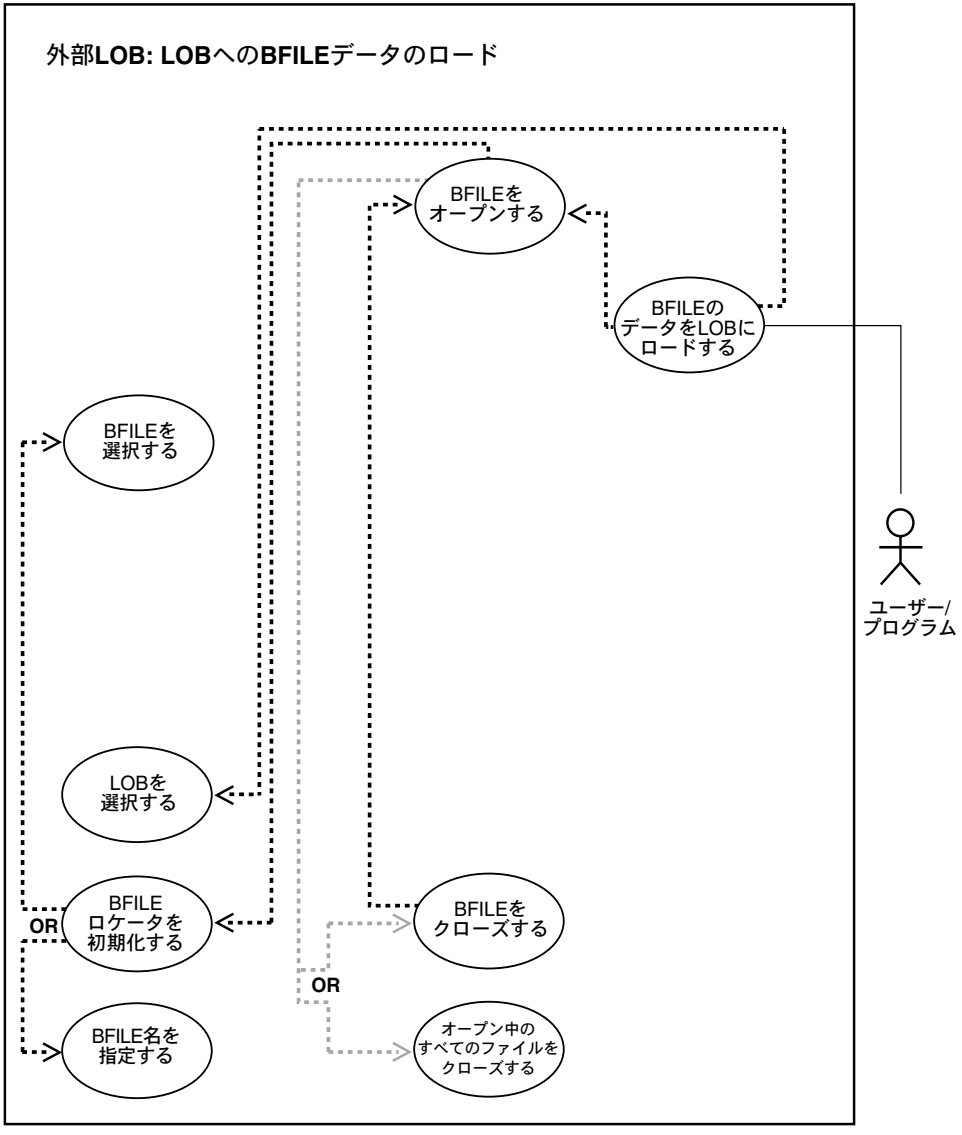
データ・ファイル（sample10.dat）

```
007,monitor_3060.jpg,ADGRAPHIC_PHOTO,
008,modem_2268.jpg,ADGRAPHIC_PHOTO,
009,keyboard_2056.jpg,ADGRAPHIC_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 12-8 利用図 : LOB への BFILE データのロード



参照：

- 内部一時 LOB に関するすべての基本操作については、12-2 ページの表 12-1「利用モデル図 : 外部 LOB（BFILE）」を参照してください。
- 12-55 ページの「BLOB への BFILE データのロード」を参照してください。
- 12-59 ページの「CLOB への BFILE データのロード」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

注意： LOADBLOBFROMFILE および LOADCLOBFROMFILE プロシージャでは、このプロシージャの機能が実装され、バイナリ・データおよび文字データをロードする機能が改善されています。改善されたプロシージャは、PL/SQL 環境のみで使用可能です。可能なかぎり、改善されたプロシージャのいずれかを使用することをお薦めします。詳細は、「BLOB への BFILE データのロード」および「CLOB への BFILE データのロード」を参照してください。

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に、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 に入れることになります。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。このため、テキストをロードする場合は LOADCLOBFROMFILE プロシージャを使用します。（12-59 ページの「CLOB への BFILE データのロード」を参照してください。）

参照： キャラクタ・セット変換の問題点については、『Oracle9*i* Database グローバリゼーション・サポート・ガイド』を参照してください。

BFILE 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関数またはファンクションのリストは、第 3 章「様々なプログラム環境での LOB のサポート」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）：『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』の第 23 章「DBMS_LOB」の「DBMS_LOB サブプログラムの要約」の「LOADFROMFILE プロシージャ」
- C（OCI）：『Oracle Call Interface 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 16 章「その他の OCI リレーショナル関数」の「LOB 関数」の「OCILobLoadFromFile()」
- COBOL（Pro*COBOL）：『Pro*COBOL Precompiler プログラマーズ・ガイド』の LOB に関する詳細、LOB 文の使用上の注意および付録 F「埋込み SQL およびプリコンパイラ・ディレクティブ」
- C/C++（Pro*C/C++）：『Pro*C/C++ Precompiler 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」および付録 F「埋込み SQL 文およびディレクティブ」の「LOB LOAD（実行可能埋込み SQL 拡張機能）」
- Visual Basic（OO4O オンライン・ヘルプ）: ヘルプの「目次」タブから、「OO4O オートメーション・サーバー・リファレンス」>「OO4O サーバーのオブジェクト」>「OraBLOB、OraCLOB」>「メソッド」>「CopyFromBFILE」を選択
- Java（JDBC）：『Oracle9*i* JDBC 開発者ガイドおよびリファレンス』の第 8 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『Oracle9*i* 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「JDBC 2.0 LOB 型と Oracle 拡張型のサポート」の「BLOB、CLOB および BFILE のサポート」

使用例

次のプロシージャの例では、宛先 LOB 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（`ADGRAPHIC_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での例を示します。

PL/SQL（DBMS_LOB）: LOB への BFILE データのロード

```
/* Loading a LOB with BFILE data.
   Procedure loadLOBFromBFILE_proc is not part of DBMS_LOB package: */

CREATE OR REPLACE PROCEDURE loadLOBFromBFILE_proc IS
   Dest_loc       BLOB;
   Src_loc        BFILE := BFILENAME('ADGRAPHIC_DIR',
                           'keyboard_graphic_3106_13001');
   Amount         INTEGER := 4000;
BEGIN
    SELECT ad_graphic INTO Dest_loc FROM Print_media
       WHERE product_id = 3060 AND ad_id = 13001 FOR UPDATE;
    /* Opening the LOB is mandatory: */
   DBMS_LOB.OPEN(Src_loc, DBMS_LOB.LOB_READONLY);
   /* Opening the LOB is optional: */
   DBMS_LOB.OPEN(Dest_loc, DBMS_LOB.LOB_READWRITE);
   DBMS_LOB.LOADFROMFILE(Dest_loc, Src_loc, Amount);
   /* Closing the LOB is mandatory if you have opened it: */
   DBMS_LOB.CLOSE(Dest_loc);
   DBMS_LOB.CLOSE(Src_loc);
   COMMIT;
END;
```

C（OCI）: LOB への BFILE データのロード

```
/* Loading a LOB with BFILE data.
   Select the lob/bfile from the Print_media table */

void selectLob(Lob_loc, errhp, svchp, stmthp)
OCILobLocator *Lob_loc;
OCIError *errhp;
OCISvcCtx *svchp;
OCIStmt *stmthp;
```

```
{
     char  selstmt[150];
     OCIDefine *dfnhp, *dfnhp2;

     strcpy(selstmt, (char *)  "SELECT ad_photo FROM Print_media \
                 WHERE product_id=3106 AND ad_id = 13001 FOR UPDATE");

     /* Prepare the SQL select statement */
     checkerr (errhp, OCIStmtPrepare(stmthp, errhp, selstmt,
                                     (ub4) strlen((char *) selstmt),
                                     (ub4) OCI_NTV_SYNTAX, (ub4)OCI_DEFAULT));
     /* Define the column being selected */
     checkerr (errhp, OCIDefineByPos(stmthp, &dfnhp, errhp, 1,
                                     (dvoid *)&Lob_loc, 0 , SQLT_BLOB,
                                     (dvoid *)0, (ub2 *)0, (ub2 *)0,
                                     OCI_DEFAULT)
       ||  OCIDefineByPos(stmthp, &dfnhp2, errhp, 2,
                                     (dvoid *)&Lob_loc, 0 , SQLT_BLOB,
                                     (dvoid *)0, (ub2 *)0, (ub2 *)0,
                                     OCI_DEFAULT));
     /* Execute the SQL select statement */
     checkerr (errhp, OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                                     (CONST OCISnapshot*) 0, (OCISnapshot*) 0,
                                     (ub4) OCI_DEFAULT));
}

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

  OCILobLocator *dest_loc;
  OCILobLocator *src_loc;

  /* Allocate locators */
  (void) OCIDescriptorAlloc((dvoid *) envhp,
                            (dvoid **) &dest_loc, (ub4)OCI_DTYPE_FILE,
                            (size_t) 0, (dvoid **) 0);

  (void) OCIDescriptorAlloc((dvoid *) envhp,
                            (dvoid **) &src_loc, (ub4)OCI_DTYPE_FILE,
                            (size_t) 0, (dvoid **) 0);

  checkerr(errhp, OCILobFileSetName(envhp, errhp, &src_loc,
                   (text *) "ADPHOTO_DIR", (ub2) strlen("ADPHOTO_DIR"),
```

```
                    (text *) "keyboard_photo_3106_13001",
                    (ub2) strlen(keyboard_photo_3106_13001")));

  selectLob(dest_loc, errhp, svchp, stmthp);

  checkerr(errhp, OCILobFileOpen(svchp, errhp, src_loc,
                                 (ub1)OCI_FILE_READONLY));
  checkerr(errhp, OCILobOpen(svchp, errhp, dest_loc, (ub1)OCI_LOB_READWRITE));
  checkerr (errhp, OCILobLoadFromFile(svchp, errhp, dest_loc, src_loc,
                                      (ub4)4000, (ub4)1, (ub4)1));
  checkerr(errhp, OCILobClose(svchp, errhp, dest_loc));
  checkerr(errhp, OCILobFileClose(svchp, errhp, src_loc));
}
```

COBOL（Pro*COBOL）: LOB への BFILE データのロード

```
      * Loading a LOB with BFILE data.
        IDENTIFICATION DIVISION.
        PROGRAM-ID. LOAD-BFILE.
        ENVIRONMENT DIVISION.
        DATA DIVISION.
        WORKING-STORAGE SECTION.
        01  USERID          PIC X(11) VALUES "SAMP/SAMP".
        01  DEST-BLOB       SQL-BLOB.
        01  SRC-BFILE       SQL-BFILE.
        01  DIR-ALIAS       PIC X(30) VARYING.
        01  FNAME           PIC X(20) VARYING.
        01  DIR-IND         PIC S9(4) COMP.
        01  FNAME-IND       PIC S9(4) COMP.
        01  AMT             PIC S9(9) COMP.
        01  ORASLNRD        PIC 9(4).

            EXEC SQL INCLUDE SQLCA END-EXEC.
            EXEC ORACLE OPTION (ORACA=YES) END-EXEC.
            EXEC SQL INCLUDE ORACA END-EXEC.

        PROCEDURE DIVISION.
        LOAD-BFILE.

       * Allocate and initialize the LOB locators:
            EXEC SQL ALLOCATE :DEST-BLOB END-EXEC.
            EXEC SQL ALLOCATE :SRC-BFILE END-EXEC.

       * Set up the directory and file information:
            MOVE "ADPHOTO_DIR" TO DIR-ALIAS-ARR.
            MOVE 9 TO DIR-ALIAS-LEN.
```

```
     MOVE "keyboard_photo_3106_13001" TO FNAME-ARR.
     MOVE 16 TO FNAME-LEN.

* Populate the BFILE:
     EXEC SQL WHENEVER NOT FOUND GOTO END-OF-BFILE END-EXEC.
     EXEC SQL
          SELECT AD_GRAPHIC INTO :SRC-BFILE
          FROM PRINT_MEDIA WHERE PRODUCT_ID = 3106 AND AD_ID = 13001
          END-EXEC.

* Open the source BFILE READ ONLY.
* Open the destination BLOB READ/WRITE:
     EXEC SQL LOB OPEN :SRC-BFILE READ ONLY END-EXEC.
     EXEC SQL LOB OPEN :DEST-BLOB READ WRITE END-EXEC.

* Load BFILE data into the BLOB:
     EXEC SQL
          LOB LOAD :AMT FROM FILE :SRC-BFILE INTO :DEST-BLOB END-EXEC.

* Close the LOBs:
     EXEC SQL LOB CLOSE :SRC-BFILE END-EXEC.
     EXEC SQL LOB CLOSE :DEST-BLOB END-EXEC.

* And free the LOB locators:
 END-OF-BFILE.
     EXEC SQL WHENEVER NOT FOUND CONTINUE END-EXEC.
     EXEC SQL FREE :DEST-BLOB END-EXEC.
     EXEC SQL FREE :SRC-BFILE END-EXEC.
     EXEC SQL ROLLBACK WORK RELEASE END-EXEC.
     STOP RUN.

 SQL-ERROR.
     EXEC SQL
         WHENEVER SQLERROR CONTINUE
     END-EXEC.
     MOVE ORASLNR TO ORASLNRD.
     DISPLAY " ".
     DISPLAY "ORACLE ERROR DETECTED ON LINE ", ORASLNRD, ":".
     DISPLAY " ".
     DISPLAY SQLERRMC.
     EXEC SQL ROLLBACK WORK RELEASE END-EXEC.
     STOP RUN.
```

C/C++（Pro*C/C++）: LOB への BFILE データのロード

```
/* Loading a LOB with BFILE data. */

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

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

void loadLOBFromBFILE_proc()
{
  OCIBlobLocator *Dest_loc;
  OCIBFileLocator *Src_loc;
  char *Dir = "ADGRAPHIC_DIR", *Name = "mousepad_graphic_2056_12001";
  int Amount = 4096;

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();

  /* Initialize the BFILE Locator: */
  EXEC SQL ALLOCATE :Src_loc;
  EXEC SQL LOB FILE SET :Src_loc DIRECTORY = :Dir, FILENAME = :Name;

  /* Initialize the BLOB Locator: */
  EXEC SQL ALLOCATE :Dest_loc;
  EXEC SQL SELECT ad_photo INTO :Dest_loc FROM Print_media
           WHERE Product_ID = 2056 AND AD_ID = 12001 FOR UPDATE;

  /* Opening the BFILE is Mandatory: */
  EXEC SQL LOB OPEN :Src_loc READ ONLY;

  /* Opening the BLOB is Optional: */
  EXEC SQL LOB OPEN :Dest_loc READ WRITE;
  EXEC SQL LOB LOAD :Amount FROM FILE :Src_loc INTO :Dest_loc;

  /* Closing LOBs and BFILEs is Mandatory if they have been OPENed: */
  EXEC SQL LOB CLOSE :Dest_loc;
  EXEC SQL LOB CLOSE :Src_loc;

  /* Release resources held by the Locators: */
  EXEC SQL FREE :Dest_loc;
```

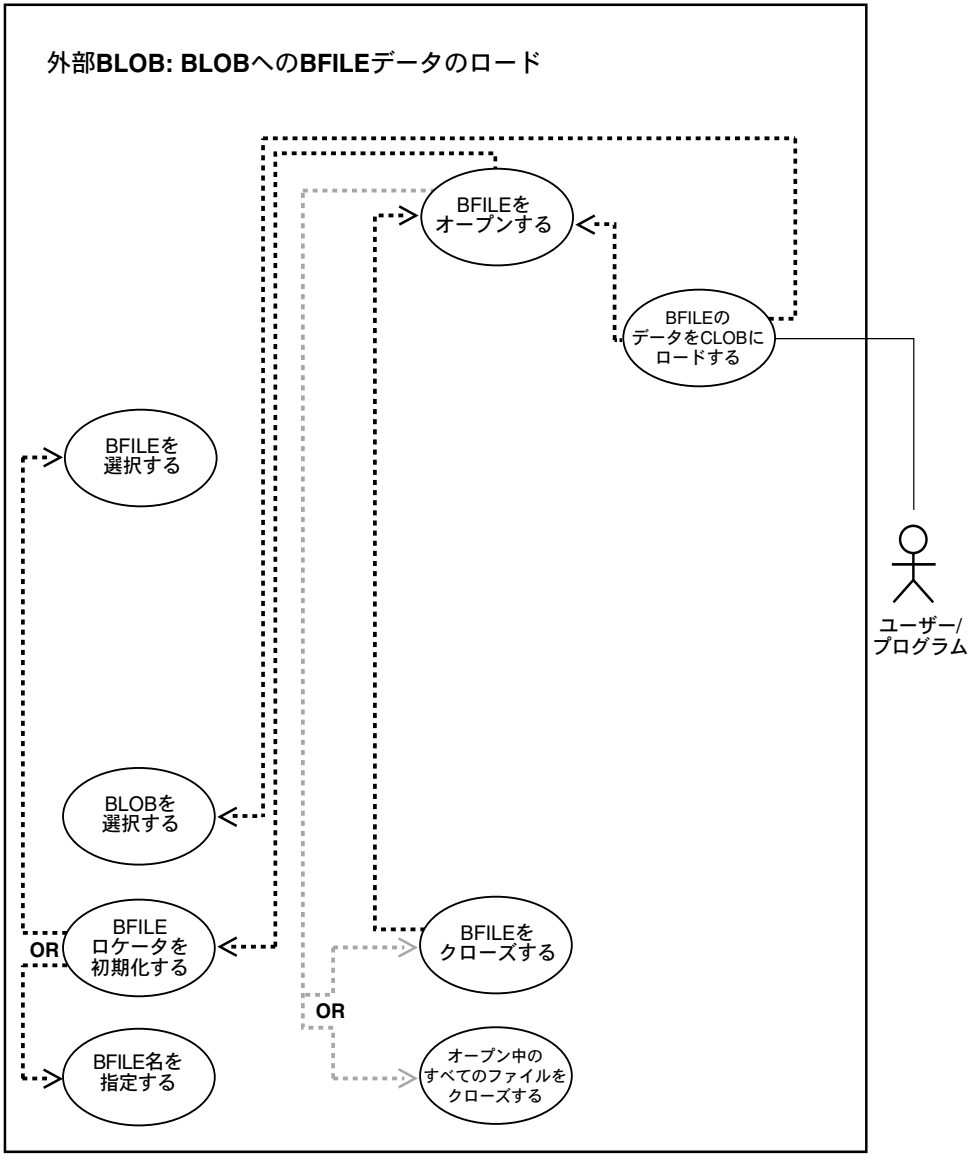
```
  EXEC SQL FREE :Src_loc;
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  loadLOBFromBFILE_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

Visual Basic（OO4O）: LOB への BFILE データのロード

```
'Loading a LOB with BFILE data

Dim OraDyn as OraDynaset, OraDyn2 as OraDynaset, OraAdGraphic as OraBFile
Dim OraAdPhoto as OraBlob

chunksize = 32767
Set OraDyn = OraDb.CreateDynaset("select * from Print_media", ORADYN_DEFAULT)

Set OraAdGraphic = OraDyn.Fields("ad_graphic").Value
Set OraAdPhoto = OraDyn.Fields("ad_photo").Value

OraDyn.Edit
'Load LOB with data from BFILE:
OraAdPhoto.CopyFromBFile (OraAdGraphic)
OraDyn.Update
```

BLOB への BFILE データのロード

図 12-9 利用図 : BLOB へのバイナリ・データのロード



参照 :

- 内部一時 LOB に関するすべての基本操作については、12-2 ページの表 12-1「利用モデル図 : 外部 LOB（BFILE）」を参照してください。
- 12-46 ページの「LOB への BFILE データのロード」を参照してください。
- 12-59 ページの「CLOB への BFILE データのロード」を参照してください。

用途

BFILE のバイナリ・データを BLOB にロードします。LOADFROMFILE と同じ結果が得られると同時にユーザへの新規のオフセットを返します。

使用上の注意

バイナリ・データをロードする場合は LOADBLOBFROMFILE を使用し、テキストをロードする場合は LOADCLOBFROMFILE を使用します。この機能は、クライアント側で BFILE をサポートしないため、ロードはサーバー側のみで行われます。LOADCLOBFROMFILE API を使用すると、BFILE のキャラクタ・セット ID を指定できるため、BFILE データ・キャラクタ・セットから宛先 CLOB/NCLOB のキャラクタ・セットへの変換が適切に実行されます。

構文

各プログラム環境で使用可能な関数またはファンクションのリストは、第 3 章「様々なプログラム環境での LOB のサポート」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）:『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』の第 23 章「DBMS_LOB」の「DBMS_LOB サブプログラムの要約」の「LOADBLOBFROMFILE プロシージャ」

使用例

この項のプロシージャの例では、製品メディアのサンプル・スキーマの Print_media 表を使用します。また、ターゲットとなる BLOB にロードするバイナリ LOB データを含む OS のソース・ディレクトリが存在することを想定しています。

例

「PL/SQL（DBMS_LOB）: 内部永続 BLOB への BFILE データのロード」の例では、PL/SQL プログラム環境での LOADBLOBFROMFILE の使用方法を示します。（他のプログラム環境はサポートされていません。）

PL/SQL（DBMS_LOB）：内部永続 BLOB への BFILE データのロード

次の項目の例を示します。

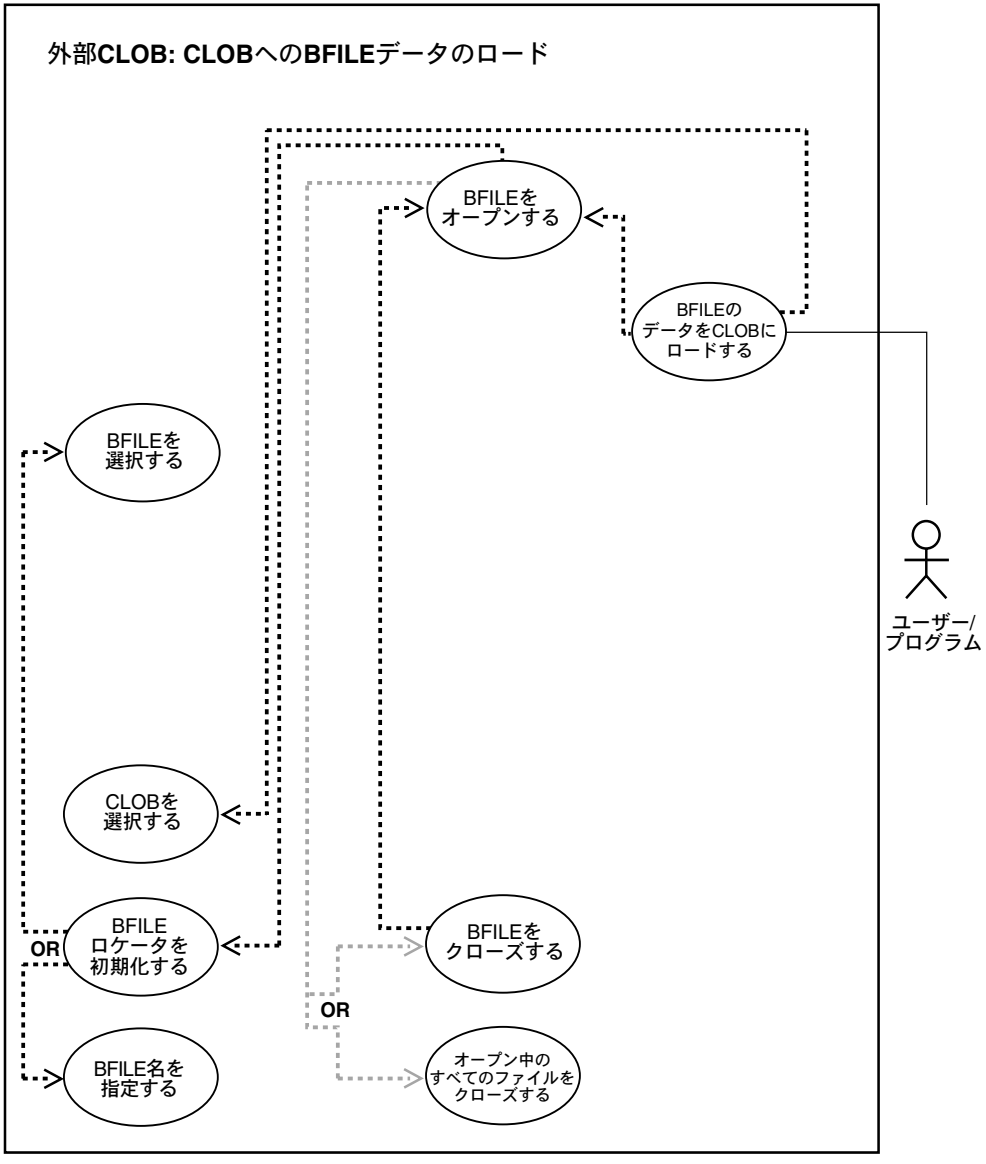
- BFILE のバイナリ・データを内部永続 BLOB にロードする方法。
- LOADBLOBFROMFILE を使用して長さを先に取得せずにファイル全体をロードする方法。
- オフセットの戻り値を使用して実際にロードされた量を計算する方法。

```
DECLARE
  src_loc     BFILE := bfilename('ADVERT_DIR','display_ad_frame') ;
  dst_loc     BLOB;
  src_offset  NUMBER := 1;
  dst_offset  NUMBER := 1;
  src_osin    NUMBER;
  dst_osin    NUMBER;
  bytes_rd    NUMBER;
  bytes_wt    NUMBER;
BEGIN
  SELECT ad_composite INTO dst_loc FROM Print_media
      WHERE product_id=3106 and ad_id=13001 FOR UPDATE;

  /* Opening the source BFILE is mandatory */
  dbms_lob.fileopen(src_loc, dbms_lob.file_readonly);

  /* Opening the LOB is optional */
  dbms_lob.OPEN(dst_loc, dbms_lob.lob_readwrite);
  /* Save the input source/destination offsets */
  src_osin := src_offset;
  dst_osin := dst_offset;
  /* Use LOBMAXSIZE to indicate loading the entire BFILE */

dbms_lob.LOADBLOBFROMFILE(dst_loc,src_loc,dbms_lob.lobmaxsize,src_offset,dst_offset)
;

  /* Closing the LOB is mandatory if you have opened it */
  dbms_lob.close(dst_loc);
  dbms_lob.filecloseall();
  COMMIT;

  /* Use the src_offset returned to calculate the actual amount read from the BFILE
*/
  bytes_rd := src_offset - src_osin;
  dbms_output.put_line(' Number of bytes read from the BFILE ' || bytes_rd ) ;
```

```
  /* Use the dst_offset returned to calculate the actual amount written to the BLOB
*/
  bytes_wt := dst_offset - dst_osin;
  dbms_output.put_line(' Number of bytes written to the BLOB ' || bytes_wt ) ;
  /* If there is no exception the number of bytes read should equal to the number of
bytes written */

END ;
```

CLOB への BFILE データのロード

図 12-10 利用図 : キャラクタ・データへの CLOB または NCLOB のロード



参照 :

- 内部一時 LOB に関するすべての基本操作については、12-2 ページの表 12-1「利用モデル図 : 外部 LOB（BFILE）」を参照してください。
- 12-46 ページの「LOB への BFILE データのロード」を参照してください。
- 12-55 ページの「BLOB への BFILE データのロード」を参照してください。

用途

BFILE のキャラクタ・データを内部永続 CLOB または NCLOB にロードします。

使用上の注意

バイナリ・データをロードする場合は LOADBLOBFROMFILE を使用し、テキストをロードする場合は LOADCLOBFROMFILE を使用します。後者のメソッドでは BFILE のキャラクタ・セット ID を指定できます。この機能はクライアント側で BFILE をサポートしないため、ロードはサーバー側のみで行われます。LOADCLOBFROMFILE API を使用すると、BFILE のキャラクタ・セット ID を指定できるため、BFILE データ・キャラクタ・セットから宛先 CLOB/NCLOB のキャラクタ・セットへの変換が適切に実行されます。

構文

各プログラム環境で使用可能な関数またはファンクションのリストは、第 3 章「様々なプログラム環境での LOB のサポート」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）：『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』の第 23 章 「DBMS_LOB」の「DBMS_LOB サブプログラムの要約」の「LOADCLOBFROMFILE プロシージャ」

使用例

この項のプロシージャの例では、製品メディアのサンプル・スキーマの Print_media 表を使用し、ターゲットとなる CLOB または NCLOB にロードするキャラクタ LOB データを含む OS のソース・ディレクトリが存在することを想定しています。

例

「PL/SQL（DBMS_LOB）: CLOB/NCLOB への BFILE データのロード」の例では、PL/SQL プログラム環境での LOADCLOBFROMFILE の使用方法を示します。（他のプログラム環境はサポートされていません。）

参照：

- PL/SQL（DBMS_LOB）: 内部永続 CLOB への BFILE データのロード（10-48 ページ）
- PL/SQL（DBMS_LOB）: 一時 CLOB/NCLOB への BFILE データのロード（11-52 ページ）

PL/SQL（DBMS_LOB）: CLOB/NCLOB への BFILE データのロード

次の項目の例を示します。

- BFILE のキャラクタ・データを内部永続 CLOB または NCLOB にロードする方法。
- デフォルト csid（0）の使用方法。
- BFILE の getlength を呼び出さずにファイル全体をロードする方法。
- 返されたオフセットを使用して実際のロード量を求める方法。

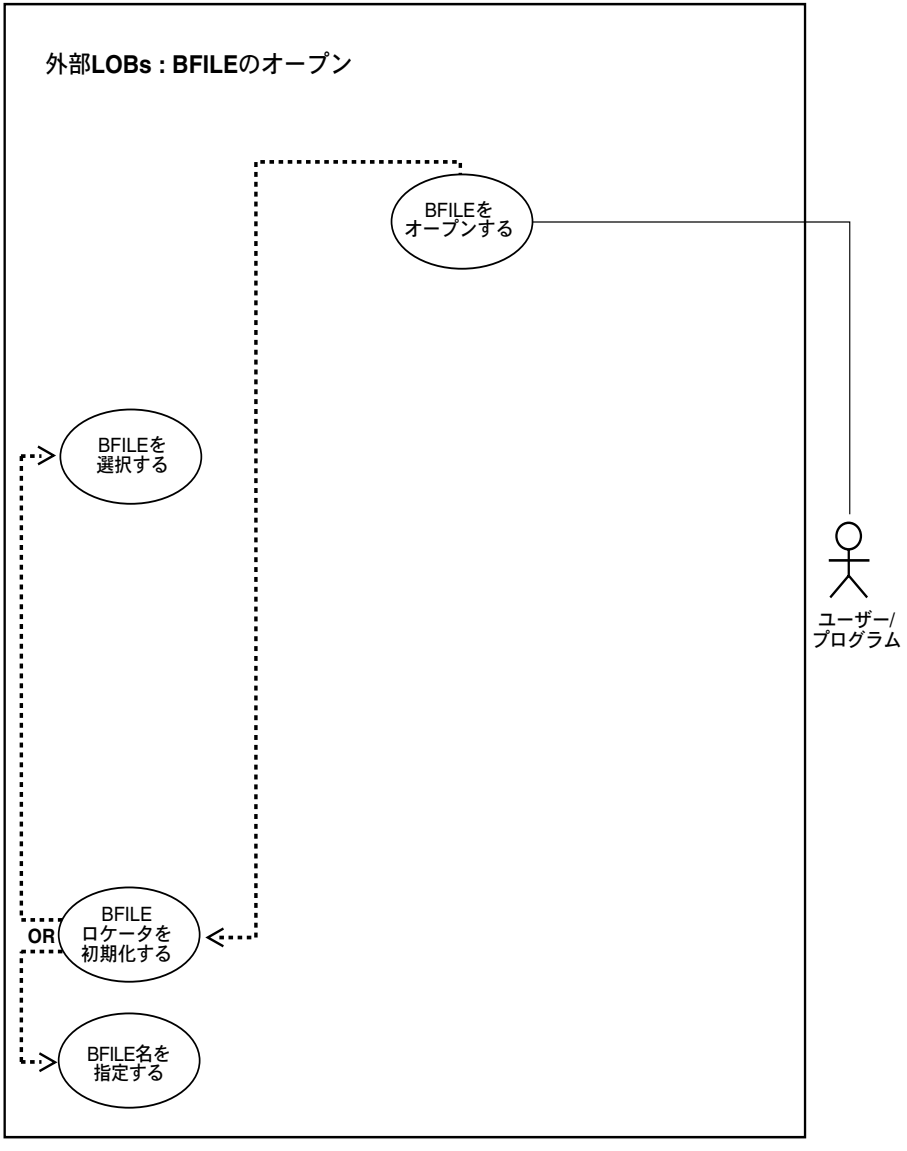
この例では、ad_source が UTF8 キャラクタ・セット形式の BFILE であり、データベース・キャラクタ・セットが UTF8 であることを想定しています。

```
DECLARE
  src_loc     bfile := bfilename('ADVERT_DIR','ad_source_1000') ;
  dst_loc     clob  ;
  amt         number := dbms_lob.lobmaxsize;
  src_offset  number := 1 ;
  dst_offset  number := 1 ;
  lang_ctx    number := dbms_lob.default_lang_ctx;
  warning     number;
BEGIN
  select ad_sourcetext into dst_loc from Print_media
     where product_id = 3000 and ad_id = 1000 for update ;
  dbms_lob.fileopen(src_loc, dbms_lob.file_readonly);

  /* The default_csid can be used when the BFILE encoding is in the same charset
   * as the destination CLOB/NCLOB charset
   */
  dbms_lob.LOADCLOBFROMFILE(dst_loc,src_loc, amt, dst_offset, src_offset,
       dbms_lob.default_csid, lang_ctx,warning) ;

  commit;
```

```
  dbms_output.put_line(' Amount specified ' || amt ) ;
  dbms_output.put_line(' Number of bytes read from source: ' ||
      (src_offset-1));
  dbms_output.put_line(' Number of characters written to destination: ' ||
      (dst_offset-1) );
  if (warning = dbms_lob.warn_inconvertible_char)
  then
    dbms_output.put_line('Warning: Inconvertible character');
  end if;

  dbms_lob.filecloseall() ;
END ;
```

BFILE をオープンする方法

図 12-11 利用図 : BFILE をオープンする方法



参照 :

- 内部一時 LOB に関するすべての基本操作については、12-2 ページの表 12-1「利用モデル図 : 外部 LOB（BFILE）」を参照してください。
- 12-65 ページの「FILEOPEN を使用した BFILE のオープン」を参照してください。
- 12-70 ページの「OPEN を使用した BFILE のオープン」を参照してください。

推奨項目 : OPEN を使用した BFILE のオープン

FILEOPEN を使用した BFILE のオープンはサポートされていますが、将来の拡張性が向上するため、OPEN の使用をお薦めします。

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、SESSION_MAX_OPEN_FILES 初期化パラメータを使用して指定します。

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。

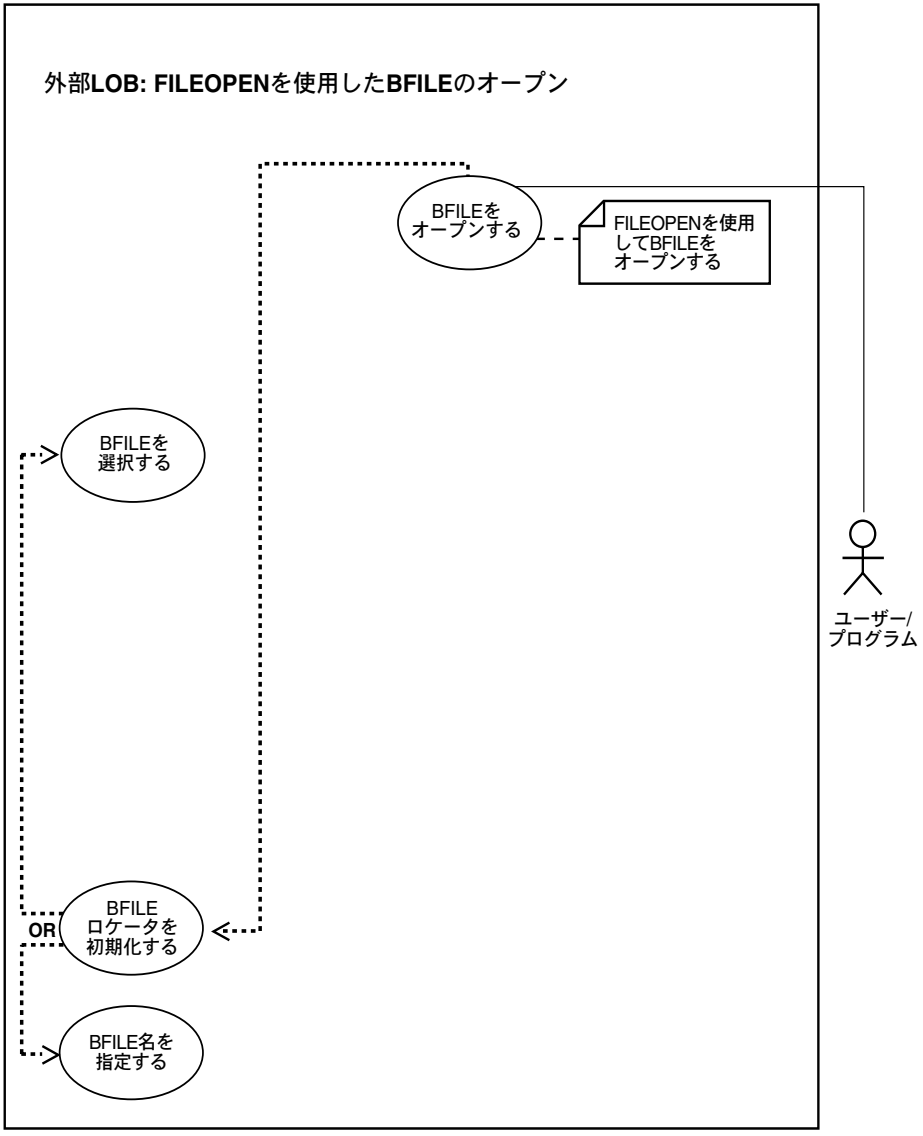
すべてのオープンしているファイルをクローズするには、FILECLOSEALL コールを使用します。

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薦めします。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 12-12 利用図 : FILEOPEN を使用した BFILE のオープン



参照 :

- 内部一時 LOB に関するすべての基本操作については、12-2 ページの表 12-1「利用モデル図 : 外部 LOB（BFILE）」を参照してください。
- 12-63 ページの「BFILE をオープンする方法」を参照してください。
- 12-70 ページの「OPEN を使用した BFILE のオープン」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薦めします。

構文

各プログラム環境で使用可能な関数またはファンクションのリストは、第 3 章「様々なプログラム環境での LOB のサポート」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）：『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』の第 23 章「DBMS_LOB」の「DBMS_LOB サブプログラムの要約」の「FILEOPEN プロシージャ」および「FILECLOSE プロシージャ」
- C（OCI）：『Oracle Call Interface 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 16 章「その他の OCI リレーショナル関数」の「LOB 関数」の「OCILobFileOpen()」、「OCILobFileClose()」、「OCILobFileSetName()」
- 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）：『Oracle9*i* JDBC 開発者ガイドおよびリファレンス』の第 8 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『Oracle9*i* 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「JDBC 2.0 LOB 型と Oracle 拡張型のサポート」の「BLOB、CLOB および BFILE のサポート」

使用例

次の例では、OS ファイル ADPHOTO_DIR で keyboard_photo3060 をオープンします。

例

次のプログラム環境での例を示します。

- PL/SQL（DBMS_LOB）: FILEOPEN を使用した BFILE のオープン（12-67 ページ）
- C（OCI）: FILEOPEN を使用した BFILE のオープン（12-67 ページ）
- COBOL（Pro*COBOL）: 今回のリリースでは例は提供されません。
- C/C++（Pro*C/C++）: 今回のリリースでは例は提供されません。
- Visual Basic（OO4O）: FILEOPEN を使用した BFILE のオープン（12-68 ページ）
- Java（JDBC）: FILEOPEN を使用した BFILE のオープン（12-68 ページ）

PL/SQL（DBMS_LOB）: FILEOPEN を使用した BFILE のオープン

```
/* Opening a BFILE with FILEOPEN  [Example script: 3973.sql] */
/* Procedure openBFILE_procOne is not part of DBMS_LOB package: */
CREATE OR REPLACE PROCEDURE openBFILE_procOne IS
   File_loc    BFILE := BFILENAME('ADPHOTO_DIR', 'keyboard_photo3060');
BEGIN
   /* Open the BFILE: */
   DBMS_LOB.FILEOPEN (File_loc, DBMS_LOB.FILE_READONLY);
   /* ... Do some processing. */
   DBMS_LOB.FILECLOSE(File_loc);
END;
```

C（OCI）: FILEOPEN を使用した BFILE のオープン

```
/* Opening a BFILE with FILEOPEN  */

void BfileOpen(envhp, errhp, svchp, stmthp)
OCIEnv  *envhp;
OCIError *errhp;
OCISvcCtx *svchp;
OCIStmt *stmthp;
{
     OCILobLocator *bfile_loc;

     /* Allocate the locator descriptor */
     (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &bfile_loc,
                               (ub4) OCI_DTYPE_FILE,
```

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

    /* Set the bfile locator information */
    checkerr(errhp, (OCILobFileSetName(envhp, errhp, &bfile_loc,
                                       (OraText *)"ADGRAPHIC_DIR",
                                       (ub2)strlen("ADGRAPHIC_DIR"),
                                       (OraText *)"keyboard_graphic_3106_13001",

(ub2)strlen("keyboard_graphic_3106_13001"))));
  checkerr(errhp, OCILobFileOpen(svchp, errhp, bfile_loc,
                                 (ub1)OCI_FILE_READONLY));
  /* ... Do some processing. */
  checkerr(errhp, OCILobFileClose(svchp, errhp, bfile_loc));

  /* Free the locator descriptor */
  OCIDescriptorFree((dvoid *)bfile_loc, (ub4)OCI_DTYPE_FILE);
}
```

Visual Basic（OO4O）: FILEOPEN を使用した BFILE のオープン

注意： 現時点では、OO4O では、OPEN を使用した BFILE のオープン（12-76 ページの「Visual Basic（OO4O）: OPEN を使用した BFILE のオープン」を参照）のみが提供されています。

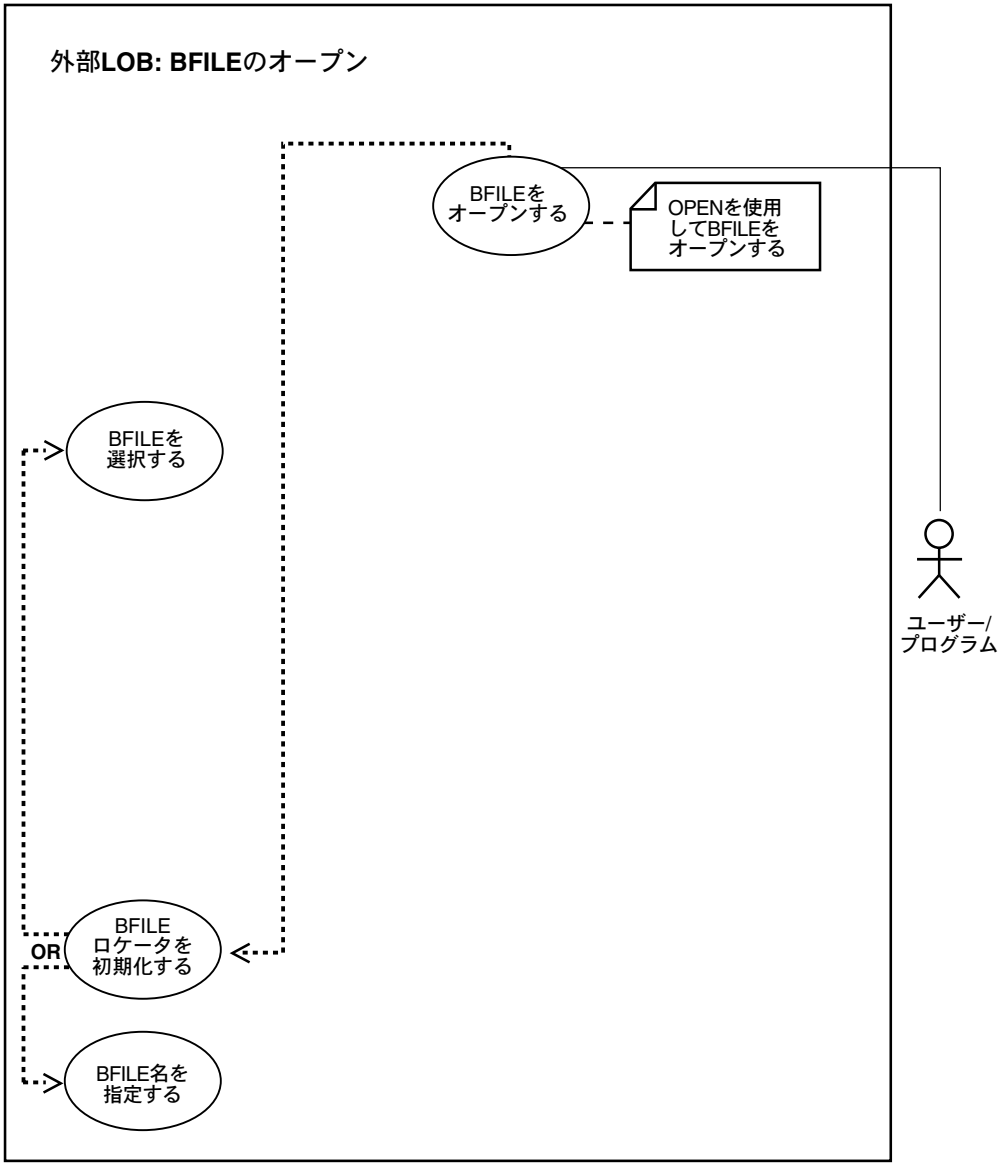
Java（JDBC）: FILEOPEN を使用した BFILE のオープン

```
// Opening a BFILE with FILEOPEN
import java.io.OutputStream;
// Core JDBC classes:
import java.sql.DriverManager;
import java.sql.Connection;
import java.sql.Statement;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

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

public class Ex4_38
{
```

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

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

    conn.setAutoCommit (false);

    // Create a Statement:
    Statement stmt = conn.createStatement ();
    try
    {
       BFILE src_lob = null;
       ResultSet rset = null;

       rset = stmt.executeQuery (
       "SELECT BFILENAME('AD_GRAPHIC', 'monitor_3060') FROM DUAL");
       if (rset.next())
       {
           src_lob = ((OracleResultSet)rset).getBFILE (1);

           src_lob.openFile();
          System.out.println("The file is now open");
        }

        // Close the BFILE, statement and connection:
        src_lob.closeFile();
        stmt.close();
        conn.commit();
        conn.close();
    }
    catch (SQLException e)
    {
        e.printStackTrace();
    }
  }
}
```

OPEN を使用した BFILE のオープン

図 12-13 利用図 : OPEN を使用した BFILE のオープン



参照：

- 内部一時 LOB に関するすべての基本操作については、12-2 ページの表 12-1「利用モデル図 : 外部 LOB（BFILE）」を参照してください。
- 12-63 ページの「BFILE をオープンする方法」を参照してください。
- 12-65 ページの「FILEOPEN を使用した BFILE のオープン」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関数またはファンクションのリストは、第 3 章「様々なプログラム環境での LOB のサポート」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）：『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』の第 23 章「DBMS_LOB」の「DBMS_LOB サブプログラムの要約」の「OPEN プロシージャ」
- C（OCI）：『Oracle Call Interface 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 16 章「その他の OCI リレーショナル関数」の「LOB 関数」の「OCILobOpen()」、「OCILobClose()」
- COBOL（Pro*COBOL）：『Pro*COBOL Precompiler プログラマーズ・ガイド』の LOB に関する詳細、LOB 文の使用上の注意および付録 F「埋込み SQL およびプリコンパイラ・ディレクティブ」の「LOB OPEN（実行可能埋込み SQL 拡張機能）」
- C/C++（Pro*C/C++）：『Pro*C/C++ Precompiler 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」および付録 F「埋込み SQL 文およびディレクティブ」の「LOB OPEN（実行可能埋込み SQL 拡張機能）」
- Visual Basic（OO4O オンライン・ヘルプ）：ヘルプの「目次」タブから、「OO4O オートメーション・サーバー・リファレンス」>「OO4O サーバーのオブジェクト」>「OraBFILE」>「メソッド」>「Open」、および「OO4O サーバーのオブジェクト」>「OraDynaset」>「メソッド」>「MoveFirst MoveLast MovePrevious MoveNext」を選択

- Java （JDBC）：『Oracle9*i* JDBC 開発者ガイドおよびリファレンス』の第 8 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『Oracle9*i* 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「JDBC 2.0 LOB 型と Oracle 拡張型のサポート」の「BLOB、CLOB および BFILE のサポート」

使用例

次の例では、OS ファイル ADPHOTO_DIR のイメージをオープンします。

例

次のプログラム環境での例を示します。

- PL/SQL（DBMS_LOB）: OPEN を使用した BFILE のオープン（12-72 ページ）
- C（OCI）: OPEN を使用した BFILE のオープン（12-73 ページ）
- C/C++（Pro*C/C++）: OPEN を使用した BFILE のオープン（12-75 ページ）
- COBOL（Pro*COBOL）: OPEN を使用した BFILE のオープン （12-73 ページ）
- Visual Basic（OO4O）: OPEN を使用した BFILE のオープン（12-76 ページ）
- Java（JDBC）: OPEN を使用した BFILE のオープン（12-76 ページ）

PL/SQL（DBMS_LOB）: OPEN を使用した BFILE のオープン

```
/* Opening a BFILE with OPEN.  */
/* Procedure openBFILE_procTwo is not part of DBMS_LOB package:  */
CREATE OR REPLACE PROCEDURE openBFILE_procTwo IS
  File_loc    BFILE := BFILENAME('ADPHOTO_DIR', 'keyboard_photo_3060_11001');
BEGIN
   /* Open the BFILE: */
   DBMS_LOB.OPEN (File_loc, DBMS_LOB.LOB_READONLY);
   /* ... Do some processing: */
   DBMS_LOB.CLOSE(File_loc);
END;
```

C（OCI）: OPEN を使用した BFILE のオープン

```
/* Opening a BFILE with OPEN.  */

void BfileFileOpen(envhp, errhp, svchp, stmthp)
OCIEnv       *envhp;
OCIError     *errhp;
OCISvcCtx    *svchp;
OCIStmt      *stmthp;
{
      OCILobLocator *bfile_loc;

     /* Allocate the locator descriptor */
     (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &bfile_loc,
                (ub4) OCI_DTYPE_FILE,
                (size_t) 0, (dvoid **) 0);

     /* Set the Bfile Locator Information */
     checkerr(errhp, (OCILobFileSetName(envhp, errhp, &bfile_loc,
                          (OraText *)"ADGRAPHIC_DIR", (ub2)strlen("ADGRAPHIC_DIR"),
                          (OraText *)"keyboard_graphic_3106_13001",
                          (ub2)strlen("keyboard_graphic_3106_13001"))));
     checkerr(errhp, OCILobOpen(svchp, errhp, bfile_loc,
                                (ub1)OCI_FILE_READONLY));
     /* ... Do some processing. */
     checkerr(errhp, OCILobClose(svchp, errhp, bfile_loc));

     /* Free the locator descriptor */
     OCIDescriptorFree((dvoid *)bfile_loc, (ub4)OCI_DTYPE_FILE);
}
```

COBOL（Pro*COBOL）: OPEN を使用した BFILE のオープン

```
* Opening a BFILE with OPEN.  [Example script: 3978.pco]
  IDENTIFICATION DIVISION.
  PROGRAM-ID. OPEN-BFILE.
  ENVIRONMENT DIVISION.
  DATA DIVISION.
  WORKING-STORAGE SECTION.
  01  USERID          PIC X(11) VALUES "SAMP/SAMP".
  01  SRC-BFILE       SQL-BFILE.
  01  DIR-ALIAS       PIC X(30) VARYING.
  01  FNAME           PIC X(20) VARYING.
  01  ORASLNRD        PIC 9(4).

      EXEC SQL INCLUDE SQLCA END-EXEC.
```

```
     EXEC ORACLE OPTION (ORACA=YES) END-EXEC.
     EXEC SQL INCLUDE ORACA END-EXEC.

 PROCEDURE DIVISION.
 OPEN-BFILE.

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

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

* Set up the directory and file information:
     MOVE "ADPHOTO_DIR" TO DIR-ALIAS-ARR.
     MOVE 9 TO DIR-ALIAS-LEN.
     MOVE "keyboard_photo_3106_13001" TO FNAME-ARR.
     MOVE 16 TO FNAME-LEN.

* Assign directory alias and file name to BFILE:
     EXEC SQL
          LOB FILE SET :SRC-BFILE
          DIRECTORY = :DIR-ALIAS, FILENAME = :FNAME END-EXEC.

* Open the BFILE read only:
     EXEC SQL LOB OPEN :SRC-BFILE READ ONLY END-EXEC.

* Close the LOB:
     EXEC SQL LOB CLOSE :SRC-BFILE END-EXEC.

* And free the LOB locator:
     EXEC SQL FREE :SRC-BFILE END-EXEC.
     EXEC SQL ROLLBACK WORK RELEASE END-EXEC.
     STOP RUN.

 SQL-ERROR.
     EXEC SQL WHENEVER SQLERROR CONTINUE END-EXEC.
     MOVE ORASLNR TO 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

```
/* Opening a BFILE using OPEN. [Example script: 3979.pc]
   In Pro*C/C++ there is only one form of OPEN used for OPENing
   BFILEs. There is no FILE OPEN, only a simple OPEN statement: */

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

void openBFILE_proc()
{
  OCIBFileLocator *Lob_loc;
  char *Dir = "GRAPHIC_DIR", *Name = "mousepad_2056";

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  /* Initialize the Locator: */
  EXEC SQL ALLOCATE :Lob_loc;
  EXEC SQL LOB FILE SET :Lob_loc DIRECTORY = :Dir, FILENAME = :Name;
  /* Open the BFILE: */
  EXEC SQL LOB OPEN :Lob_loc READ ONLY;
  /* ... Do some processing: */
  EXEC SQL LOB CLOSE :Lob_loc;
  EXEC SQL FREE :Lob_loc;
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  openBFILE_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

Visual Basic（OO4O）: OPEN を使用した BFILE のオープン

```
'Opening a BFILE using OPEN. [Example script: 3981.txt]
Dim OraDyn as OraDynaset, OraAdGraphic as OraBFile
Set OraDyn = OraDb.CreateDynaset("select * from Print_media",ORADYN_DEFAULT)
Set OraAdGraphic = OraDyn.Fields("ad_graphic").Value

'Go to the last row and open the Bfile for reading:
OraDyn.MoveLast
OraAdGraphic.Open 'Open Bfile for reading
'Do some processing:
OraAdGraphic.Close
```

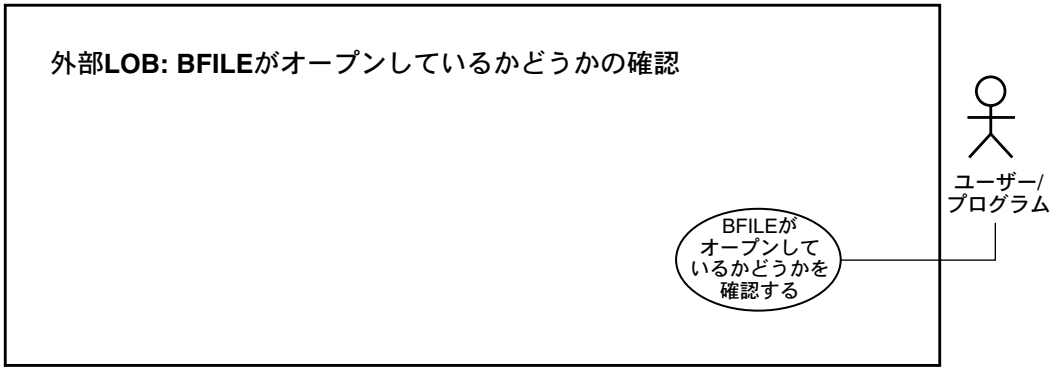
Java（JDBC）: OPEN を使用した BFILE のオープン

```
// Opening a BFILE with OPEN.  [Example script: 3982.java]
import java.io.InputStream;
import java.io.OutputStream;

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

// Oracle Specific JDBC classes:
import oracle.sql.*;
import oracle.jdbc.driver.*;
public class Ex4_41
{
  public static void main (String args [])
       throws Exception
  {
    // Load the Oracle JDBC driver:
    DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

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

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

```
    {
       BFILE src_lob = null;
       ResultSet rset = null;

       rset = stmt.executeQuery (
       "SELECT BFILENAME('ADGRAPHIC_DIR', 'monitor_graphic_3060_11001') FROM DUAL");
       if (rset.next())
       {
          src_lob = ((OracleResultSet)rset).getBFILE (1);

          OracleCallableStatement cstmt = (OracleCallableStatement)
          conn.prepareCall ("begin dbms_lob.open (?,dbms_lob.lob_readonly); end;");
          cstmt.registerOutParameter(1,OracleTypes.BFILE);
          cstmt.setBFILE (1, src_lob);
          cstmt.execute();
          src_lob = cstmt.getBFILE(1);
          System.out.println ("the file is now open");
       }

       // Close the BFILE,  statement and connection:
       src_lob.closeFile();
       stmt.close();
       conn.commit();
       conn.close();
    }
    catch (SQLException e)
    {
       e.printStackTrace();
    }
  }
}
```

BFILE がオープンしているかどうかを確認する方法

図 12-14 利用図 : BFILE がオープンしているかどうかを確認する 2 つの方法



参照 :

- 内部一時 LOB に関するすべての基本操作については、12-2 ページの表 12-1「利用モデル図 : 外部 LOB（BFILE）」を参照してください。
- 12-80 ページの「FILEISOPEN を使用した、BFILE がオープンしているかどうかの確認」を参照してください。
- 12-86 ページの「ISOPEN を使用した、BFILE がオープンしているかどうかの確認」を参照してください。

推奨項目 : 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 つの方法は、よく似ています。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薦めします。

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、SESSION_MAX_OPEN_FILES 初期化パラメータを使用して指定します。

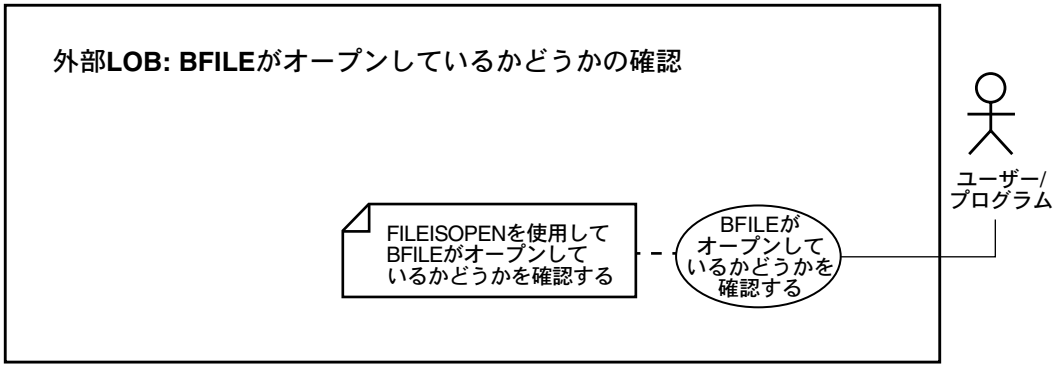
SESSION_MAX_OPEN_FILES は、1 回のセッションで同時にオープンできるファイル数の上限を定義します。このパラメータのデフォルト値は 10 です。デフォルト値を使用していれば、1 回のセッションにつき最大 10 ファイルを同時にオープンできます。データベース管理者は、init.ora ファイルでこのパラメータの値を変更できます。次に例を示します。

```
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 がオープンしているかどうかの確認

図 12-15 利用図 : FILEISOPEN を使用した、BFILE がオープンしているかどうかの確認



参照 :

- 内部一時 LOB に関するすべての基本操作については、12-2 ページの表 12-1「利用モデル図 : 外部 LOB（BFILE）」を参照してください。
- 12-78 ページの「BFILE がオープンしているかどうかを確認する方法」を参照してください。
- 12-86 ページの「ISOPEN を使用した、BFILE がオープンしているかどうかの確認」を参照してください。

用途

FILEISOPEN を使用して、BFILE 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

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薦めします。

構文

各プログラム環境で使用可能な関数またはファンクションのリストは、第 3 章「様々なプログラム環境での LOB のサポート」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）:『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』の第 23 章「DBMS_LOB」の「DBMS_LOB サブプログラムの要約」の「FILEISOPEN ファンクション」
- C（OCI）:『Oracle Call Interface 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 16 章「その他の 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）:『Oracle9*i* JDBC 開発者ガイドおよびリファレンス』の第 8 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『Oracle9*i* 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「JDBC 2.0 LOB 型と Oracle 拡張型のサポート」の「BLOB、CLOB および BFILE のサポート」

使用例

次の例では、`ad_graphic` 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せます。

例

次のプログラム環境での例を示します。

- PL/SQL（DBMS_LOB）: FILEISOPEN を使用した、BFILE がオープンしているかどうかの確認（12-82 ページ）
- C（OCI）: FILEISOPEN を使用した、BFILE がオープンしているかどうかの確認（12-82 ページ）
- C/C++（Pro*C/C++）: 今回のリリースでは例は提供されません。
- COBOL（Pro*COBOL）: 今回のリリースでは例は提供されません。
- Visual Basic（OO4O）: 例はありません。12-84 ページの「注意」を参照してください。
- Java（JDBC）: FILEISOPEN を使用した、BFILE がオープンしているかどうかの確認（12-84 ページ）

PL/SQL（DBMS_LOB）: FILEISOPEN を使用した、BFILE がオープンしているかどうかの確認

```
/* Checking if the BFILE is OPEN with FILEISOPEN. [Example script: 3984.sql]
   Procedure seeIfOpenBFILE_procOne is not part of DBMS_LOB package: */

CREATE OR REPLACE PROCEDURE seeIfOpenBFILE_procOne IS
   File_loc      BFILE;
   RetVal       INTEGER;
BEGIN
   /* Select the LOB, initializing the BFILE locator: */
   SELECT ad_graphic INTO File_loc FROM Print_media
      WHERE product_ID = 3060 AND ad_id = 11001;
   RetVal := DBMS_LOB.FILEISOPEN(File_loc);
   IF (RetVal = 1)
      THEN
      DBMS_OUTPUT.PUT_LINE('File is open');
   ELSE
      DBMS_OUTPUT.PUT_LINE('File is not open');
   END IF;
EXCEPTION
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('Operation failed');
END;
```

C（OCI）: FILEISOPEN を使用した、BFILE がオープンしているかどうかの確認

```
/* Checking if the BFILE is open with FILEISOPEN. [Example script: 3985.c] */

/* Select the lob/bfile from the Print_media table */
void selectLob(Lob_loc, errhp, svchp, stmthp)
OCILobLocator *Lob_loc;
OCIError      *errhp;
OCISvcCtx     *svchp;
OCIStmt       *stmthp;
{
     OCIDefine *dfnhp, *dfnhp2;
     text *selstmt =
             (text *) "SELECT ad_graphic FROM Print_media
                      WHERE product_id=3106 AND ad_id = 13001";

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

```
     /* Call define for the bfile column */
     checkerr (errhp, OCIDefineByPos(stmthp, &dfnhp, errhp, 1,
                                     (dvoid *)&Lob_loc, 0 , SQLT_BFILE,
                                     (dvoid *)0, (ub2 *)0, (ub2 *)0,
                                     OCI_DEFAULT)
     ||  OCIDefineByPos(stmthp, &dfnhp2, errhp, 2,
                                     (dvoid *)&Lob_loc, 0 , SQLT_BFILE,
                                     (dvoid *)0, (ub2 *)0, (ub2 *)0,
                                     OCI_DEFAULT));

     /* Execute the SQL select statement */
     checkerr (errhp, OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                                     (CONST OCISnapshot*) 0, (OCISnapshot*) 0,
                                     (ub4) OCI_DEFAULT));
}
void BfileFileIsOpen(envhp, errhp, svchp, stmthp)
OCIEnv  *envhp;
OCIError *errhp;
OCISvcCtx *svchp;
OCIStmt *stmthp;
{

     OCILobLocator *bfile_loc;
     boolean flag;

     /* Allocate the locator descriptor */
     (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &bfile_loc,
                (ub4) OCI_DTYPE_FILE,
                (size_t) 0, (dvoid **) 0);

     /* Select the bfile */
     selectLob(bfile_loc, errhp, svchp, stmthp);

     checkerr(errhp, OCILobFileOpen(svchp, errhp, bfile_loc,
                                    (ub1)OCI_FILE_READONLY));

     checkerr(errhp, OCILobFileIsOpen(svchp, errhp, bfile_loc, &flag));

     if (flag == TRUE)
     {
       printf("File is open\n");
     }
     else
     {
       printf("File is not open\n");
     }
```

```
    checkerr(errhp, OCILobFileClose(svchp, errhp, bfile_loc));

    /* Free the locator descriptor */
    OCIDescriptorFree((dvoid *)bfile_loc, (ub4)OCI_DTYPE_FILE);
}
```

Visual Basic（OO4O）: FILEISOPEN を使用した、BFILE がオープンしているかどうかの確認

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（12-84 ページの「Visual Basic（OO4O）: FILEISOPEN を使用した、BFILE がオープンしているかどうかの確認」を参照）。

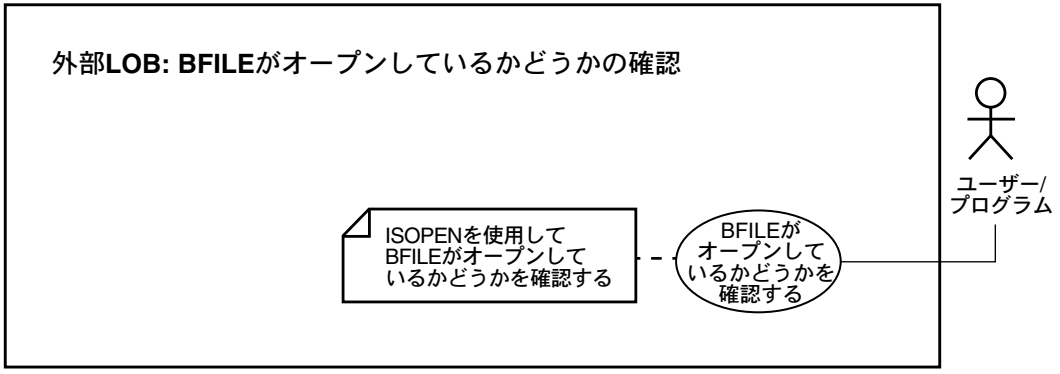
Java（JDBC）: FILEISOPEN を使用した、BFILE がオープンしているかどうかの確認

```
// Checking if the BFEIL is open with FILEISOPEN.[Example script:3986.java]

import java.io.InputStream;
import java.io.OutputStream;

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

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

public class Ex4_45
{

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

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

    conn.setAutoCommit (false);

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

    try
    {
        BFILE src_lob = null;
        ResultSet rset = null;
        boolean result = false;

        rset = stmt.executeQuery (
           "SELECT BFILENAME('ADGRAPHIC_DIR', 'monitor_graphic_3060_11001') FROM
DUAL");
        if (rset.next())
        {
           src_lob = ((OracleResultSet)rset).getBFILE (1);
        }

        result = src_lob.isFileOpen();
        System.out.println(
           "result of fileIsOpen() before opening file : " + result);
        if (!result)
        src_lob.openFile();

        result = src_lob.isFileOpen();
        System.out.println(
           "result of fileIsOpen() after opening file : " + result);

        // Close the BFILE, statement and connection:
        src_lob.closeFile();
        stmt.close();
        conn.commit();
        conn.close();
    }
    catch (SQLException e)
    {
       e.printStackTrace();
    }
  }
}
```

ISOPEN を使用した、BFILE がオープンしているかどうかの確認

図 12-16 利用図 : ISOPEN を使用した、BFILE がオープンしているかどうかの確認



参照 :

- 内部一時 LOB に関するすべての基本操作については、12-2 ページの表 12-1「利用モデル図 : 外部 LOB（BFILE）」を参照してください。
- 12-78 ページの「BFILE がオープンしているかどうかを確認する方法」を参照してください。
- 12-80 ページの「FILEISOPEN を使用した、BFILE がオープンしているかどうかの確認」を参照してください。

用途

ISOPEN を使用して、BFILE 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関数またはファンクションのリストは、第 3 章「様々なプログラム環境での LOB のサポート」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）:『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』の第 23 章「DBMS_LOB」の「DBMS_LOB サブプログラムの要約」の「ISOPEN ファンクション」

- C（OCI）：『Oracle Call Interface 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 16 章「その他の OCI リレーショナル関数」の「LOB 関数」の「OCILobFileIsOpen()」
- COBOL（Pro*COBOL）：『Pro*COBOL Precompiler プログラマーズ・ガイド』の LOB に関する詳細、LOB 文の使用上の注意および付録 F「埋込み SQL およびプリコンパイラ・ディレクティブ」の「LOB DESCRIBE（実行可能埋込み SQL 拡張機能）」
- C/C++（Pro*C/C++）：『Pro*C/C++ Precompiler 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」および付録 F「埋込み SQL 文およびディレクティブ」の「LOB DESCRIBE（実行可能埋込み SQL 拡張機能）」
- Visual Basic（OO4O オンライン・ヘルプ）：ヘルプの「目次」タブから、「OO4O オートメーション・サーバー・リファレンス」>「OO4O サーバーのオブジェクト」>「OraBFILE」>「プロパティ」>「IsOpen」、および「OO4O オブジェクト・サーバー・リファレンス」>「OO4O サーバーのオブジェクト」>「OraDynaset」を選択
- Java （JDBC）：『Oracle9*i* JDBC 開発者ガイドおよびリファレンス』の第 8 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『Oracle9*i* 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「JDBC 2.0 LOB 型と Oracle 拡張型のサポート」の「BLOB、CLOB および BFILE のサポート」

使用例

次の例では、`ad_graphic` 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せます。

例

次のプログラム環境での例を示します。

- PL/SQL（DBMS_LOB）: ISOPEN を使用した、BFILE がオープンしているかどうかの確認（12-88 ページ）
- C（OCI）: ISOPEN を使用した、BFILE がオープンしているかどうかの確認（12-88 ページ）
- COBOL（Pro*COBOL）: ISOPEN を使用した、BFILE がオープンしているかどうかの確認 （12-90 ページ）
- C/C++（Pro*C/C++）: ISOPEN を使用した、BFILE がオープンしているかどうかの確認（12-91 ページ）
- Visual Basic（OO4O）: ISOPEN を使用した、BFILE がオープンしているかどうかの確認（12-93 ページ）
- Java（JDBC）: ISOPEN を使用した、BFILE がオープンしているかどうかの確認（12-93 ページ）

PL/SQL（DBMS_LOB）: ISOPEN を使用した、BFILE がオープンしているかどうかの確認

```
/* Checking if the BFILE is open with ISOPEN. [Example script: 3987.sql] */
/* Procedure seeIfOpenBFILE_procTwo is not part of DBMS_LOB package: */

CREATE OR REPLACE PROCEDURE seeIfOpenBFILE_procTwo IS
   File_loc      BFILE;
   RetVal       INTEGER;
BEGIN
   /* Select the LOB, initializing the BFILE locator: */
   SELECT ad_graphic INTO File_loc FROM Print_media
      WHERE product_ID = 3060 AND ad_id = 11001;
   RetVal := DBMS_LOB.ISOPEN(File_loc);
   IF (RetVal = 1)
   THEN
      DBMS_OUTPUT.PUT_LINE('File is open');
   ELSE
      DBMS_OUTPUT.PUT_LINE('File is not open');
   END IF;
EXCEPTION
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('Operation failed');
END;
```

C（OCI）: ISOPEN を使用した、BFILE がオープンしているかどうかの確認

```
/* Checking if the BFILE is Open with ISOPEN. Example script:3988.c] */

/* Select the lob/bfile from the Print_media table */
void selectLob(Lob_loc, errhp, svchp, stmthp)
OCILobLocator *Lob_loc;
OCIError      *errhp;
OCISvcCtx     *svchp;
OCIStmt       *stmthp;
{
     OCIDefine *dfnhp, *dfnhp2;
     text *selstmt =
             (text *) "SELECT ad_graphic FROM Print_media
                         WHERE product_id=3106 AND ad_id = 13001";

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

```
     /* Call define for the bfile column */
     checkerr (errhp, OCIDefineByPos(stmthp, &dfnhp, errhp, 1,
                                     (dvoid *)&Lob_loc, 0 , SQLT_BFILE,
                                     (dvoid *)0, (ub2 *)0, (ub2 *)0,
                                     OCI_DEFAULT)
     ||  OCIDefineByPos(stmthp, &dfnhp2, errhp, 2,
                                     (dvoid *)&Lob_loc, 0 , SQLT_BFILE,
                                     (dvoid *)0, (ub2 *)0, (ub2 *)0,
                                     OCI_DEFAULT));
     /* Execute the SQL select statement */
     checkerr (errhp, OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                                     (CONST OCISnapshot*) 0, (OCISnapshot*) 0,
                                     (ub4) OCI_DEFAULT));
}
void BfileIsOpen(envhp, errhp, svchp, stmthp)
OCIEnv  *envhp;
OCIError *errhp;
OCISvcCtx *svchp;
OCIStmt *stmthp;
{

     OCILobLocator *bfile_loc;
     boolean flag;

     /* Allocate the locator descriptor */
     (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &bfile_loc,
                (ub4) OCI_DTYPE_FILE,
                (size_t) 0, (dvoid **) 0);

     /* Select the bfile */
     selectLob(bfile_loc, errhp, svchp, stmthp);

     checkerr(errhp, OCILobOpen(svchp, errhp, bfile_loc,
                                      (ub1)OCI_FILE_READONLY));

     checkerr(errhp, OCILobIsOpen(svchp, errhp, bfile_loc, &flag));

     if (flag == TRUE)
     {
       printf("File is open\n");
     }
     else
     {
       printf("File is not open\n");
     }
```

```
    checkerr(errhp, OCILobFileClose(svchp, errhp, bfile_loc));

    /* Free the locator descriptor */
    OCIDescriptorFree((dvoid *)bfile_loc, (ub4)OCI_DTYPE_FILE);
}
```

COBOL（Pro*COBOL）: ISOPEN を使用した、BFILE がオープンしているかどうかの確認

```
* Checking if BFILE is open with ISOPEN. [Example script: 3989.pco]
  IDENTIFICATION DIVISION.
  PROGRAM-ID. OPEN-BFILE.
  ENVIRONMENT DIVISION.
  DATA DIVISION.
  WORKING-STORAGE SECTION.
  01  USERID          PIC X(11) VALUES "SAMP/SAMP".
  01  SRC-BFILE       SQL-BFILE.
  01  DIR-ALIAS       PIC X(30) VARYING.
  01  FNAME           PIC X(20) VARYING.
  01  ORASLNRD        PIC 9(4).

      EXEC SQL INCLUDE SQLCA END-EXEC.
      EXEC ORACLE OPTION (ORACA=YES) END-EXEC.
      EXEC SQL INCLUDE ORACA END-EXEC.

  PROCEDURE DIVISION.
  OPEN-BFILE.

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

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

 * Set up the directory and file information:
      MOVE "ADPHOTO_DIR" TO DIR-ALIAS-ARR.
      MOVE 9 TO DIR-ALIAS-LEN.
      MOVE "keyboard_photo_3060_11001" TO FNAME-ARR.
      MOVE 16 TO FNAME-LEN.
```

```
* Assign directory alias and file name to BFILE:
     EXEC SQL
          LOB FILE SET :SRC-BFILE
          DIRECTORY = :DIR-ALIAS, FILENAME = :FNAME
     END-EXEC.

* Open the BFILE read only:
     EXEC SQL
          LOB OPEN :SRC-BFILE READ ONLY
     END-EXEC.

* Close the LOB:
     EXEC SQL LOB CLOSE :SRC-BFILE END-EXEC.

* And free the LOB locator:
     EXEC SQL FREE :SRC-BFILE END-EXEC.
     EXEC SQL
          ROLLBACK WORK RELEASE
     END-EXEC.
     STOP RUN.

 SQL-ERROR.
     EXEC SQL
         WHENEVER SQLERROR CONTINUE
     END-EXEC.
     MOVE ORASLNR TO 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 がオープンしているかどうかの確認

```
/* Checking if the BFILE is open with ISOPEN. [Example script: 3990.pc]
   In Pro*C/C++, there is only one form of ISOPEN to determine whether
   or not a BFILE is OPEN. There is no FILEISOPEN, only a simple ISOPEN.
   This is an attribute used in the DESCRIBE statement: */

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

```
#include <sqlca.h>

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

void seeIfOpenBFILE_proc()
{
  OCIBFileLocator *Lob_loc;
  int isOpen;

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  EXEC SQL ALLOCATE :Lob_loc;
  /* Select the BFILE into the locator: */
  EXEC SQL SELECT ad_graphic INTO :Lob_loc FROM Print_media
           WHERE product_id = 2056 AND ad_id = 12001;
  /* Determine if the BFILE is OPEN or not: */
  EXEC SQL LOB DESCRIBE :Lob_loc GET ISOPEN into :isOpen;
  if (isOpen)
    printf("BFILE is open\n");
  else
    printf("BFILE is not open\n");
  /* Note that in this example, the BFILE is not open: */
  EXEC SQL FREE :Lob_loc;
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  seeIfOpenBFILE_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

Visual Basic（OO4O）: ISOPEN を使用した、BFILE がオープンしているかどうかの確認

```
' Checking if the BFILE is open with ISOPEN. [Example script: 3992.txt]
Dim OraDyn as OraDynaset, OraAdGraphic as OraBFile, amount_read%, chunksize%, chunk

chunksize = 32767
Set OraDyn = OraDb.CreateDynaset("select * from Print_media", ORADYN_DEFAULT)
Set OraAdGraphic = OraDyn.Fields("ad_graphic").Value

If OraAdGraphic.IsOpen then
  'Process, if the file is already open:
Else
   'Process, if the file is not open, and return an error:
End If
```

Java（JDBC）: ISOPEN を使用した、BFILE がオープンしているかどうかの確認

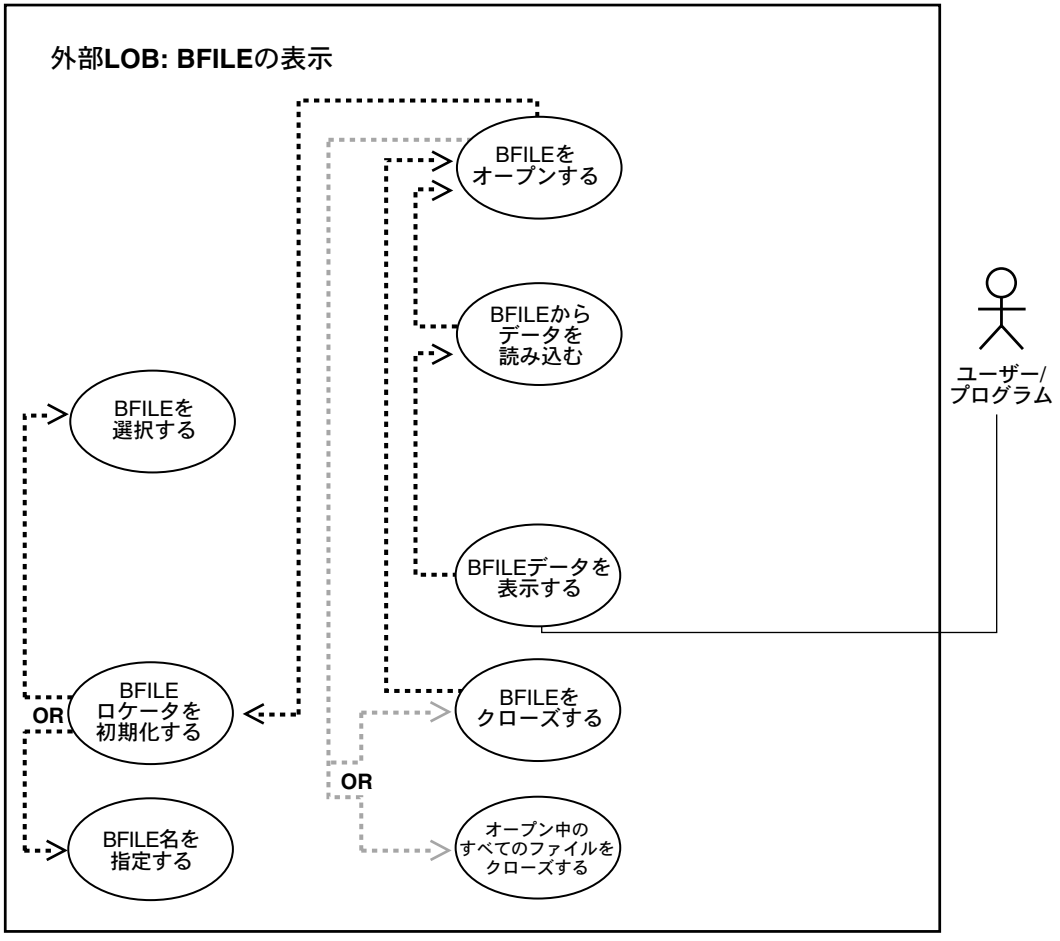
```
// Checking if the BFILE is open with ISOPEN. [Example script: 3993.java]

import java.io.InputStream;
import java.io.OutputStream;

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

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

public class Ex4_48
{
```

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

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

    // Create a Statement:
    Statement stmt = conn.createStatement ();
    try
    {
       BFILE src_lob = null;
       ResultSet rset = null;
       Boolean result = null;
       rset = stmt.executeQuery (
          "SELECT BFILENAME('ADGRAPHIC_DIR', 'monitor_graphic_3060_11001') FROM
DUAL");
       if (rset.next())
       {
          src_lob = ((OracleResultSet)rset).getBFILE (1);
       }
       result = new Boolean(src_lob.isFileOpen());
       System.out.println(
          "result of fileIsOpen() before opening file : " + result.toString());
       src_lob.openFile();
       result = new Boolean(src_lob.isFileOpen());
       System.out.println(
          "result of fileIsOpen() after opening file : " + result.toString());

       // Close the BFILE,  statement and connection:
       src_lob.closeFile();

       int i = cstmt.getInt(1);
       System.out.println("The result is: " + Integer.toString(i));

       OracleCallableStatement cstmt2 = (OracleCallableStatement)
       conn.prepareCall (
       "BEGIN DBMS_LOB.OPEN(?,DBMS_LOB.LOB_READONLY); END;");
       cstmt2.setBFILE(1, bfile);
       cstmt2.execute();
```

```
        System.out.println("The BFILE has been opened with a call to "
    +"DBMS_LOB.OPEN()");

        // Use the existing cstmt handle to re-query the status of the locator:
        cstmt.setBFILE(2, bfile);
        cstmt.execute();
        i = cstmt.getInt(1);
        System.out.println("This result is: " + Integer.toString(i));

        stmt.close();
        conn.commit();
        conn.close();
      }
      catch (SQLException e)
      {
        e.printStackTrace();
      }
    }
}
```

BFILE データの表示

図 12-17 利用図 : BFILE データの表示



参照 : 内部一時 LOB に関するすべての基本操作については、12-2 ページの表 12-1「利用モデル図 : 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関数またはファンクションのリストは、第 3 章「様々なプログラム環境での LOB のサポート」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）：『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』の第 23 章「DBMS_LOB」の「DBMS_LOB サブプログラムの要約」の「READ プロシージャ」および第 43 章「DBMS_OUTPUT」の「DBMS_OUTPUT 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）：『Oracle Call Interface 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 16 章「その他の OCI リレーショナル関数」の「LOB 関数」の「OCILobFileIsOpen()」と「OCILobRead()」
- COBOL（Pro*COBOL）：『Pro*COBOL Precompiler プログラマーズ・ガイド』の LOB に関する詳細、LOB 文の使用上の注意、付録 F「埋込み SQL およびプリコンパイラ・ディレクティブ」の「LOB READ（実行可能埋込み SQL 拡張機能）」
- C/C++（Pro*C/C++）：『Pro*C/C++ Precompiler 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（OO4O オンライン・ヘルプ）：ヘルプの「目次」タブから、「OO4O オートメーション・サーバー・リファレンス」>「OO4O サーバーのオブジェクト」>「OraBFILE」>「メソッド」>「Read」、「OO4O オートメーション・サーバー・リファレンス」>「OO4O サーバーのオブジェクト」>「OraBFILE」>「プロパティ」>「PollingAmount」、「Offset」と「Status」、および「OO4O オートメーション・サーバー・リファレンス」>「OO4O サーバーのオブジェクト」>「OraBFILE」>「例」を選択
- Java （JDBC）：『Oracle9*i* JDBC 開発者ガイドおよびリファレンス』の第 8 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『Oracle9*i* 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「JDBC 2.0 LOB 型と Oracle 拡張型のサポート」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での例を示します。

- PL/SQL（DBMS_LOB）: BFILE データの表示（12-98 ページ）
- C（OCI）: BFILE データの表示（12-99 ページ）
- COBOL（Pro*COBOL）: BFILE データの表示 （12-101 ページ）
- C/C++（Pro*C/C++）: BFILE データの表示（12-103 ページ）
- Visual Basic（OO4O）: BFILE データの表示（12-104 ページ）
- Java（JDBC）: BFILE データの表示（12-105 ページ）

PL/SQL（DBMS_LOB）: BFILE データの表示

```
/* Displaying BFILE data. [Example script: 3994.sql] */
/* Procedure displayBFILE_proc is not part of DBMS_LOB package: */

CREATE OR REPLACE PROCEDURE displayBFILE_proc IS
   File_loc  BFILE;
   Buffer   RAW(1024);
   Amount   BINARY_INTEGER := 1024;
   Position INTEGER        := 1;
BEGIN
   /* Select the LOB: */
   SELECT ad_graphic INTO File_loc
       FROM print_media WHERE Product_ID = 3060 AND ad_id = 11001;
   /* Opening the BFILE: */
   DBMS_LOB.OPEN (File_loc, DBMS_LOB.LOB_READONLY);
   LOOP
      DBMS_LOB.READ (File_loc, Amount, Position, Buffer);
      /* Display the buffer contents: */
      DBMS_OUTPUT.PUT_LINE(utl_raw.cast_to_varchar2(Buffer));
      Position := Position + Amount;
   END LOOP;
   /* Closing the BFILE: */
   DBMS_LOB.CLOSE (File_loc);
   EXCEPTION
   WHEN NO_DATA_FOUND THEN
      DBMS_OUTPUT.PUT_LINE('End of data');
END;
```

C（OCI）：BFILE データの表示

```
/* Displaying BFILE data. [Example script: 3995.c] */
/* Select the lob/bfile from the Print_media table */

void selectLob(Lob_loc, errhp, svchp, stmthp)
OCILobLocator *Lob_loc;
OCIError      *errhp;
OCISvcCtx     *svchp;
OCIStmt       *stmthp;
{
     OCIDefine *dfnhp, *dfnhp2;
     text *selstmt =
             (text *) "SELECT ad_graphic FROM Print_media
                         WHERE product_id=3106 AND ad_id = 13001";

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

     /* Call define for the bfile column */
     checkerr (errhp, OCIDefineByPos(stmthp, &dfnhp, errhp, 1,
                                     (dvoid *)&Lob_loc, 0 , SQLT_BFILE,
                                     (dvoid *)0, (ub2 *)0, (ub2 *)0,
                                     OCI_DEFAULT)
     || OCIDefineByPos(stmthp, &dfnhp2, errhp, 2,
                                     (dvoid *)&Lob_loc, 0 , SQLT_BFILE,
                                     (dvoid *)0, (ub2 *)0, (ub2 *)0,
                                     OCI_DEFAULT));

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

#define MAXBUFLEN 32767

void BfileDisplay(envhp, errhp, svchp, stmthp)
OCIEnv        *envhp;
OCIError      *errhp;
OCISvcCtx     *svchp;
OCIStmt       *stmthp;
{
   /* Assume all handles passed as input to this routine have been
      allocated and initialized */
```

```
OCILobLocator *bfile_loc;
ub1 bufp[MAXBUFLEN];
ub4 buflen, amt, offset;
boolean done;
ub4 retval;

/* Allocate the locator descriptor */
(void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &bfile_loc,
                          (ub4) OCI_DTYPE_FILE,
                          (size_t) 0, (dvoid **) 0);
  /* Select the bfile */
 selectLob(bfile_loc, errhp, svchp, stmthp);

 checkerr(errhp, OCILobFileOpen(svchp, errhp, bfile_loc,
                                OCI_FILE_READONLY));
 /* This example will READ the entire contents of a BFILE piecewise into a
    buffer using a standard polling method, processing each buffer piece
    after every READ operation until the entire BFILE has been read. */
 /* Setting amt = 0 will read till the end of LOB*/
 amt = 0;
 buflen = sizeof(bufp);
 /* Process the data in pieces */
 offset = 1;
 memset(bufp, '\0', MAXBUFLEN);
 done = FALSE;
 while (!done)
 {
     retval = OCILobRead(svchp, errhp, bfile_loc,
                         &amt, offset, (dvoid *) bufp,
                         buflen, (dvoid *)0,
                         (sb4 (*)(dvoid *, dvoid *, ub4, ub1)) 0,
                         (ub2) 0, (ub1) SQLCS_IMPLICIT);
     switch (retval)
     {
         case OCI_SUCCESS:             /* Only one piece or last piece*/
           /* process the data in bufp. amt will give the amount of data
              just read in bufp. This is in bytes for BLOBs and in characters
              for fixed  width CLOBS and in bytes for variable width CLOBs*/
           done = TRUE;
           break;
        case OCI_ERROR:
           /*  report_error();         this function is not shown here */
           done = TRUE;
           break;
        case OCI_NEED_DATA:           /* There are 2 or more pieces */
           /* process the data in bufp. amt will give the amount of
              data just read in bufp. This is in bytes for BFILEs and i
```

```
                characters for fixed width CLOBS and in bytes for variable
                width CLOBs */
              break;
            default:
              (void) printf("Unexpected ERROR: OCILobRead() LOB.\n");
              done = TRUE;
              break;
            }  /* switch */
        } /* while */

        /* Closing the BFILE is mandatory if you have opened it */
        checkerr (errhp, OCILobFileClose(svchp, errhp, bfile_loc));

        /* Free the locator descriptor */
        OCIDescriptorFree((dvoid *)bfile_loc, (ub4)OCI_DTYPE_FILE);
}
```

COBOL（Pro*COBOL）：BFILE データの表示

```
* Displaying BFILE data. [Example script: 3996.pco]
  IDENTIFICATION DIVISION.
  PROGRAM-ID.  DISPLAY-BFILE.
  ENVIRONMENT DIVISION.
  DATA DIVISION.
  WORKING-STORAGE SECTION.

  01  USERID         PIC X(9) VALUES "SAMP/SAMP".
      EXEC SQL BEGIN DECLARE SECTION END-EXEC.
  01  DEST-BLOB      SQL-BLOB.
  01  SRC-BFILE      SQL-BFILE.
  01  BUFFER         PIC X(5) VARYING.
  01  OFFSET  PIC S9(9) COMP VALUE 1.
  01  AMT            PIC S9(9) COMP.
  01  ORASLNRD       PIC  9(4).
      EXEC SQL END DECLARE SECTION END-EXEC.
  01  D-AMIPIC 99,999,99.
      EXEC SQL VAR BUFFER IS LONG RAW (100) END-EXEC.
      EXEC SQL INCLUDE SQLCA END-EXEC.
      EXEC ORACLE OPTION (ORACA=YES) END-EXEC.
      EXEC SQL INCLUDE ORACA END-EXEC.

  PROCEDURE DIVISION.
  DISPLAY-BFILE-DATA.

 * Connect to ORACLE
      EXEC SQL WHENEVER SQLERROR DO PERFORM SQL-ERROR END-EXEC.
      EXEC SQL
```

```
          CONNECT :USERID
     END-EXEC.

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

* Select the BFILE
     EXEC SQL SELECT AD_GRAPHIC INTO :SRC-BFILE
        FROM PRINT_MEDIA WHERE PRODUCT_ID = 3106 AND AD_ID = 13001
     END-EXEC.

* Open the BFILE
     EXEC SQL LOB OPEN :SRC-BFILE READ ONLY END-EXEC.

* Set the amount = 0 will initiate the polling method
     MOVE 0 TO AMT;
     EXEC SQL LOB READ :AMT FROM :SRC-BFILE INTO :BUFFER END-EXEC.

*    DISPLAY "BFILE DATA".
*    MOVE AMT TO D-AMT.
*    DISPLAY "First READ (", D-AMT, "): " BUFFER.

* Do READ-LOOP until the whole BFILE is read.
     EXEC SQL WHENEVER NOT FOUND GO TO END-LOOP END-EXEC.

 READ-LOOP.
     EXEC SQL LOB READ :AMT FROM :SRC-BFILE INTO :BUFFER END-EXEC.

*    MOVE AMT TO D-AMT.
*    DISPLAY "Next READ (", D-AMT, "): " BUFFER.

     GO TO READ-LOOP.

 END-LOOP.
     EXEC SQL WHENEVER NOT FOUND CONTINUE END-EXEC.

* Close the LOB
     EXEC SQL LOB CLOSE :SRC-BFILE END-EXEC.

* And free the LOB locator
     EXEC SQL FREE :SRC-BFILE END-EXEC.
     EXEC SQL ROLLBACK RELEASE END-EXEC.
     STOP RUN.

 SQL-ERROR.
     EXEC SQL WHENEVER SQLERROR CONTINUE END-EXEC.
     MOVE ORASLNR TO ORASLNRD.
```

```
           DISPLAY " ".
           DISPLAY "ORACLE ERROR DETECTED ON LINE ", ORASLNRD, ":".
           DISPLAY " ".
           DISPLAY SQLERRMC.
           EXEC SQL ROLLBACK WORK RELEASE END-EXEC.
           STOP RUN.
```

C/C++（Pro*C/C++）：BFILE データの表示

```
/* Displaying BFILE data. [Example script: 3997.pc]
   This example reads the entire contents of a BFILE piecewise into a
   buffer using a streaming mechanism via standard polling, displaying each
   buffer piece after every READ operation until the entire BFILE has been
   read: */

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

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

#define BufferLength 1024

void displayBFILE_proc()
{
  OCIBFileLocator *Lob_loc;
  int Amount;
  struct {
    short Length;
    char Data[BufferLength];
  } Buffer;
  /* Datatype Equivalencing is Mandatory for this Datatype: */
  EXEC SQL VAR Buffer is VARRAW(BufferLength);

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  EXEC SQL ALLOCATE :Lob_loc;
  /* Select the BFILE: */
  EXEC SQL SELECT ad_graphic INTO :Lob_loc
           FROM Print_media WHERE Product_ID = 2056 AND ad_id = 12001;
  /* Open the BFILE: */
  EXEC SQL LOB OPEN :Lob_loc READ ONLY;
```

```
  /* Setting Amount = 0 will initiate the polling method: */
  Amount = 0;
  /* Set the maximum size of the Buffer: */
  Buffer.Length = BufferLength;
  EXEC SQL WHENEVER NOT FOUND DO break;
  while (TRUE)
    {
      /* Read a piece of the BFILE into the Buffer: */
      EXEC SQL LOB READ :Amount FROM :Lob_loc INTO :Buffer;
      printf("Display %d bytes\n", Buffer.Length);
    }
  printf("Display %d bytes\n", Amount);
  EXEC SQL LOB CLOSE :Lob_loc;
  EXEC SQL FREE :Lob_loc;
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  displayBFILE_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

Visual Basic（OO4O）: BFILE データの表示

```
' Displaying BFILE data.  [Example script: 3999.txt]
Dim MySession As OraSession
Dim OraDb As OraDatabase

Dim OraDyn As OraDynaset, OraAdGraphio As OraBfile, amount_read%, chunksize%, chunk
As Variant

Set MySession = CreateObject("OracleInProcServer.XOraSession")
Set OraDb = MySession.OpenDatabase("pmschema", "pm/pm", 0&)

chunksize = 32767
Set OraDyn = OraDb.CreateDynaset("select * from Print_media", ORADYN_DEFAULT)
Set OraAdGraphic = OraDyn.Fields("ad_graphic").Value

OraAdGraphic.offset = 1
OraAdGraphic.PollingAmount = OraAdGraphic.Size 'Read entire BFILE contents

'Open the Bfile for reading:
OraAdGraphic.Open
amount_read = OraAdGraphic.Read(chunk, chunksize)
```

```
While OraAdGraphic.Status = ORALOB_NEED_DATA
    amount_read = OraAdGraphic.Read(chunk, chunksize)
Wend
OraAdGraphic.Close
```

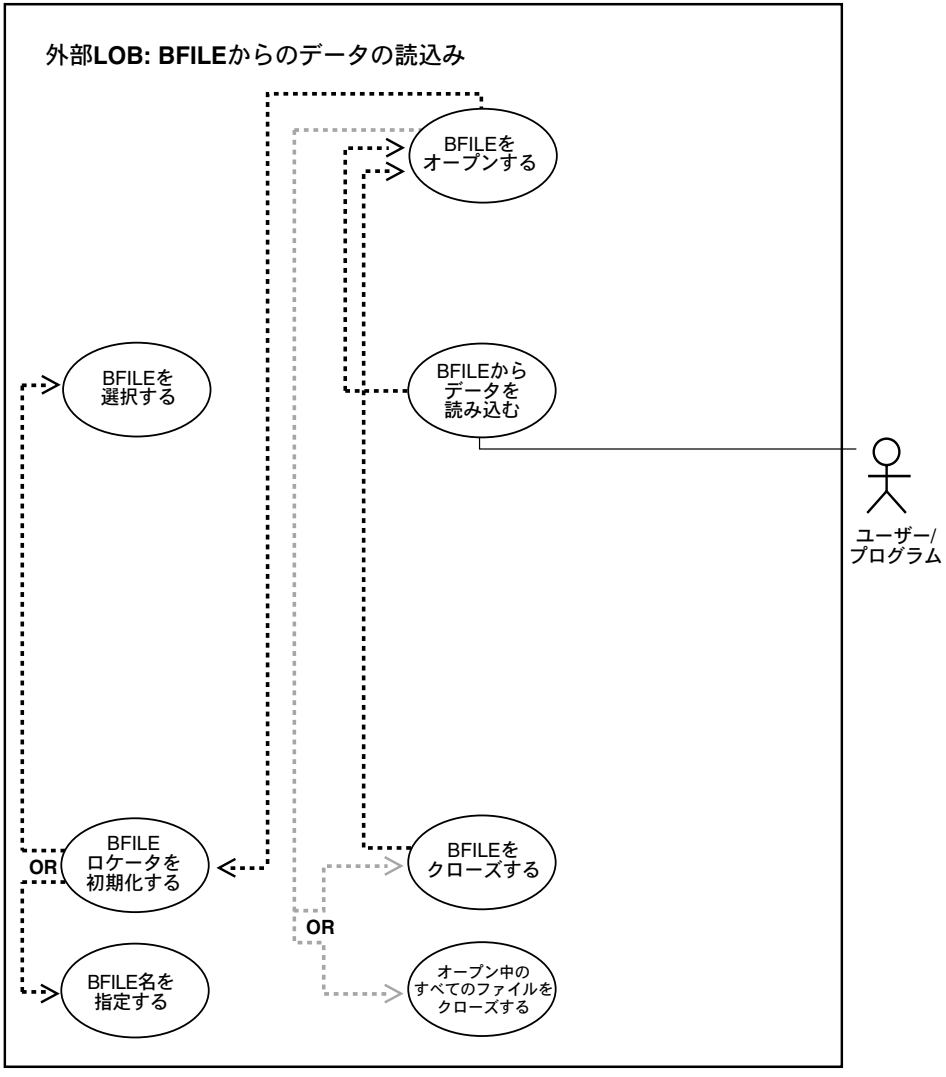
Java（JDBC）：BFILE データの表示

```
// Displaying BFILE data. [Example script: 4000.java]

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

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

public class Ex4_53
{

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

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

    conn.setAutoCommit (false);

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

    try
    {
       BFILE src_lob = null;
       ResultSet rset = null;
       Boolean result = null;
```

```
        InputStream in = null;
        byte buf[] = new byte[1000];
        int length = 0;
        boolean alreadyDisplayed = false;
        rset = stmt.executeQuery (
           "SELECT ad_graphic FROM Print_media
               WHERE product_id = 3106 AND ad_id = 13001");
        if (rset.next())
        {
           src_lob = ((OracleResultSet)rset).getBFILE (1);
        }

        // Open the BFILE:
        src_lob.openFile();

        // Get a handle to stream the data from the BFILE:
        in = src_lob.getBinaryStream();

        // This loop fills the buf iteratively, retrieving data
        // from the InputStream:
        while ((in != null) && ((length = in.read(buf)) != -1))
        {
           // the data has already been read into buf

          // We will only display the first CHUNK in this example:
          if (! alreadyDisplayed)
          {
             System.out.println("Bytes read in: " + Integer.toString(length));
             System.out.println(new String(buf));
             alreadyDisplayed = true;
          }
        }

        // Close the stream, BFILE, statement and connection:
        in.close();
        src_lob.closeFile();
        stmt.close();
        conn.commit();
        conn.close();
      }
      catch (SQLException e)
      {
         e.printStackTrace();
      }
    }
  }
```

BFILE からのデータの読込み

図 12-18 利用図 : BFILE からのデータの読込み



参照： 内部一時 LOB に関するすべての基本操作については、12-2 ページの表 12-1「利用モデル図 : 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 値を読み込む場合、LOB の終わり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

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。次に、パラメータの初期化を除いた OCI 読込みコールを示します。

```
#define MAX_LOB_SIZE 4294967295
ub4  amount =  MAX_LOB_SIZE;
ub4  offset = 1000;
OCILobRead(svchp, errhp, locp, &amount, offset, bufp, bufl, 0, 0, 0, 0)
```

注意： 大量の LOB データを最も効率よく読み込むには、ポーリングまたはコールバックによってストリーム・メカニズムを有効にした OCILobRead() を使用します。

参照： 第 10 章「内部永続 LOB」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、量パラメータをデータのサイズより大きく指定できます。PL/SQL では、量パラメータをバッファのサイズ以下に指定する必要があります。また、バッファ・サイズは 32KB 以下に制限されます。
- **OCILobRead** では、量パラメータの値を 4GB -1 に指定し、LOB の終わり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関数またはファンクションのリストは、第 3 章「様々なプログラム環境での LOB のサポート」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）：『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』の第 23 章「DBMS_LOB」の「DBMS_LOB サブプログラムの要約」の「READ プロシージャ」
- C（OCI）：『Oracle Call Interface 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 16 章「その他の OCI リレーショナル関数」の「LOB 関数」の「OCILobRead()」
- COBOL（Pro*COBOL）：『Pro*COBOL Precompiler プログラマーズ・ガイド』の LOB に関する詳細、LOB 文の使用上の注意および付録 F「埋込み SQL およびプリコンパイラ・ディレクティブ」の「LOB READ（実行可能埋込み SQL 拡張機能）」
- C/C++（Pro*C/C++）：『Pro*C/C++ Precompiler 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」および付録 F「埋込み SQL 文およびディレクティブ」の「LOB READ（実行可能埋込み SQL 拡張機能）」
- Visual Basic（OO4O オンライン・ヘルプ）：ヘルプの「目次」タブから、「OO4O オートメーション・サーバー・リファレンス」>「OO4O サーバーのオブジェクト」>「OraBFILE」>「メソッド」>「Read」、「OO4O オートメーション・サーバー・リファレンス」>「OO4O サーバーのオブジェクト」>「OraBFILE」>「プロパティ」>「PollingAmount」、「Offset」と「Status」、および「OO4O オートメーション・サーバー・リファレンス」>「OO4O サーバーのオブジェクト」>「OraBFILE」>「例」を選択
- Java （JDBC）：『Oracle9*i* JDBC 開発者ガイドおよびリファレンス』の第 8 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『Oracle9*i* 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「JDBC 2.0 LOB 型と Oracle 拡張型のサポート」の「BLOB、CLOB および BFILE のサポート」

使用例

次の例では、BFILE「ADPHOTO_DIR」から ad_graphic に写真を読み込みます。

例

次のプログラム環境での例を示します。

- PL/SQL（DBMS_LOB）: BFILE からのデータの読込み（12-110 ページ）
- C（OCI）: BFILE からのデータの読込み（12-110 ページ）
- COBOL（Pro*COBOL）: BFILE からのデータの読込み （12-112 ページ）
- C/C++（Pro*C/C++）: BFILE からのデータの読込み（12-113 ページ）

PL/SQL（DBMS_LOB）: BFILE からのデータの読込み

```
/* Reading data from a BFILE. [Example script: 4002.sql]     */
/* Procedure readBFILE_proc is not part of DBMS_LOB package: */

CREATE OR REPLACE PROCEDURE readBFILE_proc IS
   File_loc       BFILE := BFILENAME('ADPHOTO_DIR',
       'keyboard_photo_3060_11001');
   Amount         INTEGER := 32767;
   Position       INTEGER := 1;
   Buffer         RAW(32767);
BEGIN
   /* Select the LOB: */
   SELECT ad_graphic INTO File_loc FROM print_media
      WHERE Product_ID = 3060 AND ad-Id = 11001;
   /* Open the BFILE: */
   DBMS_LOB.OPEN(File_loc, DBMS_LOB.LOB_READONLY);
   /* Read data: */
   DBMS_LOB.READ(File_loc, Amount, Position, Buffer);
   /* Close the BFILE: */
   DBMS_LOB.CLOSE(File_loc);
END;
```

C（OCI）: BFILE からのデータの読込み

```
/* Reading data from a BFILE. [Example script: 4003.c] */

/* Select the lob/bfile from the Print_media table */
void selectLob(Lob_loc, errhp, svchp, stmthp)
OCILobLocator *Lob_loc;
OCIError      *errhp;
OCISvcCtx     *svchp;
OCIStmt       *stmthp;
{
     OCIDefine *dfnhp, *dfnhp2;
     text *selstmt =
             (text *) "SELECT ad_graphic FROM Print_media
                       WHERE product_id=3106 AND ad_id = 13001";

     /* Prepare the SQL select statement */
     checkerr (errhp, OCIStmtPrepare(stmthp, errhp, selstmt,
                                     (ub4) strlen((char *) selstmt),
```

```
                                    (ub4) OCI_NTV_SYNTAX, (ub4)OCI_DEFAULT));

     /* Call define for the bfile column */
     checkerr (errhp, OCIDefineByPos(stmthp, &dfnhp, errhp, 1,
                                    (dvoid *)&Lob_loc, 0 , SQLT_BFILE,
                                    (dvoid *)0, (ub2 *)0, (ub2 *)0,
                                    OCI_DEFAULT)
     || OCIDefineByPos(stmthp, &dfnhp2, errhp, 2,
                                    (dvoid *)&Lob_loc, 0 , SQLT_BFILE,
                                    (dvoid *)0, (ub2 *)0, (ub2 *)0,
                                    OCI_DEFAULT));

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

#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;

   /* Allocate the locator descriptor */
   (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &bfile_loc,
                             (ub4) OCI_DTYPE_FILE,
                             (size_t) 0, (dvoid **) 0);
     /* Select the bfile */
    selectLob(bfile_loc, errhp, svchp, stmthp);

    checkerr(errhp, OCILobFileOpen(svchp, errhp, bfile_loc,
                                   OCI_FILE_READONLY));

    /* This example will READ the entire contents of a BFILE piecewise into a
       buffer using a standard polling method, processing each buffer piece
       after every READ operation until the entire BFILE has been read. */
    /* Setting amt = 0 will read till the end of LOB*/
    amt = 0;
```

```
    buflen = sizeof(bufp);
    /* Process the data in pieces */
    offset = 1;
    memset(bufp, '\0', MAXBUFLEN);

    retval = OCILobRead(svchp, errhp, bfile_loc,
                        &amt, offset, (dvoid *) bufp,
                        buflen, (dvoid *)0,
                        (sb4 (*)(dvoid *, dvoid *, ub4, ub1)) 0,
                        (ub2) 0, (ub1) SQLCS_IMPLICIT);

    /* Closing the BFILE is mandatory if you have opened it */
    checkerr (errhp, OCILobFileClose(svchp, errhp, bfile_loc));

    /* Free the locator descriptor */
    OCIDescriptorFree((dvoid *)bfile_loc, (ub4)OCI_DTYPE_FILE);
}
```

COBOL（Pro*COBOL）: BFILE からのデータの読込み

```
 * Reading data from a BFILE.  [Example script: 4004.pco]
   IDENTIFICATION DIVISION.
   PROGRAM-ID. READ-BFILE.
   ENVIRONMENT DIVISION.
   DATA DIVISION.
   WORKING-STORAGE SECTION.
   01  BFILE1         SQL-BFILE.
   01  BUFFER2        PIC X(5) VARYING.
   01  AMT            PIC S9(9) COMP.
   01  OFFSET         PIC S9(9) COMP VALUE 1.

       EXEC SQL INCLUDE SQLCA END-EXEC.
       EXEC SQL VAR BUFFER2 IS LONG RAW(5) END-EXEC.

   PROCEDURE DIVISION.
   READ-BFILE.

  * Allocate and initialize the CLOB locator
       EXEC SQL ALLOCATE :BFILE1 END-EXEC.
       EXEC SQL WHENEVER NOT FOUND GOTO END-OF-BFILE END-EXEC.
       EXEC SQL
            SELECT AD_GRAPHIC INTO :BFILE1
            FROM PRINT_MEDIA M WHERE M.PRODUCT_ID = 3106 AND AD_ID = 13001
       END-EXEC.
  * Open the BFILE
       EXEC SQL LOB OPEN :BFILE1 READ ONLY END-EXEC.
```

```
      * Initiate polling read
           MOVE 0 TO AMT.

           EXEC SQL LOB READ :AMT FROM :BFILE1
                INTO :BUFFER2 END-EXEC.
      *
      *     Display the data here.
      *

      * Close and free the locator
           EXEC SQL WHENEVER NOT FOUND CONTINUE END-EXEC.
           EXEC SQL LOB CLOSE :BFILE1 END-EXEC.

       END-OF-BFILE.
           EXEC SQL WHENEVER NOT FOUND CONTINUE END-EXEC.
           EXEC SQL FREE :BFILE1 END-EXEC.
```

C/C++（Pro*C/C++）: BFILE からのデータの読込み

```
/* Reading data from BFILE. [Example script: 4005.pc] */

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

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

#define BufferLength 4096

void readBFILE_proc()
{
  OCIBFileLocator *Lob_loc;
  /* Amount and BufferLength are equal so only one READ is necessary: */
  int Amount = BufferLength;
  char Buffer[BufferLength];
  /* Datatype Equivalencing is Mandatory for this Datatype: */
  EXEC SQL VAR Buffer IS RAW(BufferLength);

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
```

```
  EXEC SQL ALLOCATE :Lob_loc;
  EXEC SQL SELECT ad_graphic INTO :Lob_loc
           FROM Print_media WHERE Product_ID = 2056;
  /* Open the BFILE: */
  EXEC SQL LOB OPEN :Lob_loc READ ONLY;
  EXEC SQL WHENEVER NOT FOUND CONTINUE;
  /* Read data: */
  EXEC SQL LOB READ :Amount FROM :Lob_loc INTO :Buffer;
  printf("Read %d bytes\n", Amount);
  /* Close the BFILE: */
  EXEC SQL LOB CLOSE :Lob_loc;
  EXEC SQL FREE :Lob_loc;
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  readBFILE_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

Visual Basic（OO4O）: BFILE からのデータの読込み

```
' Reading data from a BFILE [Example script: 4007.txt]

Dim MySession As OraSession
Dim OraDb As OraDatabase

Dim OraDyn As OraDynaset, OraAdGraphic As OraBfile, amount_read%, chunksize%, chunk
As Variant

Set MySession = CreateObject("OracleInProcServer.XOraSession")
Set OraDb = MySession.OpenDatabase("pmschema", "pm/pm", 0&)

chunksize = 32767
Set OraDyn = OraDb.CreateDynaset("select * from Print_media", ORADYN_DEFAULT)
Set OraAdGraphic = OraDyn.Fields("ad_graphic").Value

OraAdGraphic.offset = 1
OraAdGraphic.PollingAmount = OraAdGraphic.Size 'Read entire BFILE contents

'Open the Bfile for reading:
OraAdGraphic.Open
amount_read = OraAdGraphic.Read(chunk, chunksize)
While OraAdGraphic.Status = ORALOB_NEED_DATA
```

```
    amount_read = OraAdGraphic.Read(chunk, chunksize)
Wend
OraAdGraphic.Close
```

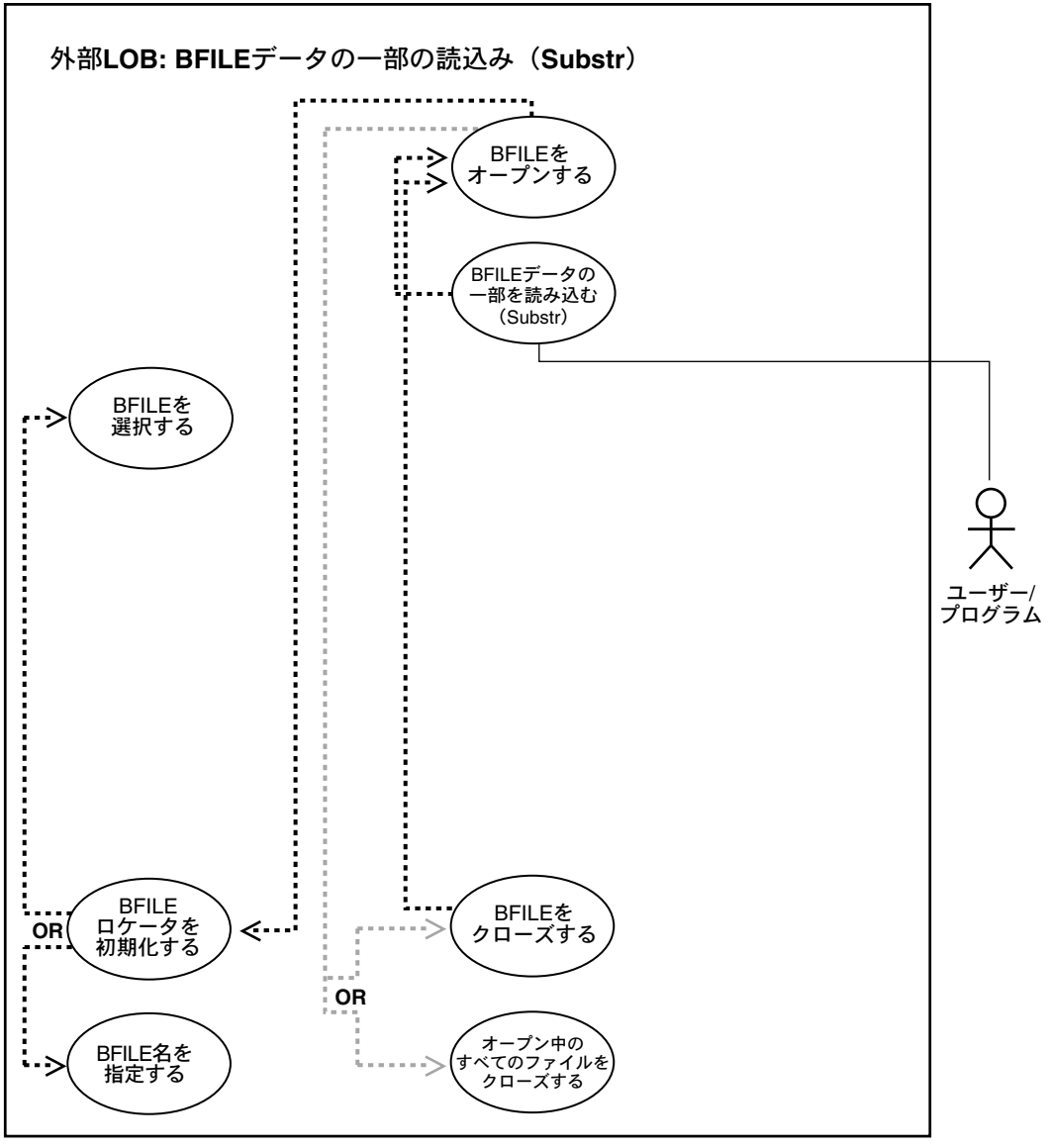
Java（JDBC）: BFILE からのデータの読込み

```
// Reading data from a BFILE.  [Example script: 4008.java]

import java.io.InputStream;
import java.io.OutputStream;

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

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

public class Ex4_53
{
  public static void main (String args [])
       throws Exception
  {
    // Load the Oracle JDBC driver:
    DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

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

    // Create a Statement
    Statement stmt = conn.createStatement ();
    try
    {
       BFILE src_lob = null;
       ResultSet rset = null;
       Boolean result = null;
       InputStream in = null;
       byte buf[] = new byte[1000];
       int length = 0;
```

```
        boolean alreadyDisplayed = false;
        rset = stmt.executeQuery (
           "SELECT ad_graphic FROM Print_media
                WHERE product_id = 3106 AND ad_id = 13001");
        if (rset.next())
        {
           src_lob = ((OracleResultSet)rset).getBFILE (1);
        }

        // Open the BFILE:
        src_lob.openFile();

        // Get a handle to stream the data from the BFILE:
        in = src_lob.getBinaryStream();

        // This loop fills the buf iteratively, retrieving data
        // from the InputStream:
        while ((in != null) && ((length = in.read(buf)) != -1))
        {
           // the data has already been read into buf

           // We will only display the first CHUNK in this example:
           if (! alreadyDisplayed)
           {
              System.out.println("Bytes read in: " + Integer.toString(length));
              System.out.println(new String(buf));
              alreadyDisplayed = true;
           }
        }

        // Close the stream, BFILE, statement and connection:
        in.close();
        src_lob.closeFile();
        stmt.close();
        conn.commit();
        conn.close();
      }
      catch (SQLException e)
      {
          e.printStackTrace();
      }
    }
  }
```

BFILE データの一部の読込み（substr）

図 12-19 利用図 : BFILE データの一部の読込み（substr）



参照： 内部一時 LOB に関するすべての基本操作については、12-2 ページの表 12-1「利用モデル図 : 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関数またはファンクションのリストは、第 3 章「様々なプログラム環境での LOB のサポート」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）：『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』の第 23 章「DBMS_LOB」の「DBMS_LOB 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）：『Pro*COBOL Precompiler プログラマーズ・ガイド』の LOB に関する詳細、LOB 文の使用上の注意、付録 F「埋込み SQL およびプリコンパイラ・ディレクティブ」の「LOB OPEN（実行可能埋込み SQL 拡張機能）」、「LOB CLOSE（実行可能埋込み SQL 拡張機能）」、および『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』の第 23 章「DBMS_LOB」の「DBMS_LOB サブプログラムの要約」の「SUBSTR ファンクション」
- C/C++（Pro*C/C++）：『Pro*C/C++ Precompiler 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」、付録 F「埋込み SQL 文およびディレクティブ」の「LOB OPEN（実行可能埋込み SQL 拡張機能）」、および『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』の第 23 章「DBMS_LOB」の「DBMS_LOB サブプログラムの要約」の「SUBSTR ファンクション」
- Visual Basic（OO4O オンライン・ヘルプ）: ヘルプの「目次」タブから、「OO4O オートメーション・サーバー・リファレンス」>「OO4O サーバーのオブジェクト」>「OraBFILE」>「メソッド」>「Open」、「OO4O オートメーション・サーバー・リファレンス」>「OO4O サーバーのオブジェクト」>「OraBFILE」>「プロパティ」>「PollingAmount」、「Offset」と「Status」、および「OO4O オートメーション・サーバー・リファレンス」>「OO4O サーバーのオブジェクト」>「OraBFILE」>「例」を選択

- Java（JDBC）：『Oracle9*i* JDBC 開発者ガイドおよびリファレンス』の第 8 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『Oracle9*i* 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「JDBC 2.0 LOB 型と Oracle 拡張型のサポート」の「BLOB、CLOB および BFILE のサポート」

使用例

次の例では、BFILE「ADPHOTO_DIR」から ad_graphic に図形イメージを読み込みます。

例

次のプログラム環境での例を示します。

- PL/SQL（DBMS_LOB）: BFILE データの一部の読込み（substr）（12-119 ページ）
- C（OCI）: 今回のリリースでは例は提供されません。
- COBOL（Pro*COBOL）: BFILE データの一部の読込み（substr）（12-120 ページ）
- C/C++（Pro*C/C++）: BFILE データの一部の読込み（substr）（12-121 ページ）
- Visual Basic（OO4O）: BFILE データの一部の読込み（substr）（12-122 ページ）
- Java（JDBC）: BFILE データの一部の読込み（substr）（12-123 ページ）

PL/SQL（DBMS_LOB）: BFILE データの一部の読込み（substr）

```
/* Reading portion of a BFILE data using substr. [Example script: 4009.sql] */
/* Procedure substringBFILE_proc is not part of DBMS_LOB package:  */

CREATE OR REPLACE PROCEDURE substringBFILE_proc IS
   File_loc        BFILE;
   Position        INTEGER := 1;
   Buffer          RAW(32767);
BEGIN
   /* Select the LOB: */
   SELECT PMtab.ad_graphic INTO File_loc FROM Print_media PMtab
      WHERE PMtab.product_id = 3060 AND PMtab.ad_id = 11001;
   /* Open the BFILE: */
   DBMS_LOB.OPEN(File_loc, DBMS_LOB.LOB_READONLY);
   Buffer := DBMS_LOB.SUBSTR(File_loc, 255, Position);
   /* Close the BFILE: */
   DBMS_LOB.CLOSE(File_loc);
END;
```

COBOL（Pro*COBOL）: BFILE データの一部の読込み（substr）

```
       * Reading portion of a BFILE data using substr. [Example script: 4010.pco]
         IDENTIFICATION DIVISION.
         PROGRAM-ID. BFILE-SUBSTR.
         ENVIRONMENT DIVISION.
         DATA DIVISION.
         WORKING-STORAGE SECTION.
         01  BFILE1          SQL-BFILE.
         01  BUFFER2         PIC X(32767) VARYING.
         01  AMT             PIC S9(9) COMP.
         01  POS             PIC S9(9) COMP VALUE 1024.
         01  OFFSET          PIC S9(9) COMP VALUE 1.

             EXEC SQL INCLUDE SQLCA END-EXEC.
             EXEC SQL VAR BUFFER2 IS VARRAW(32767) END-EXEC.

         PROCEDURE DIVISION.
         BFILE-SUBSTR.

        * Allocate and initialize the CLOB locator:
             EXEC SQL ALLOCATE :BFILE1 END-EXEC.
             EXEC SQL WHENEVER NOT FOUND GOTO END-OF-BFILE END-EXEC.
             EXEC SQL
                  SELECT PTAB.AD_GRAPHIC INTO :BFILE1
                  FROM PRINT_MEDIA PTAB WHERE PTAB.PRODUCT_ID = 3106 AND PTAB.AD_ID =
13001
             END-EXEC.

        * Open the BFILE for READ ONLY:
             EXEC SQL LOB OPEN :BFILE1 READ ONLY END-EXEC.

        * Execute PL/SQL to use its SUBSTR functionality:
             MOVE 32767 TO AMT.
             EXEC SQL EXECUTE
               BEGIN
                 :BUFFER2 := DBMS_LOB.SUBSTR(:BFILE1,:AMT,:POS);
               END;
             END-EXEC.

        * Close and free the locators:
             EXEC SQL LOB CLOSE :BFILE1 END-EXEC.

         END-OF-BFILE.
             EXEC SQL WHENEVER NOT FOUND CONTINUE END-EXEC.
             EXEC SQL FREE :BFILE1 END-EXEC.
```

C/C++（Pro*C/C++）: BFILE データの一部の読込み（substr）

```
/* Reading portion of a BFILE data using substr. [Example script: 4011.pc]
   Pro*C/C++ lacks an equivalent embedded SQL form for the DBMS_LOB.SUBSTR()
   function. However, Pro*C/C++ can interoperate with PL/SQL using anonymous
   PL/SQL blocks embedded in a Pro*C/C++ program as this example shows: */

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

#define BufferLength 256
void substringBFILE_proc()
{
  OCIBFileLocator *Lob_loc;
  int Position = 1;
  char Buffer[BufferLength];
  EXEC SQL VAR Buffer IS RAW(BufferLength);
  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  EXEC SQL ALLOCATE :Lob_loc;
  EXEC SQL SELECT PMtab.ad_graphic INTO :Lob_loc
           FROM Print_media PMtab WHERE PMtab.product_id = 2056 AND PMTab.ad_id =
12001;
  /* Open the BFILE: */
  EXEC SQL LOB OPEN :Lob_loc READ ONLY;
  /* Invoke SUBSTR() from within an anonymous PL/SQL block:  */
  EXEC SQL EXECUTE
    BEGIN
      :Buffer := DBMS_LOB.SUBSTR(:Lob_loc, 256, :Position);
    END;
  END-EXEC;
  /* Close the BFILE:  */
  EXEC SQL LOB CLOSE :Lob_loc;
  EXEC SQL FREE :Lob_loc;
}
```

```
void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  substringBFILE_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

Visual Basic（OO4O）: BFILE データの一部の読込み（substr）

```
' Reading portion of a BFILE data using substr. [Example script: 4013.txt]
Dim MySession As OraSession
Dim OraDb As OraDatabase

Dim OraDyn As OraDynaset, OraAdGraphic As OraBfile, amount_read%, chunksize%, chunk

Set MySession = CreateObject("OracleInProcServer.XOraSession")
Set OraDb = MySession.OpenDatabase("pmschema", "pm/pm", 0&)

chunk_size = 32767
Set OraDyn = OraDb.CreateDynaset("select * from Print_media", ORADYN_DEFAULT)
Set OraAdGraphic = OraDyn.Fields("ad_graphic").Value
OraMusic.PollingAmount = OraAdGraphic.Size 'Read entire BFILE contents
OraAdGraphic.offset = 255 'Read from the 255th position
'Open the Bfile for reading:
OraAdGraphic.Open
amount_read = OraAdGraphic.Read(chunk, chunk_size) 'chunk returned is a variant of
type byte array
 If amount_read <> chunk_size Then
    'Do error processing
 Else
     'Process the data
 End If
```

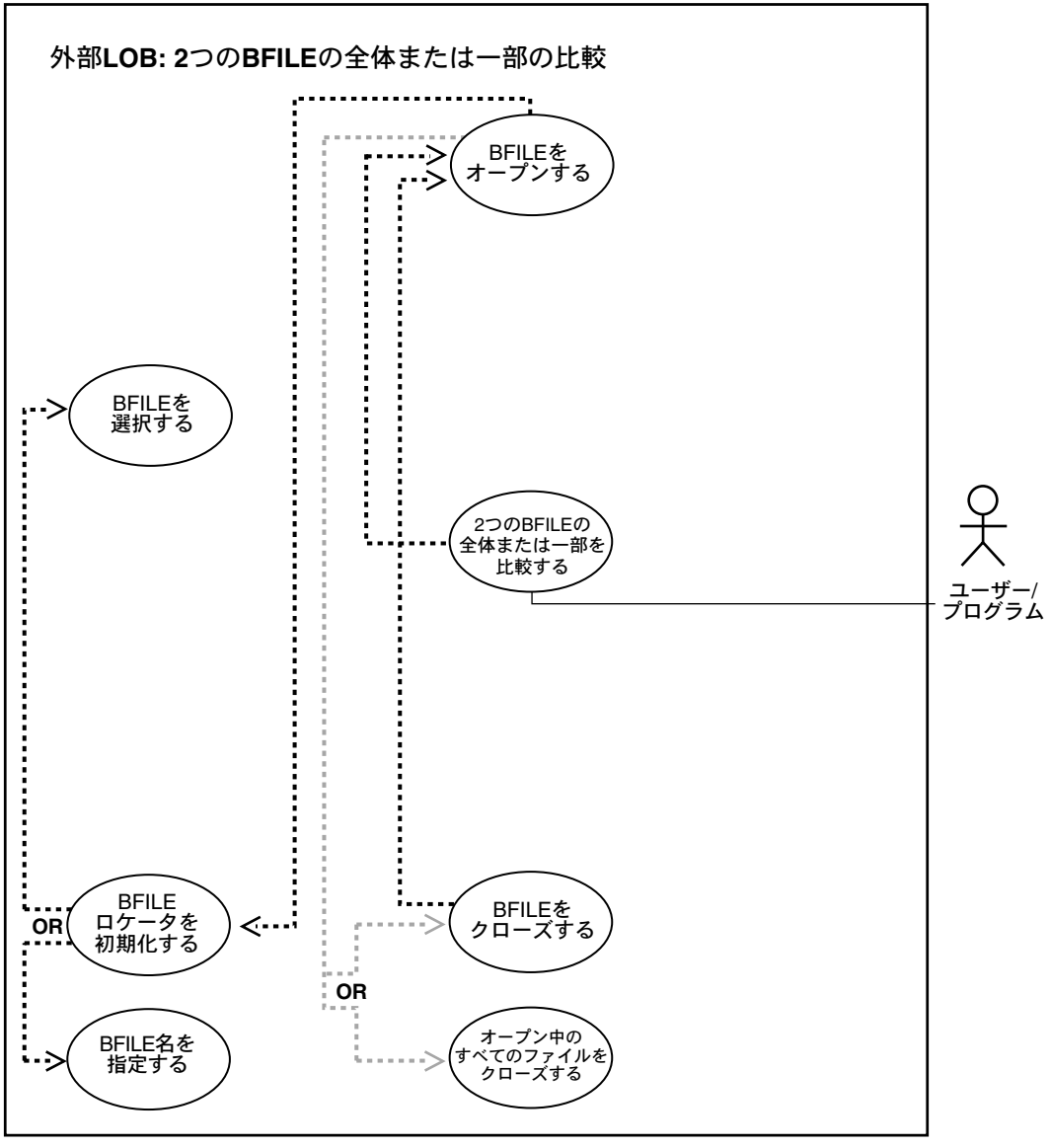
Java（JDBC）: BFILE データの一部の読込み（substr）

```
// Reading portion of a BFILE data using substr. [Example script: 4014.java]

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

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

public class Ex4_62
{

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

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

    // Create a Statement:
    Statement stmt = conn.createStatement ();
    try
    {
   BFILE src_lob = null;
       ResultSet rset = null;
   InputStream in = null;
   byte buf[] = new byte[1000];
   int length = 0;
   rset = stmt.executeQuery (
      "SELECT ad_graphic FROM Print_media WHERE product_id = 3106 AND ad_id =
13001");
   if (rset.next())
   {
     src_lob = ((OracleResultSet)rset).getBFILE (1);
   }
```

```
    // Open the BFILE:
    src_lob.openFile();

    // Get a handle to stream the data from the BFILE
    in = src_lob.getBinaryStream();

    if (in != null)
    {
       // request 255 bytes into buf, starting from offset 1.
       // length = # bytes actually returned from stream:
       length = in.read(buf, 1, 255);
       System.out.println("Bytes read in: " + Integer.toString(length));

       // Process the buf:
       System.out.println(new String(buf));
    }

    // Close the stream, BFILE, statement and connection:
    in.close();
    src_lob.closeFile();
    stmt.close();
    conn.commit();
    conn.close();
     }
     catch (SQLException e)
     {
        e.printStackTrace();
     }
  }
}
```

2 つの BFILE の全体または一部の比較

図 12-20 利用図 : 2 つの BFILE の全体または一部の比較



参照： 内部一時 LOB に関するすべての基本操作については、12-2 ページの表 12-1「利用モデル図 : 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関数またはファンクションのリストは、第 3 章「様々なプログラム環境での LOB のサポート」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）：『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』の第 23 章「DBMS_LOB」の「DBMS_LOB 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）：『Pro*COBOL Precompiler プログラマーズ・ガイド』の LOB に関する詳細、LOB 文の使用上の注意、付録 F「埋込み SQL およびプリコンパイラ・ディレクティブ」の「LOB OPEN（実行可能埋込み SQL 拡張機能）」、および『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』の第 23 章「DBMS_LOB」の「DBMS_LOB サブプログラムの要約」の「COMPARE ファンクション」
- C/C++（Pro*C/C++）：『Pro*C/C++ Precompiler 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」、付録 F「埋込み SQL 文およびディレクティブ」の「LOB OPEN（実行可能埋込み SQL 拡張機能）」、および『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』の第 23 章「DBMS_LOB」の「DBMS_LOB サブプログラムの要約」の「COMPARE ファンクション」
- Visual Basic（OO4O オンライン・ヘルプ）: ヘルプの「目次」タブから、「OO4O オートメーション・サーバー・リファレンス」>「OO4O サーバーのオブジェクト」>「OraBFILE」>「メソッド」>「Open」、「Compare」および「OO4O オートメーション・サーバー・リファレンス」>「OO4O サーバーのオブジェクト」>「OraDatabase」>「プロパティ」>「Parameter」、および「OO4O オートメーション・サーバー・リファレンス」>「OO4O サーバーのオブジェクト」>「OraBFILE」>「例」を選択
- Java （JDBC）：『Oracle9*i* JDBC 開発者ガイドおよびリファレンス』の第 8 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『Oracle9*i* 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「JDBC 2.0 LOB 型と Oracle 拡張型のサポート」の「BLOB、CLOB および BFILE のサポート」

使用例

次の例では、ファイル「ADPHOTO_DIR」内の写真が特定の ad_graphic 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での例を示します。

- PL/SQL（DBMS_LOB）: 2 つの BFILE の全体または一部の比較（12-127 ページ）
- C（OCI）: 今回のリリースでは例は提供されません。
- COBOL（Pro*COBOL）: 2 つの BFILE の全体または一部の比較（12-128 ページ）
- C/C++（Pro*C/C++）: 2 つの BFILE の全体または一部の比較（12-130 ページ）
- Visual Basic（OO4O）: 2 つの BFILE の全体または一部の比較（12-131 ページ）
- Java（JDBC）: 2 つの BFILE の全体または一部の比較（12-132 ページ）

PL/SQL（DBMS_LOB）: 2 つの BFILE の全体または一部の比較

```
/* Comparing all or parts of two BFILES. [Example script: 4015.sql] */
/* Procedure instringBFILE_proc is not part of DBMS_LOB package: */
CREATE OR REPLACE PROCEDURE instringBFILE_proc IS
   File_loc        BFILE;
   Pattern        RAW(32767);
   Position       INTEGER;
BEGIN
   /* Select the LOB: */
   SELECT PMtab.ad_graphic INTO File_loc
      FROM THE(SELECT PMtab.textdoc_ntab FROM Print_media PMtab
         WHERE Product_ID = 3060 AND ad_id = 11001) PMtab
            WHERE Segment = 1;
   /* Open the BFILE: */
   DBMS_LOB.OPEN(File_loc, DBMS_LOB.LOB_READONLY);
   /*  Initialize the pattern for which to search, find the 2nd occurrence of
       the pattern starting from the beginning of the BFILE: */
   Position := DBMS_LOB.INSTR(File_loc, Pattern, 1, 2);
   /* Close the BFILE: */
   DBMS_LOB.CLOSE(File_loc);
END;
```

COBOL（Pro*COBOL）: 2 つの BFILE の全体または一部の比較

```
* Comparing all or parts of two BFILES. [Example script: 4016.pco]
  IDENTIFICATION DIVISION.
  PROGRAM-ID. BFILE-COMPARE.
  ENVIRONMENT DIVISION.
  DATA DIVISION.
  WORKING-STORAGE SECTION.
  01  USERID          PIC X(11) VALUES "SAMP/SAMP".
  01  BFILE1          SQL-BFILE.
  01  BFILE2          SQL-BFILE.
  01  RET             PIC S9(9) COMP.
  01  AMT             PIC S9(9) COMP.
  01  DIR-ALIAS       PIC X(30) VARYING.
  01  FNAME           PIC X(20) VARYING.
  01  ORASLNRD        PIC 9(4).

      EXEC SQL INCLUDE SQLCA END-EXEC.
      EXEC ORACLE OPTION (ORACA=YES) END-EXEC.
      EXEC SQL INCLUDE ORACA END-EXEC.

  PROCEDURE DIVISION.
  BFIlE-COMPARE.

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

 * Allocate and initialize the BLOB locators:
      EXEC SQL ALLOCATE :BFILE1 END-EXEC.
      EXEC SQL ALLOCATE :BFILE2 END-EXEC.

 * Set up the directory and file information:
      MOVE "ADGRAPHIC_DIR" TO DIR-ALIAS-ARR.
      MOVE 9 TO DIR-ALIAS-LEN.
      MOVE "keyboard_graphic_3106_13001" TO FNAME-ARR.
      MOVE 17 TO FNAME-LEN.

      EXEC SQL
         LOB FILE SET :BFILE1 DIRECTORY = :DIR-ALIAS,
         FILENAME = :FNAME
      END-EXEC.

      EXEC SQL WHENEVER NOT FOUND GOTO END-OF-BFILE END-EXEC.
      EXEC SQL
           SELECT AD_GRAPHIC INTO :BFILE2
```

```
           FROM PRINT_MEDIA WHERE PRODUCT_ID = 3106 AND ad_id = 13001
     END-EXEC.

* Open the BLOBs for READ ONLY:
     EXEC SQL LOB OPEN :BFILE1 READ ONLY END-EXEC.
     EXEC SQL LOB OPEN :BFILE2 READ ONLY END-EXEC.

* Execute PL/SQL to get COMPARE functionality:
     MOVE 5 TO AMT.
     EXEC SQL EXECUTE
       BEGIN
         :RET := DBMS_LOB.COMPARE(:BFILE1,:BFILE2,
                                  :AMT,1,1);
       END;
     END-EXEC.

     IF RET = 0
*        Logic for equal BFILEs goes here
         DISPLAY "BFILES are equal"
     ELSE
*        Logic for unequal BFILEs goes here
         DISPLAY "BFILEs are not equal"
     END-IF.

     EXEC SQL LOB CLOSE :BFILE1 END-EXEC.
     EXEC SQL LOB CLOSE :BFILE2 END-EXEC.
 END-OF-BFILE.
     EXEC SQL WHENEVER NOT FOUND CONTINUE END-EXEC.
     EXEC SQL FREE :BFILE1 END-EXEC.
     EXEC SQL FREE :BFILE2 END-EXEC.
     STOP RUN.

 SQL-ERROR.
     EXEC SQL
         WHENEVER SQLERROR CONTINUE
     END-EXEC.
     MOVE ORASLNR TO ORASLNRD.
     DISPLAY " ".
     DISPLAY "ORACLE ERROR DETECTED ON LINE ", ORASLNRD, ":".
     DISPLAY " ".
     DISPLAY SQLERRMC.
     EXEC SQL
         ROLLBACK WORK RELEASE
     END-EXEC.
     STOP RUN.
```

C/C++（Pro*C/C++）: 2 つの BFILE の全体または一部の比較

```
/* Comparing all or parts of two BFILES. [Example script: 4017.pc]
   Pro*C/C++ lacks an equivalent embedded SQL form for the
   DBMS_LOB.COMPARE() function. Like the DBMS_LOB.SUBSTR() function,
   however, Pro*C/C++ can invoke DBMS_LOB.COMPARE() in an anonymous PL/SQL
   block as shown here:  */

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

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

void compareBFILEs_proc()
{
  OCIBFileLocator *Lob_loc1, *Lob_loc2;
  int Retval = 1;
  char *Dir1 = "GRAPHIC_DIR", *Name1 = "mousepad_2056";

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  EXEC SQL ALLOCATE :Lob_loc1;
  EXEC SQL LOB FILE SET :Lob_loc1 DIRECTORY = :Dir1, FILENAME = :Name1;
  EXEC SQL ALLOCATE :Lob_loc2;
  EXEC SQL SELECT Photo INTO :Lob_loc2 FROM Print_media
           WHERE Product_ID = 2056;
  /* Open the BFILEs: */
  EXEC SQL LOB OPEN :Lob_loc1 READ ONLY;
  EXEC SQL LOB OPEN :Lob_loc2 READ ONLY;
  /* Compare the BFILEs in PL/SQL using DBMS_LOB.COMPARE() */
  EXEC SQL EXECUTE
    BEGIN
      :Retval := DBMS_LOB.COMPARE(
                   :Lob_loc2, :Lob_loc1, DBMS_LOB.LOBMAXSIZE, 1, 1);
    END;
  END-EXEC;
  /* Close the BFILEs:  */
  EXEC SQL LOB CLOSE :Lob_loc1;
  EXEC SQL LOB CLOSE :Lob_loc2;
```

```
  if (0 == Retval)
    printf("BFILEs are the same\n");
  else
    printf("BFILEs are not the same\n");
  /* Release resources used by the locators: */
  EXEC SQL FREE :Lob_loc1;
  EXEC SQL FREE :Lob_loc2;
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  compareBFILEs_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

Visual Basic（OO4O）: 2 つの BFILE の全体または一部の比較

```
'Comparing all or parts of two BFILES. [Example script: 4018.txt]
'The PL/SQL packages and the tables mentioned here are not part of the
'standard OO4O installation:

Dim MySession As OraSession
Dim OraDb As OraDatabase
Dim OraDyn As OraDynaset, OraAdGraphic As OraBfile, OraMyAdGraphic As OraBfile,
OraSql As OraSqlStmt

Set MySession = CreateObject("OracleInProcServer.XOraSession")
Set OraDb = MySession.OpenDatabase("pmschema", "pm/pm", 0&)

OraDb.Connection.BeginTrans

Set OraParameters = OraDb.Parameters

OraParameters.Add "id", 3106, ORAPARM_INPUT

'Define out parameter of BFILE type:
OraParameters.Add "MyAdGraphic", Null, ORAPARM_OUTPUT
OraParameters("MyAdGraphic").ServerType = ORATYPE_BFILE
```

```
Set OraSql =
   OraDb.CreateSql(
      "BEGIN SELECT ad_graphic INTO :MyAdGraphic FROM Print_media WHERE product_id =
:id;
         END;", ORASQL_FAILEXEC)

Set OraMyAdGraphic = OraParameters("MyAdGraphic").Value

'Create dynaset:
Set OraDyn =
   OraDb.CreateDynaset(
      "SELECT * FROM Print_media WHERE product_id = 3106", ORADYN_DEFAULT)
Set OraAdGraphic = OraDyn.Fields("ad_graphic").Value

'Open the Bfile for reading:
OraAdGraphic.Open
OraMyAdGraphic.Open

If OraAdGraphic.Compare(OraMyAdGraphic) Then
    'Process the data
Else
   'Do error processing
End If
OraDb.Connection.CommitTrans
```

Java（JDBC）: 2 つの BFILE の全体または一部の比較

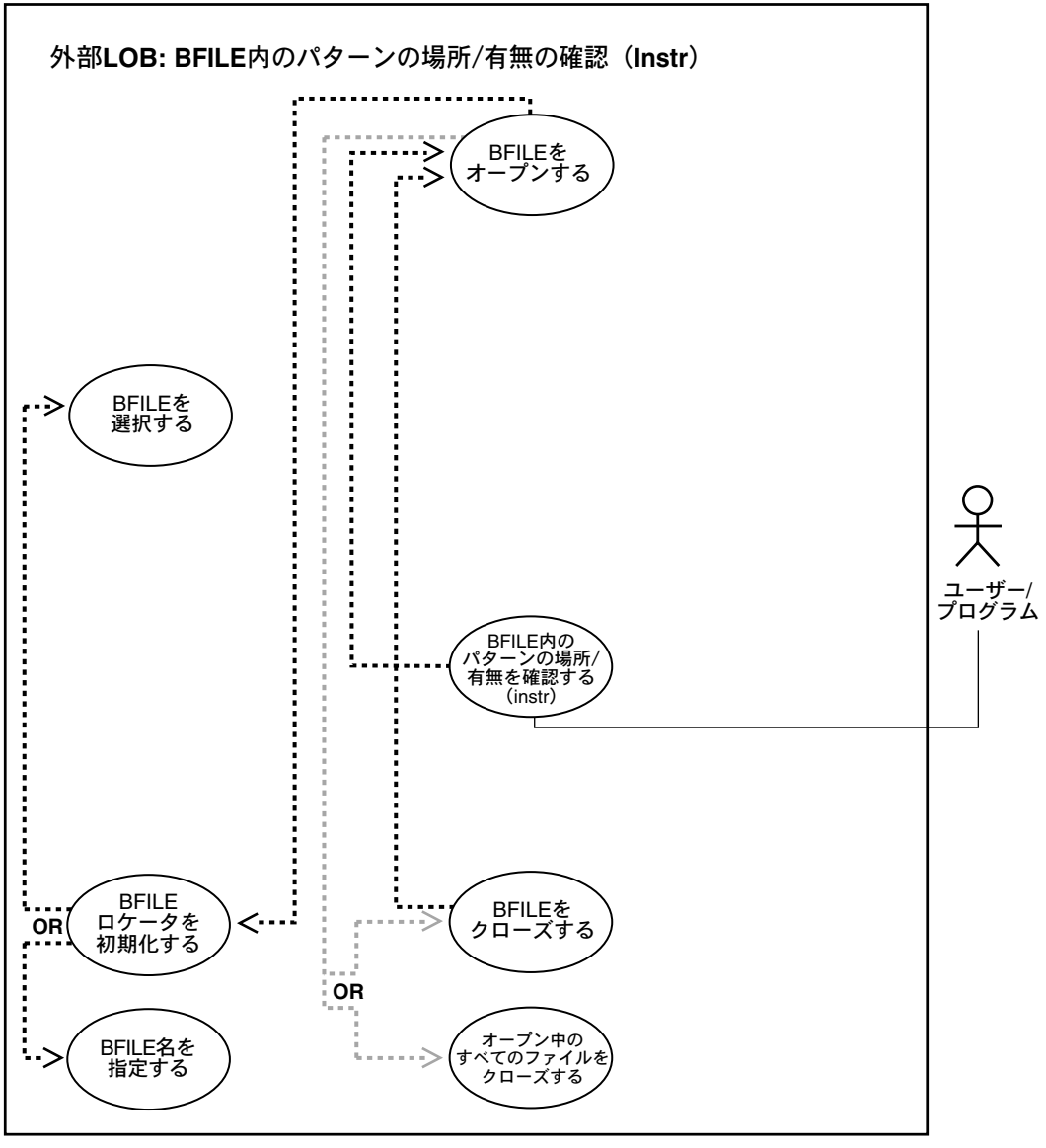
```
// Comparing all or parts of two BFILES. [Example script: 4019.java]

import java.io.InputStream;
import java.io.OutputStream;

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

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

public class Ex4_66
```

```
{

  static final int MAXBUFSIZE = 32767;

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

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

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

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

    try
    {
       BFILE lob_loc1 = null;
       BFILE lob_loc2 = null;
       ResultSet rset = null;

       rset = stmt.executeQuery (
          "SELECT ad_graphic FROM Print_media WHERE product_id = 3106");
       if (rset.next())
       {
          lob_loc1 = ((OracleResultSet)rset).getBFILE (1);
       }

       rset = stmt.executeQuery (
          "SELECT BFILENAME('AD_GRAPHIC', 'keyboard_3106') FROM DUAL");
       if (rset.next())
       {
          lob_loc2 = ((OracleResultSet)rset).getBFILE (1);
       }

       lob_loc1.openFile ();
       lob_loc2.openFile ();

       if (lob_loc1.length() > lob_loc2.length())
       System.out.println("Looking for LOB2 inside LOB1.  result = " +
          lob_loc1.position(lob_loc2, 1));
```

```
    else
      System.out.println("Looking for LOB1 inside LOB2.  result = " +
           lob_loc2.position(lob_loc1, 1));

    stmt.close();
    conn.commit();
    conn.close();

     }
     catch (SQLException e)
     {
         e.printStackTrace();
     }
   }
}
```

BFILE 内のパターンの有無の確認（instr）

図 12-21 利用図 : BFILE 内のパターンの有無の確認



参照： 内部一時 LOB に関するすべての基本操作については、12-2 ページの表 12-1「利用モデル図 : 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関数またはファンクションのリストは、第 3 章「様々なプログラム環境での LOB のサポート」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）：『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』の第 23 章「DBMS_LOB」の「DBMS_LOB 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）：『Pro*COBOL Precompiler プログラマーズ・ガイド』の LOB に関する詳細、LOB 文の使用上の注意、付録 F「埋込み SQL およびプリコンパイラ・ディレクティブ」の「LOB OPEN（実行可能埋込み SQL 拡張機能）」、および『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』の第 23 章「DBMS_LOB」の「DBMS_LOB サブプログラムの要約」の「INSTR ファンクション」
- C/C++（Pro*C/C++）：『Pro*C/C++ Precompiler 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」、付録 F「埋込み SQL 文およびディレクティブ」の「LOB OPEN（実行可能埋込み SQL 拡張機能）」、および『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』の第 23 章「DBMS_LOB」の「DBMS_LOB 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）：『Oracle9*i* JDBC 開発者ガイドおよびリファレンス』の第 8 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『Oracle9*i* 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「JDBC 2.0 LOB 型と Oracle 拡張型のサポート」の「BLOB、CLOB および BFILE のサポート」

使用例

次の例では、ad_graphic イメージ内でのパターンの発生を検索します。

例

次のプログラム環境での例を示します。

- PL/SQL（DBMS_LOB）: BFILE 内のパターンの有無の確認（instr）（12-137 ページ）
- C（OCI）: 今回のリリースでは例は提供されません。
- COBOL（Pro*COBOL）: BFILE 内のパターンの有無の確認（instr）（12-138 ページ）
- C/C++（Pro*C/C++）: BFILE 内のパターンの有無の確認（instr）（12-139 ページ）
- Visual Basic（OO4O）: 今回のリリースでは例は提供されません。
- Java（JDBC）: BFILE 内のパターンの有無の確認（instr）（12-141 ページ）

PL/SQL（DBMS_LOB）: BFILE 内のパターンの有無の確認（instr）

```
/* Checking if a pattern exists in a BFILE using instr [Example script: 4030.sql]
/* Procedure compareBFILEs_proc is not part of DBMS_LOB package: */

CREATE OR REPLACE PROCEDURE compareBFILEs_proc IS
   /* Initialize the BFILE locator: */
   File_loc1       BFILE := BFILENAME('ADPHOTO_DIR', 'keyboard_photo_3060_11001');
   File_loc2       BFILE;
   Retval         INTEGER;
BEGIN
   /* Select the LOB: */
   SELECT ad_graphic INTO File_loc2 FROM print_media
      WHERE Product_ID = 3060 AND ad_id = 11001;
   /* Open the BFILEs: */
   DBMS_LOB.OPEN(File_loc1, DBMS_LOB.LOB_READONLY);
   DBMS_LOB.OPEN(File_loc2, DBMS_LOB.LOB_READONLY);
   Retval := DBMS_LOB.COMPARE(File_loc2, File_loc1, DBMS_LOB.LOBMAXSIZE, 1, 1);
   /* Close the BFILEs: */
   DBMS_LOB.CLOSE(File_loc1);
   DBMS_LOB.CLOSE(File_loc2);
END;
```

COBOL（Pro*COBOL）: BFILE 内のパターンの有無の確認（instr）

```
     * Checking if a pattern exists in a BFILE using instr [Example script:
4021.pco]
       IDENTIFICATION DIVISION.
       PROGRAM-ID. BFILE-INSTR.
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.

       01  USERID   PIC X(11) VALUES "SAMP/SAMP".
       01  BFILE1         SQL-BFILE.

      * The length of pattern was chosen arbitrarily:
       01  PATTERN        PIC X(4) VALUE "2424".
           EXEC SQL VAR PATTERN IS RAW(4) END-EXEC.
       01  POS            PIC S9(9) COMP.
       01  ORASLNRD       PIC 9(4).

           EXEC SQL INCLUDE SQLCA END-EXEC.
           EXEC ORACLE OPTION (ORACA=YES) END-EXEC.
           EXEC SQL INCLUDE ORACA END-EXEC.

       PROCEDURE DIVISION.
       BFILE-INSTR.

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

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

           EXEC SQL WHENEVER NOT FOUND GOTO END-OF-BFILE END-EXEC.
           EXEC SQL
                SELECT AD_GRAPHIC INTO :BFILE1
                FROM PRINT_MEDIA WHERE PRODUCT_ID = 3106 AND AD_ID = 13001
           END-EXEC.

      * Open the CLOB for READ ONLY:
           EXEC SQL LOB OPEN :BFILE1 READ ONLY END-EXEC.

      * Execute PL/SQL to get INSTR functionality:
           EXEC SQL EXECUTE
             BEGIN
               :POS := DBMS_LOB.INSTR(:BFILE1,:PATTERN, 1, 2); END; END-EXEC.
```

```
            IF POS = 0
        *       Logic for pattern not found here
                DISPLAY "Pattern is not found."
            ELSE
        *       Pos contains position where pattern is found
                DISPLAY "Pattern is found."
            END-IF.

        * Close and free the LOB:
            EXEC SQL LOB CLOSE :BFILE1 END-EXEC.

         END-OF-BFILE.
            EXEC SQL WHENEVER NOT FOUND CONTINUE END-EXEC.
            EXEC SQL FREE :BFILE1 END-EXEC.
            EXEC SQL ROLLBACK WORK RELEASE END-EXEC.
            STOP RUN.

         SQL-ERROR.
            EXEC SQL WHENEVER SQLERROR CONTINUE END-EXEC.
            MOVE ORASLNR TO ORASLNRD.
            DISPLAY " ".
            DISPLAY "ORACLE ERROR DETECTED ON LINE ", ORASLNRD, ":".
            DISPLAY " ".
            DISPLAY SQLERRMC.
            EXEC SQL ROLLBACK WORK RELEASE END-EXEC.
            STOP RUN.
```

C/C++（Pro*C/C++）: BFILE 内のパターンの有無の確認（instr）

```
/* Checking if a pattern exists in a BFILE using instr [Example script: 4022.pc]
   Pro*C lacks an equivalent embedded SQL form of the DBMS_LOB.INSTR()
   function. However, like SUBSTR() and COMPARE(), Pro*C/C++ can call
   DBMS_LOB.INSTR() from within an anonymous PL/SQL block as shown here: */

#include <sql2oci.h>
#include <stdio.h>
#include <string.h>
#include <sqlca.h>

void Sample_Error()
{
  EXEC SQL WHENEVER SQLERROR CONTINUE;
  printf("%.*s\n", sqlca.sqlerrm.sqlerrml, sqlca.sqlerrm.sqlerrmc);
```

```
  EXEC SQL ROLLBACK WORK RELEASE;
  exit(1);
}

#define PatternSize 5

void instringBFILE_proc()
{
  OCIBFileLocator *Lob_loc;
  unsigned int Position = 0;
  int Product_id = 2056, Segment = 1;
  char Pattern[PatternSize];
  /* Datatype Equivalencing is Mandatory for this Datatype:  */
  EXEC SQL VAR Pattern IS RAW(PatternSize);

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  EXEC SQL ALLOCATE :Lob_loc;
  /* Use Dynamic SQL to retrieve the BFILE Locator:  */
  EXEC SQL PREPARE S FROM
    'SELECT Intab.ad_graphic \
       FROM TABLE(SELECT PMtab.textdoc_ntab FROM Print_media PMtab \
          WHERE product_id = :cid) PMtab \
             WHERE PMtab.Segment = :seg';
  EXEC SQL DECLARE C CURSOR FOR S;
  EXEC SQL OPEN C USING :Product_ID, :Segment;
  EXEC SQL FETCH C INTO :Lob_loc;
  EXEC SQL CLOSE C;
  /* Open the BFILE: */
  EXEC SQL LOB OPEN :Lob_loc READ ONLY;
  memset((void *)Pattern, 0, PatternSize);
  /* Find the first occurrance of the pattern starting from the
     beginning of the BFILE using PL/SQL: */
  EXEC SQL EXECUTE
    BEGIN
      :Position := DBMS_LOB.INSTR(:Lob_loc, :Pattern, 1, 1);
    END;
  END-EXEC;
  /* Close the BFILE: */
  EXEC SQL LOB CLOSE :Lob_loc;
  if (0 == Position)
    printf("Pattern not found\n");
  else
    printf("The pattern occurs at %d\n", Position);
  EXEC SQL FREE :Lob_loc;
}
```

```
void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  instringBFILE_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

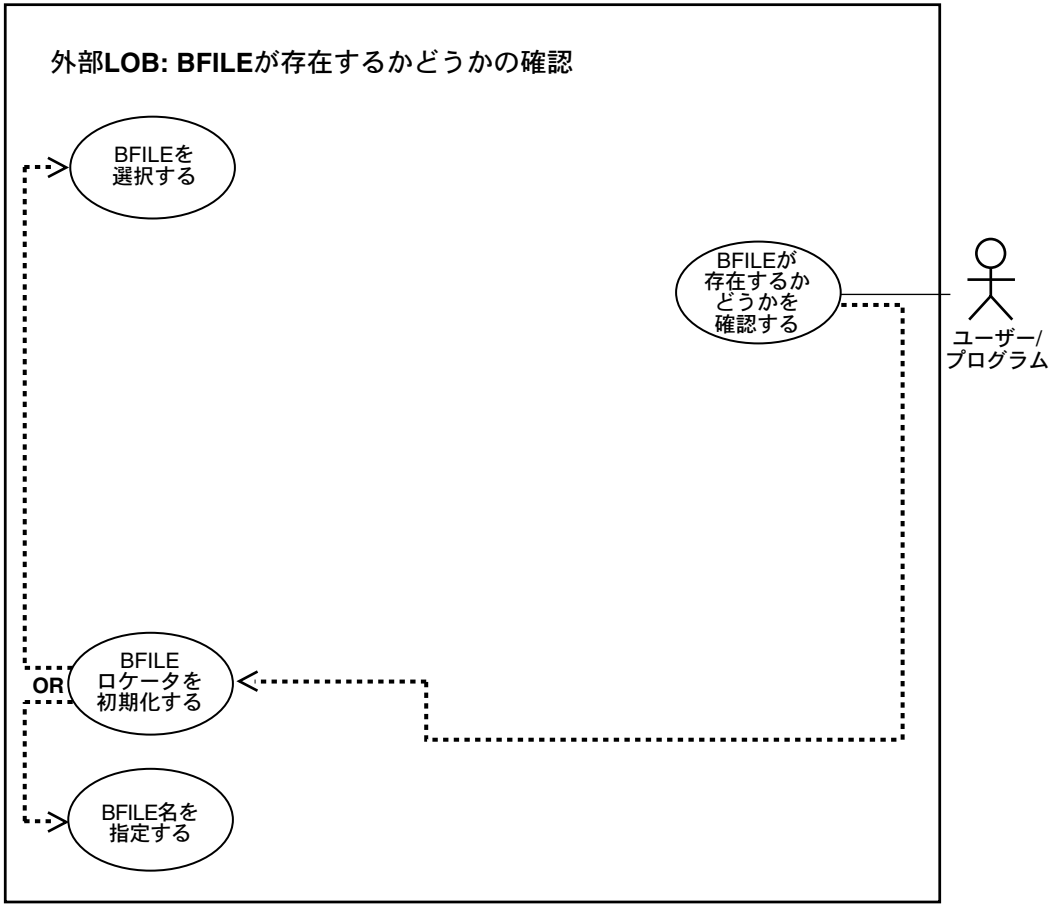
Java（JDBC）：BFILE 内のパターンの有無の確認（instr）

```
// Checking if a pattern exists in a BFILE using instr [Example script: 4024.java]

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

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

public class Ex4_70
{

  static final int MAXBUFSIZE = 32767;

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

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

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

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

    try
    {
       BFILE lob_loc = null;
       // Pattern to look for within the BFILE:
       String pattern = new String("children");

       ResultSet rset = stmt.executeQuery (
          "SELECT ad_graphic FROM Print_media
              WHERE product_id = 3106 AND ad_id = 13001");
       if (rset.next())
       {
          lob_loc = ((OracleResultSet)rset).getBFILE (1);
       }

       // Open the LOB:
       lob_loc.openFile();
       // Search for the location of pattern string in the BFILE,
       // starting at offset 1:
      long result = lob_loc.position(pattern.getBytes(), 1);
      System.out.println(
         "Results of Pattern Comparison : " + Long.toString(result));

      // Close the LOB:
      lob_loc.closeFile();

      stmt.close();
      conn.commit();
      conn.close();

    }
    catch (SQLException e)
    {
       e.printStackTrace();
    }
  }
}
```

BFILE が存在するかどうかの確認

図 12-22 利用図 : BFILE が存在するかどうかの確認



参照 : 内部一時 LOB に関するすべての基本操作については、12-2 ページの表 12-1「利用モデル図 : 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関数またはファンクションのリストは、第 3 章「様々なプログラム環境での LOB のサポート」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）：『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』の第 23 章「DBMS_LOB」の「DBMS_LOB サブプログラムの要約」の「FILEEXISTS ファンクション」
- C（OCI）：『Oracle Call Interface 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 16 章「その他の OCI リレーショナル関数」の「LOB 関数」の「OCILobFileExists()」
- COBOL（Pro*COBOL）：『Pro*COBOL Precompiler プログラマーズ・ガイド』の LOB に関する詳細、LOB 文の使用上の注意および付録 F「埋込み SQL およびプリコンパイラ・ディレクティブ」の「LOB DESCRIBE（実行可能埋込み SQL 拡張機能）」
- C/C++（Pro*C/C++）：『Pro*C/C++ Precompiler 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」および付録 F「埋込み SQL 文およびディレクティブ」の「LOB DESCRIBE（実行可能埋込み SQL 拡張機能）」
- Visual Basic（OO4O オンライン・ヘルプ）：ヘルプの「目次」タブから、「OO4O オートメーション・サーバー・リファレンス」>「OO4O サーバーのオブジェクト」>「OraBFILE」>「プロパティ」>「Exists」、「OO4O オートメーション・サーバー・リファレンス」>「OO4O サーバーのオブジェクト」>「OraDatabase」>「プロパティ」>「Parameter」、および「OO4O オートメーション・サーバー・リファレンス」>「OO4O サーバーのオブジェクト」>「OraBFILE」>「例」を選択
- Java （JDBC）：『Oracle9*i* JDBC 開発者ガイドおよびリファレンス』の第 8 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『Oracle9*i* 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「JDBC 2.0 LOB 型と Oracle 拡張型のサポート」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せます。

例

次のプログラム環境での例を示します。

PL/SQL（DBMS_LOB）: BFILE が存在するかどうかの確認

```
/* Checking if a BFILE exists [Example script: 4025.sql] */
/* Procedure seeIfExistsBFILE_proc is not part of DBMS_LOB package:  */

CREATE OR REPLACE PROCEDURE seeIfExistsBFILE_proc IS
   File_loc      BFILE;
BEGIN
   /* Select the LOB: */
   SELECT Intab.ad_graphic INTO File_loc
      FROM THE(SELECT PMtab.textdoc_ntab FROM Print_media PMtab
         WHERE PMtab.product_id = 3060 AND ad_id = 11001) PMtab
            WHERE PMtab.Segment = 1;
   /* See If the BFILE exists: */
   IF (DBMS_LOB.FILEEXISTS(File_loc) != 0)
   THEN
      DBMS_OUTPUT.PUT_LINE('Processing given that the BFILE exists');
   ELSE
      DBMS_OUTPUT.PUT_LINE('Processing given that the BFILE does not exist');
   END IF;
EXCEPTION
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('Operation failed');
END;
```

C（OCI）: BFILE が存在するかどうかの確認

```
/* Checking if a BFILE exists [Example script: 4026.c] */

/* Select the lob/bfile from the Print_media table */
void selectLob(Lob_loc, errhp, svchp, stmthp)
OCILobLocator *Lob_loc;
OCIError      *errhp;
OCISvcCtx     *svchp;
OCIStmt       *stmthp;
{
     OCIDefine *dfnhp, *dfnhp2;
     text *selstmt = (text *) "SELECT ad_graphic FROM Print_media \
                                  WHERE product_id = 3106 AND ad_id = 13001";

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

     /* Call define for the bfile column */
     checkerr (errhp, OCIDefineByPos(stmthp, &dfnhp, errhp, 1,
                                     (dvoid *)&Lob_loc, 0 , SQLT_BFILE,
                                     (dvoid *)0, (ub2 *)0, (ub2 *)0,
                                     OCI_DEFAULT)
     || OCIDefineByPos(stmthp, &dfnhp2, errhp, 2,
                                     (dvoid *)&Lob_loc, 0 , SQLT_BFILE,
                                     (dvoid *)0, (ub2 *)0, (ub2 *)0,
                                     OCI_DEFAULT));

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

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

   OCILobLocator *bfile_loc;
   boolean is_exist;
```

```
  /* Allocate the locator descriptor */
  (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &bfile_loc,
                            (ub4) OCI_DTYPE_FILE,
                            (size_t) 0, (dvoid **) 0);
  /* Select the bfile */
  selectLob(bfile_loc, errhp, svchp, stmthp);

  checkerr (errhp, OCILobFileExists(svchp, errhp, bfile_loc, &is_exist));

  if (is_exist == TRUE)
  {
    printf("File exists\n");
  }
  else
  {
    printf("File does not exist\n");
  }

  /* Free the locator descriptor */
  OCIDescriptorFree((dvoid *)bfile_loc, (ub4)OCI_DTYPE_FILE);
}
```

COBOL（Pro*COBOL）: BFILE が存在するかどうかの確認

```
      * Checking if a BFILE exists.  [Example script: 4027.pco]
        IDENTIFICATION DIVISION.
        PROGRAM-ID. BFILE-EXISTS.
        ENVIRONMENT DIVISION.
        DATA DIVISION.
        WORKING-STORAGE SECTION.

        01  USERID         PIC X(11) VALUES "SAMP/SAMP".
        01  BFILE1         SQL-BFILE.
        01  FEXISTS        PIC S9(9) COMP.
        01  ORASLNRD       PIC 9(4).

            EXEC SQL INCLUDE SQLCA END-EXEC.
            EXEC ORACLE OPTION (ORACA=YES) END-EXEC.
            EXEC SQL INCLUDE ORACA END-EXEC.

        PROCEDURE DIVISION.
        BFILE-EXISTS.
```

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

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

     EXEC SQL WHENEVER NOT FOUND GOTO END-OF-BFILE END-EXEC.
     EXEC SQL
          SELECT AD_GRAPHIC INTO :BFILE1
          FROM PRINT_MEDIA WHERE PRODUCT_ID = 3106 AND AD_ID = 13001
     END-EXEC.

     EXEC SQL
          LOB DESCRIBE :BFILE1 GET FILEEXISTS INTO :FEXISTS
     END-EXEC.

     IF FEXISTS = 1
*        Logic for file exists here
         DISPLAY "File exists"
     ELSE
*        Logic for file does not exist here
         DISPLAY "File does not exist"
     END-IF.

 END-OF-BFILE.
     EXEC SQL WHENEVER NOT FOUND CONTINUE END-EXEC.
     EXEC SQL FREE :BFILE1 END-EXEC.
     EXEC SQL ROLLBACK WORK RELEASE END-EXEC.
     STOP RUN.

 SQL-ERROR.
     EXEC SQL WHENEVER 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

```
/* Checking if a BFILE exists.  [Example script: 4028.pc] */

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

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

void seeIfBFILEExists_proc()
{
  OCIBFileLocator *Lob_loc;
  unsigned int Exists = 0;

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  EXEC SQL ALLOCATE :Lob_loc;
  EXEC SQL SELECT PMtab.ad_graphic INTO :Lob_loc
           FROM Print_media PMtab WHERE PMtab.Product_ID = 2056 AND PMtab.ad_id =
12001;
  /* See if the BFILE Exists:  */
  EXEC SQL LOB DESCRIBE :Lob_loc GET FILEEXISTS INTO :Exists;
  printf("BFILE %s exist\n", Exists ? "does" : "does not");
  EXEC SQL FREE :Lob_loc;
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  seeIfBFILEExists_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

Visual Basic（OO4O）: BFILE が存在するかどうかの確認

```
'Checking if a BFILE exists.  [Example script: 4030.txt]
'The PL/SQL packages and the tables mentioned here are not part of the
'standard OO4O installation:

Dim MySession As OraSession
Dim OraDb As OraDatabase
Dim OraAdGraphic As OraBfile, OraSql As OraSqlStmt

Set MySession = CreateObject("OracleInProcServer.XOraSession")
Set OraDb = MySession.OpenDatabase("pmschema", "pm/pm", 0&)

OraDb.Connection.BeginTrans

Set OraParameters = OraDb.Parameters

OraParameters.Add "id", 2056, ORAPARM_INPUT

'Define out parameter of BFILE type:
OraParameters.Add "MyAdGraphic", Null, ORAPARM_OUTPUT
OraParameters("MyAdGraphic").ServerType = ORATYPE_BFILE

Set OraSql =
   OraDb.CreateSql(
      "BEGIN SELECT ad_graphic INTO :MyAdGraphic FROM Print_media WHERE
      product_id = :id;
           END;", ORASQL_FAILEXEC)

Set OraAdGraphic = OraParameters("MyAdGraphic").Value

If OraAdGraphic.Exists Then
    'Process the data
Else
   'Do error processing
End If
OraDb.Connection.CommitTrans
```

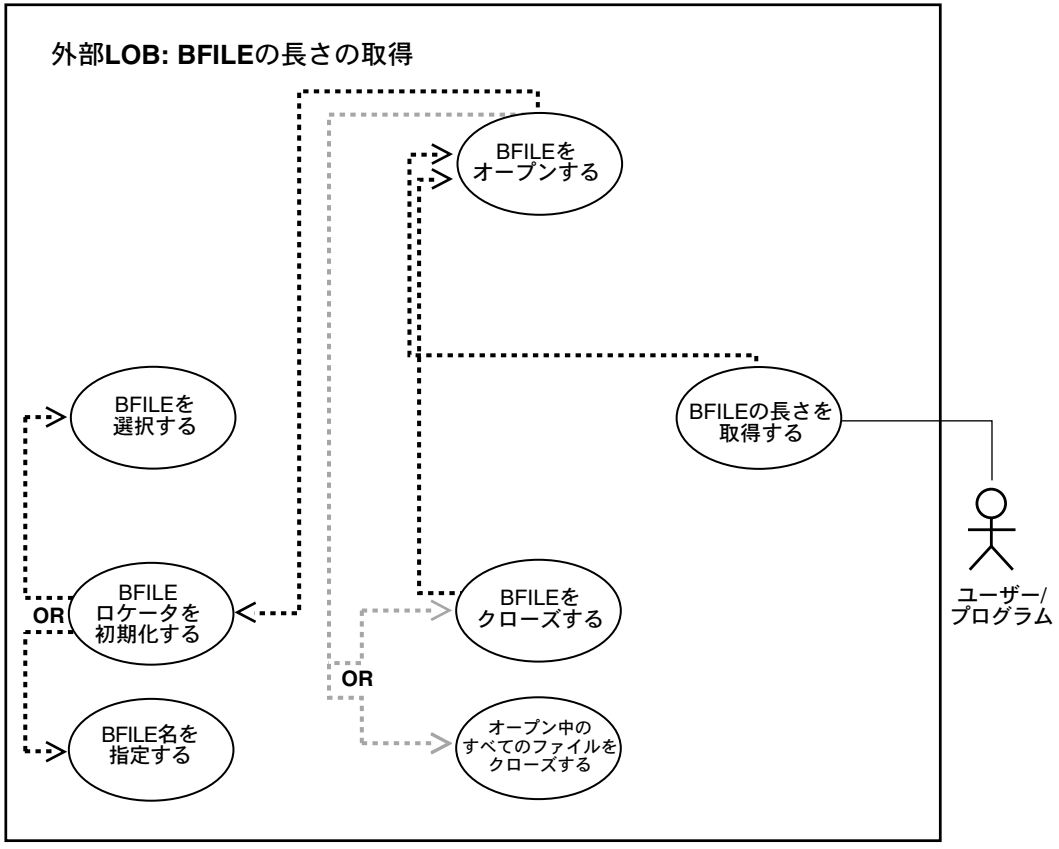
Java（JDBC）: BFILE が存在するかどうかの確認

```
// Checking if a BFILE exists.  [Example script:  4031.java]

import java.io.InputStream;
import java.io.OutputStream;

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

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

public class Ex4_74
{

  static final int MAXBUFSIZE = 32767;

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

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

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

    // Create a Statement
    Statement stmt = conn.createStatement ();
    try
    {
       BFILE lob_loc = null;
       ResultSet rset = stmt.executeQuery (
          "SELECT ad_graphic FROM Print_media
              WHERE product_id = 3106 AND ad_id = 13001");
       if (rset.next())
```

```
        {
           lob_loc = ((OracleResultSet)rset).getBFILE (1);
        }

        // See if the BFILE exists:
        System.out.println("Result from fileExists(): " + lob_loc.fileExists());

        // Return the length of the BFILE:
        long length = lob_loc.length();
        System.out.println("Length of BFILE: " + length);

        // Get the directory alias for this BFILE:
        System.out.println("Directory alias: " + lob_loc.getDirAlias());

        // Get the file name for this BFILE:
        System.out.println("File name: " + lob_loc.getName());
        stmt.close();
        conn.commit();
        conn.close();
     }
     catch (SQLException e)
     {
        e.printStackTrace();
     }
  }
}
```

BFILE の長さの取得

図 12-23 利用図 : BFILE の長さの取得



参照 : 内部一時 LOB に関するすべての基本操作については、12-2 ページの表 12-1「利用モデル図 : 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関数またはファンクションのリストは、第 3 章「様々なプログラム環境での LOB のサポート」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）：『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』の第 23 章「DBMS_LOB」の「DBMS_LOB サブプログラムの要約」の「GETLENGTH ファンクション」
- C（OCI）：『Oracle Call Interface 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 16 章「その他の OCI リレーショナル関数」の「LOB 関数」の「OCILobGetLength()」
- COBOL（Pro*COBOL）：『Pro*COBOL Precompiler プログラマーズ・ガイド』の LOB に関する詳細、LOB 文の使用上の注意および付録 F「埋込み SQL およびプリコンパイラ・ディレクティブ」の「LOB DESCRIBE（実行可能埋込み SQL 拡張機能）」
- C/C++（Pro*C/C++）：『Pro*C/C++ Precompiler 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」および付録 F「埋込み SQL 文およびディレクティブ」の「LOB DESCRIBE（実行可能埋込み SQL 拡張機能）」
- Visual Basic（OO4O オンライン・ヘルプ）：ヘルプの 「目次」タブから、「OO4O オートメーション・サーバー・リファレンス」>「OO4O サーバーのオブジェクト」>「OraBFILE」>「プロパティ」>「Size」、および「OO4O オートメーション・サーバー・リファレンス」>「OO4O サーバーのオブジェクト」>「OraBfile」>「例」を選択
- Java（JDBC）：『Oracle9*i* JDBC 開発者ガイドおよびリファレンス』の第 8 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『Oracle9*i* 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「JDBC 2.0 LOB 型と Oracle 拡張型のサポート」の「BLOB、CLOB および BFILE のサポート」

使用例

次の例では、ad_graphic 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での例を示します。

PL/SQL（DBMS_LOB）: BFILE の長さの取得

```
/* Getting the length of a BFILE. [Example script: 4032.sql]
/* Procedure getLengthBFILE_proc is not part of DBMS_LOB package: */

CREATE OR REPLACE PROCEDURE getLengthBFILE_proc IS
   File_loc      BFILE;
   Length       INTEGER;
BEGIN
   /* Initialize the BFILE locator by selecting the LOB: */
   SELECT PMtab.ad_graphic INTO File_loc FROM Print_media PMtab
      WHERE PMtab.product_id = 3060 AND PMtab.ad_id = 11001;
   /* Open the BFILE: */
   DBMS_LOB.OPEN(File_loc, DBMS_LOB.LOB_READONLY);
   /* Get the length of the LOB: */
   Length := DBMS_LOB.GETLENGTH(File_loc);
   IF Length IS NULL THEN
       DBMS_OUTPUT.PUT_LINE('BFILE is null.');
   ELSE
       DBMS_OUTPUT.PUT_LINE('The length is ' || length);
   END IF;
   /* Close the BFILE: */
   DBMS_LOB.CLOSE(File_loc);
END;
```

C（OCI）: BFILE の長さの取得

```
/* Getting the length of a BFILE. [Example script: 4033.c] */
/* Select the lob/bfile from table Print_media */

void selectLob(Lob_loc, errhp, svchp, stmthp)
OCILobLocator *Lob_loc;
OCIError      *errhp;
OCISvcCtx     *svchp;
OCIStmt       *stmthp;
{
     OCIDefine *dfnhp, *dfnhp2;
     text *selstmt = (text *) "SELECT ad_graphic FROM Print_media \
                               WHERE product_id = 3106 AND ad_id = 13001";

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

     /* Call define for the bfile column */
     checkerr (errhp, OCIDefineByPos(stmthp, &dfnhp, errhp, 1,
                                     (dvoid *)&Lob_loc, 0 , SQLT_BFILE,
                                     (dvoid *)0, (ub2 *)0, (ub2 *)0,
                                     OCI_DEFAULT)
     || OCIDefineByPos(stmthp, &dfnhp2, errhp, 2,
                                     (dvoid *)&Lob_loc, 0 , SQLT_BFILE,
                                     (dvoid *)0, (ub2 *)0, (ub2 *)0,
                                     OCI_DEFAULT));

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

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

   OCILobLocator *bfile_loc;
   ub4 len;

   /* Allocate the locator descriptor */
```

```
  (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &bfile_loc,
                            (ub4) OCI_DTYPE_FILE,
                            (size_t) 0, (dvoid **) 0);
  /* Select the bfile */
  selectLob(bfile_loc, errhp, svchp, stmthp);

  checkerr (errhp, OCILobFileOpen(svchp, errhp, bfile_loc,
                                  (ub1) OCI_FILE_READONLY));

  checkerr (errhp, OCILobGetLength(svchp, errhp, bfile_loc, &len));

  printf("Length of bfile = %d\n", len);

  checkerr (errhp, OCILobFileClose(svchp, errhp, bfile_loc));

  /* Free the locator descriptor */
  OCIDescriptorFree((dvoid *)bfile_loc, (ub4)OCI_DTYPE_FILE);
}
```

COBOL（Pro*COBOL）：BFILE の長さの取得

```
      * Getting the length of a BFILE. [Example script: 4034.pco]
        IDENTIFICATION DIVISION.
        PROGRAM-ID. BFILE-LENGTH.
        ENVIRONMENT DIVISION.
        DATA DIVISION.
        WORKING-STORAGE SECTION.

        01  USERID         PIC X(11) VALUES "SAMP/SAMP".
        01  BFILE1         SQL-BFILE.
        01  LEN            PIC S9(9) COMP.
        01  D-LEN          PIC 9(4).
        01  ORASLNRD       PIC 9(4).

            EXEC SQL INCLUDE SQLCA END-EXEC.
            EXEC ORACLE OPTION (ORACA=YES) END-EXEC.
            EXEC SQL INCLUDE ORACA END-EXEC.

        PROCEDURE DIVISION.
        BFILE-LENGTH.

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

          * Allocate and initialize the BFILE locator:
```

```
           EXEC SQL ALLOCATE :BFILE1 END-EXEC.
           EXEC SQL WHENEVER NOT FOUND GOTO END-OF-BFILE END-EXEC.
           EXEC SQL
                SELECT AD_GRAPHIC INTO :BFILE1
                FROM PRINT_MEDIA WHERE PRODUCT_ID = 3106
           END-EXEC.

      * Use LOB DESCRIBE to get length of lob:
           EXEC SQL
                LOB DESCRIBE :BFILE1 GET LENGTH INTO :LEN END-EXEC.

           MOVE LEN TO D-LEN.
           DISPLAY "Length of BFILE is ", D-LEN.

       END-OF-BFILE.
           EXEC SQL WHENEVER NOT FOUND CONTINUE END-EXEC.
           EXEC SQL FREE :BFILE1 END-EXEC.
           STOP RUN.

       SQL-ERROR.
           EXEC SQL WHENEVER SQLERROR CONTINUE END-EXEC.
           MOVE ORASLNR TO ORASLNRD.
           DISPLAY " ".
           DISPLAY "ORACLE ERROR DETECTED ON LINE ", ORASLNRD, ":".
           DISPLAY " ".
           DISPLAY SQLERRMC.
           EXEC SQL ROLLBACK WORK RELEASE END-EXEC.
           STOP RUN.
```

C/C++（Pro*C/C++）：BFILE の長さの取得

```
/* Getting the length of a BFILE. [Example script: 4035.pc] */

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

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

void getLengthBFILE_proc()
{
```

```
  OCIBFileLocator *Lob_loc;
  unsigned int Length = 0;

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  EXEC SQL ALLOCATE :Lob_loc;
  EXEC SQL SELECT PMtab.ad_graphic INTO :Lob_loc
           FROM Print_media PMtab
           WHERE PMtab.product_id = 3060 AND ad_id = 11001;
  /* Open the BFILE: */
  EXEC SQL LOB OPEN :Lob_loc READ ONLY;
  /* Get the Length: */
  EXEC SQL LOB DESCRIBE :Lob_loc GET LENGTH INTO :Length;
  /* If the BFILE is NULL or unitialized, then Length is Undefined: */
  printf("Length is %d bytes\n", Length);
  /* Close the BFILE: */
  EXEC SQL LOB CLOSE :Lob_loc;
  EXEC SQL FREE :Lob_loc;
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  getLengthBFILE_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

Visual Basic（OO4O）: BFILE の長さの取得

```
'Getting the length of a BFILE. [Example script: 4037.txt]
'The PL/SQL packages and the tables mentioned here are not part of the ' 'standard
OO4O installation:

Dim MySession As OraSession
Dim OraDb As OraDatabase

Set MySession = CreateObject("OracleInProcServer.XOraSession")
Set OraDb = MySession.OpenDatabase("pmschema", "pm/pm", 0&)

OraDb.Connection.BeginTrans

Set OraParameters = OraDb.Parameters

OraParameters.Add "id", 2056, ORAPARM_INPUT

'Define out parameter of BFILE type:
OraParameters.Add "AdGraphic", Null, ORAPARM_OUTPUT
```

```
OraParameters("MyAdGraphic").ServerType = ORATYPE_BFILE

Set OraSql =
   OraDb.CreateSql(
      "BEGIN SELECT ad_graphic INTO :MyAdGraphic FROM Print_media WHERE product_id =
:id;
         END;", ORASQL_FAILEXEC)

Set OraAdGraphic = OraParameters("MyAdGraphic").Value

If OraAdGraphic.Size = 0 Then
    MsgBox "BFile size is 0"
Else
    MsgBox "BFile size is " & OraAdGraphic.Size
End If
OraDb.Connection.CommitTrans
```

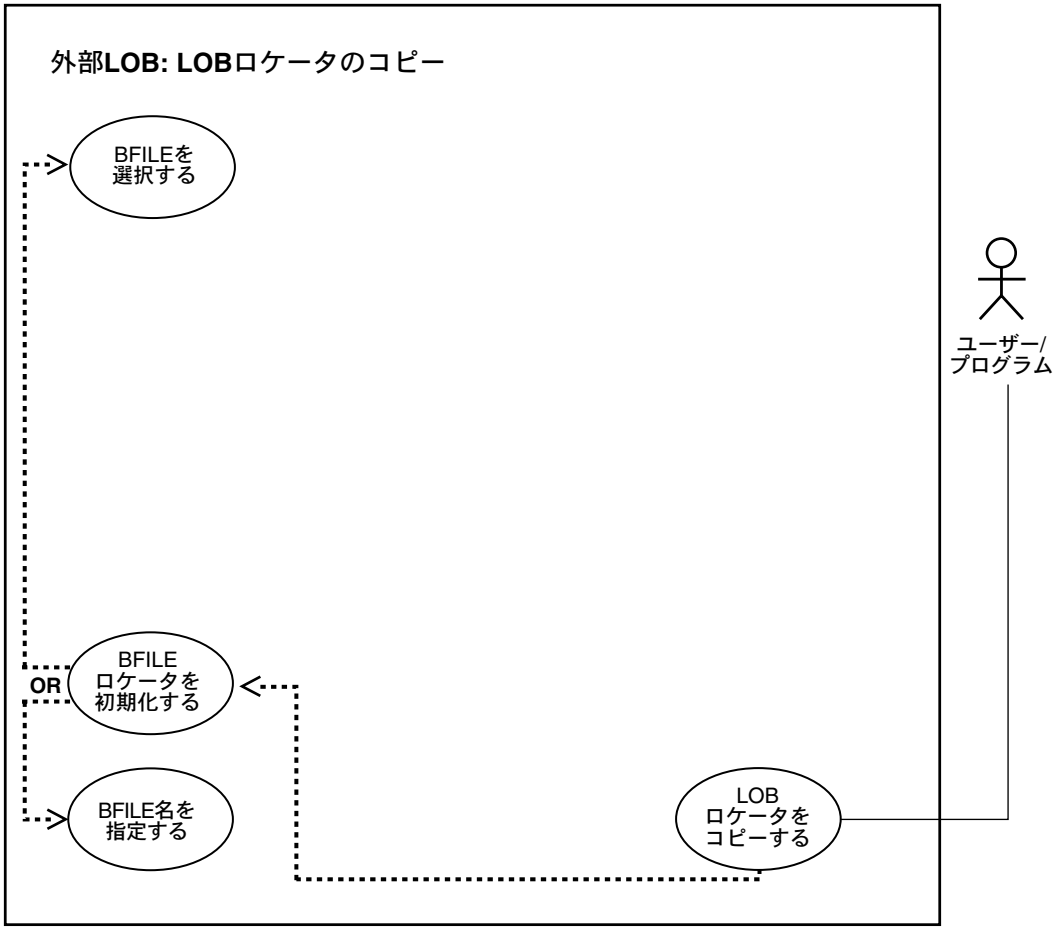
Java（JDBC）: BFILE の長さの取得

```
// Getting the length of a BFILE. [Example script: 4038.java]

import java.io.InputStream;
import java.io.OutputStream;

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

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

public class Ex4_74
{

  static final int MAXBUFSIZE = 32767;
  public static void main (String args [])
       throws Exception
  {
    // Load the Oracle JDBC driver:
    DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
```

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

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

    // Create a Statement:
    Statement stmt = conn.createStatement ();
    try
    {
       BFILE lob_loc = null;

       ResultSet rset = stmt.executeQuery (
          "SELECT ad_graphic FROM Print_media
             WHERE product_id = 3106 AND ad_id = 13001");
       if (rset.next())
       {
          lob_loc = ((OracleResultSet)rset).getBFILE (1);
       }

       // See if the BFILE exists:
       System.out.println("Result from fileExists(): " + lob_loc.fileExists());

       // Return the length of the BFILE:
       long length = lob_loc.length();
       System.out.println("Length of BFILE: " + length);

       // Get the directory alias for this BFILE:
       System.out.println("Directory alias: " + lob_loc.getDirAlias());

       // Get the file name for this BFILE:
       System.out.println("File name: " + lob_loc.getName());

       stmt.close();
       conn.commit();
       conn.close();

    }
    catch (SQLException e)
    {
       e.printStackTrace();
    }
  }
}
```

BFILE の LOB ロケータのコピー

図 12-24 利用図 : BFILE の LOB ロケータのコピー



参照 : 内部一時 LOB に関するすべての基本操作については、12-2 ページの表 12-1「利用モデル図 : 外部 LOB（BFILE）」を参照してください。

用途

BFILE 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関数またはファンクションのリストは、第 3 章「様々なプログラム環境での LOB のサポート」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:『Oracle9*i* SQL リファレンス』の第 14 章「SQL 文 : CREATE LIBRARY ～ CREATE SPFILE」の「CREATE PROCEDURE」
- PL/SQL（DBMS_LOB）: 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、第 5 章「ラージ・オブジェクト（LOB）: 詳細事項」
- C（OCI）:『Oracle Call Interface 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 16 章「その他の OCI リレーショナル関数」の「LOB 関数」の「OCILobLocatorAssign()」
- COBOL（Pro*COBOL）:『Pro*COBOL Precompiler プログラマーズ・ガイド』の LOB に関する詳細、LOB 文の使用上の注意および付録 F「埋込み SQL およびプリコンパイラ・ディレクティブ」の「LOB ASSIGN（実行可能埋込み SQL 拡張機能）」
- C/C++（Pro*C/C++）:『Pro*C/C++ Precompiler 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」および付録 F「埋込み SQL 文およびディレクティブ」の「LOB ASSIGN（実行可能埋込み SQL 拡張機能）」
- Visual Basic（OO4O）: 参照マニュアルはありません。
- Java（JDBC）:『Oracle9*i* JDBC 開発者ガイドおよびリファレンス』の第 8 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『Oracle9*i* 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「JDBC 2.0 LOB 型と Oracle 拡張型のサポート」の「BLOB、CLOB および BFILE のサポート」

使用例

次の例では、BFILE ロケータを、ad_graphic 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での例を示します。

- PL/SQL（DBMS_LOB）: BFILE の LOB ロケータのコピー（12-164 ページ）
- C（OCI）: BFILE の LOB ロケータのコピー（12-165 ページ）
- COBOL（Pro*COBOL）: BFILE の LOB ロケータのコピー （12-166 ページ）
- C/C++（Pro*C/C++）: BFILE の LOB ロケータのコピー（12-167 ページ）
- Visual Basic: 今回のリリースでは例は提供されません。
- Java（JDBC）: BFILE の LOB ロケータのコピー（12-168 ページ）

PL/SQL（DBMS_LOB）: BFILE 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、第 5 章「ラージ・オブジェクト（LOB）: 詳細事項」の「読込み一貫性のあるロケータ」を参照してください。

```
/* Copying a LOB locator for a BFILE. [Example script: 4039.sql] */
/* Procedure BFILEAssign_proc is not part of DBMS_LOB package:   */

CREATE OR REPLACE PROCEDURE BFILEAssign_proc IS
   File_loc1    BFILE;
   File_loc2    BFILE;
BEGIN
   SELECT Photo INTO File_loc1 FROM print_media
      WHERE Product_ID = 3060 AND ad_id = 11001 FOR UPDATE;
   /* Assign File_loc1 to File_loc2 so that they both refer to the same
operating
      system file: */
   File_loc2 := File_loc1;
   /* Now you can read the bfile from either File_loc1 or File_loc2. */
END;
```

C（OCI）：BFILE の LOB ロケータのコピー

```
/* Copying a LOB locator for a BFILE. [Example script: 4040.c] */
/* Select the lob/bfile from the Print_media table            */

void selectLob(Lob_loc, errhp, svchp, stmthp)
OCILobLocator *Lob_loc;
OCIError      *errhp;
OCISvcCtx     *svchp;
OCIStmt       *stmthp;
{
     OCIDefine *dfnhp, *dfnhp2;
     text *selstmt = (text *) "SELECT ad_graphic FROM Print_media \
                                WHERE product_id = 3106 AND ad_id = 13001";

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

     /* Call define for the bfile column */
     checkerr (errhp, OCIDefineByPos(stmthp, &dfnhp, errhp, 1,
                                     (dvoid *)&Lob_loc, 0 , SQLT_BFILE,
                                     (dvoid *)0, (ub2 *)0, (ub2 *)0,
                                     OCI_DEFAULT)
     || OCIDefineByPos(stmthp, &dfnhp2, errhp, 2,
                                     (dvoid *)&Lob_loc, 0 , SQLT_BFILE,
                                     (dvoid *)0, (ub2 *)0, (ub2 *)0,
                                     OCI_DEFAULT));

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

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

   OCILobLocator *src_loc;
   OCILobLocator *dest_loc;

   /* Allocate the locator descriptors */
```

```
    (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &src_loc,
                              (ub4) OCI_DTYPE_FILE,
                              (size_t) 0, (dvoid **) 0);

    (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &dest_loc,
                              (ub4) OCI_DTYPE_FILE,
                              (size_t) 0, (dvoid **) 0);

    /* Select the bfile */
    selectLob(src_loc, errhp, svchp, stmthp);

    checkerr(errhp, OCILobLocatorAssign(svchp, errhp, src_loc, &dest_loc));

    /* Free the locator descriptor */
    OCIDescriptorFree((dvoid *)src_loc, (ub4)OCI_DTYPE_FILE);
    OCIDescriptorFree((dvoid *)dest_loc, (ub4)OCI_DTYPE_FILE);
}
```

COBOL（Pro*COBOL）: BFILE の LOB ロケータのコピー

```
* Copying a LOB locator for a BFILE. [Example script: 4041.pco]
  IDENTIFICATION DIVISION.
  PROGRAM-ID. BFILE-COPY-LOCATOR.
  ENVIRONMENT DIVISION.
  DATA DIVISION.
  WORKING-STORAGE SECTION.

  01  USERID          PIC X(11) VALUES "SAMP/SAMP".
  01  BFILE1          SQL-BFILE.
  01  BFILE2          SQL-BFILE.
  01  ORASLNRD        PIC 9(4).

      EXEC SQL INCLUDE SQLCA END-EXEC.
      EXEC ORACLE OPTION (ORACA=YES) END-EXEC.
      EXEC SQL INCLUDE ORACA END-EXEC.

  PROCEDURE DIVISION.
  BILFE-COPY-LOCATOR.

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

   * Allocate and initialize the BFILE locator:
      EXEC SQL ALLOCATE :BFILE1 END-EXEC.
      EXEC SQL ALLOCATE :BFILE2 END-EXEC.
```

```
            EXEC SQL WHENEVER NOT FOUND GOTO END-OF-BFILE END-EXEC.
            EXEC SQL
                 SELECT AD_GRAPHIC INTO :BFILE1
                 FROM PRINT_MEDIA WHERE PRODUCT_ID = 3106
                 AND AD_ID = 13001 END-EXEC.
            EXEC SQLLOB ASSIGN :BFILE1 TO :BFILE2 END-EXEC.

        END-OF-BFILE.
            EXEC SQL WHENEVER NOT FOUND CONTINUE END-EXEC.
            EXEC SQL FREE :BFILE1 END-EXEC.
            EXEC SQL FREE :BFILE2 END-EXEC.
            STOP RUN.

        SQL-ERROR.
            EXEC SQL WHENEVER SQLERROR CONTINUE END-EXEC.
            MOVE ORASLNR TO ORASLNRD.
            DISPLAY " ".
            DISPLAY "ORACLE ERROR DETECTED ON LINE ", ORASLNRD, ":".
            DISPLAY " ".
            DISPLAY SQLERRMC.
            EXEC SQL ROLLBACK WORK RELEASE END-EXEC.
            STOP RUN.
```

C/C++（Pro*C/C++）：BFILE の LOB ロケータのコピー

```
/* Copying a LOB locator for a BFILE. [Example script: 4042.pc] */

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

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

void BFILEAssign_proc()
{
  OCIBFileLocator *Lob_loc1, *Lob_loc2;

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  EXEC SQL ALLOCATE :Lob_loc1;
  EXEC SQL ALLOCATE :Lob_loc2;
```

```
  EXEC SQL SELECT ad_graphic INTO :Lob_loc1
           FROM Print_media WHERE product_id = 2056 AND ad_id = 12001;
  /* Assign Lob_loc1 to Lob_loc2 so that they both refer to the same
     operating system file:  */
  EXEC SQL LOB ASSIGN :Lob_loc1 TO :Lob_loc2;
  /* Now you can read the BFILE from either Lob_loc1 or Lob_loc2 */
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  BFILEAssign_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

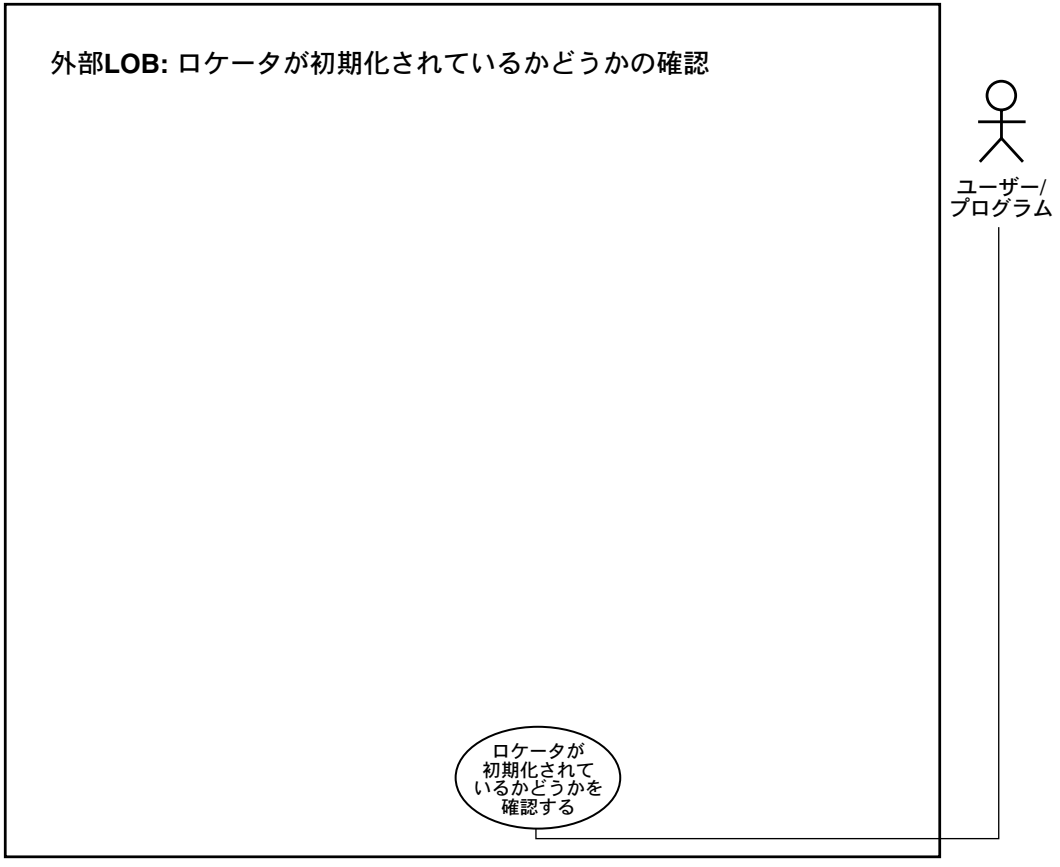
Java（JDBC）: BFILE の LOB ロケータのコピー

```
// Copying a LOB locator for a BFILE. [Example script: 4044.java]

import java.sql.DriverManager;
import java.sql.Connection;
import java.sql.Types;
import java.sql.Statement;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

// Oracle Specific JDBC classes:
import oracle.sql.*;
import oracle.jdbc.driver.*;
public class Ex4_81
{
  public static void main (String args [])
       throws Exception
  {
    // Load the Oracle JDBC driver:
    DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

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

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

```
    // Create a Statement:
    Statement stmt = conn.createStatement ();
    try
    {
       BFILE lob_loc1 = null;
       BFILE lob_loc2 = null;

       ResultSet rset = stmt.executeQuery (
          "SELECT ad_graphic FROM Print_media
                WHERE product_id = 3106 AND ad_id = 13001");
       if (rset.next())
       {
          lob_loc1 = ((OracleResultSet)rset).getBFILE (1);
       }

       // Assign lob_loc1 to lob_loc2 so that they both refer
       // to the same operating system file.
       // Now the BFILE can be read through either of the locators:
       lob_loc2 = lob_loc1;
       stmt.close();
       conn.commit();
       conn.close();
  }
    //catch (SQLException e)
    catch (Exception e)
    {
        e.printStackTrace();
    }
  }
}
```

BFILE の LOB ロケータが初期化されているかどうかの確認

図 12-25 利用図 : LOB ロケータが初期化されているかどうかの確認



参照 : 内部一時 LOB に関するすべての基本操作については、12-2 ページの表 12-1「利用モデル図 : 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、ある関数またはファンクションから別の関数またはファンクション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関数またはファンクションのリストは、第 3 章「様々なプログラム環境での LOB のサポート」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）:『Oracle Call Interface 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 16 章「その他の 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++）:『Pro*C/C++ Precompiler 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」、付録 F「埋込み SQL 文およびディレクティブ」、および『Oracle Call Interface プログラマーズ・ガイド』の第 7 章「LOB および FILE 操作」、第 16 章「その他の 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での例を示します。

- PL/SQL（DBMS_LOB）: 今回のリリースでは例は提供されません。
- 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（12-172 ページ）

- COBOL（Pro*COBOL）: 今回のリリースでは例は提供されません。
- 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（12-173 ページ）
- Visual Basic（OO4O）: 今回のリリースでは例は提供されません。
- Java（JDBC）: 今回のリリースでは例は提供されません。

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

```
/* Determining if a LOB locator  for a BFILE is initialized.
   [Example script: 4045.c]*/

/* Select the lob/bfile from the Print_media table */
void selectLob(Lob_loc, errhp, svchp, stmthp)
OCILobLocator *Lob_loc;
OCIError      *errhp;
OCISvcCtx     *svchp;
OCIStmt       *stmthp;
{
     OCIDefine *dfnhp, *dfnhp2;
     text *selstmt = (text *) "SELECT ad_graphic FROM Print_media \
                                WHERE product_id = 3106 AND ad_id = 13001";

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

     /* Call define for the bfile column */
     checkerr (errhp, OCIDefineByPos(stmthp, &dfnhp, errhp, 1,
                                     (dvoid *)&Lob_loc, 0 , SQLT_BFILE,
                                     (dvoid *)0, (ub2 *)0, (ub2 *)0,
                                     OCI_DEFAULT)
     || OCIDefineByPos(stmthp, &dfnhp2, errhp, 2,
                                     (dvoid *)&Lob_loc, 0 , SQLT_BFILE,
                                     (dvoid *)0, (ub2 *)0, (ub2 *)0,
                                     OCI_DEFAULT));

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

void BfileIsInit(envhp, errhp, svchp, stmthp)
OCIEnv       *envhp;
```

```
OCIError     *errhp;
OCISvcCtx    *svchp;
OCIStmt      *stmthp;
{

   OCILobLocator *bfile_loc;
   boolean is_init;

   /* Allocate the locator descriptors */
   (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &bfile_loc,
                             (ub4) OCI_DTYPE_FILE,
                             (size_t) 0, (dvoid **) 0);
   /* Select the bfile */
   selectLob(bfile_loc, errhp, svchp, stmthp);

   checkerr(errhp, OCILobLocatorIsInit(envhp, errhp, bfile_loc, &is_init));

   if (is_init == TRUE)
   {
     printf("Locator is initialized\n");
   }
   else
   {
     printf("Locator is not initialized\n");
   }
   /* Free the locator descriptor */
   OCIDescriptorFree((dvoid *)bfile_loc, (ub4)OCI_DTYPE_FILE);
}
```

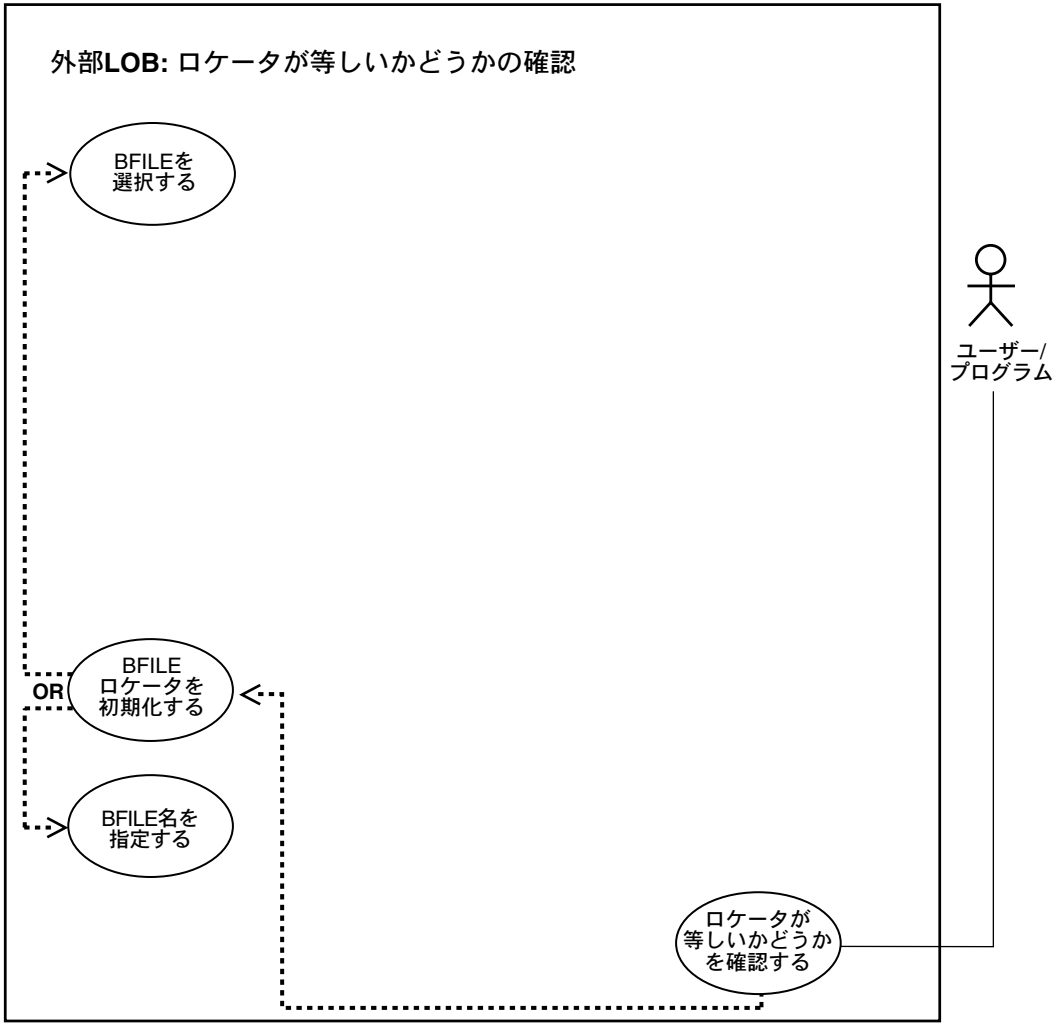
C/C++（Pro*C/C++）: BFILE の LOB ロケータが初期化されているかどうかの確認

```
/* Determining if a LOB locator  for a BFILE is initialized.
   [Example script: 4046.pc]
   Pro*C/C++ has no form of embedded SQL statement to determine if a BFILE
   locator is initialized. Locators in Pro*C/C++ are initialized when they
   are allocated with the EXEC SQL ALLOCATE statement. However, an example
   can be written that uses embedded SQL and the OCI as shown here: */

#include <sql2oci.h>
#include <stdio.h>
#include <sqlca.h>

void Sample_Error()
{
```

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

void BFILELocatorIsInit_proc()
{
  OCIBFileLocator *Lob_loc;
  OCIEnv *oeh;
  OCIError *err;
  boolean isInitialized = 0;

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  EXEC SQL ALLOCATE :Lob_loc;
  EXEC SQL SELECT Mtab.ad_graphic INTO :Lob_loc
           FROM Print_media PMtab
           WHERE PMtab.product_id = 2056 AND ad_id = 12001;
  /* Get the OCI Environment Handle using a SQLLIB Routine: */
  (void) SQLEnvGet(SQL_SINGLE_RCTX, &oeh);
  /* Allocate the OCI Error Handle: */
  (void) OCIHandleAlloc((dvoid *)oeh, (dvoid **)&err,
                        (ub4)OCI_HTYPE_ERROR, (ub4)0, (dvoid **)0);
  /* Use the OCI to determine if the locator is Initialized: */
  (void) OCILobLocatorIsInit(oeh, err, Lob_loc, &isInitialized);
  if (isInitialized)
    printf("Locator is initialized\n");
  else
    printf("Locator is not initialized\n");
  /* Note that in this example, the locator is initialized: */
  /* Deallocate the OCI Error Handle: */
  (void) OCIHandleFree(err, OCI_HTYPE_ERROR);
  /* Release resources held by the locator: */
  EXEC SQL FREE :Lob_loc;
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  BFILELocatorIsInit_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

BFILE の 2 つの LOB ロケータが等しいかどうかの確認

図 12-26 利用図 : BFILE の 2 つの LOB ロケータが他と等しいかどうかの確認



参照 : 内部一時 LOB に関するすべての基本操作については、12-2 ページの表 12-1「利用モデル図 : 外部 LOB（BFILE）」を参照してください。

用途

BFILE の 2 つの LOB 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関数またはファンクションのリストは、第 3 章「様々なプログラム環境での LOB のサポート」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）:『Oracle Call Interface 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 16 章「その他の 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++）:『Pro*C/C++ Precompiler 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」、付録 F「埋込み SQL 文およびディレクティブ」の「LOB ASSIGN（実行可能埋込み SQL 拡張機能）」、および『Oracle Call Interface プログラマーズ・ガイド』の第 7 章「LOB および FILE 操作」、第 16 章「その他の 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）:『Oracle9*i* JDBC 開発者ガイドおよびリファレンス』の第 8 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『Oracle9*i* 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「JDBC 2.0 LOB 型と Oracle 拡張型のサポート」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（第 5 章「ラージ・オブジェクト（LOB）: 詳細事項」の「読込み一貫性のあるロケータ」を参照）。

例

次のプログラム環境での例を示します。

- PL/SQL（DBMS_LOB）: 今回のリリースでは例は提供されません。
- C（OCI）: BFILE の 2 つの LOB ロケータが等しいかどうかの確認（12-177 ページ）
- COBOL（Pro*COBOL）: 今回のリリースでは例は提供されません。

C（OCI）: BFILE の 2 つの LOB ロケータが等しいかどうかの確認

```
/* Determining if one LOB locator for a BFILE is equal to another */
/* [Example script: 4047.c]                                       */

/* Select the lob/bfile from the Print_media table */
void selectLob(Lob_loc, errhp, svchp, stmthp)
OCILobLocator *Lob_loc;
OCIError      *errhp;
OCISvcCtx     *svchp;
OCIStmt       *stmthp;
{
     OCIDefine *dfnhp, *dfnhp2;
     text *selstmt = (text *) "SELECT ad_graphic FROM Print_media \
                               WHERE product_id = 3106 AND ad_id = 13001";

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

     /* Call define for the bfile column */
     checkerr (errhp, OCIDefineByPos(stmthp, &dfnhp, errhp, 1,
                                     (dvoid *)&Lob_loc, 0 , SQLT_BFILE,
                                     (dvoid *)0, (ub2 *)0, (ub2 *)0,
                                     OCI_DEFAULT)
     ||  OCIDefineByPos(stmthp, &dfnhp2, errhp, 2,
                                     (dvoid *)&Lob_loc, 0 , SQLT_BFILE,
                                     (dvoid *)0, (ub2 *)0, (ub2 *)0,
                                     OCI_DEFAULT));

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

void BfileIsEqual(envhp, errhp, svchp, stmthp)
OCIEnv       *envhp;
OCIError     *errhp;
```

```
OCISvcCtx    *svchp;
OCIStmt      *stmthp;
{

   OCILobLocator *bfile_loc1;
   OCILobLocator *bfile_loc2;
   boolean  is_equal;

   /* Allocate the locator descriptors */
   (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &bfile_loc1,
                             (ub4) OCI_DTYPE_FILE,
                             (size_t) 0, (dvoid **) 0);

   (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &bfile_loc2,
                             (ub4) OCI_DTYPE_FILE,
                             (size_t) 0, (dvoid **) 0);

   /* Select the bfile */
   selectLob(bfile_loc1, errhp, svchp, stmthp);

   checkerr(errhp,
            OCILobLocatorAssign(svchp, errhp, bfile_loc1, &bfile_loc2));

   checkerr(errhp, OCILobIsEqual(envhp, bfile_loc1, bfile_loc2, &is_equal));

   if (is_equal == TRUE)
   {
     printf("Locators are equal\n");
   }
   else
   {
     printf("Locators are not equal\n");
   }

   /* Free the locator descriptor */
   OCIDescriptorFree((dvoid *)bfile_loc1, (ub4)OCI_DTYPE_FILE);
   OCIDescriptorFree((dvoid *)bfile_loc2, (ub4)OCI_DTYPE_FILE);
}
```

C/C++（Pro*C/C++）: BFILE の 2 つの LOB ロケータが等しいかどうかの確認

```
/* Determining if one LOB locator for a BFILE is equal to another */
   [Example script: 4048.pc]
   Pro*C/C++ does not provide a mechanism to test the equality of two
   locators However, by using the OCI directly, two locators can be
   compared to determine whether or not they are equal as this example
   demonstrates: */

#include <sql2oci.h>
#include <stdio.h>
#include <sqlca.h>

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

void BFILELocatorIsEqual_proc()
{
  OCIBFileLocator *Lob_loc1, *Lob_loc2;
  OCIEnv *oeh;
  boolean isEqual = 0;

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  EXEC SQL ALLOCATE :Lob_loc1;
  EXEC SQL ALLOCATE :Lob_loc2;
  EXEC SQL SELECT ad_graphic INTO :Lob_loc1
           FROM Print_media WHERE product_id = 2056 AND ad_id = 12001;
  EXEC SQL LOB ASSIGN :Lob_loc1 TO :Lob_loc2;
  /* Now you can read the BFILE from either Lob_loc1 or Lob_loc2 */
  /* Get the OCI Environment Handle using a SQLLIB Routine: */
  (void) SQLEnvGet(SQL_SINGLE_RCTX, &oeh);
  /* Call OCI to see if the two locators are Equal: */
  (void) OCILobIsEqual(oeh, Lob_loc1, Lob_loc2, &isEqual);
  if (isEqual)
    printf("Locators are equal\n");
  else
    printf("Locators are not equal\n");
  /* Note that in this example, the LOB locators will be Equal: */
  EXEC SQL FREE :Lob_loc1;
  EXEC SQL FREE :Lob_loc2;
}
```

```
void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  BFILELocatorIsEqual_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

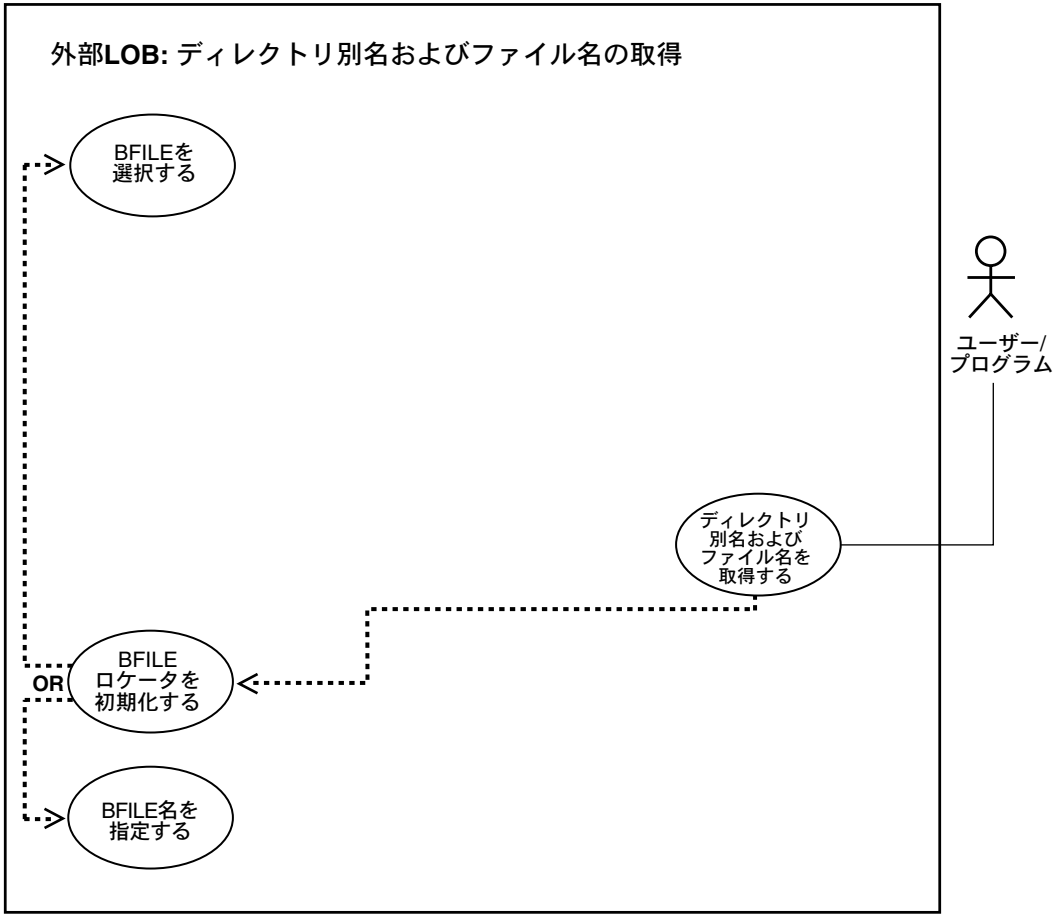
Java（JDBC）: BFILE の 2 つの LOB ロケータが等しいかどうかの確認

```
// Determining if one LOB locator for a BFILE is equal to another
// [Example script: 4050.java]

import java.sql.Connection;
import java.sql.Types;
import java.sql.Statement;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

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

public class Ex4_89
{

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

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

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

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

    try
    {
       BFILE lob_loc1 = null;
```

```
        BFILE lob_loc2 = null;

        ResultSet rset = stmt.executeQuery (
           "SELECT ad_graphic FROM Print_media
                   WHERE product_id = 3106 AND ad_id = 13001");
       if (rset.next())
       {
           lob_loc1 = ((OracleResultSet)rset).getBFILE (1);
       }

       // Set both LOBS to reference the same BFILE:
       lob_loc2 = lob_loc1;

       // Note that in this example, the Locators will be equal:
       if (lob_loc1.equals(lob_loc2))
       {
          // The Locators are equal:
          System.out.println("The BFILEs are equal");
       }
       else
       {
          // The Locators are different:
          System.out.println("The BFILEs are NOT equal");
       }

       stmt.close();
       conn.commit();
       conn.close();

     }
     catch (SQLException e)
     {
        e.printStackTrace();
     }
   }
}
```

ディレクトリ別名およびファイル名の取得

図 12-27 利用図 : ディレクトリ別名およびファイル名の取得



参照 : 内部一時 LOB に関するすべての基本操作については、12-2 ページの表 12-1「利用モデル図 : 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関数またはファンクションのリストは、第 3 章「様々なプログラム環境での LOB のサポート」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）：『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』の第 23 章「DBMS_LOB」の「DBMS_LOB サブプログラムの要約」の「FILEGETNAME プロシージャ」
- C（OCI）：『Oracle Call Interface 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 16 章「その他の OCI リレーショナル関数」の「LOB 関数」の「OCILobFileGetName()」
- COBOL（Pro*COBOL）：『Pro*COBOL Precompiler プログラマーズ・ガイド』の LOB に関する詳細、LOB 文の使用上の注意および付録 F「埋込み SQL およびプリコンパイラ・ディレクティブ」の「LOB DESCRIBE（実行可能埋込み SQL 拡張機能）」
- C/C++（Pro*C/C++）：『Pro*C/C++ Precompiler 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」および付録 F「埋込み SQL 文およびディレクティブ」の「LOB DESCRIBE（実行可能埋込み SQL 拡張機能）」
- Java （JDBC）：『Oracle9*i* JDBC 開発者ガイドおよびリファレンス』の第 8 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『Oracle9*i* 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「JDBC 2.0 LOB 型と Oracle 拡張型のサポート」の「BLOB、CLOB および BFILE のサポート」

使用例

次の例では、BFILE の `ad_graphic` 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での例を示します。

PL/SQL（DBMS_LOB）：ディレクトリ別名およびファイル名の取得

```
/* Getting the directory alias and filename [Example script: 4051.sql] */

CREATE OR REPLACE PROCEDURE getNameBFILE_proc IS
   File_loc         BFILE;
   DirAlias_name    VARCHAR2(30);
   File_name        VARCHAR2(40);
BEGIN
   SELECT ad_graphic INTO File_loc FROM Print_media
         WHERE product_id = 3060 AND ad_id = 11001;
   DBMS_LOB.FILEGETNAME(File_loc, DirAlias_name, File_name);
   /* do some processing based on the directory alias and file names */
END;
```

C（OCI）：ディレクトリ別名およびファイル名の取得

```
/* Getting the directory alias and filename [Example script: 4052.c]  */

/* Select the lob/bfile from the Print_media table */
void selectLob(Lob_loc, errhp, svchp, stmthp)
OCILobLocator *Lob_loc;
OCIError      *errhp;
OCISvcCtx     *svchp;
OCIStmt       *stmthp;
{
     OCIDefine *dfnhp, *dfnhp2;
     text *selstmt = (text *) "SELECT ad_graphic FROM Print_media \
                                   WHERE product_id = 3106";
```

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

    /* Call define for the bfile column */
    checkerr (errhp, OCIDefineByPos(stmthp, &dfnhp, errhp, 1,
                                    (dvoid *)&Lob_loc, 0 , SQLT_BFILE,
                                    (dvoid *)0, (ub2 *)0, (ub2 *)0,
                                    OCI_DEFAULT)
    ||  OCIDefineByPos(stmthp, &dfnhp2, errhp, 2,
                                    (dvoid *)&Lob_loc, 0 , SQLT_BFILE,
                                    (dvoid *)0, (ub2 *)0, (ub2 *)0,
                                    OCI_DEFAULT));

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

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

   OCILobLocator *bfile_loc;
   OraText dir_alias[32] = NULL;
   OraText filename[256]  = NULL;
   ub2 d_length = 32;
   ub2 f_length = 256;

   /* Allocate the locator descriptors */
   (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &bfile_loc,
                             (ub4) OCI_DTYPE_FILE,
                             (size_t) 0, (dvoid **) 0);

   /* Select the bfile */
   selectLob(bfile_loc, errhp, svchp, stmthp);

   checkerr(errhp, OCILobFileGetName(envhp, errhp, bfile_loc,
                       dir_alias, &d_length, filename, &f_length));
```

```
    printf("Directory Alias : [%s]\n", dir_alias);
    printf("File name : [%s]\n", filename);

    /* Free the locator descriptor */
    OCIDescriptorFree((dvoid *)bfile_loc, (ub4)OCI_DTYPE_FILE);
}
```

COBOL（Pro*COBOL）：ディレクトリ別名およびファイル名の取得

```
* Getting the directory alias and filename [Example script: 4053.pco]
 IDENTIFICATION DIVISION.
 PROGRAM-ID. BFILE-DIR-ALIAS.
 ENVIRONMENT DIVISION.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 01  USERID          PIC X(11) VALUES "SAMP/SAMP".
 01  BFILE1          SQL-BFILE.
 01  DIR-ALIAS       PIC X(30) VARYING.
 01  FNAME           PIC X(30) VARYING.
 01  ORASLNRD        PIC 9(4).

     EXEC SQL INCLUDE SQLCA END-EXEC.
     EXEC ORACLE OPTION (ORACA=YES) END-EXEC.
     EXEC SQL INCLUDE ORACA END-EXEC.

 PROCEDURE DIVISION.
 BFILE-DIR-ALIAS.

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

* Allocate and initialize the BFILE locator:
     EXEC SQL ALLOCATE :BFILE1 END-EXEC.
     EXEC SQL WHENEVER NOT FOUND GOTO END-OF-BFILE END-EXEC.

* Populate the BFILE locator:
     EXEC SQL
          SELECT AD_GRAPHIC INTO :BFILE1
          FROM PRINT_MEDIA WHERE PRODUCT_ID = 3106 AND AD_ID = 13001
     END-EXEC.
```

```
      * Use the LOB DESCRIBE functionality to get
      * the directory alias and the filename:
           EXEC SQL LOB DESCRIBE :BFILE1
                GET DIRECTORY, FILENAME INTO :DIR-ALIAS, :FNAME END-EXEC.

           DISPLAY "DIRECTORY: ", DIR-ALIAS-ARR, "FNAME: ", FNAME-ARR.
       END-OF-BFILE.
           EXEC SQL WHENEVER NOT FOUND CONTINUE END-EXEC.
           EXEC SQL FREE :BFILE1 END-EXEC.
           STOP RUN.
       SQL-ERROR.
           EXEC SQL WHENEVER SQLERROR CONTINUE END-EXEC.
           MOVE ORASLNR TO ORASLNRD.
           DISPLAY " ".
           DISPLAY "ORACLE ERROR DETECTED ON LINE ", ORASLNRD, ":".
           DISPLAY " ".
           DISPLAY SQLERRMC.
           EXEC SQL ROLLBACK WORK RELEASE END-EXEC.
           STOP RUN.
```

C/C++（Pro*C/C++）：ディレクトリ別名およびファイル名の取得

```
/* Getting the directory alias and filename [Example script: 4054.pc] */

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

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

void getBFILEDirectoryAndFilename_proc()
{
  OCIBFileLocator *Lob_loc;
  char Directory[31], Filename[255];
  /* Datatype Equivalencing is Optional: */
  EXEC SQL VAR Directory IS STRING;
  EXEC SQL VAR Filename IS STRING;
  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  EXEC SQL ALLOCATE :Lob_loc;
```

```
  /* Select the BFILE: */
  EXEC SQL SELECT ad_graphic INTO :Lob_loc
     FROM print_media WHERE product_id = 2056 AND ad_id = 12001;
  /* Open the BFILE: */
  EXEC SQL LOB OPEN :Lob_loc READ ONLY;
  /* Get the Directory Alias and Filename: */
  EXEC SQL LOB DESCRIBE :Lob_loc
     GET DIRECTORY, FILENAME INTO :Directory, :Filename;

  /* Close the BFILE: */
  EXEC SQL LOB CLOSE :Lob_loc;
  printf("Directory Alias: %s\n", Directory);
  printf("Filename: %s\n", Filename);
  /* Release resources held by the locator: */
  EXEC SQL FREE :Lob_loc;
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  getBFILEDirectoryAndFilename_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

Visual Basic（OO4O）：ディレクトリ別名およびファイル名の取得

```
'Getting the directory alias and filename [Example script: 4056.txt]
'The PL/SQL packages and tables mentioned here are not part of the
'standard OO4O installation:

Dim MySession As OraSession
Dim OraDb As OraDatabase
Dim OraAdGraphic1 As OraBfile, OraSql As OraSqlStmt

Set MySession = CreateObject("OracleInProcServer.XOraSession")
Set OraDb = MySession.OpenDatabase("pmschema", "pm/pm", 0&)
OraDb.Connection.BeginTrans
Set OraParameters = OraDb.Parameters
OraParameters.Add "id", 2056, ORAPARM_INPUT

'Define out parameter of BFILE type:
OraParameters.Add "MyAdGraphic", Null, ORAPARM_OUTPUT
OraParameters("MyAdGraphic").ServerType = ORATYPE_BFILE
```

```
Set OraSql =
   OraDb.CreateSql(
      "BEGIN SELECT ad_graphic INTO :MyAdGraphic FROM Print_media
         WHERE product_id = :id;
         END;", ORASQL_FAILEXEC)

Set OraAdGraphic1 = OraParameters("MyAdGraphic").Value
'Get Directory alias and filename:
MsgBox " Directory alias is " & OraAdGraphic1.DirectoryName &
   " Filename is " & OraAdGraphic1.filename

OraDb.Connection.CommitTrans
```

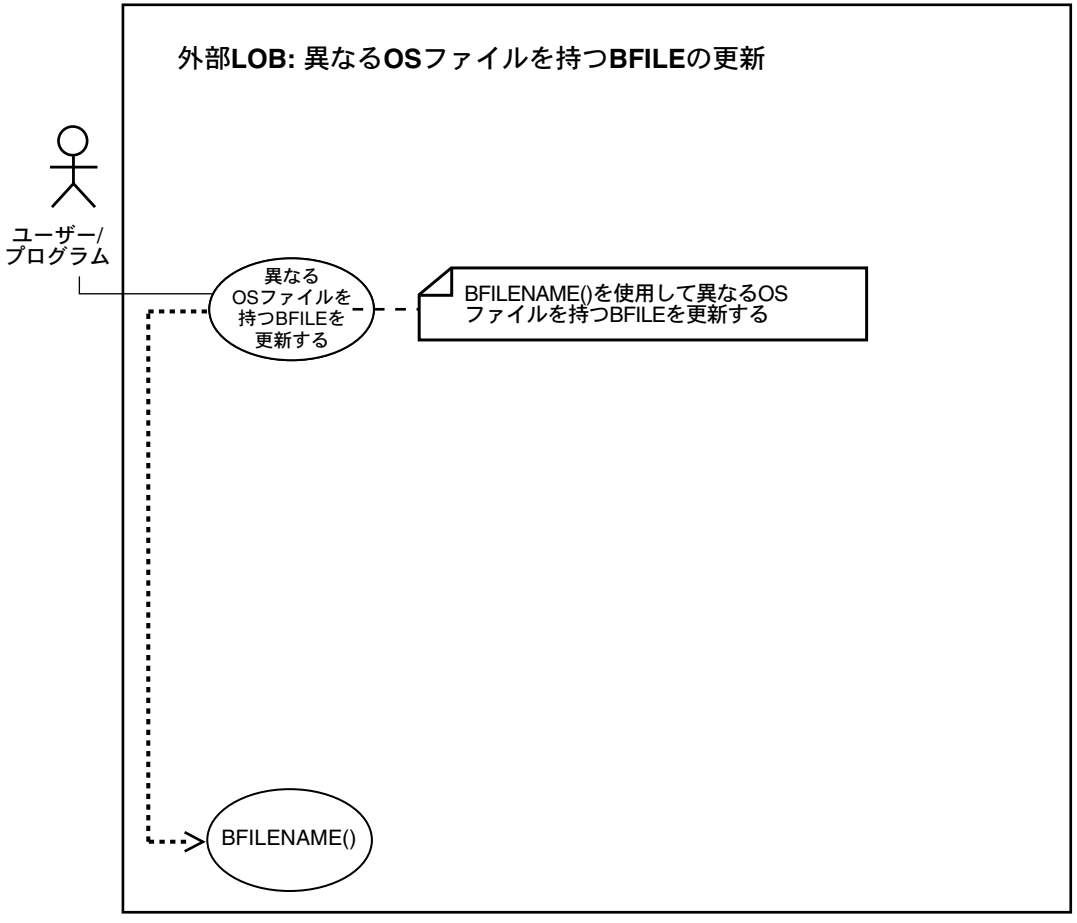
Java（JDBC）：ディレクトリ別名およびファイル名の取得

```
// Getting the directory alias and filename [Example script: 4057.java]

import java.io.InputStream;
import java.io.OutputStream;
// Core JDBC classes:
import java.sql.DriverManager;
import java.sql.Connection;
import java.sql.Types;
import java.sql.Statement;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
// Oracle Specific JDBC classes:
import oracle.sql.*;
import oracle.jdbc.driver.*;
public class Ex4_74
{
  static final int MAXBUFSIZE = 32767;
  public static void main (String args [])
       throws Exception
  {
    // Load the Oracle JDBC driver:
    DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

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

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

```
    // Create a Statement:
    Statement stmt = conn.createStatement ();
    try
    {
       BFILE lob_loc = null;
       ResultSet rset = stmt.executeQuery (
          "SELECT ad_graphic FROM Print_media
               WHERE product_id = 3106 AND ad_id = 13001");
       if (rset.next())
       {
          lob_loc = ((OracleResultSet)rset).getBFILE (1);
       }
       // See if the BFILE exists:
       System.out.println("Result from fileExists(): " + lob_loc.fileExists());

       // Return the length of the BFILE:
       long length = lob_loc.length();
       System.out.println("Length of BFILE: " + length);

       // Get the directory alias for this BFILE:
       System.out.println("Directory alias: " + lob_loc.getDirAlias());

       // Get the file name for this BFILE:
       System.out.println("File name: " + lob_loc.getName());
       stmt.close();
       conn.commit();
       conn.close();
    }
    catch (SQLException e)
    {
        e.printStackTrace();
    }
  }
}
```

BFILENAME() を使用した BFILE の更新

図 12-28 利用図 : BFILENAME() を使用した BFILE の更新



参照 :

- 12-2 ページの表 12-1「利用モデル図 : 外部 LOB（BFILE）」を参照してください。
- 12-194 ページの「別の表からの BFILE の選択による BFILE の更新」を参照してください。
- 12-196 ページの「初期化した BFILE ロケータを使用した BFILE の更新」を参照してください。

使用上の注意

BFILENAME() ファンクション BFILENAME() ファンクションは、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。

このファンクションへの directory_alias パラメータで表されるディレクトリ・オブジェクトは、BFILENAME() ファンクションが 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 OS 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、このディレクトリ・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、ディレクトリ・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() ファンクションによって初期化された BFILE ロケータを使用するファイル・アクセス時にかぎり実行されます。

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が存在しない場合は、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() ファンクションの 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場合の詳細は、12-7 ページの「ディレクトリ名の指定」を参照し、同等の OCI ベースのルーチンの詳細は、『Oracle Call Interface プログラマーズ・ガイド』の第 16 章「その他の 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。

次のマニュアルの項を参照してください。

- SQL: 『Oracle9*i* SQL リファレンス』の第 6 章「ファンクション」の「BFILENAME」および第 18 章「SQL 文 : SAVEPOINT ～ UPDATE」の「UPDATE」

使用例

次の例では、BFILENAME ファンクションによって Print_media を更新します。

例

例を SQL で示します。この例は、すべてのプログラム環境に適用されます。

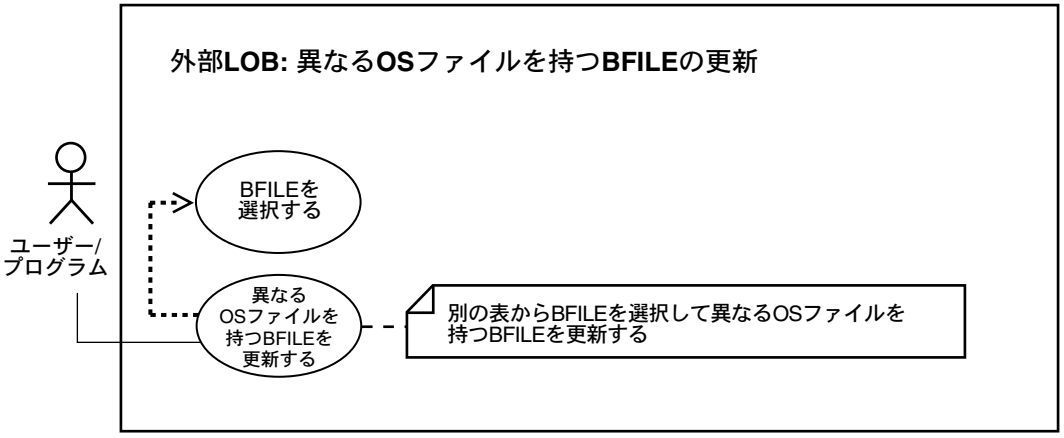
- SQL: BFILENAME() を使用した BFILE の更新 （12-193 ページ）

SQL: BFILENAME() を使用した BFILE の更新

```
/* Updating a BFILE using BFILENAME()  [Example script: 4059.sql] */

UPDATE Print_media
SET ad_graphic = BFILENAME('ADGRAPHIC_DIR', 'keyboard_graphic_3106_13001')
    WHERE product_id = 3106 AND ad_id = 13001;
```

別の表からの BFILE の選択による BFILE の更新

図 12-29 利用図 : 別の表からの BFILE の選択による BFILE の更新



参照 :

- 12-2 ページの表 12-1「利用モデル図 : 外部 LOB（BFILE）」を参照してください。
- 12-191 ページ の「BFILENAME() を使用した BFILE の更新」を参照してください。
- 12-196 ページの「初期化した BFILE ロケータを使用した BFILE の更新」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、OS のコマンドを発行して、その OS 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:『Oracle9*i* SQL リファレンス』の第 18 章「SQL 文 : SAVEPOINT ～ UPDATE」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

例を SQL で示します。この例は、すべてのプログラム環境に適用されます。

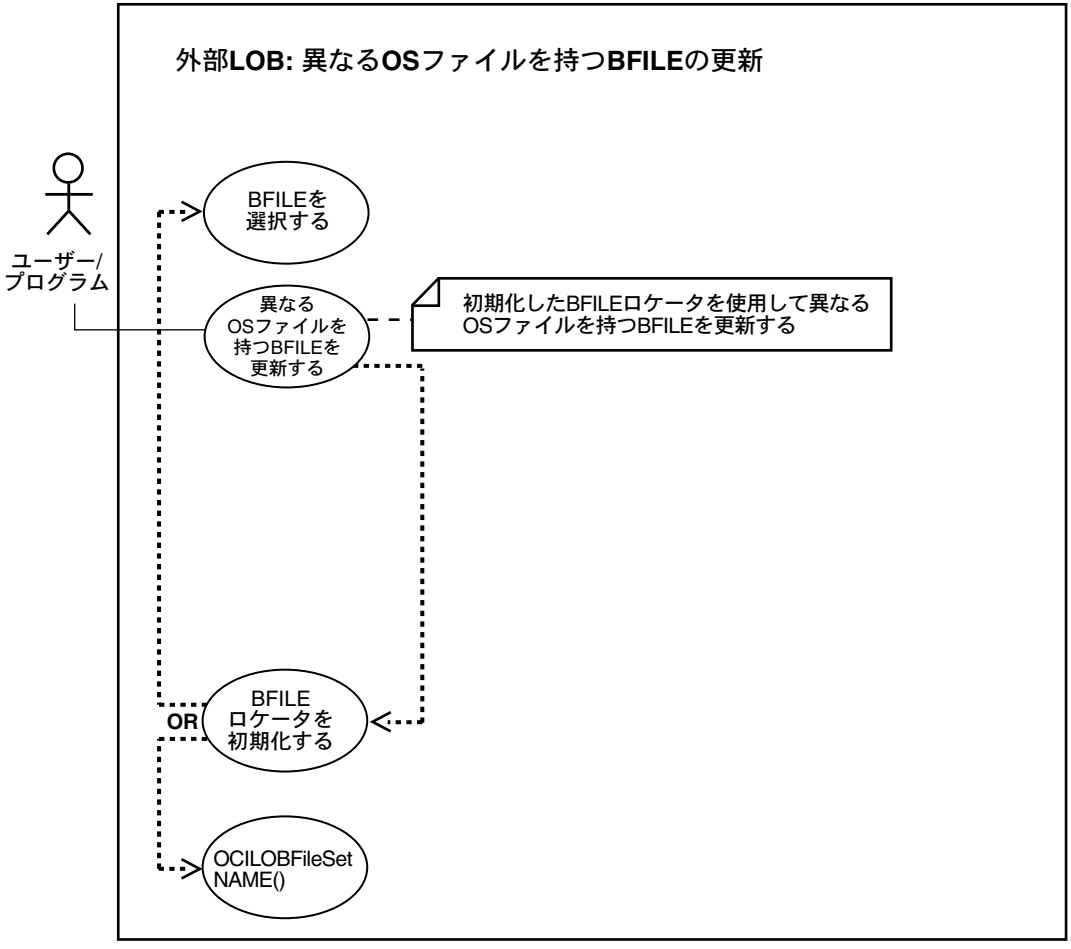
- SQL: 別の表からの BFILE の選択による BFILE の更新（12-195 ページ）

SQL: 別の表からの BFILE の選択による BFILE の更新

```
UPDATE Adheader_tab
    SET (header_name, creation_date, header_text, logo) =
       (SELECT * FROM AdheaderLib_tab AHtab
        WHERE AHtab.creation_date = '08/08/2001');
```

初期化した BFILE ロケータを使用した BFILE の更新

図 12-30 利用図 : 初期化した BFILE ロケータを使用した BFILE の更新



参照：

- 12-2 ページの表 12-1「利用モデル図 : 外部 LOB（BFILE）」を参照してください。
- 12-191 ページの「BFILENAME() を使用した BFILE の更新」を参照してください。
- 12-194 ページの「別の表からの BFILE の選択による BFILE の更新」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関数またはファンクションのリストは、第 3 章「様々なプログラム環境での LOB のサポート」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）：『Oracle9*i* SQL リファレンス』の第 18 章「SQL 文 : SAVEPOINT ～ UPDATE」の「UPDATE」
- C（OCI）：『Oracle Call Interface 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 16 章「その他の OCI リレーショナル関数」の「LOB 関数」の「OCILobFileSetName()」
- COBOL（Pro*COBOL）：『Pro*COBOL Precompiler プログラマーズ・ガイド』の LOB に関する詳細、LOB 文の使用上の注意、付録 F「埋込み SQL およびプリコンパイラ・ディレクティブ」の「ALLOCATE（実行可能埋込み SQL 拡張機能）」、および『Oracle9*i* SQL リファレンス』の第 18 章「SQL 文 : SAVEPOINT ～ UPDATE」の「UPDATE」
- C/C++（Pro*C/C++）：『Pro*C/C++ Precompiler 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」、付録 F「埋込み SQL 文およびディレクティブ」、および『Oracle9*i* SQL リファレンス』の第 18 章「SQL 文 : SAVEPOINT ～ UPDATE」の「UPDATE」

- Visual Basic（OO4O オンライン・ヘルプ）: ヘルプの「目次」タブから、「OO4O オートメーション・サーバー・リファレンス」>「OO4O サーバーのオブジェクト」>「OraBFILE」>「プロパティ」>「Open」、「DirectoryName」と「FileName」、「OO4O オートメーション・サーバー・リファレンス」>「OO4O サーバーのオブジェクト」>「OraDatabase」>「メソッド」>「ExecuteSQL」、および「OO4O オートメーション・サーバー・リファレンス」>「OO4O サーバーのオブジェクト」>「OraBfile」>「例」を選択
- Java（JDBC）: 『Oracle9*i* JDBC 開発者ガイドおよびリファレンス』の第 8 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『Oracle9*i* 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「JDBC 2.0 LOB 型と Oracle 拡張型のサポート」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での例を示します。

- PL/SQL（DBMS_LOB）: 初期化した BFILE ロケータを使用した BFILE の更新（12-198 ページ）
- C（OCI）: 初期化した BFILE ロケータを使用した BFILE の更新（12-199 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 （12-200 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（12-201 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（12-202 ページ）
- Java（JDBC）: 初期化した BFILE ロケータを使用した BFILE の更新（12-203 ページ）

PL/SQL（DBMS_LOB）: 初期化した BFILE ロケータを使用した BFILE の更新

```
/* Updating a BFILE by initializing a BFILE locator. [Example script:4061.sql]
   Procedure updateUseBindVariable_proc is not part of DBMS_LOB package: */

CREATE OR REPLACE PROCEDURE updateUseBindVariable_proc (File_loc BFILE) IS
BEGIN
   UPDATE Print_media SET ad_graphic = File_loc
         WHERE product_id = 3060 AND ad_id = 11001;
END;
```

```
DECLARE
   File_loc  BFILE;
BEGIN
   SELECT ad_graphic INTO File_loc
      FROM Print_media
         WHERE product_id = 3060 AND ad_id = 11001;
   updateUseBindVariable_proc (File_loc);
   COMMIT;
END;
```

C（OCI）：初期化した BFILE ロケータを使用した BFILE の更新

```
/* Updating a BFILE by initializing a BFILE locator. [Example script: 4062.c] */
void BfileUpdate(envhp, errhp, svchp, stmthp)
OCIEnv    *envhp;
OCIError  *errhp;
OCISvcCtx *svchp;
OCIStmt   *stmthp;
{
  OCILobLocator *Lob_loc;
  OCIBind *bndhp, *bndhp2;

  text  *updstmt =
    (text *) "UPDATE Print_media SET ad_graphic = :Lob_loc
              WHERE product_id = 3106 AND ad_id = 13001";

  OraText *Dir = (OraText *)"ADGRAPHIC_DIR",
          *Name = (OraText *)"keyboard_graphic_3106_13001";

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

  /* Allocate Locator resources: */
  (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &Lob_loc,
                            (ub4)OCI_DTYPE_FILE, (size_t) 0, (dvoid **) 0);

  checkerr (errhp, OCILobFileSetName(envhp, errhp, &Lob_loc,
                                     Dir, (ub2)strlen((char *)Dir),
                                     Name,(ub2)strlen((char *)Name)));

  checkerr (errhp, OCIBindByPos(stmthp, &bndhp, errhp, (ub4) 1,
                                (dvoid *) &Lob_loc, (sb4) 0,  SQLT_BFILE,
                                (dvoid *) 0, (ub2 *)0, (ub2 *)0,
                                (ub4) 0, (ub4 *) 0, (ub4) OCI_DEFAULT)
```

```
    || OCIBindByPos(stmthp, &bndhp2, errhp, (ub4) 2,
                            (dvoid *) &Lob_loc, (sb4) 0,  SQLT_BFILE,
                            (dvoid *) 0, (ub2 *)0, (ub2 *)0,
                           (ub4) 0, (ub4 *) 0, (ub4) OCI_DEFAULT)
                            );

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

 /* Free LOB resources: */
  OCIDescriptorFree((dvoid *) Lob_loc, (ub4) OCI_DTYPE_FILE);
}
```

COBOL（Pro*COBOL）：初期化した BFILE ロケータを使用した BFILE の更新

```
* Updating a BFILE by initializing a BFILE locator. [Example script: 4063.pco]
  IDENTIFICATION DIVISION.
  PROGRAM-ID. BFILE-UPDATE.
  ENVIRONMENT DIVISION.
  DATA DIVISION.
  WORKING-STORAGE SECTION.

  01  USERID          PIC X(11) VALUES "SAMP/SAMP".
  01  BFILE1          SQL-BFILE.
  01  BFILE-IND       PIC S9(4) COMP.
  01  DIR-ALIAS       PIC X(30) VARYING.
  01  FNAME           PIC X(30) VARYING.
  01  ORASLNRD        PIC 9(4).

      EXEC SQL INCLUDE SQLCA END-EXEC.
      EXEC ORACLE OPTION (ORACA=YES) END-EXEC.
      EXEC SQL INCLUDE ORACA END-EXEC.

  PROCEDURE DIVISION.
  BFILE-UPDATE.

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

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

   * Populate the BFILE:
      EXEC SQL WHENEVER NOT FOUND GOTO END-OF-BFILE END-EXEC.
```

```
        EXEC ORACLE OPTION (SELECT_ERROR=NO) END-EXEC.
        EXEC SQL
             SELECT AD_GRAPHIC INTO :BFILE1:BFILE-IND
             FROM PRINT_MEDIA WHERE PRODUCT_ID = 3060
             AND AD_ID = 13001 END-EXEC.

      * Make graphic associated with product_id=3106 same as product_id=3060
      * and ad_id = 13001:
        EXEC SQL
             UPDATE PRINT_MEDIA SET AD_GRAPHIC = :BFILE1:BFILE-IND
             WHERE PRODUCT_ID = 3106 AND AD_ID = 13001 END-EXEC.

      * Free the BFILE:
       END-OF-BFILE.
        EXEC SQL WHENEVER NOT FOUND CONTINUE END-EXEC.
        EXEC SQL FREE :BFILE1 END-EXEC.
        EXEC SQL ROLLBACK WORK RELEASE END-EXEC.
        STOP RUN.

       SQL-ERROR.
        EXEC SQL WHENEVER SQLERROR CONTINUE END-EXEC.
        MOVE ORASLNR TO ORASLNRD.
        DISPLAY " ".
        DISPLAY "ORACLE ERROR DETECTED ON LINE ", ORASLNRD, ":".
        DISPLAY " ".
        DISPLAY SQLERRMC.
        EXEC SQL ROLLBACK WORK RELEASE END-EXEC.
        STOP RUN.
```

C/C++（Pro*C/C++）：初期化した BFILE ロケータを使用した BFILE の更新

```
/* Updating a BFILE by initializing a BFILE locator. [Example script: 4064.pc] */

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

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

void updateUseBindVariable_proc(Lob_loc)
```

```
  OCIBFileLocator *Lob_loc;
{
  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  EXEC SQL UPDATE Print_media SET ad_graphic = :Lob_loc
       WHERE product_ID = 2056 AND ad_id = 12001;
}

void updateBFILE_proc()
{
  OCIBFileLocator *Lob_loc;

  EXEC SQL ALLOCATE :Lob_loc;
  EXEC SQL SELECT ad_graphic INTO :Lob_loc
           FROM Print_media WHERE product_id = 2056 AND ad_id 12001;
  updateUseBindVariable_proc(Lob_loc);
  EXEC SQL FREE :Lob_loc;
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  updateBFILE_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

Visual Basic（OO4O）: 初期化した BFILE ロケータを使用した BFILE の更新

```
'Updating a BFILE by initializing a BFILE locator. [Example script:4066.txt]

Dim MySession As OraSession
Dim OraDb As OraDatabase
Dim OraParameters As OraParameters, OraAdGraphic As OraBfile

Set MySession = CreateObject("OracleInProcServer.XOraSession")
Set OraDb = MySession.OpenDatabase("pmschema", "pm/pm", 0&)

OraDb.Connection.BeginTrans

Set OraParameters = OraDb.Parameters

'Define in out parameter of BFILE type:
OraParameters.Add "MyAdGraphic", Null, ORAPARM_BOTH, ORATYPE_BFILE

'Define out parameter of BFILE type:
OraDb.ExecuteSQL (
```

```
"BEGIN SELECT ad_graphic INTO :MyAdGraphic FROM Print_media
     WHERE product_id = 2056 AND ad_id = 12001;
      END;")

'Update the ad_graphic BFile for product_id=2056 AND ad_id = 12001
      to product_id=2268 AND ad_id = 21001:
OraDb.ExecuteSQL (
   "UPDATE Print_media SET ad_graphic = :MyAdGraphic
      WHERE product_id = 2268 AND ad_id = 21001")

'Get Directory alias and filename
'MsgBox " Directory alias is " & OraAdGraphic1.DirectoryName & " Filename is " &
OraAdGraphic1.filename

OraDb.Connection.CommitTrans
```

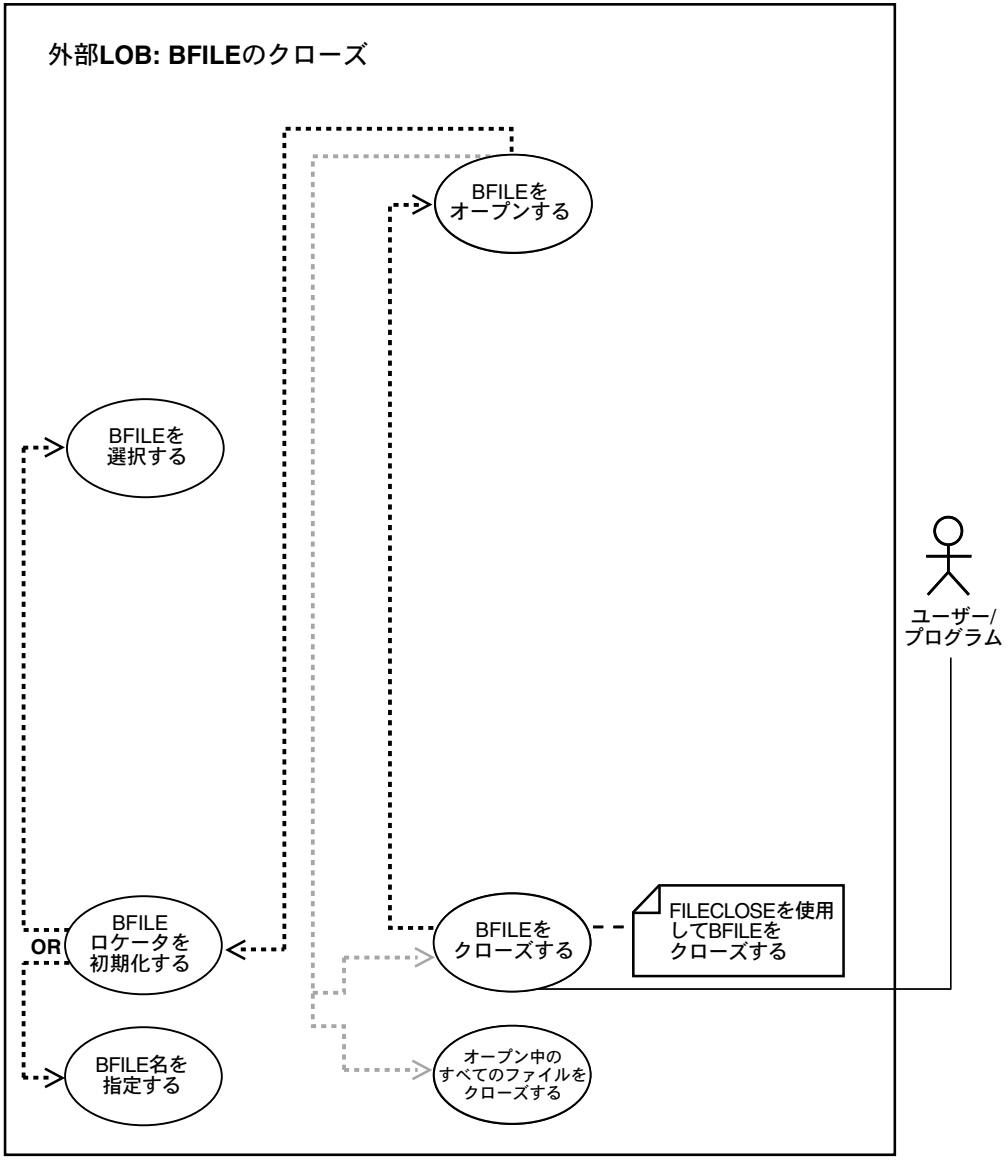
Java（JDBC）：初期化した BFILE ロケータを使用した BFILE の更新

```
// Updating a BFILE by initializing a BFILE locator. [Example script: 4067.java]

import java.io.InputStream;
import java.io.OutputStream;

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

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

public class Ex4_100
{

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

    // Connect to the database:
    Connection conn =
```

```
      DriverManager.getConnection ("jdbc:oracle:oci8:@", "samp", "samp");

    conn.setAutoCommit (false);

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

    try
    {
        BFILE src_lob = null;
        ResultSet rset = null;
        OraclePreparedStatement pstmt = null;

        rset = stmt.executeQuery (
           "SELECT ad_graphic FROM Print_media
               WHERE product_id = 3106 AND ad_id = 13001");
        if (rset.next())
        {
           src_lob = ((OracleResultSet)rset).getBFILE (1);
        }

        // Prepare a CallableStatement to OPEN the LOB for READWRITE:
        pstmt = (OraclePreparedStatement) conn.prepareStatement (
            "UPDATE Print_media SET ad_graphic = ?
                WHERE product_id = 3060 AND ad_id = 11001");
        pstmt.setBFILE(1, src_lob);
        pstmt.execute();

       //Close the statements and commit the transaction:
       stmt.close();
       pstmt.close();
       conn.commit();
       conn.close();
    }
    catch (SQLException e)
    {
       e.printStackTrace();
    }
  }
}
```

FILECLOSE を使用した BFILE のクローズ

図 12-31 利用図 : FILECLOSE を使用した BFILE のクローズ



参照 :

- 12-2 ページの表 12-1「利用モデル図 : 外部 LOB（BFILE）」を参照してください。
- 12-210 ページの「CLOSE を使用した BFILE のクローズ」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

FILECLOSE を使用した BFILE のクローズはサポートされていますが、かわりに CLOSE を使用することをお薦めします。これにより将来の拡張性が向上します。

構文

各プログラム環境で使用可能な関数またはファンクションのリストは、第 3 章「様々なプログラム環境での LOB のサポート」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）：『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』の第 23 章「DBMS_LOB」の「DBMS_LOB サブプログラムの要約」の「FILEOPEN プロシージャ」および「FILECLOSE プロシージャ」
- C（OCI）：『Oracle Call Interface 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 16 章「その他の 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）：『Oracle9*i* JDBC 開発者ガイドおよびリファレンス』の第 8 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『Oracle9*i* 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「JDBC 2.0 LOB 型と Oracle 拡張型のサポート」の「BLOB、CLOB および BFILE のサポート」

使用例

次の例では、ADPHOTO_DIR の BFILE をクローズします。

例

PL/SQL（DBMS_LOB）: FILECLOSE を使用した BFILE のクローズ

```
/* Closing a BFILE with FILECLOSE.  [Example script: 4068.sql]
   Procedure closeBFILE_procOne is not part of DBMS_LOB package: */

CREATE OR REPLACE PROCEDURE closeBFILE_procOne IS
   File_loc  BFILE := BFILENAME('ADPHOTO_DIR', 'keyboard_photo_3060_11001');
BEGIN
   DBMS_LOB.FILEOPEN(File_loc, DBMS_LOB.FILE_READONLY);
   /* ...Do some processing. */
   DBMS_LOB.FILECLOSE(File_loc);
END;
```

C（OCI）: FILECLOSE を使用した BFILE のクローズ

```
/* Closing a BFILE with FILECLOSE.  [Example script: 4069.c] */

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

   OCILobLocator *bfile_loc;

   /* Allocate the locator descriptors */
   (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &bfile_loc,
                             (ub4) OCI_DTYPE_FILE,
                             (size_t) 0, (dvoid **) 0);

   checkerr(errhp, OCILobFileSetName(envhp, errhp, &bfile_loc,
                        (OraText *) "ADGRAPHIC_DIR", (ub2) strlen("ADGRAPHIC_DIR"),
                        (OraText *) "keyboard_graphic_3106_13001",
```

```
                          (ub2) strlen("keyboard_graphic_3106_13001")));

   checkerr(errhp, OCILobFileOpen(svchp, errhp, bfile_loc,
                                        (ub1) OCI_FILE_READONLY));

   checkerr(errhp, OCILobFileClose(svchp, errhp, bfile_loc));

   /* Free the locator descriptor */
   OCIDescriptorFree((dvoid *)bfile_loc, (ub4)OCI_DTYPE_FILE);
}
```

Visual Basic（OO4O）: 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。

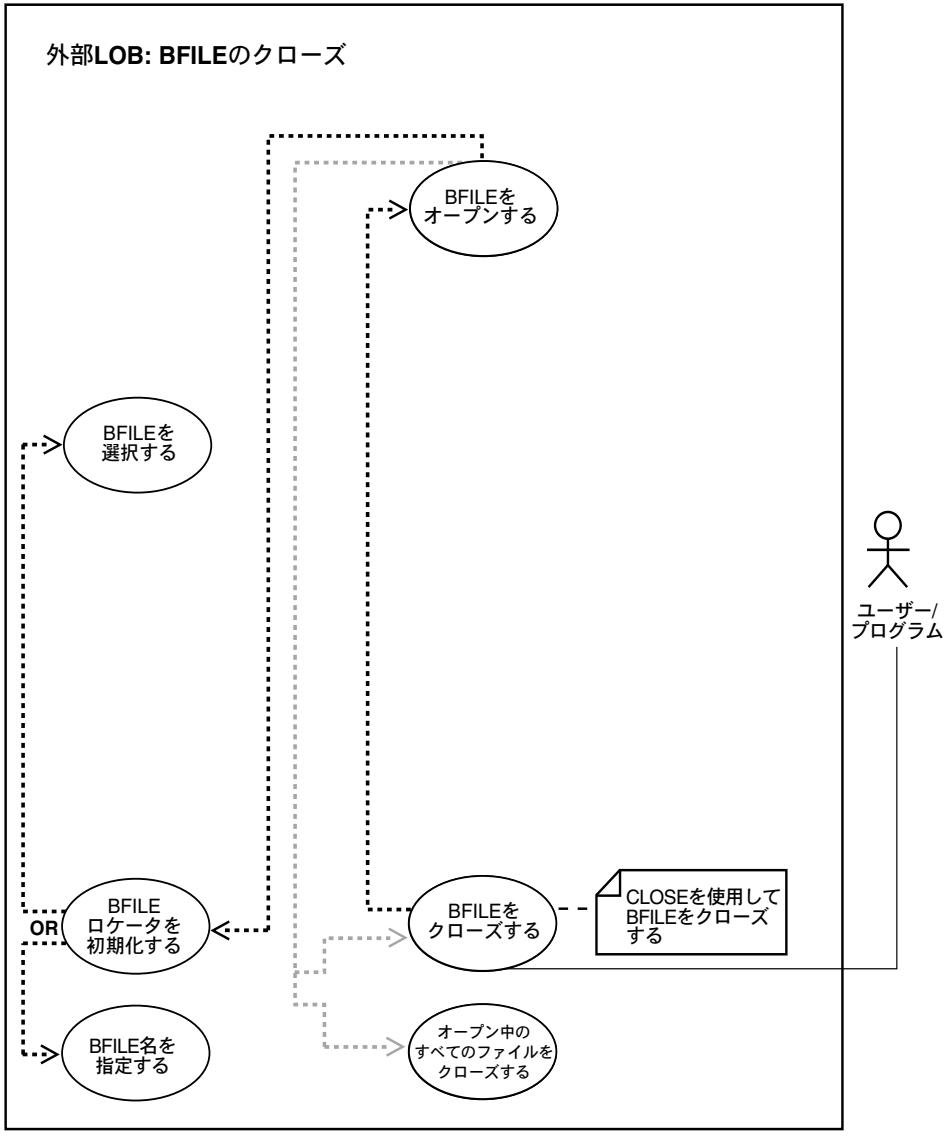
Java（JDBC）: FILECLOSE を使用した BFILE のクローズ

```
// Closing a BFILE with FILECLOSE.  [Example script: 4071.java]

import java.io.InputStream;
import java.io.OutputStream;

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

// Oracle Specific JDBC classes:
import oracle.sql.*;
import oracle.jdbc.driver.*;
public class Ex4_45
{
  public static void main (String args [])
       throws Exception
  {
    // Load the Oracle JDBC driver:
    DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

    // Connect to the database:
```

```
    Connection conn =
      DriverManager.getConnection ("jdbc:oracle:oci8:@", "samp", "samp");

    conn.setAutoCommit (false);

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

    try
    {
       BFILE src_lob = null;
       ResultSet rset = null;
       boolean result = false;

       rset = stmt.executeQuery (
          "SELECT BFILENAME('ADGRAPHIC_DIR','keyboard_graphic_3106_11001')
              FROM DUAL");
       if (rset.next())
       {
          src_lob = ((OracleResultSet)rset).getBFILE (1);
       }

       result = src_lob.isFileOpen();
       System.out.println(
          "result of fileIsOpen() before opening file : " + result);

       src_lob.openFile();

       result = src_lob.isFileOpen();
       System.out.println(
          "result of fileIsOpen() after opening file : " + result);

       // Close the BFILE,  statement and connection:
       src_lob.closeFile();
       stmt.close();
       conn.commit();
       conn.close();
    }
    catch (SQLException e)
    {
       e.printStackTrace();
    }
  }
}
```

CLOSE を使用した BFILE のクローズ

図 12-32 利用図 : CLOSE を使用したオープン中の BFILE のクローズ



参照：

- 12-2 ページの表 12-1「利用モデル図 : 外部 LOB（BFILE）」を参照してください。
- 12-205 ページの「FILECLOSE を使用した BFILE のクローズ」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

OPEN と組み合せて CLOSE を使用します。

参照： 12-70 ページの「OPEN を使用した BFILE のオープン」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関数またはファンクションのリストは、第 3 章「様々なプログラム環境での LOB のサポート」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）：『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』の第 23 章「DBMS_LOB」の「DBMS_LOB サブプログラムの要約」の「CLOSE プロシージャ」
- C（OCI）：『Oracle Call Interface 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 16 章「その他の OCI リレーショナル関数」の「LOB 関数」の「OCILobClose()」
- COBOL（Pro*COBOL）：『Pro*COBOL Precompiler プログラマーズ・ガイド』の LOB に関する詳細、LOB 文の使用上の注意および付録 F「埋込み SQL およびプリコンパイラ・ディレクティブ」の「LOB CLOSE（実行可能埋込み SQL 拡張機能）」
- C/C++（Pro*C/C++）：『Pro*C/C++ Precompiler 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」および付録 F「埋込み SQL 文およびディレクティブ」の「LOB CLOSE（実行可能埋込み SQL 拡張機能）」
- Visual Basic（OO4O オンライン・ヘルプ）：ヘルプの「目次」タブから、「OO4O オートメーション・サーバー・リファレンス」>「OO4O サーバーのオブジェクト」>「OraBFILE」>「プロパティ」>「IsOpen」、および「OO4O オートメーション・サーバー・リファレンス」>「OO4O サーバーのオブジェクト」>「OraBFILE」>「例」を選択

- Java（JDBC）：『Oracle9*i* JDBC 開発者ガイドおよびリファレンス』の第 8 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『Oracle9*i* 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「JDBC 2.0 LOB 型と Oracle 拡張型のサポート」の「BLOB、CLOB および BFILE のサポート」

使用例

次の例では、ADGRAPHIC_DIR の BFILE をクローズします。

例

- PL/SQL（DBMS_LOB）: CLOSE を使用した BFILE のクローズ（12-212 ページ）
- C（OCI）: CLOSE を使用した BFILE のクローズ（12-212 ページ）
- COBOL（Pro*COBOL）: CLOSE を使用した BFILE のクローズ（12-213 ページ）
- C/C++（Pro*C/C++）: CLOSE を使用した BFILE のクローズ（12-214 ページ）
- Visual Basic（OO4O）: CLOSE を使用した BFILE のクローズ（12-215 ページ）
- Java（JDBC）: CLOSE を使用した BFILE のクローズ（12-216 ページ）

PL/SQL（DBMS_LOB）: CLOSE を使用した BFILE のクローズ

```
/* Closing a BFILE with CLOSE.  [Example script: 4072.sql]
   Procedure closeBFILE_procTwo is not part of DBMS_LOB package: */

CREATE OR REPLACE PROCEDURE closeBFILE_procTwo IS
   File_loc  BFILE := BFILENAME('ADGRAPHIC_DIR','keyboard_graphic_3060_11001');
BEGIN
   DBMS_LOB.OPEN(File_loc, DBMS_LOB.LOB_READONLY);
   /* ...Do some processing. */
   DBMS_LOB.CLOSE(File_loc);
END;
```

C（OCI）: CLOSE を使用した BFILE のクローズ

```
/* Closing a BFILE with CLOSE.  [Example script: 4073.c] */

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

```
  OCILobLocator *bfile_loc;

  /* Allocate the locator descriptors */
  (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &bfile_loc,
                  (ub4) OCI_DTYPE_FILE,
                  (size_t) 0, (dvoid **) 0);

  checkerr(errhp, OCILobFileSetName(envhp, errhp, &bfile_loc,
                  (OraText *) "ADGRAPHIC_DIR", (ub2) strlen("ADGRAPHIC_DIR"),
                  (OraText *) "keyboard_3106",
                  (ub2) strlen("keyboard_3106")));

  checkerr(errhp, OCILobOpen(svchp, errhp, bfile_loc,
                  (ub1) OCI_LOB_READONLY));

  checkerr(errhp, OCILobClose(svchp, errhp, bfile_loc));

  /* Free the locator descriptor */
  OCIDescriptorFree((dvoid *)bfile_loc, (ub4)OCI_DTYPE_FILE);
}
```

COBOL（Pro*COBOL）: CLOSE を使用した BFILE のクローズ

```
      * Closing a BFILE with CLOSE.  [Example script: 4074.pco]
        IDENTIFICATION DIVISION.
        PROGRAM-ID. BFILE-CLOSE.
        ENVIRONMENT DIVISION.
        DATA DIVISION.
        WORKING-STORAGE SECTION.

        01  USERID   PIC X(11) VALUES "SAMP/SAMP".
        01  BFILE1         SQL-BFILE.
        01  DIR-ALIAS      PIC X(30) VARYING.
        01  FNAME          PIC X(20) VARYING.
        01  ORASLNRD        PIC 9(4).

            EXEC SQL INCLUDE SQLCA END-EXEC.
            EXEC ORACLE OPTION (ORACA=YES) END-EXEC.
            EXEC SQL INCLUDE ORACA END-EXEC.

        PROCEDURE DIVISION.
        BFILE-CLOSE.
```

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

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

       * Set up the directory and file information:
           MOVE "ADGRAPHIC_DIR" TO DIR-ALIAS-ARR.
           MOVE 9 TO DIR-ALIAS-LEN.
           MOVE "keyboard_graphic_3106_13001" TO FNAME-ARR.
           MOVE 13 TO FNAME-LEN.

           EXEC SQL
                LOB FILE SET :BFILE1
                DIRECTORY = :DIR-ALIAS, FILENAME = :FNAME END-EXEC.

           EXEC SQL
                LOB OPEN :BFILE1 READ ONLY END-EXEC.

       * Close the LOB:
           EXEC SQL LOB CLOSE :BFILE1 END-EXEC.

       * And free the LOB locator:
           EXEC SQL FREE :BFILE1 END-EXEC.
           STOP RUN.

        SQL-ERROR.
           EXEC SQL WHENEVER SQLERROR CONTINUE END-EXEC.
           MOVE ORASLNR TO ORASLNRD.
           DISPLAY " ".
           DISPLAY "ORACLE ERROR DETECTED ON LINE ", ORASLNRD, ":".
           DISPLAY " ".
           DISPLAY SQLERRMC.
           EXEC SQL ROLLBACK WORK RELEASE END-EXEC.
           STOP RUN.
```

C/C++（Pro*C/C++）: CLOSE を使用した BFILE のクローズ

```
/* Closing a BFILE with CLOSE.  [Example script: 4075.pc]
   Pro*C/C++ has only one form of CLOSE for BFILEs.  Pro*C/C++ has no
   FILECLOSE statement. A simple CLOSE statement is used instead: */

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

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

void closeBFILE_proc()
{
  OCIBFileLocator *Lob_loc;
  char *Dir = "ADGRAPHIC_DIR", *Name = "mousepad_graphic_2056_12001";

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  EXEC SQL ALLOCATE :Lob_loc;
  EXEC SQL LOB FILE SET :Lob_loc DIRECTORY = :Dir, FILENAME = :Name;
  EXEC SQL LOB OPEN :Lob_loc READ ONLY;
  /* ... Do some processing */
  EXEC SQL LOB CLOSE :Lob_loc;
  EXEC SQL FREE :Lob_loc;
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  closeBFILE_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

Visual Basic（OO4O）: CLOSE を使用した BFILE のクローズ

```
'Closing a BFILE with CLOSE.  [Example script: 4076.txt

Dim MySession As OraSession
Dim OraDb As OraDatabase

Dim OraDyn As OraDynaset, OraAdGraphic As OraBfile, amount_read%, chunksize%, chunk

Set MySession = CreateObject("OracleInProcServer.XOraSession")
Set OraDb = MySession.OpenDatabase("pmschema", "pm/pm", 0&)

chunksize = 32767
Set OraDyn = OraDb.CreateDynaset("select * from Print_media", ORADYN_DEFAULT)
Set OraAdGraphic = OraDyn.Fields("ad_graphic").Value
```

```
If OraAdGraphic.IsOpen Then
   'Process because the file is already open
   OraAdGraphic.Close
End If
```

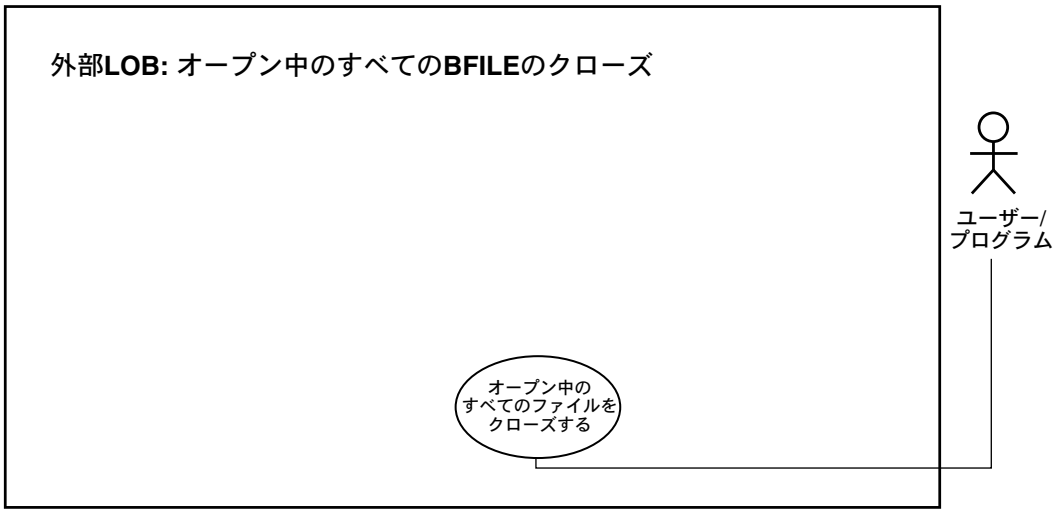
Java（JDBC）: CLOSE を使用した BFILE のクローズ

```
// Closing a BFILE with CLOSE.  [Example script: 4077.java]

import java.io.InputStream;
import java.io.OutputStream;

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

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

public class Ex4_48
{
  public static void main (String args [])
       throws Exception
  {
    // Load the Oracle JDBC driver:
    DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

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

    conn.setAutoCommit (false);

    // Create a Statement:
    Statement stmt = conn.createStatement ();
    try
    {
     BFILE src_lob = null;
       ResultSet rset = null;

       rset = stmt.executeQuery (
```

```
      "SELECT BFILENAME('ADGRAPHIC_DIR', 'keyboard_graphic_3106_13001') FROM
DUAL");
      OracleCallableStatement cstmt = null;
      if (rset.next())
      {
         src_lob = ((OracleResultSet)rset).getBFILE (1);
  cstmt = (OracleCallableStatement)conn.prepareCall
      ("begin dbms_lob.open (?,dbms_lob.lob_readonly); end;");
         cstmt.registerOutParameter(1,OracleTypes.BFILE);
         cstmt.setBFILE (1, src_lob);
         cstmt.execute();
         src_lob = cstmt.getBFILE(1);
         System.out.println ("the file is now open");
      }

      // Close the BFILE, statement and connection:
      cstmt = (OracleCallableStatement)
      conn.prepareCall ("begin dbms_lob.close(?); end;");
      cstmt.setBFILE(1,src_lob);
      cstmt.execute();
      stmt.close();
      conn.commit();
      conn.close();
    }
    catch (SQLException e)
    {
      e.printStackTrace();
    }
  }
}
```

FILECLOSEALL を使用したオープン中のすべての BFILE のクローズ

図 12-33 利用図 : オープン中のすべての BFILE のクローズ



参照 : 内部一時 LOB に関するすべての基本操作については、12-2 ページの表 12-1「利用モデル図 : 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す。

参照 : 12-64 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関数またはファンクションのリストは、第 3 章「様々なプログラム環境での LOB のサポート」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）：『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』の第 23 章「DBMS_LOB」の「DBMS_LOB サブプログラムの要約」の「FILECLOSEALL プロシージャ」
- C（OCI）：『Oracle Call Interface 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 16 章「その他の OCI リレーショナル関数」の「LOB 関数」の「OCILobFileCloseAll()」
- COBOL（Pro*COBOL）：『Pro*COBOL Precompiler プログラマーズ・ガイド』の LOB に関する詳細、LOB 文の使用上の注意および付録 F「埋込み SQL およびプリコンパイラ・ディレクティブ」の「LOB FILE CLOSE ALL（実行可能埋込み SQL 拡張機能）」
- C/C++（Pro*C/C++）：『Pro*C/C++ Precompiler 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」および付録 F「埋込み SQL 文およびディレクティブ」の「LOB FILE CLOSE ALL（実行可能埋込み SQL 拡張機能）」
- Visual Basic（OO4O オンライン・ヘルプ）：ヘルプの 「目次」タブから、「OO4O オートメーション・サーバー・リファレンス」>「OO4O サーバーのオブジェクト」>「OraBFILE」>「メソッド」>「CloseAll」、および「OO4O オートメーション・サーバー・リファレンス」>「OO4O サーバーのオブジェクト」>「OraBFILE」>「例」を選択
- Java （JDBC）：『Oracle9*i* JDBC 開発者ガイドおよびリファレンス』の第 8 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『Oracle9*i* 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「JDBC 2.0 LOB 型と Oracle 拡張型のサポート」の「BLOB、CLOB および BFILE のサポート」

使用例

ありません。

例

PL/SQL（DBMS_LOB）: FILECLOSEALL を使用したオープン中のすべての BFILE のクローズ

```
/* Closing all open BFILEs.  [Example script: 4078.sql]
   Procedure closeAllOpenFilesBFILE_proc is not part of DBMS_LOB package: */

CREATE OR REPLACE PROCEDURE closeAllOpenFilesBFILE_proc IS
BEGIN
   /* Close all open BFILEs: */
   DBMS_LOB.FILECLOSEALL;
END;
```

C（OCI）: FILECLOSEALL を使用したオープン中のすべての BFILE のクローズ

```
/* Closing all open BFILEs.  [Example script: 4079.c] */

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

   OCILobLocator *bfile_loc1;
   OCILobLocator *bfile_loc2;

   /* Allocate the locator descriptors */
```

```
  (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &bfile_loc1,
                 (ub4) OCI_DTYPE_FILE,
                 (size_t) 0, (dvoid **) 0);

  (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &bfile_loc2,
                 (ub4) OCI_DTYPE_FILE,
                 (size_t) 0, (dvoid **) 0);

  checkerr(errhp, OCILobFileSetName(envhp, errhp, &bfile_loc1,
                 (OraText *) "ADGRAPHIC_DIR", (ub2) strlen("ADGRAPHIC_DIR"),
                 (OraText *) "keyboard_graphic_3106_13001",
                 (ub2) strlen("keyboard_graphic_3106_13001")));

  checkerr(errhp, OCILobFileSetName(envhp, errhp, &bfile_loc2,
                 (OraText *) "ADGRAPHIC_DIR", (ub2) strlen("ADGRAPHIC_DIR"),
                 (OraText *) "monitor_graphic_3060_11001",
                 (ub2) strlen("monitor_graphic_306_11001")));

  checkerr(errhp, OCILobFileOpen(svchp, errhp, bfile_loc1,
                 (ub1) OCI_LOB_READONLY));

  checkerr(errhp, OCILobFileOpen(svchp, errhp, bfile_loc2,
                 (ub1) OCI_LOB_READONLY));

  checkerr(errhp, OCILobFileCloseAll(svchp, errhp));

  /* Free the locator descriptor */
  OCIDescriptorFree((dvoid *)bfile_loc1, (ub4)OCI_DTYPE_FILE);
  OCIDescriptorFree((dvoid *)bfile_loc2, (ub4)OCI_DTYPE_FILE);
}
```

COBOL（Pro*COBOL）: FILECLOSEALL を使用したオープン中のすべての BFILE のクローズ

```
      * Closing all open BFILEs.  [Example script: 4080.pco]
        IDENTIFICATION DIVISION.
        PROGRAM-ID. BFILE-CLOSE-ALL.
        ENVIRONMENT DIVISION.
        DATA DIVISION.
        WORKING-STORAGE SECTION.

        01  USERID   PIC X(11) VALUES "SAMP/SAMP".
        01  BFILE1         SQL-BFILE.
        01  BFILE2         SQL-BFILE.
        01  DIR-ALIAS1     PIC X(30) VARYING.
```

```
 01  FNAME1          PIC X(20) VARYING.
 01  DIR-ALIAS2      PIC X(30) VARYING.
 01  FNAME2          PIC X(20) VARYING.
 01  ORASLNRD        PIC 9(4).

     EXEC SQL INCLUDE SQLCA END-EXEC.
     EXEC ORACLE OPTION (ORACA=YES) END-EXEC.
     EXEC SQL INCLUDE ORACA END-EXEC.

 PROCEDURE DIVISION.
 BFILE-CLOSE-ALL.

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

* Allocate the BFILEs:
     EXEC SQL ALLOCATE :BFILE1 END-EXEC.
     EXEC SQL ALLOCATE :BFILE2 END-EXEC.

* Set up the directory and file information:
     MOVE "ADGRAPHIC_DIR" TO DIR-ALIAS1-ARR.
     MOVE 9 TO DIR-ALIAS1-LEN.
     MOVE "keyboard_graphic_3106_13001" TO FNAME1-ARR.
     MOVE 16 TO FNAME1-LEN.

     EXEC SQL
          LOB FILE SET :BFILE1
          DIRECTORY = :DIR-ALIAS1, FILENAME = :FNAME1 END-EXEC.
     EXEC SQL LOB OPEN :BFILE1 READ ONLY END-EXEC.

* Set up the directory and file information:
     MOVE "ADGRAPHIC_DIR" TO DIR-ALIAS2-ARR.
     MOVE 9 TO DIR-ALIAS2-LEN.
     MOVE "mousepad_graphic_2056_12001" TO FNAME2-ARR.
     MOVE 13 TO FNAME2-LEN.
     EXEC SQL LOB FILE SET :BFILE2
          DIRECTORY = :DIR-ALIAS2, FILENAME = :FNAME2 END-EXEC.
     EXEC SQL LOB OPEN :BFILE2 READ ONLY END-EXEC.

* Close both BFILE1 and BFILE2:
     EXEC SQL LOB FILE CLOSE ALL END-EXEC.
     STOP RUN.

 SQL-ERROR.
     EXEC SQL WHENEVER SQLERROR CONTINUE END-EXEC.
```

```
MOVE ORASLNR TO ORASLNRD.
DISPLAY " ".
DISPLAY "ORACLE ERROR DETECTED ON LINE ", ORASLNRD, ":".
DISPLAY " ".
DISPLAY SQLERRMC.
EXEC SQL ROLLBACK WORK RELEASE END-EXEC.
STOP RUN.
```

C/C++（Pro*C/C++）: FILECLOSEALL を使用したオープン中のすべての BFILE のクローズ

```
/* Closing all open BFILEs.  [Example script: 4081.pc] */

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

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

void closeAllOpenBFILEs_proc()
{
  OCIBFileLocator *Lob_loc1, *Lob_loc2;

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  EXEC SQL ALLOCATE :Lob_loc1;
  EXEC SQL ALLOCATE :Lob_loc2;
  /* Populate the Locators: */
  EXEC SQL SELECT ad_graphic INTO :Lob_loc1
              FROM Print_media
              WHERE product_id = 2056 AND ad_id = 12001;
  EXEC SQL SELECT Mtab.ad_graphic INTO Lob_loc2
              FROM Print_media PMtab
              WHERE PMtab.product_id = 3060 AND ad_id = 11001;
  /* Open both BFILEs: */
  EXEC SQL LOB OPEN :Lob_loc1 READ ONLY;
  EXEC SQL LOB OPEN :Lob_loc2 READ ONLY;
  /* Close all open BFILEs: */
  EXEC SQL LOB FILE CLOSE ALL;
  /* Free resources held by the Locators: */
```

```
  EXEC SQL FREE :Lob_loc1;
  EXEC SQL FREE :Lob_loc2;
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  closeAllOpenBFILEs_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

Visual Basic（OO4O）: FILECLOSEALL を使用したオープン中のすべての BFILE のクローズ

```
'Closing all open BFILEs.  [Example script: 4083.txt]

Dim OraParameters as OraParameters, OraAdGraphic as OraBFile
OraConnection.BeginTrans

Set OraParameters = OraDatabase.Parameters

'Define in out parameter of BFILE type:
OraParameters.Add "MyAdGraphic", Null,ORAPARAM_BOTH,ORATYPE_BFILE

'Select the ad graphic BFile for product_id 2268:
OraDatabase.ExecuteSQL("Begin SELECT ad_graphic INTO :MyAdGraphic FROM
Print_media WHERE product_id = 2268 AND ad_id = 21001; END; " )

'Get the BFile ad_graphic column:
set OraAdGraphic = OraParameters("MyAdGraphic").Value

'Open the OraAdGraphic:
OraAdGraphic.Open

'Do some processing on OraAdGraphic

'Close all the BFILEs associated with OraAdGraphic:
OraAdGraphic.CloseAll
```

Java（JDBC）: FILECLOSEALL を使用したオープン中のすべての BFILE のクローズ

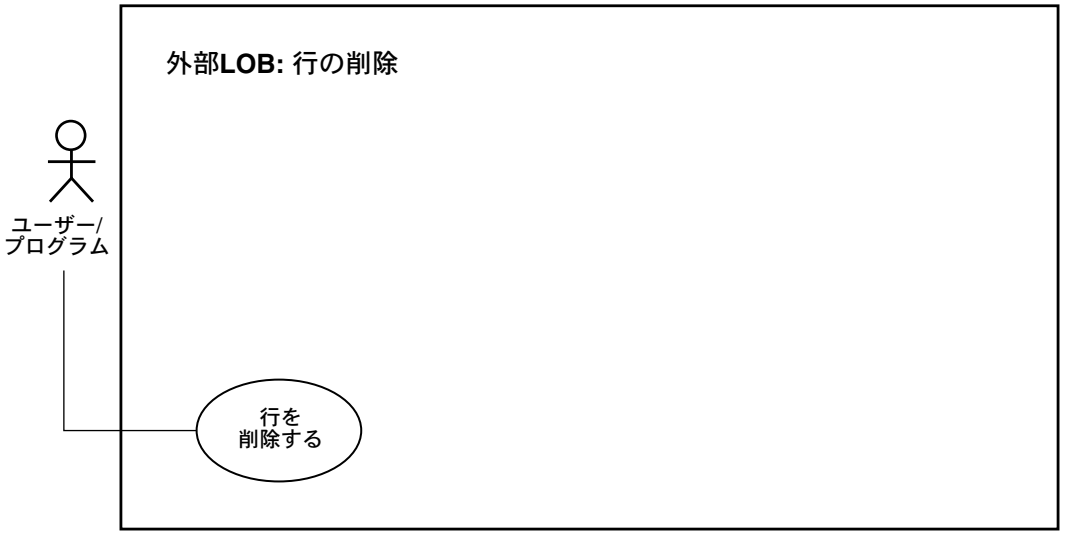
```
// Closing all open BFILEs.  [Example script: 4084.java]

import java.io.InputStream;
import java.io.OutputStream;

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

// Oracle Specific JDBC classes:
import oracle.sql.*;
import oracle.jdbc.driver.*;
public class Ex4_66
{
  static final int MAXBUFSIZE = 32767;
  public static void main (String args [])
       throws Exception
  {
    // Load the Oracle JDBC driver:
    DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

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

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

    // Create a Statement:
    Statement stmt = conn.createStatement ();
    try
    {
       BFILE lob_loc1 = null;
       BFILE lob_loc2 = null;
       ResultSet rset = null;
       OracleCallableStatement cstmt = null;
       rset = stmt.executeQuery (
          "SELECT ad_graphic FROM Print_media
              WHERE product_id = 3106 AND ad_id = 13001");
```

```
if (rset.next())
{
   lob_loc1 = ((OracleResultSet)rset).getBFILE (1);
}

rset = stmt.executeQuery (
   "SELECT BFILENAME('ADGRAPHIC_DIR', 'keyboard_graphic_3106_13001')
      FROM DUAL");
if (rset.next())
{
   lob_loc2 = ((OracleResultSet)rset).getBFILE (1);
}

cstmt = (OracleCallableStatement) conn.prepareCall (
   "BEGIN DBMS_LOB.FILEOPEN(?,DBMS_LOB.LOB_READONLY); END;");
// Open the first LOB:
cstmt.setBFILE(1, lob_loc1);
cstmt.execute();

cstmt = (OracleCallableStatement) conn.prepareCall (
   "BEGIN DBMS_LOB.FILEOPEN(?,DBMS_LOB.LOB_READONLY); END;");
// Use the same CallableStatement to open the second LOB:
cstmt.setBFILE(1, lob_loc2);
cstmt.execute();

lob_loc1.openFile ();
lob_loc2.openFile ();

// Compare MAXBUFSIZE bytes starting at the first byte of
// both lob_loc1 and lob_loc2:
cstmt = (OracleCallableStatement) conn.prepareCall (
   "BEGIN ? := DBMS_LOB.COMPARE(?, ?, ?, 1, 1); END;");
cstmt.registerOutParameter (1, Types.NUMERIC);
cstmt.setBFILE(2, lob_loc1);
cstmt.setBFILE(3, lob_loc2);
cstmt.setInt(4, MAXBUFSIZE);
cstmt.execute();
int result = cstmt.getInt(1);
System.out.println("Comparison result: " + Integer.toString(result));

// Close all BFILEs:
stmt.execute("BEGIN DBMS_LOB.FILECLOSEALL; END;");

stmt.close();
cstmt.close();
conn.commit();
conn.close();
```

```
    }
    catch (SQLException e)
    {
       e.printStackTrace();
    }
  }
}
```

BFILE を含む表の行の削除

図 12-34 利用図 : BFILE を含む表の行の削除



参照 : 内部一時 LOB に関するすべての基本操作については、12-2 ページの表 12-1「利用モデル図 : 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 OS 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、OS 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:『Oracle9*i* SQL リファレンス』の第 16 章「SQL 文 : CREATE TYPE ～ DROP ROLLBACK SEGMENT」の「DELETE」と「DROP」および第 18 章「SQL 文 : SAVEPOINT ～ UPDATE」の「TRUNCATE」

使用例

次に示す DELETE、DROP TABLE または TRUNCATE TABLE 文は、行（product_id 3106 および ad_id 13001 の製品の広告の図形イメージを参照する BFILE ロケータ）は削除しますが、図形イメージの OS ファイルは削除しません。

例

例を SQL で示します。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」（12-229 ページ）

SQL: 表からの行の削除

```
/* Deleting the row of a table containing a BFILE  [Example script: 4085.sql] */

DELETE FROM Print_media
   WHERE product_id = 3106 AND ad_id = 13001;


DROP TABLE Multimedia_tab;


TRUNCATE TABLE Multimedia_tab;
```

13

OraOLEDB を使用した LOB の操作

この章の内容は次のとおりです。

OLE DB の概要

OLE DB は、異なるストアから様々な型のデータに同じ方法でアクセスするためのオープン仕様です。OLE DB では、一連の COM インタフェースを使用して、異なる型のデータに対してアクセスおよび操作を行います。このインタフェースは、様々なデータベース・プロバイダによって提供されています。

OLE DB には、コンシューマおよびプロバイダの概念が導入されています。コンシューマとは、OLE DB インタフェースを使用またはコンシュームするクライアント・アプリケーションです。プロバイダは、OLE DB インタフェースを公開するコンポーネントです。

一般的なプロバイダは、特定のデータ・ストアからデータを取り出し、そのデータを表形式でコンシューマに公開します。

OraOLEDB: OLE DB および LOB のサポート

OraOLEDB は、Oracle の OLE DB プロバイダです。OraOLEDB によって、LOB を含む Oracle データに、高いパフォーマンスで効率的にアクセスできます。また、特定の LOB 型を更新することもできます。

OraOLEDB は、次の LOB 型をサポートします。

- **永続 LOB**: 行セット全体で読込み / 書込み
- **BFILE**: 行セット全体で読込み専用
- **一時 LOB**: 行セット全体で未サポート

行セット・オブジェクト

行セットは、OLE DB オブジェクトで、SQL の SELECT 文または REF カーソルを戻すストアド・プロシージャを実行して取得したデータに対して読込み / 書込み機能を提供します。

BFILE は、読込み専用として行セットに含むことができます。

ADO レコードセットおよび OLE DB 行セットを使用した LOB の操作

LOB データの取出しおよび格納は、プロバイダのキャッシュでは行われません。サーバー・カーソルが使用されると、OraOLEDB は、要求された LOB データのみをコンシューマに提供します。

> **注意：** Oracle データベースのほとんどの LOB 列は、最大 4GB のデータ記憶域をサポートしていますが、ADO によって列の最大サイズが 2GB に制限されます。

データベースへのラウンドトリップを抑えるには、パフォーマンスを向上するために、読込みおよび書込みを大きいチャンク・サイズで実行します。

明示的なトランザクションの使用

サーバー・カーソルを自動コミット・モードで使用する場合、LOB データに対するすべての変更は、データベースへの転送後にコミットされます。これによって、レコードセットが遅延更新モードの場合も、LOB データに対する変更および以前の遅延更新は永続的になります。LOB データの変更に対して柔軟にロールバックを行うには、LOB データの操作時に明示的にトランザクションを使用します。

ADO レコードセットおよび LOB

GetChunk()

ADO レコードセット・オブジェクトの GetChunk() メソッドは、LOB データを取り出します。同じ LOB 列に GetChunk() を再度コールすると、データは前回の続きから取り出されます。ただし、現在行を変更するか、または他の LOB 列に読込みまたは書込みを実行した場合は、元の LOB 列に対して Getchunk() を再度コールすると、データは最初から取り出されます。

AppendChunk() を使用した LOB 列へのデータの書込み

ADO レコードセット・オブジェクトの AppendChunk() メソッドは、LOB 列にデータを書き込みます。最初の AppendChunk() メソッドは、すべての既存のデータを上書きします。AppendChunk() を再度コールすると、データが追加されます。ただし、現在行を変更するか、あるいは他の LOB 列のデータを更新または読み込んだ後のコールでは、データが上書きされます。

OLE DB 行セットと LOB

次の OLE DB 行セットのメソッドは、LOB データの読込みおよび書込みを行います。

- IRowset::GetData() および ISequentialStream::Read() は、LOB データを読み込みます。
- IRowsetChange::SetData() および ISequentialStream::Write() は、LOB データを書き込みます。

OraOLEDB コマンドを使用した LOB の操作

OraOLEDB は、次の機能をサポートしています。

- コマンドを介した、LOB 入力バインド・パラメータの指定
- コマンドを介した、BFILE などのすべての型の LOB を含む表の作成

LOB の入力パラメータまたは出力パラメータは、OraOLEDB リリース 8.1.7 以上を使用したストアド・プロシージャでサポートされています。また、この場合のデータベースは、Oracle8*i* リリース 8.1 以上である必要があります。

ADO および LOB の例 1: ファイルからの LOB データの挿入

LOB 列に新しい行を挿入する ADO のサンプルを次に示します。このサンプルを実行するには、コンピュータに「c:¥myfile.txt」という名前のファイルを作成する必要があります。このファイルを作成するには、任意のエディタを使用して、「This is only a test」などの文字データを含めます。プログラムは、この文字データを使用して MULTIMEDIA_TAB 表に CLOB 列を移入します。

その後、プログラムは、新しく挿入したデータをデータベースから取り出して検証します。挿入した行は、プログラムが終了する前に削除されます。

この例では、LOB に対する次の ADO メソッドを使用します。

- `GetChunk` メソッド
- `AppendChunk` メソッド
- `ActualSize` プロパティ

```
Sub Main()
  Dim con As New ADODB.Connection
  Dim cmd As New ADODB.Command

  Dim rst As New ADODB.Recordset

  Dim LogFileName As String
  Dim LogFileNum As Integer
```

```
Dim sql As String        ' SELECT statement
Dim clob_data As Variant ' data from a text file
Dim vardata As Variant   ' data retrieved from clob data in chunks
Dim vardata_len As Long  ' length of the data retrieved from the CLOB column
Dim done As Boolean      ' done = True if finished retrieving all the data
Dim Data As Variant      ' the entire data retrieved from the CLOB column

On Error GoTo ErrorHandler

' open a text file
LogFileName = "c:\myfile.txt"
LogFileNum = FreeFile
Open LogFileName For Input As LogFileNum

' load text from file to a local variable
clob_data = Input$(LOF(LogFileNum), LogFileNum)
Close #LogFileNum

' connect as adldemo/adldemo
con.CursorLocation = adUseServer
con.Open "Provider=OraOLEDB.Oracle;Data Source=db9i;" & _
         "User Id=adldemo;Password=adldemo;"

' open a recordset
sql = "select clip_id, story from MULTIMEDIA_TAB"
rst.Open sql, con, adOpenStatic, adLockOptimistic, adCmdText

' add a new record
rst.AddNew
rst!clip_id = 1234
rst!story.AppendChunk (clob_data)
rst.Update

' fetch entire CLOB data
Do While (Not (done))
  vardata = rst!story.GetChunk(4096)
  If Not (IsNull(vardata)) Then
    Data = Data & vardata
  Else
    done = True
  End If
Loop

' validate fetched data
If Data = clob_data And Len(clob_data) = rst!story.ActualSize Then
  MsgBox "The CLOB data (of " & Len(clob_data) & " bytes) " & _
         "was inserted and retrieved properly!"
```

```
  End If

  ' cleanup
  con.Execute "delete from multimedia_tab where clip_id = 1234"
  rst.Close
  con.Close

  Exit Sub
ErrorHandler:
  MsgBox "Error: " & Err.Description
End Sub
```

14

LOB の事例

この章の内容は次のとおりです。

マルチメディア・リポジトリの構築

次の説明は、『Java Developer's Journal』誌の Samir S. Shah 氏の記事を引用しています。この記事は、『Java Developer's Journal』誌の許可を得て転載しています。

使用するツール製品

- JDeveloper 2.0 および JDK 1.1.7
- Oracle8*i* リリース 8.1.5 以上
- JDBC Thin Driver
- Java Web Server 2.0
- Oracle *inter*Media リリース 8.1.5
- プラットフォーム : Windows 2000 Server

情報リポジトリの構築は、今日のビジネスには必要不可欠です。情報リポジトリによって、ペーパーレスなオフィス環境を構築し、企業の内外でデータを共有できます。

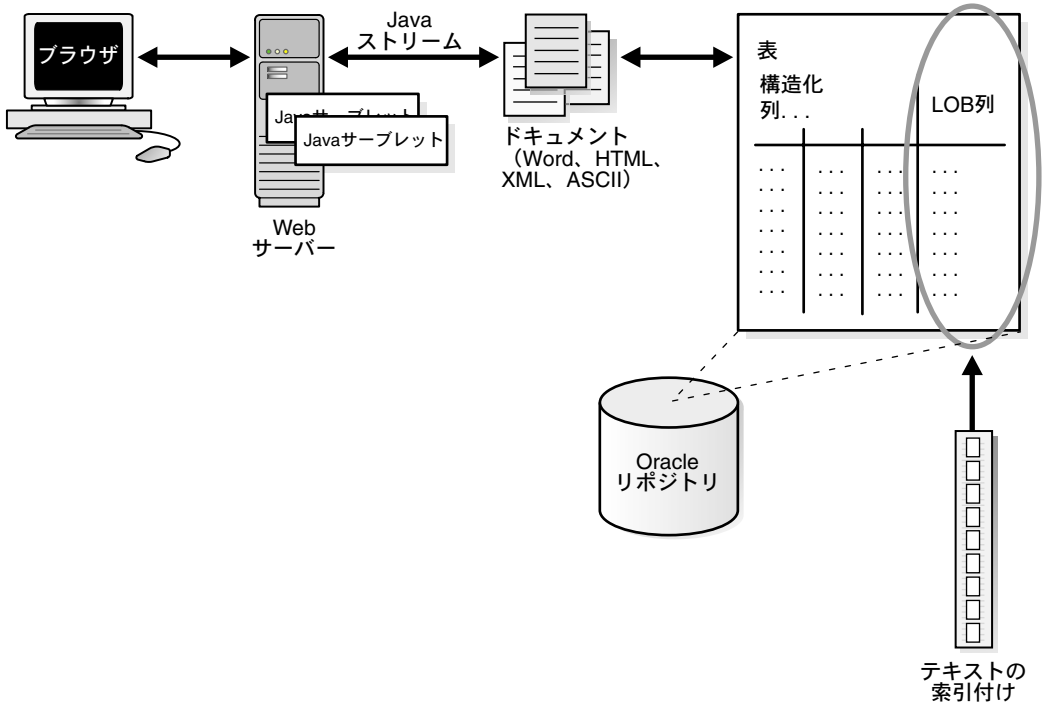
前述のツール製品を使用して、エンタープライズクラスでスケーラブル、かつマルチメディアが豊富な Web 対応の情報リポジトリを構築し、様々な形式のメディアを組み込むことができます。このリポジトリには、ドキュメント・ファイル、ビデオ・クリップ、写真、サウンド・ファイルなどの非構造化データが含まれます。このリポジトリは、Java および Oracle の LOB を使用します。

この項では、データベース表の LOB 列に格納される、Microsoft Word ファイル、HTML ファイル、XML ファイルなどのドキュメントを格納および検索するための、情報リポジトリの構築について説明します。

次の例では、Microsoft Word 形式の履歴書をリポジトリに移入し、Oracle Text を使用して索引付けし、さらに Java ストリームを使用してサーブレットからリポジトリを読み込みます。

図 14-1 を参照してください。

図 14-1 Oracle および Java を使用したデータ・リポジトリ



Java および Oracle8*i* と Oracle9*i* を使用したリポジトリの構築には、いくつかのメリットがあります。ドキュメントは、基本的に、リレーショナル・データベースのトランザクション管理機能および ACID 特性（原子性、並行性、整合性および永続性）を利用できます。これは、内部 LOB に対する変更がコミットまたはロールバックされることを意味します。また、非構造化データはデータベースに格納されるため、アプリケーションがバックアップやリカバリなどのデータベースの機能をシームレスに利用できます。このため、管理者はリレーショナルな情報およびドキュメントのバックアップを、データベースやファイル・システムごとに行う必要がなくなります。

データベース内のデータは、構造化（リレーショナル）データおよび非構造化（ドキュメント・ファイル）データを含め、すべて SQL を使用して読込み、検索およびアクセスが可能です。SQL 文は、JDBC を使用して Java から実行できます。

アプリケーションからの LOB の使用方法

Oracle8*i* および Oracle9*i* では、様々な型の LOB 列をサポートしています。そのうちの 1 つである BLOB 型を使用すると、オーディオ、ビデオ、イメージ、コメントなどのバイナリ情報をデータベース内に格納できます。各行には、最大 4GB のデータを格納できます。この項で説明するアプリケーションは、BLOB データ型を使用して Microsoft Word 形式の履歴書を格納します。

Oracle データベースは、ロケータをデータとともにインラインに格納します。このロケータは、データ（LOB 値）の実際の位置へのポインタです。LOB データは、同一の表または別の表に格納できます。ロケータを使用すると、LOB ロケータ値のみが読み込まれるため、データベースが複数の行を読み込むたびに LOB データをスキャンする必要がなくなり、効率的です。実際の LOB データは、要求した場合にのみ読み込まれます。

Java および LOB を使用する操作では、まず SELECT 文を実行して LOB ロケータを取得し、次に JDBC を使用して LOB の読込みまたは書込みを行います。

注意： JDBC ドライバの Oracle データ型の拡張機能パッケージ oracle.sql は、Oracle データベースの読込みおよび書込みに使用します。

実際の LOB データは、データベースからの Java ストリームとして具体化され、ロケータが表のデータを表現します。次のコードは、従業員番号が 7900 番である従業員の履歴書を読み込みます。従業員番号は、sam_emp 表の「resume」という名前の LOB 列に格納されています。

```
Statement st = cn.createStatement();
     ResultSet rs = st.executeQuery
     ("Select resume from sam_emp where empno=7900");
     rs.next();
oracle.sql.BLOB blob=((OracleResultSet)rs).getBLOB(1);
InputStream is=blob.getBinaryStream();
```

リポジトリの移入

ドキュメントは、Java、PL/SQL、または Oracle SQL*Loader というバルク・ロードのユーティリティを使用して LOB 列に書き込みます。新しい行を挿入するには、次の手順に従います。

1. SQL の INSERT 文を実行して、空の BLOB を作成します。
2. 同じ行を問い合せて、ロケータ・オブジェクトを取得します。このオブジェクトを使用して、ドキュメントを LOB 列に書き込みます。

注意： Java ストリームは、LOB 列に対するドキュメントの書込みに使用します。

---

3. ロケータ・オブジェクトの getBinaryOutputStream() メソッドを使用して、Java の出力ストリームを作成し、列にドキュメントまたはその他のバイナリ情報を書き込みます。たとえば、従業員番号が 9001 である新しい従業員の情報を sam_emp 表に挿入する場合、まず JDBC を使用してすべての構造化情報を空の BLOB とともに挿入します。次に、同じ行の LOB 列（resume）を選択し、oracle.sql.BLOB オブジェクト（ロケータ）を取得します。

4. oracle.sql.BLOB オブジェクトから Java の出力ストリームを作成します。たとえば、従業員番号が 9001 である新しい従業員の情報を sam_emp 表に挿入する場合、まず JDBC を使用してすべての構造化データを空の BLOB とともに挿入します。次に、同じ行の LOB 列（resume）を選択し、oracle.sql.BLOB オブジェクト（ロケータ）を取得します。最後に、このオブジェクトから Java の出力ストリームを作成します。次に例を示します。

```
st.execute("INSERT INTO sam_emp(empno, resume)
       VALUES(9001,empty_blob())");
ResultSet rs = st.executeQuery(
   "select resume from sam_emp where empno=9001 for update");
   rs.next();
oracle.sql.BLOB blob = ((OracleResultSet)rs).getBLOB(1);
OutputStream os = blob.getBinaryOutputStream();
```

   オプションで、java.awt.FileDialog クラスおよび java.io パッケージを使用して、PC からファイルを動的に選択し、読み込むことができます。その後、前述のコードを使用して、ファイルを LOB 列にロードします。

ドキュメントの検索および取得の方法は、ドキュメントのロード方法には依存しません。たとえば、ドキュメントを、PL/SQL または SQL*Loader を使用して格納し、Java サーブレットを使用して取得できます。

例 1: PL/SQL を使用した BLOB 列への Word ドキュメントの挿入

次のコード例（手順 2 ～ 5）では、sam_emp 表の resume 列に MyResume.doc を挿入します。

1. Oracle にディレクトリ・オブジェクトを作成します。ここでは、C:¥MY_DATA ディレクトリを示す MY_FILES という名前のディレクトリ・オブジェクトの作成方法を示します。

   この手順を実行するには、Oracle の CREATE DIRECTORY 権限が必要です。

   ```
   create or replace directory MY_FILES as 'C:¥MY_DATA';
   ```

2. 表に空の BLOB 列を挿入し、ロケータを戻します。

3. BFILE データ型を使用して、手順 1 で作成したディレクトリからロードする Word ファイルを指すように指定します。
4. ファイルをオープンし、手順 2 のロケータを使用してファイルを挿入します。
5. ファイルをクローズし、トランザクションをコミットします。

```
declare
    f_lob   bfile;
    b_lob   blob;

begin

    insert into sam_emp(empno,ename,resume)
 values ( 9001, 'Samir',empty_blob() )
 return documents into b_lob;

    f_lob := bfilename( 'MY_FILES', 'MyResume.doc' );
    dbms_lob.fileopen(f_lob, dbms_lob.file_readonly);
    dbms_lob.loadfromfile
   ( b_lob, f_lob, dbms_lob.getlength(f_lob) );
    dbms_lob.fileclose(f_lob);

    commit;

end;
/
```

リポジトリの検索

LOB 列に格納されているドキュメントは、Oracle Text を使用して索引付けできます。Oracle Text は、ファジー、ステミング、プロキシ、フレーズなどの拡張検索機能を提供します。また、テーマや要点別の検索も行えます。ドキュメントの索引付けには、データベース・コマンドの「CREATE INDEX」を使用します。

参照： 次のマニュアルを参照してください。

- 『Oracle Text アプリケーション開発者ガイド』
- 『Oracle Text リファレンス』

sam_emp 表の resume 列に対する索引の生成方法

次のコード例は、sam_emp 表の resume 列に対する索引の生成方法を示しています。索引が生成されると、Java アプリケーションは SELECT 文を発行するのみでリポジトリを検索できます。

次に示す手順では、sam_emp 表の resume 列に格納されたすべての Word 形式の履歴書を索引付けします。これによって、SQL を使用して履歴書を検索できるようになります。

1. 主キーが表に存在しない場合は追加します。empno を sam_emp 表の主キーに設定するには、次のコマンドを実行します。

   ```
   alter table sam_emp add constraint
   pk_sam_emp primary key(empno);
   ```

2. 管理者に依頼して、テキスト索引を作成する権限（ctxapp ロール）を取得します。
3. 適切なフィルタ・オブジェクトを使用して、索引を生成します。フィルタは、ドキュメントを索引付けするためにワード・プロセッサからテキストを抽出する方法を決定します。書式化されたドキュメントも、プレーン・テキストと同様に扱えます。

   ```
   create index ctx_doc_idx on sam_emp(resume)
   indextype is ctxsys.context parameters
   ('filter CTXSYS.INSO_FILTER');
   ```

   参照： フィルタの詳細なリストは、次のマニュアルを参照してください。

   - 『Oracle Text アプリケーション開発者ガイド』
   - 『Oracle Text リファレンス』

MyServletCtx サーブレット

次のコード例は、サーブレット「MyServletCtx」を示しています。サーブレットは、パラメータとして受け取った項目を sam_emp 表の resume 列内で検索します。検索基準に一致した行があれば、HTML 表形式でそれを戻します。HTML 表にある従業員名は、他のサーブレット「MyServlet」にハイパーリンクされており、このサーブレットがデータベースから履歴書全体を元の書式で読み込みます。

MyServletCtx.java

```
12345678901234567890123456789012345678901234567890123456789012
package package1;
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.sql.*;

/**
* This servlet searches documents stored in Oracle8i
* database repository using SQL and JDBC. The hit
* list is displayed in html table with hyper links.
* JDK 1.1.7 and Oracle Thin JDBC 1.22 complient
* driver is used.
*
* @author Samir Shah
* @version 1.0
**/
public class MyServletCtx extends HttpServlet{
   Connection cn;

   public void init(ServletConfig parm1)
   throws ServletException {
    super.init( parm1);
    try{
    DriverManager.registerDriver(
 (new oracle.jdbc.driver.OracleDriver()));
    cn =DriverManager.getConnection
 ("jdbc:oracle:thin:@sshah:1521:o8i",
          "scott", "tiger");
    }
    catch (SQLException se){se.printStackTrace();}
  }
```

```
public void doGet(HttpServletRequest req,
   HttpServletResponse res) throws IOException{

    doPost(req,res);
}

public void doPost(HttpServletRequest req,
  HttpServletResponse res) throws IOException{

  PrintWriter out = res.getWriter();
  res.setContentType("text/html");

  //The term to search in resume column
  String term = req.getParameter("term");
  if (term == null)
     term="security";

  out.print("<html>");
  out.print("<body>");
  out.print("<H1>Search Result</H1>");
  out.print("<table border=1 bgcolor=lightblue>");
  out.print("<tr><th>ID#</th><th>Name</th></tr>");
  out.print("<tr>");
  try{
   Statement st = cn.createStatement();

   //search the term in resume column using SQL
   String query =
       "Select empno,ename from sam_emp" +
       " where contains(resume,'"+term+"')>0";

   ResultSet rs = st.executeQuery(query);

    while (rs.next()){
      out.print("<td>"+ rs.getInt(1)+"</td>");
      out.print("<td>" +
        "<A HREF=http://sshah:8080/" +
        "servlet/MyServlet?term=" +
        rs.getString(1) +
        " target=Document>" +
        rs.getString(2) +
        "</A></td>");
      out.print("</tr>");
    }
```

```
      out.print("</table>");
      out.print("</body>");
      out.print("</html>");
     }//try
     catch (SQLException se){se.printStackTrace();}

  }
}
```

リポジトリからのデータの取得

Java を使用したドキュメントの取出しは、リポジトリへのドキュメントの書込みと似ています。14-4 ページの「アプリケーションからの LOB の使用方法」では、データベースからの LOB の読込み方法を説明しています。

次の「MyServlet」内のコードは、sam_emp 表から Microsoft Word 形式の履歴書を読み込みます。その後コンテンツ・タイプを設定し、出力ストリームを使用してブラウザにストリーム・アウトします。

MyServlet.java

```
12345678901234567890123456789012345678901234567890123456789012
package package1;

import javax.servlet.*;
import javax.servlet.http.*;
import java.sql.*;
import java.io.*;
import oracle.jdbc.driver.*;
import oracle.sql.*;  //for oracle.sql.BLOB

/**
* This class reads the entire document from the
* resume LOB column. It takes one parameter,term,
* to search a specific employee from the sam_emp
* table and returns the doucement stored in that
* row.
*
* JDK 1.1.7, Oracle Thin JDBC 1.22 complient driver
* Use Oracle JDBC Type extends package oracle.sql.
*
* @author Samir Shah
* @version 1.0
**/
public class MyServlet extends HttpServlet{
  Connection cn;
```

```
public void doGet(HttpServletRequest req,
HttpServletResponse res)
{
 try{
  doPost(req,res);
 }catch (IOException ie){ie.printStackTrace();}
}

public void init(ServletConfig parm1)
throws ServletException
{

  super.init( parm1);
  try{
  DriverManager.registerDriver(
  (new oracle.jdbc.driver.OracleDriver()));
  cn =DriverManager.getConnection(
      "jdbc:oracle:thin:@sshah:1521:o8i",
      "scott", "tiger");
  }
  catch (SQLException se){se.printStackTrace();}
}

public void doPost(HttpServletRequest req,
  HttpServletResponse res) throws IOException
{
   InputStream is=null;
   oracle.sql.BLOB blob=null;

   res.setContentType("application/msword");
   OutputStream os = res.getOutputStream();
   String term = req.getParameter("term");

   if (term==null)
      term="9001";

   try{
   Statement st = cn.createStatement();
   ResultSet rs = st.executeQuery
        ("Select resume from sam_emp"+
         " where empno="+term);

    while (rs.next()){
        blob=((OracleResultSet)rs).getBLOB(1);
        is=blob.getBinaryStream();
    }
```

```
      int pos=0;
      int length=0;
      byte[] b = new byte[blob.getChunkSize()];

      while((length=is.read(b))!= -1){
       pos+=length;
       os.write(b);
      }
     }//try
     catch (SQLException se)
{
     se.printStackTrace();
        }
     finally {
               is.close();
     }

  }

}
```

要約

この項では、LOB データ型および Java を使用した Word ドキュメントの格納、検索および取得方法を説明しました。

これ以外にも、Oracle9*i* データベースを使用して、XML 文書の格納、索引付け、解析および変換が行えます。データベース内に XML 文書を格納すると、リレーショナル・データおよび XML データに対して複数のリポジトリを管理する必要がなくなります。Oracle9*i* および Oracle9*i* Application Server は XML に対応しているため、Oracle XML Parser for Java を実行し、アプリケーション・サーバーに出力する前にデータベース内の XML ファイルを解析および変換できます。

LOB ベースの Web サイトの構築 : 最初の手順

課題

LOB および *inter*Media をベースにした Web サイトを設計および構築します。この Web サイトには、ユーザーが特定のサムネイルをクリックして 6 ～ 8 秒のビデオ・クリップを見ることができるようなビデオのサムネイルを含める必要があります。

最初の手順の実行例

前述の LOB ベースの Web サイトをセットアップする例を示します。

1. データベース・サーバーに、Oracle9*i*（*inter*Media を含む）をインストールします。
2. Oracle9*i* Appplication Server、IIS、Netscape Web サーバー、Apache などの Web サーバーをインストールします。
3. Web サーバーに、*inter*Media Web Agent をインストールします。
4. クライアント（PC）に、*inter*Media ClipBoard をインストールします。
5. サーバー上に、3 つ以上の次のような列を含む表を作成します。

   ```
   create table video_clips (
   move_id integer,
   thumbnail ordsys.ordimage,
   movie ordsys.ordvideo);
   ```

   注意 2 を参照してください。
6. メディア・コンテンツ（イメージ、ムービーなど）を収集またはキャプチャします。
7. デジタル・カメラまたはスキャナを使用する場合は、*inter*Media ClipBoard を使用すると効果的です。
8. *inter*Media ClipBoard を使用して、手順 5 でデータベース内に作成した表へメディア・コンテンツをアップロードします。
9. *inter*Media ClipBoard とともに DreamWeaver、FrontPage などの HTML オーサリング・ツールを使用して、Web ページを作成します。
10. *inter*Media ClipBoard を使用して、キャプション付きのサムネイルを追加します。サムネイルに、ムービー・クリップへのハイパーリンクを作成します。この時点で、単独のストリーミング・サーバーを使用しないことをお薦めします。たとえば、ムービーを Apple QuickTime のファイルとしてエンコードするという方法があります。処理が正しく行われていれば、ダウンロードと同時にムービーの再生が始まります。これは、正確には「ストリーミング」ではありません。ただし、十分な帯域幅があれば、適切な結果が得られます。

11. プラグインが必要か、領域要件が十分かを考慮します。たとえば、100 本のムービー・クリップがあり、それらすべての再生におよそ 30 分かかる場合を想定してみます。プラグインは必要ないため、Real Networks プラグインも使用しません。

必要なディスク領域は、フレーム・サイズ、フレーム・レートおよび圧縮の設定により決定します。ビデオを、720 × 480 ピクセルの解像度、毎秒 30 フレーム（30fps）で 1 秒間再生する場合、およそ 3.6MB のディスク領域が必要になります。720 × 480 の解像度は Web 上ではかなり大きなデータですが、イントラネット上であれば適切です。30fps では滑らかに再生できますが、そこまでの精度を要求しない場合もあります。サンプルを試し、320 × 240 での再生の状態を確認してください。細部まで正しく表示されているかどうかを確認し、不適切な場合は、表示が適切になるように解像度を上げてください。

注意 1:

- この設定は複雑になる場合があります。すべてをインストールして構成するだけで、かなりの工程を必要とします。Oracle DBA およびコンサルタントの支援を要請してください。
- 可能であれば、DB_BLOCKS_SIZE を 8K に設定し、DB_BLOCK_BUFFERS に可能なかぎり大きい値を設定してください。

注意 2: 前述の例は、単純化した表の作成例です。通常は、ORDImage および ORDVideo 内部の LOB に対する LOB 記憶域句が必要です。また、これらの LOB に対する個別の表領域、CHUNK 値 32768、VIDEO LOB に対する NOCACHE、IMAGE LOB に対する CACHE も設定する必要があります。

参照 : 『Oracle *inter*Media 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。

A

Unified Modeling Language 図

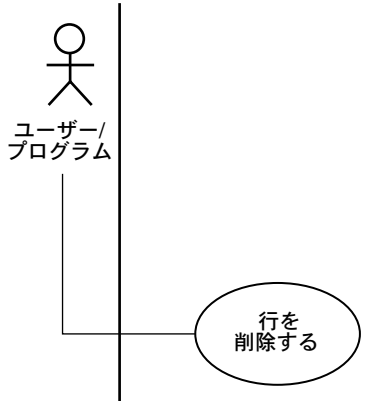
このマニュアルでは、アドバンスト・キューイングで使用されているテクノロジの説明方法として、Unified Modeling Language（UML）を使用して利用図を説明しています。この付録では、利用図および UML 表記法の概略を示します。

この付録の内容は次のとおりです。

利用図

利用図では、1 次利用は、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いずれでもかまいません。アクターは、図 A-1 に示すとおり、アクションを囲む楕円（吹出し）で示される 1 次利用に接続されます。

図 A-1 1 次利用



1 次利用を完了するには、他の操作が必要になる場合があります。図 A-2 の場合

- キュー名を指定する

が、次に示す操作を完了するために必要な下位操作（2 次利用）の 1 つです。

- メッセージをエンキューする

1 次利用から、他の必要な操作（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 A-2 下位操作を伴う 1 次利用

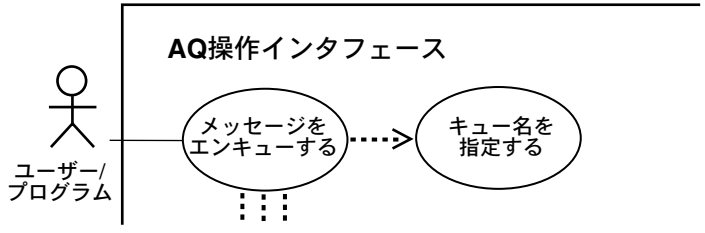


図 A-3 に示すとおり、影付きの 2 次利用は、固有の利用図に展開されます。これによって、次のことが簡単になります。

- オペレーション・ロジックの理解
- 複数ページにわたる複雑なオペレーションの継続

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利用図に展開されます。

図 A-3 2 次利用を示す影付きの利用図

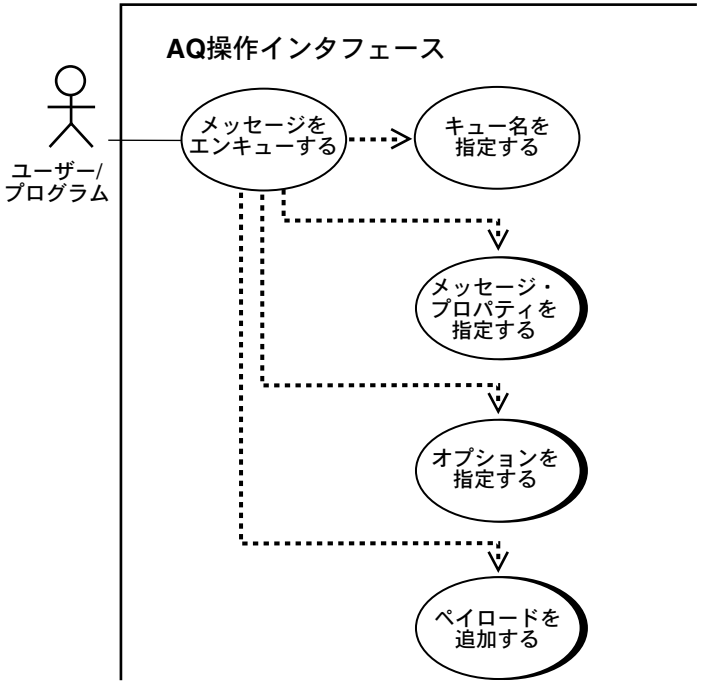


図 A-4（抜粋）には、展開された利用図が示されています。標準的な図は、通常アクターから始まりますが、ここでは利用方法自体がサブオペレーションへの出発点になっています。この例では、

- ペイロードを追加する

の展開ビューは、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 A-4 展開された利用図

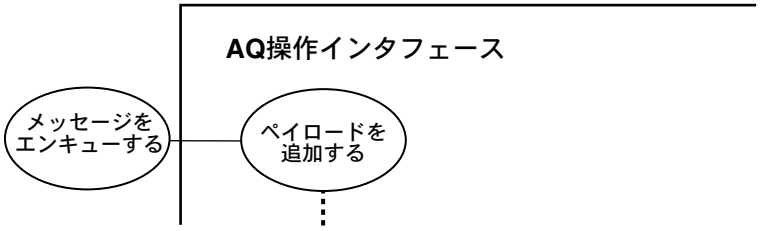
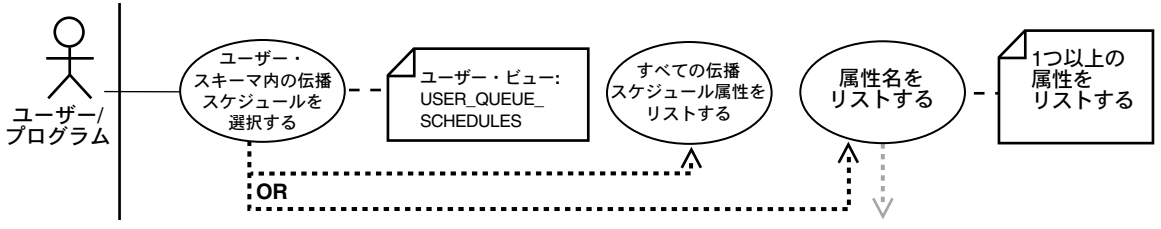


図 A-5 では、注釈ボックスの使用方法を示します。

- 注釈ボックスには、代替名を示すことができます。この場合、「ユーザー・スキーマ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。
- 注釈ボックスでは、利用操作を修飾できます。この場合、「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、すべての伝播スケジュール属性をリストしない場合は、属性を 1 つ以上リストする必要があることが示されています。

図 A-5 注釈ボックス



利用図の破線の矢印は、依存性を示します。図 A-6 では、

- 一時 LOB を解放する

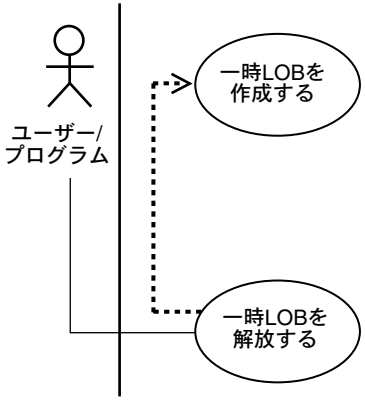
を実行するためには、まず

- 一時 LOB を作成する

必要があることを示しています。

矢印の宛先は、最初に実行する必要があるオペレーションを示します。

図 A-6 依存性



利用方法および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。図 A-7 の例では、まず

- 通知を登録する

を実行した後で、

- 通知を受け取る

必要があります。

図 A-7 利用操作および下位操作の関係

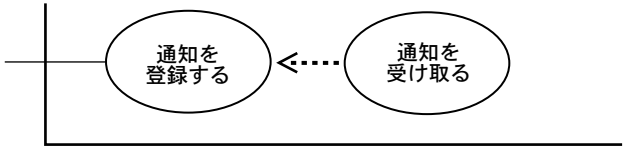


図 A-8 には、OR 条件の分岐パスが示されています。ビューを起動する際に、すべての属性のリストを選択するか、1 つ以上の属性を表示するかを選択できます。灰色の矢印は、表示させる属性を指定できることを示しています。

図 A-8 OR 条件の分岐パス

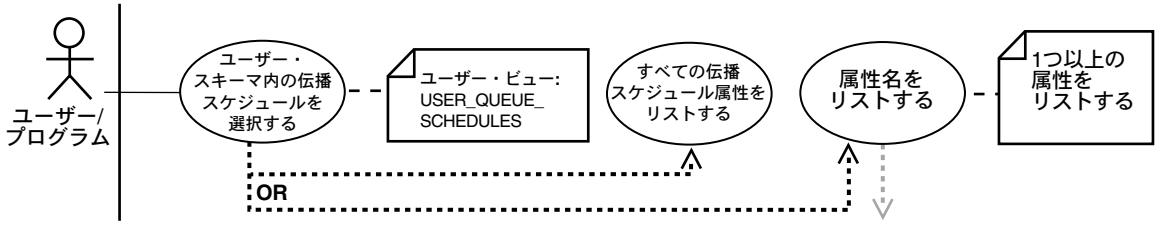


図 A-9 では、黒の破線の矢印は、宛先のオペレーションが必須であることを示しています。灰色の破線の矢印は、宛先の操作がオプションであることを示しています。この例では、

- 追加を書き込む

を LOB に対して実行するには、まず

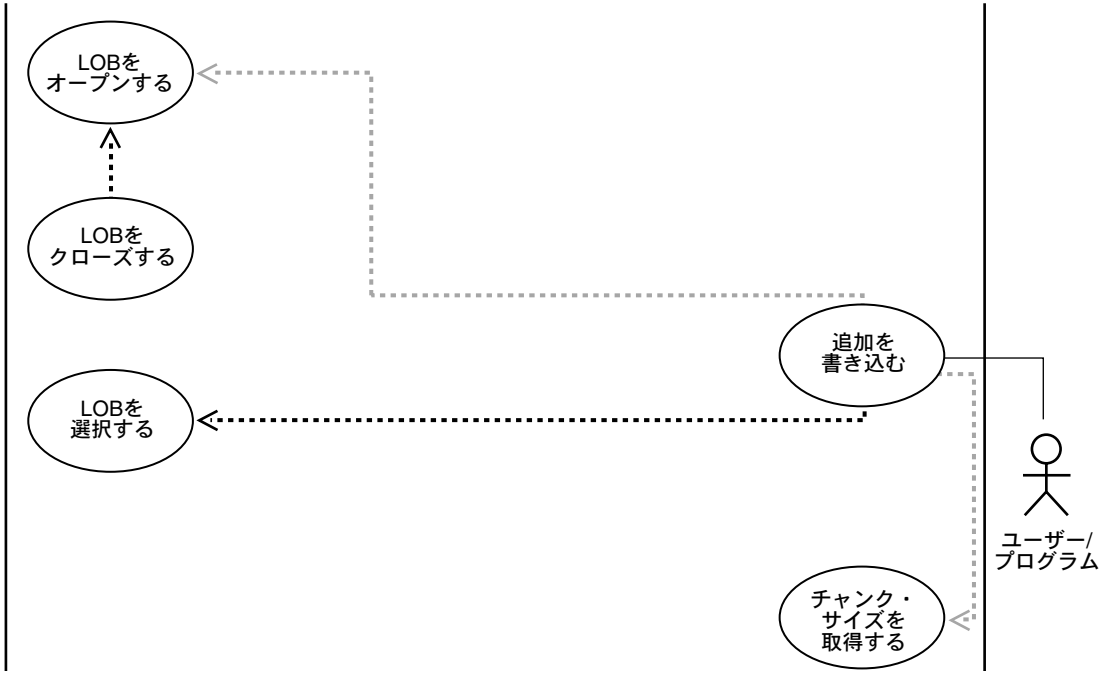
- LOB を選択する

必要があります。オプションとして、次の 2 つから選択できます。

- LOB をオープンするか、またはチャンク・サイズを取得する。

この図は、「LOB をオープンする」場合は、後で、「LOB をクローズする」必要があることを示しています。

図 A-9 必須操作およびオプションの操作



状態図

状態図は、ビューの属性を示しています。ビューの属性には、可視および不可視の 2 つの状態が存在します。この例では、ビューのすべての属性を示すために、利用図の下に状態図（図の最下部の灰色の領域に示されているキュー、名前、アドレスおよびプロトコルのボックス）が追加されています。

図 A-10 は、キュー・サブスクライバを問い合せるためにビューが使用されることを示します。4 つの属性は、単独で、いくつか組み合せて、またはすべてを指定できます。

図 A-10 ビューの属性を示すための利用図および状態図

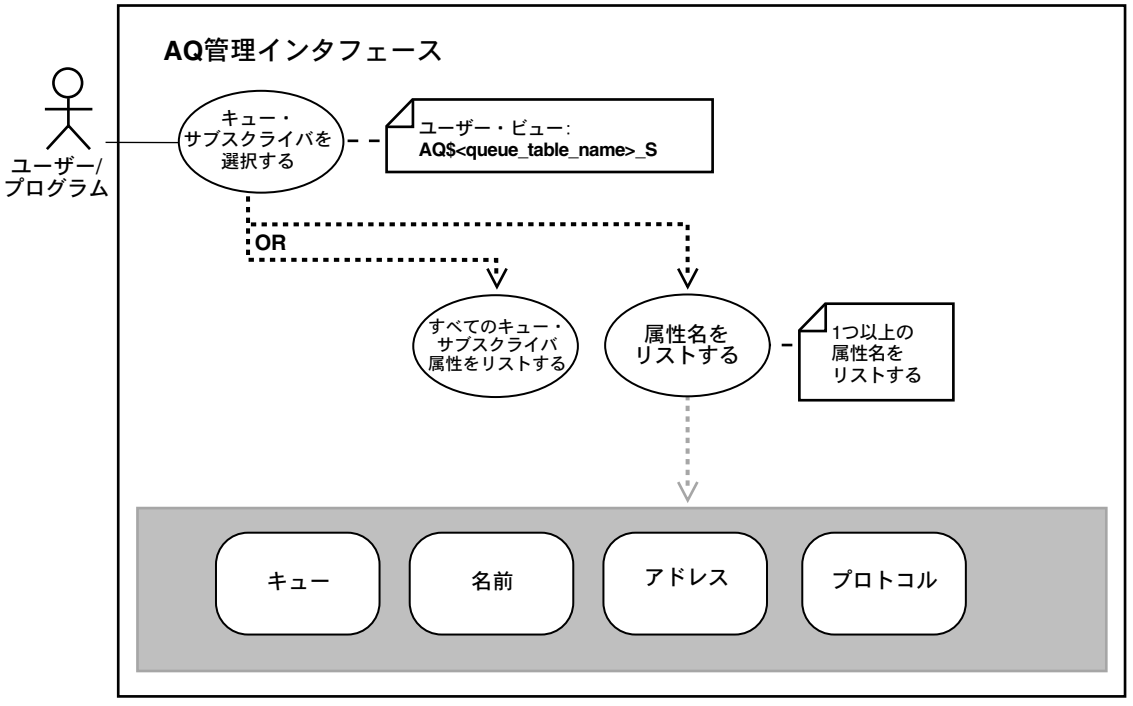
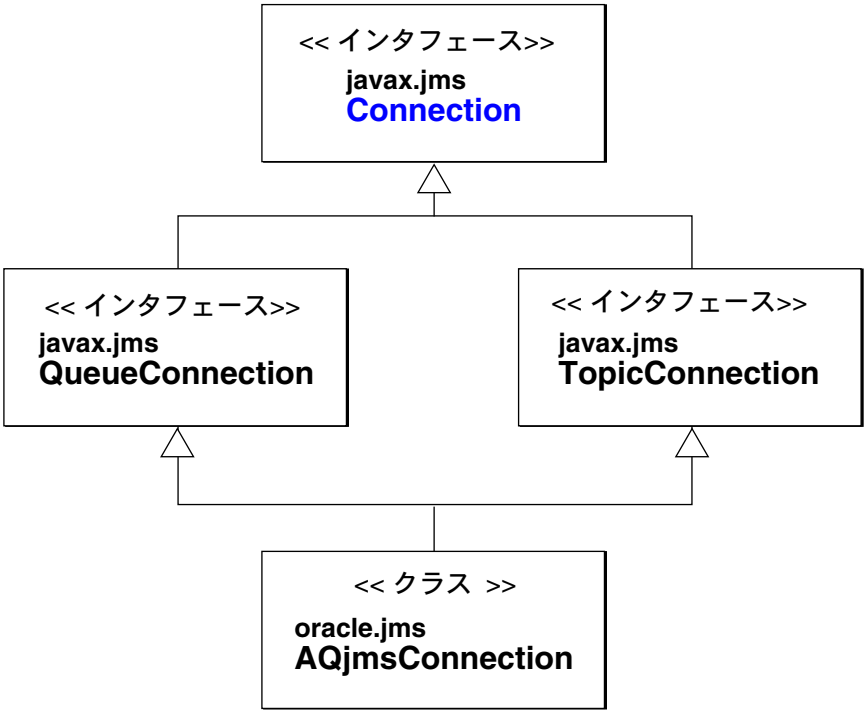


図 A-11 のクラス図は、次の情報を表示しています。

- クラス、インタフェースおよび例外が相互関係を伴っているかどうか（**<< インタフェース >>** など、<<>> を使用）
- クラスが検出されたパッケージの名前（**oracle.jms** など）
- クラスの名前（**AQjmsConnection** など）

図 A-11 クラス、インタフェースおよび例外を表すクラス図



B

マルチメディア・スキーマ

注意： マルチメディアのサンプル・スキーマは、Oracle9*i* では提供されていません。このマニュアルのほぼすべての LOB の例は、マルチメディア・スキーマのかわりに使用される製品メディア（PM）・スキーマに移行されていますが、移行されていない例もあるため、この付録でそれらの例のバックグラウンド情報を提供します。

詳細は、1-9 ページの「このマニュアルでの例」を参照してください。

この付録の内容は次のとおりです。

典型的なマルチメディア・アプリケーション

Oracle9*i* 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 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。

次のマルチメディアの使用例について考えてみます。

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）。ただし、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・オーサリング処理では、高い類似性があります。

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しょうが、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。

単なる例としての使用例

この付録の目的は、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典型的な使用例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。

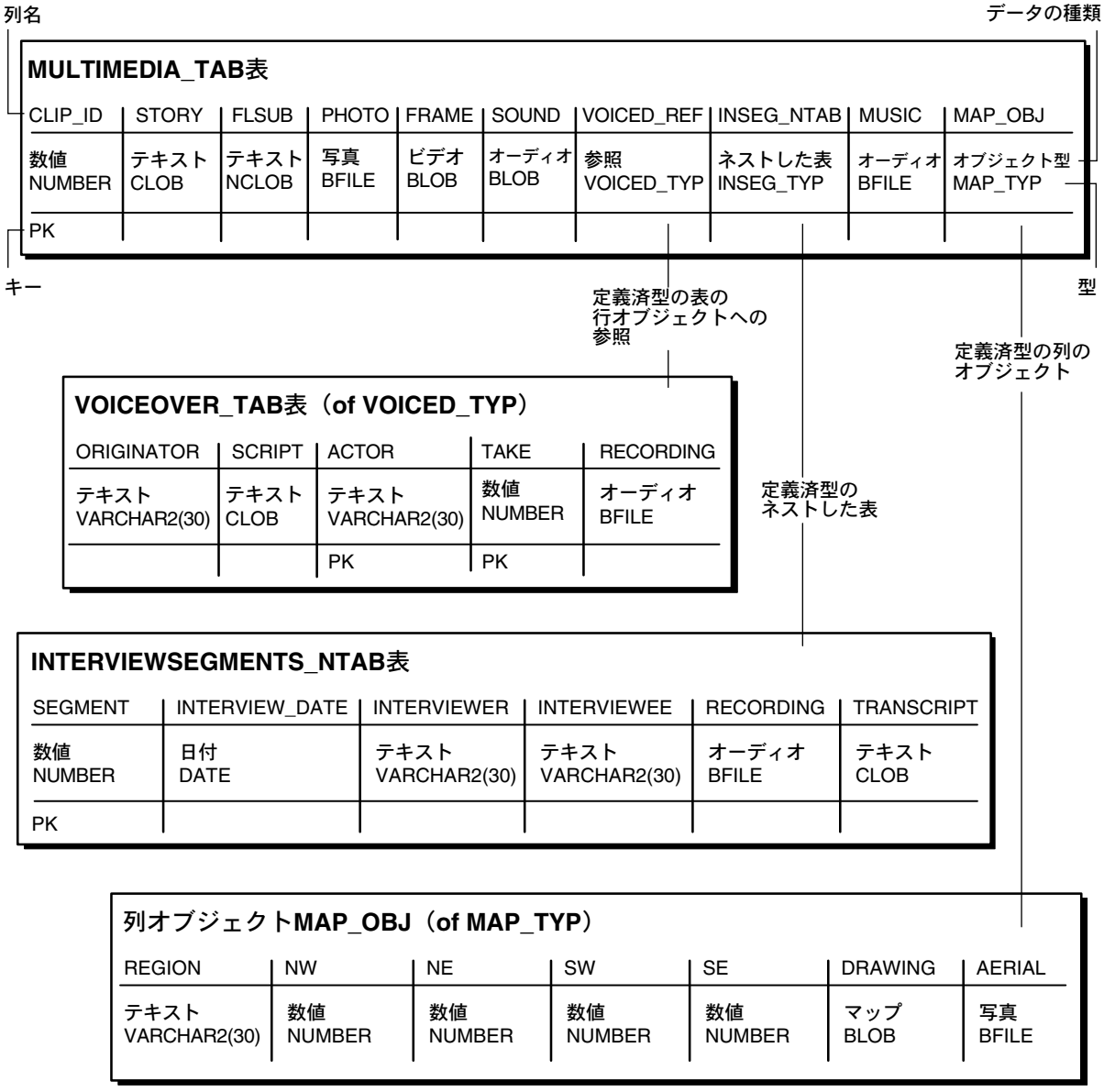
マルチメディア・スキーマ

図 B-1 に、このマニュアルの例で使用するマルチメディア・スキーマを示します。

このマルチメディア・スキーマは、次の構成要素で構成されます。

- Multimedia_tab 表
- VoiceOver_tab 表
- ネストした表 INSEG_NTAB
- 列オブジェクト MAP_OBJ

図 B-1 マルチメディア・スキーマ



Multimedia_Tab 表

図 B-1「マルチメディア・スキーマ」に、Multimedia_tab 表の構造を示します。Multimedia_tab 表の列を次に示します。

- **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 次記憶デバイスに格納されるため、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、本人が亡くなってしまった、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。

- **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ネストした表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 型のネストした表 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。

  この例では、ネストした表の 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。

- **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 OS 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 型およびネストした表の場合のように、オブジェクト型が含む LOB の数には制限がありません。

マルチメディア・スキーマ作成用のスクリプト

マルチメディア・スキーマの作成に使用するスクリプトは次のとおりです。

```
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(),
   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;
```

参照： 10-5 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。

このスクリプトは、次のファイル内の Oracle9*i* の $HOME デモ・ディレクトリにもあります。

- lobdemo.sql
- adloci.sql

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。

- 『Oracle *inter*Media ユーザーズ・ガイドおよびリファレンス』
- 『Oracle *inter*Media Java Classes ユーザーズ・ガイドおよびリファレンス』
- 『Oracle Text アプリケーション開発者ガイド』
- 『Oracle Text リファレンス』

索引

数字

A

B

C

D

E

F

G

I

J

L

N

O

P

R

S

T

U

V

W

X

あ

い

う

え

お

か

き

く

こ

さ

し

す

せ

そ

た

ち

つ

て

と

な

は

ひ

ふ

へ

ほ

ま

も

よ

り

れ

ろ

わ