Oracle C++ Call Interface

プログラマーズ・ガイド

リリース 2（9.2）

2002 年 7 月

部品番号：J06296-01

ORACLE®

Oracle C++ Call Interface プログラマーズ・ガイド , リリース 2（9.2）

部品番号：J06296-01

原本名：Oracle C++ Call Interface Programmer's Guide, Release 2 (9.2)

原本部品番号：A96583-01

原本著者：Roza Leyderman

原本協力者：Sandeepan Banerjee, Subhranshu Banergee, Kalyanji Chintakayala, Krishna Itikarlapalli, Maura Joglekar, Ravi Kasamsetty, Srinath Krishnaswamy, Shoaib Lari, Geoff Lee, Gayathri Priyalakshmi, Den Raphaely, Rajiv Ratnam, Rekha Vallam, Valarie Moore

目次

2 リレーショナル・プログラミング

3 オブジェクト・プログラミング

4 データ型

5 LOB の概要

6 メタデータ

7 Object Type Translator ユーティリティの使用方法

索引

図

表

はじめに

Oracle C++ Call Interface（OCCI）は、Application Program Interface（API）です。OCCI を使用すると、C++ 言語で作成したアプリケーションで 1 つ以上の Oracle データベース・サーバーと交信できます。OCCI によって、SQL 文の処理やオブジェクト操作など、Oracle データベース・サーバーで実行できるすべてのデータベース操作を、使用しているプログラムで実行できます。

ここでは、次の項目について説明します。

対象読者

『Oracle C++ Call Interface プログラマーズ・ガイド』は、プログラマ、システム・アナリスト、プロジェクト・マネージャ、および次の作業を担当したり、またその作業の習得に関心を持つ Oracle ユーザーを対象としています。

- Oracle 環境でのデータベース・アプリケーションの設計と開発
- Oracle 環境で実行するための既存のデータベース・アプリケーションの変換
- データベース・アプリケーション開発の管理

このマニュアルを活用するには、オブジェクト指向プログラミングの基本概念を理解し、Structured Query Language（SQL）の知識、および C++ によるアプリケーション開発に関する作業上の知識が必要です。

このマニュアルの構成

このマニュアルの構成は、次のとおりです。

第 I 部「OCCI プログラミング」

第 I 部（第 1 章～第 7 章）では、OCCI を使用してプログラミングし、Oracle データベースのリレーショナル・データおよびオブジェクト・データにアクセスできるスケーラブルなアプリケーション・ソリューションを作成する方法を説明します。

第 1 章「OCCI の概要」

この章では、OCCI の概要を説明し、OCCI の説明で使用される特別な用語や表記規則を示します。

第 2 章「リレーショナル・プログラミング」

この章では、OCCI プログラムを開発する上で必要な基本概念を説明します。OCCI プログラムに組み込む必要のある基本的なステップと、エラー・メッセージを取り出して解読する方法を説明します。

第 3 章「オブジェクト・プログラミング」

この章では、OCCI を使用して Oracle データベース・サーバー内のオブジェクトにアクセスする場合の概念を説明します。基本的なオブジェクトの概念とオブジェクト・ナビゲーショナル・アクセス、およびオブジェクト・リレーショナル・アプリケーションの基本構造についても説明します。

第 4 章「データ型」

この章では、Oracle の内部データ型と外部データ型、および必要なデータ変換について説明します。

第 5 章「LOB の概要」
この章では、LOB および関連するクラスとメソッドの概要を説明します。

第 6 章「メタデータ」
この章では、スキーマ・オブジェクトとそれに対応する要素の情報を、`MetaData()` メソッドを使用して取得する方法を説明します。

第 7 章「Object Type Translator ユーティリティの使用方法」
この章では、Object Type Translator（OTT）を使用してデータベース・オブジェクト定義を C++ の表現に変換し、OCCI アプリケーションで使用する方法を説明します。

第 II 部「API リファレンス」
第 II 部（第 8 章）では、OCCI のクラスとメソッドについて説明します。

第 8 章「OCCI のクラスとメソッド」
この章では、C++ 用の OCCI のクラスとメソッドについて説明します。

第 III 部 「付録」
第 III 部（付録 A）では、OCCI のデモ・プログラムを提示します。

付録 A「OCCI デモ・プログラム」
この付録では、OCCI のデモ・プログラムを提示し、各デモ・プログラムのコードを記載します。

関連文書
詳細は、次の Oracle リソースを参照してください。

- 『Oracle9*i* データベース概要』
- 『Oracle9*i* SQL リファレンス』
- 『Oracle9*i* データベース管理者ガイド』
- 『Oracle9*i* アプリケーション開発者ガイド - オブジェクト・リレーショナル機能』
- 『Oracle9*i* データベース新機能』
- 『Oracle Call Interface プログラマーズ・ガイド』
- 『Oracle9*i* Database Server Cache Concepts and Administration Guide』

このマニュアルに含まれる例の多くでは、Oracle のインストール時にデフォルトでインストールされるシード・データベースのサンプル・スキーマを使用しています。これらのスキーマがどのように作成されているか、およびその使用方法については、『Oracle9*i* サンプル・スキーマ』を参照してください。

リリース・ノート、インストレーション・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/`

OTN-J のユーザー名とパスワードを取得済みであれば、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

このマニュアル・セットの本文とコード例に使用されている表記規則について説明します。

- 本文の表記規則
- コード例の表記規則

本文の表記規則

本文では、特別な用語が一目でわかるように、様々な表記規則が使用されています。次の表は、本文の表記規則とその使用例を示しています。

規則	意味	例
太字	太字は、本文中に定義されている用語または用語集に含まれている用語、あるいはその両方を示します。	この句を指定する場合は、**索引構成表**を作成します。
固定幅フォントの大文字	固定幅フォントの大文字は、システムにより指定される要素を示します。この要素には、パラメータ、権限、データ型、Recovery Manager 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、パッケージとメソッド、システム指定の列名、データベース・オブジェクトと構造体、ユーザー名、およびロールがあります。	この句は、`NUMBER` 列に対してのみ指定できます。 `BACKUP` コマンドを使用すると、データベースのバックアップを作成できます。 `USER_TABLES` データ・ディクショナリ・ビューの `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。

規則	意味	例
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サンプルのユーザー指定要素を示します。この要素には、コンピュータ名とデータベース名、ネット・サービス名、接続識別子の他、ユーザー指定のデータベース・オブジェクトと構造体、列名、パッケージとクラス、ユーザー名とロール、プログラム・ユニット、およびパラメータ値があります。 **注意:** 一部のプログラム要素には、大文字と小文字の両方が使用されます。この場合は、記載されているとおりに入力してください。	`sqlplus` と入力して SQL*Plus をオープンします。 パスワードは `orapwd` ファイルに指定されています。 データ・ファイルと制御ファイルのバックアップを `/disk1/oracle/dbs` ディレクトリに作成します。 `department_id`、`department_name` および `location_id` の各列は、`hr.departments` 表にあります。 初期化パラメータ `QUERY_REWRITE_ENABLED` を `true` に設定します。 `oe` ユーザーで接続します。 これらのメソッドは `JRepUtil` クラスに実装されます。
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。*`old_release`* は、アップグレード前にインストールしたリリースです。

コード例の表記規則

コード例は、SQL、PL/SQL、SQL*Plus またはその他のコマンドラインを示します。次のように、固定幅フォントで、通常の本文とは区別して記載しています。

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

次の表に、コード例の記載上の表記規則と使用例を示します。

規則	意味	例
[ ]	大カッコで囲まれている項目は、1 つ以上のオプション項目を示します。大カッコ自体は入力しないでください。	`DECIMAL (`*`digits`* `[ , `*`precision`* `])`
{ }	中カッコで囲まれている項目は、そのうちの 1 つのみが必要であることを示します。中カッコ自体は入力しないでください。	`{ENABLE | DISABLE}`

規則	意味	例
|	縦線は、大カッコまたは中カッコ内の複数の選択肢を区切るために使用します。オプションのうち 1 つを入力します。縦線自体は入力しないでください。	{ENABLE | DISABLE} [COMPRESS | NOCOMPRESS]
. . .	水平の省略記号は、次のどちらかを示します。	
	■ 例に直接関係のないコード部分が省略されていること。	CREATE TABLE ...AS *subquery*;
	■ コードの一部が繰返し可能であること。	SELECT *col1*, *col2*, ..., *coln* FROM employees;
. . .	垂直の省略記号は、例に直接関係のない数行のコードが省略されていることを示します。	
その他の表記	大カッコ、中カッコ、縦線および省略記号以外の記号は、示されているとおりに入力してください。	acctbal NUMBER(11,2); acct CONSTANT NUMBER(4) := 3;
イタリック	イタリックの文字は、特定の値を指定する必要のあるプレースホルダまたは変数を示します。	CONNECT SYSTEM/*system_password* DB_NAME = *database_name*
大文字	大文字は、システムにより指定される要素を示します。これらの用語は、ユーザー定義用語と区別するために大文字で記載されています。大カッコで囲まれている場合を除き、記載されているとおりの順序とスペルで入力してください。ただし、この種の用語は大 / 小文字区別がないため、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、ユーザー指定のプログラム要素を示します。たとえば、表名、列名またはファイル名を示します。 **注意 :** 一部のプログラム要素には、大文字と小文字の両方が使用されます。この場合は、記載されているとおりに入力してください。	SELECT last_name, employee_id FROM employees; sqlplus hr/hr CREATE USER mjones IDENTIFIED BY ty3MU9;

OCCI の新機能

ここでは、Oracle C++ Call Interface（OCCI）の新機能について説明し、追加情報の参照先を示します。

OCCI リリース 2（9.2）の新機能

このリリースで導入された新機能は次のとおりです。

- **OTT マーカー**（7-59 ページの「ユーザー追加コードの引継ぎ」を参照）

  OTT マーカーは、ユーザーが追加したコードを引き継ぐことによって、OTT で生成されたコードの機能を拡張します。

  - 「OTT マーカーのプロパティ」（7-59 ページ）
  - 「マーカーを使用できる箇所」（7-61 ページ）
  - 「OTT マーカーの使用方法を示すコード例」（7-64 ページ）
- **基本的なオブジェクト操作のフロー**
  - 「オブジェクトの削除」（3-11 ページ）
  - 「オブジェクト・キャッシュのチューニング」（3-12 ページ）
- **ナビゲーショナル・アクセス**
  - 「オブジェクト・キャッシュでのガベージ・コレクション」（3-17 ページ）
  - 「参照のトランザクション一貫性」（3-18 ページ）

このリリースでは、次のメソッドが追加されました。

- **Blob クラス**
  - 「setEmpty()」（8-21 ページ）
- **Clob クラス**
  - 「setCharSetId()」（8-36 ページ）
  - 「setCharSetForm()」（8-36 ページ）
  - 「setEmpty()」（8-37 ページ）
- **Environment クラス**
  - 「getCacheMaxSize()」（8-67 ページ）
  - 「getCacheOptSize()」（8-67 ページ）
  - 「setCacheMaxSize()」（8-68 ページ）
  - 「setCacheOptSize()」（8-68 ページ）

- **Number クラス**
  - 「operator signed char()」（8-118 ページ）
- **Ref クラス**
  - 「isClear()」（8-132 ページ）
  - 「setNull()」（8-136 ページ）
- **ResultSet クラス**
  - 「getDatebaseNCHARParam()」（8-148 ページ）
  - 「setDatebaseNCHARParam()」（8-159 ページ）

このリリースでは、次のメソッドが変更されました。

- **Clob クラス**
  - 「getCharSetId()」（8-31 ページ）
  - 「getStream()」（8-32 ページ）
  - 「read()」（8-35 ページ）
  - 「write()」（8-38 ページ）
  - 「writeChunk()」（8-39 ページ）
- **Connection クラス**
  - 「getClientCharSet()」（8-42 ページ）
  - 「getClientNCHARCharSet()」（8-42 ページ）
- **ResultSet クラス**
  - 「getCharSet()」（8-145 ページ）
- **Statement クラス**
  - 「getCharSet()」（8-175 ページ）
  - 「setCharSet()」（8-190 ページ）

このリリースでは、次の名前変更が行われました。

- **文の動的実行**
  - STREAM_DATA_REMOVE_AVAILABLE の名前が変更されました（2-18 ページの「STREAM_DATA_AVAILABLE」を参照）。

第 I 部

OCCI プログラマーズ・ガイド

第 I 部は、次の章で構成されています。

1

OCCI の概要

この章では、Oracle C++ Call Interface（OCCI）の概要を説明し、OCCI の説明で使用される特別な用語を示します。Oracle 環境で実行する C++ アプリケーションの開発に必要な基本情報が提供されます。

この章は、次の項目で構成されています。

OCCI の概要

Oracle C++ Call Interface（OCCI）は、API の 1 つです。OCCI を使用すると、C++ 言語で作成したアプリケーションから Oracle データベース内のデータにアクセスできます。OCCI によって、C++ のプログラマは、SQL 文の処理やオブジェクト操作などの Oracle データベース操作をすべて実行できます。

OCCI を使用すると、次の内容を実現できます。

- システム・メモリーおよびネットワーク接続を効率的に使用することにより実現する、高パフォーマンスのアプリケーション
- ユーザー数および要求数の増加に対応できるスケーラブルなアプリケーション
- クライアント側による Oracle データベース・オブジェクトへのアクセスなど、Oracle データベース・オブジェクトを使用したアプリケーション開発に対する包括的なサポート
- ユーザー認証およびパスワード管理の簡素化
- n 層認証
- 2 層クライアント / サーバー環境または複数層環境での動的な接続管理およびトランザクション管理に適した、整合性のあるインタフェース
- カプセル化されたインタフェースおよび不透明なインタフェース

OCCI には、標準的なデータベースのアクセス関数と検索関数のライブラリが、実行時に C++ アプリケーションにリンクできるように、動的ランタイム・ライブラリ（OCCI クラス）の形で用意されています。このため、第三世代言語（3GL）プログラム内に SQL や PL/SQL を埋め込む必要はありません。

OCCI の利点

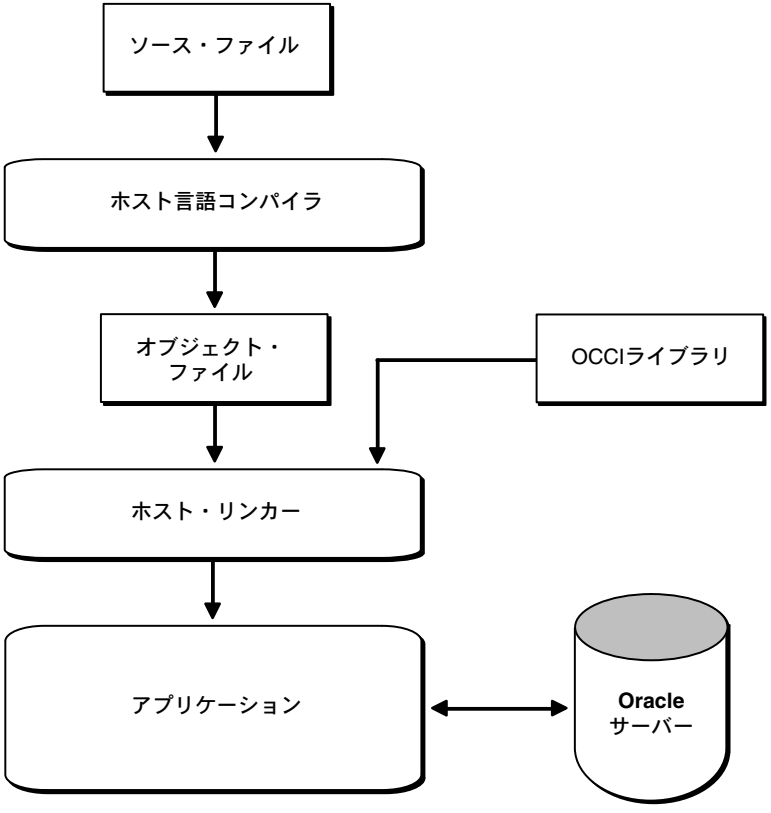
Oracle データベースにアクセスする場合、OCCI には次のような大きな利点があります。

- OCI の機能を使用できます。
- C++ およびオブジェクト指向プログラミング・パラダイムを有効に活用できます。
- 使用しやすくなります。
- JDBC の知識がある場合は容易に習得できます。
- ナビゲーショナル・インタフェースによって、ユーザー定義型のデータベース・オブジェクトを C++ クラス・インスタンスとして操作できます。

OCCI アプリケーションの構築

図 1-1 に示すように、OCCI プログラムは非データベース・アプリケーションと同じ方法でコンパイルおよびリンクできます。

図 1-1 OCCI の開発過程



Oracle では、一般に普及しているほとんどのサード・パーティのコンパイラをサポートしています。OCCI プログラムのリンクの詳細は、システムによって異なります。一部のプラットフォームでは、OCCI プログラムを正常にリンクさせるために、OCCI ライブラリに加えて他のライブラリの組込みが必要になる場合があります。

OCCI の機能

OCCI は、次の機能を備えています。

- 多数のユーザーを安全にサポートできる、スケーラブルな共有サーバー・アプリケーションを設計するための API
- データベースの接続管理、SQL 文の処理および Oracle データベース・サーバーから取り出したオブジェクトの操作を行うための SQL アクセス関数
- Oracle 型のデータ属性を操作するためのデータ型マッピングおよび操作関数

手続き型および非手続き型要素

OCCI を使用すると、SQL の非手続き型データ・アクセス機能と、C++ の手続き型機能を併せ持つ複数層アーキテクチャ上で、スケーラブルな共有サーバー・アプリケーションを開発できます。

非手続き型言語のプログラムでは、操作対象となる一連のデータが指定されますが、実行する操作の種類や操作の方法は指定されません。SQL は非手続き型という性質上、習得しやすく、データベース・トランザクションを実行しやすい言語です。また、SQL は、先進のリレーショナル・データベース・システムおよびオブジェクト・リレーショナル・データベース・システムのデータに対するアクセスや操作に使用する標準言語となっています。

手続き型言語のプログラムでは、ほとんどの文の実行が、前または後続の文、およびループや条件ブランチなど SQL では利用できない制御構造に依存しています。手続き型という性質上、このような言語は SQL よりも複雑ですが、柔軟性に富み、効果的に使用できます。

OCCI プログラムでは、非手続き型言語と手続き型言語の要素が組み合されているため、構造化プログラミング環境で Oracle データベースに容易に接続できます。

OCCI では、Oracle データベース・サーバーを通じて使用できる、SQL のデータ定義、データ操作、問合せおよびトランザクション制御の機能をすべてサポートしています。たとえば、OCCI プログラムでは、Oracle データベースに対する問合せを実行できます。問合せ文では入力（バインド）変数を使用して、データベースにデータを渡すようにプログラムに対して指示できます。たとえば、次のとおりです。

```
SELECT name FROM employees WHERE empno = :empnumber
```

この SQL 文の `:empnumber` は、アプリケーションが指定する値に対するプレースホルダです。

また、OCCI アプリケーションでは、Oracle が開発した SQL の手続き型拡張要素である PL/SQL も使用できます。PL/SQL を使用すると、SQL のみで作成されたアプリケーションよりも、さらに効果的で柔軟性のあるアプリケーションを開発できます。OCCI では、Oracle データベース・サーバー内のオブジェクトに対するアクセスおよび操作のための機能も提供します。

SQL 文の処理

OCCI アプリケーションの主なタスクの 1 つは、SQL 文を処理することです。SQL 文の種類が異なると、プログラムでは異なる処理ステップが必要になります。OCCI アプリケーションをコーディングする際には、このことを考慮に入れることが重要です。Oracle では、次の種類の SQL 文を認識します。

- データ定義言語（DDL）文
- 制御文
  - トランザクション制御文
  - 接続制御文
  - システム制御文
- データ操作言語（DML）文
- 問合せ

データ定義言語文

データ定義言語（DDL）文は、データベース内のスキーマ・オブジェクトを管理します。DDL 文は、新規の表を作成し、古い表を削除し、その他のスキーマ・オブジェクトを設定します。また、スキーマ・オブジェクトに対するアクセスを制御します。

表を作成し、その表へのアクセスを指定する例を次に示します。

```
CREATE TABLE employees (
   name       VARCHAR2(20),
   ssn        VARCHAR2(12),
   empno      NUMBER(6),
   mgr        NUMBER(6),
   salary     NUMBER(6))

GRANT UPDATE, INSERT, DELETE ON employees TO donna
REVOKE UPDATE ON employees FROM jamie
```

DDL 文によって、Oracle データベース内のオブジェクトを操作することもできます。次の例は、オブジェクト表を作成する一連の文です。

```
CREATE TYPE person_t AS OBJECT (
   name     VARCHAR2(30),
   ssn      VARCHAR2(12),
   address  VARCHAR2(50))

CREATE TABLE person_tab OF person_t
```

制御文

OCCI アプリケーションでは、トランザクション制御文、接続制御文およびシステム制御文を DML 文のように処理します。

データ操作言語（DML）文

データ操作言語（DML）文は、データベース表にあるデータを変更します。たとえば、DML 文を使用して次の処理を実行します。

- 表に新しい行を挿入します。
- 既存の行の列値を更新します。
- 表から行を削除します。
- データベース内の表をロックします。
- SQL 文の実行計画を説明します。

DML 文では、入力（バインド）変数を使用して、データベースにデータを渡すようにアプリケーションに対して指示できます。次の文で説明します。

```
INSERT INTO dept_tab VALUES(:1,:2,:3)
```

異なるバインド変数を使用して、この文を複数回実行することも、配列を挿入して、サーバーとの 1 回のラウンドトリップで複数の行を挿入することもできます。

また、DML 文では Oracle データベース内のオブジェクトを操作できます。次の例では、person_t 型のインスタンスがオブジェクト表 person_tab に挿入されます。

```
INSERT INTO person_tab
   VALUES (person_t('Steve May','123-45-6789','146 Winfield Street'))
```

問合せ

問合せは、データベースの表からデータを取り出すための文です。1 つの問合せで、0、1 行および複数行のデータを戻すことができます。すべての問合せは、次の例のように、SQL キーワード SELECT で始まります。

```
SELECT dname FROM dept
   WHERE deptno = 42
```

問合せ文では入力（バインド）変数を使用して、データベース・サーバーにデータを渡すようにプログラムに対して指示できます。たとえば、次のとおりです。

```
SELECT name
   FROM employees
   WHERE empno = :empnumber
```

この SQL 文の :empnumber は、アプリケーションが指定する値に対するプレースホルダです。

PL/SQL の概要

PL/SQL は、Oracle が開発した SQL 言語の手続き型拡張要素です。PL/SQL は、単純な問合せや SQL データ操作言語文よりも複雑なタスクを処理します。PL/SQL を使用すると、多数の構文を単一のブロックにグループ化し、1 つの単位として実行できます。たとえば、次のような構文があります。

- 1 つまたは複数の SQL 文
- 変数宣言
- 代入文
- 手続き型制御文（IF ...THEN ...ELSE 文とループ）
- 例外処理

OCCI プログラムから PL/SQL ストアド・プロシージャをコールし、OCCI プログラムの PL/SQL ブロックを使用すると、次のタスクを実行できます。

- 他の PL/SQL ストアド・プロシージャおよびストアド・ファンクションのコール
- 手続き型制御文を複数の SQL 文と結合し、1 つの単位として実行
- レコード、表、カーソル FOR ループ、例外処理などの特殊な PL/SQL 機能へのアクセス
- カーソル変数の使用
- Oracle データベース内のオブジェクトへのアクセスおよび操作

PL/SQL プロシージャまたはファンクションは、出力変数を戻すこともできます。この出力変数は **OUT バインド変数**と呼ばれます。次に例を示します。

```
BEGIN
   GET_EMPLOYEE_NAME(:1, :2);
END;
```

この例の最初のパラメータは、従業員の ID 番号を指定する入力変数です。2 番目のパラメータは OUT バインド変数で、従業員名の戻り値が格納されます。

次の PL/SQL 例では、特定の従業員番号をキーにして、従業員表から値を取り出す SQL 文が発行されます。この例は、PL/SQL 文でのプレースホルダの使用方法も示しています。

```
SELECT ename, sal, comm INTO :emp_name, :salary, :commission
   FROM emp
   WHERE ename = :emp_number;
```

この文のプレースホルダは、PL/SQL 変数ではありません。これらは、文の処理時にデータベース・サーバーとの間で受渡しを行う入力パラメータと出力パラメータを示しています。これらのプレースホルダはプログラムで指定する必要があります。

OCCI/SQL の特殊用語

このマニュアルでは、SQL 文の様々な部分を参照するために特殊な用語を使用しています。次の SQL 文の例で説明します。

```
SELECT customer, address
   FROM customers
   WHERE bus_type = 'SOFTWARE'
   AND sales_volume = :sales
```

この SQL 文は、次の部分で構成されています。

- SQL コマンド : `SELECT`
- 2 つの選択リスト項目 : `customer` および `address`
- `FROM` 句の表名 : `customers`
- `WHERE` 句の 2 つの列名 : `bus_type` および `sales_volume`
- `WHERE` 句のリテラル入力値 : `'SOFTWARE'`
- `WHERE` 句の入力（バインド）変数用のプレースホルダ : `:sales`

OCCI アプリケーションを開発する際は、プログラム中の入出力変数の値、またはこれを参照する値をデータベース・サーバーに指定しているルーチンをコールします。このマニュアルでは、データ用のプレースホルダ変数を指定することを**バインド操作**と呼びます。入力変数を指定する場合は、**`IN` バインド操作**と呼びます。出力変数を指定する場合は、**`OUT` バインド操作**と呼びます。

オブジェクトのサポート

OCCI には、オブジェクト型とオブジェクトを処理する機能があります。**オブジェクト型**は、ユーザー定義のデータ構造体で、実社会のエンティティを抽象的に表します。たとえば、データベースに person（個人）というオブジェクトの定義が含まれているとします。このオブジェクト型は、個人を識別するための特徴を表した属性 first_name、last_name および age を持つことができます。

オブジェクト型の定義は、オブジェクト型のインスタンスを表す**オブジェクト**の作成の基礎となります。オブジェクト型を構造的な定義として使用すると、たとえば、person オブジェクトを属性 John、Bonivento および 30 を持つように作成できます。オブジェクト型にはメソッド、つまりそのオブジェクト型の振舞いを表すプログラム関数を含めることもできます。

OCCI は、Oracle データベース・サーバーのオブジェクト機能を使用するプログラマに、包括的な API を提供します。その機能は、次の 5 つの主要なカテゴリに分類できます。

- クライアント側オブジェクト・キャッシュ
- オブジェクト用のランタイム環境
- オブジェクトへのアクセスおよび操作のための連想アクセス用インタフェースおよびナビゲーショナル・アクセス用インタフェース
- オブジェクト型メタデータを記述するためのメタデータ・クラス
- Oracle 内部スキーマ情報をクライアント側の言語バインド変数にマッピングする、Object Type Translator（OTT）ユーティリティ

関連項目： オブジェクト型およびオブジェクトの詳細は、次のマニュアルを参照してください。

- 『Oracle9*i* データベース概要』
- 『Oracle9*i* アプリケーション開発者ガイド - オブジェクト・リレーショナル機能』

クライアント側オブジェクト・キャッシュ

オブジェクト・キャッシュは、オブジェクトに対する参照とメモリー管理をサポートするクライアント側のメモリー・バッファです。オブジェクト・キャッシュでは、OCCI アプリケーションがサーバーからクライアント側にフェッチしたオブジェクトを格納し、追跡します。クライアント側オブジェクト・キャッシュでは、OCCI 環境を object モードで初期化するときに作成されます。同じサーバーに対して実行中の複数のアプリケーションには、それぞれ専用のオブジェクト・キャッシュがあります。クライアント側オブジェクト・キャッシュでは、現在メモリーにあるオブジェクトの追跡、オブジェクトへの参照のメンテナンス、自動オブジェクト・スワッピングの管理およびオブジェクトのメタ属性または型情報の追跡を行います。クライアント側オブジェクト・キャッシュには、次の利点があります。

- アプリケーションのパフォーマンスの改善。オブジェクトのフェッチおよび操作に必要なクライアント / サーバー間のラウンドトリップ回数を減らします。
- 拡張性の向上。クライアント側キャッシュからオブジェクト・スワッピングをサポートします。
- 並行性の向上。オブジェクト・レベルのロックをサポートします。

オブジェクト用のランタイム環境

OCCI は、オブジェクト用のランタイム環境を提供します。このランタイム環境では、Oracle オブジェクトのクライアント側での使用方法を管理する一連のメソッドを使用できます。これらのメソッドは、次のタスクの実行に必要な機能を提供します。

- オブジェクト機能にアクセスするために Oracle データベース・サーバーに接続する
- クライアント側オブジェクト・キャッシュを割当て、そのパラメータをチューニングする
- エラーおよび警告メッセージを取得する
- データベース内のオブジェクトにアクセスするトランザクションを制御する
- SQL によるオブジェクトへの連想アクセス
- パラメータまたは結果が Oracle オブジェクト型システムの型である PL/SQL プロシージャまたはファンクションの記述

連想アクセスおよびナビゲーショナル・アクセス用インタフェース

OCCI を使用したアプリケーションでは、次のような様々な種類のインタフェースを通じて、データベース内のオブジェクトにアクセスできます。

- SQL の `SELECT` 文、`INSERT` 文および `UPDATE` 文。
- C++ ポインタと参照。対応する参照をたどってクライアント側のオブジェクト・キャッシュ内のオブジェクトにアクセスします。

OCCI には、SQL の `SELECT` 文、`INSERT` 文および `UPDATE` 文を使用してオブジェクトを操作するための一連のメソッドが用意されています。Oracle オブジェクトにアクセスする場合、これらの SQL 文では、リレーショナル表にアクセスするような一貫した一連のステップが使用されます。OCCI が提供するメソッドでは、次の目的のために SQL 文を使用してオブジェクトにアクセスします。

- SQL 文や PL/SQL ストアド・プロシージャの入出力変数として、オブジェクト型のインスタンスと参照をバインドします。
- オブジェクト型のインスタンスと参照を含む SQL 文を実行します。
- オブジェクト型のインスタンスと参照をフェッチします。

- 結果セットから列の値をオブジェクトとして取得します。
- Oracle オブジェクト型の選択リスト項目を記述します。

OCCI にはオブジェクトをナビゲートするためのシームレスなインタフェースが用意されているため、一時 C++ オブジェクトを操作する場合と同様に、データベース・オブジェクトを操作できます。オブジェクト参照上のオーバーロードされた矢印（->）演算子を間接参照すると、オブジェクトをデータベースから取り出して、アプリケーション領域で透過的に使用することができます。

メタデータ・クラス

OCCI では、各 Oracle データ型が C++ クラスで表現されます。このクラスは、データ型の振舞いと特性をオーバーロードされた演算子とメソッドによって公開します。たとえば、Oracle データ型 `NUMBER` は、`Number` クラスで表現されます。

OCCI は、メタデータ・クラスを提供します。このメタデータ・クラスを使用すると、オブジェクト型を含むデータベース・オブジェクトを記述するメタデータを取得できます。

Object Type Translator ユーティリティ

Object Type Translator（OTT）ユーティリティは、Oracle オブジェクト型のスキーマ情報をクライアント側の言語バインドに変換します。つまり、OTT はオブジェクト型情報を構造体やクラスなどのホスト言語変数の宣言に変換します。OTT は、Oracle データベース・スキーマ・オブジェクトの情報が含まれる intype ファイルを入力として使用します。次に outtype ファイルおよび必要なヘッダー・ファイルと実装ファイルを生成します。このファイルは、オブジェクト・スキーマに対して実行する C++ アプリケーションにインクルードする必要があります。OTT には、次のような多くの利点があります。

- アプリケーション開発者の生産性が向上します。OTT を使用すると、アプリケーション開発者はスキーマ・オブジェクトに対応するホスト言語変数を手作業で記述する必要がなくなります。
- 選択したデータ定義言語として、SQL をメンテナンスします。OTT では、SQL を使用して作成した Oracle データベース・スキーマ・オブジェクトをホスト言語変数に自動的にマップできるため、SQL がデータ定義言語として使用しやすくなります。その結果、Oracle によって、ユーザー・データの一貫したモデルを企業全体でサポートできます。
- オブジェクト型のスキーマ展開を容易にします。OTT では、スキーマが変更されたときにインクルードされるヘッダー・ファイルを再生成できるため、Oracle アプリケーションでスキーマ展開をサポートできます。

通常、OTT は、コマンドラインから intype ファイル、outtype ファイルおよび特定のデータベース接続を指定することにより起動されます。

要約すると、OCCI では、Oracle データベース内のオブジェクトを処理する次のメソッドをサポートしています。

- オブジェクト・データとスキーマ情報を操作する SQL 文の実行
- SQL 文の入力変数としての、オブジェクト参照およびオブジェクト・インスタンスの受渡し
- SQL 文の出力を受け取る変数としての、オブジェクト参照およびオブジェクト・インスタンスの宣言
- データベースからのオブジェクト参照およびオブジェクト・インスタンスのフェッチ
- オブジェクトのインスタンスおよび参照を戻す SQL 文のプロパティの記述
- オブジェクトのパラメータまたは戻り値を指定した PL/SQL プロシージャまたはファンクションの記述
- オブジェクト機能およびリレーショナル機能を同期化するための、コミット・コールとロールバック・コールの拡張

2

リレーショナル・プログラミング

この章では、標準的なリレーショナル・データベースに格納されたデータを操作するための、Oracle C++ Call Interface（OCCI）を使用した C++ アプリケーション開発の基本を説明します。

この章は、次の項目で構成されています。

データベースへの接続

アプリケーションをデータベースに接続する方法には、多数のオプションがあります。次の各項では、そのオプションについて説明します。

環境の作成と終了

OCCI の処理はすべて、Environment クラスのコンテキスト内で行われます。OCCI 環境では、アプリケーション・モードおよびユーザー指定のメモリー管理関数が提供されます。次のコード例は、OCCI 環境の作成方法を示しています。

```
Environment *env = Environment::createEnvironment();
```

createxxx メソッドで作成した OCCI オブジェクト（接続、接続プール、文）はすべて、明示的に終了する必要があります。必要に応じて、OCCI 環境も明示的に終了する必要があります。次のコード例は、OCCI 環境の終了方法を示しています。

```
Environment::terminateEnvironment(env);
```

また、OCCI 環境の有効範囲は、その環境のコンテキスト内で作成するオブジェクトの有効範囲より大きく設定されている必要があります。次のコード例でこの概念を説明します。

```
const string userName = "SCOTT";
const string password = "TIGER";
const string connectString = "";

Environment *env = Environment::createEnvironment();
{
   Connection *conn = env->createConnection(userName, password, connectString);
   Statement *stmt = conn->createStatement("SELECT blobcol FROM mytable");
   ResultSet *rs = stmt->executeQuery();
   rs->next();
   Blob b = rs->getBlob(1);
   cout << "Length of BLOB : " << b.length();
   .
   .
   .
```

```
   stmt->closeResultSet(rs);
   conn->terminateStatement(stmt);
   env->terminateConnection(conn);
}
Environment::terminateEnvironment(env);
```

createEnvironment メソッドのモード・パラメータを使用すると、アプリケーションのモードを次のように指定できます。

- スレッド環境での実行（THREADED_MUTEXED または THREADED_UNMUTEXED）
- オブジェクトの使用（OBJECT）
- 共有データ構造の利用（SHARED）

これらのモードは、環境ごとに設定できます。

接続のオープンとクローズ

Environment クラスは、Connection オブジェクトを作成するためのファクトリ・クラスです。最初に Environment インスタンスを作成し、そのインスタンスを使用して、ユーザーが createConnection メソッドでデータベースに接続できるようにします。

次のコード例では、環境インスタンスを作成し、そのインスタンスを使用して、データベース・ユーザー scott とパスワード tiger のデータベース接続を作成しています。

```
Environment *env = Environment::createEnvironment();
Connection *conn = env->createConnection("scott", "tiger");
```

次のコード例で示すように、操作セッションの終了時に、terminateConnection メソッドを使用して、接続を明示的にクローズする必要があります。また、OCCI 環境も明示的に終了する必要があります。

```
env->terminateConnection(conn);
Environment::terminateEnvironment(env);
```

接続プールの作成

通常、共有サーバーや中間層アプリケーションでは、比較的継続時間が短い多数のスレッドがデータベースに接続できる必要があります。すべてのスレッドに対してデータベースへの接続をオープンすると、接続の使用率およびパフォーマンスが低下する場合があります。

接続プーリング機能を使用すると、アプリケーションでデータベース管理システム（DBMS）を使用して接続を管理できます。Oracle では、少数のオープン接続を作成し、未接続の 1 つを動的に選択して文を実行し、実行直後にその接続を解放します。この機能によって、アプリケーションで接続を処理し、パフォーマンスを最適化するための複雑なメカニズムを作成する必要がなくなります。

接続プールの作成

接続プールを作成するには、次のように createConnectionPool メソッドを使用します。

```
virtual ConnectionPool* createConnectionPool(
   const string &poolUserName,
   const string &poolPassword,
   const string &connectString ="",
   unsigned int minConn =0,
   unsigned int maxConn =1,
   unsigned int incrConn =1) = 0;
```

前述のメソッド例では、次のパラメータが使用されています。

- poolUserName: 接続プールの所有者。
- poolPassword: 接続プールにアクセスするためのパスワード。
- connectString ="": 接続プールが関連付けられたデータベース・サーバーを指定するデータベース名。
- minConn: 接続プールの作成時にオープンする最小接続数。
- maxConn: 接続プールで維持できる最大接続数。接続プールで最大接続数がオープンされていて、すべての接続がビジー状態の場合、接続を要求した OCCI メソッドのコールは、setErrorOnBusy() が接続プールに対してコールされていないかぎり、接続を取得するまで待機します。
- incrConn: すべての接続がビジー状態のときに、コールによって接続が要求された場合にオープンする追加の接続数。この増分は、オープン接続数の合計が、その接続プールでオープンできる最大接続数より少ない場合にのみ行われます。

次のコード例は、接続プールの作成方法を示しています。

```
const string poolUserName = "SCOTT";
const string poolPassword = "TIGER";
const string connectString = "";
const string username = "SCOTT";
const string password = "TIGER";
unsigned int maxConn = 5;
unsigned int minConn = 3;
unsigned int incrConn = 2;

ConnectionPool *connPool = env->createConnectionPool(poolUserName, poolPassword,
   connectString, minConn, maxConn, incrConn);
```

関連項目：

- OCCI の接続プール・インタフェースの使用方法の実例は、付録 A「OCCI デモ・プログラム」および「occipool.cpp」のコード例を参照してください。

これらの属性をすべて動的に構成することもできます。これによって、現行のロード（オープン接続数およびビジー接続数）を読み込み、これらの属性を適切にチューニングできる柔軟なアプリケーションを設計できます。また、setTimeOut メソッドを使用すると、指定した時間を超過してアイドル状態になっている接続をタイムアウトすることもできます。DBMS では、最適なオープン接続数を維持するために、アイドル状態の接続を定期的に終了します。

各接続プールには、データ構造（プール・ハンドル）が関連付けられています。このプール・ハンドルにはプール・パラメータが格納されます。1 つの環境に作成できる接続プールは 1 つに制限されているわけではありません。1 つの OCCI 環境に複数の接続プールを作成し、同一または異なるデータベースに接続できます。これは、ロード・バランシングが必要なアプリケーションの場合に役立ちます。ただし、プール・ハンドルにはメモリーが必要であるため、複数の接続プールを作成するとメモリー使用量が多くなります。

プロキシ接続

接続プールのユーザーに、他の接続のプロキシとして機能することを許可すると、接続プールの接続の 1 つを使用して他の接続のプロキシとして機能するデータベース・ユーザーのログイン・パスワードは不要になります。

プロキシ接続は、次のメソッドのいずれかを使用して作成できます。

```
ConnectionPool->createProxyConnection(const string &name,
   Connection::ProxyType proxyType =
   Connection::PROXY_DEFAULT);
```

または

```
ConnectionPool->createProxyConnection(const string &name,
   string roles[], int numRoles,
   Connection::ProxyType proxyType =
   Connection::PROXY_DEFAULT);
```

前述のメソッド例では、次のパラメータが使用されています。

- roles[]: このロールの配列は、プロキシ接続がクライアントに対してアクティブになった後でアクティブにするロールのリストを指定します。
- Connection::ProxyType proxyType = Connection::PROXY_DEFAULT: 列挙 Connection::ProxyType には、プロキシ認証を実行する様々な方法を表す定数がリストされます。PROXY_DEFAULT は、name がデータベース・ユーザー名を表し、現在サポートされている唯一のプロキシ認証モードであることを示すために使用します。

SQL の DDL 文と DML 文の実行

SQL は、リレーショナル・データベースを操作するための業界標準の言語です。OCCI では、Statement クラスを使用して SQL コマンドを実行します。

文ハンドルの作成

Statement ハンドルを作成するには、次の例に示すように、Connection オブジェクトの createStatement メソッドをコールします。

```
Statement *stmt = conn->createStatement();
```

SQL コマンド実行のための文ハンドルの作成

Statement ハンドルの作成後、Statement の execute、executeUpdate、executeArrayUpdate または executeQuery メソッドをコールして、SQL コマンドを実行します。これらのメソッドは、次の目的に使用されます。

- execute: 不特定な種類の文の実行
- executeUpdate: DML 文および DDL 文の実行
- executeQuery: 問合せの実行
- executeArrayUpdate: 複数 DML 文の実行

データベース表の作成

次のコード例は、executeUpdate メソッドを使用したデータベース表の作成方法を示しています。

```
stmt->executeUpdate("CREATE TABLE basket_tab
   (fruit VARCHAR2(30), quantity NUMBER)");
```

データベース表への値の挿入

同様に、executeUpdate メソッドを呼び出して、次のように SQL の INSERT 文を実行できます。

```
stmt->executeUpdate("INSERT INTO basket_tab
   VALUES('MANGOES', 3)");
```

executeUpdate メソッドは、SQL 文によって影響を受けた行数を戻します。

関連項目：

- OCCI による表の行に対する挿入、選択、更新および削除操作の実行方法の実例は、付録 A「OCCI デモ・プログラム」および「occidml.cpp」のコード例を参照してください。

文ハンドルの再利用

文ハンドルを再利用すると、SQL 文を複数回実行できます。たとえば、複数のパラメータで同じ文を繰り返し実行するには、次のように、Statement ハンドルの setSQL メソッドを使用して文を指定します。

```
stmt->setSQL("INSERT INTO basket_tab VALUES(:1,:2)");
```

この指定によって、同じ INSERT 文を必要な回数だけ実行できます。後で別の SQL 文を実行する場合は、文ハンドルを再設定します。たとえば、次のとおりです。

```
stmt->setSQL("SELECT * FROM basket_tab WHERE quantity >= :1");
```

これにより、OCCI の文ハンドルとそれに関連付けられたリソースが、不必要に割り当てられたり解放されることがなくなります。現行の文ハンドルの内容は、getSQL メソッドを使用して、いつでも取り出すことができます。

SHARED モード

SQL 文が処理されると、基礎となる特定のデータが文に関連付けられます。このデータには、問合せの結果セットおよび記述情報とともに、文のテキスト・データおよびバインド・データに関する情報が含まれています。このデータは、文を何度実行しても、別のユーザーが実行しても変わりません。

OCCI 環境を SHARED モードで初期化すると、複数の文ハンドル間で共通の文データが共有されるため、アプリケーションのメモリーを節約できます。このメモリーの節約は、複数の文ハンドルを作成するアプリケーションに特に有効です。このようなアプリケーションでは、同一または複数の接続で、同じ SQL 文が複数のユーザーのセッションで実行されるためです。

複数の文ハンドル間で共通のメタデータを共有するには、Environment を SHARED モードで作成します。

文ハンドルの終了

次のように、Statement を明示的に終了し、割当てを解除する必要があります。

```
Connection::conn->terminateStatement(Statement *stmt);
```

OCCI 環境で使用する SQL 文の種類

OCCI 環境では、次の 3 種類の SQL 文を使用します。

- 標準的な文では、SQL コマンドに値を指定します。
- パラメータ化された文には、パラメータまたは変数を指定します。
- コール可能文では、PL/SQL ストアド・プロシージャをコールします。

Statement メソッドは、すべての文、パラメータ化された文およびコール可能文の各文に適用可能なメソッドに分類されます。標準的な文はパラメータ化された文のスーパーセットで、パラメータ化された文はコール可能文のスーパーセットです。

標準的な文

前の項では、DDL コマンドおよび DML コマンドの例を記述しました。たとえば、次のとおりです。

```
stmt->executeUpdate("CREATE TABLE basket_tab
   (fruit VARCHAR2(30), quantity NUMBER)");
```

および

```
stmt->executeUpdate("INSERT INTO basket_tab
   VALUES('MANGOES', 3)");
```

これらはいずれも**標準的な文**の例です。標準的な文では、文の値を明示的に定義します。したがって、これらの例では、CREATE TABLE 文では表の名前（basket_tab）を指定し、INSERT 文では挿入する値（'MANGOES', 3）を規定しています。

パラメータ化された文

文の入力変数にプレースホルダを設定すると、複数のパラメータを使用して同じ文を実行できます。このような文は、ユーザーまたはプログラムからパラメータによる入力を受け取ることができるため、パラメータ化された文と呼ばれます。

たとえば、複数のパラメータを使用して、INSERT 文を実行するとします。最初に、Statement ハンドルの setSQL メソッドを使用して、次のように文を指定します。

```
stmt->setSQL("INSERT INTO basket_tab VALUES(:1, :2)");
```

次に、パラメータを指定するために、setxxx をコールします。*xxx* は、パラメータの型を表します。次の例では、setString メソッドと setInt メソッドを呼び出して、これらの型の値を第 1 および第 2 パラメータに入力しています。

行を挿入するには、次のようにします。

```
stmt->setString(1, "Bananas");     // value for first parameter
stmt->setInt(2, 5);                // value for second parameter
```

パラメータの指定後、行に値を挿入します。

```
stmt->executeUpdate();             // execute statement
```

別の行を挿入するには、次のようにします。

```
stmt->setString(1, "Apples");      // value for first parameter
stmt->setInt(2, 9);                // value for second parameter
```

パラメータの指定後、行に値を再度挿入します。

```
stmt->executeUpdate();             // execute statement
```

アプリケーションで同じ文を繰り返し実行している場合は、入力の型を変更するたびにリバインドが行われるので、入力パラメータの型の変更が回避されます。

コール可能文

PL/SQL ストアド・プロシージャは、その名前が示すように、データベース・サーバーに格納されているプロシージャです。アプリケーションではこのプロシージャを再利用できます。OCCI では、他の SQL 文を含むプロシージャのコールは**コール可能文**と呼ばれます。

たとえば、指定した種類のフルーツの数を戻すプロシージャ（countFruit）をコールするとします。PL/SQL ストアド・プロシージャの入力パラメータを指定するには、パラメータ化された文の場合と同様に、Statement クラスの setxxx メソッドをコールします。

```
stmt->setSQL("BEGIN countFruit(:1, :2); END:");
int quantity;
stmt->setString(1, "Apples");
// specify the first (IN) parameter of procedure
```

ただし、ストアド・プロシージャをコールする前に、registerOutParam メソッドをコールして、OUT および IN/OUT パラメータの型とサイズを指定しておく必要があります。

```
stmt->registerOutParam(2, Type::OCCIINT, sizeof(quantity));
// specify the type and size of the second (OUT) parameter
```

次に、プロシージャをコールして文を実行します。

```
stmt->executeUpdate();           // call the procedure
```

最後に、適切な getxxx メソッドをコールして、出力パラメータを取得します。

```
quantity = stmt->getInt(2);     // get the value of the second (OUT) parameter
```

配列をパラメータとして指定したコール可能文

コール可能文を通じて実行される PL/SQL ストアド・プロシージャには、値の配列をパラメータとして指定できます。配列内の要素の数とディメンションは、setDataBufferArray メソッドで指定します。

次に、setDataBufferArray メソッドの例を示します。

```
void setDataBufferArray(int paramIndex,
   void *buffer,
   Type type,
   ub4 arraySize,
   ub4 *arrayLength,
   sb4 elementSize,
   sb2 *ind = NULL,
   ub2 *rc = NULL);
```

前述のメソッド例では、次のパラメータが使用されています。

- paramIndex: パラメータ番号
- buffer: 値の配列を格納するデータ・バッファ
- Type: データ・バッファ内のデータの型
- arraySize: 配列の最大要素数
- arrayLength: 配列の要素数
- elementSize: 配列の現行要素のサイズ
- ind: インジケータ情報
- rc: リターン・コード

関連項目：

- バインド・パラメータを指定した PL/SQL プロシージャの呼出し方法の実例は、付録 A「OCCI デモ・プログラム」および「occiproc.cpp」のコード例を参照してください。

ストリーム読込み / 書込み

ストリーム・データには、次の 3 種類があります。

- IN バインド変数に対応する書込み可能ストリーム
- OUT バインド変数に対応する読込み可能ストリーム
- IN/OUT バインド変数に対応する双方向ストリーム

OCCI では、IN、OUT および IN/OUT の 3 種類のストリーム・パラメータをパラメータ化された文とコール可能文に使用できます。

行変更の反復

executeUpdate メソッドは各行に対して繰り返し発行できますが、OCCI には、1 回のネットワーク・ラウンドトリップで複数行に対してデータを送信する効率的なメカニズムがあります。これを行うには、Statement クラスの addIteration メソッドを使用して、反復ごとに異なる行を変更するバッチ操作を実行します。

INSERT、UPDATE および DELETE の各操作を反復して実行するには、次の設定を行う必要があります。

- 最大反復回数の設定
- 可変長パラメータに対する最大パラメータ・サイズの設定

最大反復回数の設定

反復実行を行う場合は、最初に setMaxIterations メソッドをコールして、文に対して実行する最大反復回数を指定します。

```
Statement->setMaxIterations(int maxIterations)
```

現行の最大反復回数の設定を取得するには、getMaxIterations メソッドをコールします。

最大パラメータ・サイズの設定

反復実行に string や Bytes などの可変長のデータ型が含まれている場合は、OCCI で最大バッファ・サイズの割当てができるように、次のように最大パラメータ・サイズを設定する必要があります。

```
Statement->setMaxParamSize(int parameterIndex, int maxParamSize)
```

Number や Date、あるいは setDataBuffer メソッドを使用するパラメータなど、固定長のデータ型については最大パラメータ・サイズを設定する必要はありません。

現行の最大パラメータ・サイズの設定を取得するには、getMaxParamSize メソッドをコールします。

反復操作の実行

最大反復回数および（必要に応じて）最大パラメータ・サイズの設定後は、次のように、パラメータ化された文を使用した反復実行を簡単に行うことができます。

```
stmt->setSQL("INSERT INTO basket_tab VALUES(:1, :2)");

stmt->setString(1, "Apples");      // value for first parameter of first row
stmt->setInt(2, 6);                // value for second parameter of first row
stmt->addIteration();              // add the iteration
```

```
stmt->setString(1, "Oranges");     // value for first parameter of second row
stmt->setInt(1, 4);                // value for second parameter of second row

stmt->executeUpdate();             // execute statement
```

例に示したように、最後の反復を除く各反復の後に addIteration メソッドをコールし、その後 executeUpdate メソッドを呼び出します。2 番目の行を挿入しない場合は、addIteration メソッドや後続の setxxx メソッドのコールは必要ありません。

反復実行の使用上の注意

- 反復実行は、標準的な文またはパラメータ化された文のいずれかによる INSERT、UPDATE および DELETE 操作にのみ使用できるように設計されています。コール可能文および問合せには使用できません。
- 反復間でのデータ型の変更はできません。たとえば、パラメータ 1 に setInt を使用した場合、後続の反復の同じパラメータには setString を使用できません。

SQL 問合せの実行

SQL 問合せを使用すると、アプリケーションは指定した制約に基づいて、データベースから情報を要求できます。問合せ結果として、結果セットが戻されます。

結果セット

データベース問合せを実行すると、問合せ結果は、結果セットと呼ばれる行セットに格納されます。OCCI では、SQL の SELECT 文は、Statement クラスの executeQuery メソッドによって実行されます。このメソッドは、問合せ結果を表す ResultSet オブジェクトを戻します。

```
ResultSet *rs = stmt->executeQuery("SELECT * FROM basket_tab");
```

結果セットのデータを取得した後は、そのデータに対して操作を実行できます。たとえば、この表の内容を出力するとします。次のコード例に示すように、ResultSet の next メソッドを使用してデータをフェッチし、getxxx メソッドを使用して結果セットの個々の列を取り出します。

```
cout << "The basket has:" << endl;

while (rs->next())
{
   string fruit = rs->getString(1);     // get the first column as string
   int quantity = rs->getInt(2);        // get the second column as int

   cout << quantity << " " << fruit << endl;
}
```

ResultSet クラスの next メソッドと status メソッドは、Status 列挙型を戻します。Status の可能な値は、次のとおりです。

- DATA_AVAILABLE
- END_OF_FETCH = 0
- STREAM_DATA_AVAILABLE

データが現在行に対して使用可能な場合、ステータスは DATA_AVAILABLE となります。すべてのデータが読み込まれると、ステータスは END_OF_FETCH に変更されます。

読み込む出力ストリームがある場合は、そのストリーム・データがすべて正常に読み込まれるまで、ステータスは STREAM_DATA_AVAILABLE となります。次にコード例を示します。

```
ResultSet *rs = stmt->executeQuery("SELECT * FROM demo_tab");
ResultSet::Status status = rs->status();     // status is DATA_AVAILABLE
while (rs->next())
{
   get data and process;
}
```

結果セット全体の横断が終了すると、ステータスは END_OF_FETCH に変更され、WHILE ループが終了します。

次に、結果セットのストリームの例を示します。

```
char buffer[4096];
ResultSet *rs = stmt->executeQuery
   ("SELECT col2 FROM tab1 WHERE col1 = 11");
ResultSet *rs = stmt->getResultSet ();

while (rs->next ())
{
   unsigned int length = 0;
   unsigned int size = 500;
   Stream *stream = rs->getStream (2);
   while (stream->status () == Stream::READY_FOR_READ)
   {
      length += stream->readBuffer (buffer +length, size);
   }
   cout << "Read "  << length << " bytes into the buffer" << endl;
}
```

問合せの指定

IN バインド変数を問合せで使用すると、問合せの WHERE 句に制約を指定できます。たとえば、次のプログラムは、最小数量 4 を持つ項目のみを出力します。

```
stmt->setSQL("SELECT * FROM basket_tab WHERE quantity >= :1");
int minimumQuantity = 4;
stmt->setInt(1, minimumQuantity);     // set first parameter
ResultSet *rs = stmt->executeQuery();
cout << "The basket has:" << endl;

while (rs->next())
   cout << rs->getInt(2) << " " << rs->getString(1) << endl;
```

プリフェッチ・カウントの設定によるパフォーマンスの最適化

ResultSet メソッドでは一度に 1 行のデータを取り出しますが、サーバーからの実際のデータ・フェッチでは、問合せ対象の行ごとにネットワーク・ラウンドトリップを発生させる必要はありません。パフォーマンスを最大にするために、サーバーに対する各ラウンドトリップでプリフェッチする行数を設定できます。

これを行うには、プリフェッチする行数（setPrefetchRowCount）またはプリフェッチで使用するメモリー・サイズ（setPrefetchMemorySize）を設定します。

この属性の両方を設定すると、指定したメモリー制限に先に到達しないかぎり、指定した行数がプリフェッチされます。指定したメモリー制限に先に到達した場合は、setPrefetchMemorySize メソッドのコールで定義したメモリー領域に適合する行数が戻ります。

デフォルトではプリフェッチはオンになっており、データベースでは常に 1 行余分にフェッチします。プリフェッチをオフにするには、プリフェッチする行数とメモリー・サイズの両方を 0（ゼロ）に設定します。

注意： LONG 列が問合せの一部である場合、プリフェッチは無効になります。LOB 列を含む問合せの場合は、問合せによってデータではなく LOB ロケータが戻されるため、プリフェッチできます。

文の動的実行

DML 操作が必要であるとわかっている場合は、executeUpdate メソッドを使用します。同様に、問合せが必要であるとわかっている場合は executeQuery を使用します。

しかし、アプリケーションで動的イベントを使用する必要がある場合、実行時にどの文の実行が必要であるかを確認できません。このため、OCCI には、execute メソッドが用意されています。execute メソッドを呼び出すと、次のいずれかのステータスが戻ります。

- UNPREPARED
- PREPARED
- RESULT_SET_AVAILABLE
- UPDATE_COUNT_AVAILABLE
- NEEDS_STREAM_DATA
- STREAM_DATA_AVAILABLE

execute メソッドの呼出しではこれらのステータスのいずれかが戻りますが、status メソッドを使用して文を調べることもできます。

```
Statement stmt = conn->createStatement();
Statement::Status status = stmt->status();       // status is UNPREPARED
stmt->setSQL("select * from emp");
status = stmt->status();                         // status is PREPARED
```

SQL 文字列で文ハンドルを作成した場合は、PREPARED 状態で作成されます。たとえば、次のとおりです。

```
Statement stmt = conn->createStatement("insert into foo(id) values(99)");
Statement::Status status = stmt->status();       // status is PREPARED
status = stmt->execute();            // status is UPDATE_COUNT_AVAILABLE
```

別の SQL 文を Statement に設定すると、ステータスが PREPARED に変更されます。たとえば、次のとおりです。

```
stmt->setSQL("select * from emp");               // status is PREPARED
status = stmt->execute();            // status is RESULT_SET_AVAILABLE
```

ステータスの定義

この項では、文ハンドルに関連付けられている Status のとりうる値について説明します。

- UNPREPARED
- PREPARED
- RESULT_SET_AVAILABLE
- UPDATE_COUNT_AVAILABLE
- NEEDS_STREAM_DATA
- STREAM_DATA_AVAILABLE

UNPREPARED

setSQL メソッドで文ハンドルに SQL 文字列の属性を指定していない場合、その文は UNPREPARED 状態になります。

```
Statement stmt = conn->createStatement();
Statement::Status status = stmt->status();   // status is UNPREPARED
```

PREPARED

SQL 文字列で Statement を作成した場合は、PREPARED 状態で作成されます。たとえば、次のとおりです。

```
Statement stmt = conn->createStatement("INSERT INTO demo_tab(id)
   VALUES(99)");
Statement::Status status = stmt->status();    // status is PREPARED
```

別の SQL 文を Statement に設定した場合も、ステータスが PREPARED に変更されます。たとえば、次のとおりです。

```
status = stmt->execute();                    // status is UPDATE_COUNT_AVAILABLE
stmt->setSQL("SELECT * FROM demo_tab");      // status is PREPARED
```

RESULT_SET_AVAILABLE

RESULT_SET_AVAILABLE ステータスは、適切に作成された問合せが実行済みで、結果セットを通じて結果にアクセスできることを示します。

文ハンドルを問合せに設定すると、ステータスは PREPARED になります。問合せを実行すると、その文のステータスは RESULT_SET_AVAILABLE に変更されます。たとえば、次のとおりです。

```
stmt->setSQL("SELECT * from EMP");           // status is PREPARED
status = stmt->execute();                    // status is RESULT_SET_AVAILABLE
```

結果セットのデータにアクセスするには、次の文を発行します。

```
ResultSet *rs = Statement->getResultSet();
```

UPDATE_COUNT_AVAILABLE

PREPARED 状態の DDL 文または DML 文を実行すると、その状態が UPDATE_COUNT_AVAILABLE に変更されます。次にコード例を示します。

```
Statement stmt = conn->createStatement("INSERT INTO demo_tab(id) VALUES(99)");
Statemnt::Status status = stmt->status();    // status is PREPARED
status = stmt->execute();                    // status is UPDATE_COUNT_AVAILABLE
```

このステータスは、文の実行によって影響を受けた行数を示します。また、次のことを示しています。

- その文に入力ストリームまたは出力ストリームが含まれていないこと
- その文が問合せではなく、DDL 文または DML 文のいずれかであること

影響を受けた行数を取得するには、次の文を発行します。

```
Statement->getUpdateCount();
```

DDL 文の結果は、更新カウント 0（ゼロ）になります。同様に、一致条件のいずれも満たさない更新の場合、結果はカウント 0（ゼロ）になります。このような場合は、実行された文の種類をレポートされたステータスから推定することはできません。

NEEDS_STREAM_DATA

書き込まれる出力ストリームがある場合は、すべてのストリーム・データが完全に書き込まれるまで実行は完了しません。このような場合は、ステータスは NEEDS_STREAM_DATA に変更され、ストリームを書き込む必要があることを示します。ストリームの書込み後、status メソッドをコールして、さらに書き込むストリーム・データがあるかどうか、つまり実行が完了したかどうかを調べます。

複数のストリーム・パラメータが文に含まれている場合は、getCurrentStreamParam メソッドを使用して書込みが必要なパラメータを検出します。

反復または配列実行の場合は、getCurrentStreamIteration メソッドによって、データを書き込む対象の反復を識別します。

ストリーム・データがすべて処理されると、ステータスは RESULT_SET_AVAILABLE または UPDATE_COUNT_AVAILABLE に変更されます。

STREAM_DATA_AVAILABLE

このステータスは、実行の完了前に、アプリケーションで一部のストリーム・データを OUT パラメータまたは IN/OUT パラメータに読み込む必要があることを示しています。ストリームの読込み後、status メソッドをコールして、さらに読み込むストリーム・データがあるかどうか、つまり実行が完了したかどうかを調べます。

複数のストリーム・パラメータが文に含まれている場合は、getCurrentStreamParam メソッドを使用して読込みが必要なパラメータを検出します。

反復または配列実行の場合は、getCurrentStreamIteration メソッドによって、データを読み込む対象の反復を識別します。

ストリーム・データがすべて処理されると、ステータスは UPDATE_COUNT_REMOVE_AVAILABLE に変更されます。

ResultSet クラスにも読込み可能なストリームがあり、Statement クラスの読込み可能なストリームと同様に機能します。

トランザクションのコミット

SQL の DML 文はすべて、トランザクションのコンテキスト内で実行されます。DML 文によって行われたアプリケーションへの変更は、トランザクションのコミットによって永続的に変更されるか、ロールバックの実行によって取り消されます。SQL の COMMIT 文および ROLLBACK 文は、executeUpdate メソッドで実行できますが、Connection::commit メソッドおよび Connection::rollback メソッドをコールすることもできます。

DML で行った変更を即時にコミットする場合は、次の文を発行して、Statement クラスの自動コミット・モードをオンにできます。

```
Statement::setAutoCommit(TRUE)
```

自動コミットを有効にした後は、変更を行うたびに自動的に永続的な変更となります。これは、各実行の直後にコミットを発行するのと同じです。

デフォルトのモードに戻るには、次の文を発行して自動コミットをオフにします。

```
Statement::setAutoCommit(FALSE)
```

エラー処理

各 OCCI メソッドでは、そのメソッドが成功したかどうかを示すリターン・コードを戻すことができます。つまり、OCCI メソッドは例外を発生させることができます。この例外は、SQLException 型です。OCCI では C++ の標準テンプレート・ライブラリ（Standard Template Library: STL）を使用するため、STL で発生した例外は OCCI メソッドでも発生します。

STL 例外は、標準の exception クラスから導出されます。exception::what() メソッドは、エラー・テキストへのポインタを戻します。このエラー・テキストは、catch ブロック内では有効であることが保証されます。

SQLException クラスには、Oracle 固有のエラー番号とメッセージが格納されます。このクラスは、標準の exception クラスから導出されるため、exception::what() メソッドを使用してエラー・テキストを取得することもできます。

また、SQLException クラスには、エラー情報を取得できる 2 つのメソッドがあります。getErrorCode メソッドは Oracle エラー番号を戻します。exception::what() が戻すエラー・テキストと同じものを、getMessage メソッドでも取得できます。getMessage メソッドは、STL 文字列を戻すため、他の STL 文字列と同様にコピーできます。

エラー処理の方針に従って、OCCI 例外を標準の例外とは別に処理するか、または両者を区別しないで処理するかを選択できます。

OCCI 例外と標準の例外を区別しない場合、catch ブロックは次のようになります。

```
catch (exception &excp)
{
   cerr << excp.what() << endl;
}
```

OCCI 例外を標準の例外とは別に処理する場合、catch ブロックは次のようになります。

```
catch (SQLException &sqlExcp)
{
   cerr <<sqlExcp.getErrorCode << ": " << sqlExcp.getErrorMessage() << endl;
}
catch (exception &excp)
{
   cerr << excp.what() << endl;
}
```

前述の catch ブロックでは、SQL 例外が最初のブロックで捕捉され、非 SQL 例外が 2 番目のブロックで捕捉されます。この 2 つのブロックの順序を逆にすると、SQL 例外は捕捉されません。SQLException は標準の exception から導出され、標準の exception catch ブロックでは SQL 例外も同様に処理されます。

> **関連項目：** Oracle エラー・メッセージの詳細は、『Oracle9*i* データベース・エラー・メッセージ』を参照してください。

NULL および切捨てデータ

通常、OCCI では、ResultSet クラスまたは Statement クラスの get*xxx* メソッドで取り出したデータ値が NULL または切捨てデータの場合は例外が発生しません。ただし、この振舞いは、setErrorOnNull メソッドまたは setErrorOnTruncate メソッドをコールして変更できます。setError*xxx* メソッドに causeException=TRUE を指定してコールすると、データ値が NULL または切捨てデータの場合に SQLException が呼び出されます。

デフォルトの動作では、SQLException は呼び出されません。この場合、NULL データは、数値の場合は 0（ゼロ）として戻され、文字の場合はヌル文字列として戻されます。

setDataBuffer メソッドおよび setDataBufferArray メソッドを使用して取り出されたデータの場合、例外処理の動作は、表 2-1、表 2-2 および表 2-3 に示したインジケータ変数とリターン・コード変数の有無によって制御されます。

表 2-1 通常データ — 非 NULL または切捨てなし

リターン・コード	インジケータ — 戻さない場合	インジケータ — 戻す場合
戻さない場合	error = 0	error = 0 indicator = 0
戻す場合	error = 0 return code = 0	error = 0 indicator = 0 return code = 0

表 2-2 NULL データ

リターン・コード	インジケータ — 戻さない場合	インジケータ — 戻す場合
戻さない場合	SQLException error = 1405	error = 0 indicator = -1
戻す場合	SQLException error = 1405 return code = 1405	error = 0 indicator = -1 return code = 1405

表 2-3 切捨てデータ

リターン・コード	インジケータ — 戻さない場合	インジケータ — 戻す場合
戻さない場合	SQLException error = 1406	SQLException error = 1406 indicator = data_len
戻す場合	error = 24345 return code = 1405	error = 24345 indicator = data_len return code = 1406

表 2-3 の data_len は、長さが SB2MAXVAL 以下であった場合に切り捨てられたデータの実際の長さです。この値を超えている場合は、インジケータが -2 に設定されます。

高度な関連テクニック

この項では、次の高度なテクニックについて説明します。

- 共有サーバー環境の利用
- パフォーマンスの最適化

共有サーバー環境の利用

スレッド・セーフティ

スレッドは、大規模なプロセス内に存在する軽量プロセスです。複数のスレッドで同一コードおよび同一データ・セグメントを共有しますが、プログラム・カウンタ、マシン・レジスタおよびスタックは、スレッドごとにあります。グローバル変数および静的変数は、すべてのスレッドに共通です。そのため、1 つのアプリケーション内で、複数のスレッドによるこれらの変数へのアクセスを管理する相互排他メカニズムが必要です。

スレッドは、一度作成されると他のスレッドとは非同期的に動作します。このため、順序を気にせずに共通のデータ要素にアクセスし、OCCI コールを実行します。このようにデータ要素に共有アクセスを行うため、複数のスレッドによってアクセスされるデータの整合性を維持するメカニズムが必要です。データ・アクセスを管理するメカニズムでは、mutex（相互排他ロック）の形式を使用し、アプリケーション内の共有リソースにアクセスする複数のスレッド間で、競合が発生しないようにします。OCCI では、mutex は OCCI 環境単位で付与されます。

Oracle データベース・サーバーおよび OCCI ライブラリのスレッド・セーフティ機能により、開発者は OCCI を共有サーバー環境で使用でき、次の利点を得ることもできます。

- 複数のスレッドで OCCI コールを実行した場合も、単一のスレッドで連続するコールを実行した場合と同じ結果になります。
- 複数のスレッドで OCCI コールを実行した場合、スレッド間で副作用がありません。
- 共有サーバー・プログラムを作成しない場合でも、スレッド・セーフの OCCI コールを実行してもパフォーマンスが低下しません。
- 複数のスレッドを使用すると、プログラムのパフォーマンスが向上します。パフォーマンスが向上するのは、別々のプロセッサでスレッドを同時実行するマルチプロセッサ・システム、および低速操作と高速操作の間でオーバーラップが発生するシングル・プロセッサ・システムの場合です。

スレッド・セーフティと 3 層アーキテクチャ

クライアント / サーバー・アプリケーションでは、クライアントを共有サーバー・プログラムにできますが、それに加え、共有サーバー・アプリケーションは 3 層アーキテクチャ（クライアント / エージェント / サーバー・アーキテクチャとも呼ばれる）で典型的に使用されます。このアーキテクチャでは、クライアントが関係するのはプレゼンテーション層のサービスのみです。エージェント（つまり、アプリケーション・サーバー）は、クライアント・アプリケーションのアプリケーション・ロジックを処理します。一般的には、この関係は多対 1 の関係で、複数のクライアントが同一のアプリケーション・サーバーを共有します。

この 3 層アーキテクチャのサーバー層は、Oracle データベース・サーバーです。アプリケーション・サーバー（エージェント）は、共有サーバーのアプリケーション・サーバーにするのが最適で、各スレッドがクライアントの要求を処理します。Oracle 環境では、この中間層アプリケーション・サーバーは OCCI プログラムまたはプリコンパイラ・プログラムです。

スレッド・セーフティの実装

OCCI でスレッド・セーフティを使用するには、最初にアプリケーションをスレッド・セーフなプラットフォーム上で実行する必要があります。次に、アプリケーションは、共有サーバー・モードで稼働していることを、createEnvironment メソッドのモード・パラメータに THREADED_MUTEXED または THREADED_UNMUTEXED を指定することで OCCI に通知する必要があります。たとえば、相互排他ロックをオンにするには、次の文を発行します。

```
Environment *env = Environment::createEnvironment(Environment::THREADED_MUTEXED);
```

createEnvironment に THREADED_MUTEXED または THREADED_UNMUTEXED を指定してコールする場合は、createEnvironment メソッドの後続のコールもすべて THREADED_MUTEXED または THREADED_UNMUTEXED のモードを指定して実行する必要があります。

共有サーバー・アプリケーションがスレッド・セーフなプラットフォーム上で稼働している場合、OCCI ライブラリでは、アプリケーションの mutex が OCCI 環境単位で管理されます。ただし、この機能を上書きし、アプリケーションで独自の mutex スキームを維持することもできます。これを行うには、createEnvironment メソッドに THREADED_UNMUTEXED のモード値を指定します。

注意：

- スレッド・セーフでないプラットフォームで稼働するアプリケーションの場合は、createEnvironment メソッドに THREADED_MUTEXED または THREADED_UNMUTEXED の値を渡さないでください。
- シングル・スレッド・アプリケーションの場合は、プラットフォームがスレッド・セーフであるかどうかにかかわらず、createEnvironment メソッドに Environment::DEFAULT の値を渡す必要があります。これは、モード・パラメータのデフォルト値でもあります。シングル・スレッド・アプリケーションを THREADED_MUTEXED モードで稼働すると、パフォーマンスが低下する可能性があります。

共有サーバーの並行性

アプリケーション・プログラマは、共有サーバー環境での並行性に関して、次の 2 つの基本的なオプションから選択できます。

- 自動シリアル化。OCCI の透過的メカニズムを利用します。
- アプリケーション対応のシリアル化。複数スレッドのメンテナンスに付随して発生する問題をプログラマが管理します。

自動シリアル化 複数スレッドが、OCCI 環境から導出されたオブジェクト（接続および接続プール）を操作している場合は、OCCI でこれらのオブジェクトへのアクセスをシリアル化するように選択できます。最初のステップとして、createEnvironment メソッドに THREADED_MUTEXED の値を渡します。この時点で、OCCI ライブラリは、環境内のスレッド・セーフ・オブジェクトに関する mutex を自動的に取得します。

OCCI 環境が THREADED_MUTEXED モードで作成されている場合は、Environment、Map、ConnectionPool および Connection オブジェクトのみがスレッド・セーフになります。たとえば、2 つのスレッドがこれらのオブジェクトの 1 つに対して同時コールを行うと、OCCI によってそれらのコールが内部的にシリアル化されます。ただし、Statement、ResultSet、SQLException、Stream など、その他の OCCI オブジェクトはスレッド・セーフではないため、アプリケーションでこれらのオブジェクトを複数スレッドから同時に操作することはできません。

OCCI コールの処理のほとんどはサーバー上で行われます。そのため、OCCI コールを使用する 2 つのスレッドの接続先が同じ場合は、サーバーで一方のスレッドの処理が終了するまでもう一方はブロックされる場合があるので注意してください。

アプリケーション対応のシリアル化 複数スレッドが、OCCI 環境から導出されたオブジェクトを操作している場合は、シリアル化を管理するように選択できます。最初のステップとして、createEnvironment モードに THREADED_UNMUTEXED の値を渡します。この場合、アプリケーションでは、同一の OCCI 環境から導出されたオブジェクトに対して相互排他

ロックの OCCI コールを行う必要があります。この方法には、アプリケーションの設計に基づいて mutex スキームを最適化し、並行性を向上できるという利点があります。

OCCI 環境をこのモードで作成した場合、OCCI では、アプリケーションは共有サーバー環境で稼働しているが、その内部 mutex の取得は不要であると認識されます。つまり、OCCI は、OCCI 環境から導出されたオブジェクトのメソッドのコールはすべて、アプリケーションによってシリアル化されるとみなします。次の 2 つの方法を実行できます。

- 各スレッドには、それぞれ専用の環境があります。つまり、専用の環境とその環境から導出されたすべてのオブジェクト（接続、接続プール、文、結果セットなど）は、スレッド間で共有されません。この場合、アプリケーションで mutex を適用する必要はありません。
- OCCI 環境またはその環境から導出されたオブジェクトをスレッド間で共有する場合は、1 つのスレッドのみが任意のオブジェクトに対して OCCI メソッドをコールするように、オブジェクトへのアクセスをシリアル化する必要があります。

いずれの場合も、OCCI によって mutex は取得されません。`THREADED_UNMUTEXED` を使用する場合は、OCCI 環境から導出された任意のオブジェクトに対するプロセスに、1 つの OCCI コールのみが存在していることを常に確認する必要があります。

注意：

- OCCI は、ハンドルを最大限に再利用できるように最適化されています。各環境にそれぞれ専用のヒープがあるため、複数環境の場合は、メモリー使用量の増加につながります。複数環境の存在は、接続、接続プール、文および結果セット・オブジェクトに関する作業が重複していることを意味します。これにより、メモリー使用量がさらに増加します。
- サーバーに対して複数の接続を行うと、サーバーとネットワークのリソース使用量が増加します。複数環境の場合は、さらに多くの接続数を伴うことになります。

パフォーマンスの最適化

パラメータ化された文で setxxx メソッドを使用してバインド・パラメータにデータを提供すると、その値は内部データ・バッファにコピーされた後、データベース・サーバーに渡されて挿入されます。このデータ・コピーに大きい文字列が含まれている場合は、特にコストがかかります。また、それぞれの新しい値について文字列の再割当てが行われ、文字列の割当てと割当て解除が繰り返し行われると、メモリー管理によるオーバーヘッドが発生します。

このため、OCCI には、このパフォーマンスの低下に対処するためのメソッドがいくつか用意されています。次のメソッドがあります。

- setDataBuffer
- executeArrayUpdate
- next（ResultSet クラスの）

setDataBuffer メソッド

高パフォーマンスのアプリケーションの場合は、setDataBuffer メソッドによって、データ・バッファをアプリケーションで管理できます。次に setDataBuffer メソッドの例を示します。

```
void setDataBuffer(int paramIndex,
   void *buffer,
   Type type,
   sb4 size,
   ub2 *length,
   sb2 *ind = NULL,
   ub2 *rc = NULL);
```

前述のメソッド例では、次のパラメータが使用されています。

- paramIndex: パラメータ番号。
- buffer: データを格納するデータ・バッファ。
- type: データ・バッファ内のデータの型。
- size: データ・バッファのサイズ。
- length: データ・バッファ内のデータの現在の長さ。
- ind: インジケータ情報。この情報は、データが NULL かどうかを示します。パラメータ化された文の場合、値 -1 は NULL 値が挿入されることを意味します。コール可能文から戻されたデータの場合、値 -1 は NULL データが取り出されたことを意味します。
- rc: リターン・コード。この変数は、Statement メソッドに提供されたデータには適用できません。ただし、コール可能文から戻されたデータの場合、このリターン・コードにパラメータ固有のエラー番号が指定されます。

setDataBuffer メソッドによって、すべてのデータ型を提供したり取得できるわけではありません。たとえば、C++ 標準ライブラリの文字列は setDataBuffer インタフェースでは提供されません。現在、次のデータ型のみを提供または取得できます。

OCCI_SQLT_CHR	OCCI_SQLT_NUM	OCCIINT
OCCIFLOAT	OCCI_SQLT_STR	OCCI_SQLT_VNU
OCCI_SQLT_PDN	OCCI_SQLT_LNG	OCCI_SQLT_VCS
OCCI_SQLT_NON	OCCI_SQLT_RID	OCCI_SQLT_DAT
OCCI_SQLT_VBI	OCCI_SQLT_BIN	OCCI_SQLT_LBI
OCCIUNSIGNED_INT	OCCI_SQLT_SLS	OCCI_SQLT_LVC
OCCI_SQLT_LVB	OCCI_SQLT_AFC	OCCI_SQLT_AVC
OCCI_SQLT_CUR	OCCI_SQLT_RDD	OCCI_SQLT_LAB
OCCI_SQLT_OSL	OCCI_SQLT_NTY	OCCI_SQLT_REF
OCCI_SQLT_CLOB	OCCI_SQLT_BLOB	OCCI_SQLT_BFILEE
OCCI_SQLT_CFILEE	OCCI_SQLT_RSET	OCCI_SQLT_NCO
OCCI_SQLT_VST	OCCI_SQLT_ODT	OCCI_SQLT_DATE
OCCI_SQLT_TIME	OCCI_SQLT_TIME_TZ	OCCI_SQLT_TIMESTAMP
OCCI_SQLT_TIMESTAMP_TZ	OCCI_SQLT_INTERVAL_YM	OCCI_SQLT_INTERVAL_DS
OCCI_SQLT_TIMESTAMP_LTZ	OCCI_SQLT_FILE	OCCI_SQLT_CFILE
OCCI_SQLT_BFILE		

setxxx メソッドで提供したデータと setDataBuffer メソッドで提供したデータには、重要な違いがあります。setxxx メソッドでデータをコピーした場合は、データのコピー後に元のデータを変更できます。たとえば、setString(str1) メソッドを使用すると、実行前に値 str1 を変更できます。使用される値 str1 は、setString(str1) のコール時の値です。しかし、setDataBuffer メソッドで提供したデータの場合は、実行が完了するまで、バッファをそのまま有効にしておく必要があります。

反復を実行、または executeArrayUpdate メソッドを使用する場合、複数行のデータおよび反復のデータは単一のバッファに格納することが可能です。この場合、*i* 番目の反復に対するデータは buffer + (*i*-1) * size のアドレスにあり、length、インジケータおよびリターン・コードはそれぞれ、*(length + *i*)、*(ind + *i*)、*(rc + *i*) にあります。

また、このインタフェースは、配列実行や、配列または OUT バインド・パラメータを持つコール可能文にも使用できます。

このメソッドを ResultSet クラスで使用すると、各フェッチに対してバッファを再割当てせずにデータを取り出すことができます。

executeArrayUpdate メソッド

すべてのデータを setDataBuffer メソッドまたは出力ストリームで提供する場合（つまり、setDataBuffer または getStream 以外の set*xxx* をコールしない場合）、反復実行を行う簡潔な方法があります。

この場合、setMaxIterations および setMaxParamSize はコールしないでください。かわりに、setDataBuffer（または getStream）メソッドを、各反復にデータを提供する適切なサイズの配列をそれぞれのパラメータに指定してコールし、続いて executeArrayUpdate(int arrayLength) メソッドをコールします。arrayLength パラメータには、各バッファに格納する要素数を指定します。これは、実質的に arrayLength に反復回数を設定して文を実行するのと同じです。

ストリーム・パラメータは一度のみ指定されるため、配列実行でも使用できます。ただし、set*xxx* メソッドを使用する場合は、addIteration メソッドをコールして複数行にデータを提供します。この 2 つの方法を比較するために、2 人の従業員を emp 表に挿入する例を次に示します。

```
Statement *stmt = conn->createStatement("insert into emp (id, ename)
                                         values(:1, :2)");
char enames[2][] = {"SMITH", "MARTIN"};
ub2 enameLen[2];
for (int i = 0; i < 2; i++)
   enameLen[i] = strlen(enames[i] + 1);
stmt->setMaxIteration(2);             // set maximum number of iterations
stmt->setInt(1, 7369);                // specify data for the first row
stmt->setDataBuffer(2, enames, OCCI_SQLT_STR, sizeof(ename[0]), &enameLen);
stmt->addIteration();
stmt->setInt(1, 7654);                // specify data for the second row
// a setDatBuffer is unnecessary for the second bind parameter as data
// provided through setDataBuffer is specified only once.
stmt->executeUpdate();
```

ただし、setDataBuffer インタフェースを使用して最初のパラメータも指定できる場合は、次のように、addIteration メソッドのかわりに executeArrayUpdate メソッドを使用できます。

```
stmt ->setSQL("insert into emp (id, ename) values (:1, :2)");
char enames[2][] = {"SMITH", "MARTIN"};
ub2 enameLen[2];
for (int i = 0; i < 2; i++)
   enameLen[i] = strlen(enames[i] + 1);
int ids[2] = {7369, 7654};
ub2 idLen[2] = {sizeof(ids[0], sizeof(ids[1])};
stmt->setDataBuffer(1, ids, OCCIINT, sizeof(ids[0]), &idLen);
stmt->setDataBuffer(2, enames, OCCI_SQLT_STR, sizeof(ename[0]), &len);
stmt->executeArrayUpdate(2);          // data for two rows is inserted.
```

next メソッドを使用した配列フェッチ

アプリケーションで setDataBuffer インタフェースまたはストリーム・インタフェースのみを使用してデータをフェッチする場合は、配列フェッチを実行できます。配列フェッチは、ResultSet->next(int numRows) メソッドをコールして実装します。これにより、各列について numRows の数のデータがフェッチされます。setDataBuffer インタフェースで指定するバッファ・サイズは、複数行のデータを保持するのに十分なサイズであることが必要です。*i* 番目の行に対するデータは、buffer + (*i* - 1) * size の位置にフェッチされます。同様に、データの長さは、*(length + (*i* - 1)) に格納されます。

```
int empno[5];
char ename[5][11];
ub2  enameLen[5];
ResultSet *resultSet = stmt->executeQuery("select empno, ename  from emp");
resultSet->setDataBuffer(1, &empno, OCCIINT);
resultSet->setDataBuffer(2, ename, OCCI_SQLT_STR, sizeof(ename[0]), enameLen);
rs->next(5);           // fetches five rows, enameLen[i] has length of ename[i]
```

3

オブジェクト・プログラミング

この章では、Oracle C++ Call Interface（OCCI）を使用してオブジェクト・リレーショナル・プログラミングを実装する方法を説明します。

この章は、次の項目で構成されています。

オブジェクト・プログラミングの概要

OCCI では、連想スタイルとナビゲーション・スタイルのデータ・アクセスがサポートされています。通常、第三世代言語（3GL）のプログラムでは、リレーショナル・データベース表で編成された関連付けに基づいた**連想アクセス**を使用して、データベースに格納されたデータを操作します。連想アクセスでは、データは SQL 文や PL/SQL プロシージャの実行によって操作されます。OCCI では、データをクライアントに移送するためのコストをかけずに、アプリケーションがデータベース・サーバーに対して SQL 文や PL/SQL プロシージャを実行できるようにして、オブジェクトへの連想アクセスをサポートしています。

OCCI を使用したオブジェクト指向のプログラムでは、この OCCI プログラミング・パラダイムの重要な側面の 1 つである**ナビゲーショナル・アクセス**も活用できます。アプリケーションでは、複数のオブジェクトを相互に関連したオブジェクトの集合としてモデル化します。これらのオブジェクトは、オブジェクトのグラフを形成し、オブジェクト間の関連は参照（REF）として実装されます。通常、ナビゲーショナル・アクセスを使用するオブジェクト・アプリケーションは、最初にオブジェクトへの REF を戻す SQL 文を発行して、データベース・サーバーから 1 つ以上のオブジェクトを取り出します。次に、アプリケーションはこの REF を使用して関連オブジェクトをたどり、必要に応じて、その他の関連オブジェクト上で計算を実行します。ナビゲーショナル・アクセスでは、初期オブジェクト集合の参照をフェッチする以外、SQL 文の実行は含まれません。OCCI の API をナビゲーショナル・アクセスで使用すると、アプリケーションは Oracle オブジェクトに対して次の操作を実行できます。

- オブジェクトの作成、アクセス、ロック、削除、コピーおよびフラッシュ
- オブジェクトへの参照の取得およびその参照を介したナビゲート

この章では、いくつかの例を示して、永続オブジェクトの作成方法、オブジェクトのアクセス方法と変更方法、および変更内容をデータベース・サーバーにフラッシュする方法を説明します。また、ナビゲーショナル・アクセスおよび連想アクセスの 2 つのアプローチを使用してオブジェクトにアクセスする方法を説明します。

OCCI でのオブジェクトの操作

リレーショナル OCCI アプリケーションに関するプログラミング原理の多くは、オブジェクト・リレーショナル・アプリケーションにも適用されます。オブジェクト・リレーショナル・アプリケーションでは、標準 OCCI コールを使用してデータベース接続を確立し、SQL 文を処理します。その違いは、発行される SQL 文によってオブジェクト参照が取り出され、OCCI のオブジェクト関数でその参照を操作できることです。オブジェクトを（そのオブジェクト参照を使用せずに）値として直接操作することもできます。

Oracle の型のインスタンスは、その存続期間によって、**永続オブジェクト**と**一時オブジェクト**に分類されます。永続オブジェクトのインスタンスは、オブジェクト識別子によって参照されるかどうかに応じて、さらに**スタンドアロン・オブジェクト**と**埋込みオブジェクト**に分割できます。

永続オブジェクト

永続オブジェクトとは、Oracle データベースに格納されているオブジェクトのことです。このオブジェクトは、OCCI アプリケーションによってオブジェクト・キャッシュ内にフェッチし、変更することができます。永続オブジェクトは、それにアクセスしているアプリケーションの存続期間を超えて存続できます。作成された永続オブジェクトは、明示的に削除されるまでデータベースに残り続けます。永続オブジェクトには、次の 2 種類があります。

- **スタンドアロン・インスタンス**はデータベース表の行に格納され、それぞれに一意のオブジェクト識別子があります。OCCI アプリケーションでは、スタンドアロン・オブジェクトへの参照を取り出してオブジェクトを確保し、確保したオブジェクトからその他の関連オブジェクトにナビゲートできます。スタンドアロン・オブジェクトは、**参照可能オブジェクト**とも呼ばれます。

  参照可能オブジェクトの選択も可能で、その場合はオブジェクトを参照でフェッチするかわりに、値でフェッチします。

- **埋込みインスタンス**はデータベース表の行に格納されず、別の構造の中に埋め込まれます。埋込みオブジェクトの例には、別のオブジェクトの属性であるオブジェクトや、データベースの表のオブジェクト列に存在するオブジェクトなどがあります。埋込みオブジェクトにはオブジェクト識別子がないため、OCCI アプリケーションで埋込みインスタンスへの REF を取得することはできません。

  埋込みオブジェクトは、**参照不可能オブジェクト**または**値インスタンス**とも呼ばれます。埋込みオブジェクトは**値**と呼ばれることもありますが、それをスカラー・データ値と混同してはなりません。どちらの意味であるかは文脈で判断できます。

次の SQL の例では、この 2 種類の永続オブジェクトの違いを説明します。

スタンドアロン・オブジェクトの作成 : 例

次のコード例は、スタンドアロン・オブジェクトの作成方法を示しています。

```
CREATE TYPE person_t AS OBJECT
   (name      varchar2(30),
    age       number(3));
CREATE TABLE person_tab OF person_t;
```

オブジェクト表 person_tab に格納されるオブジェクトは、スタンドアロン・オブジェクトです。このオブジェクトにはオブジェクト識別子があるため、参照できます。したがって、OCCI アプリケーションで確保できます。

埋込みオブジェクトの作成 : 例

次のコード例は、埋込みオブジェクトの作成方法を示しています。

```
CREATE TABLE department
   (deptno     number,
    deptname   varchar2(30),
    manager    person_t);
```

department 表の manager 列に格納されるオブジェクトは、埋込みオブジェクトです。このオブジェクトにはオブジェクト識別子がないため、参照できません。つまり、OCCI アプリケーションで確保できないため、確保を解除する必要もありません。このオブジェクトは常に、オブジェクト・キャッシュ内に値で取り込まれます。

一時オブジェクト

一時オブジェクトは、オブジェクト型のインスタンスです。一時オブジェクトは、アプリケーションの存続期間を超えて存続できません。また、一時オブジェクトは常にアプリケーションで削除できます。

Object Type Translator（OTT）ユーティリティは、次のコード例に示すように、各 C++ クラスに対して 2 つの operator new メソッドを生成します。

```
class Person : public PObject {
   .
   .
   .
public:
   dvoid *operator new(size_t size);    // creates transient instance
   dvoid *operator new(size_t size, Connection &conn, string table);
                                        // creates persistent instance
}
```

次のコード例は、一時オブジェクトの作成方法を示しています。

```
Person *p = new Person();
```

一時オブジェクトを永続オブジェクトに変換することはできません。これらの役割は、インスタンス化されたときに決定されます。

関連項目 :

- オブジェクトの詳細は、『Oracle9*i* データベース概要』を参照してください。

値

このマニュアルでは、**値**は次のいずれかを意味します。

- データベースの表の非オブジェクト列に格納されるスカラー値。OCCI アプリケーションでは、SQL 文を発行してデータベースから値をフェッチできます。
- 埋込み（参照不可能）オブジェクト。

どちらを指しているのかは文脈で判断できます。

注意： 参照可能オブジェクトは、確保するかわりに SELECT してオブジェクト・キャッシュに格納することが可能です。その場合は、オブジェクトを参照でフェッチするかわりに値でフェッチします。

C++ アプリケーションでのオブジェクトの表現

OCCI アプリケーションでオブジェクト型を操作するには、データベースにそのオブジェクト型が存在している必要があります。通常は、CREATE TYPE など、SQL の DDL 文で型を作成します。

永続オブジェクトと一時オブジェクトの作成

次の各項では、永続オブジェクトと一時オブジェクトの作成方法について説明します。

永続オブジェクトの作成

永続オブジェクトを作成する前に、環境を作成し、接続をオープンしておく必要があります。次の例は、SQL 文で作成したデータベース表 addr_tab に永続オブジェクト addr を作成する方法を示しています。

```
CREATE TYPE ADDRESS AS OBJECT (state CHAR(2), zip_code CHAR(5));
CREATE TABLE ADDR_TAB of ADDRESS;
ADDRESS *addr = new(conn, "ADDR_TAB") ADDRESS("CA", "94065");
```

永続オブジェクトは、次のいずれか 1 つが発生した場合にのみ、データベース内に作成されます。

- トランザクションがコミットされた場合（Connection::commit()）
- オブジェクト・キャッシュがフラッシュされた場合（Connection::flushCache()）
- オブジェクト自体がフラッシュされた場合（PObject::flush()）

一時オブジェクトの作成

一時オブジェクトのインスタンス ADDRESS は、次の方法で作成されます。

```
ADDRESS *addr_trans = new ADDRESS("MD", "94111");
```

OTT ユーティリティによるオブジェクト表現の作成

C++ アプリケーションでデータベースからオブジェクトのインスタンスを取り出す場合は、オブジェクトのクライアント側表現が必要です。Object Type Translator（OTT）ユーティリティでは、データベース・オブジェクト型の C++ クラス表現を生成します。たとえば、次の型の宣言をデータベースに定義したとします。

```
CREATE TYPE address AS OBJECT (state CHAR(2), zip_code CHAR(5));
```

OTT ユーティリティは、次の C++ クラスを生成します。

```
class ADDRESS : public PObject {

protected:
   string state;
   string zip;

public:
   void *operator new(size_t size);
   void *operator new(size_t size, const Session* sess, const string&
      table);
   string getSQLTypeName(size_t size);
   ADDRESS(void *ctx) : PObject(ctx) { };
   static void *readSQL(void *ctx);
   virtual void readSQL(AnyData& stream);
   static void writeSQL(void *obj, void *ctx);
   virtual void writeSQL(AnyData& stream);
}
```

これらのクラス宣言は、OTT によって、名前を指定するヘッダー・ファイル（.h）に自動的に書き込まれます。このヘッダー・ファイルをアプリケーションのソース・ファイルにインクルードすると、オブジェクトにアクセスできます。PObject（および PObject から導出されたクラス）のインスタンスは、一時または永続インスタンスです。writeSQL メソッドおよび readSQL メソッドは、オブジェクトのシリアライズのために OCCI オブジェクト・キャッシュで内部的に使用され、OCCI クライアントでは使用および変更できません。

関連項目：

- OTT ユーティリティの詳細は、第 7 章「Object Type Translator ユーティリティの使用方法」を参照してください。

OCCI オブジェクト・アプリケーションの開発

この項では、基本的な OCCI オブジェクト・アプリケーションの開発に伴うステップを説明します。

基本的なオブジェクト・プログラム構造

オブジェクトを使用した OCCI アプリケーションの基本構造は、リレーショナル OCCI アプリケーションと類似していますが、オブジェクト機能には違いがあります。OCCI オブジェクト・プログラムに伴うステップは、次のとおりです。

1. `Environment` を初期化します。OCCI プログラミング環境をオブジェクト・モードで初期化します。

   アプリケーションでは通常、ヘッダー・ファイルからデータベース・オブジェクトの C++ クラス表現をインクルードする必要があります。第 7 章「Object Type Translator ユーティリティの使用方法」で説明しているように、Object Type Translator（OTT）ユーティリティを使用すると、このクラスを作成できます。

2. 接続を確立します。環境ハンドルを使用して、データベース・サーバーとの接続を確立します。

3. SQL 文を準備します。これは、ローカル（クライアント側）のステップで、プレースホルダのバインドを行います。オブジェクト・リレーショナル・アプリケーションの場合、この SQL 文はオブジェクトへの参照（`REF`）を戻します。

4. オブジェクトにアクセスします。

   a. 準備した文をデータベース・サーバーに関連付けて実行します。

   b. ナビゲーショナル・アクセスを使用して、データベース・サーバーからオブジェクト参照（`REF`）を取り出し、オブジェクトを確保します。その後、次の一部またはすべての操作を実行できます。

      * オブジェクトの属性を操作し、そのオブジェクトに**使用済み（変更済み）**のマークを付ける。
      * 別の 1 つのオブジェクト、または一連のオブジェクトへの参照をたどる。
      * 型および属性の情報にアクセスする。
      * 複合オブジェクト検索グラフをナビゲートする。
      * 変更したオブジェクトをデータベース・サーバーにフラッシュする。

   c. 連想アクセスを使用すると、SQL によってオブジェクト全体を値でフェッチできます。また、埋込み（参照不可能）オブジェクトを選択することもできます。その後、次の一部またはすべての操作を実行できます。

      * 表に値を挿入します。
      * 既存の値を変更します。

5. トランザクションをコミットします。このステップでは、変更されたすべてのオブジェクトがデータベース・サーバーに暗黙的に書き込まれ、変更がコミットされます。
6. 再利用しない文とハンドルを解放するか、準備した文を再実行します。

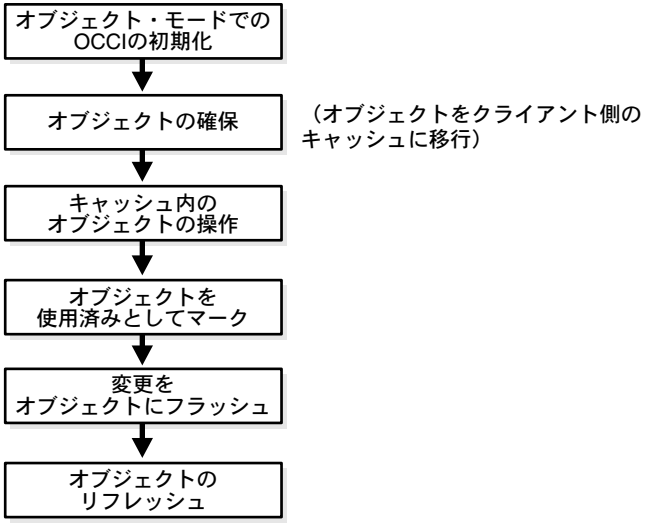
関連項目：

- OCCI によるデータベース・サーバーへの接続、SQL 文の処理およびハンドルの割当ての詳細は、第 2 章「リレーショナル・プログラミング」を参照してください。
- OTT ユーティリティの詳細は、第 7 章「Object Type Translator ユーティリティの使用方法」を参照してください。
- OCCI リレーショナル関数、`Connect` クラスおよび `getMetaData` メソッドの説明は、第 8 章「OCCI のクラスとメソッド」を参照してください。

基本的なオブジェクト操作のフロー

図 3-1 は、アプリケーションがオブジェクトを操作する方法について、そのプログラム・ロジックを簡単に示しています。図を簡略にするために、必要なステップの一部が省略されています。

図 3-1 基本的なオブジェクト操作のフロー



次の各項では、図 3-1 の各ステップについて説明します。

オブジェクト・モードでの OCCI の初期化

OCCI アプリケーションでオブジェクトにアクセスして操作する場合は、OCCI アプリケーションの最初のコールである createEnvironment メソッドの mode パラメータに、OBJECT の値を指定する必要があります。mode にこの値を指定することで、アプリケーションでオブジェクトを操作することを OCCI に通知します。この通知により、次の重要事項が有効になります。

- オブジェクト・ランタイム環境が確立されます。
- オブジェクト・キャッシュが設定されます。

> **注意：** mode パラメータに OBJECT を設定しないと、オブジェクト関連関数を使用しても結果はエラーになります。

次のコード例は、OCCI 環境の作成時に OBJECT モードを指定する方法を示しています。

```
Environment *env;
Connection *con;
Statement *stmt;

env = Environment::createEnvironment(Environment::OBJECT);
con = env->createConnection(userName, password, connectString);
```

データベース・オブジェクトがオブジェクト・キャッシュ内にロードされたときに、アプリケーションでメモリーを割り当てる必要はありません。オブジェクト・キャッシュによって、データベース・オブジェクトに対する透過的かつ効率的なメモリー管理が行われます。データベース・オブジェクトはオブジェクト・キャッシュ内にロードされると、ホスト言語（C++）表現に透過的にマップされます。

オブジェクト・キャッシュでは、オブジェクト・キャッシュ内のオブジェクト・コピーと、対応するデータベース・オブジェクトとの間の関連付けがメンテナンスされます。commit が実行されると、オブジェクト・キャッシュ内のオブジェクト・コピーに加えた変更が、データベースに対して自動的に伝播されます。

また、オブジェクト・キャッシュでは、参照をオブジェクトにマッピングするための参照表がメンテナンスされます。アプリケーションでオブジェクトへの参照を間接参照したときに、対応するオブジェクトがオブジェクト・キャッシュ内にキャッシュされていない場合は、データベース・サーバーに対してデータベースからオブジェクトをフェッチし、オブジェクト・キャッシュ内にロードするように、要求が自動的に送信されます。その後、同じ参照を間接参照すると、オブジェクト・キャッシュ自体が間接参照の対象となり、データベース・サーバーとのラウンドトリップを伴わないため、処理が速くなります。

同じ参照に関するその後の間接参照は、ラウンドトリップなしに、キャッシュからフェッチされます。ただし、コミット直後に行われた間接参照操作の場合を除きます。この場合は、オブジェクトの最新コピーがサーバーから戻されます。この機能はこのリリースにおける拡張機能で、各トランザクションの後にデータベースの最新オブジェクトが確実にキャッシュされるようにします。

さらに、オブジェクト・キャッシュでは、オブジェクト・キャッシュ内の各永続オブジェクトに対する確保カウントが保持されます。アプリケーションでオブジェクトへの参照を間接参照すると、その参照が指し示すオブジェクトの確保カウントが増えます。その後、同じオブジェクトへの参照を間接参照しても、確保カウントは変更されません。オブジェクトは、そのオブジェクトへの参照がスコープ外になるまで、オブジェクト・キャッシュ内に継続して確保され、OCCI クライアントによるアクセスが可能です。

確保カウントは、オブジェクトに対する参照カウントとして機能します。オブジェクトの確保カウントは、対象のオブジェクトへの参照が他にない場合にのみ 0（ゼロ）になり、その間このオブジェクトは、ガベージ・コレクションの対象となります。オブジェクト・キャッシュでは、最低使用頻度（LRU）アルゴリズムによって、オブジェクト・キャッシュのサイズが管理されます。このアルゴリズムにより、オブジェクト・キャッシュが最大サイズに達すると、確保カウント 0（ゼロ）のオブジェクトは解放されます。

オブジェクトの確保

通常、OCCI ユーザーは、オブジェクトの確保や確保解除を明示的に行う必要はありません。オブジェクト・キャッシュによって、キャッシュ内のすべてのオブジェクトの確保カウントが自動的に追跡され、記録されます。前述のとおり、オブジェクト・キャッシュは、参照がオブジェクトを指し示している場合は確保カウントを増やし、参照がスコープ外になったり、オブジェクトを指していない場合は確保カウントを減らします。

ただし、例外が 1 つあります。OCCI アプリケーションで `Ref<T>::ptr()` メソッドを使用してオブジェクトへのポインタを取得した場合、そのアプリケーションでは `PObject` クラスの `pin` メソッドと `unpin` メソッドを使用して、オブジェクト・キャッシュ内のオブジェクトの確保と確保解除を制御することができます。

キャッシュ内のオブジェクトの操作

オブジェクト・キャッシュではオブジェクト・コピーの内容は管理されず、オブジェクト・コピーが自動的にリフレッシュされることもありません。オブジェクト・コピーの妥当性と一貫性は、アプリケーションで確認する必要があります。

オブジェクトへの変更のフラッシュ

オブジェクト・キャッシュ内のオブジェクト・コピーに変更を加えた場合は、必ずアプリケーションで、変更したオブジェクトをデータベースにフラッシュする必要があります。

オブジェクト・キャッシュ用のメモリーは、オブジェクトがオブジェクト・キャッシュ内にロードされるときに、必要に応じて割り当てられます。

クライアント側オブジェクト・キャッシュは、プログラムのプロセス領域に割り当てられます。このオブジェクト・キャッシュは、データベース・サーバーから取り出してアプリケーションで使用できるオブジェクト用のメモリーです。

注意： OCCI 環境をオブジェクト・モードで初期化すると、実際にオブジェクト・コールを使用するかどうかに関係なく、オブジェクト・キャッシュ用のメモリーを割り当てることになります。

OCCI 環境ごとに、1 つのオブジェクト・キャッシュのみが割り当てられます。1 つの環境内で異なる接続を通じて取出しまたは作成されたオブジェクトは、すべて同じ物理オブジェクト・キャッシュを使用します。各接続には、それぞれ専用の論理オブジェクト・キャッシュがあります。

オブジェクトの削除

参照を間接参照してキャッシュに取得したオブジェクトに対しては、明示的な削除を実行しないでください。このようなオブジェクトの場合は、最初に参照が間接参照されると確保カウントが増加し、参照がスコープ外になると確保カウントが減少します。オブジェクトの確保カウントが 0（ゼロ）になると（そのオブジェクトへのすべての参照がスコープ外であることを示す）、そのオブジェクトは自動的にガベージ・コレクションの対象となり、その後、キャッシュから削除されます。

new 演算子をコールして作成された永続オブジェクトについては、トランザクションをコミットしない場合は、delete をコールする必要があります。コミットする場合、オブジェクトはコミット後にガベージ・コレクションの対象となります。これは、new を使用してこのようなオブジェクトが作成された場合、確保カウントは 0（ゼロ）から始まるためです。ただし、オブジェクトは使用済みであるため、キャッシュ内に残ります。コミット後は使用済みでなくなるため、ガベージ・コレクションの対象となります。したがって、削除の必要はありません。

コミットが実行されない場合は、delete を明示的にコールして、そのオブジェクトを破棄する必要があります。この操作は、そのオブジェクトへの参照がない場合のみ実行できます。一時オブジェクトの場合は、明示的に削除して、そのオブジェクトを破棄する必要があります。

永続オブジェクトに対して delete 演算子をコールすることはできません。マーク済みまたは使用済みではない永続オブジェクトは、確保カウントが 0（ゼロ）になると、ガベージ・コレクタによって解放されます。ただし、一時オブジェクトの場合は、明示的に削除して、破棄する必要があります。

オブジェクト・キャッシュのチューニング

オブジェクト・キャッシュには、次の 2 つの重要な関連パラメータがあります。

- 最大キャッシュ・サイズ率
- 最適キャッシュ・サイズ

これらのパラメータは、キャッシュ・メモリー使用量のレベルを参照し、対象のオブジェクトをキャッシュで自動的にエージ・アウトしてメモリーを解放する時期を判断するのに役立ちます。

現在キャッシュ内にあるオブジェクトによるメモリー使用量が最大キャッシュ・サイズに達するかそれを超えると、キャッシュでは、確保カウントが 0（ゼロ）のマークされていないオブジェクトの解放（またはエージ・アウト）が自動的に開始されます。キャッシュでは、キャッシュ内のメモリー使用量が最適なサイズに達するまで、または解放の対象となるオブジェクトがなくなるまで、オブジェクトの解放を継続します。

注意： キャッシュは、指定の最大キャッシュ・サイズより大きくなる場合があります。

オブジェクトの最大キャッシュ・サイズ（バイト単位）は、次に示すように、最適キャッシュ・サイズ（`optimal_size`）に最大キャッシュ・サイズ率（`max_size_percentage`）を増分して計算されます。

```
Maximum cache size = optimal_size + optimal_size * max_size_percentage / 100
```

最大キャッシュ・サイズ率のデフォルト値は 10% です。最適キャッシュ・サイズのデフォルト値は 8MB です。

これらのパラメータは、`Environment` クラスの次のメンバー関数を使用して設定または取得できます。

- `void setCacheMaxSize(unsigned int maxSize);`
- `unsigned int getCacheMaxSize() const;`
- `void setCacheOptSize(unsigned int OptSize);`
- `unsigned int getCacheOptSize() const;`

関連項目：

- 詳細は、第 8 章「OCCI のクラスとメソッド」を参照してください。

連想アクセスの概要

OCCI では、SQL を使用してオブジェクトを取り出し、DML 操作を実行できます。

- SQL によるオブジェクトへのアクセス
- 値の挿入と変更

関連項目：

- この項で説明した概念の実例は、付録 A「OCCI デモ・プログラム」および「occiobj.typ」と「occiobj.cpp」のコード例を参照してください。

SQL によるオブジェクトへのアクセス

前項では、ナビゲーショナル・アクセスについて説明し、初期オブジェクト集合の参照をフェッチするためにのみ SQL を使用しました。ここでは、SQL を使用してオブジェクトをフェッチする方法を説明します。

次の例は、`ResultSet::getObject` メソッドを使用して連想アクセスでオブジェクトをフェッチし、その際に、SQL を使用して `addr_tab` 表から各オブジェクトを取得する方法を示しています。

```
string sel_addr_val = "SELECT VALUE(address) FROM ADDR_TAB address";

ResultSet *rs = stmt->executeQuery(sel_addr_val);

while (rs->next())
{
   ADDRESS *addr_val = rs->getObject(1);
   cout << "state: " << addr_val->getState();
}
```

連想アクセスでフェッチされたオブジェクトは期限付きの値インスタンスで、一時オブジェクトと同様に振る舞います。`markModified`、`flush`、`markDeleted` などのメソッドは、永続オブジェクトにのみ適用できます。

これらのオブジェクトに加えた変更は、データベースに反映されません。

値の挿入と変更

前項では、SQL を使用してオブジェクトにアクセスする方法を説明しました。OCCI では、SQL を使用して、Statement::setObject メソッド・インタフェースを通じてデータベース・サーバー内に新規オブジェクトを挿入したり、データベース・サーバー内の既存オブジェクトを変更することもできます。

次の例では、一時オブジェクト Address を作成し、そのオブジェクトをデータベース表 addr_tab に挿入しています。

```
ADDRESS *addr_val = new address("NV", "12563");  // new a transient instance
stmt->setSQL("INSERT INTO ADDR_TAB values(:1)");
stmt->setObject(1, addr_val);
stmt->execute();
```

ナビゲーショナル・アクセスの概要

ナビゲーショナル・アクセスを使用して、次の一連の操作を実行します。

- データベース・サーバーからのオブジェクト参照（REF）の取出し
- オブジェクトの確保
- オブジェクト属性の操作
- オブジェクトのマークおよび変更のフラッシュ

関連項目：

- この項で説明した概念の実例は、付録 A「OCCI デモ・プログラム」および「occipobj.typ」と「occipobj.cpp」のコード例を参照してください。

データベース・サーバーからのオブジェクト参照（REF）の取出し

アプリケーションでオブジェクトを操作するには、最初にデータベース・サーバーから 1 つ以上のオブジェクトを取り出す必要があります。取出しは、1 つ以上のオブジェクトへの参照（REF）を戻す SQL 文を発行して行います。

注意： SQL 文によって、データベースから REF ではなく埋込みオブジェクトをフェッチすることもできます。

次の SQL 文では、1 つのオブジェクト address への REF をデータベース表 addr_tab から取り出しています。

```
string sel_addr = "SELECT REF(address) FROM addr_tab address
   WHERE zip_code = '94065'";
```

次のコード例は、問合せを実行し、結果セットから REF をフェッチする方法を示しています。

```
ResultSet *rs = stmt->executeQuery(sel_addr);
rs->next();
Ref<address> addr_ref = rs->getRef(1);
```

この時点で、オブジェクト参照を使用して、オブジェクトまたはデータベースからのオブジェクトにアクセスして操作できます。

関連項目：

- SQL 文の準備および実行の一般情報は、2-6 ページの「SQL の DDL 文と DML 文の実行」を参照してください。

オブジェクトの確保

フェッチを行うステップが完了すると、アプリケーションでは、オブジェクトへの REF を取得します。この時点では、まだ実際のオブジェクトは操作できません。オブジェクトを操作する前に、オブジェクトを、**確保する**必要があります。オブジェクトの確保によって、オブジェクトをオブジェクト・キャッシュ内にロードした後、オブジェクトの属性へのアクセスおよび変更、また 1 つのオブジェクトからその他のオブジェクトへの参照追跡が可能になります。アプリケーションでは、変更したオブジェクトをデータベース・サーバーに書き込む時期を制御することもできます。

注意： この項では、一度に 1 つのオブジェクトを確保する単純な操作の例を示します。複合オブジェクト検索を介した複数のオブジェクトの取出しの詳細は、3-18 ページの「複合オブジェクト検索の概要」を参照してください。

OCCI では、C++ ポインタを間接参照する場合と同様の方法で REF を間接参照することのみが必要となります。REF の間接参照によって、オブジェクトは、透過的に C++ クラス・インスタンスとしてインスタンス化されます。

前項の Address クラスの例に続けて、ユーザーが次のメソッドを追加したとします。

```
string  Address::getState()
{
   return state;
}
```

この REF を間接参照し、オブジェクトの属性およびメソッドにアクセスするには、次のように記述します。

```
string state = addr_ref->getState();     // -> pins the object
```

最初に Ref<T> (addr_ref) が間接参照されたときに、オブジェクトが確保されます（つまり、オブジェクトがデータベース・サーバーからオブジェクト・キャッシュ内にロードされます)。オブジェクトが確保された後は、Ref<T> の operator -> の動作は、C++ ポインタ (T *) の動作と同様になります。オブジェクトは、REF (addr_ref) がスコープ外となるまでオブジェクト・キャッシュ内に残留します。その後、ガベージ・コレクションの対象となります。

これでオブジェクトが確保され、アプリケーションでそのオブジェクトを変更できます。

オブジェクト属性の操作

オブジェクト属性の操作は、前項で示したようにオブジェクトにアクセスして行います。Address クラスに、state 属性を入力値に設定する次のユーザー定義メソッドがあるとします。

```
void Address::setState(string new_state)
{
   state = new_state;
}
```

次の例は、オブジェクト addr の state 属性を変更する方法を示しています。

```
addr_ref->setState("PA");
```

前述のとおり、オブジェクトがオブジェクト・キャッシュ内にない場合は、Ref<T> の operator -> の最初の呼出しによってロードされます。

オブジェクトのマークおよび変更のフラッシュ

前項の例では、オブジェクトの属性を変更しました。ただし、この時点では変更はクライアント側キャッシュにのみ存在しています。変更をデータベースに確実に書き込むには、アプリケーションで特定のステップを実行する必要があります。

オブジェクトへの変更済み（使用済み）のマーク付け

最初のステップでは、オブジェクトが変更されたことを示します。これは、オブジェクトに対して markModified メソッド（PObject の導出メソッド）をコールして行います。このメソッドは、オブジェクトに**使用済み**（変更済み）のマークを付けます。

前の例に続けて、オブジェクト属性の操作後、次のように addr_ref が参照するオブジェクトに使用済みのマークを付けます。

```
addr_ref->markModified()
```

データベースへの変更の記録

使用済みフラグが設定されているオブジェクトは、変更をデータベースに記録するために、データベース・サーバーにフラッシュする必要があります。これは、次の 3 つの方法で実行できます。

- 使用済みとマークされた 1 つのオブジェクトをフラッシュするには、POBject の導出メソッドである flush メソッドをコールします。
- オブジェクト・キャッシュ全体をフラッシュするには、Connection::flushCache メソッドを使用します。この場合、OCCI はオブジェクト・キャッシュによって保持されている使用済みリストを横断して、使用済みオブジェクトをすべてフラッシュします。
- トランザクションをコミットするには、Connection::commit メソッドをコールします。この場合も、OCCI は使用済みリストを横断して、オブジェクトをデータベース・サーバーにフラッシュします。この使用済みリストには、新しく作成された永続オブジェクトが含まれています。

オブジェクト・キャッシュでのガベージ・コレクション

オブジェクト・キャッシュには、環境ハンドルのプロパティである、次の 2 つの重要な関連パラメータがあります。

- 最大キャッシュ・サイズ
- 最適キャッシュ・サイズ

これらのパラメータは、キャッシュ・メモリー使用量のレベルを参照し、対象となるオブジェクトをキャッシュで自動的にエージ・アウトしてメモリーを使用可能にする時期を判断するのに役立ちます。

現在キャッシュ内にあるオブジェクトによるメモリー使用量が最高水位標に達するかそれを超えると、キャッシュでは、確保カウントが 0（ゼロ）のマークされていないオブジェクトの解放が自動的に開始されます。キャッシュでは、キャッシュ内のメモリー使用量が最適なサイズに達するまで、またはキャッシュ内に対象のオブジェクトがなくなるまで、オブジェクトの解放を継続します。

OCCI は、キャッシュの最大または最適サイズを設定および取得するための set メソッドおよび get メソッドを環境内で提供します。最大キャッシュ・サイズは、キャッシュの最適サイズの率として指定します。バイト単位による最大キャッシュ・サイズは、次のように計算されます。

```
maximum_cache_size = optimal_size + optimal_size * max_size_percentage/100
```

最大キャッシュ・サイズのデフォルト値は 10% で、最適キャッシュ・サイズのデフォルト値は 8MB です。オーバーロードされた PObject::new 演算子を使用して永続オブジェクトが作成されている場合、新しく作成されたオブジェクトは使用済みとしてマークされ、その確保カウントは 0（ゼロ）になります。

オブジェクトの確保カウントを参照カウントとして使用する方法、および確保カウントが 0（ゼロ）のマークされていないオブジェクトをガベージ・コレクションの対象にする方法については、3-10 ページの「オブジェクトの確保」で説明しています。新しく作成された永続オブジェクトの場合は、トランザクションのコミットまたは異常終了後、およびオブジェクトの確保カウントが 0（ゼロ）の場合（つまり、そのオブジェクトへの参照がない場合）に、オブジェクトのマークが解除されます。その後、そのオブジェクトはエージ・アウトの対象となります。

参照のトランザクション一貫性

前の項で説明したとおり、最初に `Ref<T>` を間接参照すると、オブジェクトはデータベース・サーバーからオブジェクト・キャッシュにロードされます。それ以降は、`Ref<T>` の `operator ->` の動作は C++ ポインタの動作と同じになり、キャッシュ内のオブジェクト・コピーへのアクセスを提供します。ただし、トランザクションがコミットまたは異常終了すると、キャッシュ内のオブジェクト・コピーは無効になります。これは、他のクライアントによってその内容が変更されている可能性があるためです。したがって、トランザクションの終了後に `Ref<T>` を再度間接参照すると、オブジェクト・キャッシュでは、オブジェクトがすでに無効であることが認識され、最新のコピーがデータベース・サーバーからフェッチされます。

複合オブジェクト検索の概要

ここまでの説明では、一度に 1 つのオブジェクトのみフェッチまたは確保する例を示しました。その場合、オブジェクトを取り出すためのデータベース・サーバー・ラウンドトリップが、確保オペレーションごとに別個に発生します。

オブジェクト指向アプリケーションでは、相互に関連したオブジェクトの集合として問題をモデル化することがよくあります。これらのオブジェクトは、オブジェクトのグラフを形成します。アプリケーションでは、初期オブジェクト集合の一部からオブジェクトの処理を開始し、その一部のオブジェクトの参照を使用して残りのオブジェクトを横断します。クライアント / サーバー設定では、横断のたびにオブジェクトをフェッチするためのネットワーク・ラウンドトリップが発生し、非効率的です。

このようなアプリケーションでは、**複合オブジェクト検索**（COR）を使用することによってパフォーマンスを向上できます。COR はプリフェッチ・メカニズムです。アプリケーションでは、このメカニズムを使用して、リンクされた一連のオブジェクトを 1 回のネットワーク・ラウンドトリップで取り出すための基準（内容と境界）を指定します。

注意： COR では、プリフェッチされたオブジェクトを確保しません。オブジェクトはオブジェクト・キャッシュ内にフェッチされるため、後続の確保コールはローカル操作で行われます。

複合オブジェクトは、論理的に関連したオブジェクトの集合です。ルート・オブジェクトと、指定のネスト・レベルに基づいてそれぞれプリフェッチされる一連のオブジェクトで構成されています。**ルート・オブジェクト**は、明示的にフェッチまたは確保されます。**ネスト・レベル**は、複合オブジェクトのルート・オブジェクトから指定のプリフェッチ・オブジェクトまでを最短で横断した場合の参照の数です。

アプリケーションで複合オブジェクトを指定するには、複合オブジェクトの内容と境界を記述します。複合オブジェクトのフェッチは、環境の**プリフェッチ制限**、およびオブジェクト・キャッシュ内で使用可能なメモリーの量によって制約されます。

注意： 複合オブジェクト検索を使用するとパフォーマンスのみが改善され、機能性は追加されません。したがって、使用しないことも可能です。

複合オブジェクト検索

OCCI アプリケーションで COR を実行するには、適切な属性を Ref<T> に設定した後、次のメソッドを使用して間接参照します。

```
// prefetch attributes of the specified type name up to the the specified depth
Ref<T>::setPrefetch(const string &typeName, unsigned int depth);
// prefetch all the attribute types up to the specified depth.
Ref<T>::setPrefetch(unsigned int depth);
```

アプリケーションでは、REF を通して特定の深さまで到達できる（推移閉包）オブジェクトすべてをフェッチすることも選択できます。そのためには、目的の深さをレベル・パラメータとして設定する必要があります。前述の 2 つの例では、アプリケーションで (PO object REF, OCCI_MAX_PREFETCH_DEPTH) と (PO object REF, 1) をそれぞれ指定して、必要なオブジェクトをプリフェッチすることもできます。この場合は多数の余分なフェッチが発生しますが、指定が簡単であり、データベース・サーバー・ラウンドトリップが 1 回のみで済みます。

この説明の例として、次の型宣言があるとします。

```
CREATE TYPE customer(...);
CREATE TYPE line_item(...);
CREATE TYPE line_item_varray as VARRAY(100) of REF line_item;
CREATE TYPE purchase_order AS OBJECT
( po_number         NUMBER,
  cust              REF customer,
  related_orders    REF purchase_order,
  line_items        line_item_varray)
```

purchase_order 型には、スカラー値 po_number、line_items の VARRAY および 2 つの参照が含まれます。1 つは customer 型への参照、もう 1 つは purchase_order 型への参照であり、この型がリンク済みリストとして実装されることを示しています。

複合オブジェクトをフェッチする場合は、アプリケーションで次の指定を行う必要があります。

- 目的のルート・オブジェクトへの参照。
- 複合オブジェクトの境界を指定するための 1 組以上の型および深さ情報。型情報は COR でたどる REF 属性を指示し、ネスト・レベルはリンクをたどるレベル数を指示します。

前述の purchase_order オブジェクトの場合は、アプリケーションで次の指定を行う必要があります。

- ルート purchase_order オブジェクトへの参照。
- customer、purchase_order または line_item についての 1 組以上の型および深さ情報。

発注情報をプリフェッチするアプリケーションでは、発注書を発行した顧客の情報にアクセスすることも十分考えられます。単純なナビゲーションでは、2 つのオブジェクトを取り出すためにデータベース・サーバーに 2 回アクセスする必要があります。

複合オブジェクト検索を使用すると、purchase_order オブジェクトを確保するときに customer をプリフェッチできます。この場合、複合オブジェクトは、purchase_order オブジェクトと参照する customer オブジェクトで構成されます。

前述の例で、発注情報と関連する顧客情報をプリフェッチする必要がある場合は、次に示すように、アプリケーションで purchase_order オブジェクトを指定し、customer はネスト・レベル 1 までたどるように指示します。

```
Ref<PURCHASE_ORDER> poref;
poref.setPrefetch("CUSTOMER",1);
```

発注情報と、オブジェクト・グラフに含まれるすべてのオブジェクトをプリフェッチする必要がある場合は、次に示すように、アプリケーションで purchase_order オブジェクトを指定し、customer および purchase_order は両方とも、最大のネスト・レベルまでたどるように指示します。

```
Ref<PURCHASE_ORDER> poref;
poref.setPrefetch("CUSTOMER", OCCI_MAX_PREFETCH_DEPTH);
poref.setPrefetch("PURCHASE_ORDER", OCCI_MAX_PREFETCH_DEPTH);
```

OCCI_MAX_PREFETCH_DEPTH は、ルート・オブジェクトから参照を通して到達できる指定の型のオブジェクトすべてのプリフェッチを指定します。

発注情報と発注情報に関連付けられた明細項目すべてをプリフェッチする必要がある場合は、次に示すように、アプリケーションで purchase_order オブジェクトを指定し、line_items は最大のネスト・レベルまでたどるように指示します。

```
Ref<PURCHASE_ORDER> poref;
poref.setPrefetch("LINE_ITEM", 1);
```

複合オブジェクトのプリフェッチ

複合オブジェクトを指定してフェッチした後に行う、複合オブジェクトに含まれるオブジェクトのフェッチでは、ネットワーク・ラウンドトリップは発生しません。これは、オブジェクトがすでにプリフェッチ済みであり、オブジェクト・キャッシュ内にあるためです。プリフェッチするオブジェクトが多すぎると、オブジェクト・キャッシュがあふれることがあるので注意が必要です。オブジェクト・キャッシュがあふれると、アプリケーションがすでに確保している他のオブジェクトがキャッシュから押し出され、パフォーマンスの向上ではなく逆にパフォーマンスの低下につながることがあります。

注意： プリフェッチ済みのオブジェクトをすべて保持できるメモリーがオブジェクト・キャッシュにない場合は、オブジェクトの一部がプリフェッチされないことがあります。アプリケーションでは、後でそれらのオブジェクトにアクセスするときに、ネットワーク・ラウンドトリップが発生します。

すべてのプリフェッチ・オブジェクトに対して、SELECT 権限が必要です。複合オブジェクト内のオブジェクトに対する SELECT 権限がアプリケーションにない場合、そのオブジェクトはフェッチできません。

コレクションの操作

Oracle では、可変長配列（順序付けられたコレクション）とネストした表（順序付けられていないコレクション）の 2 種類のコレクションをサポートしています。OCCI では、この 2 種類のコレクションはいずれも標準テンプレート・ライブラリ（Standard Template Library: STL）のベクター・コンテナにマップされるため、強力、柔軟かつ高速な STL ベクターによるコレクション要素へのアクセスと操作が可能になります。次の例は、VARRAY および VARRAY 型の属性が含まれたオブジェクトを作成する SQL の DDL 文を示しています。

```
CREATE TYPE ADDR_LIST AS VARRAY(3) OF REF ADDRESS;
CREATE TYPE PERSON AS OBJECT (name VARCHAR2(20), addr_l ADDR_LIST);
```

次に示すのは、OTT によって生成された C++ クラス宣言です。

```
class PERSON : public PObject
{
   protected:
      string name;
      vector< Ref< ADDRESS > > addr_1;

   public:
      void *operator new(size_t size);
      void *operator new(size_t size,
      const Session* sess,
```

```
        const string& table);
        string getSQLTypeName(size_t size);
        PERSON (void *ctx) : PObject(ctx) { };
        static void *readSQL(void *ctx);
        virtual void readSQL(AnyData& stream);
        static void writeSQL(void *obj, void *ctx);
        virtual void writeSQL(AnyData& stream);
}
```

関連項目：

- この項で説明した概念の実例は、付録 A「OCCI デモ・プログラム」および「occicoll.cpp」のコード例を参照してください。

埋込みオブジェクトのフェッチ

使用しているアプリケーションで、埋込みオブジェクト・インスタンス（オブジェクト表でなく、通常の表の列に格納されているオブジェクト）をフェッチする必要がある場合、REF 検索機能は使用できません。埋込みインスタンスにはオブジェクト識別子がないため、埋込みインスタンスへの参照は取得できません。これは埋込みインスタンスがオブジェクト・ナビゲーションの基礎として機能しないことを意味します。しかし、アプリケーションで埋込みインスタンスをフェッチすることが必要な状況は数多くあります。

たとえば、address 型が作成されたとします。

```
CREATE TYPE address AS OBJECT
( street1             varchar2(50),
  street2             varchar2(50),
  city                varchar2(30),
  state               char(2),
  zip                 number(5))
```

その型は、他の表で列のデータ型として使用できます。

```
CREATE TABLE clients
( name          varchar2(40),
  addr          address)
```

OCCI アプリケーションで次の SQL 文を発行します。

```
SELECT addr FROM clients
WHERE name='BEAR BYTE DATA MANAGEMENT'
```

この文は、clients 表の埋込みオブジェクト address を戻します。アプリケーションでは、このオブジェクトの属性値を別の処理で使用できます。3-13 ページの「連想アクセスの概要」の項で説明したように、アプリケーションで文を実行してオブジェクトをフェッチする必要があります。

NULL

データベース表で、行の中に値のない列がある場合、その列は NULL、または NULL を含むと呼ばれます。オブジェクトには、次の 2 種類の NULL を適用できます。

- オブジェクトの属性はすべて、NULL 値を持つことができます。これはオブジェクトの属性の値が不明であることを意味します。
- オブジェクトは、**アトミック NULL** にできます。これはオブジェクト全体の値が不明なことを意味します。

アトミック NULL とは、オブジェクトが存在しないということではありません。アトミック NULL のオブジェクトは値が不明なだけで、存在しています。つまり、データを持たないオブジェクトとみなすことができます。

OCCI では、オブジェクト属性のすべての型について、それぞれ対応するクラスが用意されています。たとえば、NUMBER 属性型は Number クラスにマップされ、REF は RefAny にマップされます。データ型を表す各 OCCI クラスでは、次の 2 つのメソッドを使用できます。

- isNull: オブジェクトが NULL かどうかを戻します。
- setNull: オブジェクトを NULL に設定します。

同様に、これらのメソッドは、すべてのオブジェクトによって PObject クラスから継承され、オブジェクトにアクセスして NULL 情報を自動的に設定するために使用されます。

オブジェクト参照の使用

アプリケーションでは、OCCI のポインタや参照を使用して、オブジェクトの内容に柔軟にアクセスできます。OCCI では、PObject::getRef メソッドを使用して永続オブジェクトへの参照を戻すことができます。このコールは、永続オブジェクトに対してのみ有効です。

オブジェクトの解放

OCCI ユーザーは、オーバーロードされた PObject::operator new を使用して永続オブジェクトを作成できます。PObject::operator delete メソッドをコールしてオブジェクトを解放するのは、ユーザーの役割です。

オブジェクトは、オブジェクト・キャッシュから解放しても、データベース・サーバーからは削除されません。オブジェクトをデータベース・サーバーから削除するには、PObject::markDelete メソッドをコールする必要があります。演算子 delete は、オブジェクトを解放し、オブジェクト・キャッシュ内のメモリーを再生しますが、オブジェクトをデータベース・サーバーから削除することはありません。

型の継承

オブジェクトの型の継承は、C++ や Java での継承に類似している点が多くあります。オブジェクト型は、既存のオブジェクト型のサブタイプとして作成できます。サブタイプは、元の型であるスーパータイプの属性およびメソッド（メンバー関数およびプロシージャ）をすべて継承するといえます。単一の継承のみがサポートされます。つまり、1 つのオブジェクトが複数のスーパータイプを所有することはできません。継承した属性やメソッドには、新しい属性やメソッドを追加できます。継承した任意のメソッドをオーバーライド（実装を再定義）することもできます。サブタイプは、そのスーパータイプの拡張（つまり、スーパータイプの継承）といえます。

関連項目：

- この項目の詳細は、『Oracle9*i* アプリケーション開発者ガイド - オブジェクト・リレーショナル機能』を参照してください。

たとえば、Person_t 型は、サブタイプ Student_t とサブタイプ Employee_t を所有できます。また、Student_t も独自のサブタイプ PartTimeStudent_t を所有できます。サブタイプを所有するには、型宣言にフラグ NOT FINAL が必要です。デフォルトのフラグは FINAL で、この型はサブタイプを所有できないことを意味します。

この章でこれまでに説明した型は、すべて FINAL です。Oracle8*i* リリース 8.1.7 以前に開発されたアプリケーションの型は、すべて FINAL です。FINAL の型は、NOT FINAL に変更できます。サブタイプを持たない NOT FINAL 型は、FINAL に変更できます。次の例では、Person_t が NOT FINAL として宣言されています。

```
CREATE TYPE Person_t AS OBJECT
( ssn NUMBER,
  name VARCAHR2(30),
  address VARCHAR2(100)) NOT FINAL;
```

サブタイプは、スーパータイプで宣言された属性とメソッドをすべて継承します。また、新しい属性やメソッドを宣言することもできます。その場合は、スーパータイプとは異なる名前を指定する必要があります。キーワード UNDER によって、次のようにスーパータイプを識別します。

```
CREATE TYPE Student_t UNDER Person_t
( deptid NUMBER,
  major  VARCHAR2(30)) NOT FINAL;
```

新しく宣言した属性 deptid および major は、サブタイプ Student_t に所属しています。たとえば、サブタイプ Employee_t を次のように宣言します。

```
CREATE TYPE Employee_t UNDER Person_t
( empid NUMBER,
  mgr   VARCHAR2(30));
```

関連項目：

- このコード例に関して OTT が生成するクラスの詳細は、3-27 ページの「OTT での型の継承のサポート」を参照してください。

このサブタイプ Student_t は、PartTimeStudent_t などの独自のサブタイプを所有できます。

```
CREATE TYPE PartTimeStuden_t UNDER Student_t
( numhours NUMBER) ;
```

関連項目：

- この項で説明した概念の実例は、付録 A「OCCI デモ・プログラム」および「occiinh.typ」と「occiinh.cpp」のコード例を参照してください。

代用性

ポリモフィズムの利点の一部は、プロパティが持つ代用性によるものです。代用性によって、サブタイプの値をスーパータイプ用に記述された元のコードで使用できます。この場合、サブタイプに関する事前の知識は特に必要ありません。サブタイプの値は、メソッドが特化されて異なるメカニズムが使用されている場合でも、周辺のコードに対応してスーパータイプの値と同じように振る舞います。

インスタンスの代用性とは、サブタイプのオブジェクト値をスーパータイプで宣言されたコンテキスト内で使用できるようにする機能を指します。REF の代用性とは、サブタイプへの REF をスーパータイプへの REF で宣言されたコンテキスト内で使用できるようにする機能を指します。

REF 型の属性は代用可能です。つまり、REF T として定義された属性は、T のインスタンスまたはそのサブタイプの任意のインスタンスへの REF を保持できます。

オブジェクト型の属性は代用可能です。つまり、T（オブジェクト）型として定義された属性は、T のインスタンスまたはそのサブタイプの任意のインスタンスを保持できます。

コレクション要素の型は代用可能です。つまり、T 型の要素を持つコレクションを定義した場合、そのコレクションは、T 型のインスタンスとそのサブタイプの任意のインスタンスを保持できます。オブジェクト属性の代用性の例を次に示します。

```
CREATE TYPE Book_t AS OBJECT
( title VARCHAR2(30),
  author Person_t     /* substitutable */);
```

したがって、Book_t インスタンスは、次のように、タイトル文字列および Person_t（または Person_t の任意のサブタイプ）オブジェクトを指定することによって作成できます。

```
Book_t('My Oracle Experience',
        Employee_t(12345, 'Joe', 'SF', 1111, NULL))
```

NOT INSTANTIABLE の型とメソッド

型は NOT INSTANTIABLE として宣言できます。これは、その型に（デフォルトまたはユーザー定義の）コンストラクタがないことを意味します。したがって、この型のインスタンスは構築できません。通常、このような型は、インスタンス化可能なサブタイプの定義に使用されます。このプロパティの使用方法を次に示します。

```
CREATE TYPE Address_t AS OBJECT(...) NOT INSTANTIABLE NOT FINAL;
CREATE TYPE USAddress_t UNDER Address_t(...);
CREATE TYPE IntlAddress_t UNDER Address_t(...);
```

型のメソッドは、NOT INSTANTIABLE として宣言できます。メソッドを NOT INSTANTIABLE として宣言することは、型がそのメソッドを実装しないことを意味します。また、NOT INSTANTIABLE メソッドが含まれている型は、必ず NOT INSTANTIABLE として宣言する必要があります。たとえば、次のようにします。

```
CREATE TYPE T AS OBJECT
(
  x NUMBER,
  NOT INSTANTIABLE MEMBER FUNCTION func1() RETURN NUMBER
) NOT INSTANTIABLE;
```

NOT INSTANTIABLE 型のサブタイプは、スーパータイプの任意の NOT INSTANTIABLE メソッドをオーバーライドして、具体的な実装を行うことができます。NOT INSTANTIABLE メソッドが残っている場合、サブタイプも必ず NOT INSTANTIABLE として宣言する必要があります。

NOT INSTANTIABLE サブタイプは、インスタンス化可能なスーパータイプの下に定義できます。NOT INSTANTIABLE 型を FINAL として宣言することは無効なため、使用できません。

OCCI での型の継承のサポート

次のコールは、型の継承をサポートしています。

Connection::getMetaData()

このメソッドによって、継承された型に固有の情報が提供されます。継承された型のプロパティには、その他の属性が追加されています。たとえば、型のスーパータイプを取得できます。

バインド関数および定義関数

Statement クラスの setRef、setObject および setVector メソッドは、REF、オブジェクトおよびコレクションをそれぞれバインドするために使用します。これらの関数はすべて、REF、インスタンスおよびコレクション要素の代用性をサポートしています。同様に、データをフェッチするための対応する get*xxx* メソッドも代用性をサポートしています。

OTT での型の継承のサポート

継承によるオブジェクトのクラス宣言は、そのクラスが親の型クラスから導出され、親のクラスにない属性に対応するフィールドのみが含まれていることを除くと、単純なオブジェクト宣言と似ています。この宣言の構造は、次のとおりです。

```
class <typename> : public <parentTypename> {

protected:
<OCCItype1> <attributename1>;
   .
   .
   .
<OCCItypen> <attributenamen>;

public:

void *operator new(size_t size);
void *operator new(size_t size, const Session* sess, const string& table);
string getSQLTypeName(size_t size);
<typename> (void *ctx) : <parentTypename>(ctx) { };
static void *readSQL(void *ctx);
virtual void readSQL(AnyData& stream);
static void writeSQL(void *obj, void *ctx);
virtual void writeSQL(AnyData& stream);

}
```

この構造では、変数はすべて単純なオブジェクトの場合と同じです。parentTypename は親の型の名前、つまり、その型名が継承している型のクラス名です。

サンプル OCCI アプリケーション

この章で説明した機能の一部を使用したサンプル OCCI アプリケーションを次に示します。

最初に SQL DDL を示し、次に OTT マッピングを記載します。

```
CREATE TYPE FULL_NAME AS OBJECT (first_name CHAR(20), last_name CHAR(20));
CREATE TYPE ADDRESS AS OBJECT (state CHAR(20), zip CHAR(20));
CREATE TYPE ADDRESS_TAB  AS VARRAY(3) OF REF ADDRESS;
CREATE TYPE PERSON AS OBJECT (id NUMBER, name FULL_NAME,
   curr_addr REF ADDRESS, prev_addr_l ADDRESS_TAB);
CREATE TYPE STUDENT UNDER  PERSON (school_name CHAR(20));
```

OTT によって、FULL_NAME、ADDRESS、PERSON および PFGRFDENT クラスの宣言が demo.h に生成されると想定します。次のサンプル OCCI アプリケーションは、OTT によって生成されたクラスを拡張し、ユーザー定義のメソッドをいくつか追加します。

```
/************************ myDemo.h ***************************************/

#include demo.h

// declarations for the MyFullName class.
class MyFullname : public FULLNAME {
   public:
      MyFullname(string first_name, string last_name);
      void displayInfo();
}

// declarations for the MyAddress class.
class MyAddress : public ADDRESS {
   public:
      MyAddress(string state_i, string zip_i);
      void displayInfo();
}

// declarations for the MyPerson class.
class MyPerson : public PERSON {
   public:
      MyPerson(Number id_i, MyFullname *name_i,
         Ref<MyAddress>& addr_i);
      void move(const Ref<MyAddress>& new_addr);
      void displayInfo();
}
```

```
/*************************myDemo.cpp************************************/

/* initialize MyFullName */
MyFullName::MyFullname(string first_name, string last_name)
   : FirstName(first_name), LastName(last_name)
{ }

/* display all the information in MyFullName */
void MyFullName::displayInfo()
{
   cout << "FIRST NAME is" << FirstName << endl;
   cout << "LAST NAME is" << LastName << endl;
}

/************************************************************************/
// method implementations for MyAddress class.
/************************************************************************/

/* initialize MyAddress */
MyAddress::MyAddress(string state_i, string zip_i)
   : state(state_i), zip(zip_i)
{ }

/* display all the information in MyAddress */
void MyAddress::displayInfo()
{
   cout << "STATE is" << state << endl;
   cout << "ZIP is" << zip << endl;
}

/************************************************************************/
// method implementations for MyPerson class.
/************************************************************************/

/* initialize MyPerson */
MyPerson::MyPerson(Number id_i,
MyFullName* name_i, const Ref<MyAddress>& addr_i)
   : id(id_i), name(name_i), curr_addr(addr_i)
{ }

/* Move Person from curr_addr to new_addr */
void MyPerson::move(const Ref<MyAddress>& new_addr)
{
   prev_addr_l.push_back(curr_addr);     // append curr_addr to the vector
   curr_addr = new_addr;
   this->mark_modified();                // mark the object as dirty
}
```

```
/*  Display all the information of MyPerson */
void MyPerson::displayInfo()
{
   cout << "ID is" << CPerson::id << endl;
   name->displayInfo();

   // de-referencing the Ref attribute using -> operator
   curr_addr->displayInfo();
   cout << "Prev Addr List: " << endl;
   for (int i = 0; i < prev_addr_l.size(); i++)
   {
      // access the collection elements using [] operator
      prev_addr_l[i]->displayInfo();
   }
}


/*************************************************************************/
// main function of this OCCI application.
// This application connects to the database as scott/tiger, creates
// the Person (Joe Black) whose Address is in CA, and commits the changes.
// The Person object is then retrieved from the database and its
// information is displayed. A second Address object is created (in PA),
// then the previously retrieved Person object (Joe Black) is moved to
// this new address. The Person object is then displayed again.
/*************************************************************************/
int main()
{
   Environment *env = Environment::createEnvironment()
   Connection *conn = env->createConnection("scott", "tiger");

   /* Call the OTT generated function to register the mappings */
   RegisterMappings(env);

   /* create a persistent object of type ADDRESS in the database table,
      ADDR_TAB */
   MyAddress *addr1 = new(conn, "ADDR_TAB")
   MyAddress("CA", "94065");
   MyFullName name1("Joe", "Black");

   /* create a persistent object of type Person in the database table,
      PERSON_TAB */
   MyPerson *person1 = new(conn, "PERSON_TAB")
   MyPerson(1,&name1,addr1->getRef());
```

```
   /* commit the transaction which results in the newly created objects, addr,
      person1 being flushed to the server */
   conn->commit();

   Statement *stmt = conn->createStatement();

   ResultSet *resultSet
      = stmt->executeQuery("SELECT REF(Person) from person_tab where id = 1");

   ResultSetMetaData rsMetaData = resultSet->getMetaData();

   if (Types::POBJECT != rsMetaData.getColumnType(1))
      return -1;

   Ref<MyPerson> joe_ref = (Ref<MyPerson>) resultSet.getRef(1);

   joe_ref->displayInfo();

   /* create a persistent object of type ADDRESS, in the database table,
      ADDR_TAB */
   MyAddress *new_addr1 = new(conn, "ADDR_TAB") MyAddress("PA", "92140");
   joe_ref->move(new_addr1->getRef());
   joe_ref->displayInfo();

   /* commit the transaction which results in the newly created object,
      new_addr and the dirty object, joe to be flushed to the server. Note that
      joe was marked dirty in move(). */
   conn->commit();

   /* The following delete statements delete the objects only from the
      application cache. To delete the objects from server, mark_deleted()
      should be used. */
   delete addr1;
   delete person1;
   delete new_addr1;

   conn->closeStatement(stmt);
   env->terminateConnection(conn);
   Environment::terminateEnvironment(env);
   return 0;
}
```

4

データ型

この章には、Oracle C++ Call Interface（OCCI）アプリケーションで使用される Oracle データ型のリファレンスが記載されています。この情報は、アプリケーションとデータベース・サーバー間におけるデータ転送時に発生する、データの内部表現と外部表現との間の変換を理解するのに役立ちます。

この章は、次の項目で構成されています。

Oracle データ型の概要

C++ プログラムと Oracle データベース・サーバー間では、通信を正確に行うことが重要です。OCCI アプリケーションでは、SQL 問合せを使用してデータベース表からデータを取り出すか、SQL の INSERT、UPDATE および DELETE 関数を使用して既存のデータを変更できます。ホスト言語の C++ とデータベース・サーバー間の通信を容易に行うには、C++ データ型と Oracle データ型間の変換方法を理解する必要があります。

Oracle データベースでは、値は表内の列に格納されています。Oracle 内部では、データは内部データ型と呼ばれる特定フォーマットで表現されます。NUMBER、VARCHAR2 および DATE などは、この Oracle 内部データ型の例です。

OCCI アプリケーションでは、ホスト言語データ型、またはホスト言語によって事前定義された外部データ型を使用します。データが OCCI アプリケーションとデータベース・サーバーの間で移動する際、データベースのデータは、内部データ型から外部データ型に変換されます。

OCCI 型とデータ変換

OCCI では、Type と呼ばれる列挙子が定義されています。この列挙子には、OCCI アプリケーションで使用可能なデータの表現フォーマットがリストされています。この表現フォーマットは、外部データ型と呼ばれています。OCCI アプリケーションからデータベース・サーバーにデータを送信すると、外部データ型によって、そのデータのフォーマットがデータベース・サーバーに対して示されます。また、OCCI アプリケーションがデータベース・サーバーからデータを要求すると、外部データ型によって、戻されるデータのフォーマットが示されます。

たとえば、NUMBER 列から値を取り出す場合は、その値を OCCIINT フォーマット（整変数への符号付き整数フォーマット）で取り出すようにプログラムを設定できます。あるいは、データを OCCIFLOAT フォーマット（浮動小数点フォーマット）で送信するようにクライアントを設定できます。このフォーマットは、NUMBER 型の列に挿入される C++ の float 変数に格納されています。

OCCI アプリケーションでは、set*xxx* メソッド（外部データ型はメソッド名によって暗黙的に指定されます）をコールするか、registerOutParam、setDataBuffer または setDataBufferArray メソッド（外部データ型はメソッドのコールによって明示的に指定されます）をコールして、入力パラメータを Statement にバインドします。同様に、データ値が ResultSet オブジェクトを使用してフェッチされた場合は、取り出したデータの外部表現を指定する必要があります。この指定は、get*xxx* メソッド（外部データ型はメソッド名によって暗黙的に指定されます）をコールするか、setDataBuffer メソッド（外部データ型はメソッドのコールによって明示的に指定されます）をコールして行います。

注意： 外部データ型は、内部データ型より多くあります。単一の外部データ型が単一の内部データ型にマップされる場合や、多数の外部データ型が単一の内部データ型にマップされる場合があります。この多対 1 のマッピングによって、柔軟性が向上します。

関連項目：

- 4-5 ページ「外部データ型」

内部データ型

この項では、Oracle が提供する内部（組込み）データ型について説明します。

表 4-1 は、Oracle 内部データ型とそれぞれの内部最大長を示しています。

表 4-1 Oracle 内部データ型

内部データ型	コード	内部最大長
BFILE	114	4GB
CHAR、NCHAR	96	2000 バイト
DATE	12	7 バイト
INTERVAL DAY TO SECOND REF	183	11 バイト
INTERVAL YEAR TO MONTH REF	182	5 バイト
LONG	8	2GB（2^31-1 バイト）
LONG RAW	24	2GB（2^31-1 バイト）
NUMBER	2	21 バイト
RAW	23	2000 バイト
REF	111	
REF BLOB	113	4GB
REF CLOB、REF NCLOB	112	4GB
ROWID	11	10 バイト
TIMESTAMP	180	11 バイト
TIMESTAMP WITH LOCAL TIME ZONE	231	7 バイト
TIMESTAMP WITH TIME ZONE	181	13 バイト

表 4-1 Oracle 内部データ型（続き）

内部データ型	コード	内部最大長
UROWID	208	4000 バイト
ユーザー定義型（オブジェクト型、VARRAY、ネストした表）	108	
VARCHAR2、NVARCHAR2	1	4000 バイト

関連項目：

- 『Oracle9*i* SQL リファレンス』
- 『Oracle9*i* データベース概要』

文字列とバイト配列

文字またはバイト配列が含まれた列を指定するには、CHAR、VARCHAR2、RAW、LONG および LONG RAW の 5 つの Oracle 内部データ型を使用できます。

CHAR、VARCHAR2 および LONG 列は、通常、文字データを保持します。RAW と LONG RAW は、ビットマップ化された画像のピクセル値など、文字として解釈されないバイトを保持します。文字データは、ネットワーク間のゲートウェイを介して渡すと、変形する可能性があります。たとえば、異なる言語（単一の文字が異なるバイト数で表現される）を使用しているマシン間で文字データを渡すと、データの長さが大幅に変更される可能性があります。RAW データがこのように変換されることはありません。

適切な Oracle 内部データ型を表内の各列に対して選択するのは、データベース設計者の役割です。文字データとバイト配列データを表現する方法、および OCCI プログラムの変数と Oracle データベース表間でデータを変換する方法には、様々な方法があることを理解しておく必要があります。

ユニバーサル ROWID（UROWID）

ユニバーサル ROWID（UROWID）は、Oracle 表、およびゲートウェイを使用してアクセスする DB2 表などの外部キー表の行に対する論理 ROWID と物理 ROWID を格納できるデータ型です。論理 ROWID の値は、索引構成表の行に対する主キー・ベースの論理識別子です。

UROWID データ型の列を使用するには、COMPATIBLE 初期化パラメータの値を 8.1 以上に設定する必要があります。

次の OCCI_SQLT 型は、ユニバーサル ROWID にバインドできます。

- OCCI_SQLT_CHR（VARCHAR2）
- OCCI_SQLT_VCS（VARCHAR）
- OCCI_SQLT_STR（NULL 終了文字列）
- OCCI_SQLT_LVC（LONG VARCHAR）
- OCCI_SQLT_AFC（CHAR）
- OCCI_SQLT_AVC（CHARZ）
- OCCI_SQLT_VST（文字列）
- OCCI_SQLT_RDD（ROWID 記述子）

外部データ型

ホスト OCCI アプリケーションと Oracle データベース・サーバー間の通信は、外部データ型を使用して行われます。具体的には、外部データ型が C++ データ型にマップされます。

表 4-2 は、Oracle 外部データ型、同等の C++ データ型（Oracle 内部データ型の通常の変換先）および対応する OCCI 型を示しています。

表 4-2 外部データ型、C++ データ型および OCCI 型

外部データ型	コード	C++ データ型	OCCI 型
BFILE	114	LNOCILobLocator	OCCI_SQLT_FILE
BLOB	113	LNOCILobLocator	OCCI_SQLT_BLOB
CHAR	96	char[n]	OCCI_SQLT_AFC
CLOB	112	LNOCILobLocator	OCCI_SQLT_CLOB
CHARZ	97	char[n+1]	OCCI_SQLT_RDD
DATE	12	char[7]	OCCI_SQLT_DAT
FLOAT	4	float、double	OCCIFLOAT
16 ビット符号付き INTEGER	3	signed short、signed int	OCCIINT
32 ビット符号付き INTEGER	3	signed int、signed long	OCCIINT
8 ビット符号付き INTEGER	3	signed char	OCCIINT
INTERVAL DAY TO SECOND	190	char[11]	OCCI_SQLT_INTERVAL_DS
INTERVAL YEAR TO MONTH	189	char[5]	OCCI_SQLT_INTERVAL_YM

n は、プログラム要件に従って（または、ROWID の場合はオペレーティング・システムに従って）可変長であることを示します。

表 4-2 外部データ型、C++ データ型および OCCI 型（続き）

外部データ型	コード	C++ データ型	OCCI 型
LONG	8	char[n]	OCCI_SQLT_LNG
LONG RAW	24	unsigned char[n]	OCCI_SQLT_LBI
LONG VARCHAR	94	char[n+sizeof(integer)]	OCCI_SQLT_LVC
LONG VARRAW	95	unsigned char[n+sizeof(integer)]	OCCI_SQLT_LVB
NAMED DATA TYPE	108	struct	OCCI_SQLT_NTY
NUMBER	2	unsigned char[21]	OCCI_SQLT_NUM
RAW	23	unsigned char[n]	OCCI_SQLT_BIN
REF	110	LNOCIRef	OCCI_SQLT_REF
ROWID	11	LNOCIRowid	OCCI_SQLT_RID
ROWID 記述子	104	LNOCIRowid	OCCI_SQLT_RDD
NULL で終了する STRING	5	char[n+1]	OCCI_SQLT_STR
TIMESTAMP	187	char[11]	OCCI_SQLT_TIMESTAMP
TIMESTAMP WITH LOCAL TIME ZONE	232	char[7]	OCCI_SQLT_TIMESTAMP_LTZ
TIMESTAMP WITH TIME ZONE	188	char[13]	OCCI_SQLT_TIMESTAMP_TZ
UNSIGNED INT	68	unsigned	OCCIUNSIGNED_INT
VARCHAR	9	char[n+sizeof(short integer)]	OCCI_SQLT_VCS
VARCHAR2	1	char[n]	OCCI_SQLT_CHR
VARNUM	6	char[22]	OCCI_SQLT_VNU
VARRAW	15	unsigned char[n+sizeof(short integer)]	OCCI_SQLT_VBI

次の外部データ型のほとんどは、OCCI の C++ クラスとして表現されます。詳細は、第 8 章「OCCI のクラスとメソッド」を参照してください。

OCCI BFILE	Bfile	OCCIBFILE
OCCI BLOB	Blob	OCCIBLOB
OCCI BOOL	bool	OCCIBOOL
OCCI BYTES	Bytes	OCCIBYTES

n は、プログラム要件に従って（または、ROWID の場合はオペレーティング・システムに従って）可変長であることを示します。

表 4-2 外部データ型、C++ データ型および OCCI 型（続き）

外部データ型	コード	C++ データ型	OCCI 型
OCCI ROWID		Bytes	OCCIROWID
OCCI CHAR		char	OCCICHAR
OCCI CLOB		Clob	OCCICLOB
OCCI DATE		Date	OCCIDATE
OCCI DOUBLE		double	OCCIDOUBLE
OCCI FLOAT		float	OCCIFLOAT
OCCI INTERVALDS		IntervalDS	OCCIINTERVALDS
OCCI INTERVALYM		IntervalYM	OCCIINTERVALYM
OCCI INT		int	OCCIINT
OCCI METEDATA		MetaData	OCCIMETADATA
OCCI NUMBER		Number	OCCINUMBER
OCCI REF		Ref	OCCIREF
OCCI REFANY		RefAny	OCCIREFANY
OCCI CURSOR		ResultSet	OCCICURSOR
OCCI STRING		STL string	OCCISTRING
OCCI VECTOR		STL vector	OCCIVECTOR
OCCI STREAM		Stream	OCCISTREAM
OCCI TIMESTAMP		Timestamp	OCCITIMESTAMP
OCCI UNSIGNED INT		unsigned int	OCCIUNSIGNED_INT
OCCI POBJECT		ユーザー定義型（Object Type Translator によって生成）	OCCIPOBJECT

n は、プログラム要件に従って（または、ROWID の場合はオペレーティング・システムに従って）可変長であることを示します。

注意： TIMESTAMP データ型と TIMESTAMP WITH TIME ZONE データ型は、総称して日時データ型と呼ばれています。INTERVAL YEAR TO MONTH データ型と INTERVAL DAY TO SECOND データ型は、総称して間隔データ型と呼ばれています。

次の Statement クラスのメソッドで型を使用する場合には、注意が必要です。

- registerOutParam: occiCommon.h ファイルの OCCI*xxx* フォーム（OCCIDOUBLE、OCCICURSOR など）の型のみ使用できます。ただし、いくつか例外があります。OCCIANYDATA、OCCIMETADATA、OCCISTREAM および OCCIBOOL は使用できません。
- setDataBuffer() と setDataBufferArray: occiCommon.h ファイルの OCCI_SQLT_*xxx* フォーム（OCCI_SQLT_INT など）の型のみ使用できます。

次の ResultSet クラスのメソッドで型を使用する場合には、注意が必要です。

- setDataBuffer() と setDataBufferArray: occiCommon.h ファイルの OCCI_SQLT_*xxx* フォーム（OCCI_SQLT_INT など）の型のみ使用できます。

外部データ型の説明

この項では、各外部データ型について説明します。

BFILE

BFILE 外部データ型を使用すると、データベース・サーバーのファイル・システムにあるファイルに読取り専用バイト・ストリームによるアクセスが可能となります。BFILE は、オペレーティング・システム・ファイル内のデータベース表領域外に格納されているバイナリ・データのラージ・オブジェクトです。このファイルでは、参照セマンティクスが使用されます。Oracle Server から BFILE にアクセスする場合は、基礎となるサーバーのオペレーティング・システムが、ストリームモードによるこれらのオペレーティング・システム・ファイルへのアクセスをサポートしている必要があります。

BLOB

BLOB 外部データ型には、非構造化バイナリ・ラージ・オブジェクトが格納されます。BLOB は、キャラクタ・セットのセマンティクスを伴わないビットストリームとみなすこともできます。BLOB には、最大 4GB のバイナリ・データを格納できます。

BLOB データ型には、完全なトランザクション・サポートがあります。OCCI を使用して行われた変更は、すべてトランザクションに反映されます。BLOB 値に対して行った操作は、コミットまたはロールバックできます。1 つのトランザクション内で変数の BLOB ロケータを保存し、次にそれを別のトランザクションまたはセッションで使用することはできません。

CHAR

CHAR 外部データ型は、最大 2000 文字の文字列です。文字列は、空白埋めの比較方法を使用して比較されます。

CHARZ

CHARZ 外部データ型は、入力時に文字列が NULL で終了し、出力時に Oracle で文字列の最後に NULL 終了文字が配置されることを除けば、CHAR データ型と同じです。NULL 終了文字は入出力時に文字列を区切るためのもので、表のデータの一部ではありません。

CLOB

CLOB 外部データ型には、固定幅または可変幅の文字データが格納されます。CLOB には、最大 4GB の文字データを格納できます。CLOB には、完全なトランザクション・サポートがあります。OCCI を使用して行われた変更は、すべてトランザクションに反映されます。CLOB 値に対して行った操作は、コミットまたはロールバックできます。1 つのトランザクション内で変数の CLOB ロケータを保存し、次にそれを別のトランザクションまたはセッションで使用することはできません。

DATE

DATE 外部データ型は、表 4-3 に示すように、7 バイトで構成された Oracle 内部日付 2 進形式を使用して、日付の値を更新、挿入または取り出すことができます。

表 4-3 DATE データ型の書式

	バイト 1	バイト 2	バイト 3	バイト 4	バイト 5	バイト 6	バイト 7
意味 :	世紀	年	月	日	時	分	秒
例（2000 年 6 月 1 日午後 3 時 17 分）:	120	100	6	1	16	18	1
例（BCE 4712 年 1 月 1 日）:	53	88	1	1	1	1	1

世紀と年を表すバイト（1 と 2）は、100 を加算した表記です。BCE（西暦紀元前）の日付は 99 以下です。西暦 0 以降の日付は 100 以上になります。0 以降の日付の場合、バイト 1 とバイト 2 の先頭の数字は、単に西暦であることを示します。

バイト 1 では、世紀の 2 番目と 3 番目の数字は、年（整数）を 100 で除算して計算されます。整数除算による小数部分は切り捨てられます。1992 年については、次の計算が行われます。

```
1992 / 100 = 19
```

バイト 1 の場合、119 は、20 世紀（1900 ～ 1999）を表します。120 の値は、21 世紀（2000 ～ 2099）を表すことになります。

バイト 2 では、年の 2 番目と 3 番目の数字が年のモジュロ 100 として計算されます。モジュロ除算による小数部分以外の数字は切り捨てられます。

```
1992 % 100 = 92
```

バイト 2 の場合、192 は、現在の世紀の 92 番目の年を表します。100 の値は、現在の世紀の 0 番目の年を表すことになります。

西暦 2000 年は、バイト 1 が 120、バイト 2 が 100 で表されます。

西暦 0 以前の年数の場合、世紀数と年数は、100 とその年数との差異で表されます。したがって、BCE 4712 年 1 月 1 日の場合は、100 - 47 = 53 で世紀が 53 となり、100 - 12 = 88 で年が 88 となります。

有効な日付は、BCE 4712 年 1 月 1 日から開始します。月バイトの範囲は 1 ～ 31、時間バイトの範囲は 1 ～ 24、および秒バイトの範囲は 1 ～ 60 となります。

注意： 日付に時刻が指定されていない場合、時刻はデフォルトで深夜（1、1、1）となります。

DATE 外部データ型を使用して、日付を 2 進形式で入力する場合、データベースでは一貫性または範囲のチェックが行われません。この形式によるデータは、入力前にすべて検証する必要があります。

注意： DATE 外部データ型の必要性はほとんどありません。ほとんどのプログラムでは、DD-MON-YYYY などの文字形式による日付が処理されるため、DATE の値を文字形式に変換する方が便利です。かわりに、Date データ型を使用することもできます。

DATE の列がプログラム内で文字列に変換されると、その列は使用中セッションのデフォルトの書式マスクで戻されるか、または INIT.ORA ファイルで指定したとおりに戻されます。

このデータ型は、C++ の Date データ型に対応する OCCI DATE とは異なります。

FLOAT

FLOAT 外部データ型は、小数部分のある数値を処理します。数値は、ホスト・システムの浮動小数点フォーマットで表されます。通常、長さは 4 バイトか 8 バイトです。

Oracle の数値の内部形式は 10 進数です。ほとんどの浮動小数点は 2 進数です。したがって、Oracle では、浮動小数点による表現より高い精度で数値が表されます。

INTEGER

INTEGER 外部データ型は、数値の変換に使用されます。外部 INTEGER は符号付きの 2 進数です。外部 INTEGER のサイズは、オペレーティング・システムによって異なります。Oracle から戻される数値が整数でない場合は、小数部分が切り捨てられ、エラーは戻りません。戻された数値がシステムの符号付き整数より大きくて表現できない場合は、変換時のオーバーフロー・エラーが戻されます。

注意： FLOAT と NUMBER 間の変換時に、丸めのエラーが発生する場合があります。問合せで FLOAT をバインド変数として使用すると、エラーが戻る場合があります。この問題を回避するには、FLOAT を文字列に変換し、操作では、OCCI 型の OCCI_SQLT_CHR または OCCI 型の OCCI_SQLT_STR を使用します。

INTERVAL DAY TO SECOND

INTERVAL DAY TO SECOND 外部データ型には、日数、時間数、分数および秒数に関して、2 つの日時の値の差異が格納されます。このデータ型は、次のように指定します。

```
INTERVAL DAY [(day_precision)]
   TO SECOND [(fractional_seconds_precision)]
```

この例では、次のプレースホルダが使用されています。

- *day_precision*: DAY 日時フィールドの桁数。指定できる値は 1 ～ 9 で、デフォルト値は 2 です。
- *fractional_seconds_precision*: SECOND 日時フィールドの小数部分の桁数。指定できる値は 0 ～ 9 で、デフォルト値は 6 です。

INTERVAL DAY TO SECOND リテラルをデフォルト以外の日付と秒の精度で指定するには、このリテラル内に精度を指定する必要があります。たとえば、次のように、100 日、10 時間、20 分、42 秒と 22/100 秒の間隔を指定することができます。

```
INTERVAL '100 10:20:42.22' DAY(3) TO SECOND(2)
```

また、INTERVAL DAY TO SECOND リテラルの省略フォームも使用できます。次に例を示します。

- INTERVAL '90' MINUTE は、INTERVAL '00 00:90:00.00' DAY TO SECOND(2) にマップされます。
- INTERVAL '30:30' HOUR TO MINUTE は、INTERVAL '00 30:30:00.00' DAY TO SECOND(2) にマップされます。
- INTERVAL '30' SECOND(2,2) は、INTERVAL '00 00:00:30.00' DAY TO SECOND(2) にマップされます。

INTERVAL YEAR TO MONTH

INTERVAL YEAR TO MONTH 外部データ型には、YEAR と MONTH の日時フィールドを使用して、2 つの日時の値の差異が格納されます。INTERVAL YEAR TO MONTH は、次のように指定します。

```
INTERVAL YEAR [(year_precision)] TO MONTH
```

プレースホルダ *year_precision* は、YEAR 日時フィールドの桁数です。*year_precision* のデフォルト値は 2 です。INTERVAL YEAR TO MONTH リテラルをデフォルト以外の *year_precision* で指定するには、このリテラル内に精度を指定する必要があります。たとえば、次の INTERVAL YEAR TO MONTH リテラルは、123 年 2 か月の間隔を示しています。

```
INTERVAL '123-2' YEAR(3) TO MONTH
```

また、INTERVAL YEAR TO MONTH リテラルの省略フォームも使用できます。たとえば、次のとおりです。

- INTERVAL '10' MONTH は、INTERVAL '0-10' YEAR TO MONTH にマップされます。
- INTERVAL '123' YEAR(3) は、INTERVAL '123-0' YEAR(3) TO MONTH にマップされます。

LONG

LONG 外部データ型には、LONG データ型の列に 4001 バイト以上、2GB までの文字列が格納されます。この型の列は、長い文字列の格納と取出しにのみ使用されます。メソッド、式または WHERE 句では使用できません。LONG 列の値は、通常、文字列との間で変換されます。

LONG RAW

LONG RAW 外部データ型は、最大 2GB のデータを格納できることを除けば、RAW 外部データ型と同じです。

LONG VARCHAR

LONG VARCHAR 外部データ型には、Oracle の LONG 列へのデータおよびこの列からのデータが格納されます。先頭の 4 バイトには、項目の長さが記述されます。LONG VARCHAR の最大長は 2GB です。

LONG VARRAW

LONG VARRAW 外部データ型には、Oracle の LONG RAW 列へのデータおよびこの列からのデータが格納されます。長さは、先頭の 4 バイトに記述されます。最大長は 2GB です。

NCLOB

NCLOB 外部データ型は、CLOB の各国語キャラクタ・バージョンです。この外部データ型には、マルチバイトで固定幅の各国語キャラクタ・セットのキャラクタ（NCHAR）または可変幅キャラクタ・セットのデータが格納されます。NCLOB には、最大 4GB の文字テキスト・データを格納できます。

NCLOB には、完全なトランザクション・サポートがあります。OCCI を使用して行われた変更は、すべてトランザクションに反映されます。NCLOB 値に対して行った操作は、コミットまたはロールバックできます。1 つのトランザクション内で変数の NCLOB ロケータを保存し、次にそれを別のトランザクションまたはセッションで使用することはできません。

NCLOB 属性を指定してオブジェクトを作成することはできませんが、メソッドで NCLOB パラメータを指定することは可能です。

NUMBER

NUMBER を外部データ型として使用する必要はほとんどありません。NUMBER を使用すると、Oracle では 21 バイトの内部 2 進形式で数値を戻し、入力時にこの形式が必要になります。完全な情報が必要な場合は、次の説明を参照してください。

Oracle では、NUMBER データ型の値が可変長形式で格納されます。最初のバイトは指数であり、その後に 1 ～ 20 個の仮数バイトが続きます。指数バイトの上位ビットは符号ビットです。正数の場合はそのビットが設定され、負数の場合は消去されます。下位の 7 ビットは指数を表します。この指数はオフセット 65 で基本 100 の数字です。

10 進数の指数を計算するには、基本 100 の指数に 65 を加算し、数値が正数の場合はさらに 128 を加算します。数値が負数の場合も同様に計算しますが、後でビットが反転されます。たとえば、-5 の場合は、基本 100 の指数 = 62（0x3e）になります。したがって、10 進数の指数は、（~0x3e）-128 - 65 = 0xc1 -128 -65 = 193 -128 -65 = 0 になります。

各仮数バイトは基本 100 で範囲 1 ～ 100 の数字です。正数の場合は、それに 1 を加算した数字となります。したがって、値 5 に対する仮数は 6 です。負数の場合は、1 を加算するかわりに、その数字が 101 から減算されます。つまり、数値 -5 に対する仮数は、96（101 - 5）となります。負数には、102 が含まれているバイトがデータ・バイトの後に追加されます。ただし、20 個の仮数バイトを持つ負数には、102 のバイトは後続しません。仮数バイトは基本 100 であるため、各バイトは 2 桁の 10 進数値を表すことができます。仮数は正規化され、先行する 0（ゼロ）は格納されません。

仮数を表すために最大 20 個のデータ・バイトを使用できます。ただし、精度が保証されているのは 19 個のみです。19 個のデータ・バイトは、それぞれ基本 100 を表し、NUMBER 内部データ型に対して 38 桁の最大精度を提供します。

このデータ型は、C++ の Number データ型に対応する OCCI NUMBER とは異なります。

OCCI BFILE

関連項目：

- 第 8 章「OCCI のクラスとメソッド」の「Bfile クラス」(8-5 ページ)

OCCI BLOB

関連項目：

- 第 8 章「OCCI のクラスとメソッド」の「Blob クラス」(8-13 ページ)

OCCI BYTES

関連項目：

- 第 8 章「OCCI のクラスとメソッド」の「Bytes クラス」(8-24 ページ)

OCCI CLOB

関連項目：

- 第 8 章「OCCI のクラスとメソッド」の「Clob クラス」(8-27 ページ)

OCCI DATE

関連項目：

- 第 8 章「OCCI のクラスとメソッド」の「Date クラス」(8-51 ページ)

OCCI INTERVALDS

関連項目：

- 第 8 章「OCCI のクラスとメソッド」の「IntervalDS クラス」(8-70 ページ)

OCCI INTERVALYM

関連項目：

- 第 8 章「OCCI のクラスとメソッド」の「IntervalYM クラス」(8-82 ページ)

OCCI NUMBER

関連項目：

- 第 8 章「OCCI のクラスとメソッド」の「Number クラス」(8-100 ページ)

OCCI POBJECT

関連項目：

- 第 8 章「OCCI のクラスとメソッド」の「PObject クラス」(8-124 ページ)

OCCI REF

関連項目：

- 第 8 章「OCCI のクラスとメソッド」の「Ref クラス」(8-130 ページ)

OCCI REFANY

関連項目：

- 第 8 章「OCCI のクラスとメソッド」の「RefAny クラス」(8-137 ページ)

OCCI STRING

OCCI STRING 外部データ型は、STL 文字列に対応しています。

OCCI TIMESTAMP

関連項目：

- 第 8 章「OCCI のクラスとメソッド」の「Timestamp クラス」(8-213 ページ)

OCCI VECTOR

`OCCI VECTOR` 外部データ型は、ネストした表や `VARRAY` などのコレクションを表すために使用されます。`CREATE TYPE num_type as VARRAY OF NUMBER(10)` は、C++ アプリケーションでは `vector<int>`、`vector<Number>` として表すことができます。

RAW

RAW 外部データ型は、Oracle で解釈または処理されないバイナリ・データやバイト列で使用されます。たとえば、RAW は図形の文字列で使用できます。RAW 列の最大長は 2000 バイトです。

Oracle 表内の RAW データが文字列に変換される場合、データは 16 進コードで表されます。RAW データの各バイトは、それぞれの値を示す 00 ～ FF の範囲の 2 文字で表されます。RAW を使用して文字列を入力する場合は、16 進コードを使用する必要があります。

REF

REF 外部データ型は、名前付きデータ型に対する参照です。アプリケーションで使用するために REF を割り当てるには、REF へのポインタとして変数を宣言します。

ROWID

ROWID 外部データ型は、データベース表内の特定の行を識別します。多くの場合、ROWID は次の例にあるような文の発行による問合せで戻されます。

```
SELECT ROWID, var1, var2 FROM db
```

戻された ROWID は、後続の DELETE 文で使用できます。

UPDATE 操作で SELECT を実行する場合は、ROWID が暗黙的に戻されます。

STRING

STRING 外部データ型は、VARCHAR2 外部データ型（データ型コード 1）のように動作します。ただし、STRING 外部データ型は、NULL で終了する必要があります。

このデータ型は、C++ STL 文字列データ型に対応する OCCI STRING とは異なります。

TIMESTAMP

TIMESTAMP 外部データ型は、DATE データ型を拡張したデータ型です。この外部データ型には、DATE データ型の年、月および日に加えて、時間、分および秒の各値が格納されます。TIMESTAMP データ型は、次のように指定します。

```
TIMESTAMP [(fractional_seconds_precision)]
```

プレースホルダ *fractional_seconds_precision* では、必要に応じて、SECOND 日時フィールドの小数部分の桁数を 0 ～ 9 の範囲で指定します。デフォルト値は 6 です。たとえば、次のように TIMESTAMP(2) をリテラルとして指定します。

```
TIMESTAMP '1997-01-31 09:26:50.10'
```

このデータ型は、OCCI TIMESTAMP とは異なります。

TIMESTAMP WITH LOCAL TIME ZONE

TIMESTAMP WITH TIME ZONE（TSTZ）外部データ型は TIMESTAMP の改良型で、その値に明示的なタイム・ゾーン置換が含まれています。タイム・ゾーン置換は、現地時間と UTC（協定世界時、以前はグリニッジ標準時）との差異（時分による）です。TIMESTAMP WITH TIME ZONE データ型は、次のように指定します。

```
TIMESTAMP(fractional_seconds_precision) WITH TIME ZONE
```

プレースホルダ *fractional_seconds_precision* では、必要に応じて、SECOND 日時フィールドの小数部分の桁数を 0 ～ 9 の範囲で指定します。デフォルト値は 6 です。

2 つの TIMESTAMP WITH TIME ZONE の値が UTC で同じ時刻を表している場合は、データに格納されている TIME ZONE オフセットに関係なく同一とみなされます。

TIMESTAMP WITH TIME ZONE

TIMESTAMP WITH TIME ZONE 外部データ型は TIMESTAMP の改良型で、その値にタイム・ゾーン置換が含まれています。タイム・ゾーン置換は、現地時間と UTC（協定世界時、以前はグリニッジ標準時）との差異（時分による）です。TIMESTAMP WITH TIME ZONE データ型は、次のように指定します。

```
TIMESTAMP [(fractional_seconds_precision)] WITH TIME ZONE
```

プレースホルダ *fractional_seconds_precision* では、必要に応じて、SECOND 日時フィールドの小数部分の桁数を 0 ～ 9 の範囲で指定します。デフォルト値は 6 です。たとえば、次のように、TIMESTAMP(0) WITH TIME ZONE をリテラルとして指定します。

```
TIMESTAMP '1997-01-31 09:26:50+02.00'
```

UNSIGNED INT

UNSIGNED INT 外部データ型は、符号なし 2 進整数に対して使用されます。バイト単位のサイズは、オペレーティング・システムによって異なります。ワード内のバイトの順序は、ホスト・システムのアーキテクチャによって決まります。Oracle から出力される数値が整数でない場合は、小数部分が切り捨てられ、エラーは戻りません。戻される数値が、オペレーティング・システムの符号なし整数より大きくて表現できない場合は、変換時のオーバーフロー・エラーが戻されます。

VARCHAR

VARCHAR 外部データ型には、可変長の文字列が格納されます。先頭の 2 バイトには文字列の長さが記述され、残りのバイトには実際の文字列が含まれます。バインドまたは定義コールで指定する文字列の長さには、この 2 バイトを含める必要があるため、VARCHAR 文字列の最大長は、65535 バイトではなく、65533 バイトになります。これより長い文字列を変換する場合は、LONG VARCHAR 外部データ型を使用します。

VARCHAR2

VARCHAR2 外部データ型は、最大 4000 バイトの可変長文字列です。

VARNUM

VARNUM 外部データ型は、最初のバイトに数値表現の長さが含まれることを除けば、NUMBER 外部データ型と同じです。この長さを示す値には、長さを記述するバイト自体は含まれません。最大長の VARNUM に備えて、22 バイトを確保してください。VARNUM の値をデータベースに送信するときは、長さを記述するバイトを設定する必要があります。

表 4-4 VARNUM の例

10 進数	長さバイト	指数バイト	仮数バイト	終了文字バイト
0	1	128	該当なし	該当なし
5	2	193	6	該当なし
-5	3	62	96	102
2767	3	194	28、68	該当なし
-2767	4	61	74、34	102
100000	2	195	11	該当なし
1234567	5	196	2、24、46、68	該当なし

VARRAW

VARRAW 外部データ型は、最初の 2 バイトにデータの長さを記述することを除けば、RAW 外部データ型と同じです。バインドまたは定義コールで指定する文字列の長さには、この 2 バイトが含まれている必要があります。したがって、送受信可能な VARRAW 文字列の最大長は、65535 バイトではなく、65533 バイトになります。これより長い文字列を変換する場合は、LONG VARRAW データ型を使用します。

データ変換

表 4-5 は、サポートされている、Oracle 内部データ型から外部データ型への変換、および外部データ型から内部データ列表現への変換を示しています。次の条件に注意してください。

- データベースに格納されている REF は、出力時に OCCI_SQLT_REF に変換されます。
- OCCI_SQLT_REF は、入力時に REF の内部表現に変換されます。
- データベースに格納されている名前付きデータ型は、出力時に OCCI_SQLT_NTY（アプリケーション内では C 構造体で表現されます）に変換されます。
- OCCI_SQLT_NTY（アプリケーション内では C 構造体で表現されます）は、入力時に、対応するデータ型の内部表現に変換されます。
- LOB と BFILE は、OCCI アプリケーション内では記述子で表現されるため、入出力時の変換はありません。

表 4-5 データ変換

		内部データ型							
		1	2	8	11	12	23	24	96
外部データ型		VARCHAR2	NUMBER	LONG	ROWID	DATE	RAW	LONG RAW	CHAR
1	VARCHAR	I/O	I/O	I/O	I/O[1]	I/O[2]	I/O[3]	I/O[3]	
2	NUMBER	I/O[4]	I/O	I					I/O[4]
3	INTEGER	I/O[4]	I/O	I					I/O[4]
4	FLOAT	I/O[4]	I/O	I					I/O[4]
5	STRING	I/O	I/O	I/O	I/O[1]	I/O[2]	I/O[3]	I/O[3、5]	I/O
6	VARNUM	I/O[4]	I/O	I					I/O[4]
7	DECIMAL	I/O[4]	I/O	I					I/O[4]
8	LONG	I/O	I/O	I/O	I/O[1]	I/O[2]	I/O[3]	I/O[3、5]	I/O

有効な変換

I = 入力時のみ

O = 出力時のみ

I/O = 入力時または出力時

注意：

1. 入力時、ホスト文字列は Oracle ROWID 形式であることが必要です。出力時、列値は Oracle ROWID 形式で戻されます。
2. 入力時、ホスト文字列は Oracle DATE 文字形式あることが必要です。出力時、列値は Oracle DATE 形式で戻されます。
3. 入力時、ホスト文字列は 16 進フォーマットであることが必要です。出力時、列値は 16 進フォーマットで戻されます。
4. 出力時、列値が有効な数値を表している必要があります。
5. 長さは 2000 文字以下であることが必要です。
6. 入力時、列値は 16 進フォーマットで格納されます。出力時、列値は 16 進フォーマットであることが必要です。

表 4-5 データ変換（続き）

		内部データ型							
		1	2	8	11	12	23	24	96
外部データ型		VARCHAR2	NUMBER	LONG	ROWID	DATE	RAW	LONG RAW	CHAR
9	VARCHAR	I/O	I/O	I/O	I/O[1]	I/O[2]	I/O[3]	I/O[3、5]	I/O
10	ROWID	I		I	I/O				I
12	DATE	I/O		I		I/O			I/O
15	VARRAW	I/O[6]		I[5、6]			I/O	I/O	I/O[6]
23	RAW	I/O[6]		I[5、6]			I/O	I/O	I/O[6]
24	LONG RAW	O[6]		I[5、6]			I/O	I/O	O[6]
68	UNSIGNED	I/O[4]	I/O	I					I/O[4]
94	LONG VARCHAR	I/O	I/O	I/O	I/O[1]	I/O[2]	I/O[3]	I/O[3、5]	I/O
95	LONG VARRAW	I/O[6]		I[5、6]			I/O	I/O	I/O[6]
96	CHAR	I/O	I/O	I/O	I/O[1]	I/O[2]	I/O[3]	I[3、5]	I/O
97	CHARZ	I/O	I/O	I/O	I/O[1]	I/O[2]	I/O[3]	I[3、5]	I/O
104	ROWID Desc.	I(1)			I/O				I(1)
	OCCI Number	I/O[4]	I/O	I					I/O[4]
	OCCI Bytes	I/O[6]		I[5、6]			I/O	I/O	I/O[6]
	OCCI Date	I/O		I		I/O			I/O
	OCCI Timestamp								
	STL string	I/O	I/O	I/O	I/O[1]	I/O[2]	I/O[3]	I/O[3]	

有効な変換

I = 入力時のみ

O = 出力時のみ

I/O = 入力時または出力時

注意：

1. 入力時、ホスト文字列は Oracle ROWID 形式であることが必要です。出力時、列値は Oracle ROWID 形式で戻されます。
2. 入力時、ホスト文字列は Oracle DATE 文字形式あることが必要です。出力時、列値は Oracle DATE 形式で戻されます。
3. 入力時、ホスト文字列は 16 進フォーマットであることが必要です。出力時、列値は 16 進フォーマットで戻されます。
4. 出力時、列値が有効な数値を表している必要があります。
5. 長さは 2000 文字以下であることが必要です。
6. 入力時、列値は 16 進フォーマットで格納されます。出力時、列値は 16 進フォーマットであることが必要です。

LOB データ型のデータ変換

表 4-6 LOB のデータ変換

	内部データ型	
外部データ型	**CLOB**	**BLOB**
VARCHAR	I/O	
CHAR	I/O	
LONG	I/O	
LONG VARCHAR	I/O	
STL STRING	I/O	
RAW		I/O
VARRAW		I/O
LONG RAW		I/O
LONG VARRAW		I/O
OCCI BYTES		I/O

日付、タイムスタンプおよび間隔データ型のデータ変換

文字データ型のいずれかを、日時列または間隔列のフェッチ操作または挿入操作で使用するホスト変数に対して使用することもできます。Oracle では、文字データ型と日時 / 間隔データ型との間の変換が行われます。

日付、タイムスタンプおよび間隔データ型のデータ変換

	内部型						
外部型	**VARCHAR、CHAR**	**DATE**	**TS**	**TSTZ**	**TSLTZ**	**INTERVAL YEAR TO MONTH**	**INTERVAL DAY TO SECOND**
VARCHAR2、CHAR	I/O	I/O	I/O	I/O	I/O	I/O	I/O
STL STRING	I/O	I/O	I/O	I/O	I/O	I/O	I/O
DATE	I/O	I/O	I/O	I/O	I/O	-	-
OCCI DATE	I/O	I/O	I/O	I/O	I/O	-	-
ANSI DATE	I/O	I/O	I/O	I/O	I/O	-	-
TIMESTAMP (TS)	I/O	I/O	I/O	I/O	I/O	-	-

日付、タイムスタンプおよび間隔データ型のデータ変換（続き）

	内部型						
外部型	VARCHAR、CHAR	DATE	TS	TSTZ	TSLTZ	INTERVAL YEAR TO MONTH	INTERVAL DAY TO SECOND
OCCI TIMESTAMP	I/O	I/O	I/O	I/O	I/O	-	-
TIMESTAMP WITH TIME ZONE (TSTZ)	I/O	I/O	I/O	I/O	I/O	-	-
TIMESTAMP WITH LOCAL TIME ZONE (TSLTZ)	I/O	I/O	I/O	I/O	I/O		
INTERVAL YEAR TO MONTH	I/O	-	-	-	-	I/O	-
OCCI INTERVALYM	I/O	-	-	-	-	I/O	-
INTERVAL DAY TO SECOND	I/O	-	-	-	-	-	I/O
OCCI INTERVALDS	I/O	-	-	-	-	-	I/O

注意 : タイム・ゾーンがあるソースをタイム・ゾーンのないターゲットに割り当てると、ソースのタイム・ゾーン部分は無視されます。タイム・ゾーンのないソースをタイム・ゾーンがあるターゲットに割り当てると、ターゲットのタイム・ゾーンには、セッションのデフォルト・タイム・ゾーンが設定されます。

(0) Oracle DATE を TIMESTAMP に割り当てると、DATE の TIME 部分が TIMESTAMP にコピーされます。TIMESTAMP を Oracle DATE に割り当てると、割当て後の DATE の TIME 部分には 0（ゼロ）が設定されます。この 0（ゼロ）設定は、Oracle DATE から ANSI 準拠の DATETIME データ型への移行を容易にするために行われます。

(1) ANSI DATE を Oracle DATE または TIMESTAMP に割り当てると、Oracle DATE の TIME 部分と TIMESTAMP には 0（ゼロ）が設定されます。Oracle DATE または TIMESTAMP を ANSI DATE に割り当てると、TIME 部分は無視されます。

(2) DATETIME を文字列に割り当てると、DATETIME は、セッションのデフォルトの DATETIME 形式によって変換されます。文字列を DATETIME に割り当てる場合は、その文字列にセッションのデフォルトの DATETIME 形式に基づく有効な DATETIME 値が含まれていることが必要です。

(3) 文字列を INTERVAL に割り当てる場合、その文字列が有効な INTERVAL 文字形式であることが必要です。

1. TSLTZ から CHAR、DATE、TIMESTAMP および TSTZ への変換時には、セッションのタイム・ゾーンに値が調整されます。
2. CHAR、DATE および TIMESTAMP から TSLTZ への変換時には、そのセッションのタイム・ゾーンがメモリーに格納されます。
3. TSLTZ を ANSI DATE に割り当てると、時間部分が 0（ゼロ）になります。
4. TSTZ からの変換時には、タイム・スタンプが存在しているタイム・ゾーンがメモリーに格納されます。
5. 文字列を間隔に割り当てる場合、その文字列は有効な間隔の文字フォーマットである必要があります。

5

LOB の概要

この章は、次の項目で構成されています。

LOB の概要

Oracle C++ Call Interface（OCCI）には、ラージ・オブジェクト（Large Object: LOB）の操作を実行するためのクラスとメソッドが含まれています。LOB の種類は、データベースとの位置関係に従って、内部 LOB または外部 LOB のいずれかになります。

内部 LOB（BLOB、CLOB および NCLOB）

内部 LOB は、領域の最適化と効率的なアクセスを可能にする方法でデータベースの表領域内に格納されます。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・モデルに関与します。トランザクションやメディアの障害の際は、内部 LOB をリカバリでき、内部 LOB 値を変更した際は、その変更をコミットまたはロールバックできます。つまり、データベース・オブジェクトの使用に関連するすべての ACID[1] プロパティが内部 LOB の使用にも関連しています。

内部 LOB データ型

内部 LOB のインスタンス定義には、3 つの SQL データ型があります。

- **BLOB:** 値が非構造化バイナリ・データ（ロー・データ）で構成されている LOB
- **CLOB:** 値が、Oracle データベースに定義されたデータベース・キャラクタ・セットに対応している文字データで構成されている LOB
- **NCLOB:** 値が、Oracle データベースに定義された各国語キャラクタ・セットに対応している文字データで構成されている LOB

コピー・セマンティクス

内部 LOB は、永続的または一時的にコピー・セマンティクスを使用します。LOB を挿入するか、同じ表にある別の行の LOB で更新すると、その LOB 値がコピーされるため、各行にはその LOB 値のコピーが設定されます。

コピー・セマンティクスを使用すると、LOB ロケータのみではなく、LOB ロケータと LOB 値の両方が結果的にコピーされます。

内部 LOB は、**永続** LOB と**一時** LOB に分類されます。

[1] ACID = Access Control Information Directory の略称。これは、どのディレクトリ・データに対して、誰がどのタイプのアクセス権限を持っているかを判断する属性です。この属性には、構造型アクセス項目とコンテンツ・アクセス項目に関する 1 組の規則が含まれています。詳細は、『Oracle Internet Directory 管理者ガイド』を参照してください。

外部 LOB（BFILE）

外部 LOB（BFILE）は、オペレーティング・システム・ファイル内のデータベース表領域外に格納されているバイナリ・データのラージ・オブジェクトです。このファイルでは、参照セマンティクスが使用されます。BFILE は、ハード・ディスクなどの従来の 2 次記憶装置以外に、CD-ROM、PhotoCD、DVD などの 3 次ブロック記憶装置にも配置することができます。

BFILE データ型によって、読取り専用バイト・ストリームは、データベース・サーバーのファイル・システムにある大量のファイルにアクセスできます。

Oracle から BFILE にアクセスする場合は、基礎となるサーバーのオペレーティング・システムが、ストリームモードによるこれらのオペレーティング・システム・ファイルへのアクセスをサポートしている必要があります。

注意：

- 外部 LOB は、トランザクションに関与しません。整合性と永続性に関するサポートは、オペレーティング・システムが管理する基礎となるファイル・システムで準備する必要があります。
- 1 つのデバイスには、1 つの外部 LOB が存在している必要があります。たとえば、外部 LOB がディスク配列全体にわたってストライプ化されることはありません。

外部 LOB データ型

BFILE と呼ばれる外部 LOB のインスタンスを宣言するための SQL データ型があります。BFILE は、値がバイナリ・データ（ロー・データ）で構成されている LOB です。この値は、サーバー側オペレーティング・システム・ファイルのデータベース表領域外に格納されます。

参照セマンティクス

外部 LOB（BFILE）では、参照セマンティクスが使用されます。表内の行の列に関連付けられた BFILE が別の列にコピーされると、BFILE ロケータのみがコピーされ、その BFILE が含まれている実際のオペレーティング・システム・ファイルはコピーされません。

LOB 値とロケータ

LOB 値のサイズによって、その格納場所が決定されます。LOB 値は、行データとともにインラインで格納されるか、行以外に格納されます。

LOB ロケータは、行データとともにインラインで格納され、LOB 値の格納場所を示します。

LOB 値のインライン記憶域

LOB に格納されているデータは、LOB 値と呼ばれます。内部 LOB 値は、別の行データとともにインラインで格納される場合とされない場合があります。DISABLE STORAGE IN ROW を設定せず、内部 LOB 値がおよそ 4,000 バイト未満の場合、その値はインラインで格納されます。およそ 4,000 バイト以上の場合は、行以外に格納されます。LOB は、ラージ・オブジェクトを目的にしているため、インライン記憶域は、使用しているアプリケーションで大小の LOB が混合している場合に、有意義なものとなります。

LOB 値は、およそ 4,000 バイトを超えると、行から自動的に移動します。

LOB ロケータ

LOB ロケータは、内部 LOB 値の格納場所にかかわりなく、行に格納されます。LOB ロケータは、LOB 値の実際の位置へのポインタとみなすことができます。LOB ロケータは、内部 LOB に対するロケータであり、一方、BFILE ロケータは、外部 LOB に対するロケータです。

- **内部 LOB ロケータ**: 内部 LOB の場合、その LOB 列には、データベース表領域に格納されている LOB に対するロケータが格納されます。内部 LOB 列と指定された行の属性には、それぞれ一意の LOB ロケータおよびデータベース表領域に格納されている LOB 値の個別のコピーがあります。
- **外部 LOB ロケータ**: 外部 LOB（BFILE）の場合、その LOB 列には、BFILE が含まれている外部オペレーティング・システム・ファイルに対するロケータが格納されます。外部 LOB 列と指定された行の属性には、それぞれ一意の BFILE ロケータがあります。ただし、2 つの異なる行に、同じオペレーティング・システム・ファイルを指し示す BFILE ロケータが含まれる場合があります。

LOB のクラスとメソッド

表 5-1 は、LOB 操作で使用可能なクラスとメソッドを示しています。

表 5-1 OCCI LOB のクラスとメソッド

クラス	メソッド	用途
「Bfile クラス」（8-5 ページ）	close()	外部 LOB（BFILE）のデータにアクセスする。
	closeStream()	
	fileExists()	
	getDirAlias()	
	getFileName()	
	getStream()	
	isInitialized()	
	isNull()	
	isOpen()	
	length()	
	open()	
	operator=()	
	operator==()	
	operator!=()	
	read()	
	setName()	
	setNull()	

表 5-1 OCCI LOB のクラスとメソッド（続き）

クラス	メソッド	用途
「Blob クラス」（8-13 ページ）	append()	内部 LOB（BLOB）値とロケータを操作する。
	close()	
	closeStream()	
	copy()	
	getChunkSize()	
	getStream()	
	isInitialized()	
	isNull()	
	isOpen()	
	length()	
	open()	
	operator=()	
	operator==()	
	operator!= ()	
	read()	
	setEmpty()	
	setNull()	
	trim()	
	write()	
	writeChunk()	

表 5-1 OCCI LOB のクラスとメソッド（続き）

クラス	メソッド	用途
「Clob クラス」（8-27 ページ）	append()	内部 LOB（CLOB および NCLOB）値とロケータを操作する。
	close()	
	closeStream()	
	copy()	
	getCharSetForm()	
	getCharSetId()	
	getChunkSize()	
	getStream()	
	isInitialized()	
	isNull()	
	isOpen()	
	length()	
	open()	
	operator=()	
	operator==()	
	operator!=()	
	read()	
	setCharSetForm()	
	setNull()	
	trim()	
	write()	
	writeChunk()	

関連項目： それぞれのクラスとメソッドの詳細は、第 8 章「OCCI のクラスとメソッド」を参照してください。

LOB の作成

内部 LOB または外部 LOB を作成するには、データベースで新規 LOB ロケータを初期化します。作成する LOB の型に基づいて、次のクラスのいずれかを使用します。

- Bfile
- Blob
- Clob

次に、関連する適切なメソッドを使用すると、LOB 値にアクセスできます。

注意： 内部 LOB 列または属性を変更（書込み、コピー、切捨てなど）する場合は、その LOB が含まれている行をロックする必要があります。ロック方法の 1 つとして、操作を実行する前に SELECT...FOR UPDATE 文を使用してロケータを選択する方法があります。

LOB 書込みコマンドを正常終了するには、トランザクションをオープンしておく必要があります。したがって、データの書込み前にトランザクションをコミットする場合は、COMMIT によって、トランザクションがクローズされるので、（SELECT ...FOR UPDATE 文などを再発行して）その行をロックしなおす必要があります。

LOB のオープンとクローズ

OCCI には、内部 LOB と外部 LOB を明示的にオープンし、クローズするメソッドが用意されています。

- Bfile::open() および Bfile::close()
- Blob::open() および Blob::close()
- Clob::open() および Clob::close()

特定の LOB がオープンしているかどうかをチェックするためのメソッドも、用意されています。

- Bfile::isOpen()
- Blob::isOpen()
- Clob::isOpen()

これらのメソッドによって、OCCI アプリケーションでは一連の LOB 操作の開始と終了をマークできるため、LOB のクローズ時に特定の処理（索引の更新など）を実行できます。

注意： 内部 LOB の場合のオープンに関する概念は、LOB ロケータではなく、LOB に関連付けられています。LOB ロケータには、そのロケータが参照する LOB がオープンしているかどうかの情報は格納されていません。このため、複数のロケータが同じオープン状態の LOB を指し示すことが可能です。ただし、外部 LOB の場合のオープンに関する概念は、特定の外部 LOB ロケータに関連付けられています。したがって、異なる外部 LOB ロケータを使用して、同じ BFILE に対して複数のオープン状態を実現できます。

注意： LOB 操作が open() メソッドおよび close() メソッドのコールで囲まれていない場合は、LOB の変更時に、LOB の拡張可能な索引がすべて更新されます。このため、索引は常に有効であり、いつでも使用できます。open() メソッドのコールと close() メソッドのコールで囲まれている LOB が変更された場合は、個々の LOB の変更に対するトリガーは起動されません。トリガーは、close() メソッドのコールの後にのみ起動され、索引はそのときまで更新されません。したがって、索引は open() メソッドのコールから close() メソッドのコールまでの間は無効です。

アプリケーションで、LOB 操作を open() メソッドのコールと close() メソッドのコールで囲んでいない場合は、LOB が変更されるたびに暗黙的な LOB のオープンとクローズが行われます。これによって、その LOB の変更に関連付けられているトリガーが起動します。

LOB のオープンとクローズに関する制限事項

LOB のオープンとクローズに関するメカニズムには、次の制限があります。

- アプリケーションでは、トランザクションをコミットする前に、以前にオープンした LOB をすべてクローズする必要があります。クローズしないと、エラーになります。トランザクションがロールバックされた場合は、オープンしているすべての LOB が、行った変更内容とともに廃棄されます（LOB はクローズされません）。したがって、関連付けられているトリガーは起動されません。
- 内部 LOB のオープン数に制限はありませんが、FILE のオープン数には制限があります。すでにオープンしているロケータを別のロケータに割り当てても、新規 LOB のオープンとしてはカウントされません。
- 同じトランザクション内で、異なるロケータまたは同じロケータを使用して、同じ内部 LOB を 2 回オープンまたはクローズするとエラーになります。
- オープンしていない LOB をクローズするとエラーになります。

注意： オープンしている LOB 値をクローズする必要があるトランザクションの定義は、次のいずれかです。

- SET TRANSACTION と COMMIT の間
- DATA MODIFYING DML または SELECT ...FOR UPDATE と COMMIT の間
- 自律型トランザクション・ブロック内

トランザクションがない場合にオープンしている LOB は、セッションの終了前にクローズする必要があります。セッション終了時にオープンしている LOB がある場合、そのオープン状態は廃棄され、拡張可能な索引のトリガーは起動されません。

LOB の読込みと書込み

OCCI には、LOB の読込みおよび書込み用のメソッドが用意されています。非ストリーム読込みと非ストリーム書込みの場合は、次のメソッドを使用します。

- Bfile::read()
- Blob::read() および Blob::write()
- Clob::read() および Clob::write()

ストリーム読込みとストリーム書込みの場合は、次のメソッドを使用します。

- Bfile::getStream()
- Blob::getChunkSize()、Blob::getStream() および Blob::writeChunk()
- Clob::getChunkSize()、Clob::getStream() および Clob::writeChunk()

この項の後半では、ストリームおよび非ストリームの読込みと書込みの例を説明します。

非ストリーム読込み

次のコード例では、非ストリーム読込みを使用して、NULL でない内部 LOB（この例では BLOB）からデータを取得する方法を示しています。

```
ResultSet *rset=stmt->executeQuery("SELECT ad_composite FROM print_media
                                    WHERE product_id=6666");
while(rset->next())
{
   Blob blob=rset->getBlob(1);
   if(blob.isNull())
     cerr <<"Null Blob"<<endl;
   else
   {
```

```
      blob.open(OCCI_LOB_READONLY);

      const unsigned int BUFSIZE=100;
      char buffer[BUFSIZE];
      unsigned int readAmt=BUFSIZE;
      unsigned int offset=1;

      //reading readAmt bytes from offset 1
      blob.read(readAmt,buffer,BUFSIZE,offset);

      //process information in buffer
      .
      .
      .
      blob.close();
   }
}
stmt->closeResultSet(rset);
```

前述の例のように、ストリームを使用しないで BLOB からすべての情報を読み込むには、読込みオフセットと残りの読込み量を追跡して記録し、その値を read() メソッドに渡す必要があります。

次のコード例では、非ストリーム読込みを使用して、BFILE ロケータが NULL でない BFILE からデータを読み込む方法を示しています。

```
ResultSet *rset=stmt->executeQuery("SELECT ad_graphic FROM print_media
                                    WHERE product_id=6666");
while(rset->next())
{
   Bfile file=rset->getBfile(1);
   if(bfile.isNull())
      cerr <<"Null Bfile"<<endl;
   else
   {
      //display the directory alias and the file name of the BFILE
      cout <<"File Name:"<<bfile.getFileName()<<endl;
      cout <<"Directory Alias:"<<bfile.getDirAlias()<<endl;

      if(bfile.fileExists())
      {
         unsigned int length=bfile.length();
         char *buffer=new char[length];
         bfile.read(length, buffer, length, 1);
         //read all the contents of the BFILE into buffer, then process
         .
         .
         .
```

```
         delete[] buffer;
      }
      else
         cerr <<"File does not exist"<<endl;
   }
}
stmt->closeResultSet(rset);
```

非ストリーム書込み

次のコード例では、非ストリーム書込みを使用して、NULL でない内部 LOB（この例では BLOB）にデータを書き込む方法を示しています。

```
ResultSet *rset=stmt->executeQuery("SELECT ad_composite FROM print_media
                                    WHERE product_id=6666 FOR UPDATE");
while(rset->next())
{
   Blob blob=rset->getBlob(1);
   if(blob.isNull())
      cerr <<"Null Blob"<<endl;
   else
   {
      blob.open(OCCI_LOB_READWRITE);

      const unsigned int BUFSIZE=100;
      char buffer[BUFSIZE];
      unsigned int writeAmt=BUFSIZE;
      unsigned int offset=1;

      //writing writeAmt bytes from offset 1
      //contents of buffer are replaced after each writeChunk(),
      //typically with an fread()
      while(<fread "BUFSIZE" bytes into buffer succeeds>)
      {
         blob.writeChunk(writeAmt, buffer, BUFSIZE, offset);
         offset += writeAmt;
      }
      blob.writeChunk(<remaining amt>, buffer, BUFSIZE, offset);

      blob.close();
   }
}
stmt->closeResultSet(rset);
conn->commit();
```

前述のコード例では、writeChunk() メソッドが open() メソッドと close() メソッドで囲まれています。writeChunk() メソッドは、現在オープンしている LOB を操作し、読み込むチャンクごとにトリガーが起動されないようにします。この例では、writeChunk() メソッドのかわりに、write() メソッドを使用することができます。ただし、この write() メソッドは、LOB を暗黙的にオープンおよびクローズします。

ストリーム読込み

次のコード例では、ストリーム読込みを使用して、すでに作成されている内部 LOB（この例では BLOB）からデータを取得する方法を示しています。

```
ResultSet *rset=stmt->executeQuery("SELECT ad_composite FROM print_media
                                    WHERE product_id=6666");
while(rset->next())
{
   Blob blob=rset->getBlob(1);
   if(blob.isNull())
      cerr <<"Null Blob"<<endl;
   else
   {
      Stream *instream=blob.getStream(1,0);
      //reading from offset 1 to the end of the BLOB

      unsigned int size=blob.getChunkSize();
      char *buffer=new char[size];

      while((unsigned int length=instream->readBuffer(buffer,size))!=-1)
      {
         //process "length" bytes read into buffer
         .
         .
         .
      }
      delete[] buffer;
      blob.closeStream(instream);
   }
}
stmt->closeResultSet(rset);
```

ストリーム書込み

次のコード例では、ストリーム書込みを使用して、すでに作成されている内部 LOB（この例では BLOB）にデータを書き込む方法を示しています。

```
ResultSet *rset=stmt->executeQuery("SELECT ad_composite FROM print_media
                                    WHERE product_id=6666 FOR UPDATE");
while(rset->next())
{
   Blob blob=rset->getBlob(1);
   if(blob.isNull())
      cerr <<"Null Blob"<<endl;
   else
   {
      char buffer[BUFSIZE];
      Stream *outstream=blob.getStream(1,0);

      //writing from buffer beginning at offset 1 until
      //a writeLastBuffer() method is issued.
      //contents of buffer are replaced after each writeBuffer(),
      //typically with an fread()
      while(<fread "BUFSIZE" bytes into buffer succeeds>)
         ostream->writeBuffer(buffer,BUFSIZE);
      ostream->writeLastBuffer(buffer,<remaining amt>);
      blob.closeStream(outstream);
   }
}
stmt->closeResultSet(rset);
conn->commit();
```

読込み / 書込みのパフォーマンスの改善

内部 LOB の読込みと書込みのパフォーマンスは、次のいずれかのメソッドを使用して改善することができます。

- `getChunkSize()` メソッド
- `writeChunk()` メソッド

getChunkSize() メソッドを使用する場合

`Blob` クラスと `Clob` クラスの `getChunkSize()` メソッドの利点は、内部 LOB の読込みと書込みのパフォーマンスを改善することにあります。`getChunkSize()` メソッドは、使用可能なチャンク・サイズを、BLOB に対してはバイト数で、CLOB と NCLOB に対しては文字数で戻します。使用可能なチャンク・サイズの倍のサイズで、チャンクの境界から開始しているデータを使用して読込みまたは書込みが行われると、パフォーマンスが向上します。LOB 列のチャンク・サイズは、その LOB が含まれる表の作成時に指定できます。

getChunkSize() メソッドをコールすると、LOB の使用可能なチャンク・サイズが戻されます。アプリケーションでは、同じチャンク上で実行される複数の LOB 書込みコールを発行するかわりに、一連の書込み操作をチャンク全体が書き込まれるまでバッチ処理できます。

LOB の終わりまで読み込むには、4GB の容量で read() メソッドを使用します。4GB の容量を使用した read() メソッドによって、LOB の最後まで読込みが行われるため、最初に getLength() メソッドをコールすることによるラウンドトリップを回避できます。

注意： 可変幅の文字が格納されている LOB の場合、GetChunkSize() メソッドは、LOB チャンクに適した Unicode 文字の数を戻します。

---

writeChunk() メソッドを使用する場合

OCCI には、LOB の最後まで効率的にデータを書き込むためのショートカットが用意されています。Blob クラスと Clob クラスの writeAppend() メソッドによって、アプリケーションは、最初に必要な write() メソッドのコール開始ポイントを確認するための getLength() メソッドのコールを行わずに、LOB の最後までデータを追加できます。

LOB の更新

次のコード例は、内部 LOB（この例では CLOB）を更新して空にする方法を示しています。

```
Clob clob(conn);
clob.setEmpty();
stmt->setSQL("UPDATE print_media SET ad_composite = :1
              WHERE product_id=6666");
stmt->setClob(1, clob);
stmt->executeUpdate();
conn->commit();
```

次のコード例は、BFILE の更新方法を示しています。

```
Bfile bfile(conn);
bfile.setName("MEDIA_DIR", "img1.jpg");
stmt->setSQL("UPDATE print_media SET ad_graphic = :1
              WHERE product_id=6666");
stmt->setBfile(1, bfile);
stmt->executeUpdate();
conn->commit();
```

LOB 属性を持つオブジェクト

OCCI アプリケーションでは、オーバーロードされた new() 演算子を使用して、LOB 属性を持つ永続オブジェクトまたは一時オブジェクトを作成できます。デフォルトでは、すべての LOB 属性は、デフォルトのコンストラクタを使用して構造化され、NULL に初期化されます。

LOB 属性を持つ永続オブジェクト

OCCI を使用すると、LOB 属性を持つ永続オブジェクトを新規に作成できます。作成する手順は、次のとおりです。

1. LOB 属性を持つ永続オブジェクトを作成します。

   ```
   Person *p=new(conn,"PERSON_TAB")Person();
   ```

2. Blob オブジェクトを初期化して空にします。

   ```
   p->imgBlob = Blob(conn);
   p->imgBlob.setEmpty();
   ```

   必要に応じて、対応するメソッド（setxxx メソッドおよび getxxx メソッド）を Person オブジェクトに対して使用し、同じ結果を得ます。

3. Blob オブジェクトを使用済みとしてマークします。

   ```
   p->markModified();
   ```

4. オブジェクトをフラッシュします。

   ```
   p->flush();
   ```

5. オブジェクトに対する REF を取得してから、そのオブジェクトを再確保します。これによって、オブジェクトのリフレッシュされたバージョンをデータベースから取り出し、初期化された LOB を取得します。

   ```
   Ref<Person> r = p->getRef();
   delete p;
   p = r.ptr();
   ```

6. データを書き込みます。

   ```
   p->imgBlob.write( ... );
   ```

BFILE 属性を持つ永続オブジェクトを作成する手順は、次のとおりです。

1. LOB 属性を持つ永続オブジェクトを作成します。

```
Person *p=new(conn,"PERSON_TAB")Person();
```

2. Bfile オブジェクトを初期化して空にします。

```
p->imgBfile = Bfile(conn);
p->setName(<Directory Alias>,<File Name>);
```

3. オブジェクトを使用済みとしてマークします。

```
p->markModified();
```

4. オブジェクトをフラッシュします。

```
p->flush();
```

5. データを読み込みます。

```
p->imgBfile.read( ... );
```

LOB 属性を持つ一時オブジェクト

アプリケーションでは、オーバーロードされた new() メソッドをコールして、内部 LOB（BLOB、CLOB、NCLOB）属性を持つ一時オブジェクトを作成できます。ただし、一時 LOB は現在サポートされていないため、この LOB 属性に関する操作（読込みや書込みなど）は実行できません。オーバーロードされた new() メソッドをコールして、一時的な内部 LOB 型を作成してもエラーになりませんが、アプリケーションでは、この一時 LOB による LOB 操作を使用できません。

ただし、アプリケーションでは、FILE 属性を持つ一時オブジェクトを作成し、この FILE 属性を使用して、サーバーのファイル・システムに格納されているファイルからデータを読み込むことはできます。また、アプリケーションでは、オーバーロードされた new() メソッドをコールして一時 FILE を作成し、その FILE を使用して、サーバーのファイルからデータを読み込むこともできます。

6
メタデータ

この章では、結果セットまたはデータベースに関するメタデータの取出し方法を説明します。

この章は、次の項目で構成されています。

- メタデータの概要
- データベース・メタデータの記述
- 属性の参照

メタデータの概要

データベース・オブジェクトには、そのデータベース・オブジェクトを記述する様々な属性があります。オブジェクトに対する DESCRIBE 操作を実行すると、特定のスキーマ・オブジェクトに関する情報を取得します。結果は、Metadata クラスのオブジェクトとしてアクセスできます。このクラスでは、クラス・メソッドを使用して、オブジェクトの実際値を取得できます。これを実行するには、オブジェクトの属性を、Metadata クラスの様々なメソッドに引数として渡します。

データベース全体、ResultSet クラスに含まれている列の型とプロパティ、または次のスキーマ・オブジェクトとサブスキーマ・オブジェクトのいずれかに対して、DESCRIBE 操作を明示的に実行できます。

- 表
- ビュー
- プロシージャ
- ファンクション
- パッケージ
- 型
- 型属性
- 型メソッド
- コレクション
- シノニム
- 順序
- 列
- 引数
- 結果
- リスト

検索する属性の型を指定する必要があります。MetaData クラスの getAttributeCount、getAttributeId および getAttributeType の各メソッドを使用すると、使用可能な各属性をスキャンできます。

すべての DESCRIBE 情報は、その情報に対する最後の参照が削除されるまでキャッシュされます。この方法によって、すでに開放されている DESCRIBE 情報に、ユーザーが間違えてアクセスすることがなくなります。

メタデータが明示的な記述の場合は、Connection クラスの getMetaData メソッドをコールしてメタデータを取得します。また、結果セット列のメタデータを取得するには、ResultSet クラスの getColumnListMetaData メソッドをコールします。この 2 つのメ

ソッドでは、詳細な情報を持つ MetaData オブジェクトが戻されます。MetaData クラスには、この情報にアクセスするための getxxx メソッドが用意されています。

型と属性に関する注意

DESCRIBE 操作の実行時には、次の問題に注意してください。

- ATTR_TYPECODE では、CREATE TYPE 文で型を新規作成したときに指定した型を表す型コードが戻されます。これらの型コードは、OCCITypeCode 列挙型で、OCCI_TYPECODE 定数によって表されます。

  注意： 内部 PL/SQL 型（ブール、索引付きの表）はサポートされていません。

- ATTR_DATA_TYPE では、データベース列のデータ型を表す型が戻されます。これらの値は、OCCIType 列挙型です。たとえば、LONG 型では、OCCI_SQLT_LNG 型が戻されます。

データベース・メタデータの記述

データベース・メタデータの記述は、明示的な DESCRIBE 操作と同じになります。記述対象のオブジェクトは、スキーマ内のオブジェクトである必要があります。型の記述では、getMetaData メソッドを接続からコールし、オブジェクトの名前または RefAny オブジェクトを渡します。このためには、OBJECT モードで環境を初期化する必要があります。getMetaData メソッドでは、MetaData 型のオブジェクトが戻されます。MetaData 型の各オブジェクトには、記述ツリーの一部である属性のリストがあります。この記述ツリーを再帰的に横断すると、詳細な情報が含まれているサブツリーを指し示すことができます。オブジェクトに関する詳細情報は、getxxx メソッドをコールして取得できます。

データベースとそのオブジェクトを再帰的に記述するブラウザの構築が必要な場合は、（データベースも含めて）そのデータベースにある各オブジェクトの属性数、属性 ID のリストおよび属性型のリストに関する情報にアクセスできます。この情報を使用すると、記述ツリーを最上位ノード（データベース）から表の列、型の属性、プロシージャやファンクションのパラメータに至るまで、再帰的にたどることができます。

たとえば、表とその内容を記述するという典型的なケースを想定します。getMetaData メソッドを接続からコールし、記述する表の名前を渡します。戻された MetaData オブジェクトには、表に関する情報が含まれています。記述するオブジェクト（表、列、型、コレクション、ファンクション、プロシージャなど）の型はわかっているため、表 6-1 に示すような属性リストを取得できます。対応する get*() メソッドをコールすると、表に指定されている型の変数内に値を取り出すことができます。

表 6-1 属性のグループ化

属性の型	説明
「パラメータの属性」(6-10 ページ)	すべての要素に所属する属性
「表とビューの属性」(6-11 ページ)	表とビューに所属する属性
「プロシージャ、ファンクションおよびサブプログラムの属性」(6-12 ページ)	プロシージャ、ファンクションおよびパッケージ・サブプログラムに所属する属性
「パッケージの属性」(6-12 ページ)	パッケージに所属する属性
「型の属性」(6-13 ページ)	型に所属する属性
「型属性の属性」(6-15 ページ)	型属性に所属する属性
「型メソッドの属性」(6-16 ページ)	型メソッドに所属する属性
「コレクションの属性」(6-17 ページ)	コレクション型に所属する属性
「シノニムの属性」(6-19 ページ)	シノニムに所属する属性
「順序の属性」(6-19 ページ)	順序に所属する属性
「列の属性」(6-20 ページ)	表またはビューの列に所属する属性
「引数および結果の属性」(6-21 ページ)	引数 / 結果に所属する属性
「リストの属性」(6-23 ページ)	リストの型を示す属性
「スキーマの属性」(6-23 ページ)	スキーマ固有の属性
「データベースの属性」(6-24 ページ)	データベース固有の属性

メタデータのコード例

この項では、次のメタデータを取得するためのコード例を示します。

- Connection メタデータ
- ResultSet メタデータ

Connection メタデータのコード例

次のコード例は、単純なデータベース表の属性に関するメタデータの取得方法を示しています。

```
/* Create an environment and a connection to the HR database */
.
.
.
/* Call the getMetaData method on the Connection object obtainedv*/
MetaData emptab_metaData = connection->getMetaData("EMPLOYEES",
                                                   MetaData::PTYPE_TABLE);
/* Now that you have the metadata information on the EMPLOYEES table,
   call the getxxx methods using the appropriate attributes
*/
/* Call getString */
cout<<"Schema:"<<(emptab_metaData.getString(MetaData::ATTR_OBJ_SCHEMA))<<endl;

if(emptab_metaData.getInt(emptab_metaData::ATTR_PTYPE)==MetaData::PTYPE_TABLE)
    cout<<"EMPLOYEES is a table"<<endl;
else
    cout<<"EMPLOYEES is not a table"<<endl;

/* Call getInt to get the number of columns in the table */
int columnCount=emptab_metaData.getInt(MetaData::ATTR_NUM_COLS);
cout<<"Number of Columns:"<<columnCount<<endl;

/* Call getTimestamp to get the timestamp of the table object */
Timestamp tstamp = emptab_metaData.getTimestamp(MetaData::ATTR_TIMESTAMP);
/* Now that you have the value of the attribute as a Timestamp object,
   you can call methods to obtain the components of the timestamp */
int year;
unsigned int month, day;
tstamp.getData(year, month, day);

/* Call getVector for attributes of list type,
for example ATTR_LIST_COLUMNS
*/
vector<MetaData>listOfColumns;
listOfColumns=emptab_metaData.getVector(MetaData::ATTR_LIST_COLUMNS);
```

```
/* Each of the list elements represents a column metadata,
   so now you can access the column attributes
*/
for (int i=0;i<listOfColumns.size();i++
{
   MetaData columnObj=listOfColumns[i];
   cout<<"Column Name:"<<(columnObj.getString(MetaData::ATTR_NAME))<<endl;
   cout<<"Data Type:"<<(columnObj.getInt(MetaData::ATTR_DATA_TYPE))<<endl;
   .
   .
   .
   /* and so on to obtain metadata on other column specific attributes */
}
```

次のコード例は、ユーザー定義型が含まれた列を持つデータベース表に関するメタデータの取得方法を示しています。

```
/* Create an environment and a connection to the HR database */
.
.
.
/* Call the getMetaData method on the Connection object obtained */
MetaData custtab_metaData = connection->getMetaData("CUSTOMERS",
   MetaData::PTYPE_TABLE);

/* Now that you have the metadata information on the CUSTOMERS table,
   call the getxxx methods using the appropriate attributes
*/
/* Call getString */
cout<<"Schema:"<<(custtab_metaData.getString(MetaData::ATTR_OBJ_SCHEMA))<<endl;

if(custtab_metaData.getInt(custtab_metaData::ATTR_PTYPE)==MetaData::PTYPE_TABLE)
   cout<<"CUSTOMERS is a table"<<endl;
else
   cout<<"CUSTOMERS is not a table"<<endl;

/* Call getVector to obtain a list of columns in the CUSTOMERS table */
vector<MetaData>listOfColumns;
listOfColumns=custtab_metaData.getVector(MetaData::ATTR_LIST_COLUMNS);

/* Assuming that the metadata for the column cust_address_typ
   is the fourth element in the list...
*/
MetaData customer_address=listOfColumns[3];
```

```
/* Now you can obtain the metadata for the customer_address attribute */
int typcode = customer_address.getInt(MetaData::ATTR_TYPECODE);
if(typcode==OCCI_TYPECODE_OBJECT)
   cout<<"customer_address is an object type"<<endl;
else
   cout<<"customer_address is not an object type"<<endl;

string objectName=customer_address.getString(MetaData::ATTR_OBJ_NAME);

/* Now that you have the name of the address object,
   the metadata of the attributes of the type can be obtained by using
   getMetaData on the connection by passing the object name
*/
MetaData address = connection->getMetaData(objectName);

/* Call getVector to obtain the list of the address object attributes */
vector<MetaData> attributeList =
                 address.getVector(MetaData::ATT_LIST_TYPE_ATTRS);

/* and so on to obtain metadata on other address object specific attributes */
```

次のコード例は、オブジェクトへの参照が使用されているときに、そのオブジェクトに関するメタデータを取得する方法を示しています。

次のスキーマを想定しています。

```
Type ADDRESS(street VARCHAR2(50), city VARCHAR2(20));
Table Person(id NUMBER, addr REF ADDRESS);


/* Create an environment and a connection to the HR database */
.
.
.
/* Call the getMetaData method on the Connection object obtained */
MetaData perstab_metaData = connection->getMetaData("Person",
   MetaData::PTYPE_TABLE);

/* Now that you have the metadata information on the Person table,
   call the getxxx methods using the appropriate attributes
*/
/* Call getString */
cout<<"Schema:"<<(perstab_metaData.getString(MetaData::ATTR_OBJ_SCHEMA))<<endl;
```

```
if(perstab_metaData.getInt(perstab_metaData::ATTR_PTYPE)==MetaData::PTYPE_TABLE)
   cout<<"Person is a table"<<endl;
else
   cout<<"Person is not a table"<<endl;

/* Call getVector to obtain the list of columns in the Person table
*/
vector<MetaData>listOfColumns;
listOfColumns=perstab_metaData.getVector(MetaData::ATTR_LIST_COLUMNS);

/* Each of the list elements represents a column metadata,
   so now get the datatype of the column by passing ATTR_DATA_TYPE
   to getInt
*/
for(int i=0;i<numCols;i++)
{
int dataType=colList[i].getInt(MetaData::ATTR_DATA_TYPE);
/* If the datatype is a reference, get the Ref and obtain the metadata
   about the object by passing the Ref to getMetaData
*/
if(dataType==SQLT_REF)
   RefAny refTdo=colList[i].getRef(MetaData::ATTR_REF_TDO);

/* Now you can obtain the metadata about the object as shown
MetaData tdo_metaData=connection->getMetaData(refTdo);

/* Now that you have the metadata about the TDO, you can obtain the metadata
   about the object
*/
```

Resultset メタデータのコード例

次のコード例は、選択リストに関するメタデータを ResultSet オブジェクトから取得する方法を示しています。

```
/* Create an environment and a connection to the database */
.
.
.
/* Create a statement and associate it with a select clause */
string sqlStmt="SELECT * FROM EMPLOYEES";
Statement *stmt=conn->createStatement(sqlStmt);

/* Execute the statement to obtain a ResultSet */
ResultSet *rset=stmt->executeQuery();
```

```
/* Obtain the metadata about the select list */
vector<MetaData>cmd=rset->getColumnListMetaData();

/* The metadata is a column list and each element is a column metaData */
int dataType=cmd[i].getInt(MetaData::ATTR_DATA_TYPE);
.
.
.
```

getMetaData メソッドがコールされるのは、ATTR_COLLECTION_ELEMENT 属性の場合のみです。

属性の参照

この項では、スキーマ・オブジェクトとサブスキーマ・オブジェクトに所属する属性について説明します。次の属性にグループ化されます。

- パラメータの属性
- 表とビューの属性
- プロシージャ、ファンクションおよびサブプログラムの属性
- パッケージの属性
- 型の属性
- 型属性の属性
- 型メソッドの属性
- コレクションの属性
- シノニムの属性
- 順序の属性
- 列の属性
- 引数および結果の属性
- リストの属性
- スキーマの属性
- データベースの属性

パラメータの属性

すべての要素には、その要素に固有な属性と一般的な属性がそれぞれいくつかあります。表 6-2 は、すべての要素に所属する属性を示しています。

表 6-2 すべての要素に所属する属性

属性	説明		属性のデータ型
ATTR_OBJ_ID	オブジェクトまたはスキーマの ID。		unsigned int
ATTR_OBJ_NAME	オブジェクト、スキーマまたはデータベースの名前。		string
ATTR_OBJ_SCHEMA	オブジェクトが置かれているスキーマ。		string
ATTR_OBJ_PTYPE	パラメータによって記述される情報の型。		int
	可能な値	**説明**	
	PTYPE_TABLE	表	
	PTYPE_VIEW	ビュー	
	PTYPE_PROC	プロシージャ	
	PTYPE_FUNC	ファンクション	
	PTYPE_PKG	パッケージ	
	PTYPE_TYPE	型	
	PTYPE_TYPE_ATTR	型の属性	
	PTYPE_TYPE_COLL	コレクションの型情報	
	PTYPE_TYPE_METHOD	型のメソッド	
	PTYPE_SYN	シノニム	
	PTYPE_SEQ	順序	
	PTYPE_COL	表またはビューの列	
	PTYPE_ARG	ファンクションまたはプシージャの引数	
	PTYPE_TYPE_ARG	型メソッドの引数	
	PTYPE_TYPE_RESULT	メソッドの結果	
	PTYPE_SCHEMA	スキーマ	
	PTYPE_DATABASE	データベース	
ATTR_TIMESTAMP	この記述の基本となるオブジェクトの TIMESTAMP（Oracle DATE 形式による）。		Timestamp

次の項では、様々な要素の型に固有な属性を示します。

表とビューの属性

表またはビュー（PTYPE_TABLE 型または PTYPE_VIEW 型）のパラメータには、次の表 6-3 で説明する型固有の属性があります。

表 6-3 表またはビューに所属する属性

属性	説明	属性のデータ型
ATTR_OBJID	オブジェクト ID。	unsigned int
ATTR_NUM_COLS	列数。	int
ATTR_LIST_COLUMNS	列リスト（PTYPE_LIST 型）。	vector<MetaData>
ATTR_REF_TDO	ベース型の TDO への REF（エクステント表の場合）。	RefAny
ATTR_IS_TEMPORARY	一時表またはビューであるかどうかの識別。	bool
ATTR_IS_TYPED	表またはビューに型が指定されているかどうかの識別。	bool
ATTR_DURATION	一時表の継続時間。可能な値は次のとおりです。 OCCI_DURATION_SESSION（セッション） OCCI_DURATION_TRANS（トランザクション） OCCI_DURATION_NULL（非一時表）	int

表に所属する追加の属性は、表 6-4 で説明します。

表 6-4 表に固有の属性

属性	説明	属性のデータ型
ATTR_DBA	セグメント・ヘッダーのデータ・ブロック・アドレス。	unsigned int
ATTR_TABLESPACE	表が存在している表領域。	int
ATTR_CLUSTERED	表がクラスタ化されているかどうかの識別。	bool
ATTR_PARTITIONED	表がパーティション化されているかどうかの識別。	bool
ATTR_INDEX_ONLY	表が索引のみの表であるかどうかの識別。	bool

プロシージャ、ファンクションおよびサブプログラムの属性

プロシージャまたはファンクション（PTYPE_PROC 型または PTYPE_FUNC 型）のパラメータには、表 6-5 で説明する型固有の属性があります。

表 6-5 プロシージャまたはファンクションに所属する属性

属性	説明	属性のデータ型
ATTR_LIST_ARGUMENTS	引数リスト。 6-23 ページの「リストの属性」を参照。	vector<MetaData>
ATTR_IS_INVOKER_RIGHTS	プロシージャまたはファンクションに実行者権限があるかどうかの識別。	int

パッケージ・サブプログラムに所属する追加の属性は、表 6-6 で説明します。

表 6-6 パッケージ・サブプログラムに所属する属性

属性	説明	属性のデータ型
ATTR_NAME	プロシージャまたはファンクションの名前。	string
ATTR_OVERLOAD_ID	ID 番号のオーバーロード（プロシージャまたはファンクションがパッケージの一部で、オーバーロードされる場合に該当）。戻り値は、PL/SQL ファンクションまたはプロシージャの直接問合せとは異なる場合があります。	int

パッケージの属性

パッケージ（PTYPE_PKG 型）のパラメータには、表 6-7 で説明する型固有の属性があります。

表 6-7 パッケージに所属する属性

属性	説明	属性のデータ型
ATTR_LIST_SUBPROGRAMS	サブプログラム・リスト。 6-23 ページの「リストの属性」を参照。	vector<MetaData>
ATTR_IS_INVOKER_RIGHTS	パッケージに実行者権限があるかどうかの識別。	bool

型の属性

型（PTYPE_TYPE 型）のパラメータには、表 6-8 で説明する属性があります。

表 6-8 型に所属する属性

属性	説明	属性のデータ型
ATTR_REF_TDO	列の型がオブジェクト型の場合に、その型の型記述子オブジェクトのメモリー内 REF を戻します。	**RefAny**
ATTR_TYPECODE	型コード。OCCI_TYPECODE_OBJECT または OCCI_TYPECODE_NAMEDCOLLECTION が可能です。 6-3 ページの「型と属性に関する注意」を参照。	int
ATTR_COLLECTION_TYPECODE	コレクションの型コード（型がコレクションの場合）。これ以外の場合は無効です。OCCI_TYPECODE_VARRAY または OCCI_TYPECODE_TABLE が可能です。 6-3 ページの「型と属性に関する注意」を参照。	int
ATTR_VERSION	ユーザーが割り当てたバージョンが含まれた NULL 終了文字列。	string
ATTR_IS_FINAL_TYPE	最終の型かどうかの識別。	bool
ATTR_IS_INSTANTIABLE_TYPE	インスタンス化可能な型かどうかの識別。	bool
ATTR_IS_SUBTYPE	サブタイプかどうかの識別。	bool
ATTR_SUPERTYPE_SCHEMA_NAME	スーパータイプが含まれているスキーマの名前。	string
ATTR_SUPERTYPE_NAME	スーパータイプの名前。	string
ATTR_IS_INVOKER_RIGHTS	この型が実行者権限かどうかの識別。	bool
ATTR_IS_INCOMPLETE_TYPE	この型が不完全かどうかの識別。	bool
ATTR_IS_SYSTEM_TYPE	システム型かどうかの識別。	bool
ATTR_IS_PREDEFINED_TYPE	事前定義の型かどうかの識別。	bool
ATTR_IS_TRANSIENT_TYPE	一時的な型かどうかの識別。	bool

表 6-8 型に所属する属性（続き）

属性	説明	属性のデータ型
ATTR_IS_SYSTEM_GENERATED_TYPE	システム生成の型かどうかの識別。	bool
ATTR_HAS_NESTED_TABLE	この型にネストした表の属性があるかどうかの識別。	bool
ATTR_HAS_LOB	この型に LOB 属性があるかどうかの識別。	bool
ATTR_HAS_FILE	この型に FILE 属性があるかどうかの識別。	bool
ATTR_COLLECTION_ELEMENT	コレクション要素へのハンドル。 6-17 ページの「コレクションの属性」を参照。	MetaData
ATTR_NUM_TYPE_ATTRS	型属性の数。	unsigned int
ATTR_LIST_TYPE_ATTRS	型属性のリスト。 6-23 ページの「リストの属性」を参照。	vector<MetaData>
ATTR_NUM_TYPE_METHODS	型メソッドの数。	unsigned int
ATTR_LIST_TYPE_METHODS	型メソッドのリスト。 6-23 ページの「リストの属性」を参照。	vector<MetaData>
ATTR_MAP_METHOD	型のマップ・メソッド。 6-16 ページの「型メソッドの属性」を参照。	MetaData
ATTR_ORDER_METHOD	型のオーダー・メソッド。 6-16 ページの「型メソッドの属性」を参照。	MetaData

型属性の属性

型の属性（PTYPE_TYPE_ATTR 型）のパラメータには、表 6-9 で説明する属性があります。

表 6-9 型属性に所属する属性

属性	説明	属性のデータ型
ATTR_DATA_SIZE	型属性の最大サイズ。この長さは、文字列と行に対して文字単位ではなく、バイト単位で戻されます。NUMBER に対しては、22 が戻されます。	int
ATTR_TYPECODE	型コード。 6-3 ページの「型と属性に関する注意」を参照。	int
ATTR_DATA_TYPE	型属性のデータ型。 6-3 ページの「型と属性に関する注意」を参照。	int
ATTR_NAME	型属性名である文字列へのポインタ。	string
ATTR_PRECISION	数値型属性の精度。精度が 0（ゼロ）以外でスケールが -127 の場合は、FLOAT です。それ以外の場合は、NUMBER(p, s) です。精度が 0（ゼロ）の場合、NUMBER(p, s) は、単に NUMBER で表すことができます。	int
ATTR_SCALE	数値型属性のスケール。精度が 0（ゼロ）以外でスケールが -127 の場合は、FLOAT です。それ以外の場合は、NUMBER(p, s) です。精度が 0（ゼロ）の場合、NUMBER(p, s) は、単に NUMBER として表すことができます。	int
ATTR_TYPE_NAME	型名の文字列。データ型が SQLT_NTY または SQLT_REF の場合は、戻り値に型名が含まれます。データ型が SQLT_NTY の場合は、名前付きデータ型の型名が戻されます。データ型が SQLT_REF の場合は、REF が指し示す名前付きデータ型の型名が戻されます。	string
ATTR_SCHEMA_NAME	型の作成時に使用したスキーマ名を持つ文字列。	string

表 6-9 型属性に所属する属性 （続き）

属性	説明	属性のデータ型
ATTR_REF_TDO	列の型がオブジェクトの場合に、その型の TDO のメモリー内 REF を戻します。	RefAny
ATTR_CHARSET_ID	キャラクタ・セット ID（型属性が文字列またはキャラクタ・タイプの場合）。	int
ATTR_CHARSET_FORM	キャラクタ・セット・フォーム（型属性が文字列またはキャラクタ・タイプの場合）。	int
ATTR_FSPRECISION	日時または間隔の小数秒の精度。	int
ATTR_LFPRECISION	間隔の先行フィールドの精度。	int

型メソッドの属性

型のメソッド（PTYPE_TYPE_METHOD 型）のパラメータには、表 6-10 で説明する属性があります。

表 6-10 型メソッドに所属する属性

属性	説明	属性のデータ型
ATTR_NAME	メソッド名（プロシージャまたはファンクション）。	string
ATTR_ENCAPSULATION	メソッドのカプセル化レベル（OCCI_TYPEENCAP_PRIVATE または OCCI_TYPEENCAP_PUBLIC のいずれか）。	int
ATTR_LIST_ARGUMENTS	引数リスト。	vector<MetaData>
ATTR_IS_CONSTRUCTOR	メソッドがコンストラクタかどうかの識別。	bool
ATTR_IS_DESTRUCTOR	メソッドがデストラクタかどうかの識別。	bool
ATTR_IS_OPERATOR	メソッドが演算子かどうかの識別。	bool
ATTR_IS_SELFISH	メソッドが自己参照かどうかの識別。	bool
ATTR_IS_MAP	メソッドがマップ・メソッドかどうかの識別。	bool
ATTR_IS_ORDER	メソッドがオーダー・メソッドかどうかの識別。	bool

表 6-10 型メソッドに所属する属性（続き）

属性	説明	属性のデータ型
ATTR_IS_RNDS	メソッドに "Read No Data State" が設定されているかどうかの識別。	bool
ATTR_IS_RNPS	メソッドに "Read No Process State" が設定されているかどうかの識別。	bool
ATTR_IS_WNDS	メソッドに "Write No Data State" が設定されているかどうかの識別。	bool
ATTR_IS_WNPS	メソッドに "Write No Process State" が設定されているかどうかの識別。	bool
ATTR_IS_FINAL_METHOD	最終のメソッドかどうかの識別。	bool
ATTR_IS_INSTANTIABLE_METHOD	インスタンス化可能なメソッドかどうかの識別。	bool
ATTR_IS_OVERRIDING_METHOD	オーバーライド・メソッドかどうかの識別。	bool

コレクションの属性

コレクション型（PTYPE_COLL 型）のパラメータには、表 6-11 で説明する属性があります。

表 6-11 コレクション型に所属する属性

属性	説明	属性のデータ型
ATTR_DATA_SIZE	型属性の最大サイズ。この長さは、文字列と行に対して文字単位ではなく、バイト単位で戻されます。NUMBER に対しては、22 が戻されます。	int
ATTR_TYPECODE	型コード。 6-3 ページの「型と属性に関する注意」を参照。	int
ATTR_DATA_TYPE	型属性のデータ型。 6-3 ページの「型と属性に関する注意」を参照。	int
ATTR_NUM_ELEMENTS	配列の要素数。配列のコレクションに対してのみ有効です。	unsigned int
ATTR_NAME	型属性名である文字列へのポインタ。	string

表 6-11 コレクション型に所属する属性（続き）

属性	説明	属性のデータ型
ATTR_PRECISION	数値型属性の精度。精度が 0（ゼロ）以外でスケールが -127 の場合は、FLOAT です。それ以外の場合は、NUMBER(p, s) です。精度が 0（ゼロ）の場合、NUMBER(p, s) は、単に NUMBER として表すことができます。	int
ATTR_SCALE	数値型属性のスケール。精度が 0（ゼロ）以外でスケールが -127 の場合は、FLOAT です。それ以外の場合は、NUMBER(p, s) です。精度が 0（ゼロ）の場合、NUMBER(p, s) は、単に NUMBER として表すことができます。	int
ATTR_TYPE_NAME	型名の文字列。データ型が SQLT_NTY または SQLT_REF の場合は、戻り値に型名が含まれます。データ型が SQLT_NTY の場合は、名前付きデータ型の型名が戻されます。データ型が SQLT_REF の場合は、REF が指し示す名前付きデータ型の型名が戻されます。	string
ATTR_SCHEMA_NAME	型の作成時に使用したスキーマ名を持つ文字列。	string
ATTR_REF_TDO	型属性の最大サイズ。この長さは、文字列と行に対して文字単位ではなく、バイト単位で戻されます。NUMBER に対しては、22 が戻されます。	RefAny
ATTR_CHARSET_ID	型コード。 6-3 ページの「型と属性に関する注意」を参照。	int
ATTR_CHARSET_FORM	型属性のデータ型。 6-3 ページの「型と属性に関する注意」を参照。	int

シノニムの属性

シノニム（PTYPE_SYN 型）のパラメータには、表 6-12 で説明する属性があります。

表 6-12 シノニムに所属する属性

属性	説明	属性のデータ型
ATTR_OBJID	オブジェクト ID。	unsigned int
ATTR_SCHEMA_NAME	シノニム変換のスキーマ名が含まれている NULL 終了文字列。	string
ATTR_NAME	シノニム変換のオブジェクト名が含まれている NULL 終了文字列。	string
ATTR_LINK	シノニム変換のデータベース・リンク名が含まれている NULL 終了文字列。	string

順序の属性

順序（PTYPE_SEQ 型）のパラメータには、表 6-13 で説明する属性があります。

表 6-13 順序に所属する属性

属性	説明	属性のデータ型
ATTR_OBJID	オブジェクト ID。	unsigned int
ATTR_MIN	最小値（Oracle 数値書式による）。	Number
ATTR_MAX	最大値（Oracle 数値書式による）。	Number
ATTR_INCR	増分値（Oracle 数値書式による）。	Number
ATTR_CACHE	キャッシュされた順序番号の数。順序がキャッシュされた順序でない場合は、0（ゼロ）です（Oracle 数値書式による）。	Number
ATTR_ORDER	順序が順序指定されているかどうかの識別。	bool
ATTR_HW_MARK	最高水位標（Oracle 数値書式による）。	Number

列の属性

表またはビューの列（PTYPE_COL 型）のパラメータには、表 6-14 で説明する属性があります。

表 6-14 表またはビューの列に所属する属性

属性	説明	属性のデータ型
ATTR_DATA_SIZE	列の最大サイズ。この長さは、文字列と行に対して文字単位ではなく、バイト単位で戻されます。NUMBER に対しては、22 が戻されます。	int
ATTR_DATA_TYPE	列のデータ型。 6-3 ページの「型と属性に関する注意」を参照。	int
ATTR_NAME	列名である文字列へのポインタ。	string
ATTR_PRECISION	数値列の精度。精度が 0（ゼロ）以外でスケールが -127 の場合は、FLOAT です。それ以外の場合は、NUMBER(p, s) です。精度が 0（ゼロ）の場合、NUMBER(p, s) は、単に NUMBER として表すことができます。	int
ATTR_SCALE	数値列のスケール。精度が 0（ゼロ）以外でスケールが -127 の場合は、FLOAT です。それ以外の場合は、NUMBER(p, s) です。精度が 0（ゼロ）の場合、NUMBER(p, s) は、単に NUMBER として表すことができます。	int
ATTR_IS_NULL	列に NULL 値が許可されていない場合に、0（ゼロ）を戻します。	bool
ATTR_TYPE_NAME	型名の文字列を戻します。データ型が SQLT_NTY または SQLT_REF の場合は、戻り値に型名が含まれます。データ型が SQLT_NTY の場合は、名前付きデータ型の型名が戻されます。データ型が SQLT_REF の場合は、REF が指し示す名前付きデータ型の型名が戻されます。	string
ATTR_SCHEMA_NAME	型の作成時に使用したスキーマ名を持つ文字列を戻します。	string

表 6-14 表またはビューの列に所属する属性 （続き）

属性	説明	属性のデータ型
ATTR_REF_TDO	列の型がオブジェクトの場合に、その型の TDO の REF を戻します。	RefAny
ATTR_CHARSET_ID	引数が文字列またはキャラクタ・タイプの場合に、キャラクタ・セット ID を戻します。	int
ATTR_CHARSET_FORM	引数が文字列またはキャラクタ・タイプの場合に、キャラクタ・セット・フォームを戻します。	int

引数および結果の属性

プロシージャまたはファンクション型の引数（PTYPE_ARG 型）、型メソッドの引数（PTYPE_TYPE_ARG 型）またはメソッド結果（PTYPE_TYPE_RESULT 型）のパラメータには、表 6-15 で説明する属性があります。

表 6-15 引数 / 結果に所属する属性

属性	説明	属性のデータ型
ATTR_NAME	引数名の文字列へのポインタを戻します。	string
ATTR_POSITION	引数リスト内の引数の位置。常に 0（ゼロ）を戻します。	int
ATTR_TYPECODE	型コード。 6-3 ページの「型と属性に関する注意」を参照。	int
ATTR_DATA_TYPE	引数のデータ型。 6-3 ページの「型と属性に関する注意」を参照。	int
ATTR_DATA_SIZE	引数のデータ型のサイズ。この長さは、文字列と行に対して文字単位ではなく、バイト単位で戻されます。NUMBER に対しては、22 が戻されます。	int
ATTR_PRECISION	数値引数の精度。精度が 0（ゼロ）以外でスケールが -127 の場合は、FLOAT です。それ以外の場合は、NUMBER(p, s) です。精度が 0（ゼロ）の場合、NUMBER(p, s) は、単に NUMBER として表すことができます。	int

表 6-15 引数 / 結果に所属する属性（続き）

属性	説明	属性のデータ型
ATTR_SCALE	数値引数のスケール。精度が 0（ゼロ）以外でスケールが -127 の場合は、FLOAT です。それ以外の場合は、NUMBER(p, s) です。精度が 0（ゼロ）の場合、NUMBER(p, s) は、単に NUMBER として表すことができます。	int
ATTR_LEVEL	データ型レベル。この属性は常に 0（ゼロ）を戻します。	int
ATTR_HAS_DEFAULT	引数にデフォルト値があるかどうかの識別。	int
ATTR_LIST_ARGUMENTS	次のレベルにある引数のリスト（引数がレコード型または表型の場合）。	vector<MetaData>
ATTR_IOMODE	引数のモードの識別。 IN（OCCI_TYPEPARAM_IN）に対する有効値は 0（ゼロ）です。 OUT（OCCI_TYPEPARAM_OUT）に対する有効値は 1 です。 IN/OUT（OCCI_TYPEPARAM_INOUT）に対する有効値は 2 です。	int
ATTR_RADIX	基数を戻します（数値型の場合）。	int
ATTR_IS_NULL	列に NULL 値が許可されていない場合に、0（ゼロ）を戻します。	int
ATTR_TYPE_NAME	型名の文字列を戻すか、パッケージのローカル型の場合はパッケージ名の文字列を戻します。データ型が SQLT_NTY または SQLT_REF の場合は、戻り値に型名が含まれます。データ型が SQLT_NTY の場合は、名前付きデータ型の型名が戻されます。データ型が SQLT_REF の場合は、REF が指し示す名前付きデータ型の型名が戻されます。	string
ATTR_SCHEMA_NAME	SQLT_NTY または SQLT_REF については、型の作成時に使用したスキーマ名を持つ文字列を戻すか、パッケージのローカル型の場合には、そのパッケージの作成時に使用したスキーマ名を持つ文字列を戻します。	string
ATTR_SUB_NAME	SQLT_NTY または SQLT_REF については、パッケージのローカル型の場合に、型名を持つ文字列を戻します。	string
ATTR_LINK	SQLT_NTY または SQLT_REF については、型が存在しているデータベースのデータベース・リンク名を持つ文字列を戻します。この動作は、パッケージがリモートで、そのパッケージがローカル型の場合にのみ発生します。	string

表 6-15 引数 / 結果に所属する属性（続き）

属性	説明	属性のデータ型
ATTR_REF_TDO	引数の型がオブジェクトの場合に、その型の TDO の REF を戻します。	RefAny
ATTR_CHARSET_ID	引数が文字列またはキャラクタ・タイプの場合に、キャラクタ・セット ID を戻します。	int
ATTR_CHARSET_FORM	引数が文字列またはキャラクタ・タイプの場合に、キャラクタ・セット・フォームを戻します。	int

リストの属性

属性のリスト型によって、リスト内のすべての要素を記述できます。ファンクションの引数リストの場合は、位置 0 に戻り値（PTYPE_ARG）のパラメータがあります。

このリストは、すべての要素について繰返し記述されます。結果は、C++ の vector<MetaData> に格納されます。getVector メソッドをコールして、属性のリスト型を記述します。表 6-16 にリスト属性を示します。

表 6-16 ATTR_LIST_TYPE の値

可能な値	説明
ATTR_LIST_COLUMNS	列のリスト。
ATTR_LIST_ARGUMENTS	プロシージャまたはファンクションの引数リスト。
ATTR_LIST_SUBPROGRAMS	サブプログラムのリスト。
ATTR_LIST_TYPE_ATTRIBUTES	型属性のリスト。
ATTR_LIST_TYPE_METHODS	型メソッドのリスト。
ATTR_LIST_OBJECTS	スキーマ内のオブジェクトのリスト。
ATTR_LIST_SCHEMAS	データベース内のスキーマのリスト。

スキーマの属性

スキーマ型（PTYPE_SCHEMA 型）のパラメータには、表 6-17 で説明する属性があります。

表 6-17 スキーマ固有の属性

属性	説明	属性のデータ型
ATTR_LIST_OBJECTS	スキーマ内のオブジェクトのリスト。	string

データベースの属性

データベース（PTYPE_DATABASE 型）のパラメータには、表 6-18 で説明する属性があります。

表 6-18 データベース固有の属性

属性	説明	属性のデータ型
ATTR_VERSION	データベースのバージョン。	string
ATTR_CHARSET_ID	サーバー・ハンドルからのデータベース・キャラクタ・セット ID。	int
ATTR_NCHARSET_ID	サーバー・ハンドルからのデータベース固有のキャラクタ・セット ID。	int
ATTR_LIST_SCHEMAS	データベース内のスキーマ（PTYPE_SCHEMA 型）のリスト。	vector<MetaData>
ATTR_MAX_PROC_LEN	プロシージャ名の最大長。	unsigned int
ATTR_MAX_COLUMN_LEN	列名の最大長。	unsigned int
ATTR_CURSOR_COMMIT_BEHAVIOR	カーソル、およびデータベース内のプリコンパイルされた SQL 文に、コミット操作を反映させる方法。可能な値は次のとおりです。カーソルの状態をコミット操作以前の状態に維持するには OCCI_CURSOR_OPEN、コミット時にクローズするには OCCI_CURSOR_CLOSED。ただし、アプリケーションでは、再度プリコンパイルしないで、文を再実行できます。	int
ATTR_MAX_CATALOG_NAMELEN	カタログ（データベース）名の最大長。	int
ATTR_CATALOG_LOCATION	修飾表内のカタログの位置。有効な値は、OCCI_CL_START と OCCI_CL_END です。	int
ATTR_SAVEPOINT_SUPPORT	データベースがセーブポイントをサポートしているかどうかを識別。有効な値は、OCCI_SP_SUPPORTED と OCCI_SP_UNSUPPORTED です。	int
ATTR_NOWAIT_SUPPORT	データベースが NOWAIT 句をサポートしているかどうかを識別。有効な値は、OCCI_NW_SUPPORTED と OCCI_NW_UNSUPPORTED です。	int

表 6-18 データベース固有の属性 （続き）

属性	説明	属性のデータ型
ATTR_AUTOCOMMIT_DDL	DDL 文に自動コミット・モードが必要かどうかを識別。有効な値は、OCCI_AC_DDL と OCCI_NO_AC_DDL です。	int
ATTR_LOCKING_MODE	データベースのロック・モード。有効な値は、OCCI_LOCK_IMMEDIATE と OCCI_LOCK_DELAYED です。	int

関連項目：

- この章で説明した概念の実例については、付録 A「OCCI デモ・プログラム」および「occidesc.cpp」のコード例を参照してください。

7

Object Type Translator ユーティリティの使用方法

この章では、Object Type Translator（OTT）ユーティリティについて説明します。このユーティリティを使用して、データベース・オブジェクト型、LOB 型および名前付きコレクション型を、OCCI、OCI および Pro*C/C++ のアプリケーションで使用できるように、C 構造体および C++ クラスの宣言にマッピングします。

> **注意：** JDK 1.3.1 との互換性がある Java コンパイラと Java インタプリタが正しくインストールされている必要があります。

この章は、次の項目で構成されています。

Object Type Translator ユーティリティの概要

Object Type Translator（OTT）ユーティリティは、Oracle データベース・サーバーのユーザー定義型を利用するアプリケーションの開発に役立ちます。

オブジェクト型は、SQL の CREATE TYPE 文を使用して作成できます。これらの型の定義はデータベースに格納されており、データベース表の作成時に使用できます。表が作成されると、Oracle C++ Call Interface（OCCI）、Oracle Call Interface（OCI）または Pro*C/C++ の各プログラマは、表に格納されているオブジェクトにアクセスできます。

オブジェクト・データにアクセスするアプリケーションには、データをホスト言語形式で表現する機能が必要です。この機能は、オブジェクト型を C の構造体または C++ のクラスとして表現することで実現します。

構造体またはクラスを手動でコーディングしてデータベース・オブジェクト型を表現することもできますが、時間がかかり、エラーが発生しやすくなります。OTT ユーティリティは、C に対する適切な構造体宣言または C++ に対する適切なクラスを自動的に生成することで、このステップを簡略化します。

Pro*C/C++ の場合、アプリケーションで必要なステップは、OTT ユーティリティで生成したヘッダー・ファイルを組み込むことのみです。OCI の場合、アプリケーションでは、OTT ユーティリティで生成した初期化関数をコールすることも必要です。

OCCI の場合、アプリケーションでは、次のファイルを組み込んでリンクする必要があります。

- 生成されたクラス宣言が含まれたヘッダー・ファイルを組み込みます。
- マッピングを登録する関数のプロトタイプが含まれたヘッダー・ファイルを組み込みます。
- オブジェクトをインスタンス化している間に OCCI がコールする静的メソッドが含まれた C++ ソース・ファイルとのリンクを設定します。
- マッピングを環境に登録する関数が含まれているファイルとのリンクを設定して、この関数をコールします。

C の場合、OTT ユーティリティでは、格納されているデータ型を表す C 構造体を作成する以外に、オブジェクト型またはそのフィールドが NULL かどうかを示すパラレル・インジケータ構造体も生成します。これは、C++ には適用されません。

OTT ユーティリティの使用方法

データベース型を C または C++ の表現に変換するには、OTT ユーティリティを明示的に起動する必要があります。さらに、OCI プログラマは、プログラムに必要なユーザー定義型に関する情報を使用して、タイプ・バージョン表と呼ばれるデータ構造体を初期化する必要があります。この初期化を実施するコードは、OTT ユーティリティで生成されます。

Pro*C/C++ の場合、タイプ・バージョン情報は、Pro*C/C++ へのパラメータとして渡される OUTTYPE ファイルに記録されます。

OCCI プログラマは、マッピングを環境に登録する関数を呼び出す必要があります。この関数は OTT ユーティリティによって生成されます。

ほとんどのオペレーティング・システムでは、コマンドラインから OTT ユーティリティを起動します。OTT ユーティリティでは、INTYPE ファイルを入力として受け取り、OUTTYPE ファイルと 1 つ以上の C ヘッダー・ファイルまたは 1 つ以上の C++ ヘッダー・ファイルと C++ メソッド・ファイルを生成します。OCI プログラマのために、オプションの実装ファイルが生成されます。OCCI プログラマのためには、マッピングを登録するための追加 C++ メソッド・ファイルが、プロトタイプが含まれた対応するヘッダー・ファイルとともに生成されます。

C の例

次の記述は、OTT ユーティリティを起動して C 構造体を生成するコマンドラインの例です。

```
ott userid=scott/tiger intype=demoin.typ outtype=demoout.typ code=c hfile=demo.h
```

このコマンドで、OTT ユーティリティをユーザー名 `scott` およびパスワード `tiger` でデータベースに接続し、INTYPE ファイル demoin.typ の指示に従ってデータベース型を C 構造体に変換します。その結果生成された構造体は、`code` パラメータで指定したホスト言語（C 言語）用としてヘッダー・ファイル demo.h に出力されます。OUTTYPE ファイル demoout.typ は、変換に関する情報を受け取ります。

各パラメータについては、この章の後の項で詳しく説明します。

demoin.typ ファイルと demoout.typ ファイルの例

次の記述は、demoin.typ ファイルの例です。

```
CASE=LOWER
TYPE employee
```

次の記述は、demoout.typ ファイルの例です。

```
CASE = LOWER
TYPE SCOTT.EMPLOYEE AS employee
   VERSION = "$8.0"
   HFILE = demo.h
```

この例の demoin.typ ファイルには、キーワード TYPE の前に変換対象の型があります。OUTTYPE ファイルの構造は、INTYPE ファイルに類似しており、OTT ユーティリティで取得した情報が追加されます。

OTT ユーティリティによる変換が終了すると、ヘッダー・ファイルには、INTYPE ファイルに指定されている各型の C 構造体表現と、各型に対応した NULL インジケータ構造体が含まれます。

INTYPE ファイルにリストされている employee 型は、次のように定義されていると仮定します。

```
CREATE TYPE employee AS OBJECT
(
   name       VARCHAR2(30),
   empno      NUMBER,
   deptno     NUMBER,
   hiredate   DATE,
   salary     NUMBER
);
```

OTT ユーティリティによって生成されたヘッダー・ファイル demo.h には、その他の項目の中に、次の宣言が含まれます。

```
struct employee
{
   OCIString * name;
   OCINumber empno;
   OCINumber deptno;
   OCIDate   hiredate;
   OCINumber salary;
};
typedef struct emp_type emp_type;

struct employee_ind
{
   OCIInd _atomic;
   OCIInd name;
   OCIInd empno;
   OCIInd deptno;
   OCIInd hiredate;
   OCIInd salary;
};
typedef struct employee_ind employee_ind;
```

注意： INTYPE ファイルのパラメータによって、生成された構造体の命名方法が制御されます。この例の構造体名の employee は、データベース型名の employee と一致しています。INTYPE ファイル内に CASE=lower の行があるため、構造体名には小文字が使用されます。

関連項目： 型の詳細は、7-16 ページの「OTT ユーティリティのデータ型マッピング」を参照してください。

C++ の例

次の記述は、OTT ユーティリティを起動して C++ クラスを生成する OTT コマンドの例です。

```
ott userid=scott/tiger intype=demoin.typ outtype=demoout.typ code=cpp hfile=demo.h
cppfile=demo.cpp mapfile=RegisterMappings.cpp
```

このコマンドで、OTT ユーティリティをユーザー名 scott およびパスワード tiger でデータベースに接続し、demoin.typ ファイルを INTYPE ファイルとして、demoout.typ ファイルを OUTTYPE ファイルとして使用します。生成された宣言は CODE=cpp パラメータで指定された C++ で demo.h ファイルに出力され、メソッド実装は demo.cpp ファイルに出力され、マッピングを登録する関数は、RegisterMappings.h に出力されたプロトタイプとともに、RegisterMappings.cpp に出力されます。

前項と同様に demoin.typ ファイルと employee 型を使用すると、OTT ユーティリティで次のファイルが生成されます。

- demo.h
- demo.cpp
- RegisterMappings.h
- RegisterMappings.cpp
- demoout.typ

各ファイルの内容は、次の項を参照してください。

- demo.h ファイルの内容
- demo.cpp ファイルの内容
- RegisterMappings.h ファイルの内容
- RegisterMappings.cpp ファイルの内容
- demoout.typ ファイルの内容

demo.h ファイルの内容

```
#ifndef DEMO_ORACLE
# define DEMO_ORACLE

#ifndef OCCI_ORACLE
# include <occi.h>
#endif


/*************************************************************/
//  generated declarations for the EMPLOYEE object type.
/*************************************************************/

class employee : public oracle::occi::PObject {

protected:

   OCCI_STD_NAMESPACE::string name;
   oracle::occi::Number empno;
   oracle::occi::Number deptno;
   oracle::occi::Date hiredate;
   oracle::occi::Number salary;

public:

   void *operator new(size_t size);

   void *operator new(size_t size, const oracle::occi::Connection * sess,
      const OCCI_STD_NAMESPACE::string& table);

   OCCI_STD_NAMESPACE::string getSQLTypeName() const;

   employee();

   employee(void *ctxOCCI_) : oracle::occi::PObject (ctxOCCI_) { };

   static void *readSQL(void *ctxOCCI_);

   virtual void readSQL(oracle::occi::AnyData& streamOCCI_);

   static void writeSQL(void *objOCCI_, void *ctxOCCI_);

   virtual void writeSQL(oracle::occi::AnyData& streamOCCI_);

};

#endif
```

demo.cpp ファイルの内容

```
#ifndef DEMO_ORACLE
# include "demo.h"
#endif


/************************************************************************/
//  generated method implementations for the EMPLOYEE object type.
/************************************************************************/

void *employee::operator new(size_t size)
{
  return oracle::occi::PObject::operator new(size);
}

void *employee::operator new(size_t size, const oracle::occi::Connection * sess,
  const OCCI_STD_NAMESPACE::string& table)
{
  return oracle::occi::PObject::operator new(size, sess, table,
            (char *) "SCOTT.EMPLOYEE");
}

OCCI_STD_NAMESPACE::string employee::getSQLTypeName() const
{
  return OCCI_STD_NAMESPACE::string("SCOTT.EMPLOYEE");
}

employee::employee()
{
}

void *employee::readSQL(void *ctxOCCI_)
{
  employee *objOCCI_ = new employee(ctxOCCI_);
  oracle::occi::AnyData streamOCCI_(ctxOCCI_);

  try
  {
    if (streamOCCI_.isNull())
      objOCCI_->setNull();
    else
      objOCCI_->readSQL(streamOCCI_);
  }
  catch (oracle::occi::SQLException& excep)
  {
    delete objOCCI_;
    excep.setErrorCtx(ctxOCCI_);
```

```
    return (void *)NULL;
  }
  return (void *)objOCCI_;
}

void employee::readSQL(oracle::occi::AnyData& streamOCCI_)
{
   name = streamOCCI_.getString();
   empno = streamOCCI_.getNumber();
   deptno = streamOCCI_.getNumber();
   hiredate = streamOCCI_.getDate();
   salary = streamOCCI_.getNumber();
}

void employee::writeSQL(void *objectOCCI_, void *ctxOCCI_)
{
  employee *objOCCI_ = (employee *) objectOCCI_;
  oracle::occi::AnyData streamOCCI_(ctxOCCI_);

  try
  {
    if (objOCCI_->isNull())
      streamOCCI_.setNull();
    else
      objOCCI_->writeSQL(streamOCCI_);
  }
  catch (oracle::occi::SQLException& excep)
  {
    excep.setErrorCtx(ctxOCCI_);
  }
  return;
}

void employee::writeSQL(oracle::occi::AnyData& streamOCCI_)
{
   streamOCCI_.setString(name);
   streamOCCI_.setNumber(empno);
   streamOCCI_.setNumber(deptno);
   streamOCCI_.setDate(hiredate);
   streamOCCI_.setNumber(salary);
}
```

RegisterMappings.h ファイルの内容

```
#ifndef REGISTERMAPPINGS_ORACLE
# define REGISTERMAPPINGS_ORACLE

#ifndef OCCI_ORACLE
# include <occi.h>
#endif

#ifndef DEMO_ORACLE
# include "demo.h"
#endif

void RegisterMappings(oracle::occi::Environment* envOCCI_);

#endif
```

RegisterMappings.cpp ファイルの内容

```
#ifndef REGISTERMAPPINGS_ORACLE
# include "registermappings.h"
#endif

void RegisterMappings(oracle::occi::Environment* envOCCI_)
{
  oracle::occi::Map *mapOCCI_ = envOCCI_->getMap();
  mapOCCI_->put("SCOTT.EMPLOYEE", employee::readSQL, employee::writeSQL);
}
```

demoout.typ ファイルの内容

```
CASE = LOWER
MAPFILE = RegisterMappings.cpp
MAPFUNC = RegisterMappings

TYPE SCOTT.EMPLOYEE AS employee
  VERSION = "$8.0"
  HFILE = demo.h
```

関連項目： 完全な C++ の例は、7-48 ページの「OTT クラスの拡張の例」を参照してください。

データベースでの型の作成

OTT ユーティリティを使用する最初のステップは、オブジェクト型または名前付きコレクション型を作成してデータベースに格納することです。そのためには、SQL の CREATE TYPE 文を使用します。

次の記述は、オブジェクトを作成する文の例です。

```
CREATE TYPE FULL_NAME AS OBJECT (first_name CHAR(20), last_name CHAR(20));
CREATE TYPE ADDRESS AS OBJECT (state CHAR(20), zip CHAR(20));
CREATE TYPE ADDRESS_TAB AS VARRAY(3) OF REF ADDRESS;
CREATE TYPE PERSON AS OBJECT (id NUMBER, name FULL_NAME, curr_addr REF ADDRESS,
   prev_addr_1 ADDRESS_TAB) NOT FINAL;
CREATE TYPE STUDENT UNDER PERSON (school_name CHAR(20));
```

OTT ユーティリティの起動

データベースに型を作成した後の次のステップでは、OTT ユーティリティを起動します。

OTT パラメータの指定

OTT パラメータは、コマンドラインまたは構成ファイルのいずれかで指定できます。一部のパラメータは INTYPE ファイルでも指定できます。

1 つのパラメータを複数箇所に指定する場合は、コマンドラインのパラメータ値が INTYPE ファイルのパラメータ値よりも優先されます。INTYPE ファイルの値は、ユーザー定義の構成ファイルの値よりも優先されます。また、このユーザー定義の構成ファイルの値は、デフォルトの構成ファイルの値よりも優先されます。

したがって、パラメータの優先順位は、次のとおりです。

1. OTT コマンドライン
2. INTYPE ファイルの値
3. ユーザー定義の構成ファイル
4. デフォルトの構成ファイル

グローバル・オプション（つまり、コマンドラインのオプションまたは TYPE 文の前にある INTYPE ファイルの最初のオプション）の場合は、コマンドラインの値が INTYPE ファイルの値よりも優先されます。（INTYPE ファイルでグローバル指定できるオプションは、CASE、INITFILE、INITFUNC、MAPFILE および MAPFUNC です。HFILE または CPPFILE は、グローバル指定できません。）INTYPE ファイルに TYPE 指定で記述されている内容は、特定の型に対してのみ適用され、この特定の型に対して通常適用されるコマンドラインの内容よりも優先されます。したがって、TYPE person HFILE=p.h を入力すると、その内容は person のみに適用され、コマンドラインの HFILE が上書きされます。この文は、コマンドライン・パラメータとみなされません。

コマンドラインのパラメータの設定

コマンドラインに設定されたパラメータ（オプションとも呼ばれます）は、他で設定されたパラメータやオプションよりも優先されます。

> **関連項目：** 詳細は、7-10 ページの「OTT ユーティリティの起動」を参照してください。

INTYPE ファイルのパラメータの設定

INTYPE ファイルは、OTT ユーティリティで変換する型のリストです。

CASE、CPPFILE、HFILE、INITFILE、INITFUNC、MAPFILE および MAPFUNC の各パラメータは、INTYPE ファイルに指定することができます。

> **関連項目：** 詳細は、7-14 ページの「INTYPE ファイルの概要」を参照してください。

構成ファイルのパラメータの設定

構成ファイルは、OTT パラメータが記述されているテキスト・ファイルです。ファイル内の各非空白行には、1 つのパラメータと、それに関連付けられた 1 つ以上の値が含まれています。1 行に 2 つ以上のパラメータを指定した場合は、最初のパラメータのみが使用されます。構成ファイルの非空白行では、空白を使用できません。

構成ファイルは、コマンドラインで名前を指定できます。また、デフォルトの構成ファイルは常に読み込まれます。このデフォルトの構成ファイルは、常に存在している必要がありますが、空でも構いません。デフォルトの構成ファイルの名前は ottcfg.cfg であり、ファイルの位置はオペレーティング・システム固有の設定です。

> **関連項目：** デフォルトの構成ファイルの位置は、使用しているオペレーティング・システムのマニュアルを参照してください。

コマンドラインでの OTT ユーティリティの起動

ほとんどのプラットフォームでは、コマンドラインから OTT ユーティリティを起動します。コマンドラインでは特に、入出力ファイルおよびデータベース接続情報を指定できます。

> **関連項目：** オペレーティング・システムで OTT ユーティリティを起動する方法は、使用しているオペレーティング・システムのマニュアルを参照してください。

C++ に対する OTT ユーティリティの起動

次の記述は、C++ クラスを生成する OTT ユーティリティの起動例です。

```
ott userid=scott/tiger intype=demoin.typ outtype=demoout.typ code=cpp hfile=demo.h
cppfile=demo.cpp mapfile=RegisterMappings.cpp
```

注意： OTT コマンドラインの等号（=）の前後には空白を使用できません。

OTT コマンドラインで使用する要素の説明

次の各項では、例で使用しているコマンドラインの要素を説明します。

- ott コマンド
- userid パラメータ
- intype パラメータ
- outtype パラメータ
- code パラメータ
- hfile パラメータ
- cppfile パラメータ
- mapfile パラメータ

> **関連項目：** 各 OTT コマンドライン・パラメータの説明は、7-108 ページの「OTT ユーティリティの参照」を参照してください。

ott コマンド OTT ユーティリティを起動します。必ずコマンドラインの先頭に置きます。

userid パラメータ OTT ユーティリティが使用するデータベース接続情報を指定します。前述の 2 つの例にある OTT ユーティリティでは、ユーザー名 `scott` およびパスワード `tiger` で接続を試行しています。

intype パラメータ INTYPE ファイルの名前を指定します。前述の 2 つの例にある INTYPE ファイルの名前は、demoin.typ と指定されています。

outtype パラメータ OUTTYPE ファイルの名前を指定します。OTT ユーティリティでヘッダー・ファイルが生成されると、変換された型の情報も OUTTYPE ファイルに出力されます。このファイルには、変換された各型のエントリが、そのバージョン文字列も含めて格納されます。また、C または C++ 表現で記述されたヘッダー・ファイルも格納されます。

前述の 2 つの例にある OUTTYPE ファイルの名前は、demoout.typ と指定されています。

code パラメータ 変換のターゲット言語を指定します。有効な値は、次のとおりです。

- ANSI_C（ANSI C 対応）
- C（ANSI_C と等価）
- KR_C（Kernighan & Ritchie C 対応）
- CPP（C++ 対応）

現在はデフォルト値がないため、このパラメータは必須です。

構造体の宣言は、C、ANSI_C および KR_C のいずれの C 言語オプションに対しても同一です。INITFILE ファイルに定義されている初期化関数の定義スタイルは、KR_C が使用されるかどうかによって異なります。INITFILE パラメータを使用しない場合、C 言語オプションは等価です。

注意： 前述の C の例にあるターゲット言語は、C（code=c）に指定されています。前述の C++ の例にある言語は C++（code=cpp）で、CPPFILE パラメータと MAPFILE パラメータが指定されています。

CODE パラメータを cpp に設定して C++ クラスを生成する場合は、CPPFILE パラメータと MAPFILE パラメータを使用する必要があります。

hfile パラメータ 生成された C 構造体または C++ クラスが書き込まれる C または C++ のヘッダー・ファイル名を指定します。

> **関連項目：** CPPFILE パラメータと MAPFILE パラメータの詳細は、7-108 ページの「OTT コマンドラインの構文」を参照してください。

cppfile パラメータ メソッド実装が書き込まれる C++ ソース・ファイルの名前を指定します。このファイルに生成されるメソッドは、オブジェクトをインスタンス化するときに OCCI によってコールされます。アプリケーション内で直接コールされることはありません。このパラメータは、OCCI アプリケーションの場合にのみ必要です。

mapfile パラメータ マッピングを環境に登録する関数が書き込まれる C++ ソース・ファイルの名前を指定します。関数のプロトタイプが含まれた、対応するヘッダー・ファイルが作成されます。マッピングを登録するためのこの関数は、OCCI アプリケーションの場合にのみ使用されます。

INTYPE ファイルの概要

OTT ユーティリティの実行時に、INTYPE ファイルは、変換するデータベース型を OTT ユーティリティに指定します。INTYPE ファイルは、生成する構造体やクラスの名前付けも制御します。INTYPE ファイルは、直接作成できますが、以前に OTT ユーティリティを起動したときに生成された OUTTYPE ファイルを利用することもできます。INTYPE ファイルを使用しないと、OTT ユーティリティが接続しているスキーマのすべての型が変換されます。

次の記述は、ユーザー作成 INTYPE ファイルの例です。

```
CASE=LOWER
TYPE employee
   TRANSLATE SALARY$ AS salary
             DEPTNO AS department
TYPE ADDRESS
TYPE item
TYPE "Person"
TYPE PURCHASE_ORDER AS p_o
```

1 行目の CASE パラメータは、小文字による C 識別子の生成を示しています。ただし、この CASE パラメータは、INTYPE ファイルに明示的に記述されていない識別子にのみ適用されます。そのため、常に employee と ADDRESS は、それぞれ employee と ADDRESS という C 構造体になります。これらの構造体のメンバーには、小文字で名前が指定されます。

TYPE キーワードで始まる行は、変換するデータベース内の型を指定します。この例では、EMPLOYEE 型、ADDRESS 型、ITEM 型、PERSON 型および PURCHASE_ORDER 型を変換するように設定しています。

TRANSLATE ... AS キーワードは、オブジェクト型を C 構造体に変換するときに、オブジェクト属性の名前を変更することを指定しています。この例では、employee 型の SALARY$ 属性が salary に変換されます。

最終行の AS キーワードは、オブジェクト型を構造体に変換するときに、名前を変更することを指定しています。この例では、purchase_order データベース型を p_o と呼ばれる構造体に変換します。

AS キーワードを型または属性名の変換に使用しない場合は、その型または属性のデータベース名が C 識別子名として使用されます。ただし、CASE パラメータは有効であるため、有効な C 識別子文字にマッピングできない文字は、アンダースコア（_）に置換されます。型または属性を変換する理由は、次のとおりです。

- 名前に、アルファベット、数字またはアンダースコア以外の文字が含まれている。
- 名前が C キーワードと競合する。
- 型名が、同じ適用範囲内の別の識別子と競合する。この競合は、プログラムで、異なるスキーマにある同じ名前の 2 つの型を使用する場合などに発生します。
- プログラマが別の名前に変更する。

OTT ユーティリティでは、INTYPE ファイルにリストされていない他の型の変換が必要になる場合があります。これは、OTT ユーティリティは、変換前に INTYPE ファイルにある型の依存性を分析し、必要に応じて他の型を変換するためです。たとえば、ADDRESS 型は INTYPE ファイルにリストされていなくても、Person 型に ADDRESS 型の属性がある場合、OTT ユーティリティでは、Person 型の定義が必須のため、ADDRESS を変換します。

注意： リリース 1（9.0.1）では、INTYPE ファイルに指定されていない必要なオブジェクト型を、OTT ユーティリティで生成するかどうかを指定できます。必要なオブジェクト型を OTT ユーティリティで生成しない場合は、TRANSITIVE=FALSE を設定してください。デフォルトは、TRANSITIVE=TRUE です。

INTYPE ファイルでは、通常の大 / 小文字の区別がない SQL 識別子は、大 / 小文字のどのような組合せのつづりでも可能です。引用符は付けません。

CREATE TYPE "Person" などの大 / 小文字を区別して作成されている SQL 識別子を参照するには、TYPE "Person" のように引用符を使用してください。SQL 識別子は、宣言時に引用符が付けられた場合、大 / 小文字が区別されます。引用符は、TYPE "CASE" などの OTT 予約語である SQL 識別子を参照するためにも使用できます。この場合、その SQL 識別子が、CREATE TYPE Case のように大 / 小文字の区別なしに作成されていた場合、引用符付きの名前は大文字であることが必要です。SQL 識別子の名前を参照するための OTT 予約語に引用符が付けられていない場合、OTT ユーティリティは、INTYPE ファイルに構文エラーをレポートします。

関連項目：

- INTYPE ファイルの構造に関する詳細な指定と使用可能なオプションは、7-118 ページの「INTYPE ファイルの構造」を参照してください。
- CASE パラメータに関する詳細は、7-111 ページの「CASE パラメータ」を参照してください。

OTT ユーティリティのデータ型マッピング

OTT ユーティリティによって、データベース型から C 構造体または C++ クラスが生成されると、その構造体またはクラスには、オブジェクト型の各属性に対応する 1 つの要素が含まれます。属性のデータ型は、Oracle オブジェクト・データ型で使用される型にマッピングされます。Oracle のデータ型には、事前定義の基本的な一連の型が組み込まれており、オブジェクト型やコレクションなどのユーザー定義の型の作成をサポートします。

一連の事前定義の型には、数値型や文字型を含む、プログラマによく知られている標準の型があります。ラージ・オブジェクト・データ型（BLOB や CLOB など）も含まれています。

Oracle には、オブジェクト型属性を C 構造体または C++ クラスで表現するための一連の事前定義の型も含まれています。たとえば、次のようなオブジェクト型定義とそれに対応する OTT 生成の構造体宣言を想定します。

```
CREATE TYPE employee AS OBJECT
(  name       VARCHAR2(30),
   empno      NUMBER,
   deptno     NUMBER,
   hiredate   DATE,
   salary$    NUMBER);
```

CASE パラメータが LOWER に設定され、型名または属性名の明示的なマッピングがないと仮定すると、OTT ユーティリティの出力は、次のようになります。

```
struct employee
{  OCIString * name;
   OCINumber empno;
   OCINumber deptno;
   OCIDate   hiredate;
   OCINumber salary_;
};
typedef struct emp_type emp_type;
struct employee_ind
{  OCIInd _atomic;
   OCIInd name;
   OCIInd empno;
   OCIInd deptno;
   OCIInd hiredate;
   OCIInd salary_;
}
typedef struct employee_ind employee_ind;
```

ここでは、構造体宣言のデータ型（LNOCIString、LNOCINumber、LNOCIDate、LNOCIInd）がオブジェクト型属性のデータ型をマッピングするために使用されています。たとえば、empno 属性の NUMBER データ型は、LNOCINumber データ型にマッピングされます。これらのデータ型は、バインド変数および定義変数の型としても使用できます。

関連項目：

- OCI アプリケーションでのデータ型（オブジェクト・データ型を含む）の使用方法の詳細は、『Oracle Call Interface プログラマーズ・ガイド』を参照してください。

オブジェクト・データ型の C へのマッピング

この項では、OTT ユーティリティで生成される、オブジェクト属性型の C の型へのマッピングについて説明します。表 7-1 は、属性として使用できる型から、OTT ユーティリティで生成されるオブジェクト・データ型へのマッピングのリストです。

関連項目： 様々なマッピングの例は、7-20 ページの「C に関する OTT ユーティリティの型マッピング例」を参照してください。

表 7-1 オブジェクト型属性の C オブジェクト・データ型マッピング

オブジェクト属性の型	C マッピング
BFILE	LNOCIBFileLocator*
BLOB	LNOCIBlobLocator*
CHAR(n)、CHARACTER(n)	LNOCIString*
CLOB	LNOCIClobLocator*
DATE	LNOCIDate*
DEC、DEC(n)、DEC(n,n)	LNOCINumber
DECIMAL、DECIMAL(n)、DECIMAL(n,n)	LNOCINumber
FLOAT、FLOAT(n)、DOUBLE PRECISION	LNOCINumber
INT、INTEGER、SMALLINT	LNOCINumber
INTERVAL DAY TO SECOND	LNOCIInterval
INTERVAL YEAR TO MONTH	LNOCIInterval
ネストされたオブジェクト型	ネストされたオブジェクト型の C の名前。
NESTED TABLE	typedef を使用して宣言。LNOCITable* と等価。
NUMBER、NUMBER(n)、NUMBER(n,n)	LNOCINumber
NUMERIC、NUMERIC(n)、NUMERIC(n,n)	LNOCINumber
RAW	LNOCIRaw*

表 7-1 オブジェクト型属性の C オブジェクト・データ型マッピング（続き）

オブジェクト属性の型	C マッピング
REAL	LNOCINumber
REF	typedef を使用して宣言。LNOCIRef* と等価。
TIMESTAMP、TIMESTAMP WITH TIME ZONE、TIMESTAMP WITH LOCAL TIME ZONE	LNOCIDateTime*
VARCHAR(n)	LNOCIString*
VARCHAR2(n)	LNOCIString*
VARRAY	typedef を使用して宣言。LNOCIArray* と等価。

注意： REF 型、VARRAY 型および NESTED TABLE 型の場合、OTT ユーティリティでは typedef が生成されます。この typedef で宣言された型は、構造体宣言でデータ・メンバーの型として使用されます。例については、次の項の「C に関する OTT ユーティリティの型マッピング例」を参照してください。

オブジェクト型に REF またはコレクション型の属性が含まれている場合は、その REF またはコレクション型の typedef が最初に生成されます。次に、そのオブジェクト型に対応する構造体宣言が生成されます。この構造体には、型が REF またはコレクション型へのポインタである要素が含まれます。

オブジェクト型に、型が別のオブジェクト型である属性が含まれている場合、OTT ユーティリティでは、ネストされたオブジェクト型が最初に生成されます。次に、OTT ユーティリティは、オブジェクト型属性をネストされたオブジェクト型のネストした構造体にマッピングします。

OTT ユーティリティがオブジェクト以外のデータベース属性の型をマッピングする先の C データ型は構造体です。これらの構造体は、LNOCIDate 以外は不透明です。

オブジェクト・データ型の C++ へのマッピング

この項では、OTT ユーティリティで生成される、オブジェクト属性型の C++ の型へのマッピングについて説明します。表 7-2 は、属性として使用できる型から、OTT ユーティリティで生成されるオブジェクト・データ型へのマッピングのリストです。

表 7-2 オブジェクト型属性の C++ オブジェクト・データ型マッピング

オブジェクト属性の型	C++ マッピング
BFILE	Bfile
BLOB	Blob
CHAR(n)、CHARACTER(n)	string
CLOB	Clob
DATE	Date
DEC、DEC(n)、DEC(n,n)	Number
DECIMAL、DECIMAL(n)、DECIMAL(n,n)	Number
FLOAT、FLOAT(n)、DOUBLE PRECISION	Number
INT、INTEGER、SMALLINT	Number
INTERVAL DAY TO SECOND	IntervalDS
INTERVAL YEAR TO MONTH	IntervalYM
ネストされたオブジェクト型	ネストされたオブジェクト型の C++ の名前
NESTED TABLE	vector<attribute_type>
NUMBER、NUMBER(n)、NUMBER(n,n)	Number
NUMERIC、NUMERIC(n)、NUMERIC(n,n)	Number
RAW	Bytes
REAL	Number
REF	Ref<attribute_type>
TIMESTAMP、TIMESTAMP WITH TIME ZONE、TIMESTAMP WITH LOCAL TIME ZONE	Timestamp
VARCHAR(n)	string
VARCHAR2(n)	string
VARRAY	vector<attribute_type>

C に関する OTT ユーティリティの型マッピング例

この項にある例は、OTT ユーティリティが C プログラム用に作成する様々な型のマッピングを示しています。

この例では、次のデータベース型が作成されていることを想定しています。

```
CREATE TYPE my_varray AS VARRAY(5) of integer;

CREATE TYPE object_type AS OBJECT
(object_name VARCHAR2(20));

CREATE TYPE other_type AS OBJECT
(object_number NUMBER);

CREATE TYPE my_table AS TABLE OF object_type;

CREATE TYPE many_types AS OBJECT
(  the_varchar    VARCHAR2(30),
   the_char       CHAR(3),
   the_blob       BLOB,
   the_clob       CLOB,
   the_object     object_type,
   another_ref    REF other_type,
   the_ref        REF many_types,
   the_varray     my_varray,
   the_table      my_table,
   the_date       DATE,
   the_num        NUMBER,
   the_raw        RAW(255));
```

また、この例では、INTYPE ファイルが存在しており、次のオプションが含まれていることを想定しています。

```
CASE = LOWER
TYPE many_types
```

OTT ユーティリティによって次の C 構造体が生成されます。

注意： ここでは、構造体の補足説明のためにコメントが付けられています。これらのコメントは、OTT ユーティリティの出力の一部ではありません。

```
#ifndef MYFILENAME_ORACLE
#define MYFILENAME_ORACLE

#ifndef OCI_ORACLE
#include <oci.h>
#endif

typedef OCIRef many_types_ref;
typedef OCIRef object_type_ref;
typedef OCIArray my_varray;             /* part of many_types */
typedef OCITable my_table;              /* part of many_types*/
typedef OCIRef other_type_ref;
struct object_type                      /* part of many_types */
{
   OCIString * object_name;
};
typedef struct object_type object_type;

struct object_type_ind                  /*indicator struct for*/
{                                       /*object_types*/
   OCIInd _atomic;
   OCIInd object_name;
};
typedef struct object_type_ind object_type_ind;

struct many_types
{
   OCIString *        the_varchar;
   OCIString *        the_char;
   OCIBlobLocator *   the_blob;
   OCIClobLocator *   the_clob;
   struct object_type the_object;
   other_type_ref *   another_ref;
   many_types_ref *   the_ref;
   my_varray *        the_varray;
   my_table *         the_table;
   OCIDate            the_date;
   OCINumber          the_num;
   OCIRaw *           the_raw;
};
typedef struct many_types many_types;

struct many_types_ind                   /*indicator struct for*/
{                                       /*many_types*/
   OCIInd _atomic;
   OCIInd the_varchar;
   OCIInd the_char;
```

```
   OCIInd the_blob;
   OCIInd the_clob;
   struct object_type_ind the_object;   /*nested*/
   OCIInd another_ref;
   OCIInd the_ref;
   OCIInd the_varray;
   OCIInd the_table;
   OCIInd the_date;
   OCIInd the_num;
   OCIInd the_raw;
};
typedef struct many_types_ind many_types_ind;

#endif
```

INTYPE ファイルには、変換用の項目が 1 つのみリストされていますが、2 つのオブジェクト型と 2 つの名前付きコレクション型が変換されていることに注意してください。これは、OTT ユーティリティの TRANSITIVE パラメータに、デフォルト値の TRUE があるためです。TRANSITIVE=TRUE の場合、OTT ユーティリティは、リストされている型の変換を完了するために、変換する型の属性として使用されている型をすべて自動的に変換します。

この自動変換は、オブジェクト型属性のポインタまたは参照によってのみアクセスされる型には適用されません。たとえば、many_types 型には属性の another_ref REF other_type が含まれていますが、other_type 構造体の宣言は生成されていません。

また、この例には、VARRAY、NESTED TABLE および REF の各型を宣言するために typedef を使用する方法が示されています。

前述の例で、typedef は、OTT ユーティリティによって生成されたファイルの先頭部分にあります。

```
typedef OCIRef many_types_ref;
typedef OCIRef object_type_ref;
typedef OCIArray my_varray;
typedef OCITable my_table;
typedef OCIRef other_type_ref;
```

構造体 many_types では、次のように VARRAY、NESTED TABLE および REF の各属性が宣言されています。

```
struct many_types
{
   ...
   other_type_ref *   another_ref;
   many_types_ref *   the_ref;
   my_varray *        the_varray;
   my_table *         the_table;
   ...
}
```

C++ に関する OTT の型マッピング例

次の記述は、C++ に関する OTT の型マッピングの例です。前項の例で作成された型と次のオプションを含む INTYPE ファイルを前提にしています。

```
CASE = LOWER
TYPE many_types
```

OTT ユーティリティでは、次の C++ クラス宣言が生成されます。

```
#ifndef MYFILENAME_ORACLE
# define MYFILENAME_ORACLE

#ifndef OCCI_ORACLE
# include <occi.h>
#endif


/*---------------------------------------------------------*/
//  generated declarations for the OBJECT_TYPE object type.
/*---------------------------------------------------------*/

class object_type : public oracle::occi::PObject {

protected:

   OCCI_STD_NAMESPACE::string object_name;

public:

   void *operator new(size_t size);

   void *operator new(size_t size, const oracle::occi::Connection * sess,
      const OCCI_STD_NAMESPACE::string& table);

   OCCI_STD_NAMESPACE::string getSQLTypeName() const;

   object_type();

   object_type(void *ctxOCCI_) : oracle::occi::PObject (ctxOCCI_) { };

   static void *readSQL(void *ctxOCCI_);

   virtual void readSQL(oracle::occi::AnyData& streamOCCI_);

   static void writeSQL(void *objOCCI_, void *ctxOCCI_);
```

```
   virtual void writeSQL(oracle::occi::AnyData& streamOCCI_);

};

/*************************************************************/
//  generated declarations for the OTHER_TYPE object type.
/*************************************************************/

class other_type : public oracle::occi::PObject {

protected:

   oracle::occi::Number object_number;

public:

   void *operator new(size_t size);

   void *operator new(size_t size, const oracle::occi::Connection * sess,
      const OCCI_STD_NAMESPACE::string& table);

   OCCI_STD_NAMESPACE::string getSQLTypeName() const;

   other_type();

   other_type(void *ctxOCCI_) : oracle::occi::PObject (ctxOCCI_) { };

   static void *readSQL(void *ctxOCCI_);

   virtual void readSQL(oracle::occi::AnyData& streamOCCI_);

   static void writeSQL(void *objOCCI_, void *ctxOCCI_);

   virtual void writeSQL(oracle::occi::AnyData& streamOCCI_);

};

/*************************************************************/
//  generated declarations for the MANY_TYPES object type.
/*************************************************************/

class many_types : public oracle::occi::PObject {

protected:
```

```
   OCCI_STD_NAMESPACE::string the_varchar;
   OCCI_STD_NAMESPACE::string the_char;
   oracle::occi::Blob the_blob;
   oracle::occi::Clob the_clob;
   object_type * the_object;
   oracle::occi::Ref< other_type > another_ref;
   oracle::occi::Ref< many_types > the_ref;
   OCCI_STD_NAMESPACE::vector< oracle::occi::Number > the_varray;
   OCCI_STD_NAMESPACE::vector< object_type * > the_table;
   oracle::occi::Date the_date;
   oracle::occi::Number the_num;
   oracle::occi::Bytes the_raw;

public:

   void *operator new(size_t size);

   void *operator new(size_t size, const oracle::occi::Connection * sess,
      const OCCI_STD_NAMESPACE::string& table);

   OCCI_STD_NAMESPACE::string getSQLTypeName() const;

   many_types();

   many_types(void *ctxOCCI_) : oracle::occi::PObject (ctxOCCI_) { };

   static void *readSQL(void *ctxOCCI_);

   virtual void readSQL(oracle::occi::AnyData& streamOCCI_);

   static void writeSQL(void *objOCCI_, void *ctxOCCI_);

   virtual void writeSQL(oracle::occi::AnyData& streamOCCI_);

};

#endif
```

C++ では、TRANSITIVE=TRUE の場合、OTT ユーティリティは、変換する型の属性として使用されている型をすべて自動的に変換します。これには、オブジェクト型属性のポインタまたは REF によってのみアクセスされる型も含まれます。C++ の例では、many_types オブジェクトのみが INTYPE ファイルに指定されていますが、クラス宣言はすべてのオブジェクト型に対して生成されています。これには、many_types オブジェクトの REF によってのみアクセスされる other_type オブジェクトも含まれています。

OUTTYPE ファイルの概要

OUTTYPE ファイルは、OTT コマンドラインで名前が指定されます。OTT ユーティリティによって、C または C++ ヘッダー・ファイルが生成されると、変換の結果が OUTTYPE ファイルに出力されます。このファイルには、変換された各型のエントリが、そのバージョン文字列も含めて格納され、さらに、C または C++ 表現で記述されたヘッダー・ファイルが格納されます。

ある OTT ユーティリティの実行による OUTTYPE ファイルは、その OTT ユーティリティのその後の起動で INTYPE ファイルとして使用できます。

たとえば、この章の前の例で使用した、次の単純な INTYPE ファイルを想定します。

```
CASE=LOWER
TYPE employee
   TRANSLATE SALARY$ AS salary
             DEPTNO AS department
TYPE ADDRESS
TYPE item
TYPE "Person"
TYPE PURCHASE_ORDER AS p_o
```

この INTYPE ファイルでは、プログラマによって、OTT 生成の C 識別子に対して大 / 小文字の区別が指定され、変換する型のリストが用意されています。これらの型のうちの 2 つについては、ネーミング規則が指定されています。

OTT ユーティリティ実行後の OUTTYPE ファイルは、次のようになります。

```
CASE = LOWER
TYPE EMPLOYEE AS employee
  VERSION = "$8.0"
  HFILE = demo.h
  TRANSLATE SALARY$ AS salary
             DEPTNO AS department
TYPE ADDRESS AS ADDRESS
  VERSION = "$8.0"
  HFILE = demo.h
TYPE ITEM AS item
  VERSION = "$8.0"
  HFILE = demo.h
TYPE "Person" AS Person
  VERSION = "$8.0"
  HFILE = demo.h
TYPE PURCHASE_ORDER AS p_o
  VERSION = "$8.0"
  HFILE = demo.h
```

INTYPE ファイルで指定しなかった型が OUTTYPE ファイルに含まれている場合があります。たとえば、INTYPE ファイルに person 型のみの変換を指定した場合を想定します。

```
CASE = LOWER
TYPE PERSON
```

person 型の定義に address 型の属性が含まれている場合、OUTTYPE ファイルには、PERSON と ADDRESS の両方のエントリが含まれます。person 型を完全に変換するには、最初に address を変換する必要があります。

OTT ユーティリティは、変換前に INTYPE ファイルにある型の依存性を分析し、必要に応じて他の型を変換します。

注意： リリース 1（9.0.1）では、INTYPE ファイルに指定されていない必要なオブジェクト型を、OTT ユーティリティで生成するかどうかを指定できます。必要なオブジェクト型を OTT ユーティリティで生成しない場合は、TRANSITIVE=FALSE を設定してください。デフォルトは、TRANSITIVE=TRUE です。

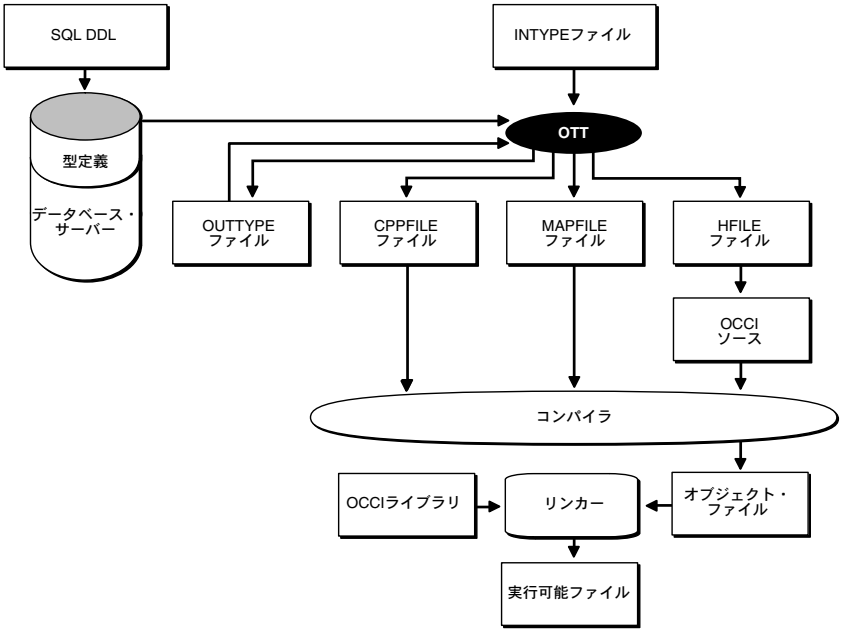
関連項目： これらのパラメータの詳細は、7-10 ページの「OTT ユーティリティの起動」を参照してください。

OTT ユーティリティと OCCI アプリケーション

OTT ユーティリティは、オブジェクトを生成し、SQL データ型を C++ クラスにマッピングします。OTT ユーティリティは、オブジェクトをインスタンス化するときに OCCI がコールするいくつかのメソッドと、マッピングを環境に登録するために OCCI アプリケーションでコールされる関数も実装しています。これらの宣言は、使用している OCCI アプリケーションにインクルード（#include）するヘッダー・ファイルに格納されます。マッピングを登録する関数のプロトタイプは、OCCI アプリケーションにもインクルードされる独立したヘッダー・ファイルに記述されます。メソッド実装は、OCCI アプリケーションにリンクされている C++ ソース・コード・ファイル（拡張子 .cpp）に格納されます。マッピングを登録する関数は、アプリケーションにもリンクされている独立した C++（.cpp）ファイルに格納されます。

図 7-1 は、OTT ユーティリティを OCCI で使用する場合のステップを示しています。これらのステップは、次の図の後に説明します。

図 7-1 OCCI における OTT ユーティリティ



1. SQL DLL を使用してデータベースに型定義を作成します。
2. OTT ユーティリティで変換するデータベース型が含まれた INTYPE ファイルを作成します。
3. C++ が生成されるように指定し、OTT ユーティリティを起動します。

   OTT ユーティリティは、次のファイルを生成します。

   - オブジェクト型の C++ クラス表現が含まれたヘッダー・ファイル（拡張子 .h）。ファイル名は、`HFILE` パラメータによって、OTT コマンドラインに指定されます。
   - マッピングを登録する関数（`MAPFUNC`）のプロトタイプが含まれたヘッダー・ファイル。
   - オブジェクトをインスタンス化している間に OCCI がコールする静的メソッドが含まれた C++ ソース・ファイル（拡張子 .cpp）。これらのメソッドは、使用している OCCI アプリケーションから直接コールしないでください。ファイル名は、`CPPFILE` パラメータによって、OTT コマンドラインに指定されます。

- マッピングを環境に登録するために使用する関数が含まれたファイル（拡張子 .cpp）。ファイル名は、MAPFILE パラメータによって、OTT コマンドラインに指定されます。
- 変換された各型のエントリ、バージョン文字列および変換結果を書き込むヘッダー・ファイルが含まれたファイル（OUTTYPE ファイル）。ファイル名は、OUTTYPE パラメータによって、OTT コマンドラインに指定されます。

4. OCCI アプリケーションを記述し、OTT ユーティリティで作成されたヘッダー・ファイルを OCCI ソース・コード・ファイルにインクルードします。

   アプリケーションで環境を宣言し、関数 MAPFUNC をコールしてマッピングを登録します。

5. OCCI アプリケーションをコンパイルして OCCI オブジェクト・コードを作成し、そのオブジェクト・コードを OCCI ライブラリにリンクしてプログラム実行可能ファイルを作成します。

C++ の OTT ユーティリティ・パラメータ

OTT ユーティリティを使用して C++ を生成するには、CODE パラメータを CODE=CPP に設定する必要があります。CODE=CPP を指定した場合は、メソッド実装ファイルとマッピング登録関数ファイルのファイル名を定義するために、CPPFILE パラメータと MAPFILE パラメータを指定する必要があります。マッピング関数の名前は、OTT ユーティリティによって MAPFILE から作成するか、MAPFUNC パラメータを使用して指定できます。また、ATTRACCESS は、生成したコードの変更を指定できるオプションのパラメータです。

次のパラメータは C++ 固有のもので、C++ クラスの生成を制御します。

- CPPFILE
- MAPFILE
- MAPFUNC
- ATTRACCESS

関連項目： これらのパラメータの詳細は、7-110 ページの「OTT ユーティリティのパラメータ」を参照してください。

OTT 生成の C++ クラス

OTT ユーティリティによって、データベース・オブジェクト型から C++ クラスが生成されると、そのクラスの宣言には、オブジェクト型の各属性に対応する 1 つの要素が含まれます。属性のデータ型は、7-19 ページの表 7-2 に定義されているように、Oracle オブジェクト・データ型で使用される型にマッピングされます。

各クラスごとに、2 つの new 演算子、1 つの getSQLTypeName メソッド、2 つのコンストラクタ、2 つの readSQL メソッドおよび 2 つの writeSQL メソッドが生成されます。getSQLTypeName メソッド、コンストラクタ、readSQL メソッドおよび writeSQL メソッドは、オブジェクト・データのマーシャリングおよびアンマーシャリングの際に OCCI によってコールされます。

デフォルトでは、オブジェクト型に対して OTT が生成した C++ クラスは PObject クラスから導出されるため、生成されたそのクラス内のコンストラクタも PObject クラスから導出されます。継承されたデータベース型の場合、クラスは、生成されたコンストラクタのように親の型のクラスから導出され、親のクラスにはない属性に対応する要素のみが含まれます。

データベース型の属性に対応する要素を含むクラス宣言およびメソッド宣言は、OTT ユーティリティによって生成されたヘッダー・ファイルに含まれます。メソッド実装は、OTT ユーティリティによって生成された CPPFILE ファイルに含まれます。

次の記述は、この一連のステップの結果、OTT ユーティリティによって生成される C++ クラスの例です。

1. 型を定義します。

```
CREATE TYPE FULL_NAME AS OBJECT (first_name CHAR(20), last_name CHAR(20));
CREATE TYPE ADDRESS AS OBJECT (state CHAR(20), zip CHAR(20));
CREATE TYPE ADDRESS_TAB AS VARRAY(3) of REF ADDRESS;
CREATE TYPE PERSON AS OBJECT (id NUMBER, name FULL_NAME, curr_addr REF ADDRESS,
   prev_addr_l ADDRESS_TAB) NOT FINAL;
CREATE TYPE STUDENT UNDER PERSON (school_name CHAR(20));
```

2. INTYPE ファイルを準備します。

```
CASE = SAME
MAPFILE = RegisterMappings_3.cpp
TYPE FULL_NAME AS FullName
  TRANSLATE first_name as FirstName
            last_name as LastName
TYPE ADDRESS
TYPE PERSON
TYPE STUDENT
```

3. OTT ユーティリティを起動します。

```
ott userid=scott/tiger intype=demoin_3.typ outype=demoout_3.typ code=cpp hfile=demo_
3.h cppfile=demo_3.cpp
```

この例では、ファイル名 demo_3.h のヘッダー・ファイル、ファイル名 demo_3.cpp の C++ ソース・ファイルおよびファイル名 demoout_3.typ の OUTTYPE ファイルが作成されます。OTT ユーティリティによって生成されるこれらのファイルについては、次の項を参照してください。

- OTT ユーティリティによって生成されるヘッダー・ファイルの例 : demo_3.h
- OTT ユーティリティによって生成される C++ ソース・ファイルの例 : demo_3.cpp
- OTT ユーティリティによって生成される OUTTYPE ファイルの例 : demoout_3.typ

OTT ユーティリティによって生成されるヘッダー・ファイルの例 : demo_3.h

この項では、OTT ユーティリティによって生成されるヘッダー・ファイル（ファイル名 demo_3.h）を前項の情報に従って示します。

```
#ifndef DEMO_3_ORACLE
# define DEMO_3_ORACLE

#ifndef OCCI_ORACLE
# include <occi.h>
#endif


/*---------------------------------------------------------*/
//  generated declarations for the FULL_NAME object type.
/*---------------------------------------------------------*/

class FullName : public oracle::occi::PObject {

protected:

   OCCI_STD_NAMESPACE::string FirstName;
   OCCI_STD_NAMESPACE::string LastName;

public:

   void *operator new(size_t size);

   void *operator new(size_t size, const oracle::occi::Connection * sess,
      const OCCI_STD_NAMESPACE::string& table);

   OCCI_STD_NAMESPACE::string getSQLTypeName() const;

   FullName();
```

```
   FullName(void *ctxOCCI_) : oracle::occi::PObject (ctxOCCI_) { };

   static void *readSQL(void *ctxOCCI_);

   virtual void readSQL(oracle::occi::AnyData& streamOCCI_);

   static void writeSQL(void *objOCCI_, void *ctxOCCI_);

   virtual void writeSQL(oracle::occi::AnyData& streamOCCI_);

};

/*************************************************************/
//  generated declarations for the ADDRESS object type.
/*************************************************************/

class ADDRESS : public oracle::occi::PObject {

protected:

   OCCI_STD_NAMESPACE::string STATE;
   OCCI_STD_NAMESPACE::string ZIP;

public:

   void *operator new(size_t size);

   void *operator new(size_t size, const oracle::occi::Connection * sess,
      const OCCI_STD_NAMESPACE::string& table);

   OCCI_STD_NAMESPACE::string getSQLTypeName() const;

   ADDRESS();

   ADDRESS(void *ctxOCCI_) : oracle::occi::PObject (ctxOCCI_) { };

   static void *readSQL(void *ctxOCCI_);

   virtual void readSQL(oracle::occi::AnyData& streamOCCI_);

   static void writeSQL(void *objOCCI_, void *ctxOCCI_);

   virtual void writeSQL(oracle::occi::AnyData& streamOCCI_);

};
```

```
/*************************************************************/
//  generated declarations for the PERSON object type.
/*************************************************************/

class PERSON : public oracle::occi::PObject {

protected:

   oracle::occi::Number ID;
   FullName * NAME;
   oracle::occi::Ref< ADDRESS > CURR_ADDR;
   OCCI_STD_NAMESPACE::vector< oracle::occi::Ref< ADDRESS > > PREV_ADDR_L;

public:

   void *operator new(size_t size);

   void *operator new(size_t size, const oracle::occi::Connection * sess,
      const OCCI_STD_NAMESPACE::string& table);

   OCCI_STD_NAMESPACE::string getSQLTypeName() const;

   PERSON();

   PERSON(void *ctxOCCI_) : oracle::occi::PObject (ctxOCCI_) { };

   static void *readSQL(void *ctxOCCI_);

   virtual void readSQL(oracle::occi::AnyData& streamOCCI_);

   static void writeSQL(void *objOCCI_, void *ctxOCCI_);

   virtual void writeSQL(oracle::occi::AnyData& streamOCCI_);

};

/*************************************************************/
//  generated declarations for the STUDENT object type.
/*************************************************************/

class STUDENT : public PERSON {

protected:

   OCCI_STD_NAMESPACE::string SCHOOL_NAME;
```

```
public:

   void *operator new(size_t size);

   void *operator new(size_t size, const oracle::occi::Connection * sess,
      const OCCI_STD_NAMESPACE::string& table);

   OCCI_STD_NAMESPACE::string getSQLTypeName() const;

   STUDENT();

   STUDENT(void *ctxOCCI_) : PERSON (ctxOCCI_) { };

   static void *readSQL(void *ctxOCCI_);

   virtual void readSQL(oracle::occi::AnyData& streamOCCI_);

   static void writeSQL(void *objOCCI_, void *ctxOCCI_);

   virtual void writeSQL(oracle::occi::AnyData& streamOCCI_);

};

#endif
```

OTT ユーティリティによって生成される C++ ソース・ファイルの例 : demo_3.cpp

この項では、OTT ユーティリティによって生成される C++ ソース・ファイル（ファイル名 demo_3.cpp）を前項の情報に従って示します。

```
#ifndef DEMO_3_ORACLE
# include "demo_3.h"
#endif


/*---------------------------------------------------------------*/
//  generated method implementations for the FULL_NAME object type.
/*---------------------------------------------------------------*/

void *FullName::operator new(size_t size)
{
  return oracle::occi::PObject::operator new(size);
}
```

```
void *FullName::operator new(size_t size, const oracle::occi::Connection * sess,
  const OCCI_STD_NAMESPACE::string& table)
{
  return oracle::occi::PObject::operator new(size, sess, table,
            (char *) "SCOTT.FULL_NAME");
}

OCCI_STD_NAMESPACE::string FullName::getSQLTypeName() const
{
  return OCCI_STD_NAMESPACE::string("SCOTT.FULL_NAME");
}

FullName::FullName()
{
}

void *FullName::readSQL(void *ctxOCCI_)
{
  FullName *objOCCI_ = new FullName(ctxOCCI_);
  oracle::occi::AnyData streamOCCI_(ctxOCCI_);

  try
  {
    if (streamOCCI_.isNull())
      objOCCI_->setNull();
    else
      objOCCI_->readSQL(streamOCCI_);
  }
  catch (oracle::occi::SQLException& excep)
  {
    delete objOCCI_;
    excep.setErrorCtx(ctxOCCI_);
    return (void *)NULL;
  }
  return (void *)objOCCI_;
}

void FullName::readSQL(oracle::occi::AnyData& streamOCCI_)
{
   FirstName = streamOCCI_.getString();
   LastName = streamOCCI_.getString();
}
```

```
void FullName::writeSQL(void *objectOCCI_, void *ctxOCCI_)
{
  FullName *objOCCI_ = (FullName *) objectOCCI_;
  oracle::occi::AnyData streamOCCI_(ctxOCCI_);

  try
  {
    if (objOCCI_->isNull())
      streamOCCI_.setNull();
    else
      objOCCI_->writeSQL(streamOCCI_);
  }
  catch (oracle::occi::SQLException& excep)
  {
    excep.setErrorCtx(ctxOCCI_);
  }
  return;
}

void FullName::writeSQL(oracle::occi::AnyData& streamOCCI_)
{
   streamOCCI_.setString(FirstName);
   streamOCCI_.setString(LastName);
}

/*---------------------------------------------------------------*/
//  generated method implementations for the ADDRESS object type.
/*---------------------------------------------------------------*/

void *ADDRESS::operator new(size_t size)
{
  return oracle::occi::PObject::operator new(size);
}

void *ADDRESS::operator new(size_t size, const oracle::occi::Connection * sess,
  const OCCI_STD_NAMESPACE::string& table)
{
  return oracle::occi::PObject::operator new(size, sess, table,
            (char *) "SCOTT.ADDRESS");
}

OCCI_STD_NAMESPACE::string ADDRESS::getSQLTypeName() const
{
  return OCCI_STD_NAMESPACE::string("SCOTT.ADDRESS");
}
```

```
ADDRESS::ADDRESS()
{
}

void *ADDRESS::readSQL(void *ctxOCCI_)
{
  ADDRESS *objOCCI_ = new ADDRESS(ctxOCCI_);
  oracle::occi::AnyData streamOCCI_(ctxOCCI_);

  try
  {
    if (streamOCCI_.isNull())
      objOCCI_->setNull();
    else
      objOCCI_->readSQL(streamOCCI_);
  }
  catch (oracle::occi::SQLException& excep)
  {
    delete objOCCI_;
    excep.setErrorCtx(ctxOCCI_);
    return (void *)NULL;
  }
  return (void *)objOCCI_;
}

void ADDRESS::readSQL(oracle::occi::AnyData& streamOCCI_)
{
   STATE = streamOCCI_.getString();
   ZIP = streamOCCI_.getString();
}

void ADDRESS::writeSQL(void *objectOCCI_, void *ctxOCCI_)
{
  ADDRESS *objOCCI_ = (ADDRESS *) objectOCCI_;
  oracle::occi::AnyData streamOCCI_(ctxOCCI_);

  try
  {
    if (objOCCI_->isNull())
      streamOCCI_.setNull();
    else
      objOCCI_->writeSQL(streamOCCI_);
  }
  catch (oracle::occi::SQLException& excep)
  {
    excep.setErrorCtx(ctxOCCI_);
  }
```

```
  return;
}

void ADDRESS::writeSQL(oracle::occi::AnyData& streamOCCI_)
{
   streamOCCI_.setString(STATE);
   streamOCCI_.setString(ZIP);
}

/*****************************************************************/
//  generated method implementations for the PERSON object type.
/*****************************************************************/

void *PERSON::operator new(size_t size)
{
  return oracle::occi::PObject::operator new(size);
}

void *PERSON::operator new(size_t size, const oracle::occi::Connection * sess,
  const OCCI_STD_NAMESPACE::string& table)
{
  return oracle::occi::PObject::operator new(size, sess, table,
           (char *) "SCOTT.PERSON");
}

OCCI_STD_NAMESPACE::string PERSON::getSQLTypeName() const
{
  return OCCI_STD_NAMESPACE::string("SCOTT.PERSON");
}

PERSON::PERSON()
{
   NAME = (FullName *) 0;
}

void *PERSON::readSQL(void *ctxOCCI_)
{
  PERSON *objOCCI_ = new PERSON(ctxOCCI_);
  oracle::occi::AnyData streamOCCI_(ctxOCCI_);

  try
  {
    if (streamOCCI_.isNull())
      objOCCI_->setNull();
    else
      objOCCI_->readSQL(streamOCCI_);
  }
```

```
  catch (oracle::occi::SQLException& excep)
  {
    delete objOCCI_;
    excep.setErrorCtx(ctxOCCI_);
    return (void *)NULL;
  }
  return (void *)objOCCI_;
}

void PERSON::readSQL(oracle::occi::AnyData& streamOCCI_)
{
   ID = streamOCCI_.getNumber();
   NAME = (FullName *) streamOCCI_.getObject();
   CURR_ADDR = streamOCCI_.getRef();
   getVector(streamOCCI_, PREV_ADDR_L);
}

void PERSON::writeSQL(void *objectOCCI_, void *ctxOCCI_)
{
  PERSON *objOCCI_ = (PERSON *) objectOCCI_;
  oracle::occi::AnyData streamOCCI_(ctxOCCI_);

  try
  {
    if (objOCCI_->isNull())
      streamOCCI_.setNull();
    else
      objOCCI_->writeSQL(streamOCCI_);
  }
  catch (oracle::occi::SQLException& excep)
  {
    excep.setErrorCtx(ctxOCCI_);
  }
  return;
}

void PERSON::writeSQL(oracle::occi::AnyData& streamOCCI_)
{
   streamOCCI_.setNumber(ID);
   streamOCCI_.setObject(NAME);
   streamOCCI_.setRef(CURR_ADDR);
   setVector(streamOCCI_, PREV_ADDR_L);
}
```

```
/*****************************************************************/
//  generated method implementations for the STUDENT object type.
/*****************************************************************/

void *STUDENT::operator new(size_t size)
{
  return oracle::occi::PObject::operator new(size);
}

void *STUDENT::operator new(size_t size, const oracle::occi::Connection * sess,
  const OCCI_STD_NAMESPACE::string& table)
{
  return oracle::occi::PObject::operator new(size, sess, table,
           (char *) "SCOTT.STUDENT");
}

OCCI_STD_NAMESPACE::string STUDENT::getSQLTypeName() const
{
  return OCCI_STD_NAMESPACE::string("SCOTT.STUDENT");
}

STUDENT::STUDENT()
{
}

void *STUDENT::readSQL(void *ctxOCCI_)
{
  STUDENT *objOCCI_ = new STUDENT(ctxOCCI_);
  oracle::occi::AnyData streamOCCI_(ctxOCCI_);

  try
  {
    if (streamOCCI_.isNull())
      objOCCI_->setNull();
    else
      objOCCI_->readSQL(streamOCCI_);
  }
  catch (oracle::occi::SQLException& excep)
  {
    delete objOCCI_;
    excep.setErrorCtx(ctxOCCI_);
    return (void *)NULL;
  }
  return (void *)objOCCI_;
}
```

```
void STUDENT::readSQL(oracle::occi::AnyData& streamOCCI_)
{
   PERSON::readSQL(streamOCCI_);
   SCHOOL_NAME = streamOCCI_.getString();
}

void STUDENT::writeSQL(void *objectOCCI_, void *ctxOCCI_)
{
  STUDENT *objOCCI_ = (STUDENT *) objectOCCI_;
  oracle::occi::AnyData streamOCCI_(ctxOCCI_);

  try
  {
    if (objOCCI_->isNull())
      streamOCCI_.setNull();
    else
      objOCCI_->writeSQL(streamOCCI_);
  }
  catch (oracle::occi::SQLException& excep)
  {
    excep.setErrorCtx(ctxOCCI_);
  }
  return;
}

void STUDENT::writeSQL(oracle::occi::AnyData& streamOCCI_)
{
   PERSON::writeSQL(streamOCCI_);
   streamOCCI_.setString(SCHOOL_NAME);
}
```

OTT ユーティリティによって生成される OUTTYPE ファイルの例 : demoout_3.typ

この項では、OTT ユーティリティによって生成される OUTTYPE ファイル（ファイル名 demoout_3.typ）を前項の情報に従って示します。

```
CASE = SAME
MAPFILE = RegisterMappings_3.cpp
MAPFUNC = RegisterMappings

TYPE SCOTT.FULL_NAME AS FullName
  VERSION = "$8.0"
  HFILE = demo_3.h
TRANSLATE FIRST_NAME AS FirstName
          LAST_NAME AS LastName
```

```
TYPE SCOTT.ADDRESS AS ADDRESS
  VERSION = "$8.0"
  HFILE = demo_3.h

TYPE SCOTT.PERSON AS PERSON
  VERSION = "$8.0"
  HFILE = demo_3.h

TYPE SCOTT.STUDENT AS STUDENT
  VERSION = "$8.0"
  HFILE = demo_3.h
```

ATTRACCESS=PRIVATE を使用した例

ATTRACCESS=PRIVATE の場合に生成されるコードの相違を示すために、次のパラメータが含まれた INTYPE ファイルを想定します。

```
CASE = SAME
TYPE PERSON
```

OTT ユーティリティによって、次のヘッダー・ファイルが生成されます。

```
#ifndef DEMO_4_ORACLE
# define DEMO_4 _ORACLE

#ifndef OCCI_ORACLE
# include <occi.h>
#endif


/*************************************************************/
//  generated declarations for the FULL_NAME object type.
/*************************************************************/

class FULL_NAME :public oracle::occi::PObject {

private:

   OCCI_STD_NAMESPACE::string FIRST_NAME;
   OCCI_STD_NAMESPACE::string LAST_NAME;

public:

   OCCI_STD_NAMESPACE::string getFirst_name() const;

   void setFirst_name(const OCCI_STD_NAMESPACE::string &value);
```

```
   OCCI_STD_NAMESPACE::string getLast_name() const;

   void setLast_name(const OCCI_STD_NAMESPACE::string &value);

   void *operator new(size_t size);

   void *operator new(size_t size, const oracle::occi::Connection * sess,
      const OCCI_STD_NAMESPACE::string& table);

   OCCI_STD_NAMESPACE::string getSQLTypeName() const;

   FULL_NAME();

   FULL_NAME(void *ctxOCCI_) :oracle::occi::PObject (ctxOCCI_) { };

   static void *readSQL(void *ctxOCCI_);

   virtual void readSQL(oracle::occi::AnyData& streamOCCI_);

   static void writeSQL(void *objOCCI_, void *ctxOCCI_);

   virtual void writeSQL(oracle::occi::AnyData& streamOCCI_);

};

/*************************************************************/
//  generated declarations for the ADDRESS object type.
/*************************************************************/

class ADDRESS :public oracle::occi::PObject {

private:

   OCCI_STD_NAMESPACE::string STATE;
   OCCI_STD_NAMESPACE::string ZIP;

public:

   OCCI_STD_NAMESPACE::string getState() const;

   void setState(const OCCI_STD_NAMESPACE::string &value);

   OCCI_STD_NAMESPACE::string getZip() const;

   void setZip(const OCCI_STD_NAMESPACE::string &value);

   void *operator new(size_t size);
```

```
   void *operator new(size_t size, const oracle::occi::Connection * sess,
      const OCCI_STD_NAMESPACE::string& table);

   OCCI_STD_NAMESPACE::string getSQLTypeName() const;

   ADDRESS();

   ADDRESS(void *ctxOCCI_) :oracle::occi::PObject (ctxOCCI_) { };

   static void *readSQL(void *ctxOCCI_);

   virtual void readSQL(oracle::occi::AnyData& streamOCCI_);

   static void writeSQL(void *objOCCI_, void *ctxOCCI_);

   virtual void writeSQL(oracle::occi::AnyData& streamOCCI_);

};

/*************************************************************/
//  generated declarations for the PERSON object type.
/*************************************************************/

class PERSON :public oracle::occi::PObject {

private:

   oracle::occi::Number ID;
   FULL_NAME * NAME;
   oracle::occi::Ref< ADDRESS > CURR_ADDR;
   OCCI_STD_NAMESPACE::vector< oracle::occi::Ref< ADDRESS > > PREV_ADDR_L;

public:

   oracle::occi::Number getId() const;

   void setId(const oracle::occi::Number &value);

   FULL_NAME * getName() const;

   void setName(FULL_NAME * value);

   oracle::occi::Ref< ADDRESS > getCurr_addr() const;

   void setCurr_addr(const oracle::occi::Ref< ADDRESS > &value);
```

```
   OCCI_STD_NAMESPACE::vector< oracle::occi::Ref< ADDRESS > >& getPrev_addr_l();

   const OCCI_STD_NAMESPACE::vector< oracle::occi::Ref< ADDRESS > >& getPrev_addr_
l() const;

   void setPrev_addr_l(const OCCI_STD_NAMESPACE::vector< oracle::occi::Ref< ADDRESS
> > &value);

   void *operator new(size_t size);

   void *operator new(size_t size, const oracle::occi::Connection * sess,
      const OCCI_STD_NAMESPACE::string& table);

   OCCI_STD_NAMESPACE::string getSQLTypeName() const;

   PERSON();

   PERSON(void *ctxOCCI_) :oracle::occi::PObject (ctxOCCI_) { };

   static void *readSQL(void *ctxOCCI_);

   virtual void readSQL(oracle::occi::AnyData& streamOCCI_);

   static void writeSQL(void *objOCCI_, void *ctxOCCI_);

   virtual void writeSQL(oracle::occi::AnyData& streamOCCI_);

};

#endif
```

ATTRACCESS=PRIVATE であるため、属性へのアクセスは private であり、アクセッサ・メソッド（getxxx）とミューテータ・メソッド（setxxx）が各属性に対して生成されます。

マップ・レジストリ関数

マッピングを環境に登録するために、1 つの関数が OTT ユーティリティによって生成されます。この関数には、OTT ユーティリティの起動によって変換されるすべての型に対するマッピングが含まれています。関数名は、MAPFUNC パラメータに指定した名前になります。指定されていない場合は、MAPFILE パラメータから導出されます。この関数に対する引数は、Environment へのポインタのみです。

この関数は、指定された Environment を使用して Map を取得し、変換した各タイプのマッピングを登録します。

前項にリストされているデータベース型と INTYPE ファイルおよび MAPFILE=RegisterMappings_3.cpp を指定した場合、生成されるマップ登録関数のフォームは次のようになります。

```
#ifndef REGISTERMAPPINGS_3_ORACLE
# include "registermappings_3.h"
#endif

void RegisterMappings_3(oracle::occi::Environment* envOCCI_)
{
  oracle::occi::Map *mapOCCI_ = envOCCI_->getMap();
  mapOCCI_->put("SCOTT.FULL_NAME", FullName::readSQL, FullName::writeSQL);
  mapOCCI_->put("SCOTT.ADDRESS", ADDRESS::readSQL, ADDRESS::writeSQL);
  mapOCCI_->put("SCOTT.PERSON", PERSON::readSQL, PERSON::writeSQL);
  mapOCCI_->put("SCOTT.STUDENT", STUDENT::readSQL, STUDENT::writeSQL);
}
```

マッピング登録関数のプロトタイプは、対応するヘッダー・ファイル RegisterMapping.h に、次のように出力されます。

```
#ifndef REGISTERMAPPINGS_3_ORACLE
# define REGISTERMAPPINGS_3_ORACLE

#ifndef OCCI_ORACLE
# include <occi.h>
#endif

#ifndef DEMO_3_ORACLE
# include "demo_3.h"
#endif

void RegisterMappings_3(oracle::occi::Environment* envOCCI_);

#endif
```

OTT C++ クラスの拡張

OTT ユーティリティによって生成されたクラスの機能を拡張するために、新しいクラスを導出することができます。メソッドをクラスに追加することもできますが、内在するリスクがあるため、お薦めしません。

> **関連項目：** OTT マーカーを使用して、OTT 生成ファイルに追加するコードを保存する方法の詳細は、7-59 ページの「ユーザー追加コードの引継ぎ」を参照してください。

OTT で生成したクラスから新しいクラスを導出する例の場合は、ADDRESS SQL オブジェクト型から CAddress クラスを生成することを想定しています。また、ADDRESS オブジェクトを表すために、MyAddress クラスの出力も想定しています。MyAddress クラスは、CAddress から導出できます。

これを行うには、OTT ユーティリティで生成するコードを次のように変更する必要があります。

- CAddress クラスのかわりに MyAddress クラスを使用して、データ型が ADDRESS の属性を表現します。
- CAddress クラスのかわりに MyAddress クラスを使用して、データ型が ADDRESS のベクターおよび REF の要素を表現します。
- ADDRESS から継承されるデータベース・オブジェクト型のベース・クラスとして、CAddress クラスのかわりに MyAddress クラスを使用します。導出クラスが MyAddress のサブタイプである場合でも、コールされた readSQL メソッドと writeSQL メソッドは、CAddress クラスのメソッドです。

> **注意：** あるクラスを、生成した別のクラスのベース・クラスとして、拡張および使用する場合は、継承する型のクラスと継承される型のクラスを、独立したファイルに生成する必要があります。

OTT ユーティリティを使用して CAddress クラス（MyAddress クラスの導出元）を生成するには、TYPE 文に次の句を指定する必要があります。

```
TYPE ADDRESS GENERATE CAdress AS MyAddress
```

OTT クラスの拡張の例

以前に作成した FULL_NAME、ADDRESS、PERSON および PFGRFDENT のデータベース型を指定し、INTYPE ファイルを変更して GENERATE ... AS 句を挿入します。

```
CASE = SAME
MAPFILE = RegisterMappings_5.cpp

TYPE FULL_NAME GENERATE CFullName AS MyFullName
  TRANSLATE first_name as FirstName
            last_name as LastName

TYPE ADDRESS GENERATE CAddress AS MyAddress
TYPE PERSON GENERATE CPerson AS MyPerson
TYPE STUDENT GENERATE CStudent AS MyStudent
```

OTT ユーティリティによって、次の C++ ソース・ファイル（拡張子 .cpp）が生成されます。

```
#ifndef MYFILENAME_ORACLE
# define MYFILENAME_ORACLE

#ifndef OCCI_ORACLE
# include <occi.h>
#endif


/*************************************************************/
//  generated declarations for the FULL_NAME object type.
/*************************************************************/

class CFullName : public oracle::occi::PObject {

protected:

   OCCI_STD_NAMESPACE::string FirstName;
   OCCI_STD_NAMESPACE::string LastName;

public:

   void *operator new(size_t size);

   void *operator new(size_t size, const oracle::occi::Connection * sess,
      const OCCI_STD_NAMESPACE::string& table);

   OCCI_STD_NAMESPACE::string getSQLTypeName() const;
```

```
   CFullName();

   CFullName(void *ctxOCCI_) : oracle::occi::PObject (ctxOCCI_) { };

   static void *readSQL(void *ctxOCCI_);

   virtual void readSQL(oracle::occi::AnyData& streamOCCI_);

   static void writeSQL(void *objOCCI_, void *ctxOCCI_);

   virtual void writeSQL(oracle::occi::AnyData& streamOCCI_);

};

/*************************************************************/
//  generated declarations for the ADDRESS object type.
/*************************************************************/

class CAddress : public oracle::occi::PObject {

protected:

   OCCI_STD_NAMESPACE::string STATE;
   OCCI_STD_NAMESPACE::string ZIP;

public:

   void *operator new(size_t size);

   void *operator new(size_t size, const oracle::occi::Connection * sess,
      const OCCI_STD_NAMESPACE::string& table);

   OCCI_STD_NAMESPACE::string getSQLTypeName() const;

   CAddress();

   CAddress(void *ctxOCCI_) : oracle::occi::PObject (ctxOCCI_) { };

   static void *readSQL(void *ctxOCCI_);

   virtual void readSQL(oracle::occi::AnyData& streamOCCI_);

   static void writeSQL(void *objOCCI_, void *ctxOCCI_);

   virtual void writeSQL(oracle::occi::AnyData& streamOCCI_);

};
```

```
/*************************************************************/
//  generated declarations for the PERSON object type.
//
//  Note the type name for the "name" attribute is MyFullName
//  and not CFullName, the "curr-addr" attribute is Ref< MyAddress >
//  and not Ref< CAddress >, and the "prev_addr_l" attribute is
//  vector< Ref< MyAddress > >.
/*************************************************************/

class CPerson : public oracle::occi::PObject {

protected:

   oracle::occi::Number ID;
   MyFullName * NAME;
   oracle::occi::Ref< MyAddress > CURR_ADDR;
   OCCI_STD_NAMESPACE::vector< oracle::occi::Ref< MyAddress > > PREV_ADDR_L;

public:

   void *operator new(size_t size);

   void *operator new(size_t size, const oracle::occi::Connection * sess,
      const OCCI_STD_NAMESPACE::string& table);

   OCCI_STD_NAMESPACE::string getSQLTypeName() const;

   CPerson();

   CPerson(void *ctxOCCI_) : oracle::occi::PObject (ctxOCCI_) { };

   static void *readSQL(void *ctxOCCI_);

   virtual void readSQL(oracle::occi::AnyData& streamOCCI_);

   static void writeSQL(void *objOCCI_, void *ctxOCCI_);

   virtual void writeSQL(oracle::occi::AnyData& streamOCCI_);

};
```

```
/************************************************************/
//  generated declarations for the STUDENT object type.
//
//  Note the parent class for CStudent is MyPerson and not
//  CPerson
/************************************************************/

class CStudent : public MyPerson {

protected:

   OCCI_STD_NAMESPACE::string SCHOOL_NAME;

public:

   void *operator new(size_t size);

   void *operator new(size_t size, const oracle::occi::Connection * sess,
      const OCCI_STD_NAMESPACE::string& table);

   OCCI_STD_NAMESPACE::string getSQLTypeName() const;

   CStudent();

   CStudent(void *ctxOCCI_) : MyPerson (ctxOCCI_) { };

   static void *readSQL(void *ctxOCCI_);

   virtual void readSQL(oracle::occi::AnyData& streamOCCI_);

   static void writeSQL(void *objOCCI_, void *ctxOCCI_);

   virtual void writeSQL(oracle::occi::AnyData& streamOCCI_);

};

#endif
```

メソッド実装は、次のとおりです。

```
#ifndef MYFILENAME_ORACLE
# include "myfilename.h"
#endif


/*****************************************************************/
//  generated method implementations for the FULL_NAME object type.
/*****************************************************************/

void *CFullName::operator new(size_t size)
{
  return oracle::occi::PObject::operator new(size);
}

void *CFullName::operator new(size_t size, const oracle::occi::Connection * sess,
  const OCCI_STD_NAMESPACE::string& table)
{
  return oracle::occi::PObject::operator new(size, sess, table,
            (char *) "SCOTT.FULL_NAME");
}

OCCI_STD_NAMESPACE::string CFullName::getSQLTypeName() const
{
  return OCCI_STD_NAMESPACE::string("SCOTT.FULL_NAME");
}

CFullName::CFullName()
{
}

void *CFullName::readSQL(void *ctxOCCI_)
{
  CFullName *objOCCI_ = new CFullName(ctxOCCI_);
  oracle::occi::AnyData streamOCCI_(ctxOCCI_);

  try
  {
    if (streamOCCI_.isNull())
      objOCCI_->setNull();
    else
      objOCCI_->readSQL(streamOCCI_);
  }
  catch (oracle::occi::SQLException& excep)
  {
```

```
    delete objOCCI_;
    excep.setErrorCtx(ctxOCCI_);
    return (void *)NULL;
  }
  return (void *)objOCCI_;
}

void CFullName::readSQL(oracle::occi::AnyData& streamOCCI_)
{
   FirstName = streamOCCI_.getString();
   LastName = streamOCCI_.getString();
}

void CFullName::writeSQL(void *objectOCCI_, void *ctxOCCI_)
{
  CFullName *objOCCI_ = (CFullName *) objectOCCI_;
  oracle::occi::AnyData streamOCCI_(ctxOCCI_);

  try
  {
    if (objOCCI_->isNull())
      streamOCCI_.setNull();
    else
      objOCCI_->writeSQL(streamOCCI_);
  }
  catch (oracle::occi::SQLException& excep)
  {
    excep.setErrorCtx(ctxOCCI_);
  }
  return;
}

void CFullName::writeSQL(oracle::occi::AnyData& streamOCCI_)
{
   streamOCCI_.setString(FirstName);
   streamOCCI_.setString(LastName);
}

/*---------------------------------------------------------------*/
//  generated method implementations for the ADDRESS object type.
/*---------------------------------------------------------------*/

void *CAddress::operator new(size_t size)
{
  return oracle::occi::PObject::operator new(size);
}
```

```
void *CAddress::operator new(size_t size, const oracle::occi::Connection * sess,
  const OCCI_STD_NAMESPACE::string& table)
{
  return oracle::occi::PObject::operator new(size, sess, table,
           (char *) "SCOTT.ADDRESS");
}

OCCI_STD_NAMESPACE::string CAddress::getSQLTypeName() const
{
  return OCCI_STD_NAMESPACE::string("SCOTT.ADDRESS");
}

CAddress::CAddress()
{
}

void *CAddress::readSQL(void *ctxOCCI_)
{
  CAddress *objOCCI_ = new CAddress(ctxOCCI_);
  oracle::occi::AnyData streamOCCI_(ctxOCCI_);

  try
  {
    if (streamOCCI_.isNull())
      objOCCI_->setNull();
    else
      objOCCI_->readSQL(streamOCCI_);
  }
  catch (oracle::occi::SQLException& excep)
  {
    delete objOCCI_;
    excep.setErrorCtx(ctxOCCI_);
    return (void *)NULL;
  }
  return (void *)objOCCI_;
}

void CAddress::readSQL(oracle::occi::AnyData& streamOCCI_)
{
   STATE = streamOCCI_.getString();
   ZIP = streamOCCI_.getString();
}
```

```
void CAddress::writeSQL(void *objectOCCI_, void *ctxOCCI_)
{
  CAddress *objOCCI_ = (CAddress *) objectOCCI_;
  oracle::occi::AnyData streamOCCI_(ctxOCCI_);

  try
  {
    if (objOCCI_->isNull())
      streamOCCI_.setNull();
    else
      objOCCI_->writeSQL(streamOCCI_);
  }
  catch (oracle::occi::SQLException& excep)
  {
    excep.setErrorCtx(ctxOCCI_);
  }
  return;
}

void CAddress::writeSQL(oracle::occi::AnyData& streamOCCI_)
{
   streamOCCI_.setString(STATE);
   streamOCCI_.setString(ZIP);
}

/*****************************************************************/
//  generated method implementations for the PERSON object type.
//
//  Note the type used in the casting in the readSQL method is
//  MyFullName and not CFullName.
/*****************************************************************/

void *CPerson::operator new(size_t size)
{
  return oracle::occi::PObject::operator new(size);
}

void *CPerson::operator new(size_t size, const oracle::occi::Connection * sess,
  const OCCI_STD_NAMESPACE::string& table)
{
  return oracle::occi::PObject::operator new(size, sess, table,
            (char *) "SCOTT.PERSON");
}
```

```
OCCI_STD_NAMESPACE::string CPerson::getSQLTypeName() const
{
  return OCCI_STD_NAMESPACE::string("SCOTT.PERSON");
}

CPerson::CPerson()
{
   NAME = (MyFullName *) 0;
}

void *CPerson::readSQL(void *ctxOCCI_)
{
  CPerson *objOCCI_ = new CPerson(ctxOCCI_);
  oracle::occi::AnyData streamOCCI_(ctxOCCI_);

  try
  {
    if (streamOCCI_.isNull())
      objOCCI_->setNull();
    else
      objOCCI_->readSQL(streamOCCI_);
  }
  catch (oracle::occi::SQLException& excep)
  {
    delete objOCCI_;
    excep.setErrorCtx(ctxOCCI_);
    return (void *)NULL;
  }
  return (void *)objOCCI_;
}

void CPerson::readSQL(oracle::occi::AnyData& streamOCCI_)
{
   ID = streamOCCI_.getNumber();
   NAME = (MyFullName *) streamOCCI_.getObject();
   CURR_ADDR = streamOCCI_.getRef();
   getVector(streamOCCI_, PREV_ADDR_L);
}

void CPerson::writeSQL(void *objectOCCI_, void *ctxOCCI_)
{
  CPerson *objOCCI_ = (CPerson *) objectOCCI_;
  oracle::occi::AnyData streamOCCI_(ctxOCCI_);
```

```
  try
  {
    if (objOCCI_->isNull())
      streamOCCI_.setNull();
    else
      objOCCI_->writeSQL(streamOCCI_);
  }
  catch (oracle::occi::SQLException& excep)
  {
    excep.setErrorCtx(ctxOCCI_);
  }
  return;
}

void CPerson::writeSQL(oracle::occi::AnyData& streamOCCI_)
{
   streamOCCI_.setNumber(ID);
   streamOCCI_.setObject(NAME);
   streamOCCI_.setRef(CURR_ADDR);
   setVector(streamOCCI_, PREV_ADDR_L);
}

/*****************************************************************/
//  generated method implementations for the STUDENT object type.
//
//  Note even though CStudent derives from MyPerson, the readSQL
//  and writeSQL methods called are those of CPerson.
/*****************************************************************/

void *CStudent::operator new(size_t size)
{
  return oracle::occi::PObject::operator new(size);
}

void *CStudent::operator new(size_t size, const oracle::occi::Connection * sess,
  const OCCI_STD_NAMESPACE::string& table)
{
  return oracle::occi::PObject::operator new(size, sess, table,
            (char *) "SCOTT.STUDENT");
}

OCCI_STD_NAMESPACE::string CStudent::getSQLTypeName() const
{
  return OCCI_STD_NAMESPACE::string("SCOTT.STUDENT");
}
```

```
CStudent::CStudent()
{
}

void *CStudent::readSQL(void *ctxOCCI_)
{
  CStudent *objOCCI_ = new CStudent(ctxOCCI_);
  oracle::occi::AnyData streamOCCI_(ctxOCCI_);

  try
  {
    if (streamOCCI_.isNull())
      objOCCI_->setNull();
    else
      objOCCI_->readSQL(streamOCCI_);
  }
  catch (oracle::occi::SQLException& excep)
  {
    delete objOCCI_;
    excep.setErrorCtx(ctxOCCI_);
    return (void *)NULL;
  }
  return (void *)objOCCI_;
}

void CStudent::readSQL(oracle::occi::AnyData& streamOCCI_)
{
   CPerson::readSQL(streamOCCI_);
   SCHOOL_NAME = streamOCCI_.getString();
}

void CStudent::writeSQL(void *objectOCCI_, void *ctxOCCI_)
{
  CStudent *objOCCI_ = (CStudent *) objectOCCI_;
  oracle::occi::AnyData streamOCCI_(ctxOCCI_);

  try
  {
    if (objOCCI_->isNull())
      streamOCCI_.setNull();
    else
      objOCCI_->writeSQL(streamOCCI_);
  }
  catch (oracle::occi::SQLException& excep)
  {
    excep.setErrorCtx(ctxOCCI_);
  }
```

```
  return;
}

void CStudent::writeSQL(oracle::occi::AnyData& streamOCCI_)
{
   CPerson::writeSQL(streamOCCI_);
   streamOCCI_.setString(SCHOOL_NAME);
}
```

ユーザー追加コードの引継ぎ

OTT 生成コードの機能を拡張するために、プログラマは OTT 生成ファイルにコードを追加する場合があります。OTT は、事前定義されたマーカー（タグ）を検索することによって、OTT 生成コードとユーザーが追加したコードを区別します。OTT では、OTT_USERCODE_START を「ユーザー・コード・マーカーの開始」、OTT_USERCODE_END を「ユーザー・コード・マーカーの終了」と認識します。

OTT マーカーをサポートするため、ユーザー・ブロックは次のように定義されます。

```
OTT_USERCODE_START + user added code + OTT_USERCODE_END
```

OTT マーカーのサポートによって、ユーザーが追加したブロックを複数の OTT 実行で引き継ぐことができます。

注意： OTT マーカーのサポートを使用するには、JDK バージョン 1.3 以上を使用する必要があります。

OTT マーカーのプロパティ

この項では、OTT マーカーのサポートのプロパティについて説明します。

1. ユーザーは、ファイルを生成するために最初に OTT を起動するときから、コマンドライン・オプション USE_MARKER=TRUE を使用する必要があります。
2. ユーザーは、マーカーを別の C++ 文と同様に配置する必要があります。マーカーは、コマンドライン・オプション USE_MARKER=TRUE の使用時に、OTT によって生成済みファイル内に次のように定義されます。

```
#ifndef OTT_USERCODE_START
# define OTT_USERCODE_START
#endif
#ifndef OTT_USERCODE_END
# define OTT_USERCODE_END
#endif
```

3. マーカーの OTT_USERCODE_START および OTT_USERCODE_END の前後には空白が必要です。
4. OTT は、次回のコード生成時に、マーカーに囲まれた指定のテキスト / コードをコピーします。

   ユーザーが変更したコード :

```
1  // --- modified generated code
2  OTT_USERCODE_START
3  // --- including "myfullname.h"
4  #ifndef MYFULLNAME_ORACLE
5  # include "myfullname.h"
6  #endif
7  OTT_USERCODE_END
8  // --- end of code addition
```

   引き継がれるコード :

```
1  OTT_USERCODE_START
2  // --- including "myfullname.h"
3  #ifndef MYFULLNAME_ORACLE
4  # include "myfullname.h"
5  #endif
6  OTT_USERCODE_END
```

   この例では、元のコードの 1 行目と 8 行目が引き継がれていません。
5. 次の例に示すように、データベースの TYPE ファイルまたは INTYPE ファイルが変更されると、OTT では、ユーザーが追加したコードを正しく引き継ぐことができません。
   a. ユーザーがタイプ名の大 / 小文字を変更すると、OTT では、ユーザーが INTYPE ファイル内のクラス名の大 / 小文字を変更する前にコードが関連付けられていたクラス名の検出に失敗します。

CASE=UPPER

```
TYPE employee
TRANSLATE SALARY$ AS salary
          DEPTNO AS department
TYPE ADDRESS
TYPE item
TYPE "Person"
TYPE PURCHASE_ORDER AS p_o
```

CASE=LOWER

```
 TYPE employee
TRANSLATE SALARY$ AS salary
        DEPTNO AS department
 TYPE ADDRESS
 TYPE item
 TYPE "Person"
 TYPE PURCHASE_ORDER AS p_o
```

b. ユーザーが別の名前でクラスを生成するように要求した場合（INTYPE ファイルの GENERATE AS 句）、OTT では、ユーザーが INTYPE ファイル内のクラス名を変更する前にコードが関連付けられていたクラス名の検出に失敗します。

```
CASE=LOWER
TYPE employee
   TRANSLATE SALARY$ AS salary
DEPTNO AS department
TYPE ADDRESS
TYPE item
TYPE "Person"
TYPE PURCHASE_ORDER AS p_o
```

```
CASE=LOWER
TYPE employee
   TRANSLATE SALARY$ AS salary
DEPTNO AS department
TYPE ADDRESS
TYPE item
TYPE "Person"
TYPE PURCHASE_ORDER AS
   purchase_order
```

6. .h ファイルまたは .cpp ファイルの解析中にエラーが発生した場合、OTT では、そのエラーをレポートし、ファイルをエラーが発生した状態のままにします。これによって、ユーザーは、レポートされたエラーを修正してから OTT に戻ることができます。

7. OTT では、次の場合にエラーが発生します。

- OTT_USERCODE_START に対応する OTT_USERCODE_END が検出されない場合
- マーカーがネストされている場合（OTT で、前の OTT_USERCODE_START に対応する OTT_USERCODE_END が検出される前に、次の OTT_USERCODE_START が検出された場合）
- OTT_USERCODE_START の前に OTT_USERCODE_END が検出された場合

マーカーを使用できる箇所

OTT マーカーを使用するには、ユーザーは、コマンドライン・オプション USE_MARKER=TRUE を使用し、OTT に対して、マーカーの使用をサポートする必要があることを、OTT の起動時に通知する必要があります。次に説明するように、ユーザーは、OTT マーカーを使用して、ユーザーが追加したコードを引き継ぐことができます。

1. .h ファイルにユーザー・コードを追加する場合。

a. グローバル・スコープにユーザー・コードを追加する場合。通常は、ユーザーが別のヘッダー・ファイルを組み込んだり宣言を転送する必要がある場合です。後のコード例を参照してください。

b. クラス宣言にユーザー・コードを追加する場合。OTT 生成のクラス宣言に、データ・メンバー用のプライベート・スコープとメソッド用のパブリック・スコープがある場合、またはデータ・メンバー用の保護付きスコープとメソッド用のパブリック・スコープがある場合です。ユーザー・ブロックは、OTT 生成のすべての宣言の後、いずれかのアクセス指定子内に追加できます。

次のコード例は、.h ファイル内でユーザー・コードを追加できる位置を示します。

```
#ifndef ...
# define ...

#ifndef OTT_USERCODE_START
# define OTT_USERCODE_START
#endif
#ifndef OTT_USERCODE_END
# define OTT_USERCODE_END
#endif

#ifndef OCCI_ORACLE
# include <occi.h>
#endif

OTT_USERCODE_START     // user added code
...
OTT_USERCODE_END

#ifndef ...            // OTT generated include
# include " ... "
#endif

OTT_USERCODE_START     // user added code
...
OTT_USERCODE_END

class <class_name_1> : public oracle::occi::PObject {
protected:
  // OTT generated data members
  ...

  OTT_USERCODE_START   // user added code
  ...                  // data member / method declaration /
  ...                  // inline method definition
  OTT_USERCODE_END

public:
  void *operator new(size_t size);
  void *operator new(size_t size, const oracle::occi::Connection * sess,
       const OCCI_STD_NAMESPACE::string& table);
  void *operator new(size_t, void *ctxOCCI_);
  OCCI_STD_NAMESPACE::string getSQLTypeName() const;
  ...
```

```
  OTT_USERCODE_START   // user added code
  ...                  // data member / method declaration /
  ...                  // inline method definition
  OTT_USERCODE_END
};

OTT_USERCODE_START     // user added code
...
OTT_USERCODE_END

class <class_name_2> : public oracle::occi::PObject {
  ...
  ...
};

OTT_USERCODE_START     // user added code
...
OTT_USERCODE_END

#endif                 // end of .h file
```

2. **.cpp ファイルにユーザー・コードを追加する場合。**OTT では、新規のユーザー定義メソッドを OTT マーカー内に追加できます。ユーザー・ブロックは、ファイルの先頭部分で、インクルード（include）の直後、かつ OTT 生成メソッドの定義の前に追加する必要があります。OTT 生成のインクルード（include）が複数ある場合、ユーザー・コードは、それらの OTT 生成のインクルード（include）の間にも追加できます。.cpp ファイル内のこれ以外の位置に追加されたユーザー・コードは引き継がれません。

   次のコード例は、.cpp ファイル内でユーザー・コードを追加できる位置を示します。

```
#ifndef OTT_USERCODE_START
# define OTT_USERCODE_START
#endif

#ifndef OTT_USERCODE_END
# define OTT_USERCODE_END
#endif

#ifndef ...
# include " ... "
#endif

OTT_USERCODE_START    // user added code
...
...
OTT_USERCODE_END
```

```
#ifndef ...
# include " ... "
#endif

OTT_USERCODE_START    // user added code
...
...
OTT_USERCODE_END

/*************************************************************/
//  generated method implementations for the ... object type.
/*************************************************************/

void *<class_name_1>::operator new(size_t size)
{
  return oracle::occi::PObject::operator new(size);
}

...

// end of .cpp file
```

OTT マーカーの使用方法を示すコード例

この項では、GENERATE AS 句の使用で発生した問題を OTT マーカーを使用して解決するコード例を示します。

この例では、GENERATE AS 句が使用されているため、ユーザー拡張クラスを意図した FULL_NAME_O が FullName として生成されています。FullName は PERSON_O 句の属性であるため、FullName のクラス宣言が検出されない場合は、コンパイラでエラーが発生します。

マーカーのサポートによって、ユーザーは、必要なヘッダー・ファイル、転送宣言またはクラス宣言を追加できるため、ユーザーが追加したブロックは OTT の複数の実行で引き継がれます。

この例では、FullName のクラス宣言は mdemo1.h ファイルに格納されています。

OTT を起動するコマンドラインは次のとおりです。

```
ott case=same userid=scott/tiger code=cpp intype=mdemo1.typ
hfile=mdemo1.h cppfile=mdemo1o.cpp use_marker=true
```

作成コマンドは次のとおりです。

```
make -f demo_rdbms.mk mdemo1
```

このデモ・プログラムで使用するファイルは次のとおりです。

- 「mdemo1.sql」（7-65 ページ）は、型や表などを作成するための SQL です。
- 「mdemo1.typ」（7-66 ページ）は、INTYPE ファイルです。
- 「mdemo1.h」（7-66 ページ）は、ユーザーが追加したコードを含む OTT 生成のヘッダー・ファイルです。
- 「mdemo1o.cpp」（7-70 ページ）は、ユーザーが追加したコードを含む OTT 生成の .cpp ファイルです。
- 「mdemo1m.cpp」（7-77 ページ）は、OTT 生成のマップ・ファイルです。
- 「mdemo1m.h」（7-78 ページ）は、マップ・ファイル用の OTT 生成のヘッダー・ファイルです。
- 「mymdemo1.h」（7-78 ページ）は、ユーザー定義のヘッダー・ファイルです。
- 「mdemo1.cpp」（7-78 ページ）は、ユーザー定義のメイン・プログラムです。

mdemo1.sql

```
// ------------------------------------------------------
//  mdemo1.sql : SQLs to create type, tables, and so on.
// ------------------------------------------------------
connect scott/tiger;

DROP TABLE PERSON_TAB;
DROP TABLE ADDR_TAB;
DROP TYPE PERSON_O;
DROP TYPE ADDRESS_O;
DROP TYPE FULL_NAME_O;

CREATE TYPE ADDRESS_O AS OBJECT (
   "state" CHAR(20),
   "zip" CHAR(20)
)
/
CREATE TABLE ADDR_TAB OF ADDRESS_O;

CREATE TYPE FULL_NAME_O AS OBJECT (
   "first_name" CHAR(20),
   "last_name" CHAR(20)
)
/
CREATE TYPE PERSON_O AS OBJECT (
   "id" integer,
   "name" FULL_NAME_O,
   "addr" REF ADDRESS_O
```

```
)
/
CREATE TABLE PERSON_TAB OF PERSON_O;

QUIT;
```

mdemo1.typ

```
// -------------------------
//  mdemo1.typ : INTYPE file
// -------------------------
CASE=SAME
MAPFILE=mdemo1m.cpp
TYPE FULL_NAME_O
 GENERATE FULL_NAME_O AS FullName
TYPE ADDRESS_O
TYPE PERSON_O
 GENERATE PERSON_O AS Person
```

mdemo1.h

```
// ------------------------------------------------------------
//  mdemo1.h   : OTT generated header file with user addeed code
// ------------------------------------------------------------
#ifndef MDEMO1_ORACLE
# define MDEMO1_ORACLE

#ifndef OTT_USERCODE_START
# define OTT_USERCODE_START
#endif

#ifndef OTT_USERCODE_END
# define OTT_USERCODE_END
#endif

#ifndef OCCI_ORACLE
# include <occi.h>
#endif

OTT_USERCODE_START
#include "mymdemo1.h"

OTT_USERCODE_END
```

```
 /*************************************************************/
 //  generated declarations for the ADDRESS_O object type.
 /*************************************************************/

 class ADDRESS_O : public oracle::occi::PObject {

 protected:

    OCCI_STD_NAMESPACE::string state;
    OCCI_STD_NAMESPACE::string zip;

 public:

void *operator new(size_t size);

void *operator new(size_t size, const oracle::occi::Connection * sess,
   const OCCI_STD_NAMESPACE::string& table);

void *operator new(size_t, void *ctxOCCI_);

OCCI_STD_NAMESPACE::string getSQLTypeName() const;

ADDRESS_O();

ADDRESS_O(void *ctxOCCI_) : oracle::occi::PObject (ctxOCCI_) { };

    static void *readSQL(void *ctxOCCI_);

    virtual void readSQL(oracle::occi::AnyData& streamOCCI_);

    static void writeSQL(void *objOCCI_, void *ctxOCCI_);

    virtual void writeSQL(oracle::occi::AnyData& streamOCCI_);

 OTT_USERCODE_START

     ADDRESS_O(string state_i, string zip_i);
     void displayInfo();

 OTT_USERCODE_END

 };
```

```
    /*************************************************************/
    //  generated declarations for the FULL_NAME_O object type.
    /*************************************************************/

        class FULL_NAME_O : public oracle::occi::PObject {

        protected:

           OCCI_STD_NAMESPACE::string first_name;
           OCCI_STD_NAMESPACE::string last_name;

        public:

           void *operator new(size_t size);

           void *operator new(size_t size, const oracle::occi::Connection * sess,
              const OCCI_STD_NAMESPACE::string& table);

           void *operator new(size_t, void *ctxOCCI_);

           OCCI_STD_NAMESPACE::string getSQLTypeName() const;

           FULL_NAME_O();

           FULL_NAME_O(void *ctxOCCI_) : oracle::occi::PObject (ctxOCCI_) { };

           static void *readSQL(void *ctxOCCI_);

           virtual void readSQL(oracle::occi::AnyData& streamOCCI_);

           static void writeSQL(void *objOCCI_, void *ctxOCCI_);

           virtual void writeSQL(oracle::occi::AnyData& streamOCCI_);

        };

        OTT_USERCODE_START

        class FullName : public FULL_NAME_O {
        public:
          FullName(string FirstName, string LastName);

  void displayInfo();
  const string getFirstName() const { return first_name;}
} ;
```

```
OTT_USERCODE_END

/*************************************************************/
//  generated declarations for the PERSON_O object type.
/*************************************************************/

class PERSON_O : public oracle::occi::PObject {

protected:

   oracle::occi::Number id;
   FullName * name;
   oracle::occi::Ref< ADDRESS_O > addr;

public:

   void *operator new(size_t size);

   void *operator new(size_t size, const oracle::occi::Connection * sess,
      const OCCI_STD_NAMESPACE::string& table);

   void *operator new(size_t, void *ctxOCCI_);

   OCCI_STD_NAMESPACE::string getSQLTypeName() const;

   PERSON_O();

   PERSON_O(void *ctxOCCI_) : oracle::occi::PObject (ctxOCCI_) { };

   static void *readSQL(void *ctxOCCI_);

   virtual void readSQL(oracle::occi::AnyData& streamOCCI_);

   static void writeSQL(void *objOCCI_, void *ctxOCCI_);

   virtual void writeSQL(oracle::occi::AnyData& streamOCCI_);

};

OTT_USERCODE_START
```

```
class Person : public PERSON_O {
public:
  Person(int id_i,
         FullName* name_i,
         Ref<ADDRESS_O>& addr_i);
  void move(const Ref<ADDRESS_O>& new_addr);
  void displayInfo();
};

OTT_USERCODE_END

#endif
```

mdemo1o.cpp

```
// ------------------------------------------------------------
//  mdemo1o.cpp : OTT generated .cpp file with user added code
// ------------------------------------------------------------
#ifndef OTT_USERCODE_START
# define OTT_USERCODE_START
#endif

#ifndef OTT_USERCODE_END
# define OTT_USERCODE_END
#endif

#ifndef MDEMO1_ORACLE
# include "mdemo1.h"
#endif

OTT_USERCODE_START

// initialize FullName
FullName::FullName(string FirstName, string LastName)
{
  first_name = FirstName;
  last_name = LastName;
}

// display all the information in FullName
void FullName::displayInfo()
{
  cout << "FIRST NAME is " << first_name << endl;
  cout << "LAST NAME is " << last_name << endl;
}
```

```
// initialize ADDRESS_O
ADDRESS_O::ADDRESS_O(string state_i, string zip_i)
{
 state = state_i;
 zip = zip_i;
}

// display all the information in ADDRESS_O
void ADDRESS_O::displayInfo()
{
  cout << "STATE is " << state << endl;
  cout << "ZIP is " << zip << endl;
}

// initialize Person
Person::Person(int id_i,
               FullName *name_i,
               Ref<ADDRESS_O>& addr_i)
{
  id = id_i;
  name = name_i;
  addr =addr_i ;
}

// Move Person from curr_addr to new_addr
void Person::move(const Ref<ADDRESS_O>& new_addr)
{
  addr = new_addr;
  this->markModified();   // mark the object as dirty
}

// Display all the information of Person
void Person::displayInfo() {
  cout << "ID is " << (int)id << endl;
  name->displayInfo();

  // de-referencing the Ref attribute using -> operator
  addr->displayInfo();

}

OTT_USERCODE_END
```

```
/************************************************************************/
//  generated method implementations for the ADDRESS_O object type.
/************************************************************************/

void *ADDRESS_O::operator new(size_t size)
{
  return oracle::occi::PObject::operator new(size);
}

void *ADDRESS_O::operator new(size_t size, const oracle::occi::Connection *
sess,
  const OCCI_STD_NAMESPACE::string& table)
{
  return oracle::occi::PObject::operator new(size, sess, table,
            (char *) "MDEMO1.ADDRESS_O");
}

void *ADDRESS_O::operator new(size_t size, void *ctxOCCI_)
{
 return oracle::occi::PObject::operator new(size, ctxOCCI_);
}

OCCI_STD_NAMESPACE::string ADDRESS_O::getSQLTypeName() const
{
  return OCCI_STD_NAMESPACE::string("MDEMO1.ADDRESS_O");
}

ADDRESS_O::ADDRESS_O()
{
}

void *ADDRESS_O::readSQL(void *ctxOCCI_)
{
  ADDRESS_O *objOCCI_ = new(ctxOCCI_) ADDRESS_O(ctxOCCI_);
  oracle::occi::AnyData streamOCCI_(ctxOCCI_);

  try
  {
    if (streamOCCI_.isNull())
      objOCCI_->setNull();
    else
      objOCCI_->readSQL(streamOCCI_);
  }
  catch (oracle::occi::SQLException& excep)
  {
    delete objOCCI_;
    excep.setErrorCtx(ctxOCCI_);
```

```
    return (void *)NULL;
  }
  return (void *)objOCCI_;
}

void ADDRESS_O::readSQL(oracle::occi::AnyData& streamOCCI_)
{
   state = streamOCCI_.getString();
   zip = streamOCCI_.getString();
}

void ADDRESS_O::writeSQL(void *objectOCCI_, void *ctxOCCI_)
{
  ADDRESS_O *objOCCI_ = (ADDRESS_O *) objectOCCI_;
  oracle::occi::AnyData streamOCCI_(ctxOCCI_);

  try
  {
    if (objOCCI_->isNull())
      streamOCCI_.setNull();
    else
      objOCCI_->writeSQL(streamOCCI_);
  }
  catch (oracle::occi::SQLException& excep)
  {
    excep.setErrorCtx(ctxOCCI_);
  }
  return;
}

void ADDRESS_O::writeSQL(oracle::occi::AnyData& streamOCCI_)
{
   streamOCCI_.setString(state);
   streamOCCI_.setString(zip);
}

/*---------------------------------------------------------------*/
//  generated method implementations for the FULL_NAME_O object type.
/*---------------------------------------------------------------*/

void *FULL_NAME_O::operator new(size_t size)
{
  return oracle::occi::PObject::operator new(size);
}
```

```
void *FULL_NAME_O::operator new(size_t size, const oracle::occi::Connection *
sess,
  const OCCI_STD_NAMESPACE::string& table)
{
  return oracle::occi::PObject::operator new(size, sess, table,
            (char *) "MDEMO1.FULL_NAME_O");
}

void *FULL_NAME_O::operator new(size_t size, void *ctxOCCI_)
{
 return oracle::occi::PObject::operator new(size, ctxOCCI_);
}

OCCI_STD_NAMESPACE::string FULL_NAME_O::getSQLTypeName() const
{
  return OCCI_STD_NAMESPACE::string("MDEMO1.FULL_NAME_O");
}

FULL_NAME_O::FULL_NAME_O()
{
}

void *FULL_NAME_O::readSQL(void *ctxOCCI_)
{
  FULL_NAME_O *objOCCI_ = new(ctxOCCI_) FULL_NAME_O(ctxOCCI_);
  oracle::occi::AnyData streamOCCI_(ctxOCCI_);

  try
  {
    if (streamOCCI_.isNull())
      objOCCI_->setNull();
    else
      objOCCI_->readSQL(streamOCCI_);
  }
  catch (oracle::occi::SQLException& excep)
  {
    delete objOCCI_;
    excep.setErrorCtx(ctxOCCI_);
    return (void *)NULL;
  }
  return (void *)objOCCI_;
}
```

```
void FULL_NAME_O::readSQL(oracle::occi::AnyData& streamOCCI_)
{
   first_name = streamOCCI_.getString();
   last_name = streamOCCI_.getString();
}

void FULL_NAME_O::writeSQL(void *objectOCCI_, void *ctxOCCI_)
{
  FULL_NAME_O *objOCCI_ = (FULL_NAME_O *) objectOCCI_;
  oracle::occi::AnyData streamOCCI_(ctxOCCI_);

  try
  {
    if (objOCCI_->isNull())
      streamOCCI_.setNull();
    else
      objOCCI_->writeSQL(streamOCCI_);
  }
  catch (oracle::occi::SQLException& excep)
  {
    excep.setErrorCtx(ctxOCCI_);
  }
  return;
}

void FULL_NAME_O::writeSQL(oracle::occi::AnyData& streamOCCI_)
{
   streamOCCI_.setString(first_name);
   streamOCCI_.setString(last_name);
}

/*---------------------------------------------------------------*/
//  generated method implementations for the PERSON_O object type.
/*---------------------------------------------------------------*/

void *PERSON_O::operator new(size_t size)
{
  return oracle::occi::PObject::operator new(size);
}

void *PERSON_O::operator new(size_t size, const oracle::occi::Connection * sess,
  const OCCI_STD_NAMESPACE::string& table)
{
  return oracle::occi::PObject::operator new(size, sess, table,
            (char *) "MDEMO1.PERSON_O");
}
```

```
void *PERSON_O::operator new(size_t size, void *ctxOCCI_)
{
 return oracle::occi::PObject::operator new(size, ctxOCCI_);
}

OCCI_STD_NAMESPACE::string PERSON_O::getSQLTypeName() const
{
  return OCCI_STD_NAMESPACE::string("MDEMO1.PERSON_O");
}

PERSON_O::PERSON_O()
{
   name = (FullName *) 0;
}

void *PERSON_O::readSQL(void *ctxOCCI_)
{
  PERSON_O *objOCCI_ = new(ctxOCCI_) PERSON_O(ctxOCCI_);
  oracle::occi::AnyData streamOCCI_(ctxOCCI_);

  try
  {
    if (streamOCCI_.isNull())
      objOCCI_->setNull();
    else
      objOCCI_->readSQL(streamOCCI_);
  }
  catch (oracle::occi::SQLException& excep)
  {
    delete objOCCI_;
    excep.setErrorCtx(ctxOCCI_);
    return (void *)NULL;
  }
  return (void *)objOCCI_;
}

void PERSON_O::readSQL(oracle::occi::AnyData& streamOCCI_)
{
   id = streamOCCI_.getNumber();
   name = (FullName *) streamOCCI_.getObject();
   addr = streamOCCI_.getRef();
}
```

```
void PERSON_O::writeSQL(void *objectOCCI_, void *ctxOCCI_)
{
  PERSON_O *objOCCI_ = (PERSON_O *) objectOCCI_;
  oracle::occi::AnyData streamOCCI_(ctxOCCI_);

  try
  {
    if (objOCCI_->isNull())
      streamOCCI_.setNull();
    else
      objOCCI_->writeSQL(streamOCCI_);
  }
  catch (oracle::occi::SQLException& excep)
  {
    excep.setErrorCtx(ctxOCCI_);
  }
  return;
}

void PERSON_O::writeSQL(oracle::occi::AnyData& streamOCCI_)
{
   streamOCCI_.setNumber(id);
   streamOCCI_.setObject(name);
   streamOCCI_.setRef(addr);
}
```

mdemo1m.cpp

```
// ------------------------------------
//  mdemo1m.cpp: OTT generated map file
// ------------------------------------
#ifndef MDEMO1M_ORACLE
# include "mdemo1m.h"
#endif

void mdemo1m(oracle::occi::Environment* envOCCI_)
{
  oracle::occi::Map *mapOCCI_ = envOCCI_->getMap();
  mapOCCI_->put("MDEMO1.ADDRESS_O", ADDRESS_O::readSQL, ADDRESS_O::writeSQL);
  mapOCCI_->put("MDEMO1.FULL_NAME_O", FULL_NAME_O::readSQL, FULL_NAME_O::writeSQL);
  mapOCCI_->put("MDEMO1.PERSON_O", PERSON_O::readSQL, PERSON_O::writeSQL);
}
```

mdemo1m.h

```
// ------------------------------------------------
//  mdemo1m.h : OTT generated header file for map file
// ------------------------------------------------
#ifndef MDEMO1M_ORACLE
# define MDEMO1M_ORACLE

#ifndef OCCI_ORACLE
# include <occi.h>
#endif

#ifndef MDEMO1_ORACLE
# include "mdemo1.h"
#endif

void mdemo1m(oracle::occi::Environment* envOCCI_);

#endif
```

mymdemo1.h

```
// ---------------------------------------
//  mymdemo1.h : User defined header file
// ---------------------------------------
#include <iostream>

#define USERNAME "scott"
#define PASSWORD "tiger"

using namespace oracle::occi;
using namespace std;
```

mdemo1.cpp

```
// ----------------------------------------
//  mdemo1.cpp : User defined main program
// ----------------------------------------
#include "mdemo1.h"
#include "mdemo1m.h"

// global Oracle variables

Environment     *env;
Connection      *conn;
Statement       *stmt;
ResultSet       *rs;
```

```
void initialize() {

  // Create environment
  env = Environment::createEnvironment(Environment::OBJECT);

  // Call the OTT generated function to register the mappings
  mdemo1m(env);

  // Create Connection
  conn = env->createConnection( USERNAME, PASSWORD );

  // Create a statement
  stmt = conn->createStatement();
}

void terminate() {

  // Terminate statement
  conn->terminateStatement(stmt);

  // Terminate connection
  env->terminateConnection(conn);

  // Terminate environment
  Environment::terminateEnvironment(env);
}

/* Do the work.
   The environment is set up.
   A single new entry is created in the Address table.
   Then it is committed.
*/
void dowrite() {

  // Create an Address
  ADDRESS_O *addr1 = new(conn, "ADDR_TAB") ADDRESS_O("GE", "1211");
  Ref<ADDRESS_O> addr1_ref = (Ref<ADDRESS_O>) addr1->getRef();

  // Create joe black
  FullName *name1= new FullName("Joe", "Black");

  Person *person1 =
    new(conn, "PERSON_TAB") Person(1,name1,addr1_ref);
  Ref<Person> person1_ref = (Ref<Person>) person1->getRef();
```

```
  // Display, using reference
  cout<<"-------------------"<<endl;
  person1_ref->displayInfo();
  cout<<"-------------------"<<endl;

  // Commit the changes
  conn->commit();

  // Clean up
  delete name1;
}

void doread()
{
  // Retrieve joe black
  string sel_joe = "SELECT REF(p) from person_tab p where \"id\" = 1";

  rs =stmt->executeQuery (sel_joe);
  rs =stmt->getResultSet ();

  // Get reference
  rs->next();
  Ref<Person> joe_ref = (Ref<Person>) rs->getRef(1);

  // Display, using reference
  cout<<"-------------------"<<endl;
  joe_ref->displayInfo();
  cout<<"-------------------"<<endl;

}

int main() {

  try {
    initialize();
    dowrite();
    doread();
    terminate();
  }
  catch (SQLException &e) {
    cout << "SQL exception :" << e.getMessage() << endl;
  }

  return 0;
}
```

プログラムの出力

```
------------------
ID is 1
FIRST NAME is Joe
LAST NAME is Black
STATE is GE
ZIP is 1211
------------------
------------------
ID is 1
FIRST NAME is Joe
LAST NAME is Black
STATE is GE
ZIP is 1211
------------------
```

OCCI アプリケーションの例

この OCCI アプリケーションの例では、OTT で生成した C++ クラスを拡張し、継承オブジェクト型を変換します。このアプリケーションの各クラスには、クラス・オブジェクトを初期化するコンストラクタとオブジェクトの属性に割り当てられている値を表示するためのメソッドが組み込まれています。また、MyPerson クラスには、curr_addr 属性を変更するためのメソッドがあります。ここにあるすべてのクラスは、生成されたクラスから導出されたクラスです。

次のコード例のように、OCCI アプリケーションに必要な型と表を作成します。

```
connect scott/tiger
create type full_name as object (first_name char(20), last_name char(20));
create type address as object (state char(20), zip char(20));
create type address_tab as varray(3) of ref address;
create type person as object (id number, name full_name,
    curr_addr ref address, prev_addr_l address_tab) not final;
create type student under person (school_name char(20));

/* tables needed in the user-written occi application */
create table addr_tab of address;
create table person_tab of person;
create table student_tab of student;
```

OTT ユーティリティ用の INTYPE ファイルには、次の情報が記述されています。

```
CASE = SAME
MAPFILE = registerMappings.cpp

TYPE FULL_NAME GENERATE CFullName AS MyFullName
HFILE=cfullname.h
CPPFILE=cfullname.cpp
  TRANSLATE first_name as FirstName
            last_name as LastName

TYPE ADDRESS GENERATE CAddress AS MyAddress
HFILE=caddress.h
CPPFILE=caddress.cpp

TYPE PERSON GENERATE CPerson AS MyPerson
HFILE=cperson.h
CPPFILE=cperson.cpp

TYPE STUDENT GENERATE CStudent AS MyStudent
HFILE=cstudent.h
CPPFILE=cstudent.cpp
```

注意： PERSON は拡張クラスであり、PFGRFDENT のベース・クラスであるため、PERSON と PFGRFDENT は個別のファイルに生成する必要があります。

OTT ユーティリティを起動するには、次のコマンドラインの文を使用します。

```
ott userid=scott/tiger code=cpp attraccess=private intype=demoin.typ
outtype=demoout.typ
```

注意： 属性へのアクセスにアクセッサとミューテータが使用されるため、attraccess=private が指定されています。

この例の OTT ユーティリティで生成されるファイルについては、次の項を参照してください。

- FULL_NAME オブジェクト型の宣言の例 : cfullname.h
- ADDRESS オブジェクト型の宣言の例 : caddress.h
- PERSON オブジェクト型の宣言の例 : cperson.h
- STUDENT オブジェクト型の宣言の例 : cstudent.h

FULL_NAME オブジェクト型の宣言の例 : cfullname.h

```
#ifndef CFULLNAME_ORACLE
# define CFULLNAME_ORACLE

#ifndef OCCI_ORACLE
# include <occi.h>
#endif


/************************************************************/
//  generated declarations for the FULL_NAME object type.
/************************************************************/

class CFullName : public oracle::occi::PObject {

private:

   OCCI_STD_NAMESPACE::string FirstName;
   OCCI_STD_NAMESPACE::string LastName;

public:

   OCCI_STD_NAMESPACE::string getFirstname() const;

   void setFirstname(const OCCI_STD_NAMESPACE::string &value);
```

```
   OCCI_STD_NAMESPACE::string getLastname() const;

   void setLastname(const OCCI_STD_NAMESPACE::string &value);

   void *operator new(size_t size);

   void *operator new(size_t size, const oracle::occi::Connection * sess,
      const OCCI_STD_NAMESPACE::string& table);

   OCCI_STD_NAMESPACE::string getSQLTypeName() const;

   CFullName();

   CFullName(void *ctxOCCI_) : oracle::occi::PObject (ctxOCCI_) { };

   static void *readSQL(void *ctxOCCI_);

   virtual void readSQL(oracle::occi::AnyData& streamOCCI_);

   static void writeSQL(void *objOCCI_, void *ctxOCCI_);

   virtual void writeSQL(oracle::occi::AnyData& streamOCCI_);

};

#endif
```

ADDRESS オブジェクト型の宣言の例 : caddress.h

```
#ifndef CADDRESS_ORACLE
# define CADDRESS_ORACLE

#ifndef OCCI_ORACLE
# include <occi.h>
#endif


/************************************************************/
//  generated declarations for the ADDRESS object type.
/************************************************************/

class CAddress : public oracle::occi::PObject {

private:
```

```
   OCCI_STD_NAMESPACE::string STATE;
   OCCI_STD_NAMESPACE::string ZIP;

public:

   OCCI_STD_NAMESPACE::string getState() const;

   void setState(const OCCI_STD_NAMESPACE::string &value);

   OCCI_STD_NAMESPACE::string getZip() const;

   void setZip(const OCCI_STD_NAMESPACE::string &value);

   void *operator new(size_t size);

   void *operator new(size_t size, const oracle::occi::Connection * sess,
      const OCCI_STD_NAMESPACE::string& table);

   OCCI_STD_NAMESPACE::string getSQLTypeName() const;

   CAddress();

   CAddress(void *ctxOCCI_) : oracle::occi::PObject (ctxOCCI_) { };

   static void *readSQL(void *ctxOCCI_);

   virtual void readSQL(oracle::occi::AnyData& streamOCCI_);

   static void writeSQL(void *objOCCI_, void *ctxOCCI_);

   virtual void writeSQL(oracle::occi::AnyData& streamOCCI_);

};

#endif
```

PERSON オブジェクト型の宣言の例 : cperson.h

```
#ifndef CPERSON_ORACLE
# define CPERSON_ORACLE

#ifndef OCCI_ORACLE
# include <occi.h>
#endif
```

```
#ifndef MYFULLNAME_ORACLE
# include "myfullname.h"
#endif

#ifndef MYADDRESS_ORACLE
# include "myaddress.h"
#endif


/*****************************************************************/
//  generated declarations for the PERSON object type.
/*****************************************************************/

class CPerson : public oracle::occi::PObject {

private:

   oracle::occi::Number ID;
   MyFullName * NAME;
   oracle::occi::Ref< MyAddress > CURR_ADDR;
   OCCI_STD_NAMESPACE::vector< oracle::occi::Ref< MyAddress > > PREV_ADDR_L;

public:

   oracle::occi::Number getId() const;

   void setId(const oracle::occi::Number &value);

   MyFullName * getName() const;

   void setName(MyFullName * value);

   oracle::occi::Ref< MyAddress > getCurr_addr() const;

   void setCurr_addr(const oracle::occi::Ref< MyAddress > &value);

   OCCI_STD_NAMESPACE::vector< oracle::occi::Ref< MyAddress > >& getPrev_addr_l();

   const OCCI_STD_NAMESPACE::vector< oracle::occi::Ref< MyAddress > >& getPrev_addr_
l() const;

   void setPrev_addr_l(const OCCI_STD_NAMESPACE::vector< oracle::occi::Ref<
MyAddress > > &value);

   void *operator new(size_t size);
```

```
   void *operator new(size_t size, const oracle::occi::Connection * sess,
      const OCCI_STD_NAMESPACE::string& table);

   OCCI_STD_NAMESPACE::string getSQLTypeName() const;

   CPerson();

   CPerson(void *ctxOCCI_) : oracle::occi::PObject (ctxOCCI_) { };

   static void *readSQL(void *ctxOCCI_);

   virtual void readSQL(oracle::occi::AnyData& streamOCCI_);

   static void writeSQL(void *objOCCI_, void *ctxOCCI_);

   virtual void writeSQL(oracle::occi::AnyData& streamOCCI_);

};

#endif
```

STUDENT オブジェクト型の宣言の例 : cstudent.h

```
#ifndef CSTUDENT_ORACLE
# define CSTUDENT_ORACLE

#ifndef OCCI_ORACLE
# include <occi.h>
#endif

#ifndef myPERSON_ORACLE
# include "myperson.h"
#endif


/*************************************************************/
//  generated declarations for the STUDENT object type.
/*************************************************************/

class CStudent : public MyPerson {

private:

   OCCI_STD_NAMESPACE::string SCHOOL_NAME;

public:
```

```
   OCCI_STD_NAMESPACE::string getSchool_name() const;

   void setSchool_name(const OCCI_STD_NAMESPACE::string &value);

   void *operator new(size_t size);

   void *operator new(size_t size, const oracle::occi::Connection * sess,
      const OCCI_STD_NAMESPACE::string& table);

   OCCI_STD_NAMESPACE::string getSQLTypeName() const;

   CStudent();

   CStudent(void *ctxOCCI_) : MyPerson (ctxOCCI_) { };

   static void *readSQL(void *ctxOCCI_);

   virtual void readSQL(oracle::occi::AnyData& streamOCCI_);

   static void writeSQL(void *objOCCI_, void *ctxOCCI_);

   virtual void writeSQL(oracle::occi::AnyData& streamOCCI_);

};

#endif
```

FULL_NAME オブジェクト型の C++ ソース・ファイルの例 : cfullname.cpp

```
#ifndef CFULLNAME_ORACLE
# include "cfullname.h"
#endif


/* ---------------------------------------------------------------- */
//  generated method implementations for the FULL_NAME object type.
/* ---------------------------------------------------------------- */

OCCI_STD_NAMESPACE::string CFullName::getFirstname() const
{
  return FirstName;
}
```

```
void CFullName::setFirstname(const OCCI_STD_NAMESPACE::string &value)
{
  FirstName = value;
}

OCCI_STD_NAMESPACE::string CFullName::getLastname() const
{
  return LastName;
}

void CFullName::setLastname(const OCCI_STD_NAMESPACE::string &value)
{
  LastName = value;
}

void *CFullName::operator new(size_t size)
{
  return oracle::occi::PObject::operator new(size);
}

void *CFullName::operator new(size_t size, const oracle::occi::Connection * sess,
  const OCCI_STD_NAMESPACE::string& table)
{
  return oracle::occi::PObject::operator new(size, sess, table,
            (char *) "SCOTT.FULL_NAME");
}

OCCI_STD_NAMESPACE::string CFullName::getSQLTypeName() const
{
  return OCCI_STD_NAMESPACE::string("SCOTT.FULL_NAME");
}

CFullName::CFullName()
{
}

void *CFullName::readSQL(void *ctxOCCI_)
{
  CFullName *objOCCI_ = new CFullName(ctxOCCI_);
  oracle::occi::AnyData streamOCCI_(ctxOCCI_);

  try
  {
    if (streamOCCI_.isNull())
      objOCCI_->setNull();
    else
      objOCCI_->readSQL(streamOCCI_);
```

```
  }
  catch (oracle::occi::SQLException& excep)
  {
    delete objOCCI_;
    excep.setErrorCtx(ctxOCCI_);
    return (void *)NULL;
  }
  return (void *)objOCCI_;
}

void CFullName::readSQL(oracle::occi::AnyData& streamOCCI_)
{
   FirstName = streamOCCI_.getString();
   LastName = streamOCCI_.getString();
}

void CFullName::writeSQL(void *objectOCCI_, void *ctxOCCI_)
{
  CFullName *objOCCI_ = (CFullName *) objectOCCI_;
  oracle::occi::AnyData streamOCCI_(ctxOCCI_);

  try
  {
    if (objOCCI_->isNull())
      streamOCCI_.setNull();
    else
      objOCCI_->writeSQL(streamOCCI_);
  }
  catch (oracle::occi::SQLException& excep)
  {
    excep.setErrorCtx(ctxOCCI_);
  }
  return;
}

void CFullName::writeSQL(oracle::occi::AnyData& streamOCCI_)
{
   streamOCCI_.setString(FirstName);
   streamOCCI_.setString(LastName);
}
```

ADDRESS オブジェクト型の C++ ソース・ファイルの例 : caddress.cpp

```
#ifndef CADDRESS_ORACLE
# include "caddress.h"
#endif


/*****************************************************************/
//  generated method implementations for the ADDRESS object type.
/*****************************************************************/

OCCI_STD_NAMESPACE::string CAddress::getState() const
{
  return STATE;
}

void CAddress::setState(const OCCI_STD_NAMESPACE::string &value)
{
  STATE = value;
}

OCCI_STD_NAMESPACE::string CAddress::getZip() const
{
  return ZIP;
}

void CAddress::setZip(const OCCI_STD_NAMESPACE::string &value)
{
  ZIP = value;
}

void *CAddress::operator new(size_t size)
{
  return oracle::occi::PObject::operator new(size);
}

void *CAddress::operator new(size_t size, const oracle::occi::Connection * sess,
  const OCCI_STD_NAMESPACE::string& table)
{
  return oracle::occi::PObject::operator new(size, sess, table,
            (char *) "SCOTT.ADDRESS");
}

OCCI_STD_NAMESPACE::string CAddress::getSQLTypeName() const
{
  return OCCI_STD_NAMESPACE::string("SCOTT.ADDRESS");
}
```

```
CAddress::CAddress()
{
}

void *CAddress::readSQL(void *ctxOCCI_)
{
  CAddress *objOCCI_ = new CAddress(ctxOCCI_);
  oracle::occi::AnyData streamOCCI_(ctxOCCI_);

  try
  {
    if (streamOCCI_.isNull())
      objOCCI_->setNull();
    else
      objOCCI_->readSQL(streamOCCI_);
  }
  catch (oracle::occi::SQLException& excep)
  {
    delete objOCCI_;
    excep.setErrorCtx(ctxOCCI_);
    return (void *)NULL;
  }
  return (void *)objOCCI_;
}

void CAddress::readSQL(oracle::occi::AnyData& streamOCCI_)
{
   STATE = streamOCCI_.getString();
   ZIP = streamOCCI_.getString();
}

void CAddress::writeSQL(void *objectOCCI_, void *ctxOCCI_)
{
  CAddress *objOCCI_ = (CAddress *) objectOCCI_;
  oracle::occi::AnyData streamOCCI_(ctxOCCI_);

  try
  {
    if (objOCCI_->isNull())
      streamOCCI_.setNull();
    else
      objOCCI_->writeSQL(streamOCCI_);
  }
  catch (oracle::occi::SQLException& excep)
  {
    excep.setErrorCtx(ctxOCCI_);
  }
```

```
  return;
}

void CAddress::writeSQL(oracle::occi::AnyData& streamOCCI_)
{
   streamOCCI_.setString(STATE);
   streamOCCI_.setString(ZIP);
}
```

PERSON オブジェクト型の C++ ソース・ファイルの例 : cperson.cpp

```
#ifndef CPERSON_ORACLE
# include "cperson.h"
#endif


/*****************************************************************/
//  generated method implementations for the PERSON object type.
/*****************************************************************/

oracle::occi::Number CPerson::getId() const
{
  return ID;
}

void CPerson::setId(const oracle::occi::Number &value)
{
  ID = value;
}

MyFullName * CPerson::getName() const
{
  return NAME;
}

void CPerson::setName(MyFullName * value)
{
  NAME = value;
}

oracle::occi::Ref< MyAddress > CPerson::getCurr_addr() const
{
  return CURR_ADDR;
}
```

```
void CPerson::setCurr_addr(const oracle::occi::Ref< MyAddress > &value)
{
  CURR_ADDR = value;
}

OCCI_STD_NAMESPACE::vector< oracle::occi::Ref< MyAddress > >& CPerson::getPrev_addr_
l()
{
  return PREV_ADDR_L;
}

const OCCI_STD_NAMESPACE::vector< oracle::occi::Ref< MyAddress > >&
CPerson::getPrev_addr_l() const
{
  return PREV_ADDR_L;
}

void CPerson::setPrev_addr_l(const OCCI_STD_NAMESPACE::vector< oracle::occi::Ref<
MyAddress > > &value)
{
  PREV_ADDR_L = value;
}

void *CPerson::operator new(size_t size)
{
  return oracle::occi::PObject::operator new(size);
}

void *CPerson::operator new(size_t size, const oracle::occi::Connection * sess,
  const OCCI_STD_NAMESPACE::string& table)
{
  return oracle::occi::PObject::operator new(size, sess, table,
            (char *) "SCOTT.PERSON");
}

OCCI_STD_NAMESPACE::string CPerson::getSQLTypeName() const
{
  return OCCI_STD_NAMESPACE::string("SCOTT.PERSON");
}

CPerson::CPerson()
{
   NAME = (MyFullName *) 0;
}

void *CPerson::readSQL(void *ctxOCCI_)
{
```

```
  CPerson *objOCCI_ = new CPerson(ctxOCCI_);
  oracle::occi::AnyData streamOCCI_(ctxOCCI_);

  try
  {
    if (streamOCCI_.isNull())
      objOCCI_->setNull();
    else
      objOCCI_->readSQL(streamOCCI_);
  }
  catch (oracle::occi::SQLException& excep)
  {
    delete objOCCI_;
    excep.setErrorCtx(ctxOCCI_);
    return (void *)NULL;
  }
  return (void *)objOCCI_;
}

void CPerson::readSQL(oracle::occi::AnyData& streamOCCI_)
{
   ID = streamOCCI_.getNumber();
   NAME = (MyFullName *) streamOCCI_.getObject();
   CURR_ADDR = streamOCCI_.getRef();
   getVector(streamOCCI_, PREV_ADDR_L);
}

void CPerson::writeSQL(void *objectOCCI_, void *ctxOCCI_)
{
  CPerson *objOCCI_ = (CPerson *) objectOCCI_;
  oracle::occi::AnyData streamOCCI_(ctxOCCI_);

  try
  {
    if (objOCCI_->isNull())
      streamOCCI_.setNull();
    else
      objOCCI_->writeSQL(streamOCCI_);
  }
  catch (oracle::occi::SQLException& excep)
  {
    excep.setErrorCtx(ctxOCCI_);
  }
  return;
}
```

```
void CPerson::writeSQL(oracle::occi::AnyData& streamOCCI_)
{
   streamOCCI_.setNumber(ID);
   streamOCCI_.setObject(NAME);
   streamOCCI_.setRef(CURR_ADDR);
   setVector(streamOCCI_, PREV_ADDR_L);
}
```

STUDENT オブジェクト型の C++ ソース・ファイルの例 : cstudent.cpp

```
#ifndef CSTUDENT_ORACLE
# include "cstudent.h"
#endif


/*****************************************************************/
//  generated method implementations for the STUDENT object type.
/*****************************************************************/

OCCI_STD_NAMESPACE::string CStudent::getSchool_name() const
{
  return SCHOOL_NAME;
}

void CStudent::setSchool_name(const OCCI_STD_NAMESPACE::string &value)
{
  SCHOOL_NAME = value;
}

void *CStudent::operator new(size_t size)
{
  return oracle::occi::PObject::operator new(size);
}

void *CStudent::operator new(size_t size, const oracle::occi::Connection * sess,
  const OCCI_STD_NAMESPACE::string& table)
{
  return oracle::occi::PObject::operator new(size, sess, table,
            (char *) "SCOTT.STUDENT");
}

OCCI_STD_NAMESPACE::string CStudent::getSQLTypeName() const
{
  return OCCI_STD_NAMESPACE::string("SCOTT.STUDENT");
}
```

```
CStudent::CStudent()
{
}

void *CStudent::readSQL(void *ctxOCCI_)
{
  CStudent *objOCCI_ = new CStudent(ctxOCCI_);
  oracle::occi::AnyData streamOCCI_(ctxOCCI_);

  try
  {
    if (streamOCCI_.isNull())
      objOCCI_->setNull();
    else
      objOCCI_->readSQL(streamOCCI_);
  }
  catch (oracle::occi::SQLException& excep)
  {
    delete objOCCI_;
    excep.setErrorCtx(ctxOCCI_);
    return (void *)NULL;
  }
  return (void *)objOCCI_;
}

void CStudent::readSQL(oracle::occi::AnyData& streamOCCI_)
{
   CPerson::readSQL(streamOCCI_);
   SCHOOL_NAME = streamOCCI_.getString();
}

void CStudent::writeSQL(void *objectOCCI_, void *ctxOCCI_)
{
  CStudent *objOCCI_ = (CStudent *) objectOCCI_;
  oracle::occi::AnyData streamOCCI_(ctxOCCI_);

  try
  {
    if (objOCCI_->isNull())
      streamOCCI_.setNull();
    else
      objOCCI_->writeSQL(streamOCCI_);
  }
  catch (oracle::occi::SQLException& excep)
  {
    excep.setErrorCtx(ctxOCCI_);
  }
```

```
  return;
}

void CStudent::writeSQL(oracle::occi::AnyData& streamOCCI_)
{
   CPerson::writeSQL(streamOCCI_);
   streamOCCI_.setString(SCHOOL_NAME);
}
```

登録マッピング・ヘッダー・ファイルの例 : registerMappings.h

```
#ifndef REGISTERMAPPINGS_ORACLE
# define REGISTERMAPPINGS_ORACLE

#ifndef OCCI_ORACLE
# include <occi.h>
#endif

#ifndef CFULLNAME_ORACLE
# include "cfullname.h"
#endif

#ifndef CADDRESS_ORACLE
# include "caddress.h"
#endif

#ifndef CPERSON_ORACLE
# include "cperson.h"
#endif

#ifndef CSTUDENT_ORACLE
# include "cstudent.h"
#endif

void registerMappings(oracle::occi::Environment* envOCCI_);

#endif
```

登録マッピングの C++ ソース・ファイルの例 : registerMappings.cpp

```
#ifndef REGISTERMAPPINGS_ORACLE
# include "registerMappings.h"
#endif

void registerMappings(oracle::occi::Environment* envOCCI_)
{
  oracle::occi::Map *mapOCCI_ = envOCCI_->getMap();
  mapOCCI_->put("SCOTT.FULL_NAME", CFullName::readSQL, CFullName::writeSQL);
  mapOCCI_->put("SCOTT.ADDRESS", CAddress::readSQL, CAddress::writeSQL);
  mapOCCI_->put("SCOTT.PERSON", CPerson::readSQL, CPerson::writeSQL);
  mapOCCI_->put("SCOTT.STUDENT", CStudent::readSQL, CStudent::writeSQL);
}
```

注意 : このデモでは、生成された他の型の属性およびベース・クラスとして使用されている型を拡張します。cperson.h ファイルには、myfullname.h と myaddress.h がインクルード（#include）されています。cstudent.h ファイルには、myperson.h がインクルード（#include）されています。PERSON は拡張クラスであり、PFGRFDENT のベース・クラスであるため、PERSON と PFGRFDENT は個別のファイルに生成する必要があります。

ユーザー作成による拡張ファイルの例 : myfullname.h

```
#ifndef MYFULLNAME_ORACLE
# define MYFULLNAME_ORACLE

using namespace oracle::occi;
using namespace std;

#ifndef CFULLNAME_ORACLE
#include "cfullname.h"
#endif

/*****************************************************************/
// declarations for the MyFullName class.
/*****************************************************************/
class MyFullName : public CFullName {
 public:
   MyFullName(string first_name, string last_name);
   void displayInfo();
};

#endif
```

ユーザー作成による拡張ファイルの例 : myaddress.h

```
#ifndef MYADDRESS_ORACLE
# define MYADDRESS_ORACLE

using namespace oracle::occi;
using namespace std;

#ifndef CADDRESS_ORACLE
#include "caddress.h"
#endif

/*---------------------------------------------------------------------*/
// declarations for the MyAddress class.
/*---------------------------------------------------------------------*/
class MyAddress : public CAddress {
 public:
   MyAddress(string state_i, string zip_i);
   void displayInfo();
};

#endif
```

ユーザー作成による拡張ファイルの例 : myperson.h

```
#ifndef MYPERSON_ORACLE
# define MYPERSON_ORACLE

using namespace oracle::occi;
using namespace std;

#ifndef CPERSON_ORACLE
#include "cperson.h"
#endif

/*---------------------------------------------------------------------*/
// declarations for the MyPerson class.
/*---------------------------------------------------------------------*/
class MyPerson : public CPerson {
 public:
   MyPerson();
   MyPerson(void *ctxOCCI_) : CPerson(ctxOCCI_) { };
   MyPerson(Number id_i, MyFullName *name_i,const Ref<MyAddress>& addr_i);
   void move(const Ref<MyAddress>& new_addr);
   void displayInfo();
};

#endif
```

ユーザー作成による拡張ファイルの例 : mystudent.h

```
#ifndef MYSTUDENT_ORACLE
# define MYSTUDENT_ORACLE

using namespace oracle::occi;
using namespace std;

#ifndef CSTUDENT_ORACLE
#include "cstudent.h"
#endif

/*---------------------------------------------------------------------*/
// declarations for the MyStudent class.
/*---------------------------------------------------------------------*/
class MyStudent : public CStudent {
 public:
   MyStudent(Number id_i, MyFullName *name_i, Ref<MyAddress>& addr_i, string school_
name);
   void displayInfo();
} ;

#endif
```

ユーザー作成による拡張ファイルの例 : mydemo.cpp

```
#include <occi.h>
using namespace oracle::occi;
using namespace std;

#include "registerMappings.h"

#include "myfullname.h"
#include "myaddress.h"
#include "myperson.h"
#include "mystudent.h"

/*---------------------------------------------------------------------*/
// method implementations for MyFullName class.
/*---------------------------------------------------------------------*/

/* initialize MyFullName */
MyFullName::MyFullName(string first_name, string last_name)
{
  setFirstname(first_name);
  setLastname(last_name);
}
```

```
/* display all the information in MyFullName */
void MyFullName::displayInfo()
{
   cout << "FIRST NAME is:  " << getFirstname() << endl;
   cout << "LAST NAME is:  " << getLastname() << endl;
}

/*---------------------------------------------------------------------*/
// method implementations for MyAddress class.
/*---------------------------------------------------------------------*/

/* initialize MyAddress */
MyAddress::MyAddress(string state_i, string zip_i)
{
   setState(state_i);
   setZip(zip_i);
}

/* display all the information in MyAddress */
void MyAddress::displayInfo()
{
   cout << "STATE is:  " << getState() << endl;
   cout << "ZIP is:  " << getZip() << endl;
}

/*---------------------------------------------------------------------*/
// method implementations for MyPerson class.
/*---------------------------------------------------------------------*/
/* Default constructor needed because CStudent inherits from MyPerson */
MyPerson::MyPerson(){}

/* initialize MyPerson */
MyPerson::MyPerson(Number id_i,  MyFullName* name_i, const Ref<MyAddress>& addr_i)
{
   setId(id_i);
   setName(name_i);
   setCurr_addr(addr_i);
}

/* Move Person from curr_addr to new_addr */
void MyPerson::move(const Ref<MyAddress>& new_addr)
{
  // append curr_addr to the vector
  getPrev_addr_l().push_back(getCurr_addr());
  setCurr_addr(new_addr);
  this->markModified();
}
```

```
/*  Display all the information of MyPerson */
void MyPerson::displayInfo()
{
  cout << "---------------------------------  " << endl;

  cout << "ID is:  " << (int)getId() << endl;

  getName()->displayInfo();

  // de-referencing the Ref attribute using -> operator
  getCurr_addr()->displayInfo();

  cout << "Prev Addr List: " << endl;
  for (int i = 0; i < getPrev_addr_l().size(); i++)
  {
     // access the collection elements using [] operator
     getPrev_addr_l()[i]->displayInfo();
  }
}

/*****************************************************************/
// method implementations for MyStudent class.
/*****************************************************************/

/* initialize MyStudent */
MyStudent::MyStudent(Number id_i, MyFullName *name_i, Ref<MyAddress>& addr_i, string
school_name_i)
{
   setId(id_i);
   setName(name_i);
   setCurr_addr(addr_i);
   setSchool_name(school_name_i);

}

/* display the information in MyStudent */
void MyStudent::displayInfo()
{
   MyPerson::displayInfo();
   cout << "SCHOOL NAME is:  " << getSchool_name() << endl;
}
```

```
void process(Connection *conn)
{
  /* creating a persistent object of type Address in the connection,
     conn, and the database table, ADDR_TAB */
  MyAddress *addr1 = new(conn, "ADDR_TAB") MyAddress("CA", "94065");

  /* commit the transaction which results in the newly created object, addr1,
     being flushed to the server */
  conn->commit();

  MyFullName name1("Joe", "Black");

  /* creating a persistent object of type Person in the connection,
     conn, and the database table, PERSON_TAB */
  MyPerson *person1 = new(conn, "PERSON_TAB") MyPerson(1,&name1,
                              addr1->getRef());

  /* commit the transaction which results in the newly created object,
     person1 being flushed to the server */
  conn->commit();

  Statement *stmt = conn->createStatement(
    "SELECT REF(per) from person_tab per ");

  ResultSet *resultSet = stmt->executeQuery();

  if (!resultSet->next())
  {
     cout << "No record found \n";
  }

  RefAny joe_refany = resultSet->getRef(1);
  Ref <MyPerson> joe_ref(joe_refany);

  /* de-referencing Ref using ptr() operator.  operator -> and operator *
   also could be used to de-reference the Ref. As part of de-referencing,
   if the referenced object is not found in the application cache, the object
   data is retrieved from the server and unmarshalled into Person instance
   through MyPerson::readSQL() method. */

  MyPerson *joe = joe_ref.ptr();
  joe->displayInfo();
```

```
/* creating a persistent object of type MyAddress, in the connection,
   conn and the database table, ADDR_TAB */
MyAddress *new_addr1 = new(conn, "ADDR_TAB") MyAddress("PA", "92140");
conn->commit();

joe->move(new_addr1->getRef());
joe->displayInfo();

/* commit the transaction which results in the newly created object,
   new_addr1 and the dirty object, joe to be flushed to the server.  */
conn->commit();

MyAddress *addr2 = new(conn, "ADDR_TAB") MyAddress("CA", "95065");
MyFullName name2("Jill", "White");
Ref<MyAddress> addrRef = addr2->getRef();
MyStudent *student2 = new(conn, "STUDENT_TAB") MyStudent(2, &name2,
                          addrRef, "Stanford");
conn->commit();

Statement *stmt2 = conn->createStatement(
 "SELECT REF(Student) from student_tab Student where id = 2");

ResultSet *resultSet2 = stmt2->executeQuery();
if (!resultSet2->next())
{
   cout << "No record found \n";
}

RefAny jillrefany = resultSet2->getRef(1);
Ref <MyStudent> jill_ref(jillrefany);

MyStudent *jill = jill_ref.ptr();

cout << jill->getPrev_addr_l().size();

jill->displayInfo();

MyAddress *new_addr2 = new(conn, "ADDR_TAB") MyAddress("CO", "80021");
conn->commit();

jill->move(new_addr2->getRef());

jill->displayInfo();

jill->markModified();
```

```
  conn->commit();

  /* The following delete statements delete the objects only from the
    application cache.  To delete the objects from the server, mark_deleted()
    should be used. */
  delete person1;
  delete addr1;
  delete new_addr1;
  delete addr2;
  delete student2;
  delete new_addr2;

  conn->terminateStatement(stmt);
  conn->terminateStatement(stmt2);

}

/*---------------------------------------------------------------------------*/
// main function of this OCCI application.
// This application connects to the database as scott/tiger, creates
// the Person (Joe Black) whose Address is in CA, and commits the changes.
// The Person object is then retrieved from the database and its
// information is displayed. A second Address object is created (in PA),
// then the previously retrieved Person object (Joe Black) is moved to
// this new address. The Person object is then displayed again.
// The similar commands are executed for "Jill White", a Student at Stanford,
// who is moved from CA to CO.
/*---------------------------------------------------------------------------*/

int main()
{

  Environment *env = Environment::createEnvironment(Environment::OBJECT );

  /* Call the OTT generated function to register the mappings */
  registerMappings(env);

  Connection *conn = env->createConnection("scott","tiger","");

  process(conn);

  env->terminateConnection(conn);
  Environment::terminateEnvironment(env);

  return 0;
}
```

注意： 各拡張クラスの宣言には、拡張するクラスに対して生成されたヘッダー・ファイルのインクルード（#include）が必要です。たとえば、myfullname.h には cperson.h のインクルード（#include）が、mystudent.h には cstudent.h のインクルード（#include）が必要です。マッピングを登録する registerMapping 関数をコールするため、mydemo.cpp には registerMappings.h のインクルード（#include）が必要です。

OTT アプリケーションの例によって生成された出力

OCCI アプリケーションの例によって生成された出力は、次のとおりです。

```
-----------------------------------
ID is:  1
FIRST NAME is:  Joe
LAST NAME is:  Black
STATE is:  CA
ZIP is:  94065
Prev Addr List:
-----------------------------------
ID is:  1
FIRST NAME is:  Joe
LAST NAME is:  Black
STATE is:  PA
ZIP is:  92140
Prev Addr List:
STATE is:  CA
ZIP is:  94065
-----------------------------------
ID is:  2
FIRST NAME is:  Jill
LAST NAME is:  White
STATE is:  CA
ZIP is:  95065
Prev Addr List:
SCHOOL NAME is:  Stanford
-----------------------------------
ID is:  2
FIRST NAME is:  Jill
LAST NAME is:  White
STATE is:  CO
ZIP is:  80021
Prev Addr List:
STATE is:  CA
ZIP is:  95065
SCHOOL NAME is:  Stanford
```

OTT ユーティリティの参照

OTT ユーティリティの動作は、OTT コマンドラインまたは CONFIG ファイルに指定されているパラメータによって制御されます。一部のパラメータは、INTYPE ファイルにも指定できます。

この項では、次の項目について詳しく説明します。

OTT コマンドラインの構文

OTT コマンドライン・インタフェースは、OTT ユーティリティを明示的に起動して、データベース型を C 構造体または C++ クラスに変換するときに使用します。オブジェクトを使用する OCI、OCCI または Pro*C/C++ の各アプリケーションを開発する場合は、必ずこのインタフェースが必要です。

OTT コマンドライン文は、コマンド OTT と、その後に続く OTT ユーティリティ・パラメータのリストで構成されています。

OTT コマンドライン文に指定できるパラメータは、次のとおりです（アルファベット順）。

```
[ATTRACCESS={PRIVATE|PROTECTED}]

[CASE={SAME|LOWER|UPPER|OPPOSITE}]

CODE={C|ANSI_C|KR_C|CPP}

[CONFIG=filename]

[CPPFILE=filename]

[ERRTYPE=filename]

[HFILE=filename]

[INITFILE=filename]

[INITFUNC=filename]
```

```
[INTYPE=filename]
[MAPFILE=filename]
[MAPFUNC=filename]
OUTTYPE=filename
[SCHEMA_NAMES={ALWAYS|IF_NEEDED|FROM_INTYPE}]
[TRANSITIVE={TRUE|FALSE}]
[USE_MARKER={TRUE|FALSE}]
[USERID=username/password[@db_name]]
```

注意： 通常、OTT コマンドに続くパラメータの順序は重要ではありません。OUTTYPE パラメータと CODE パラメータは常に必要です。

HFILE パラメータは、ほとんどの場合に使用されます。省略した場合は、INTYPE ファイル内の各型に対して、HFILE を個別に指定する必要があります。OTT ユーティリティは、INTYPE ファイルにリストされていない型の変換が必要であると判断した場合は、エラーをレポートします。したがって、INTYPE ファイルが以前に OTT OUTTYPE ファイルとして生成されている場合のみ、HFILE パラメータを省略できます。

INTYPE ファイルを省略すると、スキーマ全体が変換されます。詳細は、次の項のパラメータの説明を参照してください。

次の記述は、OTT コマンドライン文の例です（1 行で入力します）。

```
ott userid=scott/tiger intype=in.typ outtype=out.typ code=c hfile=demo.h
errtype=demo.tls case=lower
```

OTT コマンドラインの各パラメータについては、次の項を参照してください。

OTT ユーティリティのパラメータ

次のフォーマットを使用して、OTT コマンドラインにパラメータを入力します。

```
parameter=value
```

この例の parameter はリテラル・パラメータ文字列であり、value は有効なパラメータ設定値です。リテラル・パラメータ文字列は大 / 小文字を区別しません。

コマンドラインのパラメータは、空白またはタブのいずれかを使用して区切ります。

パラメータは構成ファイル内にも指定できます。ただし、この場合は、行内の空白は許可されないため、各パラメータは独立した行に指定する必要があります。また、CASE、CPPFILE、HFILE、INITFILE、INTFUNC、MAPFILE および MAPFUNC の各パラメータは、INTYPE ファイルに指定できます。

OTT ユーティリティのパラメータは、次の項で説明します。

ATTRACCESS パラメータ

C++ に対してのみ使用します。このパラメータは、OTT ユーティリティが型属性に対して PRIVATE アクセスまたは PROTECTED アクセスを生成するかどうかを指定します。PRIVATE を指定すると、OTT ユーティリティは、各型属性に対してアクセッサ・メソッド（get*xxx*）とミューテータ・メソッド（set*xxx*）を生成します。

```
ATTRACCESS=PRIVATE|PROTECTED
```

デフォルトは PROTECTED です。

CASE パラメータ

このパラメータは、OTT ユーティリティが生成する一部の C または C++ 識別子の大 / 小文字の区別に影響を与えます。CASE の有効な値は、SAME、LOWER、UPPER および OPPOSITE です。

CASE=SAME の場合、データベース型と属性名を C または C++ 識別子に変換するときに、文字の大 / 小文字は変更されません。

CASE=LOWER の場合、大文字はすべて小文字に変換されます。

CASE=UPPER の場合、小文字はすべて大文字に変換されます。

CASE=OPPOSITE の場合、大文字はすべて小文字に変換され、小文字はすべて大文字に変換されます。

```
CASE=[SAME|LOWER|UPPER|OPPOSITE]
```

このパラメータは、INTYPE ファイルに記述されていない識別子（明示的にリストされていない属性または型）のみに作用します。大 / 小文字の変換は、正当な識別子が生成された後で行われます。

INTYPE ファイルで特定した型の C 構造体識別子の大 / 小文字は、INTYPE ファイルの大 / 小文字と同じです。たとえば、次の記述が INTYPE ファイルに指定されていると想定します。

```
TYPE Worker
```

OTT ユーティリティによって、次の C 構造体が生成されます。

```
struct Worker {...};
```

次に、次の記述が INTYPE ファイルに指定されていると想定します。

```
TYPE wOrKeR
```

OTT ユーティリティによって、次の C 構造体が生成されます。この構造体は、INTYPE ファイルに指定されている大 / 小文字に従っています。

```
struct wOrKeR {...};
```

INTYPE ファイルに記述されていない、大 / 小文字の区別のない SQL 識別子は、CASE=SAME の場合は大文字で、CASE=OPPOSITE の場合は小文字で表現されます。SQL 識別子は、宣言時に引用符が付けられていない場合、大 / 小文字が区別されません。

CODE パラメータ

このパラメータは、OTT ユーティリティで出力するホスト言語を指定します。CODE=C は CODE=ANSI_C と等価です。OCCI アプリケーション用に C++ コードを生成するには、CODE=CPP を OTT ユーティリティに指定する必要があります。

```
CODE=C|KR_C|ANSI_C|CPP
```

このパラメータは必須で、デフォルト値はありません。ホスト言語のいずれか 1 つを指定する必要があります。

CONFIG パラメータ

このパラメータは、使用する OTT 構成ファイルの名前を指定します。構成ファイルには、共通に使用されるパラメータ指定がリストされます。パラメータ指定は、オペレーティング・システム固有の位置にあるシステム構成ファイルからも読み込まれます。残りのすべてのパラメータ指定は、コマンドラインまたは INTYPE ファイルに指定する必要があります。

```
CONFIG=filename
```

注意： OTT コマンドラインに指定できるのは、CONFIG パラメータのみです。このパラメータを CONFIG ファイルに指定することはできません。

CPPFILE パラメータ

C++ に対してのみ使用します。このパラメータは、OTT ユーティリティで生成するメソッド実装が格納される C++ ソース・ファイルの名前を指定します。

次の場合、このパラメータは必須です。

- INTYPE ファイルに指定されていない型の生成が必要で、2 つ以上の CPPFILE が生成されている場合。この場合、未指定の型は、コマンドラインに指定した CPPFILE に格納されます。
- INTYPE パラメータが未指定で、スキーマ内のすべての型を OTT ユーティリティで変換する場合。

このパラメータに対するこれらの制限は、Pro*C/C++ および OCI に対してすでに指定されている既存の HFILE パラメータ制限に類似しています。CPPFILE パラメータが INTYPE ファイルの個々の型に対して指定されている場合、このパラメータはオプションです。

```
CPPFILE=filename
```

ERRTYPE パラメータ

このパラメータを指定すると、INTYPE ファイルのリストが、すべての情報メッセージおよびエラー・メッセージとともに ERRTYPE ファイルに書き込まれます。情報メッセージおよびエラー・メッセージは、ERRTYPE を指定したかどうかに関係なく、標準出力されます。

基本的に、ERRTYPE ファイルは、エラー・メッセージが追加された INTYPE ファイルのコピーです。ほとんどの場合、エラー・メッセージにはエラーの原因となったテキストへのポインタが含まれます。

コマンドラインで、拡張子を付けずにファイル名を ERRTYPE パラメータに指定すると、.TLS や .HS などのプラットフォーム固有の拡張子が自動的に追加されます。

```
ERRTYPE=filename
```

HFILE パラメータ

このパラメータは、OTT ユーティリティで生成するヘッダー・ファイル（.h）の名前を指定します。コマンドラインに指定された HFILE には、型の宣言が含まれています。この宣言は、INTYPE ファイルに指定されていますが、そのヘッダー・ファイルはこのファイルには指定されていません。

このパラメータは、各型のヘッダー・ファイルが INTYPE ファイルに個別に指定されていない場合に必要です。このパラメータは、INTYPE ファイルに記述されていない型を、他の型が必要とする理由から生成する場合にも必要です。この他の型は、2 つ以上の異なるファイルで宣言されます。

コマンドラインまたは INTYPE ファイルで、拡張子を付けずにファイル名を HFILE パラメータに指定すると、.H や .h などのプラットフォーム固有の拡張子が自動的に追加されます。

```
HFILE=filename
```

INITFILE パラメータ

OCI に対してのみ使用します。このパラメータは、OTT ユーティリティで生成する初期化ファイルの名前を指定します。このパラメータを省略すると、初期化ファイルは生成されません。

Pro*C/C++ プログラムの場合、必要な初期化は SQLLIB ランタイム・ライブラリによって実行されるため、INITFILE は不要です。OCI プログラマは、INITFILE ファイルをコンパイルおよびリンクし、環境ハンドルの作成時に、初期化関数をコールする必要があります。

コマンドラインまたは INTYPE ファイルに、拡張子を付けずにファイル名を INITFILE パラメータに指定すると、.C や .c などのプラットフォーム固有の拡張子が自動的に追加されます。

```
INITFILE=filename
```

INITFUNC パラメータ

OCI に対してのみ使用します。このパラメータは、OTT ユーティリティで生成する初期化関数の名前を指定します。

このパラメータはオプションです。このパラメータを省略すると、INITFILE の名前に基づいて初期化関数の名前が作成されます。

```
INITFUNC=filename
```

INTYPE パラメータ

このパラメータは、オブジェクト型指定のリストを読み込む元のファイル名を指定します。OTT ユーティリティによって、リストにある各型が変換されます。

```
INTYPE=filename
```

USERID が第 1 パラメータ、INTYPE が第 2 パラメータの場合で、USERID= が省略されている場合は、INTYPE= を省略できます。たとえば、次のとおりです。

```
OTT username/password filename...
```

INTYPE パラメータが指定されていない場合は、ユーザーのスキーマにある型すべてが変換されます。

INTYPE ファイルは、型宣言に対する Make ファイルとみなすことができます。このファイルには、C 構造体宣言または C++ クラスを必要とする型がリストされます。INTYPE ファイルのフォーマットは、7-118 ページの「INTYPE ファイルの構造」を参照してください。

コマンドラインで、拡張子を付けずにファイル名を INTYPE パラメータに指定すると、.TYP や .typ などのプラットフォーム固有の拡張子が自動的に追加されます。

> **関連項目：** INTYPE ファイルのフォーマットに関する詳細は、7-118 ページの「INTYPE ファイルの構造」を参照してください。

MAPFILE パラメータ

C++ に対してのみ使用します。このパラメータは、OTT ユーティリティで生成するマッピング・ファイル（.cpp）とそれに対応するヘッダー・ファイル（.h）の名前を指定します。.cpp ファイルにはマッピングを登録する関数の実装が格納され、.h ファイルにはその関数のプロトタイプが格納されます。

このパラメータは、C++ の生成には必須です。CODE=CPP を指定するときは、MAPFILE パラメータの値も指定する必要があります。指定しない場合、OTT ユーティリティではエラーになります。

このパラメータは、コマンドラインまたは INTYPE ファイルに指定できます。

MAPFILE=*filename*

MAPFUNC パラメータ

C++ に対してのみ使用します。このパラメータは、OTT ユーティリティで生成する、マッピングの登録に使用する関数の名前を指定します。

このパラメータは、C++ を生成する場合はオプションです。このパラメータを省略すると、マッピングを登録する関数の名前は、MAPFILE パラメータに指定されているファイル名に基づいて作成されます。

このパラメータは、コマンドラインまたは INTYPE ファイルに指定できます。

MAPFUNC=*functionname*

OUTTYPE パラメータ

このパラメータは、OTT ユーティリティが処理するオブジェクト・データ型すべての型情報を出力するファイルの名前を指定します。このファイルには、INTYPE ファイルに明示的に指定されたすべての型が格納されます。また、変換が必要な他の型の宣言で使用されているために変換される、追加の型が格納される場合があります。このファイルは、OTT ユーティリティの今後の起動で、INTYPE ファイルとして使用できます。

OUTTYPE=*filename*

INTYPE パラメータと OUTTYPE パラメータが同一のファイルを参照している場合、INTYPE ファイルの古い情報は、新しい INTYPE の情報によって置換されます。これは、型の変更から始まり、型宣言の生成、ソースコードの編集、プリコンパイル、コンパイル、デバッグに至るサイクル内で、同一の INTYPE ファイルを繰返し使用するときに便利です。

このパラメータは必須です。

コマンドラインまたは INTYPE ファイルに、拡張子を付けずにファイル名を OUTTYPE パラメータに指定すると、.TYP や .typ などのプラットフォーム固有の拡張子が自動的に追加されます。

SCHEMA_NAMES パラメータ

デフォルト・スキーマに基づいて命名した型のデータベース名を、OUTTYPE ファイル内のスキーマ名で修飾する場合、このパラメータで制御します。OTT ユーティリティによって生成された OUTTYPE ファイルには、型名を含めて、OTT ユーティリティで処理された型の情報が記述されます。

```
SCHEMA_NAMES=ALWAYS|IF_NEEDED|FROM_INTYPE
```

デフォルト値は ALWAYS です。

> **関連項目：** 詳細は、7-122 ページの「SCHEMA_NAMES の使用方法」を参照してください。

TRANSITIVE パラメータ

このパラメータは、INTYPE ファイルに明示的にリストされていない型の依存性を変換するかどうかを指定します。有効な値は、TRUE と FALSE です。

```
TRANSITIVE=TRUE|FALSE
```

デフォルト値は TRUE です。

TRANSITIVE=TRUE を指定すると、他の型に必要な型で、INTYPE ファイルに指定されていない型が生成されます。

TRANSITIVE=FALSE を指定すると、INTYPE ファイルに指定されていない型は生成されません。これは、ファイルに指定されていない型が、生成された他の型に対する属性の型として使用されている場合にも適用されます。

USE_MARKER パラメータ

このパラメータは、ユーザーが追加したコードを引き継ぐために、OTT で OTT マーカー（使用する場合）をサポートする必要があるかどうかを指定します。有効な値は、TRUE と FALSE です。

```
USE_MARKER=TRUE|FALSE
```

デフォルト値は FALSE です。

USE_MARKER=TRUE を指定すると、OTT_USER_CODESTART と OTT_USERCODE_END の 2 つのマーカーの間に追加されたコードは、同じファイルが再生成されたときに引き継がれます。

USE_MARKER=FALSE を指定すると、OTT_USERCODE_START と OTT_USERCODE_END の 2 つのマーカーの間にユーザーがコードを追加しても、追加したコードは引き継がれません。

USERID パラメータ

このパラメータは、Oracle ユーザー名、パスワードおよびオプションのデータベース名（Oracle Net のデータベース指定文字列）を指定します。データベース名を省略すると、デフォルトのデータベースが使用されます。

```
USERID=username/password[@db_name]
```

これが第 1 パラメータである場合は、USERID= を省略して、次のように指定できます。

```
OTT username/password ...
```

このパラメータはオプションです。このパラメータを省略すると、OTT ユーティリティは、ユーザー OPS$*username* でデフォルトのデータベースに自動的に接続します。*username* は、オペレーティング・システムのユーザー名です。

OTT パラメータの指定可能な場所

OTT パラメータは、コマンドラインまたはコマンドラインで名前が指定された CONFIG ファイル、あるいはその両方に指定できます。パラメータの一部は、INTYPE ファイルにも指定できます。

OTT ユーティリティは、次のように起動します。

```
OTT parameters
```

CONFIG パラメータを使用し、構成ファイルの名前をコマンドラインで次のように指定できます。

```
CONFIG=filename
```

コマンドラインでこのパラメータの名前を指定すると、他のパラメータは、ファイル名 *filename* の構成ファイルから読み込まれます。

パラメータは、オペレーティング・システム固有の位置にあるデフォルトの構成ファイルからも読み込まれます。このファイルの存在は必要ですが、空でも構いません。構成ファイルにデータを入力する場合、空白は使用できないこと、パラメータは 1 行ごとに入力する必要があることに注意してください。

引数を指定しないで OTT ユーティリティを実行すると、オンライン・パラメータ・リファレンスが表示されます。

OTT ユーティリティによって変換される型は、INTYPE パラメータで指定されたファイルに従って名前が付けられます。CASE、CPPFILE、HFILE、INITFILE、INITFUNC、MAPFILE および MAPFUNC の各パラメータは、INTYPE ファイルにも指定できます。OTT ユーティリティによって生成された OUTTYPE ファイルには CASE パラメータが含まれ、初期化ファイルが生成された場合は INITFILE パラメータと INITFUNC パラメータが、C++ コードが生成された場合は MAPFILE パラメータと MAPFUNC パラメータが含まれます。

OUTTYPE ファイルは、C++ 用の CPPFILE と同様に、各型に対する HFILE を個別に指定します。

OTT コマンドの大 / 小文字区別は、オペレーティング・システムによって異なります。

INTYPE ファイルの構造

INTYPE ファイルと OUTTYPE ファイルには、OTT ユーティリティによって変換された型がリストされ、型名または属性名を有効な C または C 識別子に変換する方法の決定に必要なすべての情報が含まれています。これらのファイルには、1 つ以上の型指定が記述されています。また、これらのファイルには、次のオプションに関する指定が含まれている場合があります。

- CASE
- CPPFILE
- HFILE
- INITFILE
- INITFUNC
- MAPFILE
- MAPFUNC

CASE、INITFILE、INITFUNC、MAPFILE または MAPFUNC オプションを指定する場合は、すべての型指定に先行して指定する必要があります。これらのオプションがコマンドラインと INTYPE ファイルの両方に指定されている場合は、コマンドラインの値が使用されます。

> **関連項目：** 簡単なユーザー定義の INTYPE ファイルの例およびこのファイルから OTT ユーティリティによって生成される完全な OUTTYPE ファイルの例は、7-26 ページの「OUTTYPE ファイルの概要」を参照してください。

INTYPE ファイルの型指定

INTYPE ファイル内の型指定によって、変換するオブジェクト・データ型の名前が指定されます。次の記述は、ユーザー作成 INTYPE ファイルの例です。

```
TYPE employee
  TRANSLATE SALARY$ AS salary
            DEPTNO AS department
TYPE ADDRESS
TYPE PURCHASE_ORDER AS p_o
```

型指定の構造は、次のとおりです。

```
TYPE type_name
[GENERATE type_identifier]
[AS type_identifier]
[VERSION [=] version_string]
[HFILE [=] hfile_name]
[CPPFILE [=] cppfile_name]
[TRANSLATE{member_name [AS identifier]}...]
```

type_name の構文は、次のフォーマットに従います。

```
[schema_name.]type_name
```

この構文の *schema_name* は、指定されたオブジェクト・データ型を所有するスキーマの名前です。*type_name* は、その型の名前です。未指定の場合、デフォルト・スキーマは OTT ユーティリティを起動しているユーザー ID のスキーマになります。特定のスキーマを使用するには、schema_name を使用する必要があります。

型指定の構成要素は、次のとおりです。

- *type_name*: オブジェクト・データ型の名前です。
- *type_identifier*: クラスの表現に使用する C または C++ の識別子です。GENERATE 句は、OTT ユーティリティが生成するクラスの名前を指定するために使用されます。記述するクラスの名前は、AS 句によって指定されます。GENERATE 句は、通常、クラスを拡張するために使用されます。ユーザー定義型を示す C 構造体または C++ クラスの名前は、必要に応じて、GENERATE 句を伴わずに、AS 句によって指定されます。
- *version_string*: 型のバージョン文字列です。OTT ユーティリティが前の起動でコードを生成したときに使用した内容です。バージョン文字列は、OTT ユーティリティによって生成され、OUTTYPE ファイルに出力されます。このファイルは、後で OTT ユーティリティを起動するときに INTYPE ファイルとして使用できます。バージョンの文字列は、OTT ユーティリティ操作には影響を与えませんが、実行中のプログラムで使用するオブジェクト・データ型のバージョンを選択するために使用できます。
- *hfile_name*: 対応するクラスの宣言が書き込まれるヘッダー・ファイルの名前です。HFILE 句を省略すると、コマンドラインの HFILE パラメータで指定したファイルが使用されます。
- *cppfile_name*: 対応するクラスのメソッド実装が書き込まれる C++ ソース・ファイルの名前です。CPPFILE 句を省略すると、コマンドラインの CPPFILE パラメータで指定したファイルが使用されます。
- *member_name*: 識別子に変換される属性（データ・メンバー）の名前です。
- *identifier*: プログラムで属性を表現するために使用する C または C++ の識別子です。この方法によって、任意の数の属性に対して識別子を指定できます。属性が指定されていない場合には、デフォルトの名前マッピング・アルゴリズムが使用されます。

オブジェクト・データ型は、次のいずれかの場合に変換する必要があります。

- そのオブジェクト・データ型が INTYPE ファイルに指定されている場合。
- そのオブジェクト・データ型には変換が必要な別の型の宣言が必要で、TRANSITIVE パラメータが TRUE に設定されている場合。

明示的に記述されていない型が、あるファイル内で正確に宣言された型に必要とされる場合、必要とされる型の変換結果は、その結果を必要とする明示的に宣言された型と同じファイルに書き込まれます。

明示的に記述されていない型が、複数の異なるファイル内で宣言された型に必要とされる場合、必要とされる型の変換結果は、グローバルな HFILE ファイルに書き込まれます。

注意： リリース 1（9.0.1）では、INTYPE ファイルに指定されていない必要なオブジェクト型を、OTT ユーティリティで生成するかどうかを指定できます。必要なオブジェクト型を OTT ユーティリティで生成しない場合は、TRANSITIVE=FALSE を設定してください。デフォルトは、TRANSITIVE=TRUE です。

ネストされたインクルード・ファイル（#include）の生成

OTT ユーティリティで生成した HFILE ファイルには、他の必要なファイルをインクルード（#include）し、ファイル名から作成した記号を定義（#define）します。この定義（#define）された記号は、関連する HFILE ファイルがすでにインクルード（#include）されているかどうかを判断するために使用できます。たとえば、データベースに次の型があると仮定します。

```
create type px1 AS OBJECT (col1 number, col2 integer);
create type px2 AS OBJECT (col1 px1);
create type px3 AS OBJECT (col1 px1);
```

INTYPE ファイルには、次の情報が記述されます。

```
CASE=lower
type pxl
  hfile tott95a.h
type px3
  hfile tott95b.h
```

次のように、OTT ユーティリティを起動します。

```
ott scott/tiger tott95i.typ outtype=tott95o.typ code=c
```

OTT ユーティリティは、tott95a.h と tott95b.h というファイル名を持つ 2 つのヘッダー・ファイルを作成します。

tott95a.h ファイルの内容は、次のとおりです。

```
#ifndef TOTT95A_ORACLE
#define TOTT95A_ORACLE
#ifndef OCI_ORACLE
#include <oci.h>
#endif
typedef OCIRef px1_ref;
struct px1
{
   OCINumber col1;
   OCINumber col2;
}
typedef struct px1 px1;
struct px1_ind
{
   OCIInd _atomic;
   OCIInd col1;
   OCIInd col2;
}
typedef struct px1_ind px1_ind;
#endif
```

tott95b.h ファイルの内容は、次のとおりです。

```
#ifndef TOTT95B_ORACLE
#define TOTT95B_ORACLE
#ifndef OCI_ORACLE
#include <oci.h>
#endif
#ifndef TOTT95A_ORACLE
#include "tott95a.h"
#endif
typedef OCIRef px3_ref;
struct px3
{
   struct px1 col1;
};
typedef struct px3 px3;
struct px3_ind
{
   OCIInd _atomic;
   struct px1_ind col1
};
typedef struct px3_ind px3_ind;
#endif
```

tott95b.h ファイルでは、記号 TOTT95B_ORACLE がファイルの先頭に定義（#define）されています。これによって、このヘッダー・ファイルを別のファイルに条件付きでインクルード（#include）することができます。インクルードするには、次の構造体を使用します。

```
#ifndef TOTT95B_ORACLE
#include "tott95b.h"
#endif
```

この方法を使用すると、たとえば、foo.h に tott95b.h をインクルード（#include）できます。この場合、foo.h に他のファイルのインクルード（#include）があるかどうか、また、tott95b.h のインクルード（#include）があるかどうかを事前に知る必要はありません。

記号 TOTT95B_ORACLE を定義すると、ファイル oci.h がインクルード（#include）されます。OTT ユーティリティで生成した各 HFILE には oci.h がインクルードされます。このファイルには、Pro*C/C++ または OCI のプログラマにとって便利な、型と関数の宣言が格納されています。OTT ユーティリティが #include で < > を使用するのは、この場合のみです。

次に、ファイル tott95a.h がインクルードされます。これは、このファイルには tott95b.h に必要な struct px1 の宣言が指定されているためです。INTYPE ファイルの要求で、複数のファイルに型宣言を書き込む場合、OTT ユーティリティは、各 HFILE にインクルード（#include）する必要のある他のファイルを判断し、必要な #include ファイルをそれぞれ生成します。

OTT ユーティリティでは、この #include に " " が使用されます。tott95b.h がインクルードされているプログラムをコンパイルするとき、tott95a.h の検索は、ソース・プログラムが検出された場所から開始され、以降は実装定義の検索規則に従います。この方法で tott95a.h を検索できない場合は、完全なファイル名（スラッシュ文字（/）で始まる UNIX の絶対パス名など）を INTYPE ファイルで使用して、tott95a.h の位置を指定する必要があります。

SCHEMA_NAMES の使用方法

このパラメータは、OTT ユーティリティが接続しているデフォルト・スキーマに基づいた型の名前を、OUTTYPE ファイル内のスキーマ名で修飾するかどうかを決定します。

デフォルト・スキーマ以外のスキーマに基づく型の名前は、OUTTYPE ファイル内のスキーマ名で常に修飾されます。

スキーマ名の有無によって、プログラム実行中に型を検索するスキーマが決定されます。

SCHEMA_NAMES パラメータには、次の 3 つの有効値があります。

- SCHEMA_NAMES=ALWAYS （デフォルト）

  OUTTYPE ファイル内のすべての型名はスキーマ名で修飾されます。

- SCHEMA_NAMES=IF_NEEDED

  デフォルト・スキーマに所属する OUTTYPE ファイル内の型名は、スキーマ名で修飾されません。デフォルト・スキーマ以外のスキーマに所属する型名は、通常どおり、スキーマ名で修飾されます。

- SCHEMA_NAMES=FROM_INTYPE

  INTYPE ファイルに記述されている型は、INTYPE ファイル内のスキーマ名で修飾されている場合に限って、OUTTYPE ファイル内のスキーマ名で修飾されます。INTYPE ファイルには記述されていないが、型の依存性のために生成されたデフォルト・スキーマ内の型は、その型に依存している OTT ユーティリティで最初に検出された型がスキーマ名付きで記述されている場合にのみ、スキーマ名付きで記述されます。ただし、OTT ユーティリティが接続しているデフォルト・スキーマにない型は、常に、明示的に指定したスキーマ名付きで記述されます。

OTT ユーティリティで生成した OUTTYPE ファイルは、Pro*C/C++ の INTYPE ファイルです。このファイルは、データベース型名を C 構造体名と対応付けます。この情報は実行時に使用され、構造体に正しいデータベース型が選択されていることを確認します。型が OUTTYPE ファイル（Pro*C/C++ の INTYPE ファイル）内のスキーマ名で指定されている場合、その型は、プログラム実行中は名前付きスキーマ内にあります。型がスキーマ名なしで指定されている場合、その型は、プログラムが接続しているデフォルト・スキーマ内にあります。このデフォルト・スキーマは、OTT ユーティリティが使用するデフォルト・スキーマとは異なる場合があります。

SCHEMA_NAMES パラメータの使用例

SCHEMA_NAMES パラメータが FROM_INTYPE に設定され、INTYPE ファイルに次の指定がある例を想定します。

```
TYPE Person
TYPE joe.Dept
TYPE sam.Company
```

OTT で生成した構造体を使用している Pro*C/C++ アプリケーションで、sam.Company、joe.Dept および Person の各型が使用されています。スキーマ名のない Person は、アプリケーションが接続しているスキーマの Person 型を表しています。

OTT ユーティリティとアプリケーションの両方がスキーマ joe に接続する場合、アプリケーションは OTT ユーティリティが使用する型と同じ型（joe.Person）を使用します。OTT ユーティリティはスキーマ joe に接続しているが、アプリケーションはスキーマ mary に接続している場合、アプリケーションは mary.Person 型を使用します。この動作は、スキーマ joe とスキーマ mary で同一の CREATE TYPE Person 文が実行されている場合にのみ適切です。

一方、アプリケーションは、接続しているスキーマに関係なく、joe.Dept 型を使用します。この動作のためには、スキーマ名を INTYPE ファイルで使用している型名に挿入する必要があります。

ユーザーが明示的に名前を指定していない型が、OTT ユーティリティによって変換される場合があります。たとえば、次の SQL 宣言を想定します。

```
CREATE TYPE Address AS OBJECT
(
street    VARCHAR2(40),
city      VARCHAR(30),
state     CHAR(2),
zip_code  CHAR(10)
);

CREATE TYPE Person AS OBJECT
(
name      CHAR(20),
age       NUMBER,
addr      ADDRESS
);
```

OTT ユーティリティはスキーマ joe に接続し、SCHEMA_NAMES=FROM_INTYPE が指定され、ユーザーの INTYPE ファイルには、次のいずれか 1 つが含まれていると仮定します。

```
TYPE Person
```

または

```
TYPE joe.Person
```

INTYPE ファイルには、joe.Person 型でネストしたオブジェクト型として使用されている joe.Address 型は指定されていません。

INTYPE ファイルに Type Person が指定されている場合は、TYPE Person と TYPE Address が OUTTYPE ファイルに指定されます。

INTYPE ファイルに TYPE joe.Person が指定されている場合は、TYPE joe.Person と TYPE joe.Address が OUTTYPE ファイルに指定されます。

OTT ユーティリティが変換する複数の型には、joe.Address 型が埋め込まれているが、そのことが INTYPE ファイルには明示的に記述されていない場合、スキーマ名を使用するかどうかの判断は、埋め込まれた joe.Address 型を OTT ユーティリティが最初に検出したときに行われます。なんらかの理由で、ユーザーがスキーマ名を joe.Address 型には付け、Person 型には付けないようにするには、次のように、この内容を INTYPE ファイルで明示的に指定する必要があります。

```
TYPE      joe.Address
```

各型を単一のスキーマ内で宣言する通常の場合は、すべての型名を INTYPE ファイル内のスキーマ名で修飾することが最も安全です。

デフォルトの名前のマッピング

オブジェクト型または属性に対する C または C++ の識別子名を作成する場合、OTT ユーティリティは、その名前をデータベース・キャラクタ・セットから有効な C または C++ の識別子に変換します。最初に、データベース・キャラクタ・セットから OTT ユーティリティで使用するキャラクタ・セットに、名前が変換されます。次に、変換した結果の名前が INTYPE ファイルに指定されている場合は、その変換内容が使用されます。それ以外の場合、OTT ユーティリティは、CASE パラメータに指定されている大 / 小文字を適用して、名前を文字ごとにコンパイラのキャラクタ・セットに変換します。詳細を次に説明します。

OTT ユーティリティがデータベース・エンティティの名前を読み込むと、その名前は、データベース・キャラクタ・セットから OTT ユーティリティが使用するキャラクタ・セットに自動的に変換されます。OTT ユーティリティがデータベース・エンティティの名前を正常に読み込めるように、名前のすべての文字は OTT のキャラクタ・セットにある必要があります。ただし、文字のコードが 2 つのキャラクタ・セットで異なっている場合があります。

OTT ユーティリティで使用するキャラクタ・セットに必要なすべての文字が含まれていることを保証する最も簡単な方法は、データベース・キャラクタ・セットの内容と同一にすることです。ただし、OTT のキャラクタ・セットは、コンパイラのキャラクタ・セットのスーパーセットである必要があります。つまり、コンパイラのキャラクタ・セットが 7 ビット ASCII の場合は、OTT のキャラクタ・セットにはサブセットとして 7 ビット ASCII を含める必要があります。コンパイラのキャラクタ・セットが 7 ビット EBCDIC の場合は、OTT のキャラクタ・セットにはサブセットとして 7 ビット EBCDIC を含める必要があります。ユーザーは、OTT ユーティリティで使用するキャラクタ・セットを、環境変数 NLS_LANG を設定して指定するか、他のオペレーティング・システム固有のメカニズムによって指定します。

OTT ユーティリティは、読み込んだデータベース・エンティティの名前を、OTT ユーティリティで使用するキャラクタ・セットからコンパイラのキャラクタ・セットに変換します。名前の変換内容が INTYPE ファイルに指定されている場合、OTT ユーティリティはその変換内容を使用します。

それ以外の場合、OTT ユーティリティは、次のように名前の変換を行います。

1. OTT のキャラクタ・セットがマルチバイト・キャラクタ・セットの場合、名前の中で等価のシングルバイト文字を持つすべてのマルチバイト文字は、該当するシングルバイト文字に変換されます。
2. 名前は、OTT のキャラクタ・セットからコンパイラのキャラクタ・セットに変換されます。コンパイラのキャラクタ・セットは、US7ASCII などのシングルバイト・キャラクタ・セットです。
3. 文字の大 / 小文字は、CASE パラメータの定義に従って設定されます。C または C++ 識別子で無効な文字またはコンパイラのキャラクタ・セットに変換内容がない文字は、アンダースコア（_）に置換されます。1 文字でもアンダースコアに置換された場合、OTT ユーティリティは警告メッセージを表示します。名前の中のすべての文字がアンダースコアに置換された場合、OTT ユーティリティはエラー・メッセージを表示します。

文字単位の名前の変換では、コンパイラのキャラクタ・セットにあるアンダースコア、数字またはシングルバイト文字は変更されません。したがって、有効な C または C++ の識別子は変更されません。

名前の変換では、ウムラウト付きの o やアクサングラーブ付きの a などのアクセント付きシングルバイト文字を、それぞれ o、a またはアクセントなしに変換したり、マルチバイト文字をそれと等価のシングルバイトに変換する場合があります。名前に等価のシングルバイトがないマルチバイト文字がある場合、通常、その名前の変換はエラーとなります。この場合、ユーザーは、INTYPE ファイルで名前の変換を指定する必要があります。

C 言語の同一の名前に複数のデータベース識別子がマッピングされている場合に生じる名前の競合は、OTT ユーティリティでは検出されません。また、データベース識別子が C キーワードにマッピングされるときの名前の問題も検出されません。

OTT ユーティリティに影響を与える制限 : ファイル名の比較

現在、OTT ユーティリティでは、コマンドラインまたは INTYPE ファイルにユーザーが指定したファイル名を比較することによって、2 つのファイルが同一かどうかを判断しています。しかし、OTT ユーティリティで 2 つのファイル名が同一のファイルを参照しているかどうかを判別する必要がある場合、1 つの潜在的な問題が生じます。たとえば、OTT で生成されたファイル foo1.h で、foo1.h に記述されている型宣言と、/private/smith/foo1.h に記述されている別の型宣言が必要な場合、OTT ユーティリティは、2 つのファイルが同一の場合は 1 つの `#include` を、異なる場合は 2 つの `#include` を生成する必要があります。しかし、実際には OTT は 2 つのファイルが異なると見なし、次のように 2 つの `#include` を生成します。

```
#ifndef FOO1_ORACLE
#include "foo1.h"
#endif
#ifndef FOO1_ORACLE
#include "/private/smith/foo1.h"
#endif
```

foo1.h と /private/smith/foo1.h が異なるファイルである場合は、最初のファイルのみがインクルードされることになります。foo1.h と /private/smith/foo1.h が同一ファイルである場合は、冗長な `#include` が記述されることになります。

そのため、1 つのファイルをコマンドラインまたは INTYPE ファイルに複数回指定する場合は、ファイルの各指定では同じファイル名を正確に使用する必要があります。

第 II 部

OCCI API リファレンス

第Ⅱ部には、次の章があります。

- 第 8 章「OCCI のクラスとメソッド」

8

OCCI のクラスとメソッド

この章では、C++ 用の OCCI のクラスとメソッドについて説明します。

次の項目について説明します。

OCCI クラスの概要

表 8-1 は、すべての OCCI クラスに関する簡単な説明です。この項の後で、各クラスとそのメソッドの詳細を説明します。

表 8-1 OCCI クラス

クラス	説明
「Bfile クラス」（8-5 ページ）	SQL BFILE 値にアクセスできます。
「Blob クラス」（8-13 ページ）	SQL BLOB 値にアクセスできます。
「Bytes クラス」（8-24 ページ）	バイトの比較、バイトの検索およびバイトの抽出について、一連のバイトを個別に検査します。
「Clob クラス」（8-27 ページ）	SQL CLOB 値にアクセスできます。
「Connection クラス」（8-40 ページ）	特定のデータベースへの接続を示します。
「ConnectionPool クラス」（8-46 ページ）	特定のデータベースへの接続プールを示します。
「Date クラス」（8-51 ページ）	SQL DATE データ項目の抽象化を指定します。また、日付値の OCCI エスケープ構文をサポートするための書式操作と解析操作も提供します。
「Environment クラス」（8-63 ページ）	OCCI オブジェクトのメモリーや他のリソースを管理するための OCCI 環境を提供します。OCCI ドライバ・マネージャは OCI 環境ハンドルにマップされます。
「IntervalDS クラス」（8-70 ページ）	期間を日、時、分および秒で示します。
「IntervalYM クラス」（8-82 ページ）	期間を年と月で示します。
「Map クラス」（8-92 ページ）	SQL 構造型のマッピングを C++ クラスに格納するために使用します。
「MetaData クラス」（8-93 ページ）	データベースの既存のスキーマ・オブジェクトまたはデータベース全体で、ResultSet の列の型とプロパティを決定するために使用します。
「Number クラス」（8-100 ページ）	桁の丸め動作を制御できます。
「PObject クラス」（8-124 ページ）	型を定義するときに、永続インスタンスまたは一時インスタンスを指定できます。PObject から導出されたクラス・インスタンスは、永続または一時のいずれかにできます。永続の場合、PObject から導出したクラス・インスタンスは、PObject クラスを継承します。一時の場合、継承はありません。
「Ref クラス」（8-130 ページ）	SQL REF 値の C++ でのマッピング。このマッピングは、データベースにある SQL 構造型の値への参照です。

表 8-1 OCCI クラス（続き）

クラス	説明
「RefAny クラス」（8-137 ページ）	SQL REF 値の C++ でのマッピング。このマッピングは、データベースにある SQL 構造型の値への参照です。
「ResultSet クラス」（8-141 ページ）	OCCI の Statement の実行で生成されたデータの表にアクセスできます。
「SQLException クラス」（8-163 ページ）	データベース・アクセス・エラーの情報を提供します。
「Statement クラス」（8-165 ページ）	SQL 文を実行するために使用します。SQL 文には、問合せ文および INSERT/UPDATE/DELETE 文が含まれます。
「Stream クラス」（8-209 ページ）	プリコンパイルされた DML 文またはストアド・プロシージャ・コールにストリーム・データ（通常は、LONG データ型）を渡すために使用します。
「Timestamp クラス」（8-213 ページ）	SQL TIMESTAMP データ項目の抽象化を指定します。また、タイム・スタンプ値の OCCI エスケープ構文をサポートするための書式操作と解析操作も提供します。

OCCI のクラスとメソッド

OCCI クラスは、oracle::occi ネームスペースに定義されています。oracle::occi ネームスペース内の OCCI クラス名は、次の 3 つの方法の 1 つを使用して参照できます。

- 各 OCCI クラス名に対してスコープ解決演算子（::）を使用します。
- 各 OCCI クラス名に対して using 宣言を使用します。
- すべての OCCI クラス名に対して using ディレクティブを使用します。

スコープ解決演算子

スコープ解決演算子（::）は、oracle::occi ネームスペースと OCCI クラス名を明示的に指定するために使用されます。Connection オブジェクトである myConnection を宣言するには、スコープ解決演算子を使用して、次の構文で宣言します。

```
oracle::occi::Connection myConnection;
```

using 宣言

using 宣言は、OCCI クラス名をコンパイル・ユニットで競合せずに使用できる場合に使用されます。oracle::occi ネームスペースで OCCI クラス名を宣言するには、次の構文を使用します。

```
using oracle::occi::Connection;
```

この宣言により、Connection は oracle::occi::Connection を参照し、次のように myConnection を宣言できます。

```
Connection myConnection;
```

using ディレクティブ

using ディレクティブは、すべての OCCI クラス名をコンパイル・ユニットで競合せずに使用できる場合に使用されます。oracle::occi ネームスペースですべての OCCI クラス名を宣言するには、次の構文を使用します。

```
using oracle::occi;
```

次に、using 宣言と同じように、次の宣言によって OCCI クラス Connection を参照できます。

```
Connection myConnection;
```

Bfile クラス

Bfile クラスは、BFILE 型オブジェクトの共通プロパティを定義します。BFILE とは、Oracle データベースの外のオペレーティング・システム・ファイルに格納されているラージ・バイナリ・ファイルです。Bfile オブジェクトには、BFILE への論理ポインタが含まれていますが、BFILE 自体は含まれていません。

Bfile クラスのメソッドを使用すると、Bfile オブジェクトに関連する特定のタスクを実行できます。

getBfile() や setBfile() など、ResultSet クラスと Statement クラスのメソッドによって、SQL BFILE の値にアクセスできます。

NULL の Bfile オブジェクトを作成するには、次の構文を使用します。

```
Bfile();
```

NULL の Bfile オブジェクトに対して有効なメソッドは、setNull()、isNull() および operator=() のみです。

未初期化の Bfile オブジェクトを構築するには、次の構文を使用します。

```
Bfile(const Connection *connectionp);
```

未初期化の Bfile オブジェクトは、次の方法で初期化できます。

- setName() メソッドを使用します。次に、この BFILE を表に挿入して BFILE を変更すると、SELECT ... FOR UPDATE を使用してその内容を取り出すことができます。write() メソッドによって BFILE は変更されますが、変更されたデータは、トランザクションがコミットされたときのみ表にフラッシュされます。INSERT は必要ありません。
- 初期化した Bfile オブジェクトを割り当てます。

既存の Bfile オブジェクトのコピーを作成するには、次の構文を使用します。

```
Bfile(const Bfile &srcBfile);
```

Bfile メソッドの概要

表 8-2 Bfile メソッド

メソッド	概要
「close()」（8-7 ページ）	前にオープンした BFILE をクローズします。
「closeStream()」（8-7 ページ）	BFILE から取得したストリームをクローズします。
「fileExists()」（8-7 ページ）	BFILE があるかどうかを確認します。
「getDirAlias()」（8-7 ページ）	BFILE のディレクトリ別名を戻します。
「getFileName()」（8-7 ページ）	BFILE の名前を戻します。
「getStream()」（8-8 ページ）	Stream オブジェクトとして BFILE からデータを戻します。
「isInitialized()」（8-8 ページ）	Bfile オブジェクトが初期化されているかどうかを確認します。
「isNull()」（8-8 ページ）	Bfile オブジェクトがアトミック NULL かどうかを確認します。
「isOpen()」（8-9 ページ）	BFILE がオープンしているかどうかを確認します。
「length()」（8-9 ページ）	BFILE のバイト数を戻します。
「open()」（8-9 ページ）	BFILE を読取り専用アクセスでオープンします。
「operator=()」（8-9 ページ）	BFILE ロケータを Bfile オブジェクトに割り当てます。
「operator==()」（8-10 ページ）	2 つの Bfile オブジェクトが等しいかどうかを確認します。
「operator!=()」（8-10 ページ）	2 つの Bfile オブジェクトが等しくないかどうかを確認します。
「read()」（8-10 ページ）	BFILE の特定部分をバッファに読み込みます。
「setName()」（8-11 ページ）	BFILE のディレクトリ別名とファイル名を設定します。
「setNull()」（8-12 ページ）	Bfile オブジェクトをアトミック NULL に設定します。

close()

このメソッドは、前にオープンした BFILE をクローズします。

構文

```
void close();
```

closeStream()

このメソッドは、BFILE から取得したストリームをクローズします。

構文

```
void closeStream(Stream *stream);
```

パラメータ

stream
クローズするストリームを指定します。

fileExists()

このメソッドは、BFILE があるかどうかをテストします。BFILE がある場合は true を、それ以外の場合は false を戻します。

構文

```
bool fileExists() const;
```

getDirAlias()

このメソッドは、BFILE に関連付けられているディレクトリ別名を持つ文字列を戻します。

構文

```
string getDirAlias() const;
```

getFileName()

このメソッドは、BFILE に関連付けられているファイル名を持つ文字列を戻します。

構文

```
string getFileName() const;
```

getStream()

このメソッドは、BFILE から読み込まれた Stream オブジェクトを戻します。ストリームがすでにオープンしている場合は、その Bfile オブジェクトで別のストリームをオープンすることはできません。ストリームは、Bfile オブジェクト操作を実行する前にクローズする必要があります。

構文

```
Stream* getStream(unsigned int offset = 1,
   unsigned int amount = 0);
```

パラメータ

offset

BFILE から読み込むデータの開始位置を指定します。*offset* が未指定の場合、データは BLOB の最初から書き込まれます。

次の値が有効です。

1 以上の数値。

amount

BFILE から読み込むバイトの合計数を指定します。*amount* が 0（ゼロ）の場合は、入力 *offset* の位置から BFILE の終わりまでのデータがストリーム・モードで読み込まれます。

isInitialized()

このメソッドは、Bfile オブジェクトが初期化されているかどうかを確認します。Bfile オブジェクトが初期化されている場合は true を、それ以外の場合は false を戻します。

構文

```
bool isInitialized() const;
```

isNull()

このメソッドは、Bfile オブジェクトがアトミック NULL かどうかを確認します。Bfile オブジェクトがアトミック NULL の場合は true を、それ以外の場合は false を戻します。

構文

```
bool isNull() const;
```

isOpen()

このメソッドは、BFILE がオープンしているかどうかを確認します。BFILE は、この Bfile オブジェクトでのコールによってオープンされた場合にのみ、オープンしているとみなされます。（別の Bfile オブジェクトがこのファイルをオープンしている場合があります。これは、このファイルを異なる複数の Bfile オブジェクトに関連付けることで、同一のファイルの複数のオープンが実行できるためです）。BFILE がオープンしている場合は true を、それ以外の場合は false を戻します。

構文

```
bool isOpen() const;
```

length()

このメソッドは、BFILE のバイト数（ファイル・マーカーの終了も含めて）を戻します。

構文

```
unsigned int length() const;
```

open()

このメソッドは、既存の BFILE を読取り専用アクセスでオープンします。この関数は、1 つの Bfile オブジェクトに対して初めてコールされたときに有効となります。

構文

```
void open();
```

operator=()

このメソッドは、Bfile オブジェクトを現行の Bfile オブジェクトに割り当てます。ソースの Bfile オブジェクトは、この Bfile オブジェクトがデータベースに格納されている場合にのみ、この Bfile オブジェクトに割り当てられます。

構文

```
Bfile& operator=(const Bfile &srcBfile);
```

パラメータ

srcBfile
現行の Bfile オブジェクトに割り当てる Bfile オブジェクトを指定します。

operator==()

このメソッドは、2 つの Bfile オブジェクトを比較して、等しいかどうかを調べます。2 つの Bfile オブジェクトが同じ BFILE を参照している場合、それらのオブジェクトは等しいと判断されます。2 つの Bfile オブジェクトが NULL の場合は、false を戻します。2 つの Bfile オブジェクトが等しい場合は true を、それ以外の場合は false を戻します。

構文

```
bool operator==(const Bfile &srcBfile) const;
```

パラメータ

srcBfile
現行の Bfile オブジェクトと比較する Bfile オブジェクトを指定します。

operator!=()

このメソッドは、2 つの Bfile オブジェクトを比較して、等しくないかどうかを調べます。2 つの Bfile オブジェクトが同じ BFILE を参照している場合、それらのオブジェクトは等しいと判断されます。2 つの Bfile オブジェクトが等しくない場合は true を、それ以外の場合は false を戻します。

構文

```
bool operator!=(const Bfile &srcBfile) const;
```

パラメータ

srcBfile
現行の Bfile オブジェクトと比較する Bfile オブジェクトを指定します。

read()

このメソッドは、BFILE の一部またはすべてを指定されたバッファに読み込み、読み込まれたバイト数を戻します。

構文

```
unsigned int read(unsigned int amt,
   unsigned char *buffer,
   unsigned int bufsize,
   unsigned int offset = 1) const;
```

パラメータ

amt
読み込むバイト数を指定します。

次の値が有効です。

1 以上の数値。

buffer
BFILE データを読み込むバッファを指定します。

次の値が有効です。

amt 以上の数値。

bufsize
BFILE データを読み込むバッファのサイズを指定します。

次の値が有効です。

amt 以上の数値。

offset
BFILE から読み込むデータの開始位置を指定します。*offset* が未指定の場合、データは BFILE の最初から書き込まれます。

setName()

このメソッドは、BFILE のディレクトリ別名とファイル名を設定します。

構文

```
void setName(const string &dirAlias,
   const string &fileName);
```

パラメータ

dirAlias
BFILE に関連付けるディレクトリ別名を指定します。

fileName
BFILE に関連付けるファイル名を指定します。

setNull()

このメソッドは、Bfile オブジェクトをアトミック NULL に設定します。

構文

```
void setNull();
```

Blob クラス

Blob クラスは、BLOB 型オブジェクトの共通プロパティを定義します。BLOB は、データベース表の行の列値として格納されるラージ・バイナリ・オブジェクトです。Blob オブジェクトには、BLOB への論理ポインタが含まれていますが、BLOB 自体は含まれていません。

Blob クラスのメソッドを使用すると、Blob オブジェクトに関連する特定のタスクを実行できます。

getBlob() や setBlob() など、ResultSet クラスと Statement クラスのメソッドによって、SQL BLOB の値にアクセスできます。

NULL の Blob オブジェクトを作成するには、次の構文を使用します。

```
Blob();
```

NULL の Blob オブジェクトに対して有効なメソッドは、setNull()、isNull() および operator=() のみです。

未初期化の Blob オブジェクトを構築するには、次の構文を使用します。

```
Blob(const Connection *connectionp);
```

未初期化の Blob オブジェクトは、次の方法で初期化できます。

- setEmpty() メソッドを使用します。次に、この BLOB を表に挿入して BLOB を変更すると、SELECT ... FOR UPDATE を使用してその内容を取り出すことができます。write() メソッドによって BLOB は変更されますが、変更されたデータは、トランザクションがコミットされたときのみ表にフラッシュされます。UPDATE は必要ありません。
- 初期化した Blob オブジェクトを割り当てます。

Blob オブジェクトのコピーを作成するには、次の構文を使用します。

```
Blob(const Blob &srcBlob);
```

Blob メソッドの概要

表 8-3 Blob メソッド

メソッド	概要
「append()」（8-15 ページ）	指定した BLOB を現行の BLOB の最後に追加します。
「close()」（8-15 ページ）	前にオープンした BLOB をクローズします。
「closeStream()」（8-15 ページ）	BLOB から取得した Stream オブジェクトをクローズします。
「copy()」（8-16 ページ）	BFILE または BLOB の指定部分を現行の BLOB にコピーします。
「getChunkSize()」（8-17 ページ）	BLOB のチャンク・サイズを戻します。
「getStream()」（8-17 ページ）	Stream オブジェクトとして BLOB からデータを戻します。
「isInitialized()」（8-17 ページ）	Blob オブジェクトが初期化されているかどうかを確認します。
「isNull()」（8-18 ページ）	Blob オブジェクトがアトミック NULL かどうかを確認します。
「isOpen()」（8-18 ページ）	BLOB がオープンしているかどうかを確認します。
「length()」（8-18 ページ）	BLOB のバイト数を戻します。
「open()」（8-18 ページ）	BLOB を読取り専用または読取り / 書込みアクセスでオープンします。
「operator=()」（8-19 ページ）	BLOB ロケータを Blob オブジェクトに割り当てます。
「operator==()」（8-19 ページ）	2 つの Blob オブジェクトが等しいかどうかを確認します。
「operator!= ()」（8-19 ページ）	2 つの Blob オブジェクトが等しくないかどうかを確認します。
「read()」（8-20 ページ）	BLOB の一部をバッファに読み込みます。
「setEmpty()」（8-21 ページ）	Blob オブジェクトを空に設定します。
「setEmpty()」（8-21 ページ）	Blob オブジェクトを空に設定し、渡されたパラメータへの接続ポインタを初期化します。
「setNull()」（8-21 ページ）	Blob オブジェクトをアトミック NULL に設定します。

表 8-3 Blob メソッド（続き）

メソッド	概要
「trim()」（8-21 ページ）	BLOB を指定した長さに切り捨てます。
「write()」（8-22 ページ）	バッファをオープンしていない BLOB に書き込みます。
「writeChunk()」（8-23 ページ）	バッファをオープンしている BLOB に書き込みます。

append()

このメソッドは、BLOB を現行の BLOB の最後に追加します。

構文

```
void append(const Blob &srcBlob);
```

パラメータ

srcBlob
現行の BLOB に追加する BLOB を指定します。

close()

このメソッドは、BLOB をクローズします。

構文

```
void close();
```

closeStream()

このメソッドは、BLOB から取得した Stream オブジェクトをクローズします。

構文

```
void closeStream(Stream *stream);
```

パラメータ

stream
クローズする Stream オブジェクトを指定します。

copy()

このメソッドは、BFILE または BLOB の一部またはすべてを現行の BLOB にコピーします。

構文
様々な構文があります。

```
void copy(const Bfile &srcBfile,
   unsigned int numBytes,
   unsigned int dstOffset = 1,
   unsigned int srcOffset = 1);

void copy(const Blob &srcBlob,
   unsigned int numBytes,
   unsigned int dstOffset = 1,
   unsigned int srcOffset = 1);
```

パラメータ

srcBfile
データのコピー元の BFILE を指定します。

srcBlob
データのコピー元の BLOB を指定します。

numBytes
ソース BFILE またはソース BLOB からコピーするバイト数を指定します。

次の値が有効です。

1 以上の数値。

dstOffset
現行の BLOB に書き込むデータの開始位置を指定します。

次の値が有効です。

1 以上の数値。

srcOffset
ソース BFILE またはソース BLOB から読み込むデータの開始位置を指定します。

次の値が有効です。

1 以上の数値。

getChunkSize()

このメソッドは、BLOB のチャンク・サイズを戻します。

BLOB が格納される表を作成するとき、ユーザーはチャンク係数を指定できます。チャンク係数は、Oracle ブロックの倍数です。この数値は、BLOB に対するアクセスまたは修正時に、LOB データ・レイヤーが使用するチャンク・サイズに対応しています。

構文

```
unsigned int getChunkSize() const;
```

getStream()

このメソッドは、BLOB から Stream オブジェクトを戻します。ストリームがすでにオープンしている場合、別のストリームを Blob オブジェクトでオープンすることはできません。したがって、ユーザーは、Blob オブジェクト操作を実行する前に、ストリームを常にクローズする必要があります。

構文

```
Stream* getStream(unsigned int offset = 1,
   unsigned int amount = 0);
```

パラメータ

offset

BLOB から読み込むデータの開始位置を指定します。

次の値が有効です。

1 以上の数値。

amount

連続して読み込むバイト数の合計を指定します。*amount* が 0（ゼロ）の場合は、*offset* 値から BLOB の最後までのデータが読み込まれます。

isInitialized()

このメソッドは、Blob オブジェクトが初期化されているかどうかを確認します。Blob オブジェクトが初期化されている場合は true を、それ以外の場合は false を戻します。

構文

```
bool isInitialized() const;
```

isNull()

このメソッドは、Blob オブジェクトがアトミック NULL かどうかを確認します。Blob オブジェクトがアトミック NULL の場合は true を、それ以外の場合は false を戻します。

構文

```
bool isNull() const;
```

isOpen()

このメソッドは、BLOB がオープンしているかどうかを確認します。BLOB がオープンしている場合は true を、それ以外の場合は false を戻します。

構文

```
bool isOpen() const;
```

length()

このメソッドは、BLOB のバイト数を戻します。

構文

```
unsigned int length() const;
```

open()

このメソッドは、BLOB を読取り / 書込みまたは読取り専用モードでオープンします。

構文

```
void open(LobOpenMode mode = OCCI_LOB_READWRITE);
```

パラメータ

mode

BLOB をオープンするモードを指定します。

次の値が有効です。

OCCI_LOB_READWRITE

OCCI_LOB_READONLY

operator=()

このメソッドは、BLOB を現行の BLOB に割り当てます。割当て元 BLOB は、割当て先 BLOB が表に格納されているときのみ、その割当て先 BLOB にコピーされます。

構文

```
Blob& operator=(const Blob &srcBlob);
```

パラメータ

srcBlob
データのコピー元の BLOB を指定します。

operator==()

このメソッドは、2 つの Blob オブジェクトを比較して、等しいかどうかを調べます。2 つの Blob オブジェクトが同じ BLOB を参照している場合、それらのオブジェクトは等しいと判断されます。2 つの Blob オブジェクトが NULL の場合は、等しいとは判断されません。2 つの Blob オブジェクトが等しい場合は true を、それ以外の場合は false を戻します。

構文

```
bool operator==(const Blob &srcBlob) const;
```

パラメータ

srcBlob
現行の Blob オブジェクトと比較する Blob オブジェクトを指定します。

operator!= ()

このメソッドは、2 つの Blob オブジェクトを比較して、等しくないかどうかを調べます。2 つの Blob オブジェクトが同じ BLOB を参照している場合、それらのオブジェクトは等しいと判断されます。2 つの Blob オブジェクトが NULL の場合は、等しいとは判断されません。2 つの Blob オブジェクトが等しくない場合は true を、それ以外の場合は false を戻します。

構文

```
bool operator!=(const Blob &srcBlob) const;
```

パラメータ

srcBlob
現行の Blob オブジェクトと比較する Blob オブジェクトを指定します。

read()

このメソッドは、BLOB の一部またはすべてをバッファに読み込みます。実際に読み込まれたバイト数が戻ります。

構文
```
unsigned int read(unsigned int amt,
   unsigned char *buffer,
   unsigned int bufsize,
   unsigned int offset = 1) const;
```

パラメータ

amt
BLOB から連続して読み込むバイト数を指定します。

buffer
BLOB データを読み込むバッファを指定します。

bufsize
バッファのサイズを指定します。

次の値が有効です。

amt 以上の数値。

offset
BLOB から読み込むデータの開始位置を指定します。*offset* が未指定の場合、データは BLOB の最初から書き込まれます。

次の値が有効です。

1 以上の数値。

setEmpty()

このメソッドは、Blob オブジェクトを空に設定します。

構文

```
void setEmpty();
```

setEmpty()

このメソッドは、Blob オブジェクトを空に設定し、渡されたパラメータへの接続ポインタを初期化します。

構文

```
void setEmpty(const Connection* connectionp);
```

パラメータ

connectionp

Blob オブジェクトの新しい接続ポインタを指定します。

setNull()

このメソッドは、Blob オブジェクトをアトミック NULL に設定します。

構文

```
void setNull();
```

trim()

このメソッドは、BLOB を新しく指定した長さに切り捨てます。

構文

```
void trim(unsigned int newlen);
```

パラメータ

newlen
BLOB の新しい長さを指定します。

次の値が有効です。

BLOB の現在の長さ以下の数値。

write()

このメソッドは、バッファのデータを BLOB に書き込みます。このメソッドは、BLOB を暗黙的にオープンしてバッファのデータを BLOB にコピーし、BLOB を暗黙的にクローズします。BLOB がすでにオープンしている場合は、かわりに writeChunk() を使用します。実際に書き込まれたバイト数が戻ります。

構文
```
unsigned int write(unsigned int amt,
   unsigned char *buffer,
   unsigned int bufsize,
   unsigned int offset = 1);
```

パラメータ

amt
BLOB に連続して書き込むバイト数を指定します。

buffer
BLOB に書き込むデータが含まれているバッファを指定します。

bufsize
BLOB に書き込むデータが含まれているバッファのサイズを指定します。

次の値が有効です。

amt 以上の数値。

offset
BLOB に書き込むデータの開始位置を指定します。offset が未指定の場合、データは BLOB の最初から書き込まれます。

次の値が有効です。

1 以上の数値。

writeChunk()

このメソッドは、バッファのデータを前にオープンした BLOB に書き込みます。実際に書き込まれたバイト数が戻ります。

構文

```
unsigned int writeChunk(unsigned int amt,
   unsigned char *buffer,
   unsigned int bufsize,
   unsigned int offset = 1);
```

パラメータ

amt

BLOB に連続して書き込むバイト数を指定します。

buffer

書き込むデータが含まれているバッファを指定します。

bufsize

書き込むデータが含まれているバッファのサイズを指定します。

次の値が有効です。

amt 以上の数値。

offset

BLOB に書き込むデータの開始位置を指定します。*offset* が未指定の場合、データは BLOB の最初から書き込まれます。

次の値が有効です。

1 以上の数値。

Bytes クラス

Bytes クラスのメソッドを使用すると、Bytes オブジェクトに関連する特定のタスクを実行できます。

Bytes オブジェクトを作成するには、次の構文を使用します。

```
Bytes(Environment *env = NULL);
```

文字配列から、バイト・サブ配列が含まれる Bytes オブジェクトを作成するには、次の構文を使用します。

```
Bytes(unsigned char *value,
   unsigned int count,
   unsigned int offset = 0,
   Environment *env = NULL);
```

Bytes オブジェクトのコピーを作成するには、次の構文を使用します。

```
Bytes(const Bytes &e);
```

Bytes メソッドの概要

表 8-4 Bytes メソッド

メソッド	概要
「byteAt()」（8-25 ページ）	Bytes オブジェクトの指定した位置のバイトを戻します。
「getBytes()」（8-25 ページ）	Bytes オブジェクトからバイト配列を戻します。
「isNull()」（8-26 ページ）	Bytes オブジェクトが NULL かどうかをテストします。
「length()」（8-26 ページ）	Bytes オブジェクトのバイト数を戻します。
「setNull()」（8-26 ページ）	Bytes オブジェクトを NULL に設定します。

byteAt()

このメソッドは、Bytes オブジェクトの指定した位置のバイトを戻します。

構文

```
unsigned char byteAt(unsigned int index) const;
```

パラメータ

index

Bytes オブジェクトから戻されるバイトの位置を指定します。Bytes オブジェクトの最初のバイトには、0（ゼロ）を指定します。

getBytes()

このメソッドは、Bytes オブジェクトから指定したバイト配列にバイトをコピーします。

構文

```
void getBytes(unsigned char *dst,
   unsigned int count,
   unsigned int srcBegin = 0,
   unsigned int dstBegin = 0) const;
```

パラメータ

dst

Bytes オブジェクトのデータを書き込む宛先バッファを指定します。

count

コピーするバイト数を指定します。

srcBegin

Bytes オブジェクトから読み込むデータの開始位置を指定します。Bytes オブジェクトの最初のバイト位置は、0（ゼロ）です。

dstBegin

宛先バッファに書き込むデータの開始位置を指定します。*dst* の最初のバイト位置は、0（ゼロ）です。

isNull()

このメソッドは、Bytes オブジェクトがアトミック NULL かどうかをテストします。Bytes オブジェクトがアトミック NULL の場合は true を、それ以外の場合は false を戻します。

構文

```
bool isNull() const;
```

length()

このメソッドは、Bytes オブジェクトの長さを戻します。

構文

```
unsigned int length() const;
```

setNull()

このメソッドは、Bytes オブジェクトをアトミック NULL に設定します。

構文

```
void setNull();
```

Clob クラス

Clob クラスは、CLOB 型オブジェクトの共通プロパティを定義します。CLOB は、データベース表の行の列値として格納されるラージ・キャラクタ・オブジェクトです。Clob オブジェクトには、CLOB への論理ポインタが含まれていますが、CLOB 自体は含まれていません。

Clob クラスのメソッドを使用すると、Clob オブジェクトに関連する特定のタスクを実行できます。これには、SQL CLOB の長さを取得するためのメソッド、クライアント上の CLOB をマテリアライズするためのメソッドおよび CLOB の一部を抽出するためのメソッドも含まれます。

getClob() や setClob() など、ResultSet クラスと Statement クラスのメソッドによって、SQL CLOB の値にアクセスできます。

NULL の Clob オブジェクトを作成するには、次の構文を使用します。

```
Clob();
```

NULL の Clob オブジェクトに対して有効なメソッドは、setNull()、isNull() および operator=() のみです。

未初期化の Clob オブジェクトを構築するには、次の構文を使用します。

```
Clob(const Connection *connectionp);
```

未初期化の Clob オブジェクトは、次の方法で初期化できます。

- setEmpty() メソッドを使用します。次に、この CLOB を表に挿入して CLOB を変更すると、SELECT ... FOR UPDATE を使用してその内容を取り出すことができます。write() メソッドによって CLOB は変更されますが、変更されたデータは、トランザクションがコミットされたときのみ表にフラッシュされます。INSERT は必要ありません。
- 初期化した Clob オブジェクトを割り当てます。

Clob オブジェクトのコピーを作成するには、次の構文を使用します。

```
Clob(const Clob &srcClob);
```

Clob メソッドの概要

表 8-5 Clob メソッド

メソッド	概要
「append()」（8-29 ページ）	CLOB を現行の CLOB の最後に追加します。
「close()」（8-29 ページ）	前にオープンした CLOB をクローズします。
「closeStream()」（8-30 ページ）	CLOB から取得した Stream オブジェクトをクローズします。
「copy()」（8-30 ページ）	CLOB または BFILE のすべてまたは一部を現行の CLOB にコピーします。
「getCharSetForm()」（8-31 ページ）	CLOB のキャラクタ・セット・フォームを戻します。
「getCharSetId()」（8-31 ページ）	CLOB のキャラクタ・セット ID を戻します。
「getChunkSize()」（8-31 ページ）	CLOB のチャンク・サイズを戻します。
「getStream()」（8-32 ページ）	Stream オブジェクトとして CLOB からデータを戻します。
「isInitialized()」（8-32 ページ）	Clob オブジェクトが初期化されているかどうかを確認します。
「isNull()」（8-32 ページ）	Clob オブジェクトがアトミック NULL かどうかを確認します。
「isOpen()」（8-33 ページ）	CLOB がオープンしているかどうか確認します。
「length()」（8-33 ページ）	現行の CLOB の文字数を戻します。
「open()」（8-33 ページ）	CLOB を読取り専用または読取り / 書込みアクセスでオープンします。
「operator=()」（8-33 ページ）	CLOB ロケータを現行の Clob オブジェクトに割り当てます。
「operator==()」（8-34 ページ）	2 つの Clob オブジェクトが等しいかどうかを確認します。
「operator!=()」（8-34 ページ）	2 つの Clob オブジェクトが等しくないかどうかを確認します。
「read()」（8-35 ページ）	CLOB の一部をバッファに読み込みます。
「setCharSetId()」（8-36 ページ）	CLOB に関連付けられたキャラクタ・セット ID を設定します。
「setCharSetForm()」（8-36 ページ）	CLOB に関連付けられたキャラクタ・セット・フォームを設定します。

表 8-5 Clob メソッド（続き）

メソッド	概要
「setEmpty()」（8-36 ページ）	Clob オブジェクトを空に設定します。
「setEmpty()」（8-37 ページ）	Clob オブジェクトを空に設定し、渡されたパラメータへの接続ポインタを初期化します。
「setNull()」（8-37 ページ）	Clob オブジェクトをアトミック NULL に設定します。
「trim()」（8-37 ページ）	CLOB を指定した長さに切り捨てます。
「write()」（8-38 ページ）	バッファをオープンしていない CLOB に書き込みます。
「writeChunk()」（8-39 ページ）	バッファをオープンしている CLOB に書き込みます。

append()

このメソッドは、CLOB を現行の CLOB の最後に追加します。

構文

```
void append(const Clob &srcClob);
```

パラメータ

srcClob
現行の CLOB に追加する CLOB を指定します。

close()

このメソッドは、CLOB をクローズします。

構文

```
void close();
```

closeStream()

このメソッドは、CLOB から取得した Stream オブジェクトをクローズします。

構文

```
void closeStream(Stream *stream);
```

パラメータ

stream
クローズする Stream オブジェクトを指定します。

copy()

このメソッドは、BFILE または CLOB の一部またはすべてを現行の CLOB にコピーします。

構文
様々な構文があります。

```
void copy(const Bfile &srcBfile,
   unsigned int numBytes,
   unsigned int dstOffset = 1,
   unsigned int srcOffset = 1);

void copy(const Clob &srcClob,
   unsigned int numBytes,
   unsigned int dstOffset = 1,
   unsigned int srcOffset = 1);
```

パラメータ

srcBfile
データのコピー元の BFILE を指定します。

srcClob
データのコピー元の CLOB を指定します。

numBytes
ソース BFILE またはソース CLOB からコピーする文字数を指定します。

次の値が有効です。

1 以上の数値。

dstOffset
現行の CLOB に書き込むデータのを開始位置を指定します。

次の値が有効です。

1 以上の数値。

srcOffset
ソース BFILE またはソース CLOB から読み込むデータの開始位置を指定します。

次の値が有効です。

1 以上の数値。

getCharSetForm()

このメソッドは、CLOB のキャラクタ・セット・フォームを戻します。

構文
```
CharSetForm getCharSetForm() const;
```

getCharSetId()

このメソッドは、CLOB のキャラクタ・セット ID を文字列形式で戻します。

構文
```
string getCharSetId() const;
```

getChunkSize()

このメソッドは、CLOB のチャンク・サイズを戻します。

CLOB が格納される表を作成するとき、ユーザーはチャンク係数を指定できます。チャンク係数は、Oracle ブロックの倍数です。この数値は、CLOB に対するアクセスおよび修正時に、LOB データ・レイヤーが使用するチャンク・サイズに対応しています。

構文
```
unsigned int getChunkSize() const;
```

getStream()

このメソッドは、CLOB から Stream オブジェクトを戻します。ストリームがすでにオープンしている場合は、別のストリームを Clob オブジェクトでオープンすることはできません。したがって、ユーザーは Clob オブジェクト操作を実行する前に、ストリームを常にクローズする必要があります。デフォルトでは、クライアントのキャラクタ・セット ID とフォームが使用されます。ただし、これらが、setCharSetId() と setCharSetForm() のコールで明示的に設定されていない場合にかぎります。

構文

```
Stream* getStream(unsigned int offset = 1,
   unsigned int amount = 0);
```

パラメータ

offset

CLOB から読み込むデータの開始位置を指定します。*offset* が未指定の場合、データは CLOB の最初から書き込まれます。

次の値が有効です。

1 以上の数値。

amount

連続して読み込む文字数の合計を指定します。*amount* が 0（ゼロ）の場合、*offset* 値から CLOB の最後までのデータが読み込まれます。

isInitialized()

このメソッドは、Clob オブジェクトが初期化されているかどうかを確認します。Clob オブジェクトが初期化されている場合は true を、それ以外の場合は false を戻します。

構文

```
bool isInitialized() const;
```

isNull()

このメソッドは、Clob オブジェクトがアトミック NULL かどうかを確認します。Clob オブジェクトがアトミック NULL の場合は true を、それ以外の場合は false を戻します。

構文

```
bool isNull() const;
```

isOpen()

このメソッドは、CLOB がオープンしているかどうかを確認します。CLOB がオープンしている場合は true を、それ以外の場合は false を戻します。

構文

```
bool isOpen() const;
```

length()

このメソッドは、CLOB の文字数を戻します。

構文

```
unsigned int length() const;
```

open()

このメソッドは、CLOB を読取り / 書込みまたは読取り専用モードでオープンします。

構文

```
void open(LObOpenMode mode = OCCI_LOB_READWRITE);
```

パラメータ

mode

CLOB をオープンするモードを指定します。

次の値が有効です。

OCCI_LOB_READWRITE

OCCI_LOB_READONLY

operator=()

このメソッドは、CLOB を現行の CLOB に割り当てます。割当て元 CLOB は、割当て先 CLOB が表に格納されているときのみ、その割当て先 CLOB にコピーされます。

構文

```
Clob& operator=(const Clob &srcClob);
```

パラメータ

srcClob
データのコピー元の CLOB を指定します。

operator==()

このメソッドは、2 つの Clob オブジェクトを比較して、等しいかどうかを調べます。2 つの Clob オブジェクトが同じ CLOB を参照している場合、それらのオブジェクトは等しいと判断されます。2 つの Clob オブジェクトが NULL の場合は、等しいとは判断されません。2 つの Clob オブジェクトが等しい場合は true を、それ以外の場合は false を戻します。

構文

```
bool operator==(const Clob &srcClob) const;
```

パラメータ

srcClob
現行の Clob オブジェクトと比較する Clob オブジェクトを指定します。

operator!=()

このメソッドは、2 つの Clob オブジェクトを比較して、等しくないかどうかを調べます。2 つの Clob オブジェクトが同じ CLOB を参照している場合、それらのオブジェクトは等しいと判断されます。2 つの Clob オブジェクトが NULL の場合は、等しいとは判断されません。2 つの Clob オブジェクトが等しくない場合は true を、それ以外の場合は false を戻します。

構文

```
bool operator!=(const Clob &srcClob) const;
```

パラメータ

srcClob
現行の Clob オブジェクトと比較する Clob オブジェクトを指定します。

read()

このメソッドは、CLOB の一部またはすべてをバッファに読み込みます。実際に読み込まれた文字数が戻ります。デフォルトでは、クライアントのキャラクタ・セット ID とフォームが使用されます。ただし、これらが、setCharSetId() と setCharSetForm() のコールで明示的に設定されていない場合にかぎります。

構文

```
unsigned int read(unsigned int amt,
   unsigned char *buffer,
   unsigned int bufsize,
   unsigned int offset = 1) const;
```

パラメータ

amt

CLOB から連続して読み込む文字数を指定します。

buffer

CLOB データを読み込むバッファを指定します。

bufsize

バッファのサイズを指定します。

次の値が有効です。

amt 以上の数値。

offset

CLOB から読み込むデータの開始位置を指定します。*offset* が未指定の場合、データは CLOB の最初から読み込まれます。

次の値が有効です。

1 以上の数値。

setCharSetId()

このメソッドは、Clob に関連付けられたキャラクタ・セット ID を設定します。キャラクタ・セット ID は、読取り / 書込みおよび getStream() 操作で使用されます。値が明示的に設定されていない場合は、デフォルトのクライアントのキャラクタ・セット ID が使用されます。

サポートされているキャラクタ・セットのリストは、『Oracle9*i* Database グローバリゼーション・サポート・ガイド』の付録 A を参照してください。

構文

```
void setCharSetId( const OCCI_STD_NAMESPACE::string &charset);
```

setCharSetForm()

CLOB に関連付けられたキャラクタ・セット・フォームを設定します。キャラクタ・セット・フォームは、読取り / 書込みおよび getStream() 操作で使用されます。値が明示的に設定されていない場合、デフォルトでは OCCI_SQLCS_IMPLICIT が使用されます。

構文

```
void setCharSetForm( CharSetForm csfrm );
```

パラメータ

csfrm

Clob のキャラクタ・セット・フォームを指定します。

setEmpty()

このメソッドは、Clob オブジェクトを空に設定します。

構文

```
void setEmpty();
```

setEmpty()

このメソッドは、Clob オブジェクトを空に設定し、渡されたパラメータへの接続ポインタを初期化します。

構文

```
void setEmpty( const Connection* connectionp);
```

パラメータ

connectionp

Clob オブジェクトの新しい接続ポインタを指定します。

setNull()

このメソッドは、Clob オブジェクトをアトミック NULL に設定します。

構文

```
void setNull();
```

trim()

このメソッドは、CLOB を新しく指定した長さに切り捨てます。

構文

```
void trim(unsigned int newlen);
```

パラメータ

newlen

CLOB の新しい長さを指定します。

次の値が有効です。

CLOB の現在の長さ以下の数値。

write()

このメソッドは、バッファのデータを CLOB に書き込みます。このメソッドは、CLOB を暗黙的にオープンしてバッファのデータを CLOB にコピーし、CLOB を暗黙的にクローズします。CLOB がすでにオープンしている場合は、かわりに writeChunk() を使用します。実際に書き込まれた文字数が戻ります。デフォルトでは、クライアントのキャラクタ・セット ID とフォームが使用されます。これらが、setCharSetId() と setCharSetForm() のコールで明示的に設定されていない場合にかぎります。

構文

```
unsigned int write(unsigned int amt,
   unsigned char *buffer,
   unsigned int bufsize,
   unsigned int offset = 1);
```

パラメータ

amt

CLOB に連続して書き込む文字数を指定します。

buffer

CLOB に書き込むデータが含まれているバッファを指定します。

bufsize

CLOB に書き込むデータが含まれているバッファのサイズを指定します。

次の値が有効です。

amt 以上の数値。

offset

CLOB に書き込むデータの開始位置を指定します。*offset* が未指定の場合、データは CLOB の最初から書き込まれます。

次の値が有効です。

1 以上の数値。

writeChunk()

このメソッドは、バッファのデータを前にオープンした CLOB に書き込みます。実際に書き込まれた文字数が戻ります。 デフォルトでは、クライアントのキャラクタ・セット ID とフォームが使用されます。これらが、setCharSetId() と setCharSetForm() のコールで明示的に設定されていない場合にかぎります。

構文

```
unsigned int writechunk(unsigned int amt,
   unsigned char *buffer,
   unsigned int bufsize,
   unsigned int offset = 1);
```

パラメータ

amt

CLOB に連続して書き込む文字数を指定します。

buffer

CLOB に書き込むデータが含まれているバッファを指定します。

bufsize

CLOB に書き込むデータが含まれているバッファのサイズを指定します。

次の値が有効です。

amt 以上の数値。

offset

CLOB に書き込むデータの開始位置を指定します。*offset* が未指定の場合、データは CLOB の最初から書き込まれます。

次の値が有効です。

1 以上の数値。

Connection クラス

Connection クラスは、特定のデータベースとの接続を示します。接続のコンテキスト内で、SQL 文が実行され、結果が戻ります。

接続を作成するには、次の構文を使用します。

```
Connection();
```

Connection メソッドの概要

表 8-6 Connection メソッド

メソッド	概要
「changePassword()」（8-41 ページ）	現行ユーザーのパスワードを変更します。
「commit()」（8-41 ページ）	前回のコミットまたはロールバック以降の変更をコミットし、セッションが保持するデータベース・ロックを解放します。
「createStatement()」（8-42 ページ）	SQL 文を実行する Statement オブジェクトを作成します。
「flushCache()」（8-42 ページ）	接続に関連付けられているオブジェクト・キャッシュをフラッシュします。
「getClientCharSet()」（8-42 ページ）	デフォルトのクライアント・キャラクタ・セットを戻します。
「getClientNCHARCharSet()」（8-42 ページ）	デフォルトのクライアント NCHAR キャラクタ・セットを戻します。
「getMetaData()」（8-43 ページ）	接続からアクセス可能なオブジェクトのメタデータを戻します。
「getOCIServer()」（8-44 ページ）	接続に関連付けられている OCI サーバー・コンテキストを戻します。
「getOCIServiceContext()」（8-44 ページ）	接続に関連付けられている OCI サービス・コンテキストを戻します。
「getOCISession()」（8-44 ページ）	接続に関連付けられている OCI セッション・コンテキストを戻します。

表 8-6 Connection メソッド（続き）

メソッド	概要
「rollback()」（8-44 ページ）	前回のコミットまたはロールバック以降の変更をすべてロールバックし、セッションが保持するデータベース・ロックを解放します。
「terminateStatement()」（8-45 ページ）	Statement オブジェクトをクローズし、そのオブジェクトに関連付けられているすべてのリソースを解放します。

changePassword()

このメソッドは、データベースに現在接続しているユーザーのパスワードを変更します。

構文

```
void changePassword(const string &user,
   const string &oldPassword,
   const string &newPassword);
```

パラメータ

user

データベースに現在接続しているユーザーを指定します。

oldPassword

ユーザーの現行のパスワードを指定します。

newPassword

ユーザーの新規パスワードを指定します。

commit()

このメソッドは、前回のコミットまたはロールバック以降の変更すべてをコミットし、セッションが現在保持しているデータベース・ロックを解放します。

構文

```
void commit();
```

createStatement()

このメソッドは、指定された SQL 文で Statement オブジェクトを作成します。

構文

```
Statement* createStatement(const string &sql ="");
```

パラメータ

sql

Statement オブジェクトに関連付ける SQL 文字列を指定します。

flushCache()

このメソッドは、接続に関連付けられているオブジェクト・キャッシュをフラッシュします。

構文

```
void flushCache();
```

getClientCharSet()

このメソッドは、セッションのキャラクタ・セットを戻します。

構文

```
string getClientCharSet() const;
```

getClientNCHARCharSet()

このメソッドは、セッションの NCHAR キャラクタ・セットを戻します。

構文

```
string getClientNCHARCharSet() const;
```

getMetaData()

このメソッドは、データベース内のオブジェクトのメタデータを戻します。

構文

様々な構文があります。

```
MetaData getMetaData(const string &object,
   MetaData::ParamType prmtyp = MetaData::PTYPE_UNK) const;

MetaData getMetaData(const RefAny &ref) const;
```

パラメータ

object

記述するスキーマ・オブジェクトを指定します。

prmtyp

記述するスキーマ・オブジェクトの型を指定します。可能な値は、MetaData::ParamType を使用して列挙できます。

次の値が有効です。

PTYPE_TABLE ― 表

PTYPE_VIEW ― ビュー

PTYPE_PROC ― プロシージャ

PTYPE_FUNC ― ファンクション

PTYPE_PKG ― パッケージ

PTYPE_TYPE ― 型

PTYPE_TYPE_ATTR ― 型の属性

PTYPE_TYPE_COLL ― コレクション型の情報

PTYPE_TYPE_METHOD ― 型のメソッド

PTYPE_SYN ― シノニム

PTYPE_SEQ ― 順序

PTYPE_COL ― 表またはビューの列

PTYPE_ARG ― ファンクションまたはプロシージャの引数

PTYPE_TYPE_ARG ― 型メソッドの引数

PTYPE_TYPE_RESULT − メソッドの結果

PTYPE_SCHEMA − スキーマ

PTYPE_DATABASE − データベース

PTYPE_UNK − 不明な型

ref
記述する型の型記述子オブジェクト（TDO）への REF を指定します。

getOCIServer()

このメソッドは、接続に関連付けられている OCI サーバー・コンテキストを戻します。

構文

```
LNOCIServer* getOCIServer() const;
```

getOCIServiceContext()

このメソッドは、接続に関連付けられている OCI サービス・コンテキストを戻します。

構文

```
LNOCISvcCtx* getOCIServiceContext() const;
```

getOCISession()

このメソッドは、接続に関連付けられている OCI セッション・コンテキストを戻します。

構文

```
LNOCISession* getOCISession() const;
```

rollback()

このメソッドは、前回のコミットまたはロールバック以降の変更すべてを取り消し、セッションが現在保持しているデータベース・ロックを解放します。

構文

```
void rollback();
```

terminateStatement()

このメソッドは、Statement オブジェクトをクローズし、そのオブジェクトに関連付けられているすべてのリソースを解放します。

構文

```
void terminateStatement(Statement *statement);
```

パラメータ

statement
クローズする Statement を指定します。

ConnectionPool クラス

ConnectionPool クラスは、特定のデータベースに対する接続プールを示します。

接続プールを作成するには、次の構文を使用します。

```
ConnectionPool();
```

ConnectionPool メソッドの概要

表 8-7 ConnectionPool メソッド

メソッド	概要
「createConnection()」（8-47 ページ）	プール接続を作成します。
「createProxyConnection()」（8-47 ページ）	プロキシ接続を作成します。
「getBusyConnections()」（8-48 ページ）	接続プール内のビジー接続数を戻します。
「getIncrConnections()」（8-48 ページ）	接続プール内の増分接続数を戻します。
「getMaxConnections()」（8-48 ページ）	接続プール内の最大接続数を戻します。
「getMinConnections()」（8-48 ページ）	接続プール内の最小接続数を戻します。
「getOpenConnections()」（8-49 ページ）	接続プール内のオープン接続数を戻します。
「getPoolName()」（8-49 ページ）	接続プールの名前を戻します。
「getTimeOut()」（8-49 ページ）	接続プール内の接続のタイムアウト間隔を戻します。
「setErrorOnBusy()」（8-49 ページ）	接続プールの全接続がビジーで、それ以上の接続をオープンできない場合に、SQLException が生成されることを指定します。
「setPoolSize()」（8-49 ページ）	接続プールについて、最小、最大および増分のプール接続数を設定します。
「setTimeOut()」（8-49 ページ）	接続プール内の接続のタイムアウト間隔を秒数で戻します。
「terminateConnection()」（8-50 ページ）	接続を破棄します。

createConnection()

このメソッドは、プール接続を作成します。

構文

```
Connection* createConnection(const string &userName,
   const string &password);
```

パラメータ

userName
接続するユーザー名を指定します。

password
ユーザーのパスワードを指定します。

createProxyConnection()

このメソッドは、接続プールからプロキシ接続を作成します。

構文

様々な構文があります。

```
Connection* createProxyConnection(cont string &name,
   Connection::ProxyType proxyType = Connection::PROXY_DEFAULT);

Connection* createProxyConnection(const string &name,
   string roles[],
   int numRoles,
   Connection::ProxyType proxyType = Connection::PROXY_DEFAULT);
```

パラメータ

name
接続するユーザー名を指定します。

roles
データベース・サーバー上でアクティブにするロールを指定します。

numRoles
データベース・サーバー上でアクティブにするロール数を指定します。

proxyType
実行するプロキシ認証の種類を指定します。

次の値が有効です。

PROXY_DEFAULT − データベース・ユーザー名を示します。

PROXY_EXTERNAL_AUTH − 外部ユーザー名を示します。

getBusyConnections()

このメソッドは、接続プール内のビジー接続数を戻します。

構文
```
unsigned int getBusyConnections() const;
```

getIncrConnections()

このメソッドは、接続プール内の増分接続数を戻します。

構文
```
unsigned int getIncrConnections() const;
```

getMaxConnections()

このメソッドは、接続プール内の最大接続数を戻します。

構文
```
unsigned int getMaxConnections() const;
```

getMinConnections()

このメソッドは、接続プール内の最小接続数を戻します。

構文
```
unsigned int getMinConnections() const;
```

getOpenConnections()

このメソッドは、接続プール内のオープン接続数を戻します。

構文

```
unsigned int getOpenConnections() const;
```

getPoolName()

このメソッドは、接続プールの名前を戻します。

構文

```
string getPoolName() const;
```

getTimeOut()

このメソッドは、接続プール内の接続のタイムアウト間隔を戻します。

構文

```
unsigned int getTimeOut() const;
```

setErrorOnBusy()

このメソッドは、接続プールの全接続がビジーで、それ以上の接続をオープンできない場合に、SQLException が生成されることを指定します。

構文

```
void setErrorOnBusy();
```

setPoolSize()

このメソッドは、接続プールについて、最小、最大および増分のプール接続数を設定します。

構文

```
void setPoolSize(unsigned int minConn = 0,
   unsigned int maxConn = 1,
   unsigned int incrConn = 1);
```

パラメータ

minConn
接続プールの最小接続数を指定します。

maxConn
接続プールの最大接続数を指定します。

incrConn
接続プールの増分接続数を指定します。

setTimeOut()

このメソッドは、接続プール内の接続のタイムアウト間隔を設定します。OCCI は、アイドル時間が指定タイムアウト間隔を超えている接続プールに関連している接続を、すべて終了します。

構文

```
void setTimeOut(unsigned int connTimeOut = 0);
```

パラメータ

connTimeOut
タイムアウト間隔を秒数で指定します。

terminateConnection()

このメソッドは、プール接続またはプロキシ接続を終了します。

構文

```
void terminateConnection(Connection *connection);
```

パラメータ

connection
終了するプール接続またはプロキシ接続を指定します。

Date クラス

Date クラスは、SQL DATE データ項目の抽象化を指定します。また、Date クラスは、書式操作と解析操作を追加して、日付値の OCCI エスケープ構文をサポートします。

SQL92 DATE は Oracle Date のサブセットであるため、このクラスはこの両方をサポートするために使用できます。

NULL の Date オブジェクトを作成するには、次の構文を使用します。

```
Date();
```

Date オブジェクトのコピーを作成するには、次の構文を使用します。

```
Date(const Date &a);
```

Date オブジェクトを作成するには、次の整数パラメータを使用します。

変数	値
year	0 を除いた -4712 ～ 9999
month	1 ～ 12
day	1 ～ 31
minutes	0 ～ 59
seconds	0 ～ 59

次の構文を使用します。

```
Date(const Environment *envp,
   int year = 1,
   unsigned int month = 1,
   unsigned int day = 1,
   unsigned int hour = 0,
   unsigned int minute = 0,
   unsigned int seconds = 0);
```

Date クラスのオブジェクトは、クライアント側の数値計算ではスタンドアロン・クラス・オブジェクトとして使用できます。また、データベースからのフェッチおよびデータベースへの設定にも使用できます。

次のコード例では、データベースから取り出された DATE 列値、Date オブジェクトを使用したバインド、およびスタンドアロン Date オブジェクトを使用した計算が示されています。

```
/* Create a connection */
Environment *env = Environment::createEnvironment(Environment::DEFAULT);
Connection *conn = env->createConnection(user, passwd, db);

/* Create a statement and associate a DML statement to it */
string sqlStmt = "SELECT job-id, start_date from JOB_HISTORY
                            where end_date = :x";
Statement *stmt = conn->createStatement(sqlStmt);

/* Create a Date object and bind it to the statement */
Date edate(env, 2000, 9, 3, 23, 30, 30);
stmt->setDate(1, edate);
ResultSet *rset = stmt->executeQuery();

/* Fetch a date from the database */
while(rset->next())
{
   Date sd = rset->getDate(2);
   Date temp = sd;    /*assignment operator */
   /* Methods on Date */
   temp.getDate(year, month, day, hour, minute, second);
   temp.setMonths(2);
   IntervalDS inter = temp.daysBetween(sd);
   .
   .
   .
}
```

Date メソッドの概要

表 8-8 Date メソッド

メソッド	概要
「addDays()」（8-54 ページ）	*n* 日を追加した Date オブジェクトを戻します。
「addMonths()」（8-54 ページ）	*n* か月を追加した Date オブジェクトを戻します。
「daysBetween()」（8-54 ページ）	現行の Date オブジェクトと指定した日付の間の日数を戻します。
「fromBytes()」（8-55 ページ）	Date オブジェクトの外部 Bytes 表現を Date オブジェクトに変換します。

表 8-8 Date メソッド（続き）

メソッド	概要
「fromText()」（8-55 ページ）	指定されたフォーマットと nls パラメータを使用して、日付を所定の入力文字列から変換します。
「getDate()」（8-55 ページ）	Date オブジェクトの日付と時刻のコンポーネントを戻します。
「getSystemDate()」（8-57 ページ）	システム日付が含まれている Date オブジェクトを戻します。
「isNull()」（8-57 ページ）	Date が NULL の場合は true を、それ以外の場合は false を戻します。
「lastDay()」（8-57 ページ）	月の最終日の Date を戻します。
「nextDay()」（8-57 ページ）	翌日の曜日の Date を戻します。
「operator=()」（8-58 ページ）	日付の値を別の日付に割り当てます。
「operator==()」（8-58 ページ）	a と b が同じ場合は true を、それ以外の場合は false を戻します。
「operator!=()」（8-58 ページ）	a と b が同じでない場合は true を、それ以外の場合は false を戻します。
「operator>()」（8-59 ページ）	a が b より後の場合は true を、それ以外の場合は false を戻します。
「operator>=()」（8-59 ページ）	a が b 以後の場合は true を、それ以外の場合は false を戻します。
「operator<()」（8-59 ページ）	a が b より前の場合は true を、それ以外の場合は false を戻します。
「operator<=()」（8-60 ページ）	a が b 以前の場合は true を、それ以外の場合は false を戻します。
「setDate()」（8-60 ページ）	入力された日付コンポーネントから日付を設定します。
「setNull()」（8-61 ページ）	Date オブジェクトをアトミック NULL に設定します。
「toBytes()」（8-61 ページ）	Date オブジェクトを外部 Bytes 表現に変換します。
「toText()」（8-61 ページ）	Date オブジェクトを文字列として取得します。
「toZone()」（8-62 ページ）	あるタイム・ゾーンから別のタイム・ゾーンに変換された Date オブジェクトを戻します。

addDays()

このメソッドは、指定した日数を Date オブジェクトに追加して、新しい日付を戻します。

構文

```
Date addDays(int i) const;
```

パラメータ

i
現行の Date オブジェクトに追加する日数を指定します。

addMonths()

このメソッドは、指定した月数を Date オブジェクトに追加して、新しい日付を戻します。

構文

```
Date addMonths(int i) const;
```

パラメータ

i
現行の Date オブジェクトに追加する月数を指定します。

daysBetween()

このメソッドは、現行の Date オブジェクトと指定した日付の間の日数を戻します。

構文

```
IntervalDS daysBetween(const Date &d) const;
```

パラメータ

d
間の日数を計算するために使用する日付を指定します。

fromBytes()

このメソッドは、Bytes オブジェクトを Date オブジェクトに変換します。

構文

```
void fromBytes(const Bytes &byteStream,
const Environment *envp = NULL);
```

パラメータ

byteStream
Bytes 形式の外部書式での日付。

envp
OCCI 環境を指定します。

fromText()

このメソッドは、文字列を Date オブジェクトに変換します。

構文

```
void fromText(const string &datestr,
   const string &fmt = "",
   const string &nlsParam = "",
   const Environment *envp = NULL);
```

パラメータ

datestr
変換する日付文字列を指定します。

fmt
書式文字列を指定します。

nlsParam
使用する nls パラメータを表す文字列を指定します。

envp
nls パラメータの取得元の環境を指定します。

getDate()

このメソッドは、日付を年、月、日、時、分および秒の日付コンポーネントのフォームで戻します。

構文

```
void getDate(int &year,
   unsigned int &month,
   unsigned int &day,
   unsigned int &hour,
   unsigned int &min,
   unsigned int &sec) const;
```

パラメータ

year
日付の年のコンポーネントを指定します。

month
日付の月のコンポーネントを指定します。

day
日付の日のコンポーネントを指定します。

hour
日付の時のコンポーネントを指定します。

min
日付の分のコンポーネントを指定します。

seconds
日付の秒のコンポーネントを指定します。

getSystemDate()

このメソッドは、システム日付を戻します。

構文

```
static Date getSystemDate(const Environment *envp);
```

パラメータ

envp
システム日付が戻される環境を指定します。

isNull()

このメソッドは、Date オブジェクトが NULL かどうかをテストします。Date オブジェクトが NULL の場合は true を、それ以外の場合は false を戻します。

構文

```
bool isNull() const;
```

lastDay()

このメソッドは、現在の月の最終日を示す日付を戻します。

構文

```
Date lastDay() const;
```

nextDay()

このメソッドは、指定した曜日の翌日を示す日付を戻します。

構文

```
Date nextDay(const string &dow) const;
```

パラメータ

dow
曜日を表す文字列を指定します。

operator=()

このメソッドは、等号（=）の右側にある日付オブジェクトを、等号（=）の左側にある日付オブジェクトに割り当てます。

構文

```
Date& operator=(const Date &d);
```

パラメータ

d
割り当てる日付オブジェクトを指定します。

operator==()

このメソッドは、指定した 2 つの日付を比較します。2 つの日付が等しい場合は `true` を、それ以外の場合は `false` を戻します。

構文

```
bool operator==(const Date &a,
   const Date &b);
```

パラメータ

a、b
等しいかどうかを比較する日付を指定します。

operator!=()

このメソッドは、指定した 2 つの日付を比較します。2 つの日付が等しくない場合は `true` を、それ以外の場合は `false` を戻します。

構文

```
bool operator!=(const Date &a,
   const Date &b);
```

パラメータ

a、b
等しくないかどうかを比較する日付を指定します。

operator>()

このメソッドは、指定した 2 つの日付を比較します。第 1 日付が第 2 日付に対して先の日付である場合は true を、それ以外の場合は false を戻します。いずれかの日付が NULL の場合は false を戻します。2 つの日付の種類が同じではない場合、false を戻します。

構文

```
bool operator>(const Date &a,
   const Date &b);
```

パラメータ

a、b
比較する日付を指定します。

operator>=()

このメソッドは、指定した 2 つの日付を比較します。第 1 日付が第 2 日付以降である場合は true を、それ以外の場合は false を戻します。いずれかの日付が NULL の場合は false を戻します。2 つの日付の種類が同じではない場合、false を戻します。

構文

```
bool operator>=(const Date &a,
   const Date &b);
```

パラメータ

a、b
比較する日付を指定します。

operator<()

このメソッドは、指定した 2 つの日付を比較します。第 1 日付が第 2 日付に対して前の日付である場合は true を、それ以外の場合は false を戻します。いずれかの日付が NULL の場合は false を戻します。2 つの日付の種類が同じではない場合、false を戻します。

構文

```
bool operator<(const Date &a,
   const Date &b);
```

パラメータ

a、b
比較する日付を指定します。

operator<=()

このメソッドは、指定した 2 つの日付を比較します。第 1 日付が第 2 日付以前である場合は true を、それ以外の場合は false を戻します。いずれかの日付が NULL の場合は false を戻します。2 つの日付の種類が同じではない場合、false を戻します。

構文

```
bool operator<=(const Date &a,
   const Date &b);
```

パラメータ

a、b
比較する日付を指定します。

setDate()

このメソッドは、日付を指定した値に設定します。

構文

```
void setDate(int year = 1,
   unsigned int month = 1,
   unsigned int day = 1,
   unsigned int hour = 0,
   unsigned int minute = 0,
   unsigned int seconds = 0);
```

パラメータ

year
年の値を指定する引数を指定します。

有効な値は、-4713 ～ 9999 です。

month
月の値を指定する引数を指定します。

有効な値は、1 ～ 12 です。

day
日の値を指定する引数を指定します。

有効な値は、1 ～ 31 です。

hour
時の値を指定する引数を指定します。

有効な値は、0 ～ 23 です。

minute
分の値を指定する引数を指定します。

有効な値は、0 ～ 59 です。

seconds
秒の値を指定する引数を指定します。

有効な値は、0 ～ 59 です。

setNull()

このメソッドは、日付をアトミック NULL に設定します。

構文

```
void setNull();
```

toBytes()

このメソッドは、日付を Bytes 表現で戻します。

構文

```
Bytes toBytes() const;
```

toText()

このメソッドは、*fmt* と *nlsParam* を使用して日付書式化された値の文字列を戻します。

構文

```
string toText(const string &fmt = "",
   const string &nlsParam = "") const;
```

パラメータ

fmt
書式文字列を指定します。

nlsParam
使用する nls パラメータを表す文字列を指定します。

toZone()

このメソッドは、あるタイム・ゾーンから別のタイム・ゾーンに変換された Date 値を戻します。

次の値が有効です。

ゾーン・コード	値
AST、ADT	大西洋標準時または大西洋夏時間
BST、BDT	ベーリング標準時またはベーリング夏時間
CST、CDT	中部標準時または中部夏時間
EST、EDT	東部標準時または東部夏時間
GMT	グリニッジ標準時
HST、HDT	アラスカ・ハワイ標準時またはアラスカ・ハワイ夏時間
MST、MDT	山地標準時または山地夏時間
NST	ニューファンドランド標準時
PST、PDT	太平洋標準時または太平洋夏時間
YST、YDT	ユーコン標準時またはユーコン夏時間

構文

```
Date toZone(const string &zone1,
   const string &zone2) const;
```

パラメータ

zone1

変換元タイム・ゾーンを表す文字列を指定します。

zone2

変換先タイム・ゾーンを表す文字列を指定します。

Environment クラス

Environment クラスは、OCCI オブジェクトのメモリーやその他のリソースを管理するための OCCI 環境を提供します。

アプリケーションは複数の OCCI 環境を保持できます。各環境ごとに、独自のヒープやスレッド・セーフティ mutex などを持つことができます。

Environment オブジェクトを作成するには、次の構文を使用します。

```
Environment()
enum Mode
{
   DEFAULT = OCI_DEFAULT,
   OBJECT = OCI_OBJECT,
   SHARED = OCI_SHARED,
   NO_USERCALLBACKS = OCI_NO_UCB,
   THREADED_MUTEXED = OCI_THREADED,
   THREADED_UNMUTEXED = OCI_THREADED | OCI_ENV_NO_MUTEX
};
```

Environment メソッドの概要

表 8-9 Environment メソッド

メソッド	概要
「createConnection()」（8-64 ページ）	データベースへの接続を作成します。
「createConnectionPool()」（8-65 ページ）	接続プールを作成します。
「createEnvironment()」（8-66 ページ）	環境を作成し、指定したメモリー管理関数を使用します。
「getCacheMaxSize()」（8-67 ページ）	キャッシュの最大ヒープ・サイズを取り出します。
「getCacheOptSize()」（8-67 ページ）	キャッシュの最適ヒープ・サイズを取り出します。
「getCurrentHeapSize ()」（8-67 ページ）	現行の環境にあるすべてのオブジェクトに現在割り当てられているメモリー・サイズを戻します。
「getMap()」（8-67 ページ）	現行の環境のマップを戻します。
「getOCIEnvironment()」（8-68 ページ）	現行の環境に関連付けられている OCI 環境を戻します。
「setCacheMaxSize()」（8-68 ページ）	クライアント側オブジェクト・キャッシュの最大サイズを、最適サイズの率として設定します。

表 8-9 Environment メソッド（続き）

メソッド	概要
「setCacheOptSize()」（8-68 ページ）	クライアント側オブジェクト・キャッシュの最適サイズをバイト単位で設定します。
「terminateConnection ()」（8-69 ページ）	接続をクローズし、関連するすべてのリソースを解放します。
「terminateConnectionPool()」（8-69 ページ）	接続プールをクローズし、関連しているすべてのリソースを解放します。
「terminateEnvironment()」（8-69 ページ）	環境を破棄します。

createConnection()

このメソッドは、指定したデータベースへの接続を確立します。

構文

```
Connection * createConnection(const string &username,
   const string &password,
   const string &connectString);
```

パラメータ

username
接続するユーザー名を指定します。

password
ユーザーのパスワードを指定します。

connectString
接続先データベースを指定します。

createConnectionPool()

このメソッドは、指定したパラメータに基づいて接続プールを作成します。

構文

```
ConnectionPool* createConnectionPool(const string &poolUserName,
   const string &poolPassword,
   const string &connectString = "",
   unsigned int minConn = 0,
   unsigned int maxConn = 1,
   unsigned int incrConn = 1);
```

パラメータ

poolUserName
プールのユーザー名を指定します。

poolPassword
プールのパスワードを指定します。

connectString
接続先データベースを指定します。

minConn
プール内の最小接続数を指定します。このメソッドによって、この最小数の接続がオープンします。追加の接続は、必要な場合のみオープンします。通常、minConn は、アプリケーションで予定している同時実行文の数に設定する必要があります。

maxConn
プール内の最大接続数を指定します。

有効な値は、1 以上です。

incrConn
現行の接続数が connMax より少ない場合に、オープンする接続の増分数を指定します。

有効な値は、1 以上です。

createEnvironment()

このメソッドは、Environment オブジェクトを作成します。このオブジェクトは、setMemMgrFunctions() メソッドに指定されているメモリー管理関数を使用して作成されます。メモリー管理関数が指定されていない場合、OCCI はデフォルトの関数を使用します。最終的には、Environment オブジェクトをクローズし、取得したシステム・リソースをすべて解放する必要があります。

指定したモードが THREADED_MUTEXED または THREADED_UN_MUTEXED の場合、3 種類のメモリー管理関数すべてはスレッド・セーフであることが必要です。

構文

```
static Environment * createEnvironment(Mode mode = DEFAULT,
   void *ctxp = 0,
   void *(*malocfp)(void *ctxp, size_t size) = 0,
   void *(*ralocfp)(void *ctxp, void *memptr, size_t newsize) = 0,
   void (*mfreefp)(void *ctxp, void *memptr) = 0);
```

パラメータ

mode

次の値が有効です。

DEFAULT ― スレッド・セーフではなく、オブジェクト・モードではありません。

THREADED_MUTEXED ― スレッド・セーフで、OCCI によって内部的に mutex 化されます。

THREADED_UN-MUTEXTED ― スレッド・セーフで、クライアント側で mutex 化する必要があります。

OBJECT ― オブジェクト機能を使用します。

ctxp

ユーザー定義のメモリー管理関数のためのコンテキスト・ポインタを指定します。

size_t

ユーザー定義のメモリー割当て関数によって割り当てられるメモリーのサイズを指定します。

malocfp

ユーザー定義のメモリー割当て関数を指定します。

ralocfp
ユーザー定義のメモリー再割当て関数を指定します。

mfreefp
ユーザー定義のメモリー解放関数を指定します。

getCacheMaxSize()

このメソッドは、キャッシュの最大ヒープ・サイズを取り出します。

構文

```
unsigned int getCacheMaxSize() const;
```

getCacheOptSize()

このメソッドは、キャッシュの最適ヒープ・サイズを取り出します。

構文

```
unsigned int getCacheOptSize() const;
```

getCurrentHeapSize ()

このメソッドは、この環境のすべてのオブジェクトに現在割り当てられているメモリー・サイズを戻します。

構文

```
unsigned int getCurrentHeapSize() const;
```

getMap()

この環境のマップへのポインタを戻します。

構文

```
Map *getMap() const;
```

getOCIEnvironment()

このメソッドは、この環境に関連付けられている OCI 環境へのポインタを戻します。

構文

```
LNOCIEnv *getOCIEnvironment() const;
```

setCacheMaxSize()

このメソッドは、クライアント側オブジェクト・キャッシュの最大サイズを、最適サイズの率として設定します。デフォルト値は 10% です。

構文

```
void setCacheMaxSize(unsigned int maxsize);
```

パラメータ

maxsize
最大サイズ値を率として指定します。

setCacheOptSize()

このメソッドは、クライアント側オブジェクト・キャッシュの最適サイズをバイト単位で設定します。デフォルト値は 8MB です。

構文

```
void setCacheOptSize(unsigned int optsize);
```

パラメータ

optsize
最適サイズ値をバイト単位で指定します。

terminateConnection ()

このメソッドは、環境への接続をクローズし、関連しているすべてのシステム・リソースを解放します。

構文

```
void terminateConnection(Connection *connection);
```

パラメータ

connection
終了する接続インスタンスへのポインタを指定します。

terminateConnectionPool()

このメソッドは、接続プールの接続をクローズし、関連しているすべてのシステム・リソースを解放します。

構文

```
void terminateConnectionPool(ConnectionPool *poolp);
```

パラメータ

poolp
終了する接続プール・インスタンスへのポインタを指定します。

terminateEnvironment()

このメソッドは、環境をクローズし、関連しているすべてのシステム・リソースを解放します。

構文

```
void terminateEnvironment(Environment *env);
```

パラメータ

env
クローズする環境を指定します。

IntervalDS クラス

先行フィールド精度は、日を入力する際の桁数によって決まります。小数秒精度は、入力時の小数桁数によって決まります。

```
IntervalDS(const Environment *env,
   int day = 0,
   int hour = 0,
   int minute = 0,
   int second = 0,
   int fs = 0);
```

day

日のコンポーネントを指定します。

有効な値は、-10 の 9 乗～ 10 の 9 乗です。

hour

時のコンポーネントを指定します。

有効な値は、-23 ～ 23 です。

minute

分のコンポーネントを指定します。

有効な値は、-59 ～ 59 です。

second

秒のコンポーネントを指定します。

有効な値は、-59 ～ 59 です。

fs

小数秒のコンポーネントを指定します。

次のコンストラクタでは、NULL の `IntervalDS` オブジェクトが作成されます。NULL の `intervalDS` は、`fromText` メソッドを割り当てるか、コールすることによって初期化できます。NULL の `intervalDS` オブジェクトでコールできるメソッドは、`setNull` と `isNull` です。

```
IntervalDS();
```

次のコンストラクタでは、IntervalDS オブジェクトが Interval の参照のコピーとして作成されます。

```
IntervalDS(const Interval &src);
```

次のコード例では、デフォルト・コンストラクタによる NULL 値の作成、日時時間隔に NULL 以外の値を割り当て、それに基づいて操作を実行する方法が示されています。

```
Environment *env = Environment::createEnvironment();

//create a null day-second interval
IntervalDS ds;
if(ds.isnull())
   cout << "\n ds is null";

//assign a non null value to ds
IntervalDS anotherDS(env, "10 20:14:10.2");
ds = anotherDS;

//now all operations are valid on DS...
int DAY = ds.getDay();
```

次のコード例では、NULL の日時時間隔を作成して fromText メソッドで初期化し、次に、+= 演算子で日時時間隔に追加し、* 演算子を使用して乗算した後で、2 つの日時時間隔を比較し、一方の日時時間隔を toText メソッドで文字列に変換する方法が示されています。

```
Environment *env = Environment::createEnvironment();

//create a null day-second interval
IntervalDS ds1

//initialize a null day-second interval by using the fromText method
ds1.fromText("20 10:20:30.9","",env);

IntervalDS addWith(env,2,1);
ds1 += addWith;    //call += operator

IntervalDS mulDs1=ds1 * Number(env,10);    //call * operator
if(ds1==mulDs1)    //call == operator
   .
   .
   .
string strds=ds1.toText(2,4);    //2 is leading field precision
                                 //4 is the fractional field precision
```

IntervalDS メソッドの概要

表 8-10 IntervalDS メソッド

メソッド	概要
「fromText()」（8-73 ページ）	instring で表現されている値を使用して IntervalDS を戻します。
「getDay()」（8-73 ページ）	日の時間隔値を戻します。
「getFracSec()」（8-74 ページ）	小数秒の時間隔値を戻します。
「getHour()」（8-74 ページ）	時の時間隔値を戻します。
「getMinute()」（8-74 ページ）	分の時間隔値を戻します。
「getSecond()」（8-74 ページ）	秒の時間隔値を戻します。
「isNull()」（8-74 ページ）	IntervalDS が NULL の場合は true を戻し、それ以外の場合は false を戻します。
「operator*()」（8-75 ページ）	2 つの IntervalDS 値の積を戻します。
「operator*=()」（8-75 ページ）	乗算による割当てを行います。
「operator=()」（8-75 ページ）	単純割当てを行います。
「operator==()」（8-76 ページ）	a と b が等しいかどうかをチェックします。
「operator!=()」（8-76 ページ）	a と b が等しくないかどうかをチェックします。
「operator/()」（8-76 ページ）	IntervalDS を（a / b）の値で戻します。
「operator/=()」（8-77 ページ）	除算による割当てを行います。
「operator>()」（8-77 ページ）	a が b を超えているかどうかをチェックします。
「operator>=()」（8-77 ページ）	a が b 以上かどうかをチェックします。
「operator<()」（8-79 ページ）	a が b 未満かどうかをチェックします。
「operator<=()」（8-78 ページ）	a が b 以下かどうかをチェックします。
「operator-()」（8-78 ページ）	IntervalDS を（a - b）の値で戻します。
「operator-=()」（8-79 ページ）	減算による割当てを行います。
「operator+()」（8-79 ページ）	2 つの IntervalDS 値の合計を戻します。
「operator+=()」（8-79 ページ）	加算による割当てを行います。
「set()」（8-80 ページ）	日時時間隔を設定します。

表 8-10 IntervalDS メソッド（続き）

メソッド	概要
「setNull()」（8-80 ページ）	日時時間隔を NULL に設定します。
「toText()」（8-81 ページ）	時間隔を表す文字列を戻します。

fromText()

このメソッドは、指定した文字列から時間隔を作成します。文字列は、関連する環境に関連付けられている nls パラメータを使用して変換されます。

nls パラメータは *env* から選択されます。*env* が NULL の場合は、nls パラメータがインスタンスに関連付けられている環境から選択されます（関連付けられている場合）。

構文

```
void fromText(const string &inpstr,
   const string &nlsParam = "",
   const Environment *env = NULL);
```

パラメータ

inpstr

日時時間隔を示す、'10 20:14:10.2' などの ' 日 時 : 分 : 秒 ' 形式の文字列を指定します。

nlsParam

このパラメータは、現在使用されていません。

env

nls パラメータが使用される環境を指定します。

getDay()

このメソッドは、時間隔の日要素を戻します。

構文

```
int getDay() const;
```

getFracSec()

このメソッドは、時間隔の小数秒要素を戻します。

構文

```
int getFracSec() const;
```

getHour()

このメソッドは、時間隔の時要素を戻します。

構文

```
int getHour() const;
```

getMinute()

このメソッドは、時間隔の分要素を戻します。

構文

```
int getMinute() const;
```

getSecond()

このメソッドは、時間隔の秒要素を戻します。

構文

```
int getSecond() const;
```

isNull()

このメソッドは、時間隔が NULL かどうかをテストします。時間隔が NULL の場合は `true` を、それ以外の場合は `false` を戻します。

構文

```
bool isNull() const;
```

operator*()

このメソッドは、時間隔に係数を乗算し、その結果を戻します。

構文

```
const IntervalDS operator*(const IntervalDS &a,
   const Number &factor);
```

パラメータ

a
乗算する時間隔を指定します。

factor
時間隔に乗算する係数を指定します。

operator*=()

このメソッドは、IntervalDS と a の積を IntervalDS に割り当てます。

構文

```
IntervalDS& operator*=(const IntervalDS &a);
```

パラメータ

a
日時時間隔を指定します。

operator=()

このメソッドは、指定された値を時間隔に割り当てます。

構文

```
IntervalDS& operator=(const IntervalDS &src);
```

パラメータ

src
割り当てる値を指定します。

operator==()

このメソッドは、指定した 2 つの時間隔を比較します。2 つの時間隔が等しい場合は true を、それ以外の場合は false を戻します。いずれかの時間隔が NULL の場合は SQLException が発生します。

構文

```
bool operator==(const IntervalDS &a,
   const IntervalDS &b);
```

パラメータ

a、b
比較する時間隔を指定します。

operator!=()

このメソッドは、指定した 2 つの時間隔を比較します。2 つの時間隔が等しくない場合は true を、それ以外の場合は false を戻します。いずれかの時間隔が NULL の場合は SQLException が発生します。

構文

```
bool operator!=(const IntervalDS &a,
   const IntervalDS &b);
```

パラメータ

a、b
比較する時間隔を指定します。

operator/()

このメソッドは、時間隔を一定の係数で除算した結果を戻します。

構文

```
const IntervalDS operator/(const IntervalDS &a,
   const Number &factor);
```

パラメータ

a
除算する時間隔を指定します。

factor
時間隔を除算する係数を指定します。

operator/=()

このメソッドは、IntervalDS と a の商を IntervalDS に割り当てます。

構文

```
IntervalDS& operator/=(const IntervalDS &a);
```

パラメータ

a
日時時間隔を指定します。

operator>()

このメソッドは、指定した 2 つの時間隔を比較します。第 1 時間隔が第 2 時間隔を超えている場合は true を、それ以外の場合は false を戻します。いずれかの時間隔が NULL の場合は SQLException が発生します。

構文

```
bool operator>(const IntervalDS &a,
   const IntervalDS &b);
```

パラメータ

a、b
比較する時間隔を指定します。

operator>=()

このメソッドは、指定した 2 つの時間隔を比較します。第 1 時間隔が第 2 時間隔以上の場合は true を、それ以外の場合は false を戻します。いずれかの時間隔が NULL の場合は SQLException が発生します。

構文

```
bool operator>=(const IntervalDS &a,
   const IntervalDS &b);
```

パラメータ

a、b
比較する時間隔を指定します。

operator<()

このメソッドは、指定した 2 つの時間隔を比較します。第 1 時間隔が第 2 時間隔未満の場合は true を、それ以外の場合は false を戻します。いずれかの時間隔が NULL の場合は SQLException が発生します。

構文

```
bool operator<(const IntervalDS &a,
   const IntervalDS &b);
```

パラメータ

a、b
比較する時間隔を指定します。

operator<=()

このメソッドは、指定した 2 つの時間隔を比較します。第 1 時間隔が第 2 時間隔以下の場合は true を、それ以外の場合は false を戻します。いずれかの時間隔が NULL の場合は SQLException が発生します。

構文

```
bool operator<=(const IntervalDS &a,
   const IntervalDS &b);
```

パラメータ

a、b
比較する時間隔を指定します。

operator-()

このメソッドは、a と b の 2 つの時間隔の差を戻します。

構文

```
const IntervalDS operator-(const IntervalDS &a,
   const IntervalDS &b);
```

パラメータ

a、b
比較する時間隔を指定します。

operator-=()

このメソッドは、IntervalDS と a の差を IntervalDS に割り当てます。

構文

```
IntervalDS& operator-=(const IntervalDS &a);
```

パラメータ

a
日時時間隔を指定します。

operator+()

このメソッドは、指定された時間隔の和を戻します。

構文

```
const IntervalDS operator+(const IntervalDS &a,
   const IntervalDS &b);
```

パラメータ

a、b
比較する時間隔を指定します。

operator+=()

このメソッドは、IntervalDS と a の和を IntervalDS に割り当てます。

構文

```
IntervalDS& operator+=(const IntervalDS &a);
```

パラメータ

a
日時時間隔を指定します。

set()

このメソッドは、間隔を指定した値に設定します。

構文

```
void set(int day,
   int hour,
   int minute,
   int second,
   int fracsec);
```

パラメータ

day
日のコンポーネントを指定します。

hour
時のコンポーネントを指定します。

minute
分のコンポーネントを指定します。

second
秒のコンポーネントを指定します。

fracsec
小数秒のコンポーネントを指定します。

setNull()

このメソッドは、時間隔を NULL に設定します。

構文

```
void setNull();
```

toText()

このメソッドは、時間隔を表す文字列を戻します。

構文

```
string toText(unsigned int lfprec,
   unsigned int fsprec,
   const string &nlsParam = "") const;
```

パラメータ

lfprec
先行フィールド精度を指定します。

fsprec
小数秒の精度を指定します。

nlsParam
このパラメータは使用されていません。

IntervalYM クラス

IntervalYM は、SQL92 の年月時間隔データ型をサポートしています。

先行フィールド精度は、入力時の桁数によって決まります。

```
IntervalYM(const Environment *env,
   int year = 0,
   int month=0);
```

year
有効な値は、-10 の 9 乗～ 10 の 9 乗です。

month
有効な値は、-11 ～ 11 です。

次のコンストラクタでは、NULL の IntervalYM オブジェクトが作成されます。NULL の intervalYM は、fromText メソッドを割り当てるか、コールすることによって初期化できます。NULL の intervalYM オブジェクトでコールできるメソッドは、setNull と isNull です。

```
IntervalYM();
```

次のコンストラクタでは、*src* から IntervalYM オブジェクトが作成されます。

```
IntervalYM(const IntervalYM &src);
```

次のコード例では、デフォルト・コンストラクタによる NULL 値の作成、年月時間隔に NULL 以外の値を割り当て、それに基づいて操作を実行する方法が示されています。

```
Environment *env = Environment::createEnvironment();

//create a null year-month interval
IntervalYM ym
if(ym.isnull())
   cout << "\n ym is null";

//assign a non null value to ym
IntervalYM anotherYM(env, "10-30");
ym = anotherYM;

//now all operations are valid on ym...
int yr = ym.getYear();
```

次のコード例では、結果セットから年月時間隔の列を取得し、+= 演算子を使用して年月時間隔に追加し、次に、* 演算子を使用して乗算して、2 つの年月時間隔を比較し、その後、一方の年月時間隔を toText メソッドで文字列に変換する方法が示されています。

```
//SELECT WARRANT_PERIOD from PRODUCT_INFORMATION
//obtain result set
resultset->next();

//get interval value from resultset
IntervalYM ym1 = resultset->getIntervalYM(1);

IntervalYM addWith(env, 10, 1);
ym1 += addWith;    //call += operator

IntervalYM mulYm1 = ym1 * Number(env, 10);    //call * operator

if(ym1<mulYm1)    //comparison
   .
   .
   .;
string strym = ym1.toText(3);    //3 is the leading field precision
```

IntervalYM メソッドの概要

表 8-11 IntervalYM メソッド

メソッド	概要
「fromText()」（8-84 ページ）	instring で表現されている値を使用して IntervalYM を戻します。
「getMonth()」（8-85 ページ）	月の時間隔値を戻します。
「getYear()」（8-85 ページ）	年の時間隔値を戻します。
「isNull()」（8-85 ページ）	時間隔が NULL かどうかをチェックします。
「operator*()」（8-85 ページ）	2 つの IntervalYM 値の積を戻します。
「operator*=()」（8-86 ページ）	乗算による割当てを行います。
「operator=()」（8-86 ページ）	単純割当てを行います。
「operator==()」（8-86 ページ）	a と b が等しいかどうかをチェックします。
「operator!=()」（8-87 ページ）	a と b が等しくないかどうかをチェックします。
「operator/()」（8-87 ページ）	時間隔を（a / b）の値で戻します。
「operator/=()」（8-87 ページ）	除算による割当てを行います。

表 8-11 IntervalYM メソッド（続き）

メソッド	概要
「operator>()」（8-88 ページ）	a が b を超えているかどうかをチェックします。
「operator>=()」（8-88 ページ）	a が b 以上かどうかをチェックします。
「operator<()」（8-89 ページ）	a が b 未満かどうかをチェックします。
「operator<=()」（8-89 ページ）	a が b 以下かどうかをチェックします。
「operator-()」（8-89 ページ）	時間隔を（a - b）の値で戻します。
「operator-=()」（8-90 ページ）	減算による割当てを行います。
「operator+()」（8-90 ページ）	2 つの IntervalYM 値の合計を戻します。
「operator+=()」（8-90 ページ）	加算による割当てを行います。
「set()」（8-91 ページ）	時間隔を指定した値に設定します。
「setNull()」（8-91 ページ）	時間隔を NULL に設定します。
「toText()」（8-91 ページ）	時間隔を表す文字列を戻します。

fromText()

このメソッドは、時間隔を inpstr の値に初期化します。文字列は、環境に設定されている nls パラメータを使用して解釈されます。

nls パラメータは *env* から選択されます。*env* が NULL の場合は、nls パラメータがインスタンスに関連付けられている環境から選択されます（関連付けられている場合）。

構文

```
void fromText(const string &inpstr,
   const string &nlsParam = "",
   const Environment *env = NULL);
```

パラメータ

inpstr

' 年－月 ' 形式の年月時間隔を示す入力文字列を指定します。

nlsParam

このパラメータは、現在使用されていません。

env

nls パラメータが使用される環境を指定します。

getMonth()

このメソッドは、時間隔の月要素を戻します。

構文

```
int getMonth() const;
```

getYear()

このメソッドは、時間隔の年要素を戻します。

構文

```
int getYear() const;
```

isNull()

このメソッドは、時間隔が NULL かどうかをテストします。時間隔が NULL の場合は `true` を、それ以外の場合は `false` を戻します。

構文

```
bool isNull() const;
```

operator*()

このメソッドは、時間隔に係数を乗算し、その結果を戻します。

構文

```
const IntervalYM operator*(const IntervalDS &a
   const Number &factor);
```

パラメータ

a
乗算する時間隔を指定します。

factor
時間隔に乗算する係数を指定します。

operator*=()

このメソッドは、時間隔に係数を乗算します。

構文

```
IntervalYM& operator*=(const Number &factor);
```

パラメータ

factor
時間隔に乗算する係数を指定します。

operator=()

このメソッドは、指定された値を時間隔に割り当てます。

構文

```
const IntervalYM& operator=(const IntervalYM &src);
```

パラメータ

src
年月時間隔を指定します。

operator==()

このメソッドは、指定した 2 つの時間隔を比較します。2 つの時間隔が等しい場合は `true` を、それ以外の場合は `false` を戻します。いずれかの時間隔が NULL の場合は SQLException が発生します。

構文

```
bool operator==(const IntervalYM &a,
   const IntervalYM &b);
```

パラメータ

a、b
比較する時間隔を指定します。

operator!=()

このメソッドは、指定した 2 つの時間隔を比較します。2 つの時間隔が等しくない場合は true を、それ以外の場合は false を戻します。いずれかの時間隔が NULL の場合は SQLException が発生します。

構文

```
bool operator!=(const IntervalYM &a,
   const IntervalYM &b);
```

パラメータ

a、b
比較する時間隔を指定します。

operator/()

このメソッドは、時間隔を係数で除算した結果を戻します。

構文

```
const IntervalYM operator/(const IntervalYM &a,
   const Number &factor);
```

パラメータ

a
除算する時間隔を指定します。

factor
時間隔を除算する係数を指定します。

operator/=()

このメソッドは、時間隔を係数で除算します。

構文

```
IntervalYM& operator/=(const Number &a);
```

パラメータ

a
時間隔を除算する係数を指定します。

operator>()

このメソッドは、指定した 2 つの時間隔を比較します。第 1 時間隔が第 2 時間隔を超えている場合は `true` を、それ以外の場合は `false` を戻します。いずれかの時間隔が NULL の場合は SQLException が発生します。

構文

```
bool operator>(const IntervalYM &a,
   const IntervalYM &b);
```

パラメータ

a、b
比較する時間隔を指定します。

operator>=()

このメソッドは、指定した 2 つの時間隔を比較します。第 1 時間隔が第 2 時間隔以上の場合は `true` を、それ以外の場合は `false` を戻します。いずれかの時間隔が NULL の場合は SQLException が発生します。

構文

```
bool operator>=(const IntervalYM &a,
   const IntervalYM &b);
```

パラメータ

a、b
比較する時間隔を指定します。

operator<()

このメソッドは、指定した 2 つの時間隔を比較します。第 1 時間隔が第 2 時間隔未満の場合は true を、それ以外の場合は false を戻します。いずれかの時間隔が NULL の場合は SQLException が発生します。

構文

```
bool operator<(const IntervalYM &a,
   const IntervalYM &b);
```

パラメータ

a、b
比較する時間隔を指定します。

operator<=()

このメソッドは、指定した 2 つの時間隔を比較します。第 1 時間隔が第 2 時間隔以下の場合は true を、それ以外の場合は false を戻します。いずれかの時間隔が NULL の場合は SQLException が発生します。

構文

```
bool operator<=(const IntervalYM &a,
   const IntervalYM &b);
```

パラメータ

a、b
比較する時間隔を指定します。

operator-()

このメソッドは、指定された時間隔の差を戻します。

構文

```
const IntervalYM operator-(const IntervalYM &a,
   const IntervalYM &b);
```

パラメータ

a、b
比較する時間隔を指定します。

operator-=()

このメソッドは、この時間隔と別の時間隔との差を計算します。

構文

```
IntervalYM& operator-=(const IntervalYM &a);
```

パラメータ

a
年月時間隔を指定します。

operator+()

このメソッドは、指定された時間隔の和を戻します。

構文

```
const IntervalYM operator+(const IntervalYM &a,
   const IntervalYM &b);
```

パラメータ

a、b
比較する時間隔を指定します。

operator+=()

このメソッドは、IntervalYM と a の和を IntervalYM に割り当てます。

構文

```
IntervalYM& operator+=(const IntervalYM &a);
```

set()

このメソッドは、間隔を指定した値に設定します。

構文

```
void set(int year,
   int month);
```

パラメータ

year
年のコンポーネントを指定します。

有効な値は、-10 の 9 乗～ 10 の 9 乗です。

month
月のコンポーネントを指定します。

有効な値は、-11 ～ 11 です。

setNull()

このメソッドは、時間隔を NULL に設定します。

構文

```
void setNull();
```

toText()

このメソッドは、時間隔を表す文字列を戻します。

構文

```
string toText(unsigned int lfprec,
   const string &nlsParam = "") const;
```

パラメータ

lfprec
先行フィールド精度を指定します。

nlsParam
このパラメータは、現在使用されていません。

Map クラス

Map クラスは、SQL 構造型のマッピングを C++ クラスに格納するために使用します。

Object Type Translator（OTT）は、各ユーザー定義型ごとに、C++ クラス宣言を生成し、静的な readSQL() メソッドと writeSQL() メソッドを実装します。readSQL() メソッドは、サーバーからのオブジェクトが C++ クラス・インスタンスとしてアプリケーション内に構築されるとコールされます。writeSQL() メソッドは、アプリケーション・キャッシュ内のオブジェクトがサーバーに書き込まれるか、フラッシュされるときに、そのオブジェクトをサーバーのデータに格納するためにコールされます。OTT によって生成された readSQL() メソッドと writeSQL() メソッドは、OCCI 標準 C++ マッピングに基づいています。

OTT で生成された標準マッピングをカスタマイズしたマッピングでオーバーライドする場合、カスタマイズが必要な各 SQL 構造型ごとに、readSQL() メソッドと writeSQL() メソッドに加えて、カスタム C++ クラスを実装する必要があります。また、そのような各クラスのエントリを Environment の Map メンバーに追加する必要があります。

Map メソッドの概要

表 8-12 Map メソッド

メソッド	概要
「put()」（8-92 ページ）	カスタマイズする型のマップ・エントリを追加します。

put()

このメソッドは、カスタマイズする型のマップ・エントリを追加します。

カスタマイズする型のマップ・エントリである type_name を追加します。type_name の C++ クラスに readSQL() と writeSQL() の静的メソッドを含めて、実装する必要があります。

次に、この情報をマップ・オブジェクトに追加する必要があります。ユーザーが標準マッピングをオーバーライドする場合は、このオブジェクトを接続に登録する必要があります。この登録は、環境が作成された後に、このメソッドをコールすることで実行できます。

構文

```
void put(const string&, void *(*)(void *),
   void (*)(void *, void *));
```

MetaData クラス

MetaData オブジェクトは、ResultSet の列またはデータベース内の既存のスキーマ・オブジェクトの型やプロパティを記述するために使用できます。データベース全体の情報も提供します。

オブジェクトのパラメータ・タイプは、次のとおりです。

- PTYPE_TABLE
- PTYPE_VIEW
- PTYPE_PROC
- PTYPE_FUNC
- PTYPE_PKG
- PTYPE_TYPE
- PTYPE_TYPE_ATTR
- PTYPE_TYPE_COLL
- PTYPE_TYPE_METHOD
- PTYPE_SYN
- PTYPE_SEQ
- PTYPE_COL
- PTYPE_ARG
- PTYPE_TYPE_ARG
- PTYPE_TYPE_RESULT
- PTYPE_SCHEMA
- PTYPE_DATABASE
- PTYPE_UNK

属性値は、次のとおりです。

- DURATION_SESSION
- DURATION_TRANS
- DURATION_NULL
- TYPEENCAP_PRIVATE

- TYPEENCAP_PUBLIC
- TYPEPARAM_IN
- TYPEPARAM_OUT
- TYPEPARAM_INOUT
- CURSOR_OPEN
- CURSOR_CLOSED
- CL_START
- CL_END
- SP_SUPPORTED
- SP_UNSUPPORTED
- NW_SUPPORTED
- NW_UNSUPPORTED
- AC_DDL
- NO_AC_DDL
- LOCK_IMMEDIATE
- LOCK_DELAYED

これらの属性値は、この属性値が結果となる一部の属性を渡して、取得メソッドを実行する際に戻されます。

コンストラクタでは、次の構文を使用します。

```
MetaData(const MetaData &omd);
```

omd

コピー元のソース・メタデータ・オブジェクトを指定します。

MetaData メソッドの概要

表 8-13 MetaData メソッド

メソッド	概要
「getAttributeCount()」（8-95 ページ）	属性の数を MetaData オブジェクトで取得します。
「getAttributeId()」（8-96 ページ）	指定された属性の ID を取得します。
「getAttributeType()」（8-96 ページ）	指定された属性の型を取得します。
「getBoolean()」（8-96 ページ）	属性の値を C++ の bool 型で取得します。
「getInt()」（8-97 ページ）	属性の値を C++ の int 型で取得します。
「getMetaData()」（8-97 ページ）	属性の値を MetaData オブジェクトで取得します。
「getNumber()」（8-97 ページ）	指定された属性を Number オブジェクトで戻します。
「getRef()」（8-98 ページ）	属性の値を Ref<T> で取得します。
「getString()」（8-98 ページ）	属性の値を文字列で取得します。
「getTimeStamp()」（8-98 ページ）	属性の値を Timestamp オブジェクトで取得します。
「getUInt()」（8-99 ページ）	属性の値を C++ の unsigned int 型で取得します。
「getVector()」（8-99 ページ）	属性の値を C++ の vector で取得します。
「operator=()」（8-99 ページ）	あるメタデータ・オブジェクトを別のメタデータ・オブジェクトに割り当てます。

getAttributeCount()

このメソッドは、メタデータ・オブジェクトに関連する属性の数を戻します。

構文

```
unsigned int getAttributeCount() const;
```

getAttributeId()

このメソッドは、指定された属性番号で示された属性の属性 ID（ATTR_NUM_COLS など）を戻します。

構文

```
AttrId getAttributeId(unsigned int attributenum) const;
```

パラメータ

attributenum
戻される属性 ID に対応する属性の番号を指定します。

getAttributeType()

このメソッドは、指定された属性番号で示された属性の型（NUMBER、INT など）を戻します。

構文

```
Type getAttributeType(unsigned int attributenum) const;
```

パラメータ

attributenum
戻される属性の型に対応する属性の番号を指定します。

getBoolean()

このメソッドは、属性の値を C++ の bool 型で戻します。値が SQL の NULL の場合、結果は false です。

構文

```
bool getBoolean(MetaData::AttrId attrid) const;
```

パラメータ

attrid
属性 ID を指定します。

getInt()

このメソッドは、属性の値を C++ の int 型で戻します。値が SQL の NULL の場合、結果は 0（ゼロ）です。

構文

```
int getInt(MetaData::AttrId attrid) const;
```

パラメータ

attrid
属性 ID を指定します。

getMetaData()

このメソッドは、属性の値を保持する MetaData インスタンスを戻します。メタデータ属性の値は、MetaData インスタンスとして取り出すことができます。このメソッドは、メタデータ型の値についてのみコールできます。

構文

```
MetaData getMetaData(MetaData::AttrId attrid) const;
```

パラメータ

attrid
属性 ID を指定します。

getNumber()

このメソッドは、属性の値を Number オブジェクトで戻します。値が SQL の NULL の場合、結果は NULL です。

構文

```
Number getNumber(MetaData::AttrId attrid) const;
```

パラメータ

attrid
属性 ID を指定します。

getRef()

このメソッドは、属性の値を RefAny で戻します。値が SQL の NULL の場合、結果は NULL です。

構文

```
RefAny getRef(MetaData::AttrId attrid) const;
```

パラメータ

attrid
属性 ID を指定します。

getString()

このメソッドは、属性の値を文字列で戻します。値が SQL の NULL の場合、結果は NULL です。

構文

```
string getString(MetaData::AttrId attrid) const;
```

パラメータ

attrid
属性 ID を指定します。

getTimeStamp()

このメソッドは、属性の値を Timestamp オブジェクトで戻します。値が SQL の NULL の場合、結果は NULL です。

構文

```
Timestamp getTimestamp(MetaData::AttrId attrid) const;
```

パラメータ

attrid
属性 ID を指定します。

getUInt()

このメソッドは、属性の値を C++ の unsigned int 型で戻します。値が SQL の NULL の場合、結果は 0（ゼロ）です。

構文

```
unsigned int getUInt(MetaData::AttrId attrid) const;
```

パラメータ

attrid
属性 ID を指定します。

getVector()

このメソッドは、属性値が含まれている C++ の vector を戻します。コレクション属性値は、C++ の vector インスタンスとして取り出すことができます。このメソッドは、リスト型の属性についてのみコールできます。

構文

```
vector<MetaData> getVector(MetaData::AttrId attrid) const;
```

パラメータ

attrid
属性 ID を指定します。

operator=()

このメソッドは、ある MetaData オブジェクトを別の MetaData オブジェクトに割り当てます。これによって、割り当てられる MetaData オブジェクトの参照カウントが 1 つ増加します。

構文

```
void operator=(const MetaData &omd);
```

パラメータ

omd
割り当てられる MetaData オブジェクトを指定します。

Number クラス

Number クラスは、精度に制限があり符号付きで 10 を基数とする数値を処理します。Number は、10 進数で 38 桁の精度が保証されます。次に示す範囲内の正の数値すべては、完全な 38 桁の精度で表現できます。

```
10^-130
```

および

```
9.99999999999999999999999999999999999999*10^125
```

表現可能な負の数値の範囲は、正の数値と対称的になります。

数値 0（ゼロ）を正確に表現できます。また、Oracle 番号には、正数と負数について無限大の表現があります。一般的に、これらの番号はオーバーフローを示すために使用されます。

内部記憶用の型は不透明でプライベートです。Number インスタンスの作成時に、位取りは保持されません。

Number は、NaN の概念をサポートしていません。また、IEEE-754-85 に準拠していません。Number は、正の無限大と負の無限大をサポートしていません。

様々な構文があります。

デフォルトのコンストラクタです。

```
Number();
```

```
Number(const Number &srcNum);
```

システム固有の long double 型を Number に変換します。この Number は、プラットフォーム固有の定数 LDBL_DIG の精度を使用して作成されます。

```
Number(long double val);
```

システム固有の double 型を Number に変換します。この Number は、プラットフォーム固有の定数 DBL_DIG の精度を使用して作成されます。

```
Number(double val);
```

システム固有の float 型を Number に変換します。この Number は、プラットフォーム固有の定数 FLT_DIG の精度を使用して作成されます。

```
Number(float val);
```

システム固有の long 型を Number に変換します。

```
Number(long val);
```

システム固有の int 型を Number に変換します。

```
Number(int val);
```

システム固有の short 型を Number に変換します。

```
Number(short val);
```

システム固有の char 型を Number に変換します。

```
Number(char val);
```

システム固有の signed char 型を Number に変換します。

```
Number(signed char val);
```

システム固有の unsigned long 型を Number に変換します。

```
Number(unsigned long val);
```

システム固有の unsigned int 型を Number に変換します。

```
Number(unsigned int val);
```

システム固有の unsigned short 型を Number に変換します。

```
Number(unsigned short val);
```

システム固有の符号なし文字配列を Number に変換します。

```
Number(unsigned char val);
```

Number クラスからのオブジェクトは、クライアント側の数値計算でスタンドアロン・クラス・オブジェクトとして使用できます。データベースからフェッチする場合やデータベースに設定する場合にも使用できます。

次のコード例では、データベースからの Number 列値の取出し、Number オブジェクトを使用するバインド、およびスタンドアロン Number オブジェクトを使用する計算が示されています。

```
/* Create a connection */
Environment *env = Environment::createEnvironment(Environment::DEFAULT);
Connection *conn = env->createConnection(user, passwd, db);

/* Create a statement and associate a select clause with it */
string sqlStmt = "SELECT department_id FROM DEPARTMENTS";
Statement *stmt = conn->createStatement(sqlStmt);
```

```
/* Execute the statement to get a result set */
ResultSet *rset = stmt->executeQuery();
while(rset->next())
{
   Number deptId = rset->getNumber(1);
   /* Display the department id with the format string 9,999 */
   cout << "Department Id" << deptId.toText(env, "9,999");

   /* Use the number obtained as a bind value in the following query */
   stmt->setSQL("SELECT * FROM EMPLOYEES WHERE department_id = :x");
   stmt->setNumber(1, deptId);
   ResultSet *rset2 = stmt->executeQuery();
   .
   .
   .
}
/* Using a Number object as a standalone and the operations on them */

/* Create a number to a double value */
double value = 2345.123;
Number nu1 (value);

/* Some common Number methods */
Number abs = nu1.abs();    /* absolute value */
Number sqrt = nu1.squareroot();    /* square root */

/* Cast operators can be used */
long lnum = (long) nu1;

/* Unary increment/decrement prefix/postfix notation */
nu1++;
--nu1;

/* Arithmetic operations */
Number nu2(nu1);

/* Assignment operators */
Number nu3;
nu3 = nu2;
nu2 = nu2 + 5.89;
Number nu4;
nu4 = nu1 + nu2;
```

```
/* Comparison operators */
if(nu1>nu2)
   .
   .
   .
else if(nu1 == nu2)
   .
   .
   .
```

Number メソッドの概要

表 8-14 Number メソッド

メソッド	概要
「abs()」（8-106 ページ）	数値の絶対値を戻します。
「arcCos()」（8-106 ページ）	数値のアーク・コサインを戻します。
「arcSin()」（8-106 ページ）	数値のアーク・サインを戻します。
「arcTan()」（8-106 ページ）	数値のアーク・タンジェントを戻します。
「arcTan2()」（8-107 ページ）	入力数値 y とこの数値 x のアーク・タンジェントを戻します。
「ceil()」（8-107 ページ）	数値の値以上で、最小の整数値を戻します。
「cos()」（8-107 ページ）	数値のコサインを戻します。
「exp()」（8-107 ページ）	数値の自然指数を戻します。
「floor()」（8-108 ページ）	数値の値以下で、最大の整数値を戻します。
「fromBytes()」（8-108 ページ）	Bytes オブジェクトから導出した Number を戻します。
「fromText()」（8-108 ページ）	指定された数値文字列、書式文字列および nls パラメータから Number を戻します。
「hypCos()」（8-109 ページ）	数値の双曲線コサインを戻します。
「hypSin()」（8-109 ページ）	数値の双曲線サインを戻します。
「hypTan()」（8-109 ページ）	数値の双曲線タンジェントを戻します。
「intPower()」（8-109 ページ）	指定された整数値で累乗した数値を戻します。
「isNull()」（8-110 ページ）	Number が NULL かどうかをチェックします。
「ln()」（8-110 ページ）	数値の自然対数を戻します。

表 8-14 Number メソッド（続き）

メソッド	概要
「log()」（8-110 ページ）	指定された基本値に対する数値の対数を戻します。
「operator++()」（8-110 ページ）	number を 1 つ分増加します。
「operator++()」（8-111 ページ）	number を 1 つ分増加します。
「operator--()」（8-111 ページ）	number を 1 つ分減少します。
「operator--()」（8-111 ページ）	number を 1 つ分減少します。
「operator*()」（8-111 ページ）	2 つの Number 値の積を戻します。
「operator/()」（8-112 ページ）	2 つの Number 値の商を戻します。
「operator%()」（8-112 ページ）	2 つの Number 値のモジュロを戻します。
「operator+()」（8-112 ページ）	2 つの Number 値の和を戻します。
「operator-()」（8-113 ページ）	Number の負の値を戻します。
「operator-()」（8-113 ページ）	2 つの Number 値の差を戻します。
「operator<()」（8-113 ページ）	数値が別の数値未満かどうかをチェックします。
「operator<=()」（8-114 ページ）	数値が別の数値以下かどうかをチェックします。
「operator>()」（8-114 ページ）	数値が別の数値を超えているかどうかをチェックします。
「operator>=()」（8-114 ページ）	数値が別の数値以上かどうかをチェックします。
「operator==()」（8-115 ページ）	2 つの数値が等しいかどうかをチェックします。
「operator!=()」（8-115 ページ）	2 つの数値が等しくないかどうかをチェックします。
「operator=()」（8-116 ページ）	ある数値を別の数値に割り当てます。
operator*=()（8-116 ページ）	乗算による割当てを行います。
「operator/=()」（8-116 ページ）	除算による割当てを行います。
「operator%=()」（8-117 ページ）	余りによる割当てを行います。
「operator+=()」（8-117 ページ）	加算による割当てを行います。
「operator-=()」（8-117 ページ）	減算による割当てを行います。
「operator char()」（8-118 ページ）	Number をシステム固有の char 型に変換して戻します。
「operator signed char()」（8-118 ページ）	Number をシステム固有の signed char 型に変換して戻します。

表 8-14 Number メソッド（続き）

メソッド	概要
「operator double()」（8-118 ページ）	Number をシステム固有の double 型に変換して戻します。
「operator float()」（8-118 ページ）	Number をシステム固有の float 型に変換して戻します。
「operator int()」（8-118 ページ）	Number をシステム固有の int 型に変換して戻します。
「operator long()」（8-119 ページ）	Number をシステム固有の long 型に変換して戻します。
「operator long double()」（8-119 ページ）	Number をシステム固有の long double 型に変換して戻します。
「operator short()」（8-119 ページ）	Number をシステム固有の short int 型に変換して戻します。
「operator unsigned char()」（8-119 ページ）	Number を符号なしでシステム固有の unsigned char 型に変換して戻します。
「operator unsigned int()」（8-119 ページ）	Number を符号なしでシステム固有の unsigned int 型に変換して戻します。
「operator unsigned long()」（8-120 ページ）	Number を符号なしでシステム固有の unsigned long 型に変換して戻します。
「operator unsigned short()」（8-120 ページ）	Number を符号なしでシステム固有の unsigned short int 型に変換して戻します。
「power()」（8-120 ページ）	指定された別の数値で累乗した Number を戻します。
「prec()」（8-120 ページ）	Number を指定の桁の精度に丸めて戻します。
「round()」（8-121 ページ）	Number を指定の小数点以下の桁に丸めて戻します。負数も可能です。
「setNull()」（8-121 ページ）	Number を NULL に設定します。
「shift()」（8-121 ページ）	指定された値 * 10^n（n は正または負）と等しい Number を戻します。
「sign()」（8-121 ページ）	指定された値の符号を戻します。指定された値 <0 の場合は -1、指定された値 ==0 の場合は 0（ゼロ）、指定された値 >0 の場合は 1 を戻します。
「sin()」（8-122 ページ）	数値のサインを戻します。
「squareroot()」（8-122 ページ）	数値の平方根を戻します。

表 8-14 Number メソッド（続き）

メソッド	概要
「tan()」（8-122 ページ）	数値のタンジェントを戻します。
「toBytes()」（8-122 ページ）	Number を示す Bytes オブジェクトを戻します。
「toText()」（8-123 ページ）	数値を、書式と nls パラメータに基づいて書式化した文字列で戻します。
「trunc()」（8-123 ページ）	小数点以下 n 桁に切り捨てた値の Number を戻します。負数も可能です。

abs()

このメソッドは、Number オブジェクトの絶対値を戻します。

構文

```
const Number abs() const;
```

arcCos()

このメソッドは、Number オブジェクトのアーク・コサインを戻します。

構文

```
const Numberconst Number arcCos() const;
```

arcSin()

このメソッドは、Number オブジェクトのアーク・サインを戻します。

構文

```
const Number arcSin() const;
```

arcTan()

このメソッドは、Number オブジェクトのアーク・タンジェントを戻します。

構文

```
const Number arcTan() const;
```

arcTan2()

このメソッドは、パラメータを指定して Number オブジェクトのアーク・タンジェントを戻します。戻される値は atan2(y,x) で、y は指定されたパラメータ、x は現行の Number オブジェクトです。

構文

```
const Number arcTan2(const Number &val) const
```

パラメータ

val

arcTangent 関数の atan2(y,x) に数値パラメータ y を指定します。

ceil()

このメソッドは、Number オブジェクト以上で、最も小さい整数を戻します。

構文

```
const Number ceil() const;
```

cos()

このメソッドは、Number オブジェクトのコサインを戻します。

構文

```
const Number cos() const;
```

exp()

このメソッドは、Number オブジェクトの自然指数を戻します。

構文

```
const Number exp() const;
```

floor()

このメソッドは、Number オブジェクト以下で、最も大きい整数を戻します。

構文

```
const Number floor() const;
```

fromBytes()

このメソッドは、指定されたバイト文字列で表現した Number オブジェクトを戻します。

構文

```
void fromBytes(const Bytes &s);
```

パラメータ

s
バイト文字列を指定します。

fromText()

このメソッドは、文字列の値から導出した Number オブジェクトを戻します。

構文

```
void fromText(const Environment *envp,
   const string &number,
   const string &fmt,
   const string &nlsParam = "");
```

パラメータ

envp
OCCI 環境を指定します。

number
Number オブジェクトに変換する数値文字列を指定します。

fmt
書式文字列を指定します。

nlsParam
nls パラメータ文字列を指定します。*nlsParam* が指定されている場合は、この指定によって、変換に使用される nls パラメータが決まります。*nlsParam* が指定されていない場合は、nls パラメータが *envp* から選択されます。

hypCos()

このメソッドは、Number オブジェクトの双曲線コサインを戻します。

構文
```
const Number hypCos() const;
```

hypSin()

このメソッドは、Number オブジェクトの双曲線サインを戻します。

構文
```
const Number hypSin() const;
```

hypTan()

このメソッドは、Number オブジェクトの双曲線タンジェントを戻します。

構文
```
const Number hypTan() const;
```

intPower()

このメソッドは、指定された値で累乗した数値オブジェクトの値を持つ Number を戻します。

構文
```
const Number intPower(int val) const;
```

パラメータ

val
数値を累乗する整数を指定します。

isNull()

このメソッドは、Number オブジェクトが NULL かどうかをテストします。Number オブジェクトが NULL の場合は true を、それ以外の場合は false を戻します。

構文

```
bool isNull() const;
```

ln()

このメソッドは、Number オブジェクトの自然対数を戻します。

構文

```
const Number ln() const;
```

log()

このメソッドは、指定されたパラメータを底として Number オブジェクトの対数を戻します。

構文

```
const Number log(const Number &val) const;
```

パラメータ

val
対数計算に使用する底を指定します。

operator++()

単項の operator++() です。このメソッドは、1 つ分増加した Number オブジェクトを戻します。これは、前置演算子です。

構文

```
Number& operator++();
```

operator++()

単項の operator++() です。このメソッドは、指定された整数分増加した Number オブジェクトを戻します。これは後置演算子です。

構文

```
const Number operator++(int);
```

operator--()

単項の operator--() です。このメソッドは、1 つ分減少した Number オブジェクトを戻します。これは、前置演算子です。

構文

```
Number& operator--();
```

operator--()

単項の operator--() です。このメソッドは、指定された整数分減少した Number オブジェクトを戻します。これは後置演算子です。

構文

```
const Number operator--(int);
```

operator*()

このメソッドは、指定されたパラメータの積を戻します。

構文

```
Number operator*(const Number &a,
    const Number &b);
```

パラメータ

a、b
乗算する数値を指定します。

operator/()

このメソッドは、指定された 2 つのパラメータの商を戻します。

構文

```
Number operator/(const Number &dividend,
   const Number &divisor);
```

パラメータ

dividend
被除数の値を指定します。

divisor
除数の値を指定します。

operator%()

このメソッドは、指定されたパラメータを除算した剰余を戻します。

構文

```
Number operator%(const Number &a,
   const Number &b);
```

パラメータ

a、b
剰余演算のオペランドである数値を指定します。

operator+()

このメソッドは、指定されたパラメータの合計を戻します。

構文

```
Number operator+(const Number &a,
   const Number &b);
```

パラメータ

a、b
加算する数値を指定します。

operator-()

単項の operator-() です。このメソッドは、Number オブジェクトの負の値を戻します。

構文

```
const Number operator-();
```

operator-()

このメソッドは、指定されたパラメータの差異を戻します。

構文

```
Number operator-(const Number &subtrahend,
   const Number &subtractor);
```

パラメータ

subtrahend
減数の値を指定します。

subtractor
減算される数値を指定します。

operator<()

このメソッドは、指定された第 1 パラメータが第 2 パラメータ未満かどうかをチェックします。第 1 パラメータが第 2 パラメータ未満の場合は true を、それ以外の場合は false を戻します。いずれかのパラメータが無限大の場合は false を戻します。

構文

```
bool operator<(const Number &a,
   const Number &b);
```

パラメータ

a、b
比較する数値を指定します。

operator<=()

このメソッドは、指定された第 1 パラメータが第 2 パラメータ以下であるかどうかをチェックします。第 1 パラメータが第 2 パラメータ以下の場合は true を、それ以外の場合は false を戻します。いずれかのパラメータが無限大の場合は false を戻します。

構文

```
bool operator<=(const Number &a,
   const Number &b);
```

パラメータ

a、b
比較する数値を指定します。

operator>()

このメソッドは、指定された第 1 パラメータが第 2 パラメータを超えているかどうかをチェックします。第 1 パラメータが第 2 パラメータを超えている場合は true を、それ以外の場合は false を戻します。いずれかのパラメータが無限大の場合は false を戻します。

構文

```
bool operator>(const Number &a,
   const Number &b);
```

パラメータ

a、b
比較する数値を指定します。

operator>=()

このメソッドは、指定された第 1 パラメータが第 2 パラメータ以上であるかどうかをチェックします。第 1 パラメータが第 2 パラメータ以上の場合は true を、それ以外の場合は false を戻します。いずれかのパラメータが無限大の場合は false を戻します。

構文

```
bool operator>=(const Number &a,
   const Number &b);
```

パラメータ

a、b
比較する数値を指定します。

operator==()

このメソッドは、指定された 2 つのパラメータが等しいかどうかをチェックします。2 つのパラメータが等しい場合は true を、それ以外の場合は false を戻します。いずれかのパラメータが正の無限大または負の無限大の場合は false を戻します。

構文

```
bool operator==(const Number &a,
   const Number &b);
```

パラメータ

a、b
比較する数値を指定します。

operator!=()

このメソッドは、指定された第 1 パラメータと第 2 パラメータが等しくないかどうかをチェックします。2 つのパラメータが等しくない場合は true を、それ以外の場合は false を戻します。

構文

```
bool operator!=(const Number &a,
   const Number &b);
```

パラメータ

a、b
比較する数値を指定します。

operator=()

このメソッドは、指定されたパラメータの値を Number オブジェクトに割り当てます。

構文

```
Number& operator=(const Number &a);
```

パラメータ

a
割り当てる数値を指定します。

operator*=()

このメソッドは、Number オブジェクトを指定されたパラメータで乗算し、その積を Number オブジェクトに割り当てます。

構文

```
Number& operator*=(const Number &a);
```

パラメータ

a
Number 型のパラメータを指定します。

operator/=()

このメソッドは、Number オブジェクトを指定されたパラメータで除算し、その商を Number オブジェクトに割り当てます。

構文

```
Number& operator/=(const Number &a);
```

パラメータ

a
Number 型のパラメータを指定します。

operator%=()

このメソッドは、Number オブジェクトを指定されたパラメータで除算し、剰余を Number オブジェクトに割り当てます。

構文

```
Number& operator%=(const Number &a);
```

パラメータ

a

Number 型のパラメータを指定します。

operator+=()

このメソッドは、Number オブジェクトと指定されたパラメータを加算し、その和を Number オブジェクトに割り当てます。

構文

```
Number& operator+=(const Number &a);
```

パラメータ

a

Number 型のパラメータを指定します。

operator-=()

このメソッドは、Number オブジェクトから指定されたパラメータを減算し、その差を Number オブジェクトに割り当てます。

構文

```
Number& operator-=(const Number &a);
```

パラメータ

a

Number 型のパラメータを指定します。

operator char()

このメソッドは、Number オブジェクトの値をシステム固有の char 型に変換して戻します。

構文

```
operator char() const;
```

operator signed char()

このメソッドは、Number オブジェクトの値をシステム固有の signed char 型に変換して戻します。

構文

```
operator signed char() const;
```

operator double()

このメソッドは、Number オブジェクトの値をシステム固有の double 型に変換して戻します。

構文

```
operator double() const;
```

operator float()

このメソッドは、Number オブジェクトの値をシステム固有の float 型に変換して戻します。

構文

```
operator float() const;
```

operator int()

このメソッドは、Number オブジェクトの値をシステム固有の int 型に変換して戻します。

構文

```
operator int()const;
```

operator long()

このメソッドは、Number オブジェクトの値をシステム固有の long 型に変換して戻します。

構文

```
operator long() const;
```

operator long double()

このメソッドは、Number オブジェクトの値をシステム固有の long double 型に変換して戻します。

構文

```
operator long double() const;
```

operator short()

このメソッドは、Number オブジェクトの値をシステム固有の short int 型に変換して戻します。

構文

```
operator short() const;
```

operator unsigned char()

このメソッドは、Number オブジェクトの値をシステム固有の unsigned char 型に変換して戻します。

構文

```
operator unsigned char() const;
```

operator unsigned int()

このメソッドは、Number オブジェクトの値をシステム固有の unsigned int 型に変換して戻します。

構文

```
operator unsigned int() const;
```

operator unsigned long()

このメソッドは、Number オブジェクトの値をシステム固有の unsigned long 型に変換して戻します。

構文

```
operator unsigned long() const;
```

operator unsigned short()

このメソッドは、Number オブジェクトの値をシステム固有の unsigned short 型に変換して戻します。

構文

```
operator unsigned short() const;
```

power()

このメソッドは、指定されたパラメータの値で累乗した Number オブジェクトの値を戻します。

構文

```
const Number power(const Number &val) const;
```

パラメータ

val
この数値で累乗する値を指定します。

prec()

このメソッドは、パラメータに指定された桁数の精度に丸めた Number オブジェクトの値を戻します。

構文

```
const Number prec(int digits) const;
```

パラメータ

digits
精度の桁数を指定します。

round()

このメソッドは、パラメータに指定された小数点以下の桁数に丸めた Number オブジェクトの値を戻します。

構文

```
const Number round(int decplace) const;
```

パラメータ

decplace
小数点の右側の桁数を指定します。

setNull()

このメソッドは、Number オブジェクトの値を NULL に設定します。

構文

```
void setNull();
```

shift()

このメソッドは、指定されたパラメータに従って 10 の累乗倍した Number オブジェクトを戻します。

構文

```
const Number shift(int val) const;
```

パラメータ

val
整数値を指定します。

sign()

このメソッドは、Number オブジェクトの値の符号を戻します。Number オブジェクトが負の場合は、-1 を戻します。Number オブジェクトが 0（ゼロ）の場合は、0（ゼロ）を戻します。Number オブジェクトが正の場合は、1 を戻します。

構文

```
const int sign() const;
```

sin()

このメソッドは、Number オブジェクトのサイン値を戻します。

構文

```
const Number sin();
```

squareroot()

このメソッドは、Number オブジェクトの平方根を戻します。

構文

```
const Number squareroot() const;
```

tan()

このメソッドは、Number オブジェクトのタンジェント値を戻します。

構文

```
const Number tan() const;
```

toBytes()

このメソッドは、Number オブジェクトを Bytes オブジェクトに変換します。バイト表現は、長さを除いた書式を前提にしています。つまり、Byte.length() メソッドでは有効なバイト長が指定され、0 番目のバイトは指数バイトです。

構文

```
Bytes toBytes() const;
```

toText()

このメソッドは、指定されたパラメータに基づいて、Number オブジェクトを書式化した文字列に変換します。

> **関連項目：** TO_CHAR の詳細は、『Oracle9*i* SQL リファレンス』を参照してください。

構文

```
string toText(const Environment *envp,
   const string &fmt,
   const string &nlsParam = "") const;
```

パラメータ

envp
OCCI 環境を指定します。

fmt
書式文字列を指定します。

nlsParam
nls パラメータ文字列を指定します。*nlsParam* が指定されている場合は、この指定によって、変換に使用される nls パラメータが決まります。*nlsParam* が指定されていない場合は、nls パラメータが *envp* から選択されます。

trunc()

このメソッドは、指定されたパラメータに従って小数点以下の桁数を切り捨てた Number オブジェクトを戻します。

構文

```
const Number trunc(int decplace) const;
```

パラメータ

decplace
小数点の右側の値の切り捨てる桁の位置を指定します。

PObject クラス

OCCI には、アプリケーションが次の操作をオブジェクトに対して実行できるように、オブジェクト・ナビゲーショナル・コールが用意されています。

- オブジェクトの作成、アクセス、ロック、削除、コピーおよびフラッシュ
- オブジェクトへの参照の取得

このクラスによって、型の定義に携わる担当者は、クラスが永続または一時インスタンスを保持できる時点を指定できます。PObject から導出されたクラスのインスタンスは、永続または一時のいずれかです。永続性可能なクラス（"A" と呼ばれる）は、PObject クラスを継承します。

```
class A : PObject { ... }
```

lock() や refresh() などの一部のメソッドは、永続インスタンスのみに適用でき、一時インスタンスには適用できません。

NULL の PObject を作成するには、次の構文を使用します。

```
PObject();
```

NULL の PObject に対して有効なメソッドは、setNull()、isNull および operator=() のみです。

PObject のコピーを作成するには、次の構文を使用します。

```
PObject(const PObject& obj);
```

PObject メソッドの概要

表 8-15 PObject メソッド

メソッド	概要
「flush()」（8-125 ページ）	修正した永続オブジェクトをデータベース・サーバーにフラッシュします。
「getConnection()」（8-125 ページ）	PObject オブジェクトがインスタンス化された元の接続を戻します。
「getRef()」（8-126 ページ）	指定された永続オブジェクトへの参照を戻します。
「isLocked()」（8-126 ページ）	永続オブジェクトがロックされているかどうかを確認します。
「isNull()」（8-126 ページ）	オブジェクトが NULL かどうかを確認します。

表 8-15 PObject メソッド（続き）

メソッド	概要
「lock()」（8-126 ページ）	データベース・サーバーの永続オブジェクトをロックします。ロックがない場合、デフォルトのモードはロックの待機です。
「markDelete()」（8-127 ページ）	永続オブジェクトに削除済みマークを設定します。
「markModified()」（8-127 ページ）	永続オブジェクトに修正済みまたは使用済みマークを設定します。
「operator=()」（8-127 ページ）	ある PObject を別の PObject に割り当てます。
「operator delete()」（8-127 ページ）	永続オブジェクトをアプリケーション・キャッシュのみから削除します。
「operator new()」（8-128 ページ）	永続 / 一時インスタンスを新規作成します。
「pin()」（8-128 ページ）	オブジェクトを確保します。
「setNull()」（8-129 ページ）	オブジェクトの値を NULL に設定します。
「unmark()」（8-129 ページ）	オブジェクトの使用済みマークを解除します。
「unpin()」（8-129 ページ）	オブジェクトの確保を解除します。デフォルトのモードでは、オブジェクトの確保カウントは 1 つ分減少します。

flush()

このメソッドは、修正した永続オブジェクトをデータベース・サーバーにフラッシュします。

構文

```
void flush();
```

getConnection()

このメソッドは、永続オブジェクトがインスタンス化された元の接続を戻します。

構文

```
const Connection *getConnection() const;
```

getRef()

このメソッドは、永続オブジェクトへの参照を戻します。

構文

```
RefAny getRef() const;
```

isLocked()

このメソッドは、永続オブジェクトがロックされているかどうかを確認します。永続オブジェクトがロックされている場合は true を、それ以外の場合は false を戻します。

構文

```
bool isLocked() const;
```

isNull()

このメソッドは、永続オブジェクトが NULL かどうかを確認します。永続オブジェクトが NULL の場合は true を、それ以外の場合は false を戻します。

構文

```
bool isNull() const;
```

lock()

このメソッドは、データベース・サーバーの永続オブジェクトをロックします。

構文

```
void lock(PObject::LockOption lock_option);
```

パラメータ

lock_option

オブジェクトが別のユーザーによってすでにロックされている場合に、ロック操作を待機するかどうかを指定します。デフォルト値は OCCI_LOCK_WAIT で、ロック操作を待機します。

次の値が有効です。

OCCI_LOCK_WAIT

OCCI_LOCK_NOWAIT

markDelete()

このメソッドは、永続オブジェクトに削除済みマークを設定します。

構文

```
void markDelete();
```

markModified()

このメソッドは、永続オブジェクトに修正済みまたは使用済みマークを設定します。

構文

```
void mark_Modified();
```

operator=()

このメソッドは、永続オブジェクトの値をこの PObject オブジェクトに割り当てます。オブジェクトの性質（一時または永続）は保持されます。NULL 情報がソース・インスタンスからコピーされます。

構文

```
PObject& operator=(const PObject& obj);
```

パラメータ

obj
コピー元のオブジェクトを指定します。

operator delete()

このメソッドは、永続または一時オブジェクトを削除するために使用します。永続オブジェクトの場合、この削除演算子は、オブジェクトをアプリケーション・キャッシュのみから削除します。データベース・サーバーからオブジェクトを削除するには、markDelete() メソッドをコールします。

構文

```
void operator delete(void *obj,
   size_t size);
```

operator new()

このメソッドは、新規オブジェクトを作成するために使用します。接続および表名が指定されている場合は、永続オブジェクトが作成されます。それ以外の場合は、一時オブジェクトが作成されます。

構文

様々な構文があります。

```
void *operator new(size_t size);

void *operator new(size_t size,
   const Connection *x,
   const string& tablename,
   const char *type_name);
```

パラメータ

size

x

永続オブジェクトが作成されるデータベースへの接続を指定します。

tablename

データベース・サーバーにある表の名前を指定します。

type_name

この C++ クラスに対応する SQL の型名を指定します。フォーマットは、<schemaname>.<typename> です。

pin()

このメソッドは、オブジェクトを確保し、確保カウントを 1 つ増やします。オブジェクトが確保されている間は、このオブジェクト・インスタンスへの参照がない場合でもキャッシュによって解放されることはありません。

構文

```
void pin();
```

setNull()

このメソッドは、オブジェクトの値を NULL に設定します。

構文

```
void setNull();
```

unmark()

このメソッドは、永続オブジェクトの修正済みまたは削除済みマークを解除します。

構文

```
void unmark();
```

unpin()

このメソッドは、永続オブジェクトの確保を解除します。デフォルトのモードでは、オブジェクトの確保カウントは 1 つ分減少します。このメソッドが OCCI_PINCOUNT_RESET でコールされると、オブジェクトの確保カウントがリセットされます。

確保カウントがリセットされると、このオブジェクトを指し示している参照（Ref）はすべて無効になります。キャッシュは、解放の対象となるオブジェクトを設定し、必要に応じてメモリーを再生します。

構文

```
void unpin(UnpinOption mode=OCCI_PINCOUNT_DECR);
```

パラメータ

mode

確保カウントを減らすか、0（ゼロ）にリセットするかを指定します。

次の値が有効です。

OCCI_PINCOUNT_RESET

OCCI_PINCOUNT_DECR

Ref クラス

SQL REF 値の C++ プログラミング言語によるマッピングです。このマッピングは、データベースにある SQL 構造型の値への参照です。

各 REF 値には、参照先オブジェクトの一意の識別子があります。SQL REF 値は、参照する SQL 構造型のかわりに使用できます。また、表の列値または構造型の属性値として使用できます。

SQL REF 値は、SQL 構造型への論理ポインタであるため、デフォルトの場合、Ref オブジェクトも論理ポインタです。したがって、Ref オブジェクトとして SQL REF 値を取得しても、クライアントの構造型の属性をマテリアライズしません。

Ref オブジェクトは永続記憶域に保存でき、operator*、operator-> または ptr() メソッドによって間接参照されます。T は、PObject から導出したクラスであることが必要です。次の項では、T* と PObject* が同じ意味で使用されています。

NULL の Ref オブジェクトを作成するには、次の構文を使用します。

```
Ref();
```

NULL の Ref オブジェクトに対して有効なメソッドは、isNull および operator=() のみです。

Ref オブジェクトのコピーを作成するには、次の構文を使用します。

```
Ref(const Ref<T> &src);
```

Ref メソッドの概要

表 8-16 Ref メソッド

メソッド	概要
「clear()」（8-131 ページ）	参照を消去します。
「getConnection()」（8-131 ページ）	この Ref が作成された元の接続を戻します。
「getRef()」（8-132 ページ）	Ref を戻します。
「isClear()」（8-132 ページ）	Ref が消去されているかどうかをチェックします。
「isNull()」（8-132 ページ）	Ref が NULL かどうかをチェックします。
「markDelete()」（8-132 ページ）	参照オブジェクトに削除済みマークを設定します。
「operator->()」（8-132 ページ）	Ref を間接参照し、必要に応じてオブジェクトを確保します。

表 8-16 Ref メソッド（続き）

メソッド	概要
「operator*()」（8-133 ページ）	この演算子は、Ref を間接参照し、必要に応じてオブジェクトを確保 / フェッチします。
「operator==()」（8-133 ページ）	Ref とポインタが同じオブジェクトを参照しているかどうかをチェックします。
「operator!=()」（8-133 ページ）	Ref とポインタが異なるオブジェクトを参照しているかどうかをチェックします。
「operator=()」（8-134 ページ）	代入演算子です。
「ptr()」（8-134 ページ）	Ref を間接参照し、必要に応じてオブジェクトを確保 / フェッチします。
「setPrefetch()」（8-135 ページ）	プリフェッチ用にたどるオブジェクト属性の型とレベルを指定します。
「setLock()」（8-135 ページ）	参照元オブジェクトにロック・オプションを設定します。
「setNull()」（8-136 ページ）	Ref を NULL に設定します。
「unmarkDelete()」（8-136 ページ）	参照先オブジェクトの削除マークを解除します。

clear()

このメソッドは、Ref オブジェクトを消去します。

構文

```
void clear();
```

getConnection()

このメソッドは、Ref オブジェクトがインスタンス化された元の接続を戻します。

構文

```
const Connection *getConnection() const;
```

getRef()

このメソッドは、Ref オブジェクトから OCI Ref を戻します。

構文

```
LNOCIRef *getRef() const;
```

isClear()

このメソッドは、Ref オブジェクトが消去されているかどうかをチェックします。

構文

```
bool isClear();
```

isNull()

このメソッドは、Ref オブジェクトが NULL かどうかをテストします。Ref オブジェクトが NULL の場合は true を、それ以外の場合は false を戻します。

構文

```
bool isNull() const;
```

markDelete()

このメソッドは、参照オブジェクトに削除済みマークを設定します。

構文

```
void markDelete();
```

operator->()

このメソッドは、Ref オブジェクトを間接参照し、必要に応じて参照オブジェクトを確保またはフェッチします。参照先オブジェクトのプリフェッチ属性が設定されている場合は、結果的にオブジェクトのグラフのプリフェッチとなる場合があります。

構文

様々な構文があります。

```
T * operator->();

const T * operator->() const;
```

operator*()

このメソッドは、Ref オブジェクトを間接参照し、必要に応じて参照オブジェクトを確保またはフェッチします。参照先オブジェクトのプリフェッチ属性が設定されている場合は、結果的にオブジェクトのグラフのプリフェッチとなる場合があります。

オブジェクトを削除する必要はありません。有効範囲外になると、デストラクタが自動的にコールされます。

構文

様々な構文があります。

```
T & operator *();

const T & operator*() const;
```

operator==()

このメソッドは、2 つの Ref オブジェクトが同じオブジェクトを参照しているかどうかを確認します。2 つの Ref オブジェクトが同じオブジェクトを参照している場合は true を、それ以外の場合は false を戻します。

構文

```
bool operator == (const Ref<T> &ref) const;
```

パラメータ

ref
比較するオブジェクトの Ref オブジェクトを指定します。

operator!=()

このメソッドは、2 つの Ref オブジェクトが同じオブジェクトを参照しているかどうかを確認します。2 つの Ref オブジェクトが同じオブジェクトを参照していない場合は true を、それ以外の場合は false を戻します。

構文

```
bool operator!= (const Ref<T> &ref) const;
```

パラメータ

ref
比較するオブジェクトの Ref オブジェクトを指定します。

operator=()

Ref またはオブジェクトを Ref に割り当てます。最初の場合は複数の Ref が割り当てられ、2 番目の場合は Ref がオブジェクトから作成され、その後に割り当てられます。

構文

様々な構文があります。

```
Ref<T>& operator=(const Ref<T> &src);

Ref<T>& operator=(const T *obj);
```

パラメータ

src
割り当てられるソースの Ref オブジェクトを指定します。

obj
Ref オブジェクトが割り当てられるソース・オブジェクトのポインタを指定します。

ptr()

Ref を間接参照し、必要に応じてオブジェクトを確保 / フェッチします。Ref のプリフェッチ属性が設定されている場合は、結果的にオブジェクトのグラフのプリフェッチとなる場合があります。

構文

様々な構文があります。

```
T * ptr();

const T * ptr() const;
```

setPrefetch()

複合オブジェクト検索のプリフェッチ・オプションを設定します。

このメソッドは、Ref を介して到達できる（推移閉包）すべてのオブジェクトをプリフェッチする場合のレベルを指定します。選択した属性の型のみをプリフェッチする場合は、setPrefetch(type_name, depth) を使用します。

このメソッドは、参照先のオブジェクトのどの Ref 属性をオブジェクトのプリフェッチ用にたどるか（複合オブジェクト検索）、およびどのレベルまでそれらのリンクをたどるかを指定します。

構文

様々な構文があります。

```
void setPrefetch(const string &typeName,
   unsigned int depth);


void setPrefetch(unsigned int depth);
```

パラメータ

typeName
プリフェッチする Ref 属性の型を指定します。

depth
リンクをたどる深さのレベルを指定します。

setLock()

このメソッドは、間接参照する場合のオブジェクトのロック方法を指定します。

構文

```
void setLock(LockOptions);
```

パラメータ

LockOptions
ロック・オプションを指定します。

次の値が有効です。

OCCI_LOCK_NONE

OCCI_LOCK_X

OCCI_LOCK_X_NOWAIT

setNull()

このメソッドは、Ref オブジェクトを NULL に設定します。

構文

```
void setNull();
```

unmarkDelete()

このメソッドは、参照オブジェクトの使用済みマークを解除します。

構文

```
void unmarkDelete();
```

RefAny クラス

RefAny クラスは、あらゆる型への参照をサポートするために設計されています。このクラスの主な目的は、一般的な参照を処理し、型の階層で Ref の変換を可能にすることです。RefAny オブジェクトは、Ref<x> と Ref<y>（x と y は異なる型）の 2 つの型の中継として使用できます。

Ref<T> は、常に RefAny に変換できます。Ref<T> テンプレートには変換を実行するためのメソッドがあります。各 Ref<T> には、コンストラクタ、および RefAny への参照を取得する割当て演算子があります。

```
RefAny();
RefAny(const Connection *sessptr, const OCIRef *Ref);
RefAny(const RefAny& src);
```

RefAny メソッドの概要

表 8-17 RefAny メソッド

メソッド	概要
「clear()」（8-138 ページ）	参照を消去します。
「getConnection()」（8-138 ページ）	この Ref が作成された元の接続を戻します。
「getRef()」（8-138 ページ）	Ref を戻します。
「isNull()」（8-138 ページ）	RefAny オブジェクトが NULL かどうかをチェックします。
「markDelete()」（8-138 ページ）	オブジェクトを削除済みにマークします。
「operator=()」（8-139 ページ）	代入演算子です。
「operator==()」（8-139 ページ）	等しいかどうかをチェックします。
「operator!=()」（8-139 ページ）	等しくないかどうかをチェックします。
「unmarkDelete()」（8-140 ページ）	オブジェクトの削除済みマークを解除します。

clear()

このメソッドは、参照を消去します。

構文

```
void clear();
```

getConnection()

この Ref がインスタンス化された元の接続を戻します。

構文

```
const Connection * getConnection() const;
```

getRef()

基礎となる OCIRef * を戻します。

構文

```
LNOCIRef* getRef() const;
```

isNull()

この Ref が指し示すオブジェクトが NULL の場合は true を、それ以外の場合には false を戻します。

構文

```
bool isNull() const;
```

markDelete()

このメソッドは、参照オブジェクトに削除済みマークを設定します。

構文

```
void markDelete();
```

operator=()

Ref またはオブジェクトを Ref に割り当てます。最初の場合は複数の Ref が割り当てられ、2 番目の場合は Ref がオブジェクトから作成され、その後に割り当てられます。

構文

```
RefAny& operator=(const RefAny& src);
```

パラメータ

src
割り当てられるソースの RefAny オブジェクトを指定します。

operator==()

この Ref と RefAny オブジェクトを比較し、両方がキャッシュ内の同じオブジェクトを参照している場合は true を、それ以外の場合は false を戻します。

構文

```
bool operator== (const RefAny &refAnyR) const;
```

パラメータ

refAnyR
比較する RefAny オブジェクトを指定します。

operator!=()

この Ref と RefAny オブジェクトを比較し、両方がキャッシュ内の同じオブジェクトを参照していない場合は true を、それ以外の場合は false を戻します。

構文

```
bool operator!= (const RefAny &refAnyR) const;
```

パラメータ

refAnyR
比較する RefAny オブジェクトを指定します。

unmarkDelete()

このメソッドは、参照オブジェクトの使用済みマークを解除します。

構文

```
void unmarkDelete();
```

ResultSet クラス

ResultSet は、Statement の実行で生成されたデータの表へのアクセスを提供します。表の行は順序に従って取り出されます。1 行の中の列値は、どのような順序でもアクセスできます。

ResultSet は、データの現在行を指し示すカーソルを保持しています。最初、カーソルは第 1 行目の前にあります。next メソッドによって、カーソルが次の行に移動します。

get ... () メソッドを使用して、現在行の列値を取り出します。列の索引番号または名前のいずれかを使用して、値を取り出すことができます。通常は、列の索引を使用するほうが効率的です。各列には、1 から順に番号が付けられています。

get ... () メソッドの場合、OCCI は、基礎となるデータを指定された C++ 型に変換して、C++ の値を戻そうとします。

SQL 型は ResultSet::get ... () メソッドを使用して C++ 型にマップされます。

ResultSet の列の番号、型およびプロパティは、getColumnListMetaData メソッドが戻す MetaData オブジェクトによって指定されます。

```
enum Status
{
    END_OF_FETCH = 0,
    DATA_AVAILABLE,
    STREAM_DATA_AVAILABLE
};
```

ResultSet()

これは、ResultSet コンストラクタです。

構文

```
ResultSet()
```

ResultSet メソッドの概要

表 8-18 ResultSet メソッド

メソッド	説明
「cancel()」（8-144 ページ）	ResultSet を取り消します。
「closeStream()」（8-144 ページ）	指定された Stream をクローズします。
「getBfile()」（8-144 ページ）	現在行の列の値を Bfile で戻します。
「getBlob()」（8-145 ページ）	現在行の列の値を Blob オブジェクトで戻します。
「getBytes()」（8-145 ページ）	現在行の列の値を Bytes 配列で戻します。
「getCharSet()」（8-145 ページ）	データをフェッチするキャラクタ・セットを戻します。
「getClob()」（8-146 ページ）	現在行の列の値を Clob オブジェクトで戻します。
「getColumnListMetaData()」（8-146 ページ）	結果セット列の記述情報を MetaData オブジェクトで戻します。
「getCurrentStreamColumn()」（8-146 ページ）	現行の読込み可能 Stream の列索引を戻します。
「getCurrentStreamRow()」（8-147 ページ）	処理している ResultSet の現在行を戻します。
「getCursor()」（8-147 ページ）	ネストしたカーソルを ResultSet で戻します。
「getDate()」（8-147 ページ）	現在行の列の値を Date オブジェクトで戻します。
「getDatebaseNCHARParam()」（8-148 ページ）	データが NCHAR キャラクタ・セットにあるかどうかを戻します。
「getDouble()」（8-148 ページ）	現在行の列の値を C++ の double 型で戻します。
「getFloat()」（8-148 ページ）	現在行の列の値を C++ の float 型で戻します。
「getInt()」（8-149 ページ）	現在行の列の値を C++ の int 型で戻します。
「getIntervalDS()」（8-149 ページ）	現在行の列の値を IntervalDS で戻します。
「getIntervalYM()」（8-149 ページ）	現在行の列の値を IntervalYM で戻します。
「getMaxColumnSize()」（8-150 ページ）	列から読み込むデータの最大量を戻します。
「getNumArrayRows()」（8-150 ページ）	next(int numRows) が END_OF_DATA を戻したときに、最後の配列のフェッチで、実際にフェッチした行数を戻します。
「getNumber()」（8-150 ページ）	現在行の列の値を Number オブジェクトで戻します。

表 8-18 ResultSet メソッド（続き）

メソッド	説明
「getObject()」（8-150 ページ）	現在行の列の値を PObject で戻します。
「getRef()」（8-151 ページ）	現在行の列の値を Ref で戻します。
「getRowid()」（8-151 ページ）	SELECT FOR UPDATE 文の現行の ROWID を戻します。
「getRowPosition()」（8-151 ページ）	現在行の位置の ROWID を戻します。
「getStatement()」（8-152 ページ）	ResultSet の Statement を戻します。
「getStream()」（8-152 ページ）	現在行の列の値を Stream で戻します。
「getString()」（8-152 ページ）	現在行の列の値を文字列で戻します。
「getTimestamp()」（8-152 ページ）	現在行の列の値を Timestamp オブジェクトで戻します。
「getUInt()」（8-153 ページ）	現在行の列の値を C++ の unsigned int 型で戻します。
「getVector()」（8-153 ページ）	指定されたコレクション・パラメータをベクターで戻します。
「getVectorOfRefs()」（8-153 ページ）	指定のパラメータ（REF のコレクションを表す）を REF のベクターで戻します。
「isNull()」（8-156 ページ）	値が NULL かどうかをチェックします。
「isTruncated()」（8-156 ページ）	切捨てが発生したかどうかをチェックします。
「next()」（8-157 ページ）	ResultSet の次の行を現在行にします。
「preTruncationLength()」（8-157 ページ）	
「setBinaryStreamMode()」（8-158 ページ）	列をバイナリ・ストリームで戻すように指定します。
「setCharacterStreamMode()」（8-158 ページ）	列をキャラクタ・ストリームで戻すように指定します。
「setCharSet()」（8-159 ページ）	データを戻すキャラクタ・セットを指定します。
「setDatebaseNCHARParam()」（8-159 ページ）	パラメータを、データベースの NCHAR キャラクタ・セットのデータが含まれている列から取り出そうとする場合は、true 値を渡すことによって、そのことを OCCI に通知する必要があります。
「setDataBuffer()」（8-160 ページ）	データを読み込むデータ・バッファを指定します。

表 8-18 ResultSet メソッド（続き）

メソッド	説明
「setErrorOnNull()」（8-161 ページ）	NULL 値を読み込む場合の例外を使用可能 / 使用禁止にします。
「setErrorOnTruncate()」（8-161 ページ）	切捨てが発生する場合の例外を使用可能 / 使用禁止にします。
「setMaxColumnSize()」（8-162 ページ）	列から読み込むデータの最大量を指定します。
「status()」（8-162 ページ）	ResultSet の現在のステータスを戻します。

cancel()

このメソッドは、結果セットを取り消します。

構文

```
void cancel();
```

closeStream()

このメソッドは、パラメータ *stream* によって指定されたストリームをクローズします。

構文

```
void closeStream(Stream *stream);
```

パラメータ

stream
クローズするストリームを指定します。

getBfile()

このメソッドは、現在行の列の値を Bfile で戻します。列の値を戻します。値が SQL の NULL の場合、結果は NULL です。

構文

```
Bfile getBfile(unsigned int colIndex);
```

パラメータ

colIndex
第 1 列は 1、第 2 列は 2 のように指定します。

getBlob()

現在行の列の値を Blob で取得します。列の値を戻します。値が SQL の NULL の場合、結果は NULL です。

構文

```
Blob getBlob(unsigned int colIndex);
```

パラメータ

colIndex
第 1 列は 1、第 2 列は 2 のように指定します。

getBytes()

現在行の列の値を Bytes 配列で取得します。バイト数はサーバーによって戻される行の値を示します。列の値を戻します。値が SQL の NULL の場合、結果は空の配列です。

構文

```
Bytes getBytes(unsigned int colIndex);
```

パラメータ

colIndex
第 1 列は 1、第 2 列は 2 のように指定します。

getCharSet()

データをフェッチするキャラクタ・セットを文字列で取得します。

構文

```
string getCharSet(unsigned int paramIndex) const;
```

パラメータ

paramIndex
第 1 パラメータは 1、第 2 パラメータは 2 のように指定します。

getClob()

現在行の列の値を Clob で取得します。列の値を戻します。値が SQL の NULL の場合、結果は NULL です。

構文

```
Clob getClob(unsigned int colIndex);
```

パラメータ

colIndex

第 1 列は 1、第 2 列は 2 のように指定します。

getColumnListMetaData()

ResultSet の列の番号、型およびプロパティは、getMetaData メソッドによって指定されます。ResultSet の列の説明を戻します。指定された列の値を PObject で戻します。C++ オブジェクトの型は、Environment のマップに登録されている列の SQL 型に対応する C++ PObject 型になります。このメソッドは、SQL ユーザー定義型のデータをマテリアライズするために使用します。

構文

```
vector<MetaData> getColumnListMetaData() const;
```

getCurrentStreamColumn()

結果セットに入力 Stream パラメータがある場合、このメソッドは読込みが必要な現行の入力 Stream の列索引を戻します。出力 Stream の読込みが不要な場合、または結果セットに入力 Stream 列がない場合、このメソッドは 0（ゼロ）を戻します。通常は、読込みが必要な現行の入力 Stream 列の列索引を戻します。

構文

```
unsigned int getCurrentStreamColumn() const;
```

getCurrentStreamRow()

結果に入力 Stream がない場合、このメソッドは OCCI で処理中の結果セットの現在行を戻します。一連の行の配列にあるすべての行が処理された後に、このメソッドがコールされた場合は、0（ゼロ）を戻します。通常は、処理中の現在行の行番号を戻します。第 1 行目は 1、第 2 行目は 2 のようになります。

構文

```
unsigned int getCurrentStreamRow() const;
```

getCursor()

ネストしたカーソルを ResultSet で取得します。この結果セットからデータをフェッチできます。ネストしたカーソルは、CURSOR(SELECT ...) 句でネストされた問合せの結果として生じます。

```
SELECT ename, CURSOR(SELECT  dname, loc FROM dept) FROM emp WHERE ename = 'JONES'
```

戻される REF CURSOR が複数の場合は、次の REF CURSOR を取得してフェッチを開始する前に、各カーソルからのデータを完全にフェッチする必要があります。ネストしたカーソルに対して ResultSet を 1 つ戻します。

構文

```
ResultSet * getCursor(unsigned int colIndex);
```

パラメータ

colIndex
第 1 列は 1、第 2 列は 2 のように指定します。

getDate()

現在行の列の値を Date オブジェクトで取得します。列の値を戻します。値が SQL の NULL の場合、結果は NULL です。

構文

```
Date getDate(unsigned int colIndex);
```

パラメータ

colIndex
第 1 列は 1、第 2 列は 2 のように指定します。

getDatebaseNCHARParam()

データが NCHAR キャラクタ・セットにあるかどうかを戻します。

構文

```
bool getDatebaseNCHARParam(unsigned int paramIndex) const;
```

パラメータ

paramIndex
パラメータ索引を指定します。

getDouble()

現在行の列の値を C++ の double 型で取得します。列の値を戻します。値が SQL の NULL の場合、結果は 0（ゼロ）です。

構文

```
double getDouble(unsigned int colIndex);
```

パラメータ

colIndex
第 1 列は 1、第 2 列は 2 のように指定します。

getFloat()

現在行の列の値を C++ の float 型で取得します。列の値を戻します。値が SQL の NULL の場合、結果は 0（ゼロ）です。

構文

```
float getFloat(unsigned int colIndex);
```

パラメータ

colIndex
第 1 列は 1、第 2 列は 2 のように指定します。

getInt()

現在行の列の値を C++ の int 型で取得します。列の値を戻します。値が SQL の NULL の場合、結果は 0（ゼロ）です。

構文

```
int getInt(unsigned int colIndex);
```

パラメータ

colIndex
第 1 列は 1、第 2 列は 2 のように指定します。

getIntervalDS()

現在行の列の値を IntervalDS オブジェクトで取得します。列の値を戻します。値が SQL の NULL の場合、結果は NULL です。

構文

```
IntervalDS getIntervalDS(unsigned int colIndex);
```

パラメータ

colIndex
第 1 列は 1、第 2 列は 2 のように指定します。

getIntervalYM()

現在行の列の値を IntervalYM オブジェクトで取得します。列の値を戻します。値が SQL の NULL の場合、結果は NULL です。

構文

```
IntervalYM getIntervalYM(unsigned int colIndex);
```

パラメータ

colIndex
第 1 列は 1、第 2 列は 2 のように指定します。

getMaxColumnSize()

列に読み込むデータの最大量を取得します。

構文

```
unsigned int getMaxColumnSize(unsigned int colIndex) const;
```

パラメータ

colIndex
第 1 列は 1、第 2 列は 2 のように指定します。

getNumArrayRows()

next(int numRows) が END_OF_DATA を戻したときに、最後の配列のフェッチで、実際にフェッチした行数を戻します。最終の配列フェッチで実際にフェッチした行数を戻します。

構文

```
unsigned int getNumArrayRows() const;
```

getNumber()

現在行の列の値を Number オブジェクトで取得します。列の値を戻します。値が SQL の NULL の場合、結果は NULL です。

構文

```
Number getNumber(unsigned int colIndex);
```

パラメータ

colIndex
第 1 列は 1、第 2 列は 2 のように指定します。

getObject()

列の値を保持している PObject へのポインタを戻します。

構文

```
PObject * getObject(unsigned int colIndex);
```

パラメータ

colIndex
第 1 列は 1、第 2 列は 2 のように指定します。

getRef()

現在行の列の値を RefAny で取得します。Ref 値の取得によって、Ref が参照するデータはインスタンス化されません。また、Ref 値は、この値が作成されたセッションまたは接続がオープンしている間、値が保持されます。列値を保持している RefAny を戻します。

構文
```
RefAny getRef(unsigned int colIndex);
```

パラメータ

colIndex
第 1 列は 1、第 2 列は 2 のように指定します。

getRowid()

SELECT ... FOR UPDATE 文の現行の ROWID を取得します。この ROWID は、準備した DELETE 文などにバインドできます。SELECT ... FOR UPDATE 文の現行の ROWID を戻します。

構文
```
Bytes getRowid(unsigned int colIndex);
```

パラメータ

colIndex
第 1 列は 1、第 2 列は 2 のように指定します。

getRowPosition()

このメソッドは、現在行の位置の ROWID を取得します。

構文
```
Bytes getRowPosition() const
```

getStatement()

ResultSet の Statement を戻します。

構文

```
const Statement* getStatement() const;
```

getStream()

現在行の列の値を Stream で戻します。

構文

```
Stream * getStream(unsigned int colIndex);
```

パラメータ

colIndex

第 1 列は 1、第 2 列は 2 のように指定します。

getString()

現在行の列の値を文字列で取得します。列の値を戻します。値が SQL の NULL の場合、結果は空の文字列です。

構文

```
string getString(unsigned int colIndex);
```

パラメータ

colIndex

第 1 列は 1、第 2 列は 2 のように指定します。

getTimestamp()

現在行の列の値を Timestamp オブジェクトで取得します。列の値を戻します。値が SQL の NULL の場合、結果は NULL です。

構文

```
Timestamp getTimestamp(unsigned int colIndex);
```

パラメータ

colIndex
第 1 列は 1、第 2 列は 2 のように指定します。

getUInt()

現在行の列の値を C++ の int 型で取得します。列の値を戻します。値が SQL の NULL の場合、結果は 0（ゼロ）です。

構文

```
unsigned int getUInt(unsigned int colIndex);
```

パラメータ

colIndex
第 1 列は 1、第 2 列は 2 のように指定します。

getVector()

このメソッドは、現行の位置の列をベクターで戻します。列はコレクション型（VARRAY またはネストした表）であることが必要です。コレクション内の要素の SQL 型は、ベクター内のオブジェクトのデータ型との互換性が必要です。

構文
様々な構文があります。

```
void getVector(ResultSet *rs,
   unsigned int index,
   vector<int> &vect);

void getVector(ResultSet *rs,
   unsigned int index,
   vector<unsigned int> &vect);

void getVector(ResultSet *rs,
   unsigned int index,
   vector<float> &vect);

void getVector(ResultSet *rs,
   unsigned int index,
   vector<double> &vect);
```

```
void getVector(ResultSet *rs,
   unsigned int index,
   vector<string> &vect);

void getVector(ResultSet *rs,
   unsigned int index,
   vector<Date> &vect);

void getVector(ResultSet *rs,
   unsigned int index,
   vector<Timestamp> &vect);

void getVector(ResultSet *rs,
   unsigned int index,
   vector<RefAny> &vect);

void getVector(ResultSet *rs,
   unsigned int index,
   vector<Blob> &vect);

void getVector(ResultSet *rs,
   unsigned int index,
   vector<Clob> &vect);

void getVector(ResultSet *rs,
   unsigned int index,
   vector<Bfile> &vect);

void getVector(ResultSet *rs,
   unsigned int index,
   vector<Number> &vect);

void getVector(ResultSet *rs,
   unsigned int index,
   vector<IntervalDS> &vect);

void getVector(ResultSet *rs,
   unsigned int index,
   vector<IntervalYM> &vect);

void getVector(ResultSet *rs,
   unsigned int index,
   vector< Ref<T> > &vect);
```

```
void getVector(ResultSet *rs,
   unsigned int index,
   vector<T *> &vect);

void getVector(ResultSet *rs,
   unsigned int index,
   vector<T> &vect);
```

最後の 2 つの構文は同一です。前者は関数テンプレートの一部の順序付けがサポートされているプラットフォームで使用し、後者はこれがサポートされていないプラットフォームで使用します。

次の構文も同様です。

```
void getVector(ResultSet *rs,
   unsigned int index,
   vector< Ref<T> > &vect);
```

この構文は、関数テンプレートの一部の順序付けがサポートされているプラットフォームでのみ使用できます。この関数は将来使用できなくなる予定です。かわりに `getVectorOfRefs()` を使用できます。

パラメータ

rs
結果セットを指定します。

index
列の索引を（第 1 列は 1、第 2 列は 2 のように）指定します。

vect
オブジェクトのベクター（OUT パラメータ）への参照を指定します。

getVectorOfRefs()

このメソッドは、現行の位置の列を REF のベクターで戻します。列は REF のコレクション型（VARRAY またはネストした表）であることが必要です。

構文

```
void getVectorOfRefs(ResultSet *rs,
   unsigned int index,
   vector< Ref<T> > &vect);
```

パラメータ

rs
結果セットを指定します。

index
列の索引を（第 1 列は 1、第 2 列は 2 のように）指定します。

vect
REF のベクター（OUT パラメータ）への参照を指定します。

Ref<T> に対しては、特化したメソッドの getVector ではなく、この関数の使用をお薦めします。

isNull()

列には、SQL の NULL の値が含まれることがあります。isNull() は、最後の列の読込みに、この特別な値があったかどうかをレポートします。値が SQL の NULL かどうかを調べるには、最初に列に対して get*xxx* をコールしてその値の読込みを試行し、次に、isNull() をコールする必要があります。最終列の読込みが SQL の NULL であった場合は true を戻します。

構文

```
bool isNull(unsigned int colIndex) const;
```

パラメータ

colIndex
第 1 列は 1、第 2 列は 2 のように指定します。

isTruncated()

このメソッドは、パラメータの値が切り捨てられているかどうかをチェックします。パラメータの値が切り捨てられている場合は true を、それ以外の場合は false を戻します。

構文

```
bool isTruncated(unsigned int paramIndex) const;
```

パラメータ

paramIndex
第 1 パラメータは 1、第 2 パラメータは 2 のように指定します。

next()

最初、ResultSet は第 1 行の前に位置しています。1 回目の next のコールで第 1 行が現在行になります。2 回目のコールで第 2 行は現在になり、その後も同様に続きます。前の行で読込み可能なストリームがオープンしている場合、その行は暗黙的にクローズします。ResultSet の警告連鎖は、新しい行が読み込まれると消去されます。

ストリーム・モードではない場合、next() は常に RESULT_SET_AVAILABLE または END_OF_DATA を戻します。RESULT_SET_REMOVE_AVAILABLE ステータスが戻ると、データは get*xxx* メソッドで使用できます。このバージョンの next() が使用されると、setDataBuffer() インタフェースでフェッチしているデータに対して配列フェッチが行われます。つまり、get*xxx*() メソッドはコールされません。各列の numRows 分のデータは、setDataBuffer() インタフェースで指定されたバッファ内に取得されます。配列フェッチでは、ストリーム入力が可能なため、get*xxx*Stream() メソッドをコールすることもできます（各列に対して 1 回）。

次のいずれかを戻します。

- DATA_AVAILABLE − get*xxx*() をコールするか、setDataBuffer() で指定したバッファからデータを読み込みます。
- END_OF_FETCH − これ以上のデータはありません。これは、配列フェッチの一連の行の最後を意味します。この値は 0（ゼロ）に定義されます。
- STREAM_DATA_AVAILABLE − getCurrentStreamColumn メソッドをコールし、ストリームを読み込みます。

構文

```
Status next(unsigned int numRows =1);
```

パラメータ

numRows

配列フェッチに対してフェッチする行数を指定します。

preTruncationLength()

切捨て前のパラメータの実際の長さを戻します。

構文

```
int preTruncationLength(unsigned int paramIndex) const;
```

パラメータ

paramIndex
第 1 パラメータは 1、第 2 パラメータは 2 のように指定します。

setBinaryStreamMode()

getStream メソッドが、列をバイナリ・ストリームで戻すように定義します。

構文

```
void setBinaryStreamMode(unsigned int colIndex,
   unsigned int size);
```

パラメータ

colIndex
バイナリ・ストリームで取り出される列の位置を（第 1 列は 1、第 2 列は 2 のように）指定します。

size
バイナリ・ストリームで読み込まれるデータ量を指定します。

setCharacterStreamMode()

getStream メソッドが、列をキャラクタ・ストリームで戻すように定義します。

構文

```
void setCharacterStreamMode(unsigned int colIndex,
   unsigned int size);
```

パラメータ

colIndex
キャラクタ・ストリームで取り出される列の位置を（第 1 列は 1、第 2 列は 2 のように）指定します。

size
キャラクタ・ストリームで読み込まれるデータ量を指定します。

setCharSet()

指定された列のデフォルトのキャラクタ・セットを上書きします。データが、データベースのキャラクタ・セットからこの列に指定されたキャラクタ・セットに変換されます。

構文

```
void setCharSet(unsigned int colIndex,
   string charSet);
```

パラメータ

colIndex
第 1 列は 1、第 2 列は 2 のように指定します。

charSet
希望するキャラクタ・セットを文字列で指定します。

setDatebaseNCHARParam()

パラメータを、データベースの NCHAR キャラクタ・セットのデータが含まれている列から取り出そうとする場合は、true 値を渡すことによって、そのことを OCCI に通知する必要があります。デフォルトをリストアするには false を渡します。

構文

```
void setDatabaseNCHARParam( unsigned int paramIndex,
    bool isNCHAR);
```

パラメータ

paramIndex
パラメータ索引を指定します。

isNCHAR
true または false を指定します。

setDataBuffer()

データをフェッチするデータ・バッファを指定します。この *buffer* パラメータは、ユーザーが割り当てたデータ・バッファへのポインタです。データの現在の長さを **length* パラメータに指定する必要があります。データ量は、*size* パラメータを超えないようにします。*type* には、データのデータ型を指定します。OCCI 固有でも C++ 固有の型でもない型のみ使用できます、つまり、STL 文字列のみを使用できます。Bytes や Date などの OCCI クラスは使用できません。

配列フェッチ用に、setDataBuffer() を使用してデータをフェッチする場合は、各結果セットごとに 1 回のみコールする必要があります。各行のデータは、buffer + (i - 1)*size（*i* は行番号）の位置にあるとみなされます。同様に、データの長さは、*(length + (i - 1)) とみなされます。

構文

```
void setDataBuffer(unsigned int colIndex,
   void *buffer,
   Type type,
   sb4 size = 0,
   ub2 *length = NULL,
   sb2 *ind = NULL,
   ub2 *rc = NULL);
```

パラメータ

colIndex

第 1 列は 1、第 2 列は 2 のように指定します。

buffer

ユーザーが割り当てたバッファへのポインタを指定します。配列フェッチが行われる場合、バッファには numRows * size のバイト数が必要です。

type

バッファに入る（取得する）データの型を指定します。

size

データ・バッファのサイズを指定します。配列フェッチの場合、このサイズはデータ項目の各要素のサイズです。

length
バッファのデータ長へのポインタを指定します。配列フェッチの場合、データ長は各バッファ要素に対する長さデータの配列です。配列のサイズは arrayLength と等しくしてください。

ind
インジケータ変数または配列へのポインタ（IN/OUT）を指定します。

rc
列レベルのリターン・コードの配列へのポインタ（OUT）を指定します。

setErrorOnNull()

このメソッドは、結果セットの colIndex 列での NULL 値の読込みに対する例外を使用可能 / 使用禁止にします。

構文

```
void setErrorOnNull(unsigned int colIndex,
   bool causeException);
```

パラメータ

colIndex
第 1 列は 1、第 2 列は 2 のように指定します。

causeExcetpion
true の場合、例外は使用可能です。false の場合は使用禁止です。

setErrorOnTruncate()

このメソッドは、切捨てが発生する際の例外を使用可能 / 使用禁止にします。

構文

```
void setErrorOnTruncate(unsigned int paramIndex,
   bool causeException);
```

パラメータ

paramIndex
第 1 パラメータは 1、第 2 パラメータは 2 のように指定します。

causeException
true の場合、例外は使用可能です。false の場合は使用禁止です。

setMaxColumnSize()

列に読み込むデータの最大量を設定します。

構文

```
void setMaxColumnSize(unsigned int colIndex,
   unsigned int max);
```

パラメータ

colIndex

第 1 列は 1、第 2 列は 2 のように指定します。

max

読み込むデータの最大量を指定します。

status()

結果セットの現在のステータスを戻します。このメソッドは、結果のステータスを調査するために繰り返してコールできます。RESULT_SET_AVAILABLE ステータスが戻ると、データは get*xxx* メソッドで使用できます。次のいずれかを戻します。

- DATA_AVAILABLE − get*xxx*() をコールするか、setDataBuffer() メソッドで指定したバッファからデータを読み込みます。
- STREAM_DATA_AVAILABLE − getCurrentStream() をコールして、ストリームを読み込みます。
- END_OF_FETCH

構文

```
Status status() const;
```

SQLException クラス

SQLException クラスは、生成されたエラー、エラー・コードおよび関連するメッセージに関する情報を提供します。

SQLException()

これは、SQLException コンストラクタです。

構文
様々な構文があります。

```
SQLException();

SQLException(const SQLException &e);
```

SQLException メソッドの概要

表 8-19 SQLException

メソッド	概要
「getErrorCode()」（8-163 ページ）	データベース・エラー・コードを戻します。
「getMessage()」（8-164 ページ）	例外のメッセージ文字列を戻します。
「setErrorCtx()」（8-164 ページ）	エラー・コンテキストを設定します。

getErrorCode()

データベース・エラー・コードを取得します。

構文

```
int getErrorCode() const;
```

getMessage()

エラー・メッセージ文字列が作成されていた場合は、この SQLException のエラー・メッセージ文字列を戻します。SQLException にエラー・メッセージが作成されていない場合は、NULL を戻します。

構文

```
string getMessage() const;
```

setErrorCtx()

エラー・コンテキストへのポインタを設定します。

構文

```
void setErrorCtx(void *ctx);
```

パラメータ

ctx
エラー・コンテキストへのポインタを指定します。

Statement クラス

Statement オブジェクトは、SQL 文を実行するために使用します。この SQL 文は、結果セットを戻す問合せ文または更新カウントを戻す非問合せ文の場合があります。

非問合せ SQL は、INSERT 文、UPDATE 文または DELETE 文にできます。非問合せ SQL 文は、DDL 文（CREATE や GRANT など）またはストアド・プロシージャ・コールにもできます。

INSERT/UPDATE/DELETE 文の問合せまたはストアド・プロシージャ・コール文には、IN バインド・パラメータを指定できます。INSERT/UPDATE/DELETE 文を戻す DML またはストアド・プロシージャ・コールには、OUT バインド・パラメータを指定できます。ストアド・プロシージャ・コール文には、IN/OUT パラメータと呼ばれる IN と OUT 両方のパラメータを指定できます。

Statement クラスのメソッドは、3 つのカテゴリに分類されます。

- すべての文に適用可能な Statement メソッド
- IN バインド・パラメータ付きでプリコンパイルされた SQL 文に適用可能なメソッド
- コール可能文および OUT バインド・パラメータ付きの文を戻す DML に適用可能なメソッド

コンストラクタでは、次の構文を使用します。

```
Statement()
enum Status
  {
    UNPREPARED,
    PREPARED,
    RESULT_SET_AVAILABLE,
    UPDATE_COUNT_AVAILABLE,
    NEEDS_STREAM_DATA,
    STREAM_DATA_AVAILABLE
  };
```

Statement メソッドの概要

表 8-20 Statement メソッド

メソッド	説明
「addIteration()」（8-170 ページ）	実行する反復を追加します。
「closeResultSet()」（8-170 ページ）	自動リリースを待機せずに、結果セットのデータベースおよび OCCI リソースをただちに解放します。
「closeStream()」（8-170 ページ）	パラメータ *stream* によって指定されたストリームをクローズします。
「execute()」（8-171 ページ）	SQL 文を実行します。
「executeArrayUpdate()」（8-172 ページ）	バインド・パラメータに `setDataBuffer()` またはストリーム・インタフェースのみを使用する `INSERT/UPDATE/DELETE` 文を実行します。
「executeQuery()」（8-173 ページ）	単一の `ResultSet` を戻す SQL 文を実行します。
「executeUpdate()」（8-173 ページ）	`ResultSet` を戻さない SQL 文を実行します。
「getAutoCommit()」（8-174 ページ）	現在の自動コミット状態を戻します。
「getBfile()」（8-174 ページ）	`BFILE` の値を `Bfile` オブジェクトで戻します。
「getBlob()」（8-174 ページ）	`BLOB` の値を `Blob` オブジェクトで戻します。
「getBytes()」（8-174 ページ）	`SQL BINARY` または `VARBINARY` パラメータの値を `Bytes` で戻します。
「getCharSet()」（8-175 ページ）	指定されたパラメータに有効なキャラクタ・セットを戻します。
「getClob()」（8-175 ページ）	`CLOB` の値を `Clob` オブジェクトで戻します。
「getConnection()」（8-175 ページ）	
「getCurrentIteration()」（8-176 ページ）	処理中の現行の反復の反復番号を戻します。
「getCurrentStreamIteration()」（8-176 ページ）	ストリーム・データの読込みまたは書込みを行う現行の反復を戻します。
「getCurrentStreamParam()」（8-176 ページ）	読込みまたは書込みが必要な現行の出力ストリームのパラメータ索引を戻します。
「getCursor()」（8-176 ページ）	`OUT` パラメータの `REF CURSOR` 値を `ResultSet` で戻します。
「getDatabaseNCHARParam()」（8-177 ページ）	データが `NCHAR` キャラクタ・セットにあるかどうかを戻します。

表 8-20 Statement メソッド（続き）

メソッド	説明
「getDate()」（8-177 ページ）	パラメータの値を Date オブジェクトで戻します。
「getDouble()」（8-177 ページ）	パラメータの値を C++ の double 型で戻します。
「getFloat()」（8-178 ページ）	パラメータの値を C++ の float 型で戻します。
「getInt()」（8-178 ページ）	パラメータの値を C++ の int 型で戻します。
「getIntervalDS()」（8-178 ページ）	パラメータの値を IntervalDS オブジェクトで戻します。
「getIntervalYM()」（8-179 ページ）	パラメータの値を IntervalYM オブジェクトで戻します。
「getMaxIterations()」（8-179 ページ）	最大反復回数の現在の制限を戻します。
「getMaxParamSize()」（8-179 ページ）	最大パラメータ・サイズの現在の制限を戻します。
「getNumber()」（8-180 ページ）	パラメータの値を Number オブジェクトで戻します。
「getObject()」（8-180 ページ）	パラメータの値を PObject オブジェクトで戻します。
「getOCIStatement()」（8-180 ページ）	Statement に関連付けられている OCI 文ハンドルを戻します。
「getRef()」（8-181 ページ）	REF パラメータの値を RefAny オブジェクトで戻します。
「getResultSet()」（8-181 ページ）	現在の結果を ResultSet で戻します。
「getRowid()」（8-181 ページ）	ROWID パラメータの値を Bytes オブジェクトで戻します。
「getSQL()」（8-181 ページ）	Statement オブジェクトに関連付けられている現行の SQL 文字列を戻します。
「getStream()」（8-182 ページ）	パラメータの値をストリームで戻します。
「getString()」（8-182 ページ）	パラメータの値を文字列で戻します。
「getTimestamp()」（8-182 ページ）	パラメータの値を Timestamp オブジェクトで戻します。
「getUInt()」（8-183 ページ）	パラメータの値を C++ の unsigned int 型で戻します。
「getUpdateCount()」（8-183 ページ）	非問合せ文に対する現在の結果を更新カウントで戻します。
「getVector()」（8-153 ページ）	指定されたパラメータをベクターで戻します。

表 8-20 Statement メソッド（続き）

メソッド	説明
「isNull()」（8-186 ページ）	パラメータが NULL かどうかをチェックします。
「isTruncated()」（8-186 ページ）	値が切り捨てられているかどうかをチェックします。
「preTruncationLength()」（8-187 ページ）	
「registerOutParam()」（8-187 ページ）	OUT パラメータの型と最大サイズを登録します。
「setAutoCommit()」（8-188 ページ）	自動コミット・モードを指定します。
「setBfile()」（8-188 ページ）	パラメータを Bfile 値に設定します。
「setBinaryStreamMode()」（8-189 ページ）	列をバイナリ・ストリームで戻すように指定します。
「setBlob()」（8-189 ページ）	パラメータを Blob 値に設定します。
「setBytes()」（8-189 ページ）	パラメータを Bytes 配列に設定します。
「setCharacterStreamMode()」（8-190 ページ）	列をキャラクタ・ストリームで戻すように指定します。
「setCharSet()」（8-190 ページ）	指定されたパラメータにキャラクタ・セットを指定します。
「setClob()」（8-191 ページ）	パラメータを Clob 値に設定します。
「setDate()」（8-191 ページ）	パラメータを Date 値に設定します。
「setDatabaseNCHARParam()」（8-192 ページ）	データがデータベースの NCHAR キャラクタ・セットにある場合は true を、デフォルトをリストアする場合は false を設定します。
「setDataBuffer()」（8-192 ページ）	読込みまたは書込み可能なデータがあるデータ・バッファを指定します。
「setDataBufferArray()」（8-194 ページ）	読込みまたは書込み可能なデータがあるデータ・バッファの配列を指定します。
「setDouble()」（8-195 ページ）	パラメータを C++ の double 型の値に設定します。
「setErrorOnNull()」（8-196 ページ）	NULL 値の読込みに対する例外を使用可能 / 使用禁止にします。
「setErrorOnTruncate()」（8-196 ページ）	切捨てが発生する場合の例外を使用可能 / 使用禁止にします。
「setFloat()」（8-197 ページ）	パラメータを C++ の float 型の値に設定します。
「setInt()」（8-197 ページ）	パラメータを C++ の int 型の値に設定します。
「setIntervalDS()」（8-197 ページ）	パラメータを IntervalDS 値に設定します。

表 8-20 Statement メソッド（続き）

メソッド	説明
「setIntervalYM()」（8-198 ページ）	パラメータを IntervalYM 値に設定します。
「setMaxIterations()」（8-198 ページ）	DML 文に対して行われる起動の最大数を設定します。
「setMaxParamSize()」（8-199 ページ）	パラメータとの間で受渡しできるデータの最大量を設定します。
「setNull()」（8-199 ページ）	パラメータを SQL の NULL に設定します。
「setNumber()」（8-200 ページ）	パラメータを Number 値に設定します。
「setObject()」（8-200 ページ）	オブジェクトを使用してパラメータの値を設定します。
「setPrefetchMemorySize()」（8-201 ページ）	サーバーへの各ラウンドトリップでフェッチしたデータを格納するために、OCCI によって繰り返し使用されるメモリー・サイズを設定します。
「setPrefetchRowCount()」（8-201 ページ）	サーバーへの各ラウンドトリップで、OCCI によって繰り返しフェッチされる行数を設定します。
「setRef()」（8-201 ページ）	パラメータを RefAny 値に設定します。
「setRowid()」（8-202 ページ）	バインドの位置に対して、ROWID バイト配列を設定します。
「setSQL()」（8-202 ページ）	新規 SQL 文字列を Statement オブジェクトに関連付けます。
「setString()」（8-203 ページ）	パラメータを文字列値に設定します。
「setTimestamp()」（8-203 ページ）	パラメータを Timestamp 値に設定します。
「setUInt()」（8-203 ページ）	パラメータを C++ の unsigned int 型の値に設定します。
「setVector()」（8-204 ページ）	パラメータをベクターに設定します。
「status()」（8-207 ページ）	文の現在のステータスを戻します。このステータスは、書き込むストリーム・データがある場合に役立ちます。

addIteration()

設定パラメータを指定した後、実行する反復を追加します。

構文

```
void addIteration();
```

closeResultSet()

多くの場合、結果セットのデータベースと OCCI リソースは、自動的にクローズされるのを待たずに、ただちに解放することが理想的です。closeResultSet メソッドは、この即時解放を行います。

構文

```
void closeResultSet(ResultSet *resultSet);
```

パラメータ

resultSet
クローズする結果セットを指定します。結果セットは、この文で getResultSet メソッドをコールすることによって取得しておく必要があります。

closeStream()

パラメータ stream によって指定されたストリームをクローズします。

構文

```
void closeStream(Stream *stream);
```

パラメータ

stream
クローズするストリームを指定します。

execute()

結果セットまたは更新カウントのいずれかを戻す SQL 文を実行します。この文には、書き込む必要がある読込み可能なストリームが含まれている場合があります。この場合の実行結果は、すぐに使用できない場合があります。戻り値は、次のいずれかです。

- UNPREPARED
- PREPARED
- RESULT_SET_AVAILABLE
- UPDATE_COUNT_AVAILABLE
- NEEDS_STREAM_DATA
- STREAM_DATA_AVAILABLE

RESULT_SET_AVAILABLE が戻された場合は、getResultSet() メソッドをコールして、結果セットを取得する必要があります。

UPDATE_COUNT_AVAILABLE が戻された場合は、getUpdateCount メソッドをコールして、更新カウントを確認する必要があります。

NEEDS_STREAM_DATA が戻された場合は、ストリーム化された IN バインド・パラメータに対して、出力 Streams を書き込む必要があります。ストリーム・パラメータが複数の場合は、getCurrentStreamParam メソッドをコールして、ストリームが必要なバインド・パラメータを確認します。文が繰り返し実行される場合は、getCurrentIteration をコールして、繰り返して書込みが必要なストリームを確認する必要があります。

STREAM_DATA_AVAILABLE が戻された場合は、ストリーム化された OUT バインド・パラメータに、入力 Streams を読み込む必要があります。ストリーム・パラメータが複数の場合は、getCurrentStreamParam メソッドをコールして、ストリームが必要なバインド・パラメータを確認します。文が繰り返し実行される場合は、getCurrentIteration をコールして、繰り返して読込みが必要なストリームを確認する必要があります。

文を戻す DML の各コールに対して、1 つの OUT 値のみが戻される場合は、文を戻す DML を反復的に実行できます。出力ストリームが OUT バインド変数に使用されている場合、その出力ストリームは順に従って完全に読み込む必要があります。getCurrentStreamParam メソッドは、読み込む必要があるストリームを示します。同様に、getCurrentIteration は、データの準備ができている反復を示します。

戻り値は、次のとおりです。

- RESULT_SET_AVAILABLE − getResultSet() をコールします。
- UPDATE_COUNT_AVAILABLE − getUpdateCount() をコールします。
- NEEDS_STREAM_DATA − getCurrentStream()、getCurrentIteration() および write（または read）ストリームをコールします。

構文

```
Status execute(const string &sql = "");
```

パラメータ

sql

実行する SQL 文を指定します。setSQL メソッドを使用して SQL と文を関連付けた場合は、NULL を指定できます。

executeArrayUpdate()

バインド・パラメータに setDataBuffer() またはストリーム・インタフェースのみを使用する INSERT/UPDATE/DELETE 文を実行します。このバインド・パラメータは、size arrayLength パラメータの配列であることが必要です。この文には、書き込む必要がある読込み可能なストリームが含まれている場合があります。戻り値は、次のいずれかです。

- UPDATE_COUNT_AVAILABLE
- NEEDS_STREAM_DATA
- STREAM_DATA_AVAILABLE
- PREPARED
- UNPREPARED

UPDATE_COUNT_AVAILABLE が戻された場合は、getUpdateCount() をコールして、更新カウントを確認する必要があります。

NEEDS_STREAM_DATA が戻された場合は、ストリーム化されたバインド・パラメータに対して、出力 Streams を書き込む必要があります。ストリーム・パラメータが複数の場合は、getCurrentStreamParam() をコールして、ストリームが必要なバインド・パラメータを確認できます。getCurrentIteration() をコールして、書込みが必要なストリームの反復を確認できます。

STREAM_DATA_AVAILABLE が戻された場合は、ストリーム化された OUT バインド・パラメータに、入力 Streams を読み込む必要があります。ストリーム・パラメータが複数の場合は、getCurrentStreamParam() をコールして、ストリームが必要なバインド・パラメータを確認できます。文が繰り返し実行される場合は、getCurrentIteration() をコールして、繰り返して読込みが必要なストリームを確認する必要があります。

文を戻す DML の各コールに対して、1 つの OUT 値のみが戻される場合は、文を戻す DML に対して配列実行を行うこともできます。出力ストリームが OUT バインド変数に使用されている場合、その出力ストリームは順に従って完全に読み込む必要があります。getCurrentStreamParam() メソッドは、読み込む必要があるストリームを示します。同様に、getCurrentIteration() は、データの準備ができている反復を示します。

配列実行は、問合せ文またはコール可能文に対しては実行できません。

構文

```
Status executeArrayUpdate(unsigned int arrayLength);
```

パラメータ

arrayLength

バインド変数の各バッファに指定する要素数を指定します。文は、各反復ごとに使用する各配列要素で、ここで指定した回数だけ繰返し実行されます。戻り値は、次のとおりです。

- UPDATE_COUNT_AVAILABLE − getUpdateCount() をコールします。
- NEEDS_STREAM_DATA − getCurrentStream()、getCurrentIteration() および write（または read）ストリームをコールします。

executeQuery()

単一の ResultSet を戻す SQL 文を実行します。ストリーム・パラメータを持つ問合せではない文に対しては、コールしないでください。問合せによって作成されたデータが含まれた ResultSet を戻します。

構文

```
ResultSet * executeQuery(const string &sql = "");
```

パラメータ

sql

実行する SQL 文を指定します。setSQL() を使用して SQL と文を関連付けた場合は、NULL を指定できます。

executeUpdate()

SQL の INSERT 文、UPDATE 文、DELETE 文などの非問合せ文、CREATE/ALTER などの DDL 文またはストアド・プロシージャ・コールを実行します。INSERT、UPDATE または DELETE の場合は行カウントを、戻りがない SQL 文の場合は 0（ゼロ）を戻します。

構文

```
unsigned int executeUpdate(const string &sql = "");
```

パラメータ

sql

実行する SQL 文を指定します。setSQL メソッドを使用して SQL と文を関連付けた場合は、NULL を指定できます。

getAutoCommit()

現在の自動コミット状態を取得します。自動コミット・モードの現在の状態を戻します。

構文

```
bool getAutoCommit() const;
```

getBfile()

BFILE パラメータの値を Bfile オブジェクトで取得します。パラメータの値を戻します。

構文

```
Bfile getBfile(unsigned int paramIndex);
```

パラメータ

paramIndex
第 1 パラメータは 1、第 2 パラメータは 2 のように指定します。

getBlob()

BLOB パラメータの値を Blob オブジェクトで取得します。パラメータの値を戻します。

構文

```
Blob getBlob(unsigned int paramIndex);
```

パラメータ

paramIndex
第 1 パラメータは 1、第 2 パラメータは 2 のように指定します。

getBytes()

SQL BINARY または VARBINARY パラメータの値を Bytes で取得します。パラメータの値を戻します。値が SQL の NULL の場合、結果は NULL です。

構文

```
Bytes getBytes(unsigned int paramIndex);
```

パラメータ

paramIndex
第 1 パラメータは 1、第 2 パラメータは 2 のように指定します。

getCharSet()

指定されたパラメータに有効なキャラクタ・セットを文字列で戻します。

構文
```
string getCharSet(unsigned int paramIndex) const;
```

パラメータ

paramIndex
第 1 パラメータは 1、第 2 パラメータは 2 のように指定します。

getClob()

CLOB パラメータの値を Clob オブジェクトで取得します。パラメータの値を戻します。

構文
```
Clob getClob(unsigned int paramIndex);
```

パラメータ

paramIndex
第 1 パラメータは 1、第 2 パラメータは 2 のように指定します。

getConnection()

構文
```
const Connection* getConnection() const;
```

getCurrentIteration()

プリコンパイルされた SQL 文に出力 Streams がない場合、このメソッドは、OCCI で処理中の文の現在の反復を戻します。一連の反復ですべてのコールが処理された後に、このメソッドがコールされた場合は、0（ゼロ）を戻します。通常は、処理中の現行の反復の反復番号を戻します。最初の反復の番号は 1、2 回目の反復の番号は 2 のようになります。文の実行が終了した場合は、0（ゼロ）を戻します。

構文

```
unsigned int getCurrentIteration() const;
```

getCurrentStreamIteration()

データの準備ができている現在のパラメータ・ストリームを戻します。

構文

```
unsigned int getCurrentStreamIteration() const;
```

getCurrentStreamParam()

プリコンパイルされた SQL 文に出力 Stream パラメータがある場合、このメソッドは書込みが必要な現行の出力 Stream のパラメータ索引を戻します。出力 Stream への書込みが不要な場合、またはプリコンパイルされた SQL 文に出力 Stream パラメータがない場合、このメソッドは 0（ゼロ）を戻します。

書込みが必要な現行の出力 Stream パラメータのパラメータ索引を戻します。

構文

```
unsigned int getCurrentStreamParam() const;
```

getCursor()

OUT パラメータの REF CURSOR 値を ResultSet で戻します。この結果セットからデータをフェッチできます。OUT パラメータは、Statement.registerOutParam(int paramIndex, CURSOR) メソッドを使用して CURSOR として登録する必要があります。バッチ・コールのために戻される REF CURSOR が複数ある場合は、次の REF CURSOR を取得してフェッチを開始する前に、各カーソルからのデータを完全にフェッチする必要があります。OUT パラメータの値に対して ResultSet を 1 つ戻します。

構文

```
ResultSet * getCursor(unsigned int paramIndex);
```

パラメータ

paramIndex
第 1 パラメータは 1、第 2 パラメータは 2 のように指定します。

getDatabaseNCHARParam()

データが NCHAR キャラクタ・セットにあるかどうかを戻します。

構文
```
bool getDatabaseNCHARParam(unsigned int paramIndex) const;
```

パラメータ

paramIndex
第 1 パラメータは 1、第 2 パラメータは 2 のように指定します。

getDate()

SQL DATE パラメータの値を Date オブジェクトで取得します。パラメータの値を戻します。値が SQL の NULL の場合、結果は NULL です。

構文
```
Date getDate(unsigned int paramIndex) const;
```

パラメータ

paramIndex
第 1 パラメータは 1、第 2 パラメータは 2 のように指定します。

getDouble()

DOUBLE パラメータの値を C++ の double 型で取得します。パラメータの値を戻します。値が SQL の NULL の場合、結果は 0（ゼロ）です。

構文
```
double getDouble(unsigned int paramIndex);
```

パラメータ

paramIndex
第 1 パラメータは 1、第 2 パラメータは 2 のように指定します。

getFloat()

FLOAT パラメータの値を C++ の float 型で取得します。パラメータの値を戻します。値が SQL の NULL の場合、結果は 0（ゼロ）です。

構文
```
float getFloat(unsigned int paramIndex);
```

パラメータ

paramIndex
第 1 パラメータは 1、第 2 パラメータは 2 のように指定します。

getInt()

INTEGER パラメータの値を C++ の int 型で取得します。パラメータの値を戻します。値が SQL の NULL の場合、結果は 0（ゼロ）です。

構文
```
unsigned int getInt(unsigned int paramIndex);
```

パラメータ

paramIndex
第 1 パラメータは 1、第 2 パラメータは 2 のように指定します。

getIntervalDS()

パラメータの値を IntervalDS オブジェクトで取得します。

構文
```
IntervalDS getIntervalDS(unsigned int paramIndex);
```

パラメータ

paramIndex
第 1 パラメータは 1、第 2 パラメータは 2 のように指定します。

getIntervalYM()

パラメータの値を IntervalYM オブジェクトで取得します。

構文
```
IntervalYM getIntervalYM(unsigned int paramIndex);
```

パラメータ

paramIndex
第 1 パラメータは 1、第 2 パラメータは 2 のように指定します。

getMaxIterations()

最大反復回数の現在の制限を取得します。デフォルトは 1 です。現在の最大反復回数を戻します。

構文
```
unsigned int getMaxIterations() const;
```

getMaxParamSize()

maxParamSize 制限（バイト数）は、受け渡すパラメータの値の最大データ量です。文字型およびバイナリ型に対してのみ適用されます。制限を超過すると、超過した分のデータは暗黙的に廃棄されます。最大パラメータ・サイズの現在の制限を戻します。

構文
```
unsigned int getMaxParamSize(unsigned int paramIndex) const;
```

パラメータ

paramIndex
第 1 パラメータは 1、第 2 パラメータは 2 のように指定します。

getNumber()

NUMERIC パラメータの値を NUMERIC オブジェクトで取得します。パラメータの値を戻します。値が SQL の NULL の場合、結果は NULL です。

構文

```
Number getNumber(unsigned int paramIndex);
```

パラメータ

paramIndex

第 1 パラメータは 1、第 2 パラメータは 2 のように指定します。

getObject()

パラメータの値を PObject オブジェクトで取得します。このメソッドは registerOutParam を使用して、このパラメータに登録された SQL の型に対応する型の PObject を戻します。このメソッドは、データベース固有の抽象データ型を読み込むために使用できます。OUT パラメータの値を保持している PObject を 1 つ戻します。

構文

```
PObject * getObject(unsigned int paramIndex);
```

パラメータ

paramIndex

第 1 パラメータは 1、第 2 パラメータは 2 のように指定します。

getOCIStatement()

Statement に関連付けられている OCI 文ハンドルを取得します。Statement に関連付けられている OCI 文ハンドルを戻します。

構文

```
LNOCIStmt * getOCIStatement() const;
```

getRef()

REF パラメータの値を RefAny オブジェクトで取得します。パラメータの値を戻します。

構文

```
RefAny getRef(unsigned int paramIndex);
```

パラメータ

paramIndex
第 1 パラメータは 1、第 2 パラメータは 2 のように指定します。

getResultSet()

現在の結果を ResultSet で戻します。

構文

```
ResultSet * getResultSet();
```

getRowid()

rowid パラメータの値を Bytes で取得します。

構文

```
Bytes getRowid(unsigned int paramIndex);
```

パラメータ

paramIndex
第 1 パラメータは 1、第 2 パラメータは 2 のように指定します。

getSQL()

Statement オブジェクトに関連付けられている現行の SQL 文字列を戻します。

構文

```
string getSQL() const;
```

getStream()

構文

```
Stream * getStream(unsigned int paramIndex);
```

パラメータ

paramIndex

第 1 パラメータは 1、第 2 パラメータは 2 のように指定します。

getString()

CHAR、VARCHAR または LONGVARCHAR パラメータの値を文字列で取得します。パラメータの値を戻します。値が SQL の NULL の場合、結果は空の文字列です。

構文

```
string getString(unsigned int paramIndex);
```

パラメータ

paramIndex

第 1 パラメータは 1、第 2 パラメータは 2 のように指定します。

getTimestamp()

SQL TIMESTAMP パラメータの値を Timestamp オブジェクトで取得します。パラメータの値を戻します。値が SQL の NULL の場合、結果は NULL です。

構文

```
Timestamp getTimestamp(unsigned int paramIndex);
```

パラメータ

paramIndex

第 1 パラメータは 1、第 2 パラメータは 2 のように指定します。

getUInt()

BIGINT パラメータの値を C++ の unsigned int 型で取得します。パラメータの値を戻します。値が SQL の NULL の場合、結果は 0（ゼロ）です。

構文

```
unsigned int getUInt(unsigned int paramIndex);
```

パラメータ

paramIndex
第 1 パラメータは 1、第 2 パラメータは 2 のように指定します。

getUpdateCount()

現在の結果を更新カウントで戻します。

構文

```
unsigned int getUpdateCount() const;
```

getVector()

このメソッドは、現行の位置の列をベクターで戻します。位置の列は、索引で指定し、コレクション型（VARRAY またはネストした表）であることが必要です。コレクション内の要素の SQL 型は、ベクターの型との互換性が必要です。

構文
様々な構文があります。

```
void getVector(Statement *stmt,
   unsigned int paramIndex,
   vector<int> &vect;

void getVector(Statement *stmt,
   unsigned int paramindex,
   vector<string> &vect;

void getVector(Statement *stmt,
   unsigned int index,
   vector<unsigned int> &vect;
```

```
void getVector(Statement *stmt,
   unsigned int paramindex,
   vector<float> &vect;

void getVector(Statement *stmt,
   unsigned int paramindex,
   vector<double> &vect;

void getVector(Statement *stmt,
   unsigned int paramindex,
   vector<Date> &vect;

void getVector(Statement *stmt,
   unsigned int paramindex,
   vector<Timestamp> &vect;

void getVector(Statement *stmt,
   unsigned int paramIndex,
   vector<RefAny> &vect;

void getVector(Statement *stmt,
  unsigned int paramIndex,
   vector<Blob> &vect;

void getVector(Statement *stmt,
   unsigned int paramIndex,
   vector<Clob> &vect;

void getVector(Statement *stmt,
   unsigned int paramIndex,
   vector<Bfile> &vect;

void getVector(Statement *stmt,
   unsigned int paramIndex,
   vector<Number> &vect;

void getVector(Statement *stmt,
   unsigned int paramIndex,
   vector<IntervalDS> &vect;

void getVector(Statement *stmt,
   unsigned int paramIndex,
   vector<IntervalYM> &vect;
```

```
void getVector(Statement *stmt,
   unsigned int paramIndex,
   vector<T *> &vect);

void getVector(Statement *stmt,
   unsigned int paramIndex,
   vector<T> &vect);
```

最後の 2 つの構文は同一です。前者は関数テンプレートの一部の順序付けがサポートされているプラットフォームで使用し、後者はこれがサポートされていないプラットフォームで使用します。

次の構文も同様です。

```
void getVector(Statement *stmt,
   unsigned int paramIndex,
   vector< Ref<T> > &vect);
```

この構文は、関数テンプレートの一部の順序付けがサポートされているプラットフォームでのみ使用できます。この関数は今後使用できなくなります。かわりに getVectorOfRefs() を使用できます。

パラメータ

statement
文を指定します。

paramIndex
第 1 パラメータは 1、第 2 パラメータは 2 のように指定します。

vect
値が取得されるベクター（OUT パラメータ）への参照を指定します。

getVectorOfRefs()

このメソッドは、現行の位置の列を REF のベクターで戻します。列は REF のコレクション型（VARRAY またはネストした表）であることが必要です。

構文

```
void getVectorOfRefs(Statement *stmt,
   unsigned int index,
   vector< Ref<T> > &vect);
```

パラメータ

stmt
文を指定します。

index
列の索引を（第 1 列は 1、第 2 列は 2 のように）指定します。

vect
REF のベクター（OUT パラメータ）への参照を指定します。

Ref<T> に対しては、特化した関数の getVector ではなく、getVectorOfRefs の使用をお薦めします。

isNull()

OUT パラメータには、SQL の NULL の値が含まれる場合があります。isNull は、最後の値の読込みに、この特別な値かあるかどうかをレポートします。最初に、パラメータで getXXX をコールしてこの値を読み込み、次に、isNull() をコールして、値が SQL の NULL かどうかを確認する必要があります。最後のパラメータ読込みが SQL の NULL の場合は true を戻します。

構文

```
bool isNull(unsigned int paramIndex ) const;
```

パラメータ

paramIndex
第 1 パラメータは 1、第 2 パラメータは 2 のように指定します。

isTruncated()

このメソッドは、パラメータの値が切り捨てられているかどうかをチェックします。パラメータの値が切り捨てられている場合は true を、それ以外の場合は false を戻します。

構文

```
bool isTruncated(unsigned int paramIndex) const;
```

パラメータ

paramIndex
第 1 パラメータは 1、第 2 パラメータは 2 のように指定します。

preTruncationLength()

切捨て前のパラメータの実際の長さを戻します。

構文

```
int preTruncationLength(unsigned int paramIndex) const;
```

パラメータ

paramIndex
第 1 パラメータは 1、第 2 パラメータは 2 のように指定します。

registerOutParam()

このメソッドは、PL/SQL ストアド・プロシージャの各 out パラメータの type を登録します。PL/SQL ストアド・プロシージャを実行する前に、このメソッドを明示的にコールして各 out パラメータの type を登録する必要があります。このメソッドは、out パラメータに対してのみコールします。IN/OUT パラメータの場合は setxxx メソッドを使用します。

out パラメータの値を読み込む場合は、登録されているパラメータの SQL 型に対応する getxxx メソッドを使用する必要があります。たとえば、指定されている型が OCCIINT または OCCINumber の場合は、getInt または getNumber を使用します。

PL/SQL ストアド・プロシージャに ROWID 型の out パラメータがある場合、このメソッドに指定する type には OCCISTRING を指定します。out パラメータの値は、getString() メソッドをコールして取得できます。

PL/SQL ストアド・プロシージャに ROWID 型の IN/OUT パラメータがある場合は、setString() メソッドと getString() メソッドをコールして、型を設定し、IN/OUT パラメータの値を取得します。

構文

```
void registerOutParam(unsigned int paramIndex,
   Type type,
   unsigned int maxSize = 0,
   const string &sqltype = "");
```

パラメータ

paramIndex
第 1 パラメータは 1、第 2 パラメータは 2 のように指定します。

type
type によって定義される SQL 型コードです。Date や Bytes などの OCCI データ型に対応するデータ型のみです。

maxSize
取得する値の最大サイズを指定します。OCCIBYTES 型および OCCISTRING 型の場合、*maxSize* は 1 以上です。

sqltype
データベースの型名を指定します（CREATE TYPE で作成した型を使用します）。

setAutoCommit()

Statement を自動コミット・モードにできます。この場合、実行済みの文も自動的にコミットされます。デフォルトでは、自動コミットモードはオフです。

構文

```
void setAutoCommit(bool autoCommit);
```

パラメータ

autoCommit
true は自動コミット使用可能、false は自動コミット使用禁止です。

setBfile()

パラメータを Bfile 値に設定します。

構文

```
void setBfile(unsigned int paramIndex,
   const Bfile &x);
```

パラメータ

paramIndex
第 1 パラメータは 1、第 2 パラメータは 2 のように指定します。

x
パラメータの値を指定します。

setBinaryStreamMode()

getStream メソッドが、列をバイナリ・ストリームで戻すように定義します。

構文

```
void setBinaryStreamMode(unsigned int colIndex,
   unsigned int size);
```

パラメータ

colIndex
ストリームとしてバインドする列を指定します。第 1 列は 1、第 2 列は 2 のように指定します。

size
バイナリ・ストリームで読み込まれたり、戻されるデータ量を指定します。

setBlob()

パラメータを Blob 値に設定します。

構文

```
void setBlob(unsigned int paramIndex,
   const Blob &x);
```

パラメータ

paramIndex
第 1 パラメータは 1、第 2 パラメータは 2 のように指定します。

x
パラメータの値を指定します。

setBytes()

パラメータを Bytes 配列に設定します。

構文

```
void setBytes(unsigned int paramIndex,
   const Bytes &x);
```

パラメータ

paramIndex
第 1 パラメータは 1、第 2 パラメータは 2 のように指定します。

x
パラメータの値を指定します。

setCharacterStreamMode()

getStream メソッドが、列をキャラクタ・ストリームで戻すように定義します。

構文

```
void setCharacterStreamMode(unsigned int colIndex,
   unsigned int size);
```

パラメータ

colIndex
第 1 列は 1、第 2 列は 2 のように指定します。

size

setCharSet()

指定されたパラメータのデフォルトのキャラクタ・セットを上書きします。 データは、指定されたキャラクタ・セットにあるとみなされ、データベース・キャラクタ・セットに変換されます。OUT バインドの場合、これは、データベース・キャラクタの変換先のキャラクタ・セットを指定します。

構文

```
void setCharSet(unsigned int paramIndex,
   string charSet);
```

パラメータ

paramIndex
第 1 パラメータは 1、第 2 パラメータは 2 のように指定します。

charSet
選択したキャラクタ・セットを文字列で指定します。

setClob()

パラメータを Clob 値に設定します。

構文

```
void setClob(unsigned int paramIndex,
   const Clob &x);
```

パラメータ

paramIndex
第 1 パラメータは 1、第 2 パラメータは 2 のように指定します。

x
パラメータの値を指定します。

setDate()

パラメータを Date 値に設定します。

構文

```
void setDate(unsigned int paramIndex,
   const Date &x);
```

パラメータ

paramIndex
第 1 パラメータは 1、第 2 パラメータは 2 のように指定します。

x
パラメータの値を指定します。

setDatabaseNCHARParam()

パラメータを、データベースの NCHAR キャラクタ・セットのデータが含まれている列に挿入しようとする場合は、true 値を渡すことによって、そのことを OCCI に通知する必要があります。デフォルトをリストアするには、false を渡します。戻り値によって、指定されたパラメータに有効なキャラクタ・セットが戻されます。

構文

```
void setDatabaseNCHARParam(unsigned int paramIndex,
   bool isNCHAR);
```

パラメータ

paramIndex
第 1 パラメータは 1、第 2 パラメータは 2 のように指定します。

isNCHAR
このパラメータに、データベースの NCHAR キャラクタ・セットのデータが含まれている場合は true を、それ以外の場合は false を戻します。

setDataBuffer()

データが使用可能なデータ・バッファを指定します。このメソッドは、コール可能文の OUT バインド・パラメータ（および将来的には、OUT バインドを戻す DML）に対しても使用します。

この *buffer* パラメータは、ユーザーが割り当てたデータ・バッファへのポインタです。データの現在の長さを **length* パラメータに指定する必要があります。データ量は、*size* パラメータを超えないようにします。*type* には、データのデータ型を指定します。

すべての types がバッファに指定できるわけではありません。たとえば、OCCI に割り当てられているすべての型（Bytes や Date など）を setDataBuffer インタフェースでは指定できません。同様に、C++ 標準ライブラリ文字列は、setDataBuffer インタフェースでは提供できません。type は、VARCHAR2、CSTRING、CHARZ などの OCI データ型にできます。

反復実行や配列実行で、setDataBuffer() を使用してデータを指定する場合は、最初の反復のみで 1 回のみコールする必要があります。後続の反復で、OCCI は、データの位置を buffer + i*size（i は反復番号）であるとみなします。同様に、データの長さは、*(length + i) とみなされます。

構文

```
void setDataBuffer(unsigned int paramIndex,
   void *buffer,
   Type type,
   sb4 size,
   ub2 *length,
   sb2 *ind = NULL,
   ub2 *rc= NULL);
```

パラメータ

paramIndex

第 1 パラメータは 1、第 2 パラメータは 2 のように指定します。

buffer

ユーザーが割り当てたバッファへのポインタを指定します。反復実行や配列実行を行う場合、バッファには numIterations * size バイトが必要です。

type

バッファに入る（取得する）データの型を指定します。

size

データ・バッファのサイズを指定します。反復実行や配列実行の場合、このサイズはデータ項目の各要素のサイズです。

length

バッファのデータ長へのポインタを指定します。反復実行や配列実行の場合、データ長は各バッファ要素に対する長さデータの配列です。配列のサイズは arrayLength と等しくしてください。

ind

インジケータを指定します。反復実行および配列実行の場合、インジケータはすべてのバッファ要素に対するものです。

rc

リターン・コードです。反復実行および配列実行の場合、リターン・コードはすべてのバッファ要素に対するものです。

setDataBufferArray()

読込みまたは書込み可能なデータがあるデータ・バッファの配列を指定します。配列パラメータの読込み / 書込みを行うストアド・プロシージャの IN、OUT および IN/OUT バインド・パラメータに使用します。

ストアド・プロシージャは、IN、IN/OUT または OUT パラメータの値の配列を保持できます。この場合、パラメータは setDataBufferArray() メソッドを使用して指定する必要があります。配列は、反復実行や配列実行に対する setDataBuffer() メソッドと同じように指定されますが、配列の要素数は *arrayLength パラメータによって決まります。

OUT および IN/OUT パラメータの場合、配列の最大要素数は、arraySize パラメータで指定されます。プリコンパイルされた反復 SQL 文の場合、配列の要素数は反復回数によって決まります。配列実行の場合の配列の要素数は、executeArrayUpdate() メソッドの arrayLength パラメータによって決まります。ただし、ストアド・プロシージャの配列パラメータの場合、配列の要素数は、setDataBufferArray() メソッドの *arrayLength パラメータで指定する必要があります。これは、各パラメータに異なるサイズの配列がある場合があるためです。

この指定方法はプリコンパイルされた SQL 文の場合と異なります。反復実行および配列実行の場合、各パラメータの配列の要素数は同じであり、要素の数は文の反復数によって決まります。しかし、コール可能文は 1 回のみ実行され、それぞれのパラメータは、長さの異なる可変長の配列である可能性があります。

OUT バインドと IN/OUT バインドの場合、配列の要素数は、同様に *arrayLength で戻ります。クライアントは、*buffer* に elementSize * arraySize 分のバイトが割り当てられていることを確認する必要があります。

構文

```
void setDataBufferArray(unsigned int paramIndex,
   void *buffer,
   Type type,
   ub4 arraySize,
   ub4 *arrayLength,
   sb4 elementSize,
   ub2 *elementLength,
   sb2 *ind = NULL,
   ub2 *rc = NULL);
```

パラメータ

paramIndex

第 1 パラメータは 1、第 2 パラメータは 2 のように指定します。

buffer
ユーザーが割り当てたバッファへのポインタを指定します。バッファには size * arraySize 分のバイトが必要です。

type
バッファに入る（取得する）データの型を指定します。

arraySize
配列の最大要素数を指定します。

arrayLength
配列の現行の要素数へのポインタを指定します。

elementSize
各要素に対するデータ・バッファのサイズを指定します。

elementLength
長さの配列へのポインタを指定します。elementLength[i] は、配列内の i 番目の現在の長さです。

ind
インジケータの配列へのポインタを指定します。これは、すべてのバッファ要素に対するインジケータです。

rc
リターン・コードの配列へのポインタを指定します。

setDouble()

パラメータを C++ の double 型の値に設定します。

構文

```
void setDouble(unsigned int paramIndex,
   double x);
```

パラメータ

paramIndex
第 1 パラメータは 1、第 2 パラメータは 2 のように指定します。

x
パラメータの値を指定します。

setErrorOnNull()

文の paramIndex パラメータで、NULL 値の読込みに対する例外を使用可能 / 使用禁止にします。例外を使用可能にして、*paramIndex* パラメータで get*xxx* をコールすると、パラメータの値が NULL の場合は、結果が SQLException になります。このコールは、例外を使用禁止にする際にも使用できます。

構文

```
void setErrorOnNUll(unsigned int paramIndex,
   bool causeException);
```

パラメータ

paramIndex
第 1 パラメータは 1、第 2 パラメータは 2 のように指定します。

causeExcetpion
true の場合は例外を使用でき、false の場合は使用禁止です。

setErrorOnTruncate()

このメソッドは、切捨てが発生する際の例外を使用可能 / 使用禁止にします。

構文

```
void setErrorOnTruncate(unsigned int paramIndex,
   bool causeException);
```

パラメータ

paramIndex
第 1 パラメータは 1、第 2 パラメータは 2 のように指定します。

causeException
true の場合、例外は使用可能です。false の場合は使用禁止です。

setFloat()

パラメータを C++ の float 型の値に設定します。

構文

```
void setFloat(unsigned int paramIndex,
   float x);
```

パラメータ

paramIndex
第 1 パラメータは 1、第 2 パラメータは 2 のように指定します。

x
パラメータの値を指定します。

setInt()

パラメータを C++ の int 型の値に設定します。

構文

```
void setInt(unsigned int paramIndex,
   int x);
```

パラメータ

paramIndex
第 1 パラメータは 1、第 2 パラメータは 2 のように指定します。

x
パラメータの値を指定します。

setIntervalDS()

パラメータを IntervalDS 値に設定します。

構文

```
void setIntervalDS(unsigned int paramIndex,
   const IntervalDS &x);
```

パラメータ

paramIndex
第 1 パラメータは 1、第 2 パラメータは 2 のように指定します。

x
パラメータの値を指定します。

setIntervalYM()

パラメータを IntervalYM 値に設定します。

構文

```
void setIntervalYM(unsigned int paramIndex,
   const IntervalYM &x);
```

パラメータ

paramIndex
第 1 パラメータは 1、第 2 パラメータは 2 のように指定します。

x
パラメータの値を指定します。

setMaxIterations()

DML 文に対して行われる起動の最大数を設定します。プリコンパイルされた SQL 文にパラメータが設定される前にコールする必要があります。反復数を大きくすると、1 回のラウンドトリップでサーバーに送信されるパラメータ数が多くなります。ただし、数を大きくすることによって、すべてのパラメータに対して多くのメモリーが確保されます。この設定は、単に最大制限です。実質的な反復数は、実行される addIterations() の数に依存します。

構文

```
void setMaxIterations(unsigned int maxIterations);
```

パラメータ

maxIterations
この文に対して許可する最大の反復回数を指定します。

setMaxParamSize()

このメソッドは、指定されたパラメータに関して受渡しされるデータの最大量を設定します。文字とバイナリ・データに対してのみ適用されます。最大量を超過すると、超過したデータは廃棄されます。このメソッドは、LONG 列を取り扱う場合に大変有効です。文字列または Bytes データ型に読込みまたは書込みを行って、LONG 列の切捨てを行う際に使用できます。

setString メソッドまたは setBytes メソッドが、PL/SQL プロシージャの IN/OUT パラメータに値をバインドするためにコールされていて、OUT の値のサイズが IN の値のサイズを超過することが予想される場合は、setMaxParamSize をコールします。

構文

```
void setMaxParamSize(unsigned int paramIndex,
   unsigned int maxSize);
```

パラメータ

paramIndex
第 1 パラメータは 1、第 2 パラメータは 2 のように指定します。

maxSize
新規の最大パラメータ・サイズ制限（> 0（ゼロ））を指定します。

setNull()

パラメータを SQL の NULL に設定します。パラメータの SQL 型を指定する必要があります。

構文

```
void setNull(unsigned int paramIndex,
   Type type);
```

パラメータ

paramIndex
第 1 パラメータは 1、第 2 パラメータは 2 のように指定します。

type
型に定義されている SQL 型コードを指定します。

setNumber()

パラメータを Number 値に設定します。

構文

```
void setNumber(unsigned int paramIndex,
   const Number &x);
```

パラメータ

paramIndex
第 1 パラメータは 1、第 2 パラメータは 2 のように指定します。

x
パラメータの値を指定します。

setObject()

オブジェクトを使用してパラメータの値を設定します。整数値に対して C++ 言語に相当するオブジェクトを使用します。OCCI 仕様によって、C++ の Object 型から SQL 型への標準マッピングが指定されます。指定されたパラメータの C++ オブジェクトは、データベースに送信される前に対応する SQL 型に変換されます。

構文

```
void setObject(unsigned int paramIndex,
   PObject * x);
```

パラメータ

paramIndex
第 1 パラメータは 1、第 2 パラメータは 2 のように指定します。

x
入力パラメータの値が含まれているオブジェクトを指定します。

setPrefetchMemorySize()

サーバーへの各ラウンドトリップでフェッチしたデータを格納するために、OCCI によって繰り返し使用されるメモリー・サイズを設定します。0（ゼロ）の値は、ラウンドトリップ中にフェッチしたデータの量が FetchRowCount パラメータによって制限されることを意味します。両方のパラメータが 0（ゼロ）でない場合、2 つの内の量の少ないほうが使用されます。

構文

```
void setPrefetchMemorySize(unsigned int bytes);
```

パラメータ

bytes
サーバーへの各ラウンドトリップ中にフェッチされたデータの格納に使用するバイト数を指定します。

setPrefetchRowCount()

サーバーへの各ラウンドトリップで、OCCI によって繰り返しフェッチされる行数を設定します。0（ゼロ）の値は、ラウンドトリップ中にフェッチしたデータの量が FetchMemorySize パラメータによって制限されることを意味します。両方のパラメータが 0（ゼロ）でない場合、2 つの内の量の少ないほうが使用されます。両方のパラメータが 0（ゼロ）の場合、行カウントが内部的に 1 行にデフォルト設定され、これが、getFetchRowCount メソッドから戻る値となります。

構文

```
void setPrefetchRowCount(unsigned int rowCount);
```

パラメータ

rowCount
サーバーへの各ラウンドトリップでフェッチする行数を指定します。

setRef()

パラメータを RefAny 値に設定します。

構文

```
void setRef(unsigned int paramIndex,
   RefAny &x);
```

パラメータ

paramIndex
第 1 パラメータは 1、第 2 パラメータは 2 のように指定します。

x
パラメータの値を指定します。

setRowid()

バインドの位置に対して ROWID バイト配列を設定します。

構文

```
void setRowid(unsigned int paramIndex,
   const Bytes &x);
```

パラメータ

paramIndex
第 1 パラメータは 1、第 2 パラメータは 2 のように指定します。

x
パラメータの値を指定します。

setSQL()

このコールによって、新規 SQL 文字列を Statement オブジェクトに関連付けることができます。前の SQL 文に関連付けられているリソースは解放されます。特に、前に取得した結果セットは無効になります。Statement の作成時に、空の SQL 文字列 "" が使用された場合は、実行前に、適切な SQL 文字列を使用して setSQL メソッドをコールする必要があります。

構文

```
void setSQL(const string &sql);
```

パラメータ

sql
SQL 文を指定します。

setString()

パラメータを文字列値に設定します。

構文

```
void setString(unsigned int paramIndex,
   const string &x);
```

パラメータ

paramIndex
第 1 パラメータは 1、第 2 パラメータは 2 のように指定します。

x
パラメータの値を指定します。

setTimestamp()

パラメータを Timestamp 値に設定します。

構文

```
void setTimestamp(unsigned int paramIndex,
   const Timestamp &x);
```

パラメータ

paramIndex
第 1 パラメータは 1、第 2 パラメータは 2 のように指定します。

x
パラメータの値を指定します。

setUInt()

パラメータを C++ の unsigned int 型の値に設定します。

構文

```
void setUInt(unsigned int paramIndex,
   unsigned int x);
```

パラメータ

paramIndex
第 1 パラメータは 1、第 2 パラメータは 2 のように指定します。

x
パラメータの値を指定します。

setVector()

このメソッドは、パラメータをベクターに設定します。このメソッドは、型がコレクション型（VARRAY またはネストした表）の場合に使用する必要があります。コレクション内の要素の SQL 型は、ベクターの型との互換性が必要です。たとえば、コレクションが VARCHAR2 の VARRAY の場合は、vector<string> を使用してください。

構文

様々な構文があります。

```
void setVector(Statement *stmt,
   unsigned int paramIndex,
   vector<int> &vect,
   string sqltype);

void setVector(Statement *stmt,
   unsigned int paramIndex,
   vector<unsigned int> &vect,
   string sqltype);

void setVector(Statement *stmt,
   unsigned int paramIndex,
   vector<double> &vect,
   string sqltype);

void setVector(Statement *stmt,
   unsigned int paramIndex,
   vector<float> &vect,
   string sqltype);

void setVector(Statement *stmt,
   unsigned int paramIndex,
   vector<string> &vect,
   string sqltype);
```

```
void setVector(Statement *stmt,
   unsigned int paramIndex,
   vector<RefAny> &vect,
   string sqltype);

void setVector(Statement *stmt,
   unsigned int paramIndex,
   vector<Blob> &vect,
   string sqltype);

void setVector(Statement *stmt,
   unsigned int paramIndex,
   vector<Clob> &vect,
   string sqltype);

void setVector(Statement *stmt,
   unsigned int paramIndex,
   vector<Bfile> &vect,
   string sqltype);

void setVector(Statement *stmt,
   unsigned int paramIndex,
   vector<Timestamp> &vect,
   string sqltype);

void setVector(Statement *stmt,
   unsigned int paramIndex,
   vector<IntervalDS> &vect,
   string sqltype);

void setVector(Statement *stmt,
   unsigned int paramIndex,
   vector<IntervalYM> &vect,
   string sqltype);

void setVector(Statement *stmt,
   unsigned int paramIndex,
   vector<Date> &vect,
   string sqltype);

void setVector(Statement *stmt,
   unsigned int paramIndex,
   vector<Number> &vect,
   string sqltype);
```

```
template <class T>
void setVector( Statement *stmt, unsigned int paramIndex,
    const OCCI_STD_NAMESPACE::vector< T > &vect,
    const OCCI_STD_  NAMESPACE::string &sqltype) ;

template <class T>
void setVector( Statement *stmt, unsigned int paramIndex,
    const OCCI_STD_NAMESPACE::vector<T* > &vect,
    const OCCI_STD_NAMESPACE::string &sqltype) ;
```

最後の 2 つの構文は同一です。前者は関数テンプレートの一部の順序付けがサポートされているプラットフォームで使用し、後者はこれがサポートされていないプラットフォームで使用します。

次の構文も同様です。

```
void setVector(Statement *stmt,
   unsigned int paramIndex,
   vector<Ref<T>> &vect,
   string sqltype);
```

この構文は、関数テンプレートの一部の順序付けがサポートされているプラットフォームでのみ使用できます。この関数は今後使用できなくなります。かわりに `setVectorOfRefs()` を使用できます。

パラメータ

stmt
パラメータを設定する文を指定します。

paramIndex
パラメータ索引を、第 1 パラメータは `1`、第 2 パラメータは `2` のように指定します。

vect
設定するベクターを指定します。

sqltype
パラメータ / 列の SQL 型を指定します。

たとえば、`CREATE TYPE num_coll AS VARRAY OF NUMBER` の場合を考えてみます。また、列 / パラメータの型は `num_coll` とします。この場合、SQL 型は `num_coll` になります。

setVectorOfRefs()

このメソッドは、パラメータをベクターに設定します。このメソッドは、型が REF のコレクション型（VARRAY またはネストした表）の場合に使用する必要があります。

構文

```
template  <class T>
  void setVectorOfRefs(Statement *stmt, unsigned int paramIndex,
  const OCCI_STD_NAMESPACE::vector<Ref<T> > &vect,
  const OCCI_STD_NAMESPACE::string &sqltype) ;
```

パラメータ

stmt
パラメータを設定する文を指定します。

paramIndex
パラメータ索引を、第 1 パラメータは 1、第 2 パラメータは 2 のように指定します。

vect
設定するベクターを指定します。

sqltype
パラメータ / 列の SQL 型を指定します。

Ref<T> に対しては、特化した関数の setVector ではなく、setVectorOfRefs の使用をお薦めします。

status()

文の現在のステータスを戻します。書き込む（または読み込む）ストリーム・データがある場合に役に立ちます。getCurrentStreamParam や getCurrentIteration などの他のメソッドをコールすると、書き込む必要があるストリーム・パラメータおよび反復実行や配列実行の現在の反復数を確認できます。

この status メソッドは、実行のステータスを調査するために繰り返してコールできます。次のいずれかを戻します。

- RESULT_SET_AVAILABLE − getResultSet() をコールします。
- UPDATE_COUNT_AVAILABLE − getUpdateCount() をコールします。
- NEEDS_STREAM_DATA − getCurrentStream()、getCurrentIteration() および書込みストリームをコールします。

- STREAM_DATA_AVAILABLE － getCurrentStream()、getCurrentIteration() および読込みストリームをコールします。
- PREPARED
- UNPREPARED

構文

```
Status status() const;
```

Stream クラス

Stream を使用してストリーム・データ（通常は LONG）を読込みまたは書込みします。

- 読込み可能なストリームは、結果セットからストリーム・データを取得するため、またはストアド・プロシージャ・コールから OUT バインド変数を取得するために使用されます。読込み可能なストリームは、データの最後に達するまで完全に読み込む必要があります。そうでない場合は、クローズして、不要なデータを廃棄する必要があります。
- 書込み可能なストリームは、ストリーム・データ（通常は LONG）を、コール可能文も含めて、パラメータ化された文に提供するために使用されます。

```
Stream()
enum Status
   {
       READY_FOR_READ,
       READY_FOR_WRITE,
       INACTIVE
   };
```

Stream メソッドの概要

表 8-21 Stream メソッド

メソッド	概要
「readBuffer()」（8-210 ページ）	ストリームを読み込み、Stream オブジェクトから読み込んだデータ量を戻します。
「readLastBuffer()」（8-210 ページ）	ストリームから最後のバッファを読み込みます。
「writeBuffer()」（8-211 ページ）	バッファからストリームにデータを書き込みます。
「writeLastBuffer()」（8-211 ページ）	バッファからストリームに最後のデータを書き込みます。
「status()」（8-212 ページ）	ストリームの現在のステータスを戻します。

readBuffer()

Stream からデータを読み込みます。size パラメータによって、読み込むバイト文字の最大数が指定されます。Stream オブジェクトから読み込んだデータ量を戻します。-1 は、ストリームのデータの終了を意味します。

構文

```
int readBuffer(char *buffer,
   unsigned int size);
```

パラメータ

buffer
データ・バッファのポインタを指定します。ポインタは、コール元で割当ておよび解放する必要があります。

size
バッファのサイズを指定します。

readLastBuffer()

このメソッドは、Stream から最後のバッファを読み込みます。読み込まないデータを廃棄するためにコールすることもできます。

size パラメータによって、読み込むバイト文字の最大数が指定されます。Stream オブジェクトから読み込んだデータ量を戻します。-1 は、ストリームのデータの終了を意味します。

構文

```
int readLastBuffer(char *buffer,
   unsigned int size);
```

パラメータ

buffer
データ・バッファのポインタを指定します。ポインタは、コール元で割当ておよび解放する必要があります。

size
読み込む最大バイト数を指定します。

writeBuffer()

バッファからストリームにデータを書き込みます。書き込まれるデータ量は、サイズによって決まります。

構文

```
void writeBuffer(char *buffer,
   unsigned int size);
```

パラメータ

buffer
データ・バッファへのポインタを指定します。

size
バッファ内の文字数を指定します。

writeLastBuffer()

このメソッドは、バッファからストリームに最後のデータを書き込みます。データの最後のチャンクを書き込むためにコールすることもできます。

書き込まれるデータ量は、サイズによって決まります。

構文

```
void writeLastBuffer(char *buffer,
   unsigned int size);
```

パラメータ

buffer
データ・バッファへのポインタを指定します。

size
書き込むバイト数を指定します。

status()

ストリームの現在のステータスを戻します。戻り値は、次のいずれかです。

- READY_FOR_READ
- READY_FOR_WRITE
- INACTIVE

構文

```
Status status() const;
```

Timestamp クラス

このクラスは、SQL92 の TIMESTAMP 型と TIMESTAMPTZ 型に準拠しています。

OCCI では、Timestamp クラスに対応する SQL データ型は、TIMESTAMP WITH TIME ZONE 型であることを前提にしています。

```
Timestamp(const Environment *env,
   int year = 1,
   unsigned int month = 1,
   unsigned int day = 1,
   unsigned int hour = 0,
   unsigned int min = 0,
   unsigned int sec = 0,
   unsigned int fs = 0,
   int tzhour = 0
   int tzmin=0
```

パラメータ値の範囲は次のとおりです。範囲外の場合は SQLException が発生します。

year － -4713 ～ 9999

month － 1 ～ 12

day － 1 ～ 31

hour － 0 ～ 23

min － 0 ～ 59

sec － 0 ～ 61

tzhour － -12 ～ 14

tzmin － -59 ～ 59

次のコンストラクタは、NULL の Timestamp オブジェクトを戻します。NULL のタイムスタンプは、fromText メソッドを割り当てるか、コールすることによって初期化できます。NULL の Timestamp オブジェクトでコールできるメソッドは、setNull、isNull および operator=() です。

```
Timestamp();
```

次のコンストラクタでは src から Timestamp オブジェクトが作成されます。

```
Timestamp(const Timestamp &src);
```

次のコード例では、デフォルト・コンストラクタによる NULL 値の作成、タイムスタンプに NULL 以外の値を割り当て、それに基づいて操作を実行する方法が示されています。

```
Environment *env = Environment::createEnvironment();

//create a null timestamp
Timestamp ts;
if(ts.isnull())
   cout << "\n ts is Null";

//assign a non null value to ts
Timestamp notNullTs(env, 2000, 8, 17, 12, 0, 0, 0, 5, 30);
ts = notNullTs;

//now all operations are valid on ts...
int yr;
unsigned int mth, day;
ts.getDate(yr, mth, day);
```

次のコード例では、NULL のタイムスタンプを初期化するために fromText メソッドを使用する方法が示されています。

```
Environment *env = Environment::createEnvironment();

Timestamp ts1;
ts1.fromText("01:16:17.12 04/03/1825", "hh:mi:ssxff dd/mm/yyyy", "", env);
```

次のコードは、結果セットからタイムスタンプ列を取得してタイムスタンプが NULL かどうかをチェックし、文字列フォーマットでタイムスタンプの値を取得して 2 つのタイムスタンプの差異を判断する方法を示した例です。

```
Timestamp reft(env, 2001, 1, 1);
ResultSet *rs=stmt->executeQuery("select order_date from orders
                                  where customer_id=1");
rs->next();

//retrieve the timestamp column from result set
Timestamp ts=rs->getTimestamp(1);

//check timestamp for null
if(!ts.isNull())
{
   //get the timestamp value in string format
   string tsstr=ts.toText("dd/mm/yyyy hh:mi:ss [tzh:tzm]",0);

   if(reft<ts //compare timestamps
   {
      IntervalDS ds=reft.subDS(ts); //get difference between timestamps
   }
}
```

Timestamp メソッドの概要

表 8-22 Timestamp メソッド

メソッド	概要
「fromText()」（8-216 ページ）	文字列で指定されている値からタイムスタンプを設定します。
「getDate()」（8-217 ページ）	Timestamp オブジェクトから日付を取得します。
「getTime()」（8-217 ページ）	TimeStamp オブジェクトから時刻を取得します。
「getTimeZoneOffset()」（8-218 ページ）	タイム・ゾーンの時および分のオフセット値を戻します。
「intervalAdd()」（8-218 ページ）	値（タイムスタンプの値 + 時間隔）が設定された Timestamp オブジェクトを戻します。
「intervalSub()」（8-219 ページ）	値（タイムスタンプの値 - 時間隔）が設定された Timestamp オブジェクトを戻します。
「isNull()」（8-219 ページ）	Timestamp が NULL かどうかをチェックします。
「operator=()」（8-219 ページ）	単純割当てを行います。
「operator==()」（8-220 ページ）	a と b が等しいかどうかをチェックします。
「operator!=()」（8-220 ページ）	a と b が等しくないかどうかをチェックします。
「operator>()」（8-220 ページ）	a が b を超えているかどうかをチェックします。
「operator>=()」（8-221 ページ）	a が b 以上かどうかをチェックします。
「operator<()」（8-221 ページ）	a が b 未満かどうかをチェックします。
「operator<=()」（8-222 ページ）	a が b 以下かどうかをチェックします。
「setDate()」（8-222 ページ）	このタイムスタンプの年、月、日のコンポーネントを設定します。
「setNull()」（8-223 ページ）	Timestamp の値を NULL に設定します。
「setTime()」（8-223 ページ）	このタイムスタンプの日、時、分、秒および小数秒の要素を設定します。
「setTimeZoneOffset()」（8-224 ページ）	タイム・ゾーンの時および分のオフセットを設定します。
「subDS()」（8-224 ページ）	タイムスタンプの値 -val を示す IntervalDS を戻します。

表 8-22 Timestamp メソッド（続き）

メソッド	概要
「subYM()」（8-224 ページ）	タイムスタンプの値 -val を示す IntervalYM を戻します。
「toText()」（8-225 ページ）	

fromText()

このメソッドは、文字列で指定されている値からタイムスタンプを設定します。文字列は指定された書式であることが必要です。*nlsParam* が指定されている場合は、この指定によって、変換に使用される nls パラメータが決まります。*nlsParam* が指定されていない場合、nls パラメータは指定された環境から選択されます。環境が指定されていない場合、nls パラメータはインスタンスに関連付けられている環境から選択されます（関連付けられている場合）。

構文

```
void fromText(const string &timestmpStr,
   const string &fmt,
   const string &nlsParam = "",
   const Environment *env = NULL);
```

パラメータ

timestmpStr
入力文字列を指定します。

fm
書式文字列を指定します。

nlsParam
言語名を指定します。

現在サポートされているのは、イギリス英語とアメリカ英語です。

env
nls パラメータが使用される環境を指定します。

getDate()

Timestamp の年、月および日の値を戻します。

構文

```
void getDate(int &year,
   unsigned int &month,
   unsigned int &day) const;
```

パラメータ

year
年のコンポーネントを指定します。

month
月のコンポーネントを指定します。

day
日のコンポーネントを指定します。

getTime()

時、分、秒および小数秒（fs）のコンポーネントを戻します。

構文

```
void getTime(unsigned int &hour,
   unsigned int &minute,
   unsigned int &second,
   unsigned int &fs) const;
```

パラメータ

hour
時のコンポーネントを指定します。

minute
分のコンポーネントを指定します。

second
秒のコンポーネントを指定します。

fs
小数秒のコンポーネントを指定します。

getTimeZoneOffset()

タイム・ゾーン・オフセットを時および分で戻します。

構文

```
void getTimeZoneOffset(int &hour,
   int &minute) const;
```

パラメータ

hour
タイム・ゾーンの時を指定します。

minute
タイム・ゾーンの分を指定します。

intervalAdd()

タイムスタンプに時間隔を加算します。

構文
様々な構文があります。

```
Timestamp intervalAdd(const IntervalDS& val) const;

Timestamp intervalAdd(const IntervalYM& val) const;
```

パラメータ

val
加算する時間隔を指定します。

intervalSub()

タイムスタンプから時間隔を減算し、その結果をタイムスタンプで戻します。

Timestamp をタイムスタンプの値 - val の値で戻します

構文
様々な構文があります。

```
Timestamp intervalSub(const IntervalDS& val) const;

Timestamp intervalSub(const IntervalYM& val) const;
```

パラメータ

val
減算する時間隔を指定します。

isNull()

Timestamp が NULL の場合は true を、それ以外の場合は false を戻します。

構文

```
bool isNull() const;
```

operator=()

指定された Timestamp オブジェクトをこのオブジェクトに割り当てます。

構文

```
Timestamp & operator=(const Timestamp &src);
```

パラメータ

src
割り当てる値を指定します。

operator==()

このメソッドは、指定した 2 つのタイムスタンプを比較します。2 つのタイムスタンプが等しい場合、つまり a が b と同じ場合は true を、それ以外の場合は false を戻します。a または b が NULL の場合は、false を戻します。

構文

```
bool operator==(const Timestamp &a,
   const Timestamp &b);
```

パラメータ

a、b
比較するタイムスタンプを指定します。

operator!=()

このメソッドは、指定した 2 つのタイムスタンプを比較します。2 つのタイムスタンプが等しくない場合は true を、それ以外の場合は false を戻します。いずれかのタイムスタンプが NULL の場合は false を戻します。

構文

```
bool operator!=(const Timestamp &a,
   const Timestamp &b);
```

パラメータ

a、b
比較するタイムスタンプを指定します。

operator>()

a が b より後の場合は true を、それ以外の場合は false を戻します。a または b が NULL の場合は、false を戻します。

構文

```
bool operator>(const Timestamp &a,
   const Timestamp &b);
```

パラメータ

a

a、b
比較するタイムスタンプを指定します。

operator>=()

このメソッドは、指定した 2 つのタイムスタンプを比較します。第 1 タイムスタンプが第 2 タイムスタンプ以上の場合は `true` を、それ以外の場合は `false` を戻します。いずれかのタイムスタンプが NULL の場合は `false` を戻します。

構文

```
bool operator>=(const Timestamp &a,
   const Timestamp &b);
```

パラメータ

a、b
比較するタイムスタンプを指定します。

operator<()

`a` が `b` より前の場合は `true` を、それ以外の場合は `false` を戻します。`a` または `b` が NULL の場合は、`false` を戻します。

構文

```
bool operator<(const Timestamp &a,
   const Timestamp &b);
```

パラメータ

a、b
比較するタイムスタンプを指定します。

operator<=()

このメソッドは、指定した 2 つのタイムスタンプを比較します。第 1 タイムスタンプが第 2 タイムスタンプ以下の場合は `true` を、それ以外の場合は `false` を戻します。いずれかのタイムスタンプが NULL の場合は `false` を戻します。

構文

```
bool operator<=(const Timestamp &a,
   const Timestamp &b);
```

パラメータ

a、b
比較するタイムスタンプを指定します。

setDate()

このタイムスタンプの年、月、日の要素を設定します。

構文

```
void setDate(int year,
   unsigned int month,
   unsigned int day);
```

パラメータ

year
設定する年のコンポーネントを指定します。

有効な値は、-4713 ～ 9999 です。

month
設定する月のコンポーネントを指定します。

有効な値は、1 ～ 12 です。

day
設定する日のコンポーネントを指定します。

有効な値は、1 ～ 31 です。

setNull()

タイムスタンプを NULL に設定します。

構文

```
void setNull();
```

setTime()

このタイムスタンプの日、時、分、秒および小数秒の要素を設定します。

構文

```
void setTime(unsigned int hour,
   unsigned int minute,
   unsigned int second,
   unsigned int fs);
```

パラメータ

hour
時のコンポーネントを指定します。

有効な値は、0 ～ 23 です。

minute
分のコンポーネントを指定します。

有効な値は、0 ～ 59 です。

second
秒のコンポーネントを指定します。

有効な値は、0 ～ 59 です。

fs
小数秒のコンポーネントを指定します。

setTimeZoneOffset()

タイム・ゾーンの時および分のオフセットを設定します。

構文

```
void setTimeZoneOffset(int hour,
   int minute);
```

パラメータ

hour
タイム・ゾーンの時を指定します。

有効な値は、-12 ～ 14 です。

minute
タイム・ゾーンの分を指定します。

有効な値は、-59 ～ 59 です。

subDS()

このタイムスタンプと指定されたタイムスタンプとの差を計算し、その差を IntervalDS で戻します。

構文

```
IntervalDS subDS(const Timestamp& val) const;
```

パラメータ

val
減算するタイムスタンプを指定します。

subYM()

2 つのタイムスタンプの値の差を計算し、その差を IntervalYM で戻します。

構文

```
IntervalYM subYM(const Timestamp& val) const;
```

パラメータ

val
減算するタイムスタンプを指定します。

toText()

タイムスタンプを表す文字列を、指定された書式で戻します。nlsParam が指定されている場合は、この指定によって、変換に使用される nls パラメータが決まります。nlsParam が指定されていない場合は、nls パラメータがインスタンスに関連付けられている環境から選択されます（関連付けられている場合）。

構文

```
string toText(const string &fmt,
   unsigned int fsprec,
   const string &nlsParam = "") const;
```

パラメータ

fmt
書式文字列を指定します。

fsprec
小数秒のコンポーネントで必要な精度を指定します。

nlsParam
現在サポートされているのは、イギリス英語とアメリカ英語です。

関連項目：

- 書式モデルは、『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』の表 A-1 を参照してください。

第 III 部

付録

第Ⅲ部には、次の付録があります。

- 付録 A「OCCI デモ・プログラム」

A

OCCI デモ・プログラム

Oracle では、OCCI コールの使用方法を示すコード例を提供しています。これらのプログラムはあくまでも例であり、すべてのプラットフォームで実行できるとは限りません。

デモ・プログラム（デモ）は、Oracle のインストールによって使用可能になります。デモ・プログラムの位置、名前および可用性は、プラットフォームによって異なります。UNIX ワークステーションでは、デモ・プログラムは `$ORACLE_HOME/rdbms/demo` ディレクトリにインストールされています。

`$ORACLE_HOME/rdbms/demo` ディレクトリには、デモのみではなく、ユーザーが OCCI アプリケーションまたは外部プロシージャの作成時にテンプレートとして使用する demo_rdbms.mk ファイルも含まれています。新しい demo_rdbms.mk ファイルの作成は、ユーザーのマクロをリンク・ラインに追加して、demo_rdbms.mk ファイルをカスタマイズすることによって行います。ただし、変更した demo_rdbms.mk ファイルのメンテナンスを単純化するために、demo_rdbms.mk ファイルに指定したマクロを保持する必要があります。

特定のヘッダー・ファイルまたは SQL ファイルがアプリケーションで必要になった場合には、これらのファイルも用意されています。設定やプログラム実行のヒントは、デモ・プログラムの最初に記載されたコメントの情報を参照してください。

これらのデモを実行する前に、ユーザー名とパスワードにそれぞれ `SCOTT` と `TIGER` を使用して、SQL ファイル occidemo.sql を実行する必要があります。

表 A-1 は、重要なデモ・プログラムおよびそのプログラムで示される OCCI 機能のリストです。

表 A-1 OCCI デモ・プログラム

プログラム名	機能
occiblob.cpp	BLOB の読込みおよび書込み方法
occiclob.cpp	CLOB の読込みおよび書込み方法
occicoll.cpp	Nested Table 型の表の列に対する単純な挿入、削除および更新操作の実行方法
occidesc.cpp	表、プロシージャおよびオブジェクトに関するメタデータの取得方法
occidml.cpp	OCCI による表の行に対する挿入、選択、更新および削除操作の実行方法
occiinh.cpp	サブタイプ表の行に対する挿入、選択、更新および削除操作によるオブジェクトの継承
occiobj.cpp	列の 1 つにオブジェクトが含まれた表の行に対する挿入、選択、更新および削除操作の実行方法
occipobj.cpp	永続オブジェクトの挿入、選択および更新操作の実行方法とオブジェクトの確保、確保解除、削除マークの設定およびフラッシュの実行方法
occipool.cpp	OCCI の接続プール・インタフェースの使用方法
occiproc.cpp	バインド・パラメータを指定した PL/SQL プロシージャの呼出し方法
occistre.cpp	OCCI の ResultSet ストリームの使用方法

OCCI デモ・プログラム

ここでは、デモ Make ファイルに加え、OCCI デモ・プログラムの各ファイルをリストします。

demo_rdbms.mk

次のコードは、デモ・プログラムを作成する Make ファイルです。

```
#
# Example for building demo OCI programs:
#
# 1. All OCI demos (including extdemo2, extdemo4 and extdemo5):
#
#    make -f demo_rdbms.mk demos
#
```

```
# 2. A single OCI demo:
#
#    make -f demo_rdbms.mk build EXE=demo OBJS="demo.o ..."
#    e.g. make -f demo_rdbms.mk build EXE=oci02 OBJS=oci02.o
#
# 3. A single OCI demo with static libraries:
#
#    make -f demo_rdbms.mk build_static EXE=demo OBJS="demo.o ..."
#    e.g. make -f demo_rdbms.mk build_static EXE=oci02 OBJS=oci02.o
#
# 4. To regenerate shared library:
#
#    make -f demo_rdbms.mk generate_sharedlib
#
# 5. All OCCI demos
#
#    make -f demo_rdbms.mk occidemos
#
# 6. A single OCCI demo:
#
#    make -f demo_rdbms.mk <demoname>
#    e.g. make -f demo_rdbms.mk occidml
#    OR
#    make -f demo_rdbms.mk buildocci EXE=demoname OBJS="demoname.o ..."
#    e.g. make -f demo_rdbms.mk buildocci EXE=occidml OBJS=occidml.o
#
# 7. A single OCCI demo with static libraries:
#
#    make -f demo_rdbms.mk buildocci_static EXE=demoname OBJS="demoname.o ..."
#    e.g. make -f demo_rdbms.mk buildocci_static EXE=occiblob OBJS=occiblob.o
#
# 8. All OCI Connection Pooling, Session Pooling and Statement Cache demos
#
#    make -f demo_rdbms.mk cpdemos
#
# 9. A single OCI Connection Pooling demo:
#
#    make -f demo_rdbms.mk <demoname>
#    e.g. make -f demo_rdbms.mk ocicp
#    OR
#    make -f demo_rdbms.mk buildcp EXE=demoname OBJS="demoname.o ..."
#    e.g. make -f demo_rdbms.mk buildcp EXE=ocicp OBJS=ocicp.o
#
```

```
# 10. A single OCI Connection Pooling demo with static libraries:
#
#    make -f demo_rdbms.mk buildcp_static EXE=demoname OBJS="demoname.o ..."
#    e.g. make -f demo_rdbms.mk buildcp_static EXE=ocicp OBJS=ocicp.o
#
# 11. A single OCI Session Pooling demo:
#
#    make -f demo_rdbms.mk <demoname>
#    e.g. make -f demo_rdbms.mk ocisp
#    OR
#    make -f demo_rdbms.mk buildcp EXE=demoname OBJS="demoname.o ..."
#    e.g. make -f demo_rdbms.mk buildcp EXE=ocisp OBJS=ocisp.o
#
# 12. A single OCI Session Pooling demo with static libraries:
#
#    make -f demo_rdbms.mk buildcp_static EXE=demoname OBJS="demoname.o ..."
#    e.g. make -f demo_rdbms.mk buildcp_static EXE=ocisp OBJS=ocisp.o
#
# 13. A single OCI Statement Cache demo:
#
#    make -f demo_rdbms.mk <demoname>
#    e.g. make -f demo_rdbms.mk ocisc
#    OR
#    make -f demo_rdbms.mk buildcp EXE=demoname OBJS="demoname.o ..."
#    e.g. make -f demo_rdbms.mk buildcp EXE=ocisc OBJS=ocisc.o
#
# 14. A single OCI Statement Cache demo with static libraries:
#
#    make -f demo_rdbms.mk buildcp_static EXE=demoname OBJS="demoname.o ..."
#    e.g. make -f demo_rdbms.mk buildcp_static EXE=ocisc OBJS=ocisc.o
#
# Example for building demo DIRECT PATH API programs:
#
# 1. All DIRECT PATH API demos:
#
#    make -f demo_rdbms.mk demos_dp
#
# 2. A single DIRECT PATH API demo:
#
#    make -f demo_rdbms.mk build_dp EXE=demo OBJS="demo.o ..."
#    e.g. make -f demo_rdbms.mk build_dp EXE=cdemdplp OBJS=cdemdplp.o
#
#
```

```
# Example for building external procedures demo programs:
#
# 1. All external procedure demos:
#
# 2. A single external procedure demo whose 3GL routines do not use the
#    "with context" argument:
#
#    make -f demo_rdbms.mk extproc_no_context SHARED_LIBNAME=libname
#                                             OBJS="demo.o ..."
#    e.g. make -f demo_rdbms.mk extproc_no_context SHARED_LIBNAME=epdemo.so
#                                             OBJS="epdemo1.o epdemo2.o"
#
# 3. A single external procedure demo where one or more 3GL routines use the
#    "with context" argument:
#
#    make -f demo_rdbms.mk extproc_with_context SHARED_LIBNAME=libname
#                                             OBJS="demo.o ..."
#    e.g. make -f demo_rdbms.mk extproc_with_context SHARED_LIBNAME=epdemo.so
#                                             OBJS="epdemo1.o epdemo2.o"
#    e.g. make -f demo_rdbms.mk extproc_with_context
#                       SHARED_LIBNAME=extdemo2.so OBJS="extdemo2.o"
#    e.g. or For EXTDEMO2 DEMO ONLY: make -f demo_rdbms.mk demos
#
# 4. To link C++ demos:
#
#    make -f demo_rdbms.mk c++demos
#
#
# NOTE: 1. ORACLE_HOME must be either:
#                  . set in the user's environment
#                  . passed in on the command line
#                  . defined in a modified version of this makefile
#
#       2. If the target platform support shared libraries (e.g. Solaris)
#          look in the platform specific documentation for information
#          about environment variables that need to be properly
#          defined (e.g. LD_LIBRARY_PATH in Solaris).
#

include $(ORACLE_HOME)/rdbms/lib/env_rdbms.mk

# flag for linking with non-deferred option (default is deferred mode)
NONDEFER=false

DEMO_DIR=$(ORACLE_HOME)/rdbms/demo
DEMO_MAKEFILE = $(DEMO_DIR)/demo_rdbms.mk
```

```
DEMOS = cdemo1 cdemo2 cdemo3 cdemo4 cdemo5 cdemo81 cdemo82 \
        cdemobj cdemolb cdemodsc cdemocor cdemolb2 cdemolbs \
        cdemodr1 cdemodr2 cdemodr3 cdemodsa obndra \
        cdemoext cdemothr cdemofil cdemofor \
        oci02 oci03 oci04 oci05 oci06 oci07 oci08 oci09 oci10 \
        oci11 oci12 oci13 oci14 oci15 oci16 oci17 oci18 oci19 oci20 \
        oci21 oci22 oci23 oci24 oci25 readpipe cdemosyev \
        ociaqdemo00 ociaqdemo01 ociaqdemo02 cdemoucb nchdemo1

DEMOS_DP = cdemdpco cdemdpin cdemdpit cdemdplp cdemdpno cdemdpro cdemdpss

C++DEMOS = cdemo6
OCCIDEMOS = occiblob occiclob occicoll occidesc occidml occipool occiproc \
            occistre
OCCIOTTDEMOS = occiobj occiinh occipobj
# OTT Markers Support
OCCIOTTDEMOSWITHMARKER = mdemo1
OTTUSR = scott
OTTPWD = tiger
CPDEMOS = ocicp ocicpproxy ocisp ocisc

.SUFFIXES: .o .cob .for .c .pc .cc .cpp

demos: $(DEMOS) extdemo2 extdemo4 extdemo5

demos_dp: $(DEMOS_DP)

generate_sharedlib:
        $(SILENT)$(ECHO) "Building client shared library ..."
        $(SILENT)$(ECHO) "Calling script $$ORACLE_HOME/bin/genclntsh ..."
        $(GENCLNTSH)
        $(SILENT)$(ECHO) "The library is $$ORACLE_HOME/lib/libclntsh.so... DONE"

BUILD=build
$(DEMOS):
        $(MAKE) -f $(DEMO_MAKEFILE) $(BUILD) EXE=$@ OBJS=$@.o

$(DEMOS_DP): cdemodp.c cdemodp0.h cdemodp.h
        $(MAKE) -f $(DEMO_MAKEFILE) build_dp EXE=$@ OBJS=$@.o

c++demos: $(C++DEMOS)

$(C++DEMOS):
        $(MAKE) -f $(DEMO_MAKEFILE) buildc++ EXE=$@ OBJS=$@.o
```

```
buildc++: $(OBJS)
        $(MAKECPLPLDEMO)

occidemos:      $(OCCIDEMOS) $(OCCIOTTDEMOS) $(OCCIOTTDEMOSWITHMARKER)

$(OCCIDEMOS):
        $(MAKE) -f $(DEMO_MAKEFILE) buildocci EXE=$@ OBJS=$@.o

$(OCCIOTTDEMOS):
        $(MAKE) -f $(DEMO_MAKEFILE) ott OTTFILE=$@
        $(MAKE) -f $(DEMO_MAKEFILE) buildocci EXE=$@ OBJS="$@.o $@o.o $@m.o"

# OTT Markers Support
$(OCCIOTTDEMOSWITHMARKER):
        $(MAKE) -f $(DEMO_MAKEFILE) ott_mrkr OTTFILE=$@
        $(MAKE) -f $(DEMO_MAKEFILE) buildocci EXE=$@ OBJS="$@.o $@o.o $@m.o"

buildocci: $(OBJS)
        $(MAKEOCCISHAREDDEMO)

buildocci_static: $(OBJS)
        $(MAKEOCCISTATICDEMO)

ott:
        $(ORACLE_HOME)/bin/ott \
                userid=$(OTTUSR)/$(OTTPWD) \
                intype=$(OTTFILE).typ \
                outtype=$(OTTFILE)out.type \
                code=cpp \
                hfile=$(OTTFILE).h \
                cppfile=$(OTTFILE)o.cpp \
                attraccess=private

# OTT Markers Suppport
ott_mrkr:
        $(ORACLE_HOME)/bin/ott \
                userid=$(OTTUSR)/$(OTTPWD) \
                intype=$(OTTFILE).typ \
                outtype=$(OTTFILE)out.type \
                code=cpp \
                hfile=$(OTTFILE).h \
                cppfile=$(OTTFILE)o.cpp \
                use_marker=true
```

```
cpdemos:        $(CPDEMOS)
$(CPDEMOS):
        $(MAKE) -f $(DEMO_MAKEFILE) buildcp EXE=$@ OBJS=$@.o
buildcp: $(OBJS)
        $(MAKECPSHAREDDEMO)
buildcp_static: $(OBJS)
        $(MAKECPSTATICDEMO)

# Pro*C rules
# SQL Precompiler macros

pc1:
        $(PCC2C)

.pc.c:
        $(MAKE) -f $(DEMO_MAKEFILE) PCCSRC=$* I_SYM=include= pc1

.pc.o:
        $(MAKE) -f $(DEMO_MAKEFILE) PCCSRC=$* I_SYM=include= pc1
        $(PCCC2O)

.cc.o:
        $(CCC2O)

.cpp.o:
        $(CCC2O)

build: $(LIBCLNTSH) $(OBJS)
        $(BUILDEXE)

extdemo2:
        $(MAKE) -f $(DEMO_MAKEFILE) extproc_with_context SHARED_LIBNAME=extdemo2.so
OBJS="extdemo2.o"

extdemo4:
        $(MAKE) -f $(DEMO_MAKEFILE) extproc_with_context SHARED_LIBNAME=extdemo4.so
OBJS="extdemo4.o"

extdemo5:
        $(MAKE) -f $(DEMO_MAKEFILE) extproc_with_context SHARED_LIBNAME=extdemo5.so
OBJS="extdemo5.o"

.c.o:
        $(C2O)
```

```
build_dp: $(LIBCLNTSH) $(OBJS) cdemodp.o
        $(DPTARGET)

build_static: $(OBJS)
        $(O2STATIC)

# extproc_no_context and extproc_with_context are the current names of these
# targets.  The old names, extproc_nocallback and extproc_callback are
# preserved for backward compatibility.

extproc_no_context extproc_nocallback: $(OBJS)
        $(BUILDLIB_NO_CONTEXT)

extproc_with_context extproc_callback: $(OBJS) $(LIBCLNTSH)
        $(BUILDLIB_WITH_CONTEXT)

clean:
        $(RM) -f $(DEMOS) $(CPDEMOS) extdemo2 extdemo4 extdemo5 *.o *.so
        $(RM) -f $(OCCIDEMOS)  $(OCCIOTTDEMOS) occi*m.cpp occi*o.cpp occi*.typ
occiobj*.h occiinh*.h occipobj*.h
        $(RM) -f $(OCCIOTTDEMOSWITHMARKER) mdemo*m.cpp mdemo*o.cpp mdemo*.typ
mdemo*.h
        $(RM) -f $(DEMOS_DP)
```

occiblob.cpp

次のコード例は、BLOB の読込みおよび書込み方法を示しています。

```
#include <iostream.h>
#include <occi.h>
using namespace oracle::occi;
using namespace std;

/**
 * The demo sample has starts from startDemo method.  This method is called
 * by main.  startDemo calls other methods, the supporting methods for
 * startDemo are,
 * insertRows    - insert the rows into the table
 * deleteRows    - delete the rows inserted
 * insertBlob    - Inserts a blob and an empty_blob
 * populateBlob  - populates a given blob
 * dumpBlob      - prints the blob as an integer stream
 */
```

```
class  demoBlob
{
  private:
  string username;
  string password;
  string url;

  void insertRows (Connection *conn)
   throw (SQLException)
  {
    Statement *stmt = conn->createStatement ("INSERT INTO print_media(product_id,ad_
id,ad_composite,ad_sourcetext) VALUES (6666,11001,'10001','SHE')");
    stmt->executeUpdate();
    stmt->setSQL ("INSERT INTO print_media(product_id,ad_id,ad_composite,ad_
sourcetext) VALUES (7777,11001,'1010','HEM')");
    stmt->executeUpdate();
    conn->commit();
    conn->terminateStatement (stmt);

  }

  void deleteRows (Connection *conn)
   throw (SQLException)
  {
    Statement *stmt = conn->createStatement ("DELETE print_media WHERE product_id =
6666 AND ad_id=11001");
    stmt->executeUpdate();
    stmt->setSQL ("DELETE print_media WHERE product_id = 7777 AND ad_id=11001");
    stmt->executeUpdate();
    conn->commit();
    conn->terminateStatement (stmt);

  }

  /**
   * populating the blob;
   */
  void populateBlob (Blob &blob, int size)
   throw (SQLException)
  {
    Stream *outstream = blob.getStream (1,0);
    char *buffer = new char[size];
```

```
  memset (buffer, (char)10, size);
  outstream->writeBuffer (buffer, size);
  char *c = (char *)"";
  outstream->writeLastBuffer (c,0);
  delete (buffer);
  blob.closeStream (outstream);
}

/**
 * printing the blob data as integer stream
 */
void dumpBlob (Blob &blob, int size)
 throw (SQLException)
{
  Stream *instream = blob.getStream (1,0);
  char *buffer = new char[size];
  memset (buffer, NULL, size);

  instream->readBuffer (buffer, size);
  cout << "dumping blob: ";
  for (int i = 0; i < size; ++i)
    cout << (int) buffer[i];
  cout << endl;

  delete (buffer);
  blob.closeStream (instream);
}



/**
 * public methods
 */
public:
demoBlob ()
{
  /**
   * default values of username & password
   */
   username = "SCOTT";
   password = "TIGER";
   url = "";
}
```

```
  void setUsername (string u)
  {
    username = u;
  }

  void setPassword (string p)
  {
    password = p;
  }

  void setUrl (string u)
  {
    url = u;
  }

  void runSample ()
   throw (SQLException)
  {
    Environment *env = Environment::createEnvironment (
    Environment::DEFAULT);
    try
    {
      Connection *conn = env->createConnection (username, password, url);
      Statement *stmt1;
      insertRows (conn);
      /**
       * Reading a populated blob & printing its property.
       */
      string sqlQuery = "SELECT  ad_composite FROM print_media WHERE product_
id=6666";
      Statement *stmt = conn->createStatement (sqlQuery);

      ResultSet *rset = stmt->executeQuery ();
      while (rset->next ())
      {
        Blob blob = rset->getBlob (1);
        cout << "Opening the blob in Read only mode" << endl;
        blob.open (OCCI_LOB_READONLY);
        int blobLength=blob.length ();
        cout << "Length of the blob is: " << blobLength << endl;
        dumpBlob (blob, blobLength);
        blob.close ();
      }
      stmt->closeResultSet (rset);
```

```
      /**
       * Reading a populated blob & printing its property.
       */
      stmt->setSQL ("SELECT ad_composite FROM print_media WHERE product_id =7777 FOR
UPDATE");
      rset = stmt->executeQuery ();
      while (rset->next ())
      {
        Blob blob = rset->getBlob (1);
        cout << "Opening the blob in read write mode" << endl;
        blob.open (OCCI_LOB_READWRITE);
        cout << "Populating the blob" << endl;
        populateBlob (blob, 20);
        int blobLength=blob.length ();
        cout << "Length of the blob is: " << blobLength << endl;
        dumpBlob (blob, blobLength);
        blob.close ();
      }
      stmt->closeResultSet (rset);
      deleteRows (conn);
      conn->terminateStatement (stmt);
      env->terminateConnection (conn);
    }
    catch (SQLException ea)
    {
      cout << ea.what();
    }
    Environment::terminateEnvironment (env);
  }

};//end of class  demoBlob

int main (void)
{
  demoBlob *b = new demoBlob ();
  b->setUsername ("SCOTT");
  b->setPassword ("TIGER");
  b->runSample ();
}
```

occiclob.cpp

次のコード例は、CLOB の読込みおよび書込み方法を示しています。

```
#include <iostream.h>
#include <occi.h>
using namespace oracle::occi;
using namespace std;

/**
 * The demo sample has starts from startDemo method.  This method is called
 * by main.  startDemo calls other methods, the supporting methods for
 * startDemo are,
 * insertRows    -  inserts the rows into the table1
 * deleteRows    -  delete the rows inserted
 * insertClob    - Inserts a clob and an empty_clob
 * populateClob  - populates a given clob
 * dumpClob      - prints the clob as an integer stream
 */

class  demoClob
{
  private:
  string username;
  string password;
  string url;

  void insertRows (Connection *conn)
   throw (SQLException)
  {
    Statement *stmt = conn->createStatement ("INSERT INTO print_media(product_id,ad_
id,ad_composite,ad_sourcetext) VALUES (3333,11001,'10001','SHE')");
    stmt->executeUpdate();
    stmt->setSQL ("INSERT INTO print_media(product_id,ad_id,ad_composite,ad_
sourcetext) VALUES (4444,11001,'1010','HEM')");
    stmt->executeUpdate();
    conn->commit();
    conn->terminateStatement (stmt);

  }

  void deleteRows (Connection *conn)
   throw (SQLException)
  {
    Statement *stmt = conn->createStatement ("DELETE print_media WHERE product_
id=3333 AND ad_id=11001");
    stmt->executeUpdate();
    stmt->setSQL("DELETE print_media WHERE product_id=4444 AND ad_id=11001");
```

```
  stmt->executeUpdate();
  conn->commit();
  conn->terminateStatement (stmt);

}

/**
 * populating the clob;
 */
void populateClob (Clob &clob, int size)
 throw (SQLException)
{
  Stream *outstream = clob.getStream (1,0);
  char *buffer = new char[size];

  memset (buffer,'H', size);
  outstream->writeBuffer (buffer, size);
  char *c = (char *)"";
  outstream->writeLastBuffer (c,0);
  delete (buffer);
  clob.closeStream (outstream);
}

/**
 * printing the clob data as integer stream
 */
void dumpClob (Clob &clob, int size)
 throw (SQLException)
{
  Stream *instream = clob.getStream (1,0);
  char *buffer = new char[size];
  memset (buffer, NULL, size);

  instream->readBuffer (buffer, size);
  cout << "dumping clob: ";
  for (int i = 0; i < size; ++i)
    cout << (char) buffer[i];
  cout << endl;

  delete (buffer);
  clob.closeStream (instream);
}
```

```
  /**
   * public methods
   */
  public:
  demoClob ()
  {
    /**
     * default values of username & password
     */
     username = "SCOTT";
     password = "TIGER";
     url = "";
  }

  void setUsername (string u)
  {
    username = u;
  }

  void setPassword (string p)
  {
    password = p;
  }

  void setUrl (string u)
  {
    url = u;
  }

  void runSample ()
   throw (SQLException)
  {
    Environment *env = Environment::createEnvironment (
    Environment::DEFAULT);
    try
    {
      Connection *conn = env->createConnection (username, password, url);
      Statement *stmt1;
      insertRows (conn);
      /**
       * Reading a populated clob & printing its property.
       */
      string sqlQuery = "SELECT  ad_sourcetext FROM print_media WHERE product_
id=3333";
      Statement *stmt = conn->createStatement (sqlQuery);
```

```
      ResultSet *rset = stmt->executeQuery ();
      while (rset->next ())
      {
        Clob clob = rset->getClob (1);
        cout << "Opening the clob in Read only mode" << endl;
        clob.open (OCCI_LOB_READONLY);
        int clobLength=clob.length ();
        cout << "Length of the clob is: " << clobLength << endl;
        dumpClob (clob, clobLength);
        clob.close ();
      }
      stmt->closeResultSet (rset);

      /**
       * Reading a populated clob & printing its property.
       */
      stmt->setSQL ("SELECT ad_sourcetext FROM print_media WHERE product_id =4444
FOR UPDATE");
      rset = stmt->executeQuery ();
      while (rset->next ())
      {
        Clob clob = rset->getClob (1);
        cout << "Opening the clob in read write mode" << endl;
        clob.open (OCCI_LOB_READWRITE);
        cout << "Populating the clob" << endl;
        populateClob (clob, 20);
        int clobLength=clob.length ();
        cout << "Length of the clob is: " << clobLength << endl;
        dumpClob (clob, clobLength);
        clob.close ();
      }
      stmt->closeResultSet (rset);
      conn->terminateStatement (stmt);
      deleteRows(conn);
      env->terminateConnection (conn);
    }
    catch (SQLException ea)
    {
      cout << ea.what();
    }
    Environment::terminateEnvironment (env);
  }

};//end of class  demoClob
```

```
int main (void)
{
  demoClob *b = new demoClob ();
  b->setUsername ("SCOTT");
  b->setPassword ("TIGER");
  b->runSample ();
}
```

occicoll.cpp

次のコード例は、Nested Table 型の表の列に対する単純な挿入、削除および更新操作の実行方法を示しています。

```
/**
*occicoll - To exhibit simple insert, delete & update operations"
*     " on table having a Nested Table column
*
*Description
*  Create a program which has insert,delete and update on a
*  table having a Nested table column.
*  Perform all these operations using OCCI interface.
*/

#include <iostream.h>
#include <occi.h>
using namespace oracle::occi;
using namespace std;

typedef vector<string> journal;

class occicoll
{
  private:

  Environment *env;
  Connection *conn;
  Statement *stmt;
  string tableName;
  string typeName;

  public:

  occicoll (string user, string passwd, string db)
  {
    env = Environment::createEnvironment (Environment::OBJECT);
    conn = env->createConnection (user, passwd, db);
```

```
    initRows();
  }

  ~occicoll ()
  {
    env->terminateConnection (conn);
    Environment::terminateEnvironment (env);
  }

  void setTableName (string s)
  {
    tableName = s;
  }

  void initRows ()
  {
    try{
    Statement *st1 = conn->createStatement ("DELETE FROM journal_tab");
    st1->executeUpdate ();
    st1->setSQL("INSERT INTO journal_tab (jid, jname) VALUES (22, journal ('NATION',
'TIMES'))");
    st1->executeUpdate ();
    st1->setSQL("INSERT INTO journal_tab (jid, jname) VALUES (33, journal
('CRICKET', 'ALIVE'))");
    st1->executeUpdate ();
    conn->commit();
    conn->terminateStatement (stmt);
    }catch(SQLException ex)
    {
     cout<<ex.what();
    }
  }
  /**
   * Insertion of a row
   */
  void insertRow ()
  {
    int c1 = 11;
    journal c2;

    c2.push_back ("LIFE");
    c2.push_back ("TODAY");
    c2.push_back ("INVESTOR");

    cout << "Inserting row with jid = " << 11 <<
      " and journal_tab (LIFE, TODAY, INVESTOR )" << endl;
    try{
```

```
  stmt = conn->createStatement (
   "INSERT INTO journal_tab (jid, jname) VALUES (:x, :y)");
  stmt->setInt (1, c1);
  setVector (stmt, 2, c2, "JOURNAL");
  stmt->executeUpdate ();
  }catch(SQLException ex)
  {
   cout<<"Exception thrown for insertRow"<<endl;
   cout<<"Error number: "<<  ex.getErrorCode() << endl;
   cout<<ex.getMessage() << endl;
  }
  cout << "Insertion - Successful" << endl;
  conn->terminateStatement (stmt);

}

// Displaying all the rows of the table
void displayAllRows ()
{
  cout << "Displaying all the rows of the table" << endl;
  stmt = conn->createStatement (
    "SELECT jid, jname FROM journal_tab");

  journal c2;

  ResultSet *rs = stmt->executeQuery();
  try{
  while (rs->next())
  {
    cout << "jid: " << rs->getInt(1) << endl;
    cout << "jname: ";
    getVector (rs, 2, c2);
    for (int i = 0; i < c2.size(); ++i)
      cout << c2[i] << " ";
    cout << endl;
  }
 }catch(SQLException ex)
  {
   cout<<"Exception thrown for displayRow"<<endl;
   cout<<"Error number: "<<  ex.getErrorCode() << endl;
   cout<<ex.getMessage() << endl;
  }
  stmt->closeResultSet (rs);
  conn->terminateStatement (stmt);

} // End of displayAllRows()
```

```
// Deleting a row in a nested table
void deleteRow (int c1, string str)
{
  cout << "Deleting a row in a nested table of strings" << endl;
  stmt = conn->createStatement (
    "SELECT jname FROM journal_tab WHERE jid = :x");
  journal c2;
  stmt->setInt (1, c1);

  ResultSet *rs = stmt->executeQuery();
  try{
  if (rs->next())
  {
    getVector (rs, 1, c2);
    c2.erase (find (c2.begin(), c2.end(), str));
  }
  stmt->setSQL ("UPDATE journal_tab SET jname = :x WHERE jid = :y");
  stmt->setInt (2, c1);
  setVector (stmt, 1, c2, "JOURNAL");
  stmt->executeUpdate ();
  }catch(SQLException ex)
  {
   cout<<"Exception thrown for delete row"<<endl;
   cout<<"Error number: "<<  ex.getErrorCode() << endl;
   cout<<ex.getMessage() << endl;
  }

  cout << "Deletion - Successful" << endl;
  conn->commit();
  stmt->closeResultSet (rs);
  conn->terminateStatement (stmt);
} // End of deleteRow (int, string)

// Updating a row of the nested table of strings
void updateRow (int c1, string str)
{
  cout << "Updating a row of the nested table of strings" << endl;
  stmt = conn->createStatement (
    "SELECT jname FROM journal_tab WHERE jid = :x");

  journal c2;

  stmt->setInt (1, c1);
  ResultSet *rs = stmt->executeQuery();
  try{
  if (rs->next())
  {
```

```
      getVector (rs, 1, c2);
      c2[0] = str;
    }
    stmt->setSQL ("UPDATE journal_tab SET jname = :x WHERE jid = :y");
    stmt->setInt (2, c1);
    setVector (stmt, 1, c2, "JOURNAL");
    stmt->executeUpdate ();
    }catch(SQLException ex)
    {
     cout<<"Exception thrown for updateRow"<<endl;
     cout<<"Error number: "<<  ex.getErrorCode() << endl;
     cout<<ex.getMessage() << endl;
    }

    cout << "Updation - Successful" << endl;
    conn->commit();
    stmt->closeResultSet (rs);
    conn->terminateStatement (stmt);
  } // End of UpdateRow (int, string)

};//end of class occicoll

int main (void)
{
  string user = "SCOTT";
  string passwd = "TIGER";
  string db = "";

  try
  {
    cout << "occicoll - Exhibiting simple insert, delete & update operations"
     " on table having a Nested Table column" << endl;
    occicoll *demo = new occicoll (user, passwd, db);

    cout << "Displaying all rows before the operations" << endl;
    demo->displayAllRows ();

    demo->insertRow ();

    demo->deleteRow (11, "TODAY");

    demo->updateRow (33, "New_String");

    cout << "Displaying all rows after all the operations" << endl;
    demo->displayAllRows ();
```

```
    delete (demo);
    cout << "occicoll - done" << endl;
  }catch (SQLException ea)
  {
    cerr << "Error running the demo: " << ea.getMessage () << endl;
  }
}
```

occidesc.cpp

次のコード例は、表、プロシージャおよびオブジェクトに関するメタデータの取得方法を示しています。

```
/**
*  occidesc - Describing the various objects of the database.
*
*  DESCRIPTION :
*    This program describes the objects of the database, like, table, object
*    and procedure.
*
*/

#include <iostream.h>
#include <occi.h>
using namespace oracle::occi;
using namespace std;

class occidesc
{
  private:

  Environment *env;
  Connection *conn;
  public :
  /**
   * Constructor for the occidesc demo program.
   */
  occidesc (string user, string passwd, string db) throw (SQLException)
  {
    env = Environment::createEnvironment (Environment::OBJECT);
    conn = env->createConnection (user, passwd, db);
  }// end of constructor occidesc (string, string, string )
```

```
/**
 * Destructor for the occidesc demo program.
 */
~occidesc () throw (SQLException)
{
  env->terminateConnection (conn);
  Environment::terminateEnvironment (env);
}  // end of ~occidesc ()

// Describing a subtype
void describe_type()
{
  cout << "Describing the object - PERSON_OBJ " << endl;
  MetaData metaData = conn->getMetaData ((char *)"PERSON_OBJ");
  int mdTyp = metaData.getInt(MetaData::ATTR_PTYPE);
  if (mdTyp == MetaData::PTYPE_TYPE)
  {
    cout << "PERSON_OBJ is a type" << endl;
  }
  int typcode = metaData.getInt(MetaData::ATTR_TYPECODE);
  if (typcode == OCCI_TYPECODE_OBJECT)
     cout << "PERSON_OBJ is an object type" << endl;
  else
     cout << "PERSON_OBJ is not an object type" << endl;
  int numtypeattrs = metaData.getInt(MetaData::ATTR_NUM_TYPE_ATTRS);
  cout << "Object has " << numtypeattrs << " attributes" << endl;
  try
  {
    cout << "Object id: " << metaData.getUInt (MetaData::ATTR_OBJ_ID)
      << endl;
  }
  catch (SQLException ex)
  {
    cout << ex.getMessage() << endl;
  }
  cout << "Object Name: " <<
    metaData.getString (MetaData::ATTR_OBJ_NAME) << endl;
  cout << "Schema Name: " <<
    (metaData.getString(MetaData::ATTR_OBJ_SCHEMA)) << endl;
  cout << "Attribute version: " <<
    (metaData.getString(MetaData::ATTR_VERSION)) << endl;
  if (metaData.getBoolean(MetaData::ATTR_IS_INCOMPLETE_TYPE))
    cout << "Incomplete type" << endl;
  else
    cout << "Not Incomplete type" << endl;
  if (metaData.getBoolean(MetaData::ATTR_IS_SYSTEM_TYPE))
    cout << "System type" << endl;
```

```
else
  cout << "Not System type" << endl;
if (metaData.getBoolean(MetaData::ATTR_IS_PREDEFINED_TYPE))
  cout << "Predefined Type" << endl;
else
  cout << "Not Predefined Type" << endl;
if (metaData.getBoolean(MetaData::ATTR_IS_TRANSIENT_TYPE))
  cout << "Transient Type" << endl;
else
  cout << "Not Transient Type" << endl;
if (metaData.getBoolean(MetaData::ATTR_IS_SYSTEM_GENERATED_TYPE))
  cout << "System-generated type" << endl;
else
  cout << "Not System-generated type" << endl;
if (metaData.getBoolean(MetaData::ATTR_HAS_NESTED_TABLE))
  cout << "Has nested table" << endl;
else
  cout << "Does not have nested table" << endl;
if (metaData.getBoolean(MetaData::ATTR_HAS_LOB))
  cout << "Has LOB" << endl;
else
  cout << "Does not have LOB" << endl;
if (metaData.getBoolean(MetaData::ATTR_HAS_FILE))
  cout << "Has BFILE" << endl;
else
  cout << "Does not have BFILE" << endl;
if (metaData.getBoolean(MetaData::ATTR_IS_INVOKER_RIGHTS))
  cout << "Object is Invoker rights" << endl;
else
  cout << "Object is Not Invoker rights" << endl;
RefAny ref = metaData.getRef (MetaData::ATTR_REF_TDO);
MetaData md1 = conn->getMetaData (ref);
vector<MetaData> v1 =
  md1.getVector (MetaData::ATTR_LIST_TYPE_ATTRS);

for (int i = 0; i < v1.size (); ++i)
{
  MetaData md2 = (MetaData)v1[i];
  cout << "Column Name :" <<
    (md2.getString(MetaData::ATTR_NAME)) << endl;
  cout << " Data Type :" <<
    (printType (md2.getInt(MetaData::ATTR_DATA_TYPE))) << endl;
  cout << " Size :" << md2.getInt(MetaData::ATTR_DATA_SIZE) << endl;
  cout << " Precision :" << md2.getInt(MetaData::ATTR_PRECISION) << endl;
  cout << " Scale :" << md2.getInt(MetaData::ATTR_SCALE) << endl << endl;
}
```

```
  cout << "describe_type - done" << endl;
} // end of describe_type()

// Describing a table
void describe_table ()
{
  cout << "Describing the table - AUTHOR_TAB" << endl;
  vector<MetaData> v1;
  MetaData metaData = conn->getMetaData("AUTHOR_TAB");
  cout << "Object name:" <<
    (metaData.getString(MetaData::ATTR_OBJ_NAME)) << endl;
  cout << "Schema:" <<
    (metaData.getString(MetaData::ATTR_OBJ_SCHEMA)) << endl;
  if (metaData.getInt(MetaData::ATTR_PTYPE) ==
    MetaData::PTYPE_TABLE)
  {
    cout << "AUTHOR_TAB is a table" << endl;
  }
  else
    cout << "AUTHOR_TAB is not a table" << endl;
  if (metaData.getBoolean(MetaData::ATTR_PARTITIONED))
    cout << "Table is partitioned" << endl;
  else
    cout << "Table is not partitioned" << endl;
  if (metaData.getBoolean(MetaData::ATTR_IS_TEMPORARY))
    cout << "Table is temporary" << endl;
  else
    cout << "Table is not temporary" << endl;
  if (metaData.getBoolean(MetaData::ATTR_IS_TYPED))
    cout << "Table is typed" << endl;
  else
    cout << "Table is not typed" << endl;
  if (metaData.getBoolean(MetaData::ATTR_CLUSTERED))
    cout << "Table is clustered" << endl;
  else
    cout << "Table is not clustered" << endl;
  if (metaData.getBoolean(MetaData::ATTR_INDEX_ONLY))
    cout << "Table is Index-only" << endl;
  else
    cout << "Table is not Index-only" << endl;
  cout << "Duration:";
  switch (metaData.getInt(MetaData::ATTR_DURATION))
  {
    case MetaData::DURATION_SESSION : cout << "Connection" << endl;
                                             break;
    case MetaData::DURATION_TRANS   : cout << "Transaction" << endl;
                                             break;
```

```
  case MetaData::DURATION_NULL  : cout << "Table not temporary" << endl;
                                                  break;
}
try
{
  cout << "Data Block Address:" <<
    metaData.getUInt (MetaData::ATTR_RDBA) << endl;
}
catch (SQLException ex)
{
  cout << ex.getMessage() << endl;
}
try
{
  cout << "Tablespace:" <<
    metaData.getInt (MetaData::ATTR_TABLESPACE) << endl;
}
catch (SQLException ex)
{
  cout << ex.getMessage() << endl;
}
try
{
  cout << "Object Id:" <<
    metaData.getUInt(MetaData::ATTR_OBJID) << endl;
}
catch (SQLException ex)
{
  cout << ex.getMessage() << endl;
}

int columnCount = metaData.getInt(MetaData::ATTR_NUM_COLS);
cout << "Number of Columns : " << columnCount << endl;

v1 =  metaData.getVector(MetaData::ATTR_LIST_COLUMNS);
for(int i=0; i < v1.size(); i++)
{
  MetaData md = v1[i];
  cout << " Column Name :" <<
    (md.getString(MetaData::ATTR_NAME)) << endl;
  cout << " Data Type :" <<
    (printType (md.getInt(MetaData::ATTR_DATA_TYPE))) << endl;
  cout << " Size :" << md.getInt(MetaData::ATTR_DATA_SIZE) << endl;
  cout << " Precision :" << md.getInt(MetaData::ATTR_PRECISION) << endl;
  cout << " Scale :" << md.getInt(MetaData::ATTR_SCALE) << endl;
  bool isnull = md.getBoolean(MetaData::ATTR_IS_NULL);
  if (isnull)
```

```
      cout << " Allows null" << endl;
    else
      cout << " Does not allow null" << endl;
  }
  cout << "describe_table - done" << endl;
}  // end of describe_table ()

// Describing a procedure
void describe_proc ()
{
  cout << "Describing the procedure - DEMO_PROC" << endl;
  MetaData metaData = conn->getMetaData("DEMO_PROC");
  vector<MetaData> v1 =
    metaData.getVector ( MetaData::ATTR_LIST_ARGUMENTS );
  cout << "The number of arguments are:" << v1.size() << endl;
  cout << "Object Name :" <<
    (metaData.getString(MetaData::ATTR_OBJ_NAME)) << endl;
  cout << "Schema Name :" <<
    (metaData.getString(MetaData::ATTR_OBJ_SCHEMA)) << endl;
  if (metaData.getInt(MetaData::ATTR_PTYPE) == MetaData::
    PTYPE_PROC)
  {
    cout << "DEMO_PROC is a procedure" << endl;
  }
  else
  {
    if (metaData.getInt(MetaData::ATTR_PTYPE) == MetaData::
      PTYPE_FUNC)
    {
      cout << "DEMO_PROC is a function" << endl;
    }
  }
  try
  {
  cout << "Object Id:" <<
    metaData.getUInt(MetaData::ATTR_OBJ_ID) << endl;
  }
  catch (SQLException ex)
  {
    cout << ex.getMessage() << endl;
  }
  try
  {
    cout << "Name :" <<
      (metaData.getString(MetaData::ATTR_NAME)) << endl;
```

```
}
catch (SQLException ex)
{
  cout << ex.getMessage() << endl;
}

if (metaData.getBoolean(MetaData::ATTR_IS_INVOKER_RIGHTS))
  cout << "It is Invoker-rights" << endl;
else
  cout << "It is not Invoker-rights" << endl;
cout << "Overload Id:" <<
  metaData.getInt(MetaData::ATTR_OVERLOAD_ID) << endl;

for(int i=0; i < v1.size(); i++)
{
  MetaData md = v1[i];
  cout << "Column Name :" <<
    (md.getString(MetaData::ATTR_NAME)) << endl;
  cout << "DataType :" <<
    (printType (md.getInt(MetaData::ATTR_DATA_TYPE)))
    << endl;
  cout << "Argument Mode:";
  int mode = md.getInt (MetaData::ATTR_IOMODE);
  if (mode == 0)
    cout << "IN" << endl;
  if (mode == 1)
    cout << "OUT" << endl;
  if (mode == 2)
    cout << "IN/OUT" << endl;
  cout << "Size :" <<
    md.getInt(MetaData::ATTR_DATA_SIZE) << endl;
  cout << "Precision :" <<
    md.getInt(MetaData::ATTR_PRECISION) << endl;
  cout << "Scale :" <<
    md.getInt(MetaData::ATTR_SCALE) << endl;
  int isNull = md.getInt ( MetaData::ATTR_IS_NULL);
  if (isNull != 0)
    cout << "Allows null," << endl;
  else
    cout << "Does not allow null," << endl;
```

```
    int hasDef = md.getInt ( MetaData::ATTR_HAS_DEFAULT);
    if (hasDef != 0)
      cout << "Has Default" << endl;
    else
      cout << "Does not have Default" << endl;
  }
  cout << "test1 - done" << endl;
}

// Method which prints the data type
string printType (int type)
{
  switch (type)
  {
    case OCCI_SQLT_CHR : return "VARCHAR2";
                         break;
    case OCCI_SQLT_NUM : return "NUMBER";
                         break;
    case OCCIINT : return "INTEGER";
                         break;
    case OCCIFLOAT : return "FLOAT";
                         break;
    case OCCI_SQLT_STR : return "STRING";
                         break;
    case OCCI_SQLT_VNU : return "VARNUM";
                         break;
    case OCCI_SQLT_LNG : return "LONG";
                         break;
    case OCCI_SQLT_VCS : return "VARCHAR";
                         break;
    case OCCI_SQLT_RID : return "ROWID";
                         break;
    case OCCI_SQLT_DAT : return "DATE";
                         break;
    case OCCI_SQLT_VBI : return "VARRAW";
                         break;
    case OCCI_SQLT_BIN : return "RAW";
                         break;
    case OCCI_SQLT_LBI : return "LONG RAW";
                         break;
    case OCCIUNSIGNED_INT : return "UNSIGNED INT";
                         break;
    case OCCI_SQLT_LVC : return "LONG VARCHAR";
                         break;
    case OCCI_SQLT_LVB : return "LONG VARRAW";
                         break;
```

```
      case OCCI_SQLT_AFC : return "CHAR";
                           break;
      case OCCI_SQLT_AVC : return "CHARZ";
                           break;
      case OCCI_SQLT_RDD : return "ROWID";
                           break;
      case OCCI_SQLT_NTY : return "NAMED DATA TYPE";
                           break;
      case OCCI_SQLT_REF : return "REF";
                           break;
      case OCCI_SQLT_CLOB: return "CLOB";
                           break;
      case OCCI_SQLT_BLOB: return "BLOB";
                           break;
      case OCCI_SQLT_FILE: return "BFILE";
                           break;
    }
  } // End of printType (int)


}; // end of class occidesc

int main (void)
{
  string user = "SCOTT";
  string passwd = "TIGER";
  string db = "";

  cout << "occidesc - Describing the various objects of the database" << endl;

  occidesc *demo = new occidesc (user, passwd, db);
  demo->describe_table();
  demo->describe_type();
  demo->describe_proc();
  delete demo;

}// end of main ()
```

occidml.cpp

次のコード例は、OCCI による表の行に対する挿入、選択、更新および削除操作の実行方法を示しています。

```
/**
 * occidml - To exhibit the insertion, Selection, updating and deletion of
 *            a row through OCCI.
 *
 * Description
 *   Create a program which has insert, select, update & delete as operations.
 *   Perform all these operations using OCCI interface.
 */

#include <iostream.h>
#include <occi.h>
using namespace oracle::occi;
using namespace std;

class  occidml
{
  private:

  Environment *env;
  Connection *conn;
  Statement *stmt;
  public:

  occidml (string user, string passwd, string db)
  {
    env = Environment::createEnvironment (Environment::DEFAULT);
    conn = env->createConnection (user, passwd, db);
  }

  ~occidml ()
  {
    env->terminateConnection (conn);
    Environment::terminateEnvironment (env);
  }

  /**
   * Insertion of a row with dynamic binding, PreparedStatement functionality.
   */
  void insertBind (int c1, string c2)
  {
    string sqlStmt = "INSERT INTO author_tab VALUES (:x, :y)";
    stmt=conn->createStatement (sqlStmt);
    try{
```

```
  stmt->setInt (1, c1);
  stmt->setString (2, c2);
  stmt->executeUpdate ();
  cout << "insert - Success" << endl;
  }catch(SQLException ex)
  {
   cout<<"Exception thrown for insertBind"<<endl;
   cout<<"Error number: "<<  ex.getErrorCode() << endl;
   cout<<ex.getMessage() << endl;
  }

  conn->terminateStatement (stmt);
}

/**
 * Inserting a row into the table.
 */
void insertRow ()
{
  string sqlStmt = "INSERT INTO author_tab VALUES (111, 'ASHOK')";
  stmt = conn->createStatement (sqlStmt);
  try{
  stmt->executeUpdate ();
  cout << "insert - Success" << endl;
  }catch(SQLException ex)
  {
   cout<<"Exception thrown for insertRow"<<endl;
   cout<<"Error number: "<<  ex.getErrorCode() << endl;
   cout<<ex.getMessage() << endl;
  }

  conn->terminateStatement (stmt);
}

/**
 * updating a row
 */
void updateRow (int c1, string c2)
{
  string sqlStmt =
    "UPDATE author_tab SET author_name = :x WHERE author_id = :y";
  stmt = conn->createStatement (sqlStmt);
  try{
  stmt->setString (1, c2);
  stmt->setInt (2, c1);
  stmt->executeUpdate ();
  cout << "update - Success" << endl;
```

```
  }catch(SQLException ex)
  {
   cout<<"Exception thrown for updateRow"<<endl;
   cout<<"Error number: "<<  ex.getErrorCode() << endl;
   cout<<ex.getMessage() << endl;
  }

  conn->terminateStatement (stmt);
}


/**
 * deletion of a row
 */
void deleteRow (int c1, string c2)
{
  string sqlStmt =
    "DELETE FROM author_tab WHERE author_id= :x AND author_name = :y";
  stmt = conn->createStatement (sqlStmt);
  try{
  stmt->setInt (1, c1);
  stmt->setString (2, c2);
  stmt->executeUpdate ();
  cout << "delete - Success" << endl;
  }catch(SQLException ex)
  {
   cout<<"Exception thrown for deleteRow"<<endl;
   cout<<"Error number: "<<  ex.getErrorCode() << endl;
   cout<<ex.getMessage() << endl;
  }

  conn->terminateStatement (stmt);
}

/**
 * displaying all the rows in the table
 */
void displayAllRows ()
{
  string sqlStmt = "SELECT author_id, author_name FROM author_tab";
  stmt = conn->createStatement (sqlStmt);
  ResultSet *rset = stmt->executeQuery ();
  try{
  while (rset->next ())
  {
    cout << "author_id: " << rset->getInt (1) << "  author_name: "
      << rset->getString (2) << endl;
```

```
    }
    }catch(SQLException ex)
    {
     cout<<"Exception thrown for displayAllRows"<<endl;
     cout<<"Error number: "<<  ex.getErrorCode() << endl;
     cout<<ex.getMessage() << endl;
    }

    stmt->closeResultSet (rset);
    conn->terminateStatement (stmt);
  }

}; // end of class  occidml


int main (void)
{
  string user = "SCOTT";
  string passwd = "TIGER";
  string db = "";

  cout << "occidml - Exhibiting simple insert, delete & update operations"
    << endl;
  occidml *demo = new occidml (user, passwd, db);

  cout << "Displaying all records before any operation" << endl;
  demo->displayAllRows ();

  cout << "Inserting a record into the table author_tab "
    << endl;
  demo->insertRow ();

  cout << "Displaying the records after insert " << endl;
  demo->displayAllRows ();

  cout << "Inserting a records into the table author_tab using dynamic bind"
    << endl;
  demo->insertBind (222, "ANAND");

  cout << "Displaying the records after insert using dynamic bind" << endl;
  demo->displayAllRows ();

  cout << "deleting a row with author_id as 222 from author_tab table" << endl;
  demo->deleteRow (222, "ANAND");
```

```
  cout << "updating a row with author_id as 444 from author_tab table" << endl;
  demo->updateRow (444, "ADAM");

  cout << "displaying all rows after all the operations" << endl;
  demo->displayAllRows ();

  delete (demo);
  cout << "occidml - done" << endl;
}
```

occiinh.typ

```
CASE=LOWER
MAPFILE=occiinhm.cpp
TYPE FOREIGN_STUDENT as foreign_student
```

occiinh.cpp

次のコード例は、サブタイプ表の行に対する挿入、選択、更新および削除操作によるオブジェクトの継承を示しています。

```
/**
 * occiinh.cpp - To exhibit the insertion, selection, updating and deletion
 *               of a row of a table of derived object.
 *
 * Description
 *   Create a program which has insert, select, update & delete as operations
 *   of a object.  Perform all these operations using OCCI interface.
 *   Hierarchy
 *   person_typ <---- student <----- parttime_stud <----- foreign_student
**/

#include <iostream.h>
#include <occi.h>
using namespace oracle::occi;
using namespace std;

#include "occiinhm.h"

/* Add on your methods in this class*/
class foreign_student_obj : public foreign_student
{
   /* New methods can be added here */
};
```

```
class occiinh
{
  private:

  Environment *env;
  Connection *con;

  // This method will return the Ref
  RefAny getRefObj(string sqlString)
  {
    Statement *stmt = con->createStatement (sqlString);
    ResultSet *rs;
    try
    {
       rs = stmt->executeQuery ();
       if ( rs->next() )
       {
          RefAny ref1 = rs->getRef (1);
          stmt->closeResultSet (rs);
          con->terminateStatement (stmt);
          return ref1;
       }
    }
    catch(SQLException ex)
    {
      cout << "Error in fetching ref" << endl;
    }
    stmt->closeResultSet (rs);
    con->terminateStatement (stmt);
  }


  public:

  occiinh (string user, string passwd, string db)
   throw (SQLException)
  {
    env = Environment::createEnvironment (Environment::OBJECT);
    occiinhm(env);
    con = env->createConnection (user, passwd, db);
  }// end of constructor occiinh (string, string, string)
```

```
~occiinh ()
 throw (SQLException)
{
  env->terminateConnection (con);
  Environment::terminateEnvironment (env);
}// end of destructor

/**
 * Insertion of a row
 */
void insertRow ()
 throw (SQLException)
{
  cout << "Inserting a record (joe)" << endl;
  string sqlStmt =
  "INSERT INTO foreign_student_tab VALUES(:a)";
  Statement *stmt = con->createStatement (sqlStmt);
  string fs_name = "joe";
  Number fs_ssn (4);
  Date fs_dob(env, 2000, 5, 11, 16, 05, 0);
  Number fs_stud_id (400);
  Ref< person_typ > fs_teammate = getRefObj(
     "SELECT REF(a) FROM person_tab a where name='john'");
  Number fs_course_id(4000);
  Ref< student > fs_partner = getRefObj(
    "SELECT REF(a) FROM student_tab a");
  string fs_country = "india";
  Ref< parttime_stud > fs_leader = getRefObj(
    "SELECT REF(a) FROM parttime_stud_tab a");
  foreign_student_obj fsobj;
  foreign_student_obj *fs_obj=&fsobj;
  fs_obj->setname(fs_name);
  fs_obj->setssn(fs_ssn);
  fs_obj->setdob(fs_dob);
  fs_obj->setstud_id(fs_stud_id);
  fs_obj->setteammate(fs_teammate);
  fs_obj->setcourse_id(fs_course_id);
  fs_obj->setpartner(fs_partner);
  fs_obj->setcountry(fs_country);
  fs_obj->setleader(fs_leader);
  stmt->setObject(1, fs_obj);
  stmt->executeUpdate();
```

```
  con->terminateStatement (stmt);
  cout << "Insertion Successful" << endl;
}// end of insertRow ();


/**
 * updating a row
 */
void updateRow ()
 throw (SQLException)
{
  cout << "Upadating record (Changing name,teammate and course_id)" << endl;
  string sqlStmt =
   "UPDATE foreign_student_tab SET name=:x, teammate=:y, course_id=:z";
  Statement *stmt = con->createStatement (sqlStmt);
  string fs_name = "jeffree";
  Ref< person_typ > fs_teammate = getRefObj(
     "SELECT REF(a) FROM person_tab a where name='jill'");
  Number fs_course_id(5000);
  stmt->setString(1, fs_name);
  stmt->setRef(2,fs_teammate);
  stmt->setInt(3, fs_course_id);
  stmt->executeUpdate ();
  con->commit();
  con->terminateStatement (stmt);
  cout << "Updation Successful" << endl;
}// end of updateRow (int, string);


/**
 * deletion of a row
 */
void deleteRow ()
 throw (SQLException)
{
  cout << "Deletion of jeffree record " << endl;
  string sqlStmt = "DELETE FROM foreign_student_tab where name=:x";
  Statement *stmt = con->createStatement (sqlStmt);
  string fs_name = "jeffree";
  stmt->setString(1,fs_name);
  stmt->executeUpdate();
  con->commit();
  con->terminateStatement (stmt);
  cout << "Deletion Successful" << endl;
}// end of deleteRow (int, string);
```

```
/**
 * displaying all the rows in the table
 */
void displayAllRows ()
 throw (SQLException)
{
  int count=0;
  string sqlStmt = "SELECT REF(a) FROM foreign_student_tab a";
  Statement *stmt = con->createStatement (sqlStmt);
  ResultSet *resultSet = stmt->executeQuery ();

  while (resultSet->next ())
  {
     count++;
     RefAny fs_refany = resultSet->getRef(1);
     Ref <foreign_student_obj> fs_ref(fs_refany);
     fs_ref.setPrefetch(4);
     string fmt = "DD-MON-YYYY";
     string nlsParam = "NLS_DATE_LANGUAGE = American";
     Date fs_dob = fs_ref->getdob();
     string date1 = fs_dob.toText (fmt, nlsParam);
     cout << "Foreign Student Information" << endl;
     cout << "Name      : " << fs_ref->getname();
     cout << "  SSN       : " << (int)fs_ref->getssn();
     cout << "  DOB       : " << date1 << endl;
     cout << "Stud id   : " << (int)fs_ref->getstud_id() ;
     cout << "  Course id : " << (int)fs_ref->getcourse_id();
     cout << "  Country   : " << fs_ref->getcountry() <<endl;
     Ref <person_typ> fs_teammate = (Ref <person_typ>)
       fs_ref->getteammate();
     cout << "Teammate's Information " << endl;
     cout << "Name      : " << fs_teammate->getname();
     cout << "  SSN       : " << (int)fs_teammate->getssn();
     fs_dob = fs_teammate->getdob();
     date1 = fs_dob.toText(fmt, nlsParam);
     cout << "  DOB       : " << date1 << endl << endl;
     /* Leader */
     Ref< parttime_stud > fs_leader = (Ref < parttime_stud >)
        fs_ref->getleader();
     /* Leader's Partner */
     Ref < student > fs_partner = (Ref <student> )
        fs_leader->getpartner();
    /* Leader's Partenr's teammate */
      fs_teammate = (Ref <person_typ>) fs_partner->getteammate();
```

```
        cout << "Leader Information " << endl;
        cout << "Name      : " << fs_leader->getname();
        cout << "  SSN       : " << (int)fs_leader->getssn();
        fs_dob = fs_leader->getdob();
        date1 = fs_dob.toText(fmt, nlsParam);
        cout << "  DOB       : " << date1 << endl;
        cout << "Stud id   : " << (int)fs_leader->getstud_id();
        cout << "  Course id : " << (int)fs_leader->getcourse_id() << endl;

        cout << "Leader's Partner's Information " << endl;
        cout << "Name      : " << fs_partner->getname() ;
        cout << "  SSN       : " << (int)fs_partner->getssn();
        fs_dob = fs_partner->getdob();
        date1 = fs_dob.toText(fmt, nlsParam);
        cout << "  DOB       : " << date1 ;
        cout << "  Stud id   : " << (int)fs_partner->getstud_id() << endl;


        cout << "Leader's Partner's Teammate's Information " << endl;
        cout << "Name      : " << fs_teammate->getname();
        cout << "  SSN       : " << (int)fs_teammate->getssn();
        fs_dob = fs_teammate->getdob();
        date1 = fs_dob.toText(fmt, nlsParam);
        cout << "  DOB       : " << date1 << endl << endl;

    }//end of while (resultSet->next ());
    if (count <=0)
       cout << "No record found " << endl;
    stmt->closeResultSet (resultSet);
    con->terminateStatement (stmt);
  }// end of updateRow (string);

}; // end of class occiinh


int main (void)
{
  string user = "scott";
  string passwd = "tiger";
  string db = "";

  try
  {
    cout << "occiinh - Exhibiting simple insert, delete & update operations"
     " on Oracle objects" << endl;
    occiinh *demo = new occiinh (user, passwd, db);
```

```
    cout << "displaying all rows before operations" << endl;
    demo->displayAllRows ();

    demo->insertRow ();
    cout << "displaying all rows after insertions" << endl;
    demo->displayAllRows ();

    demo->updateRow ();
    cout << "displaying all rows after updations" << endl;
    demo->displayAllRows ();

    demo->deleteRow ();
    cout << "displaying all rows after deletions" << endl;
    demo->displayAllRows ();


    delete (demo);
    cout << "occiinh - done" << endl;
  }catch (SQLException ea)
  {
    cerr << "Error running the demo: " << ea.getMessage () << endl;
  }
}// end of int main (void);
```

occiobj.typ

```
CASE=SAME
MAPFILE=occiobjm.cpp
TYPE address as address
```

occiobj.cpp

次のコード例は、列の 1 つにオブジェクトが含まれた表の行に対する挿入、選択、更新および削除操作の実行方法を示しています。

```
/**
 * occiobj.cpp - To exhibit the insertion, selection, updating and deletion
 *               of a row containing object as one of the column.
 *
 * Description
 *   Create a program which has insert, select, update & delete as operations
 *   of a object.  Perform all these operations using OCCI interface.
 */
```

```
#include <iostream.h>
#include <occi.h>
using namespace oracle::occi;
using namespace std;

#include "occiobjm.h"

class occiobj
{
  private:

  Environment *env;
  Connection *con;
  Statement *stmt;
  public:

  occiobj (string user, string passwd, string db)
  {
    env = Environment::createEnvironment (Environment::OBJECT);
    occiobjm (env);
    con = env->createConnection (user, passwd, db);
  }

  ~occiobj ()
  {
    env->terminateConnection (con);
    Environment::terminateEnvironment (env);
  }

  /**
   * Insertion of a row
   */
  void insertRow (int c1, int a1, string a2)
  {
    cout << "Inserting record - Publisher id :" << c1 <<
      ", Publisher address :" << a1 << ", " << a2 <<endl;
    string sqlStmt = "INSERT INTO publisher_tab VALUES (:x, :y)";
    try{
    stmt = con->createStatement (sqlStmt);
    stmt->setInt (1, c1);
    address *o = new address ();
    o->setStreet_no (Number ( a1));
    o->setCity (a2);
    stmt->setObject (2, o);
    stmt->executeUpdate ();
    cout << "Insert - Success" << endl;
    delete (o);
```

```
  }catch(SQLException ex)
  {
   cout<<"Exception thrown for insertRow"<<endl;
   cout<<"Error number: "<<  ex.getErrorCode() << endl;
   cout<<ex.getMessage() << endl;
  }

  con->terminateStatement (stmt);
}


/**
 * updating a row
 */
void updateRow (int c1, int a1, string a2)
{
  cout << "Upadating record with publisher id :"<< c1 << endl;
  string sqlStmt =
    "UPDATE publisher_tab SET publisher_add= :x WHERE publisher_id = :y";
  try{
  stmt = con->createStatement (sqlStmt);
  address *o = new address ();
  o->setStreet_no (Number ( a1));
  o->setCity (a2);
  stmt->setObject (1, o);
  stmt->setInt (2, c1);
  stmt->executeUpdate ();
  cout << "Update - Success" << endl;
  delete (o);
  }catch(SQLException ex)
  {
   cout<<"Exception thrown for updateRow"<<endl;
   cout<<"Error number: "<<  ex.getErrorCode() << endl;
   cout<<ex.getMessage() << endl;
  }
  con->terminateStatement (stmt);
}


/**
 * deletion of a row
 */
void deleteRow (int c1, int a1, string a2)
{
  cout << "Deletion of record where publisher id :" << c1 <<endl;
  string sqlStmt =
  "DELETE FROM publisher_tab WHERE publisher_id= :x AND publisher_add = :y";
```

```
  try{
  stmt = con->createStatement (sqlStmt);
  stmt->setInt (1, c1);

  address *o = new address ();
  o->setStreet_no (Number ( a1));
  o->setCity (a2);
  stmt->setObject (2, o);
  stmt->executeUpdate ();
  cout << "Delete - Success" << endl;
  delete (o);
  }catch(SQLException ex)
  {
   cout<<"Exception thrown for deleteRow"<<endl;
   cout<<"Error number: "<<  ex.getErrorCode() << endl;
   cout<<ex.getMessage() << endl;
  }

  con->terminateStatement (stmt);
}

/**
 * displaying all the rows in the table
 */
void displayAllRows ()
{
  string sqlStmt = "SELECT publisher_id, publisher_add FROM publisher_tab";
  try{
  stmt = con->createStatement (sqlStmt);
  ResultSet *rset = stmt->executeQuery ();

  while (rset->next ())
  {
    cout << "publisher id: " << rset->getInt (1)
         << " publisher address: address (" ;
    address *o =  (address *)rset->getObject (2);
    cout << (int)o->getStreet_no () << ", " << o->getCity () << ")" << endl;
  }

  stmt->closeResultSet (rset);
  }catch(SQLException ex)
  {
   cout<<"Exception thrown for displayAllRows"<<endl;
   cout<<"Error number: "<<  ex.getErrorCode() << endl;
   cout<<ex.getMessage() << endl;
  }
```

```
    con->terminateStatement (stmt);
  }

};//end of class occiobj;


int main (void)
{
  string user = "SCOTT";
  string passwd = "TIGER";
  string db = "";

  try
  {
    cout << "occiobj - Exhibiting simple insert, delete & update operations"
     " on Oracle objects" << endl;
    occiobj *demo = new occiobj (user, passwd, db);

    cout << "displaying all rows before operations" << endl;
    demo->displayAllRows ();

    demo->insertRow (12, 122, "MIKE");

    demo->deleteRow (11, 121, "ANNA");

    demo->updateRow (23, 123, "KNUTH");

    cout << "displaying all rows after all operations" << endl;
    demo->displayAllRows ();

    delete (demo);
    cout << "occiobj - done" << endl;
  }catch (SQLException ea)
  {
    cerr << "Error running the demo: " << ea.getMessage () << endl;
  }
}
```

occipobj.typ

```
CASE=SAME
MAPFILE=occipobjm.cpp
TYPE address as  address
```

occipobj.cpp

次のコード例は、永続オブジェクトの挿入、選択および更新操作の実行方法と永続オブジェクトの確保、確保解除、削除マークの設定およびフラッシュの実行方法を示しています。

```
/**
 * occipobj.cpp - Manipulation (Insertion, selection & updating) of
 *                persistant objects, along with pinning, unpinning, marking
 *                for deletion & flushing.
 *
 * Description
 *   Create a program which has insert, select, update & delete as operations
 *   of a persistant object.  Along with the these the operations on Ref. are
 *   pinning, unpinning, marked for deletion & flushing.
 *
 */

#include <iostream.h>
#include <occi.h>
using namespace oracle::occi;
using namespace std;

#include "occipobjm.h"

class address_obj : public address
{
  public:
   address_obj()
   {
   }

   address_obj(Number sno,string cty)
   {
     setStreet_no(sno);
     setCity(cty);
   }
};

class occipobj
{
  private:

  Environment *env;
  Connection *conn;
  Statement *stmt;
  string tableName;
  string typeName;
```

```
public:

occipobj (string user, string passwd, string db)
{
  env = Environment::createEnvironment (Environment::OBJECT);
  occipobjm (env);
  conn = env->createConnection (user, passwd, db);
}

~occipobj ()
{
  env->terminateConnection (conn);
  Environment::terminateEnvironment (env);
}

void setTableName (string s)
{
  tableName = s;
}

/**
 * Insertion of a row
 */
void insertRow (int a1, string a2)
{
  cout << "Inserting row ADDRESS (" << a1 << ", " << a2 << ")" << endl;
  Number n1(a1);
  address_obj *o = new (conn, tableName) address_obj(n1, a2);
  conn->commit ();
  cout << "Insertion - Successful" << endl;
}

/**
 * updating a row
 */
void updateRow (int b1, int a1, string a2)
{
  cout << "Updating a row with attribute a1 = " << b1 << endl;
  stmt = conn->createStatement
    ("SELECT REF(a) FROM address_tab a WHERE street_no = :x FOR UPDATE");
  stmt->setInt (1, b1);
  ResultSet *rs = stmt->executeQuery ();
  try{
  if ( rs->next() )
  {
    RefAny rany = rs->getRef (1);
    Ref <address_obj > r1(rany);
```

```
      address_obj *o = r1.ptr();
      o->markModified ();
      o->setStreet_no (Number (a1));
      o->setCity (a2);
      o->flush ();
    }
    }catch(SQLException ex)
    {
     cout<<"Exception thrown updateRow"<<endl;
     cout<<"Error number: "<<  ex.getErrorCode() << endl;
     cout<<ex.getMessage() << endl;
    }

    conn->commit ();
    conn->terminateStatement (stmt);
    cout << "Updation - Successful" << endl;
  }


  /**
   * deletion of a row
   */
  void deleteRow (int a1, string a2)
  {
    cout << "Deleting a row with object ADDRESS (" << a1 << ", " << a2
     << ")" << endl;
    stmt = conn->createStatement
    ("SELECT REF(a) FROM address_tab a WHERE street_no = :x AND city = :y FOR
UPDATE");
    stmt->setInt (1, a1);
    stmt->setString (2, a2);
    ResultSet *rs = stmt->executeQuery ();
    try{
    if ( rs->next() )
    {
      RefAny rany = rs->getRef (1);
      Ref<address_obj > r1(rany);
      address_obj *o = r1.ptr();
      o->markDelete ();
    }
    }catch(SQLException ex)
    {
     cout<<"Exception thrown for deleteRow"<<endl;
     cout<<"Error number: "<<  ex.getErrorCode() << endl;
     cout<<ex.getMessage() << endl;
    }
```

```
    conn->commit ();
    conn->terminateStatement (stmt);
    cout << "Deletion - Successful" << endl;
  }

  /**
   * displaying all the rows in the table
   */
  void displayAllRows ()
  {
    string sqlStmt = "SELECT REF (a) FROM address_tab a";
    stmt = conn->createStatement (sqlStmt);
    ResultSet *rset = stmt->executeQuery ();
    try{
    while (rset->next ())
    {
      RefAny rany = rset->getRef (1);
      Ref<address_obj > r1(rany);
      address_obj *o = r1.ptr();
      cout << "ADDRESS(" << (int)o->getStreet_no () << ", " << o->getCity () << ")"
<< endl;
    }
    }catch(SQLException ex)
    {
     cout<<"Exception thrown for displayAllRows"<<endl;
     cout<<"Error number: "<<  ex.getErrorCode() << endl;
     cout<<ex.getMessage() << endl;
    }

    stmt->closeResultSet (rset);
    conn->terminateStatement (stmt);
  }

}; // end of class occipobj


int main (void)
{
  string user = "SCOTT";
  string passwd = "TIGER";
  string db = "";
```

```
  try
  {
    cout << "occipobj - Exhibiting simple insert, delete & update operations"
     " on persistent objects" << endl;
    occipobj *demo = new occipobj (user, passwd, db);

    cout << "Displaying all rows before the opeations" << endl;
    demo->displayAllRows ();

    demo->setTableName ("ADDRESS_TAB");

    demo->insertRow (21, "KRISHNA");

    demo->deleteRow (22, "BOSTON");

    demo->updateRow (33, 123, "BHUMI");

    cout << "Displaying all rows after all the operations" << endl;
    demo->displayAllRows ();

    delete (demo);
    cout << "occipobj - done" << endl;
  }catch (SQLException ea)
  {
    cerr << "Error running the demo: " << ea.getMessage () << endl;
  }
}
```

occipool.cpp

次のコード例は、OCCI の接続プール・インタフェースの使用方法を示しています。

```
/**
*  occipool - Demontrating the Connection Pool interface of OCCI.
*
*  DESCRIPTION :
*    This program demonstates the creating and using of connection pool in the
*    database and fetching records of a table.
*
*/

#include <iostream.h>
#include <occi.h>
using namespace oracle::occi;
using namespace std;
```

```
class occipool
{
  private:

  Environment *env;
  Connection *con;
  Statement *stmt;
  public :
  /**
   * Constructor for the occipool test case.
   */
  occipool ()
  {
    env = Environment::createEnvironment (Environment::DEFAULT);
  }// end of constructor occipool ()

  /**
   * Destructor for the occipool test case.
   */
  ~occipool ()
  {
    Environment::terminateEnvironment (env);
  }  // end of ~occipool ()

  /**
   * The testing logic of the test case.
   */
  dvoid select ()
  {
    cout << "occipool - Selecting records using ConnectionPool interface" <<
    endl;
    const string poolUserName = "SCOTT";
    const string poolPassword = "TIGER";
    const string connectString = "";
    const string username = "SCOTT";
    const string passWord = "TIGER";
    unsigned int maxConn = 5;
    unsigned int minConn = 3;
    unsigned int incrConn = 2;
    ConnectionPool *connPool = env->createConnectionPool
      (poolUserName, poolPassword, connectString, minConn, maxConn, incrConn);
    try{
    if (connPool)
      cout << "SUCCESS - createConnectionPool" << endl;
    else
      cout << "FAILURE - createConnectionPool" << endl;
    con = connPool->createConnection (username, passWord);
```

```
    if (con)
      cout << "SUCCESS - createConnection" << endl;
    else
      cout << "FAILURE - createConnection" << endl;
    }catch(SQLException ex)
    {
     cout<<"Exception thrown for createConnectionPool"<<endl;
     cout<<"Error number: "<<  ex.getErrorCode() << endl;
     cout<<ex.getMessage() << endl;
    }

    cout << "retrieving the data" << endl;
    stmt = con->createStatement
      ("SELECT author_id, author_name FROM author_tab");
    ResultSet *rset = stmt->executeQuery();
    while (rset->next())
    {
      cout << "author_id:" << rset->getInt (1) << endl;
      cout << "author_name:" << rset->getString (2) << endl;
    }
    stmt->closeResultSet (rset);
    con->terminateStatement (stmt);
    connPool->terminateConnection (con);
    env->terminateConnectionPool (connPool);

    cout << "occipool - done" << endl;
  } // end of test (Connection *)

}; // end of class occipool

int main (void)
{
  string user = "SCOTT";
  string passwd = "TIGER";
  string db = "";

  occipool *demo = new occipool ();

  demo->select();
  delete demo;

}// end of main ()
```

occiproc.cpp

次のコード例は、バインド・パラメータを指定した PL/SQL プロシージャの呼出し方法を示しています。

```
/**
*  occiproc - Demonstrating the invoking of a PL/SQL function and procedure
*  using OCCI.
*
*  DESCRIPTION :
*    This program demonstrates the invoking a PL/SQL function and procedure
*    having IN, IN/OUT and OUT parameters.
*
*/

#include <iostream.h>
#include <occi.h>
using namespace oracle::occi;
using namespace std;

class occiproc
{
  private:

  Environment *env;
  Connection *con;

  public :
  /**
   * Constructor for the occiproc demo program.
   */
  occiproc (string user, string passwd, string db) throw (SQLException)
  {
    env = Environment::createEnvironment (Environment::DEFAULT);
    con = env->createConnection (user, passwd, db);
  }// end of constructor occiproc (string, string, string )

  /**
   * Destructor for the occiproc demo program.
   */
  ~occiproc () throw (SQLException)
  {
    env->terminateConnection (con);
    Environment::terminateEnvironment (env);
  }  // end of ~occiproc ()

  // Function to call a PL/SQL procedure
  void callproc ()
```

```
{
  cout << "callproc - invoking a PL/SQL procedure having IN, OUT and IN/OUT ";
  cout << "parameters" << endl;
  Statement *stmt = con->createStatement
    ("BEGIN demo_proc(:v1, :v2, :v3); END;");
  cout << "Executing the block :" << stmt->getSQL() << endl;
  stmt->setInt (1, 10);
  stmt->setString (2, "IN");
  stmt->registerOutParam (2, OCCISTRING, 30, "");
  stmt->registerOutParam (3, OCCISTRING, 30, "");
  int updateCount = stmt->executeUpdate ();
  cout << "Update Count:" << updateCount << endl;

  string c1 = stmt->getString (2);
  string c2 = stmt->getString (3);
  cout << "Printing the INOUT & OUT parameters:" << endl;
  cout << "Col2:" << c1 << " Col3:" << c2 << endl;

  con->terminateStatement (stmt);
  cout << "occiproc - done" << endl;
} // end of callproc ()

// Function to call a PL/SQL function
void callfun ()
{ cout << "callfun - invoking a PL/SQL function having IN, OUT and IN/OUT ";
  cout << "parameters" << endl;
  Statement *stmt = con->createStatement
    ("BEGIN :a := demo_fun(:v1, :v2, :v3); END;");
  cout << "Executing the block :" << stmt->getSQL() << endl;
  stmt->setInt (2, 10);
  stmt->setString (3, "IN");
  stmt->registerOutParam (1, OCCISTRING, 30, "");
  stmt->registerOutParam (3, OCCISTRING, 30, "");
  stmt->registerOutParam (4, OCCISTRING, 30, "");
  int updateCount = stmt->executeUpdate ();
  cout << "Update Count : " << updateCount << endl;

  string c1 = stmt->getString (1);
  string c2 = stmt->getString (3);
  string c3 = stmt->getString (4);

  cout << "Printing the INOUT & OUT parameters :" << endl;
  cout << "Col2:" << c2 << " Col3:" << c3 << endl;
  cout << "Printing the return value of the function :";
  cout << c1 << endl;
```

```
    con->terminateStatement (stmt);
    cout << "occifun - done" << endl;
  } // end of callfun ()
}; // end of class occiproc

int main (void)
{
  string user = "SCOTT";
  string passwd = "TIGER";
  string db = "";

  cout << "occiproc - invoking a PL/SQL function and procedure having ";
  cout << "parameters" << endl;

  occiproc *demo = new occiproc (user, passwd, db);
  demo->callproc();
  demo->callfun();
  delete demo;

}// end of main ()
```

occistre.cpp

次のコード例は、OCCI の ResultSet ストリームの使用方法を示しています。

```
/**
 * occistrm - Demonstrating the usage of streams for VARCHAR2 data
 *
 * Description
 *   This demo program selects VARCHAR2 data using stream operations.
 */

#include <iostream.h>
#include <occi.h>
using namespace oracle::occi;
using namespace std;

class occistrm
{
  private:

  Environment *env;
  Connection *conn;

  public:
```

```
  occistrm (string user, string passwd, string db)
   throw (SQLException)
  {
    env = Environment::createEnvironment (Environment::DEFAULT);
    conn = env->createConnection (user, passwd, db);
  }// end of constructor occistrm (string, string, string)

  ~occistrm ()
   throw (SQLException)
  {
    env->terminateConnection (conn);
    Environment::terminateEnvironment (env);
  }// end of destructor

  /**
   * displaying all the rows in the table
   */
  void displayAllRows ()
  {
    Statement *stmt = conn->createStatement (
      "SELECT summary FROM book WHERE bookid = 11");
    stmt->execute ();
    ResultSet *rs = stmt->getResultSet ();
    rs->setCharacterStreamMode(1, 4000);
    char buffer[500];
    int length = 0;
    unsigned int size = 500;

    while (rs->next ())
    {
      Stream *stream = rs->getStream (1);
      while( (length=stream->readBuffer(buffer, size))!=-1)
      {
        cout << "Read " << length << " bytes from stream" << endl;
      }
    }
    stmt->closeResultSet (rs);
    conn->terminateStatement (stmt);
  }// end of updateRow (string);

};  // end of class occistrm
```

```
int main (void)
{
  string user = "SCOTT";
  string passwd = "TIGER";
  string db = "";

  cout << "occistrm - Exhibiting usage of streams for VARCHAR2 data"
    << endl;
  occistrm *demo = new occistrm (user, passwd, db);

  demo->displayAllRows ();

  delete (demo);
  cout << "occistrm - done" << endl;
}// end of int main (void);
```

索引

L

M

N

O

P

R

S

T

U

V

あ

い

う

え

お

か

き

く

こ

さ

し

す

せ

そ

た

て

と

な

ね

は

ひ

ふ

め

り

る

れ