

Oracle8i

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

リリース 8.1

2000 年 2 月

部品番号 : J00944-01

Oracle8i 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, リリース 8.1

部品番号 : J00944-01

原本名 : Application Developer's Guide - Object-Relational Features, Release 2 (8.1.6)

原本部品番号 : A76976-01

原著者 : John Russell

原本協力者 : S. Banerjee, V. Krishnamurthy, M. Krishnaprasad, G. Lee, S. Muralidhar, D. Raphaely, R. Urbano, S. Krishnaswamy, M. Morsi, R. Murthy, K. Osinski, E. Rohwedder, N. Shariatpanahy

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

Printed in Japan.

制限付権利の説明

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

当プログラムのリバース・エンジニアリング等は禁止されています。

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

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

危険な用途への使用について

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

Restricted Rights Notice

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

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

目次

はじめに	ix
------------	----

1 Oracle オブジェクトの概要

Oracle オブジェクトの基本	1-1
オブジェクト・リレーショナル・モデル	1-1
オブジェクト型	1-2
オブジェクト	1-3
メソッド	1-3
オブジェクト表	1-5
オブジェクト・ビュー	1-6
REF データ型	1-6
コレクション	1-8
継承	1-10
オブジェクト指向モデルの例	1-10

2 Oracle オブジェクトの管理

オブジェクト型および参照の使用	2-1
NULL オブジェクトおよび属性	2-2
オブジェクトおよびコレクションのデフォルト値	2-3
オブジェクト表の制約	2-3
オブジェクト表および NESTED TABLE の索引	2-4
オブジェクト表のトリガー	2-5
REF 型の列および属性のルール	2-5
名前解決	2-6

引数なしのメソッド・コール	2-8
コレクションの使用	2-8
コレクションの問合せ	2-8
コレクション・アンネスティング	2-9
コレクションの DML	2-9
オブジェクト型およびそれらのメソッドの権限	2-10
システム権限	2-10
スキーマ・オブジェクト権限	2-11
新しい型または表での型の使用	2-11
例	2-11
オブジェクト型へのアクセスおよびオブジェクト・アクセスの権限	2-12
依存性および不完全な型	2-14
不完全な型の再定義	2-15
表の型依存性	2-15
オブジェクト型のインポート/エクスポート/ロード	2-15

3 Oracle プログラム環境のオブジェクト・サポート

SQL	3-1
PL/SQL	3-2
Oracle Call Interface (OCI)	3-2
OCI プログラムにおける結合アクセス	3-3
OCI プログラムのナビゲーション・アクセス	3-4
オブジェクト・キャッシュ	3-4
オブジェクトを操作する OCI プログラムの作成	3-5
Pro*C/C++	3-6
Pro*C/C++ での結合アクセス	3-6
Pro*C/C++ でのナビゲーション・アクセス	3-7
Oracle のオブジェクト型と C 言語のデータ型の変換	3-7
オブジェクト・タイプ・トランスレータ (OTT)	3-8
Oracle Objects For OLE (Visual Basic、Excel、ActiveX、Active Server Page 対応)	3-9
Visual Basic でのオブジェクトの表現形式 (OraObject)	3-10
Visual Basic での REF の表現形式 (OraRef)	3-10
Visual Basic での VARRAY および NESTED TABLE の表現形式 (OraCollection)	3-11
Java: JDBC、Oracle SQLJ および JPublisher	3-12
JDBC を利用した Oracle オブジェクト・データへのアクセス	3-12

SQLJ を利用した Oracle オブジェクト・データへのアクセス	3-12
JPublisher を使用した JDBC および SQLJ プログラム用 Java クラスの作成	3-13

4 オブジェクト・モデルのリレーショナル・データへの適用

オブジェクト・ビュー使用の利点	4-2
オブジェクト・ビューの定義	4-3
アプリケーションにおけるオブジェクト・ビューの使用	4-4
オブジェクト・ビューにおけるオブジェクトのネスト	4-4
オブジェクト・ビューにおける NULL オブジェクトの識別	4-6
オブジェクト・ビューにおける NESTED TABLE および VARRAY の使用	4-6
オブジェクト・ビューに対するオブジェクト識別子の指定	4-7
オブジェクト・ビューが提供するオブジェクトへの参照の作成	4-9
オブジェクト・ビューを利用した逆関連のモデル化	4-10
オブジェクト・ビューの更新	4-11
ビューにおける NESTED TABLE 列の更新	4-11
変更および妥当性チェックを制御する INSTEAD-OF トリガーの使用	4-11
オブジェクト・モデルのリモート表への適用	4-12
オブジェクト・ビューにおける複雑な関連の定義	4-13
循環参照を示す表および型	4-14
循環参照を持つオブジェクト・ビューの作成	4-16

5 Oracle オブジェクトの設計上の考慮点

列または行としてのオブジェクトの表現	5-1
列オブジェクトの記憶域	5-2
オブジェクト表の行オブジェクトの記憶域	5-7
オブジェクト比較のパフォーマンス	5-7
オブジェクト識別子 (OID) の記憶域上の考慮点	5-8
REF の記憶域サイズ	5-9
REF 列での整合性制約	5-9
SCOPED REF のパフォーマンスおよび記憶域上の考慮点	5-9
SCOPED REF の索引作成	5-10
WITH ROWID オプションを使用したオブジェクト・アクセスの高速化	5-11
ネストを解除する問合せを使用したりレーショナル形式でのオブジェクト・データの表示	5-12
VARRAY の記憶域上の考慮点	5-13
VARRAY および NESTED TABLE のパフォーマンス	5-14
NESTED TABLE	5-14

NESTED TABLE の記憶域	5-14
NESTED TABLE の索引	5-17
NESTED TABLE のロケータ	5-18
セット・メンバーシップ問合せの最適化	5-18
NESTED TABLE での DML 操作	5-19
他のコレクション内でのコレクションのネスト	5-20
メソッド関数に対する言語の選択	5-26
静的メソッド	5-28
実行者権限を使用した再使用コードの作成	5-29
タイプ・メソッドの戻り値に関するファンクション索引	5-30
リリース 8.1 での新しいオブジェクト形式	5-31
オブジェクト表およびオブジェクト列のレプリケーション	5-31
Oracle の継承の実装結果	5-32
継承のシミュレート	5-32
オブジェクトでの制約	5-37
型の発展	5-38
パフォーマンス・チューニング	5-38
Oracle オブジェクトでのパラレル問合せ	5-38

6 Oracle オブジェクトの高度なトピック

オブジェクトの記憶域	6-1
リーフ・レベル属性	6-1
列にまたがって分割される行オブジェクト	6-1
列オブジェクトを持つ表の非表示列	6-2
REF	6-2
NESTED TABLE の内部レイアウト	6-3
VARRAY の内部レイアウト	6-3
オブジェクト識別子	6-3
オブジェクトに対する OCI のヒントおよび技法	6-4
オブジェクト・モードでの OCI プログラムの初期化	6-4
新しいオブジェクトの作成	6-4
オブジェクトの更新	6-5
オブジェクトの削除	6-5
オブジェクト・キャッシュ・サイズの制御	6-5
クライアント・キャッシュへのオブジェクトの取出し（確保）	6-6
ロック技法の選択方法	6-8

オブジェクト・キャッシュからのオブジェクトのフラッシュ	6-9
関連オブジェクトのプリフェッチ（複合オブジェクト検索）	6-9
OCI および Oracle オブジェクトのデモ	6-11
オブジェクト・ビューが提供するオブジェクトでの OCI オブジェクト・キャッシュの使用	6-11
Oracle オブジェクトを持つ表のパーティション化	6-14
オブジェクト・ビューでのパラレル問合せ	6-15
ロケータで NESTED TABLE のパフォーマンスを向上させる方法	6-15

7 Oracle オブジェクトを使用したプログラミングに関する FAQ

Oracle オブジェクトに関する一般的な質問	7-2
オブジェクト・リレーショナル機能は、別のオプションですか？	7-2
Oracle8i のオブジェクト・リレーショナルおよび拡張テクノロジーの設計目標は何ですか？	7-2
Oracle8i のオブジェクト・リレーショナル・テクノロジーの主な機能は何ですか？	7-2
Oracle8i の新しいオブジェクト・リレーショナル機能にはどのようなものがありますか？	7-5
オブジェクト型	7-6
構造化データとは何ですか？	7-6
ユーザー定義型、ユーザー定義ファンクションおよび抽象データ型はどこにありますか？	7-6
オブジェクト型とは何ですか？	7-7
オブジェクト型はなぜ便利なのですか？	7-7
Oracle8i では、オブジェクト・データはどのように格納および管理されますか？	7-7
Oracle8i では、継承はサポートされていますか？	7-8
オブジェクト・メソッド	7-8
オブジェクト・メソッドはどの言語を使用して記述できますか？	7-8
オブジェクト・メソッドに PL/SQL と Java のどちらを使用するかは、 どのように判断しますか？	7-8
外部プロシージャはいつ使用する必要がありますか？	7-8
定義者権限および実行者権限とは何ですか？	7-9
オブジェクト参照	7-9
オブジェクト参照とは何ですか？	7-9
オブジェクト参照はいつ使用する必要がありますか？外部キーとの違いは何ですか？	7-9
主キーに基づいてオブジェクト参照を作成できますか？	7-10
SCOPED REF とは何ですか？また、いつ使用する必要がありますか？	7-10
PL/SQL および Java でオブジェクト参照を使用してオブジェクトを操作できますか？	7-10
コレクション	7-10
Oracle8i ではどのような種類のコレクションがサポートされていますか？	7-10

コレクションのモデル化に VARRAY と NESTED TABLE のどちらを使用するかは、 どのように判断すればよいですか？	7-11
Oracle8i オブジェクトでは、コレクション内のコレクションがサポートされていますか？	7-11
コレクション・ロケータとは何ですか？	7-11
コレクション・アンネスティングとは何ですか？	7-11
オブジェクト・ビュー	7-12
オブジェクト・ビューとオブジェクト表の違いは何ですか？	7-12
オブジェクト・ビューは更新可能ですか？	7-12
オブジェクト・キャッシュ	7-12
なぜオブジェクト・キャッシュが必要なのですか？	7-12
オブジェクト・ロックはオブジェクト・キャッシュでサポートされますか？	7-13
ラージ・オブジェクト (LOB)	7-13
Oracle8i を使用してラージ・オブジェクトを管理するにはどのようにすればよいですか？	7-13
ユーザー定義オペレータ	7-14
ユーザー定義オペレータとは何ですか？	7-14
ユーザー定義オペレータはなぜ役に立つのですか？	7-14

8 オブジェクト・リレーショナル機能を使用したサンプル・アプリケーション

概要	8-2
発注書の例	8-3
リレーショナル・モデルでのアプリケーションの実装	8-4
エンティティおよび関連	8-5
リレーショナル・モデルでの表の作成	8-5
リレーショナル・モデルでの値の挿入	8-8
リレーショナル・モデルでのデータの問合せ	8-9
リレーショナル・モデルでのデータの更新	8-10
リレーショナル・モデルでのデータの削除	8-10
純粋なリレーショナル・モデルの制限事項	8-11
オブジェクト・リレーショナル・データベース・システムの発展	8-12
オブジェクト・リレーショナル・モデルでのアプリケーションの実装	8-13
型の定義	8-14
メソッドの定義	8-21
オブジェクト表の作成	8-23
オブジェクト表のテンプレートとしてのオブジェクトのデータ型	8-25
オブジェクト識別子および参照	8-26

埋込みオブジェクト付きのオブジェクト表	8-26
Java を使用したオブジェクトの操作	8-39
oracle.sql.* クラス（緩い型指定）の使用	8-39
強力な型指定（SQLData または CustomDatum）の使用	8-41
Oracle Objects for OLE でのオブジェクトの操作	8-45
データの選択	8-45
データの挿入	8-46
データの更新	8-48
メソッド関数のコール	8-50

索引

はじめに

『Oracle8i 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』では、Oracle Server リリース 8.1 のオブジェクト・リレーショナル機能を使用するアプリケーションの作成方法について説明します。このマニュアルの内容は、すべてのプラットフォームで動作する Oracle Server のバージョンに適用されますが、システム固有の情報は含みません。

内容は次のとおりです。

- [このマニュアルについて](#)
- [対象読者](#)
- [機能範囲および可用性](#)
- [関連マニュアル](#)
- [このマニュアルの構成](#)
- [このマニュアルの表記規則](#)

このマニュアルについて

アプリケーション開発者は、再使用可能なコードを作成したり、データベース・スキーマにおけるアプリケーション・ドメインを正確にモデル化するために役立つ機能に関心があるでしょう。このマニュアルでは、これらに関する一連のアプリケーション開発機能について説明します。データベース・アプリケーションの開発方法については、あらかじめ理解しておいてください。C++ や Java などのオブジェクト指向言語で環境を整備する際に役立ちます。

対象読者

このマニュアルは、新しいアプリケーションを開発したり、既存のアプリケーションを Oracle 環境で実行できるように変換するプログラマを対象としています。オブジェクト・リレーショナル機能は、マルチメディア、地理情報システム (GIS) および複雑なデータを処理するアプリケーションでよく使用されます。オブジェクト・ビュー機能は、既存のリレーショナル・スキーマ上に新しいアプリケーションを作成する場合に役立ちます。

このマニュアルは、アプリケーション・プログラミングの実践的な知識があり、構造化問合せ言語 (SQL) を使用してリレーショナル・データベース・システム内の情報にアクセスできることを前提としています。

機能範囲および可用性

このマニュアルには、Oracle8i および Oracle8i Enterprise Edition 製品の特徴および機能を説明した情報が含まれています。Oracle8i および Oracle8i Enterprise Edition の基本機能は同じです。このマニュアルで説明されているオブジェクト・リレーショナル機能の中核は、Oracle8i Server の一部です。その他の拡張機能は Enterprise Edition のみで使用可能であり、そのうちの *interMedia* 製品群などの機能はオプションです。

関連マニュアル

PL/SQL は、SQL に対してオラクル社が開発したプロシージャ型拡張機能です。この PL/SQL について学習し、この高水準プログラミング言語の詳細が必要な場合は、『Oracle8i PL/SQL ユーザーズ・ガイドおよびリファレンス』を参照してください。

アプリケーション開発の一般的な情報については、『Oracle8i アプリケーション開発者ガイド 基礎編』を参照してください。

Java を介した Oracle のオブジェクト・リレーショナル機能の使用については、『Oracle8i JDBC 開発者ガイドおよびリファレンス』および『Oracle8i Java ストアド・プロシージャ開発者ガイド』を参照してください。

Oracle Call Interface (OCI) については、『Oracle8i コール・インタフェース・プログラマーズ・ガイド』を参照してください。

Oracle Server にアクセスする第3世代言語 (3GL) アプリケーションを作成するには、OCI を使用できます。

オラクル社は、プリコンパイラの Pro* シリーズも提供しています。これを使用することで、ご使用のアプリケーション・プログラムに SQL および PL/SQL を組み込むことができます。埋込み SQL を取り込んだ 3GL アプリケーション・プログラムを、Ada、C、C++、COBOL または FORTRAN で作成する場合は、該当するプリコンパイラのマニュアルを参照してください。たとえば、C または C++ でプログラムする場合は、『Oracle8i Pro*C/C++ プリコンパイラ・プログラマーズ・ガイド』を参照してください。

Oracle Developer/2000 は、フォーム作成プログラム、レポート作成ツール、PL/SQL 用のデバッグ環境などを含む数種類のツールを提供するコオペラティブ開発環境です。Developer/2000 を使用する場合は、該当する Oracle Tools のドキュメントを参照してください。

SQL の詳細は、『Oracle8i SQL リファレンス』および『Oracle8i 管理者ガイド』を参照してください。Oracle の基本概念は、『Oracle8i 概要』を参照してください。

このマニュアルの構成

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

第1章「Oracle オブジェクトの概要」

Oracle オブジェクトを使用する際に必要な、バックグラウンドの概念および用語を説明します。

第2章「Oracle オブジェクトの管理」

オブジェクトおよびオブジェクト型を使用した、基本操作の実行方法について説明します。

第3章「Oracle プログラム環境のオブジェクト・サポート」

SQL、PL/SQL、Oracle Call Interface (OCI)、Pro*C/C++、Oracle Objects For OLE、Java、JDBC および Oracle SQLJ におけるオブジェクト・リレーショナル機能について説明します。この章の内容は、教育および計画向けの高度なものです。次の各章で、オブジェクト・リレーショナル機能の使用方法について詳細に説明します。

第4章「オブジェクト・モデルのリレーショナル・データへの適用」

オブジェクト・ビューについて説明します。オブジェクト・ビューを使用すると、基礎となるリレーショナル・スキーマを変更することなくオブジェクト指向アプリケーションを開発できます。

第5章「Oracle オブジェクトの設計上の考慮点」

Oracle のオブジェクト指向モデルの実装法およびパフォーマンス特性について説明します。

第6章「Oracle オブジェクトの高度なトピック」

オブジェクト指向アプリケーションをスケールアップする場合に、記憶域およびパフォーマンスを管理するために必要な機能について説明します。

第7章「Oracle オブジェクトを使用したプログラミングに関する FAQ」

オブジェクト指向プログラミングを使用したプログラミングを始める場合、または他のデータベース・システムやオブジェクト指向言語などの経験者が Oracle に取り組む場合に役立つヒントを提供します。

第8章「オブジェクト・リレーショナル機能を使用したサンプル・アプリケーション」

リレーショナル・プログラムを、オブジェクト指向プログラム、スキーマなどのすべてのものに再作成する方法について例を示します。

このマニュアルの表記規則

このマニュアルで使用する表記規則および本文の形式は、次のとおりです。

[]

大カッコは、任意に選択する項目を示します。カッコは入力しないでください。

{ }

中カッコは、複数の項目を囲み、その中の 1 つのみが必須であることを示します。

|

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

...

コード例の省略記号は、その説明内容に関係のないコードを省略していることを示します。

固定幅フォント

固定幅フォントは、SQL または C のコード例を示します。

イタリック

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

大文字

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

このマニュアルでは、ある情報に対して読者の注意を促すために、特別な形式を使用しています。太字テキストの見出しで始まる段落は、特別な意味を持つ場合があります。次の見出しで始まる各段落では、それぞれ特別な情報を示します。

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

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

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

Oracle オブジェクトの概要

C++、Java、Perl などの最新言語でのプログラミング経験があれば、オブジェクト指向プログラミングの概念は理解していることでしょう。Oracle は、これらの言語における設計および問題解決の技術をデータベース・アプリケーション開発に利用できるように、多くのオブジェクト指向機能を提供しています。

複合型の不足、階層構造を持つデータをフラット化し、主キーおよび外部キーを利用しての表へのマッピング、状況が変わるごとに大量のアプリケーション論理をコピーおよび適用する必要があることは、データベース・プログラマにとってわずらわしい問題でした。Oracle のオブジェクト指向機能は、このような問題の解決に役立ちます。

このマニュアルは、データベース・アプリケーション開発に慣れていることを前提としています。1 つ以上のプログラミング言語を SQL データ定義言語 (DDL) および SQL データ操作言語 (DML) とともに使用して、通常のすべてのデータベース処理を実行できる必要があります。

Oracle のオブジェクト・リレーショナル機能を初めて使用する場合や、この章の内容を完全に理解できない場合は、[第 7 章「Oracle オブジェクトを使用したプログラミングに関する FAQ」](#)を参照してください。

Oracle オブジェクトの基本

この項では、Oracle のオブジェクト・リレーショナル機能を利用する際に必要な概念および用語の概要を示します。

オブジェクト・リレーショナル・モデル

オブジェクト・リレーショナル・モデルは、既存のアプリケーションで使用されている既存のリレーショナル機能を残したまま、オブジェクト指向機能をデータベースに導入するための進化した方法です。この項以降では、従来の優れた機能を活かしながら、オブジェクト指向機能がどのように Oracle8i に統合されているかを説明しています。

オブジェクト・リレーショナル・データベース・システムと第3世代言語

データベースをまったく使用せずに、第3世代言語（3GL）を使用してオブジェクト指向アプリケーションを作成しないのはなぜでしょうか。

第1に、RDBMSでは、既存の機能を基に新しい機能を構築します。はじめから構築し直すわけではありません。

第2に、3GLを使用した情報管理には、持続性がないという問題があります。持続性があったとしても、必要なパフォーマンスを得るために同一のアドレス空間にアプリケーション論理およびデータ論理を置くため、セキュリティを犠牲にしまいます。持続性およびセキュリティの両方が基本要件であるRDBMSのユーザーにとっては、どちらの要件も欠かせません。

このため、リレーショナル・モデルを使用しているアプリケーション開発者は、ある形式でSQLにマッピングすることによって複合型をシミュレートする必要があります。この方法では、効率が悪いだけでなく、実装に関する深刻な問題があります。次の処理を行う必要があります。

- 書込み時のアプリケーション論理からデータ論理への変換
- 読み込み時のデータ論理からアプリケーション論理への変換

これでは、クライアントとサーバーのアドレス空間の間の通信量が大量になるため、パフォーマンスが低下します。クライアントおよびサーバーが別々のマシン上にある場合、ネットワーク・ラウンドトリップで多くのオーバーヘッドが発生する可能性があります。

オブジェクト・リレーショナル（O-R）・テクノロジーによって、これらの問題を解決できます。このマニュアルでは、その方法を説明します。

オブジェクト型

オブジェクト型は、実世界のエンティティ（発注書など）をアプリケーション・プログラムが処理できるように抽象化したものです。オブジェクト型は、Java および C++ のクラスと似ています。

オブジェクト型は、テンプレートおよびそのテンプレートを適用する構造体としてのオブジェクトとみなすことができます。オブジェクト型は、多くの異なるデータ構造を表すことができます。明細項目、イメージ、空間データなどです。

オブジェクト型はスキーマ・オブジェクトであり、他のスキーマ・オブジェクトと同様に管理制御されます（第2章「[Oracle オブジェクトの管理](#)」を参照）。

オブジェクト型を使用すると、オブジェクト・モデルをフラット化してリレーショナル表およびリレーショナル列に割り当てるかわりに、データベース・スキーマに直接マップすることができます。オブジェクト型によって、関連する個々のデータを1つの単位にまとめ、データの動作をデータ自体とともに格納できます。アプリケーション・コードは、データをオブジェクトとして取り出し、操作できます。

オブジェクト型は、次の3種類のコンポーネントを含むスキーマ・オブジェクトです。

- 名前 - スキーマ内でオブジェクト型を一意に識別します。
- 属性 - 実世界のエンティティ構造および状態をモデル化します。属性は、組込み型またはオブジェクト型にすることができます。
- メソッド - 実世界のエンティティで実行可能な動作の疑似操作を実装するファンクションまたはプロシージャです。

オブジェクト

オブジェクト型の変数を作成すると、結果はオブジェクトになります。オブジェクトには、それ自体の型をベースにした属性およびメソッドがあります。オブジェクトは具体的なものであるため、値をその属性に割り当て、そのメソッドをコールできます。

メソッド

オブジェクト型のメソッドは、アプリケーションでコールされ、オブジェクトの動作をモデル化するファンクションまたはプロシージャです。PL/SQL または Java で作成されたメソッドはデータベースに格納され、頻繁にコールされるデータ処理集中型のプロシージャおよびショート・プロシージャに使用されます。C などその他の言語のプロシージャは外部に格納され、あまりコールされない計算処理集中型のプロシージャに使用されます。

オブジェクト型のメソッドは、メンバー、静的および比較の3つの大きなカテゴリに分けられます。

メンバー・メソッドは、暗黙的な SELF パラメータを常に持つファンクションまたはプロシージャです。このため、特定のオブジェクトの属性を使用して実行することができます。起動するには、ドット表記法 `OBJECT_VARIABLE.METHOD()` を使用します。メンバー・メソッドは、オブザーバ・メソッドまたはミューテータ・メソッドの作成に役立ちます。その場合、処理は特定のオブジェクトに影響し、どのパラメータも渡す必要がありません。

静的メソッドは、暗黙的な SELF パラメータを持たないファンクションまたはプロシージャです。このメソッドは、`TYPE_NAME.METHOD()` のように型名でメソッドを修飾することで起動できます。静的メソッドは、オブジェクト状態よりグローバル・データを処理するプロシージャ、またはオブジェクトにかかわらず同じ値を戻すファンクションに有効です。

比較メソッドは、オブジェクトのインスタンスを比較し、ソートおよび IF/THEN 論理を可能にします。

たとえば、PURCHASE_ORDER が GET_VALUE という名前のメソッドを持つとします。各発注情報オブジェクトは、それぞれの GET_VALUE メソッドを持ちます。X_OBJ および Y_OBJ が発注情報オブジェクトを保持する PL/SQL 変数で、W_NUM および Z_NUM が数を保持する変数の場合、次の 2 つの文でオブジェクトから値を取り出すことができます。

```
w_num = x_obj.get_value();
z_num = y_obj.get_value();
```

両方のオブジェクトはどちらも同じ型であり、GET_VALUE メソッドを持ちます。メソッド・コールは、それ自体の一連のデータ X_OBJ および Y_OBJ の属性上で動作するため、どのようなパラメータも必要としません。このように自己参照的スタイルでメソッドを起動すると、メソッドは、メソッドのコール先であるオブジェクトごとに適切なデータを使用します。

コンストラクタ・メソッド すべてのオブジェクト型は、システム定義のコンストラクタ・メソッド、つまり、新しいオブジェクトを作成しその属性の値を設定するメソッドを持っています。コンストラクタ・メソッドの名前は、オブジェクト型の名前です。このメソッドのパラメータは、オブジェクト型の属性の名前および型です。コンストラクタ・メソッドはファンクションであり、新しいオブジェクトをその値として戻します。

次に例を示します。

```
purchase_order(
    1000376,
    person ("John Smith","1-800-555-1212"),
    NULL )
```

この式は、次の属性を持つ発注情報オブジェクトを表します。

```
id          1000376
contact     person("John Smith","1-800-555-1212")
lineitems   NULL
```

person ("John Smith", "1-800-555-1212") という式は、オブジェクト型 PERSON のコンストラクタ・ファンクションを起動します。このファンクションが戻すオブジェクトが、発注書の CONTACT 属性になります。

NULL オブジェクトおよび NULL 属性については、2-2 ページの「[NULL オブジェクトおよび属性](#)」を参照してください。

比較メソッド Oracle には、指定された組込み型の 2 つのデータ項目を比較する機能、およびデータが他のものと比べて大きいか、等しいか、小さいかを判断する機能があります。オブジェクト型の 2 項目を比較するには、その型の作成者は、マップ・メソッドまたはオーダー・メソッドを使用して、型の順序関係を定義する必要があります。

マップ・メソッドは、比較およびソートに使用できる組込み型の単一の値を作成します。たとえば、RECTANGLE というオブジェクト型を定義した場合、マップ・メソッド AREA は HEIGHT 属性と WIDTH 属性を掛けた値を戻します。この後、Oracle は 2 つの四角形をその面積で比較することができます。

オーダー・メソッドはさらに一般的です。このメソッドはそれ自体の内部論理を使用して、指定されたオブジェクト型の 2 つのオブジェクトを比較し、順序関係をコード化する値を戻します。最初の項目が小さい場合は -1、等しい場合は 0、最初の項目が大きい場合は 1 になります。

たとえば、オーダー・メソッドでは、ある地点からの距離に基づいた住所の集合をソートできます。また、個々の値の比較より複雑な操作もできます。

オブジェクト型を定義するときに、マップ・メソッドまたはオーダー・メソッドを指定できます。ただし、両方を指定することはできません。オブジェクト型に比較メソッドがなければ、Oracle は 2 つのオブジェクト間で値が大きい、小さいかという関連を判断できません。ただし、この型の 2 つのオブジェクトが等しいかどうかの判断を試みることはできません。

Oracle は、比較メソッドを持たない型の 2 つのオブジェクトを、対応する属性を比較することで次のように比較します。

- すべての属性が NULL 以外で等しい場合、2 つのオブジェクトは等しいとみなされます。
- 2 つのオブジェクトの属性で、等しくない NULL 以外の値を持つものがあれば、2 つのオブジェクトは等しくないとみなされます。
- その他の場合、オブジェクトは等しくないとみなされます。

システムはスカラー値比較を非常に効率的に実行できるため、ユーザー定義関クションのコールは、カーネルで実装される関クションのコールより遅いということを考えると、オーダー・メソッドを使用するオブジェクトのソートは、マップ・メソッドが戻すマップされたスカラー値のソートと比べてかなり遅くなります。

オブジェクト表

オブジェクト表は特別な種類の表であり、各行が 1 つのオブジェクトを表します。

たとえば、次の文は、PERSON 型のオブジェクトに対するオブジェクト表を定義します。

```
CREATE TABLE person_table OF person;
```

Oracle では、この表を次の 2 つの方法で表示することができます。

- 各行が PERSON オブジェクトである単一列表。ここでは、オブジェクト指向操作を実行できます。
- オブジェクト型 PERSON の各属性（NAME および PHONE）が列を占めている複数列表。ここでは、リレーショナル操作を実行できます。

たとえば、次の指示を実行できます。

```
INSERT INTO person_table VALUES (  
    "John Smith",  
    "1-800-555-1212" );  
  
SELECT VALUE(p) FROM person_table p  
    WHERE p.name = "John Smith";
```

最初の指示で、PERSON オブジェクトを複数列表として PERSON_TABLE に挿入します。次の指示で、単一列表として PERSON_TABLE から選択します。

行オブジェクトおよび列オブジェクト オブジェクト表の中で、完全な行を占めるオブジェクトを行オブジェクトといいます。より大きい行の中で列を占めるオブジェクト、または他のオブジェクトの属性であるオブジェクトを、列オブジェクトといいます。

オブジェクト・ビュー

オブジェクト・ビュー（[第4章「オブジェクト・モデルのリレーショナル・データへの適用」](#)を参照）は、オブジェクト・リレーショナル機能を使用して関連するデータにアクセスする方法です。基礎となるリレーショナル・スキーマを変更することなく、オブジェクト指向アプリケーションを開発できます。

REF データ型

REF は、行オブジェクトを指す論理的なポインタで、Oracle の組込みデータ型です。REF および REF のコレクションは、オブジェクト間の対応、特に、多対 1 関連をモデル化します。これによって、外部キーの必要性を削減できます。REF は、オブジェクト間をナビゲートする簡単なメカニズムを提供します。ポインタに続いてドット表記法を使用することができます。必要な場合には Oracle が結合し、必要でなければ結合を解除することもできます。

REF は、REF が参照するオブジェクトを検査または更新するために使用できます。また、REF が参照するオブジェクトのコピーを取得するためにも使用できます。REF を変更して、同じオブジェクト型の異なるオブジェクトを指すようにすることができます。また、REF に NULL 値を割り当てることもできます。

SCOPED REF 列型、コレクション要素またはオブジェクト型属性を REF として宣言する場合、指定されたオブジェクト表への参照のみを含むように制約することができます。このような REF を SCOPED REF といいます。SCOPED REF では、有効範囲付きでない REF に比べて、少ない記憶領域でより効率的なアクセスが可能です。

参照先がない REF オブジェクトの削除または権限の変更によって、REF が識別したオブジェクトを使用禁止にすることもできます。このような REF を、参照先がない REF といいます。Oracle の SQL は、この条件に関して REF をテストできる述語 (IS DANGLING) を提供します。

REF の参照解除 REF が参照するオブジェクトにアクセスすることを、REF を参照解除するといいます。Oracle は、これを実行する Deref 演算子を提供します。

参照先がない REF を参照解除すると、NULL オブジェクトを戻します。

また、Oracle では、REF の暗黙の参照解除もできます。次に例を示します。

```
CREATE TYPE person AS OBJECT (  
    name    VARCHAR2(30),  
    manager REF person );
```

X が PERSON 型のオブジェクトを表すとすると、次の SQL 式は、

```
x.manager.name;
```

ポインタを X という人から X の上司に移し、上司の名前を取り出します。(SQL では、このように REF に続けることができますが、PL/SQL ではできません。)

REF の取得 オブジェクト表からオブジェクトを選択して REF 演算子に適用することで、その行オブジェクトに対する REF を取得することができます。たとえば、次のように識別番号 1000376 を使用して発注書に対する REF を取得することができます。

```
DECLARE OrderRef REF to purchase_order;  
  
SELECT REF(po) INTO OrderRef  
    FROM purchase_order_table po  
    WHERE po.id = 1000376;
```

問合せは確実に 1 つの行を戻す必要があります。

オブジェクトおよび REF の記憶域の詳細は、[第 5 章「Oracle オブジェクトの設計上の考慮点」](#)を参照してください。

コレクション

1 対多関連をモデル化するために、Oracle は 2 つのコレクション・データ型、VARRAY および NESTED TABLE をサポートしています。たとえば、発注書は明細項目の任意の数値を持っているため、この明細項目をコレクションに加えることができます。

- VARRAY は要素の最大数値を持ちます。ただし、上限は変更できません。要素の順序は定義されています。VARRAY は不透明なオブジェクト（RAW または BLOB）として格納されます。
- NESTED TABLE はいくつもの要素を持ち、また、通常の表と同様に選択、挿入、削除などを行うことができます。要素の順序は定義されません。NESTED TABLE は記憶表に格納され、各要素は記憶表の行にそれぞれマップされます。

順序に従って要素をループする必要がある場合、項目の固定数値のみを格納する必要がある場合、または、値のコレクション全体を 1 つの値として取り出して操作する必要がある場合は、VARRAY を使用してください。

コレクション上で効率的な問合せを実行する必要がある場合、任意の数の要素を扱う必要がある場合、または、大量の挿入 / 更新 / 削除を行う必要がある場合は、NESTED TABLE を使用してください。非常に大きいコレクションからサブセットのみを取り出す場合は、コレクションを NESTED TABLE としてモデル化し、結果セットに対してロケータを取り出すことができます。

たとえば、発注情報オブジェクトは明細項目の NESTED TABLE を持ち、四角形オブジェクトは 4 つの座標を持つ VARRAY を含むことができます。

VARRAY または NESTED TABLE の作成

コンストラクタ・メソッドをコールすることで、コレクション型のオブジェクトを作成できます。コンストラクタ・メソッドの名前は型の名前であり、その引数は新しいコレクション要素をカンマで区切ったリストです。

空のリストでコンストラクタ・メソッドをコールすると、その型の空のコレクションを作成できます。空のコレクションは、NULL のコレクションではありません。

VARRAY

配列とは、順序付けられたデータ要素の集合です。指定された配列のすべての要素は、同じデータ型の要素です。各要素は、配列における要素の位置に対する番号を持ちます。

配列における要素の数は、配列のサイズです。Oracle は、配列を可変サイズにすることができ、これを VARRAY といいます。配列型を宣言する場合は、最大サイズを指定する必要があります。

たとえば、次の文で配列型を宣言するとします。

```
CREATE TYPE prices AS VARRAY(10) OF NUMBER(12,2);
```

PRICES 型の VARRAY は最大 10 個の要素を持ちます。各要素のデータ型は NUMBER(12,2) です。

配列型の作成では、領域は割り当てられません。データ型を定義し、次のものとして使用できるようにします。

- リレーショナル表の列のデータ型
- オブジェクト型属性
- PL/SQL 変数、パラメータまたはファンクション戻り値の型

通常、VARRAY はインラインに、つまり、その行にある他のデータと同じ表領域に格納されます。VARRAY が十分に大きい場合、Oracle はこれを BLOB として格納します (2-15 ページの「[オブジェクト型のインポート / エクスポート / ロード](#)」を参照)。

参照： VARRAY の詳細は、5-13 ページの「[VARRAY の記憶域上の考慮点](#)」を参照してください。

NESTED TABLE

NESTED TABLE は順序付けされないデータ要素の集合であり、これらのすべての要素は同じデータ型を持ちます。NESTED TABLE は単一列を持ち、この列の型は組込型またはオブジェクト型です。NESTED TABLE の列がオブジェクト型の場合、表は、オブジェクト型の各属性に対してそれぞれ列がある複数列表としても表示できます。

たとえば、発注書の例では、次の SQL 文でオブジェクト型 "lineitem" を各要素のデータ型とする NESTED TABLE 型 "lineitem_table" を宣言します。

```
CREATE TYPE lineitem_table AS TABLE OF lineitem;
```

NESTED TABLE 型の定義では、領域は割り当てられません。型を定義し、次のものとして使用できるようにします。

- リレーショナル表の列のデータ型
- オブジェクト型属性
- PL/SQL 変数、パラメータまたはファンクション戻り型

NESTED TABLE 型がリレーショナル表の列の型として、またはオブジェクト表の基礎となるオブジェクト型の属性として表示される場合、Oracle は NESTED TABLE のすべてのデータを単一表に格納し、この表に含まれるリレーショナル表またはオブジェクト表に対応付けます。たとえば、次の文では、オブジェクト型 PURCHASE_ORDER に対してオブジェクト表を定義します。

```
CREATE TABLE purchase_order_table OF purchase_order
  NESTED TABLE lineitems STORE AS lineitems_table;
```

2 行目で、PURCHASE_ORDER_TABLE におけるすべての PURCHASE_ORDER オブジェクトの LINEITEMS 属性の記憶表として、LINEITEMS_TABLE が指定されます。

ネスト・カーソルを使用すると、簡単に NESTED TABLE の要素に個別にアクセスできます。

参照： ネスト・カーソルの詳細は、『Oracle8i SQL リファレンス』を参照してください。NESTED TABLE の使用については、5-14 ページの「[NESTED TABLE](#)」を参照してください。

継承

継承は、関連オブジェクトのグループ用に一般化された属性および動作を含むオブジェクトを作成するオブジェクト指向開発において使用される技法です。さらに一般的なオブジェクト型は、スーパータイプといわれます。スーパータイプから継承される特別なオブジェクト型を、サブタイプといいます。

継承の一般的な例は、Person および Employee の例です。人の集合には、従業員および非従業員が含まれます。より一般的な例 Person はスーパータイプで、特殊ケース Employee はサブタイプです。また、別の例では、Vehicle はスーパータイプで、Car および Truck はサブタイプです。

オブジェクト指向モデルの例

オブジェクト型の集合を定義する方法の例を示します。

オブジェクト型は PERSON、LINEITEM、LINEITEM_TABLE および PURCHASE_ORDER です。

NAME、PHONE、ITEM_NAME などが、対応するオブジェクト型の属性になります。属性 CONTACT はオブジェクトであり、属性 LINEITEMS は NESTED TABLE です。

LINEITEM_TABLE は、各行が LINEITEM 型のオブジェクトである NESTED TABLE 型です。

```
CREATE TYPE person AS OBJECT (
  name      VARCHAR2(30),
  phone     VARCHAR2(20) );

CREATE TYPE lineitem AS OBJECT (
  item_name VARCHAR2(30),
```

```
quantity    NUMBER,
unit_price  NUMBER(12,2) );

CREATE TYPE lineitem_table AS TABLE OF lineitem;

CREATE TYPE purchase_order AS OBJECT (
    id        NUMBER,
    contact   person,
    lineitems lineitem_table,

    MEMBER FUNCTION
    get_value RETURN NUMBER );
```

これは、簡略化された例です。この例では、GET_VALUE メソッドの本体の宣言は示されていません。GET_VALUE メソッドの本体は、CREATE OR REPLACE TYPE BODY 文を使用して指定します。

オブジェクト型を定義しても、記憶域は割り当てられません。

NUMBER または VARCHAR2 型を使用できる場所のほとんどにおいて、SQL 文で LINEITEM、PERSON または PURCHASE_ORDER を使用できます。

たとえば、連絡先を記録するリレーショナル表を定義できます。

```
CREATE TABLE contacts (
    contact   person
    date      DATE );
```

CONTACTS 表は、オブジェクト型で定義された列を1つ持つリレーショナル表です。リレーショナル表の列を占めるオブジェクトを、列オブジェクトといいます（1-6 ページの「[行オブジェクトおよび列オブジェクト](#)」を参照）。

Oracle オブジェクトの管理

この章では、Oracle がデータベースの他の部分と関連してどのように機能し、どのように DML 操作および DDL 操作を実行するかについて説明します。内容は次のとおりです。

- [オブジェクト型および参照の使用](#)
- [コレクションの使用](#)
- [オブジェクト型およびそれらのメソッドの権限](#)
- [依存性および不完全な型](#)
- [オブジェクト型のインポート / エクスポート / ロード](#)

オブジェクト型および参照の使用

この項では、次のオブジェクト型および参照について説明します。

- [NULL オブジェクトおよび属性](#)
- [オブジェクトおよびコレクションのデフォルト値](#)
- [オブジェクト表の制約](#)
- [オブジェクト表および NESTED TABLE の索引](#)
- [オブジェクト表のトリガー](#)
- [REF 型の列および属性のルール](#)
- [名前解決](#)

NULL オブジェクトおよび属性

表列、オブジェクト、オブジェクト属性、コレクション、コレクション要素など、オブジェクトに対応付けられたものの多くには、NULL を指定できます。つまり、その項目は NULL に初期化されるか、または初期化されないかのいずれかです。通常、その項目の値はわかりませんが、後で使用可能になる場合があります。

自分自身の値が NULL であるオブジェクトは、アトミック NULL といわれます。オブジェクトの属性を NULL に指定することもできます。この 2 つの NULL の使用方法は異なります。オブジェクトのすべての属性が NULL の場合でも、属性の変更およびメソッドの呼出しを実行できます。オブジェクトがアトミック NULL の場合は、何もできません。

たとえば、次のように定義された CONTACTS 表について考えます。

```
CREATE TYPE person AS OBJECT (  
    name          VARCHAR2(30),  
    phone         VARCHAR2(20) );  
  
CREATE TABLE contacts (  
    contact      person  
    date        DATE );
```

次の文は、

```
INSERT INTO contacts VALUES (  
    person (NULL, NULL),  
    '24 Jun 1997' );
```

次の文とは異なる結果を戻します。

```
INSERT INTO contacts VALUES (  
    NULL,  
    '24 Jun 1997' );
```

両方の場合とも、Oracle は CONTACTS に新しい行用の領域を割り当て、DATE 列を指定された値に設定します。最初の場合では、Oracle はオブジェクト用の領域を PERSON 列に割り当て、その各属性に NULL を設定します。2 番目の場合では、PERSON 列を NULL に設定し、オブジェクト用の領域は割り当てません。

NULL 値のチェックは省略できる場合があります。リレーショナル表の行または行オブジェクト自体を NULL に初期化することはできません。オブジェクトの NESTED TABLE は、NULL の値を持つ要素を含むことができません。

NESTED TABLE または VARRAY には NULL を指定できるため、その状態を処理する必要があります。NULL コレクションと何も要素を持たない空のコレクションは異なります。

オブジェクトおよびコレクションのデフォルト値

リレーショナル表の列にオブジェクト型またはコレクション型を宣言した場合、DEFAULT 句を含めることができます。明示的に列に値を指定しない場合には、この DEFAULT 句で指定した値が使用されます。DEFAULT 句には、そのオブジェクトまたはコレクションのコンストラクタ・メソッドのリテラル起動を含める必要があります。

コンストラクタ・メソッドのリテラル起動は、すべての引数が定数、またはコンストラクタ・メソッドの別のリテラル起動です。変数または関数は使用できません。

次に例を示します。

```
CREATE TYPE person AS OBJECT (
  id          NUMBER
  name        VARCHAR2(30),
  address     VARCHAR2(30) );

CREATE TYPE people AS TABLE OF person;
```

次に、NESTED TABLE 型である PEOPLE のコンストラクタ・メソッドのリテラル起動を示します。

```
people ( person(1, 'John Smith', '5 Cherry Lane'),
         person(2, 'Diane Smith', NULL) )
```

次の例は、コンストラクタ・メソッドのリテラル起動を使用してデフォルトを指定します。

```
CREATE TABLE department (
  d_no      CHAR(5) PRIMARY KEY,
  d_name    CHAR(20),
  d_mgr     person DEFAULT person(1, 'John Doe', NULL),
  d_emps    people DEFAULT people() )
NESTED TABLE d_emps STORE AS d_emps_tab;
```

PEOPLE() という用語が、空の NESTED TABLE PEOPLE のコンストラクタ・メソッドのリテラル起動であることに注意してください。

オブジェクト表の制約

他の表と同様に、オブジェクト表にも制約を定義できます。

列オブジェクトのリーフ・レベル・スカラー属性には、参照範囲が制限されていない REF を除き、制約を定義できます。

次に一例を示します。

最初の例は、オブジェクト表 PERSON_EXTENT の SSNO 列に PRIMARY KEY 制約を定義します。

```
CREATE TYPE location (  
    building_no NUMBER,  
    city          VARCHAR2(40) );  
  
CREATE TYPE person (  
    ssno          NUMBER,  
    name          VARCHAR2(100),  
    address       VARCHAR2(100),  
    office        location );  
  
CREATE TABLE person_extent OF person (  
    ssno          PRIMARY KEY );
```

次の例の DEPARTMENT 表には、列のデータ型が前述の例で定義されたオブジェクト型 LOCATION である列があります。この例では、表の DEPT_LOC 列に表示される LOCATION オブジェクトのスカラー属性に制約を定義します。

```
CREATE TABLE department (  
    deptno        CHAR(5) PRIMARY KEY,  
    dept_name     CHAR(20),  
    dept_mgr      person,  
    dept_loc      location,  
    CONSTRAINT dept_loc_cons1  
        UNIQUE (dept_loc.building_no, dept_loc.city),  
    CONSTRAINT dept_loc_cons2  
        CHECK (dept_loc.city IS NOT NULL) );
```

オブジェクト表および NESTED TABLE の索引

オブジェクト表の列や属性、および NESTED TABLE の記憶表には、他の表と同様に、索引を定義できます。

次の例で示すように、列オブジェクトのリーフ・レベル・スカラー属性に索引を定義できます。SCOPED REF の場合、REF 型の列または属性にのみ索引を定義できます。

ここで、DEPT_ADDR は列オブジェクト、CITY は索引を作成する DEPT_ADDR のリーフ・レベル・スカラー属性です。

```
CREATE TABLE department (  
    deptno        CHAR(5) PRIMARY KEY,  
    dept_name     CHAR(20),  
    dept_mgr      person,  
    dept_loc      location,  
    CONSTRAINT dept_loc_cons1  
        UNIQUE (dept_loc.building_no, dept_loc.city),  
    CONSTRAINT dept_loc_cons2  
        CHECK (dept_loc.city IS NOT NULL) );
```

```

deptno      CHAR(5) PRIMARY KEY,
dept_name   CHAR(20),
dept_addr   address );

CREATE INDEX i_dept_addr1
            ON department (dept_addr.city);

```

Oracle が索引定義で列名を指定するところには、行オブジェクトのスカラー属性も指定できます。

オブジェクト表のトリガー

オブジェクト表には、他の表と同様に、トリガーを定義できます。NESTED TABLE の記憶表に、トリガーは指定できません。

トリガーを利用した LOB の変更はできません。トリガーをオブジェクト型とともに使用する場合は他にはありません。

次の例は、前述の項で定義した PERSON_EXTENT 表にトリガーを定義します。

```

CREATE TABLE movement (
    ssno          NUMBER,
    old_office    location,
    new_office    location );

CREATE TRIGGER trig1
    BEFORE UPDATE
        OF office
        ON person_extent
    FOR EACH ROW
        WHEN new.office.city = 'REDWOOD SHORES'
    BEGIN
        IF :new.office.building_no = 600 THEN
            INSERT INTO movement (ssno, old_office, new_office)
                VALUES (:old.ssno, :old.office, :new.office);
        END IF;
    END;

```

REF 型の列および属性のルール

Oracle では、REF 型の列または属性は、SCOPE 句または参照制約句を使用して、制約なしにすることも制約付きにすることもできます。REF 列が制約なしの場合、対応するオブジェクト型のオブジェクト表に含まれているすべての行オブジェクトに対するオブジェクト参照を格納できます。

Oracle は、そのような列に格納されたオブジェクト参照が、有効な既存の行オブジェクトを指すことを保証しません。したがって、REF 列には、既存の行オブジェクトを指していないオブジェクト参照が含まれていることがあります。そのような REF 値は、参照先がない参照として参照されます。現在、Oracle では、制約なし REF 列に主キー・ベースのオブジェクト識別子を含むオブジェクト参照を格納することはできません。

REF 列には、参照範囲を特定のオブジェクト表に制限するような制約も指定できます。SCOPE 制約が付いた列に格納されたすべての REF 値は、SCOPE 句で指定された表の行オブジェクトを指します。ただし、REF 値には参照先がない場合があります。

REF 列には、外部キーの指定に似た参照制約を指定することもできます。参照制約のルールは、そのような列に適用されます。つまり、このような列に格納されたオブジェクト参照は、指定されたオブジェクト表内の有効な既存の行オブジェクトを指す必要があります。

REF 列には、UNIQUE 制約または PRIMARY KEY 制約を指定できません。ただし、NOT NULL 制約は指定できます。

名前解決

Oracle SQL では、いくつかのリレーショナル操作で表名を省略できます。たとえば、ASSIGNMENT が PROJECTS の列で、TASK が DEPTS の列の場合、次のように記述できます。

```
SELECT *  
FROM projects  
WHERE EXISTS  
    (SELECT * FROM depts  
     WHERE assignment = task);
```

Oracle は、各列がどの表に属しているかを判断します。

メンテナンスしやすいように、表名または表別名で列名を修飾することもできます。

```
SELECT * FROM projects WHERE EXISTS  
    (SELECT * FROM depts WHERE projects.assignment = depts.task);
```

```
SELECT * FROM projects pj WHERE EXISTS  
    (SELECT * FROM depts dp WHERE pj.assignment = dp.task);
```

オブジェクト・リレーショナル機能で表別名を指定する必要がある場合もあります。

表別名が必要な場合

修飾されていない名前を使用すると、問題が起きる場合があります。2 番目の表 (DEPTS) に ASSIGNMENT 列を追加した後、問合せの変更をしなかった場合、Oracle は自動的に問合せを再コンパイルし、この新しいバージョンの問合せで DEPTS 表の ASSIGNMENT 列が使用されます。この状況を内側獲得といいます。

内側獲得および SQL 文における同様の誤解析を回避するために、表別名を使用して、メソッドまたはオブジェクトの属性への参照を修飾してください。

REF を介した属性参照にも同じ要件が適用されます。この要件を獲得回避規則といいます。

たとえば、次の文について考えます。

```
CREATE TYPE person AS OBJECT (ssno VARCHAR(20));
CREATE TABLE ptab1 OF person;
CREATE TABLE ptab2 (c1 person);
```

これらは、オブジェクト型 PERSON および 2 つの表を定義します。1 つ目の表は、オブジェクト型 PERSON のオブジェクト表です。2 番目の表は、PERSON 型の列を 1 つ持ちます。

ここで、次の問合せについて考えてみます。

```
SELECT      ssno FROM ptab1    ; --Correct
SELECT    c1.ssno FROM ptab2    ; --Wrong
SELECT p.c1.ssno FROM ptab2 p ; --Correct
```

- 最初の SELECT 文では、SSNO は PTAB1 の列の名前です。これはリレーショナル問合せと判断されるため、さらに修飾は必要ありません。
- 2 番目の SELECT 文では、SSNO は C1 という名前の列にある PERSON オブジェクトの属性の名前です。この参照には、3 番目の SELECT 文に示すように、表別名が必要です。

オブジェクト属性への参照は、表名がスキーマ名によってさらに修飾される場合でも、表名ではなく表別名を使用して修飾する必要があります。

たとえば、次の式は SCOTT スキーマ、PROJECTS 表、ASSIGNMENT 列およびその列の DUEDATE 属性を参照しようとしています。ただし、PROJECTS が表名で別名ではないため、許可されません。

```
scott.projects.assignment.duedate
```

表別名は、問合せ全体を通して一意である必要があります。問合せに適切に表示されるスキーマ名は異なる名前である必要があります。

注意： 列にオブジェクト型が含まれているかどうかにかかわらず、すべての UPDATE 文、DELETE 文、SELECT 文および副問合せに表別名を定義し、それらを使用して列参照を修飾することをお勧めします。

引数なしのメソッド・コール

メソッドは、ファンクションまたはサブルーチンです。それらを起動する適切な構文では、メソッド名の後のコールする引数をカッコで囲みます。あいまいさを回避するために、引数を持たないメソッドのコールには、空のカッコを付ける必要があります。

たとえば、表 TB にはオブジェクト型 T の列 C があり、T には引数なしのメソッド m がある場合、次の問合せ構文が適切です。

```
SELECT p.c.m() FROM tb p;
```

これは、引数のないコールのカッコはオプションである PL/SQL ファンクションおよびプロシージャのルールとは異なります。

コレクションの使用

この項では、次のコレクションの使用方法について説明します。

- [コレクションの問合せ](#)
- [コレクション・アンネスティング](#)
- [コレクションの DML](#)

コレクションの問合せ

Oracle8i では、コレクション列の問合せに TABLE 式が使用されることがあります。たとえば、表 (EMPLOYEES) の NESTED TABLE 列 (PROJECTS) の問合せは、次のようになります。

```
SELECT * FROM TABLE(SELECT t.projects FROM employees t WHERE t.eno = 1000);
```

```
SELECT t.eno, CURSOR(SELECT * FROM TABLE(t.projects)) FROM employees t;
```

TABLE 式は、変数およびパラメータなどの一時値を含むコレクション値式の問合せに使用できます。

注意： TABLE 式は、前回のリリースで導入された THE 副問合せのかわりになります。THE 副問合せ式は、将来的に使用されなくなります。

コレクション・アンネスティング

多くのツールおよびアプリケーションにはコレクション型を扱う機能がないため、データをフラット化するビューが必要です。そのようなツールを使用して Oracle コレクション・データを参照するには、コレクションの要素を 1 つ以上のリレーショナル列にネスト解除またはフラット化する必要があります。それには、NESTED TABLE の要素をその NESTED TABLE を含む行と結合します。

NESTED TABLE を含む型を定義し、その NESTED TABLE にアクセスするための名前を指定した、次のようなオブジェクト・リレーショナル・スキーマを考えます。

```
CREATE TYPE emp_set_t IS NESTED TABLE of emp_t;  
CREATE TYPE dept_t(deptno NUMBER, emps emp_set_t);  
CREATE TABLE depts OF dept_t NESTED TABLE emps STORE AS depts_emps;
```

次の問合せは、NESTED TABLE EMPS の各要素と親行 DEPTS を結合することによって、DEPTS 表の EMPS 列データをネスト解除します。

```
SELECT d.deptno, e.* FROM depts d, TABLE(d.emps) e;
```

Oracle8i は、外部結合結果を生成する次の構文もサポートしています。

```
SELECT d.*, e.* FROM depts d, TABLE(d.emps)(+) e;
```

(+) は、DEPTS および D.EMPS の依存結合は引数 NULL を容認することを示しています。つまり、D.EMPS が NULL または空であっても、DEPTS の行が出力されます。D.EMPS に対応する列に NULL 値がある DEPTS の行があります。

コレクションの DML

Oracle では、NESTED TABLE 列に次の DML 操作をサポートします。

- コレクション自体を初期化する INSERT 文や UPDATE 文
- コレクションの要素に対する操作
 - コレクションへの新しい要素の挿入
 - コレクションからの要素の削除

■ コレクションの要素の更新

VARRAY 列におけるコレクション要素に対する DML 処理はサポートされていません。ただし、VARRAY 列はコレクション全体を単位として挿入または更新できます。

NESTED TABLE における要素単位の更新の場合、DML 文は TABLE 式を使用して、操作する NESTED TABLE を識別します。NESTED TABLE の DML 操作がシリアライズ化されることに注意してください。つまり、あるトランザクション内で DML 文が NESTED TABLE を操作したとき、他のトランザクションからの同じ NESTED TABLE への変更は、先のトランザクションが終了するまでブロックされます。

次の DML 文は、NESTED TABLE における要素単位の操作を示しています。

```
INSERT INTO TABLE(SELECT e.projects
                     FROM      employees e
                     WHERE     e.eno = 100)
VALUES (1, 'Project Neptune');

UPDATE TABLE(SELECT e.projects
               FROM      employees e
               WHERE     e.eno = 100) p
SET VALUE(p) = project_t(1, 'Project Pluto')
WHERE p.pno = 1;

DELETE FROM TABLE(SELECT e.projects
                     FROM      employee e
                     WHERE     e.eno = 100) p
WHERE p.pno = 1;
```

オブジェクト型およびそれらのメソッドの権限

オブジェクト型に対する権限は、システム・レベルおよびスキーマ・オブジェクト・レベルで存在します。

システム権限

Oracle では、オブジェクト型に次のシステム権限を定義します。

- CREATE TYPE: 自スキーマにオブジェクト型を作成できます。
- CREATE ANY TYPE: 任意のスキーマにオブジェクト型を作成できます。
- ALTER ANY TYPE: 任意のスキーマ内のオブジェクト型を変更できます。

- DROP ANY TYPE: 任意のスキーマ内のオブジェクト型を削除できます。
 - EXECUTE ANY TYPE: 任意のスキーマ内のオブジェクト型を使用および参照できます。
- CONNECT ロールおよび RESOURCE ロールには、CREATE TYPE システム権限が含まれます。DBA ロールには、前述のすべての権限が含まれます。

スキーマ・オブジェクト権限

オブジェクト型に適用できるスキーマ・オブジェクト権限は、EXECUTE のみです。

オブジェクト型に EXECUTE 権限があると、その型を次のように使用できます。

- 表の定義
- リレーショナル表の列の定義
- オブジェクト型の変数またはパラメータの宣言

EXECUTE は、コンストラクタなどの型のメソッドを起動します。

メソッド実行および対応する権限は、ストアド PL/SQL プロシージャのものと同じです。

新しい型または表での型の使用

次の場合は、前の項で説明した権限の他に、特定の権限が必要です。

- 別のユーザーが作成した型を使用する型または表を作成する場合
- 新しく作成した型または表の使用権限を別のユーザーに付与する場合

新しい型または表を定義するには、EXECUTE ANY TYPE システム権限、または使用する型に対する EXECUTE オブジェクト権限が必要です。これらの権限は、ロールを介してではなく、明示的に付与される必要があります。

新しい型または表へのアクセス権限を他のユーザーに付与するには、GRANT オプションを含む適切な EXECUTE オブジェクト権限、またはオプション WITH ADMIN OPTION を含む EXECUTE ANY TYPE システム権限が必要です。これらの権限は、ロールを介してではなく、明示的に付与される必要があります。

例

CONNECT ロールおよび RESOURCE ロールを持つ USER1、USER2、USER3 という 3 人のユーザーがいるとします。

USER1 は、USER1 スキーマ内で次の DDL を実行します。

```
CREATE TYPE type1 AS OBJECT ( attr1 NUMBER );
CREATE TYPE type2 AS OBJECT ( attr2 NUMBER );
GRANT EXECUTE ON type1 TO user2;
GRANT EXECUTE ON type2 TO user2 WITH GRANT OPTION;
```

USER2 は、USER2 スキーマ内で次の DDL を実行します。

```
CREATE TABLE tab1 OF user1.type1;
CREATE TYPE type3 AS OBJECT ( attr3 user1.type2 );
CREATE TABLE tab2 (col1 user1.type2 );
```

USER2 は、GRANT オプションを含んだ、USER1 の TYPE2 に対する EXECUTE 権限を持っているため、次の文は正常に実行されます。

```
GRANT EXECUTE ON type3 TO user3;
GRANT SELECT on tab2 TO user3;
```

ただし、USER2 は、GRANT オプションを含んだ、USER1.TYPE1 に対する EXECUTE 権限を持っていないため、次の文は正常に実行されません。

```
GRANT SELECT ON tab1 TO user3;
```

USER3 は、次の処理を正常に実行できます。

```
CREATE TYPE type4 AS OBJECT (attr4 user2.type3);
CREATE TABLE tab3 OF type4;
```

オブジェクト型へのアクセスおよびオブジェクト・アクセスの権限

オブジェクト型は EXECUTE 権限のみを使用しますが、オブジェクト表はリレーショナル表と同じ次のすべての権限を使用します。

- SELECT: 表から 1 つのオブジェクトおよびその属性にアクセスできます。
- UPDATE: 表内のオブジェクトの属性を変更できます。
- INSERT: 表に新しいオブジェクトを追加できます。
- DELETE: 表からオブジェクトを削除できます。

同様の表権限および列権限が、オブジェクト型の表列の使用を規制します。

オブジェクト表の列を選択するには、オブジェクト表の型に対する権限は必要ありません。ただし、行全体を選択するには必要です。

次のスキーマについて考えます。

```
CREATE TYPE emp_type as object (  
    eno    NUMBER,  
    ename  CHAR(31),  
    eaddr  addr_t );
```

```
CREATE TABLE emp OF emp_type;
```

次の2つの問合せも考えます。

```
SELECT VALUE(e) FROM emp e;  
SELECT eno, ename FROM emp;
```

どちらの問合せの場合も、Oracle は EMP 表に対するユーザーの SELECT 権限をチェックします。最初の問合せの場合、データを解析するには、ユーザーは EMP_TYPE 型情報を取得する必要があります。問合せが EMP_TYPE 型にアクセスすると、Oracle はユーザーの EXECUTE 権限をチェックします。

2 番目の問合せの実行は、オブジェクト型を起動しません。そのため、Oracle は型権限をチェックしません。

さらに、前の項からのスキーマを使用して、USER3 は次の問合せを実行できます。

```
SELECT tab1.col1.attr2 from user2.tab1 tab1;  
SELECT t.attr4.attr3.attr2 FROM tab3 t;
```

USER3 による 2 つの選択では、USER3 は基礎となる型の明示的な権限を持っていませんが、型所有者および表所有者は GRANT オプションを含む必要な権限を持っているため、この文は正常に実行されます。

Oracle は次の要求についての権限をチェックし、ユーザーに各処理に対する権限がない場合はエラーを戻します。

- PEF 値を使用してオブジェクト・キャッシュ内に 1 つのオブジェクトを確保すると、Oracle は、そのオブジェクトを含むオブジェクト表に対する SELECT 権限、およびそのオブジェクト型に対する EXECUTE 権限をチェックします（OCI オブジェクト・キャッシュの詳細は、6-4 ページの「[オブジェクトに対する OCI のヒントおよび技法](#)」を参照してください）。
- 既存オブジェクトの変更、またはオブジェクト・キャッシュからのオブジェクトのフラッシュを行うと、Oracle は、接続先オブジェクトに対する UPDATE 権限をチェックします。新しいオブジェクトをフラッシュすると、Oracle は、接続先オブジェクト表に対する INSERT 権限をチェックします。
- オブジェクトを削除すると、Oracle は、接続先の表に対する DELETE 権限をチェックします。

- メソッドを起動すると、Oracle は、対応するオブジェクト型に対する EXECUTE 権限をチェックします。

Oracle は、オブジェクト表に対する列レベルの権限は提供しません。

依存性および不完全な型

型は、互いに依存するように定義できます。たとえば、EMPLOYEE の 1 つの属性が従業員が属する部門で、DEPARTMENT の 1 つの属性が部門を管理する従業員というようなオブジェクト型 EMPLOYEE および DEPARTMENT を定義できます。

このように、直接的または中間の型を介して互いに依存する型を、相互依存といいます。依存性を示す矢印のついた相互依存型の図には、必ず、1 つの型で開始および終了する矢印のパスがあります。

そのような循環依存を定義するには、サイクルの少なくとも 1 つのブランチに REF を使用する必要があります。

たとえば、次の型を定義できます。

```
CREATE TYPE department;  
  
CREATE TYPE employee AS OBJECT (  
    name    VARCHAR2(30),  
    dept    REF department,  
    supv    REF employee );  
  
CREATE TYPE emp_list AS TABLE OF REF employee;  
  
CREATE TYPE department AS OBJECT (  
    name    VARCHAR2(30),  
    mgr     REF employee,  
    staff   emp_list );
```

これは、適切な相互依存型における SQL DDL 文の適切な順序です。Oracle は、これをエラーなしでコンパイルします。次の文はオプションです。

```
CREATE TYPE department;
```

これによって、エラーなしでコンパイルできます。DEPARTMENT は、不完全なオブジェクト型として確立されます。不完全オブジェクト型への REF は、エラーなしでコンパイルされるため、EMPLOYEE のコンパイルが引き続き行われます。

DEPARTMENT の定義を終了する最後の文に到達したとき、DEPARTMENT のすべてのコンポーネントは正常にコンパイルされています。したがって、コンパイルはエラーなしで終了します。

オプションで DEPARTMENT を不完全型として宣言しなかった場合、EMPLOYEE のコンパイルはエラーになります。その場合、Oracle は、スキーマ・オブジェクトのライブラリに、EMPLOYEE を不完全オブジェクト型として自動的に追加します。これによって、EMP_LIST 宣言および DEPARTMENT 宣言は、エラーなしでコンパイルされます。その後、EMPLOYEE が再コンパイルされると、エラーなしでコンパイルが終了し、EMPLOYEE は完全なオブジェクト型になります。

不完全な型の再定義

不完全オブジェクト型を宣言すると、それをオブジェクト型として完全な状態にする必要があります。不完全オブジェクト型を、表型、配列型などに宣言することはできません。その型を削除することはできます。

前の項で、EMPLOYEE のコンパイルに失敗した場合など、Oracle が型を不完全オブジェクト型として識別している場合も同様です。

表の型依存性

SQL コマンド REVOKE および DROP TYPE は、参照した型に表または依存する別の型があった場合、エラーを戻して異常終了します。

これらのコマンドのいずれかに FORCE オプションが含まれていると、動作はオーバーライドされます。コマンドは正常に実行され、影響する表または型は無効になります。

型定義に依存するデータを含む表の場合、型を変更すると表のデータにアクセスできなくなることがあります。これは、型に必要な権限が取り消されるか、型または依存する型が削除された場合に起きます。その後、表は無効になり、アクセスできなくなります。

必要な権限が再度付与された場合、権限がなくなったために無効になった表は、自動的に有効になり、アクセス可能になります。

依存する型が削除されたために無効になった表に、再びアクセスすることはできません。唯一可能なアクションは、表を削除することです。

オブジェクト型のインポート / エクスポート / ロード

エクスポート・ユーティリティおよびインポート・ユーティリティは、Oracle データベースへ、または Oracle データベースから、データを移動します。また、データのバックアップまたはアーカイブ、および異なるリリースへの Oracle RDBMS の移行に役立ちます。

エクスポートおよびインポートは、オブジェクト型をサポートします。エクスポートは、オブジェクト型定義および関連するすべてのデータをダンプ・ファイルに書き込みます。インポートは、ダンプ・ファイルからこれらの項目を再作成します。

SQL*Loader は、行オブジェクト、列オブジェクト、およびコレクションと参照を含むオブジェクトのロードをサポートします。Oracle8i では、オブジェクトに対して、従来型パス・ロードのみをサポートします。

従来型パス・ロードを使用しない場合は、まず、ダイレクト・パス・ロードを使用してデータをリレーショナル・データベースにロードします。その後、CREATE TABLE...AS SELECT コマンドを使用して、オブジェクト表および列オブジェクトを含む表を作成します。ただし、この方法では、実際のデータの 2 倍のデータを保持する十分な領域が必要です。

参照： Oracle オブジェクトのエクスポート、インポートおよびロードの詳細は、『Oracle8i ユーティリティ・ガイド』を参照してください。

Oracle プログラム環境のオブジェクト・サポート

Oracle8i では、DDL コマンドを使用してオブジェクト型を作成し、DML コマンドを使用してオブジェクトを操作できます。オブジェクト・サポートは、次の Oracle のアプリケーション・プログラミング環境に組み込まれています。

- SQL
- PL/SQL
- Oracle Call Interface (OCI)
- Pro*C/C++
- オブジェクト・タイプ・トランスレータ (OTT)
- Oracle Objects For OLE (Visual Basic、Excel、ActiveX、Active Server Page 対応)
- Java: JDBC、Oracle SQLJ および JPublisher

SQL

Oracle SQL DDL では、オブジェクト型に対して次のようなサポートが提供されます。

- オブジェクト型、NESTED TABLE および VARRAY の定義
- 権限の指定
- リレーショナル表の列のデータ型としてのオブジェクト型
- オブジェクト表の作成

Oracle SQL DML では、オブジェクト型に対して次のようなサポートが提供されます。

- オブジェクトおよびコレクションの間合せおよび更新
- REF の操作

参照： Oracle SQL 構文の詳細は、『Oracle8i SQL リファレンス』を参照してください。

PL/SQL

PL/SQL において、ファンクションおよびプロシージャ内で、オブジェクト型をサポートする SQL 機能を使用できます。

PL/SQL ファンクションおよびプロシージャのパラメータおよび変数には、オブジェクト型を使用できます。

オブジェクト型に対応付けられたメソッドを PL/SQL で実装できます。これらのメソッド（ファンクションおよびプロシージャ）は、ユーザーのスキーマの一部としてサーバーに格納します。

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

Oracle Call Interface (OCI)

OCI とは、アプリケーションが Oracle データベース内でデータおよびスキーマを操作するために使用できる、一連の C ライブラリ関数のことです。OCI では、次の項で説明するように、データベース・アクセスに従来の 3GL およびオブジェクト指向技法の両方がサポートされています。

OCI の重要なコンポーネントは、オブジェクト・キャッシュと呼ばれる作業領域を管理する一連のコールです。オブジェクト・キャッシュは、クライアント側のメモリー・ブロックです。プログラムはオブジェクト・キャッシュにオブジェクト全体を格納できるため、サーバーへの追加のラウンドトリップなしに、オブジェクト間をナビゲートできます。

オブジェクト・キャッシュは、それを使用するアプリケーションによって完全に制御および管理されます。Oracle Server は、オブジェクト・キャッシュにアクセスできません。オブジェクト・キャッシュを使用するアプリケーション・プログラムは、サーバーとのデータ一貫性をメンテナンスし、競合する同時アクセスから作業領域を保護する必要があります。

OCI では、次の機能が提供されます。

- SQL を使用して、サーバー上のオブジェクトにアクセスします。
- ポインタまたは REF を利用することによって、オブジェクト・キャッシュ内のオブジェクトに対するアクセスや操作、および管理を行うことができます。
- Oracle の日付、文字列および数値を C のデータ型に変換します。

- オブジェクト・キャッシュのメモリー・サイズを管理します。

OCI では、個別のオブジェクトをロックすることによって、同時実行性を改善します。また、複雑なオブジェクト検索をサポートして、パフォーマンスを改善します。

OCI 開発者は、オブジェクト・タイプ・トランスレータを使用して、Oracle オブジェクト型に対応する C のデータ型を生成できます。

参照： OCI でオブジェクトを使用する場合の詳細は、『Oracle8i コール・インタフェース・プログラマーズ・ガイド』を参照してください。

OCI プログラムにおける結合アクセス

3GL プログラムは、SQL 文および PL/SQL プロシージャを実行し、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します。データは、通常、クライアント側にデータを転送せずに、サーバー上で処理されます。OCI では、オブジェクト・データを操作する SQL 文を実行する API を提供することによって、この結合アクセスをサポートしています。特に、OCI では、次のことができます。

- オブジェクト・データおよびオブジェクト型スキーマ情報を操作する SQL 文を実行します。
- オブジェクト、オブジェクト参照 (REF) およびコレクションを、SQL 文の入力変数として渡します。
- オブジェクト、REF およびコレクションを、SQL 文フェッチの出力として戻します。
- オブジェクト、REF およびコレクションを戻す SQL 文のプロパティを記述します。
- オブジェクト型のパラメータを引数としてとる、またはオブジェクト型の結果を返す、PL/SQL プロシージャまたはファンクションを記述および実行します。
- 拡張されたコミットおよびロールバック関数を介して、オブジェクト機能とリレーショナル機能を並用できます。

参照： 3-6 ページの「[Pro*C/C++ での結合アクセス](#)」を参照してください。

OCI プログラムのナビゲーション・アクセス

オブジェクト指向のプログラミング・パラダイムでは、アプリケーションは、実社会のエンティティを、オブジェクト・グラフを形成する相互関連オブジェクトの集合としてモデル化します。オブジェクト間の関連は、参照として実装されます。アプリケーションは、いくつかのイニシャル・オブジェクトの集合から始めて、これらのイニシャル・オブジェクトの参照を使用して残りのオブジェクトにアクセスし、各オブジェクトで計算処理を実行することで、オブジェクトを処理します。OCI では、ナビゲーション・アクセスとして知られる、オブジェクト参照を利用してオブジェクト間のアクセスを行うための API が提供されています。特に、OCI では、次のことができます。

- クライアント・マシン上のメモリーにオブジェクトをキャッシュします。
- オブジェクト参照を解除し、オブジェクト・キャッシュ内の対応するオブジェクトを確保します。確保されたオブジェクトは、ホスト言語表現に透過的にマップされます。
- 確保されたオブジェクトが不要になった場合、キャッシュに通知します。
- 1 回のコールで、関連するオブジェクトをまとめてデータベースからクライアント・キャッシュにフェッチします。
- オブジェクトをロックします。
- キャッシュ上でオブジェクトを作成、更新および削除します。
- クライアント・キャッシュ上でオブジェクトに対して行われた変更を、データベースに反映します。

参照： 3-7 ページの「[Pro*C/C++ でのナビゲーション・アクセス](#)」を参照してください。

オブジェクト・キャッシュ

高パフォーマンスのオブジェクト・ナビゲーション・アクセスをサポートするために、OCI ランタイムでは、メモリーにオブジェクトをキャッシュするためのオブジェクト・キャッシュを提供しています。オブジェクト・キャッシュは、オブジェクト・キャッシュ内のデータベース・オブジェクトへの参照 (REF) をサポートし、データベース・オブジェクトは、それらの参照を介して識別 (確保) されます。オブジェクト・キャッシュは、データベース・オブジェクトに対して透過的で効率的なメモリー管理を提供するため、アプリケーションは、データベース・オブジェクトがキャッシュにロードされたときに、メモリーを割り当てたり解放したりする必要はありません。

さらに、データベース・オブジェクトは、キャッシュにロードされた際、ホスト言語に透過的にマップされます。たとえば、C 言語では、データベース・オブジェクトは対応する C 構造体へマップされます。オブジェクト・キャッシュは、キャッシュ上のオブジェクトと対応するデータベース・オブジェクトとの間の整合性を保ちます。トランザクションのコミット時にキャッシュ上のオブジェクト・コピーに対して行われた変更は、データベースに自動的に反映されます。

オブジェクト・キャッシュでは、REF をオブジェクトにマップするために、高速参照表がメンテナンスされます。アプリケーションが REF を参照解除するときに、対応するオブジェクトがまだキャッシュされていない場合は、オブジェクト・キャッシュからサーバーに自動的に要求が送信され、データベースからオブジェクトがフェッチされてオブジェクト・キャッシュにロードされます。同一の REF に対するその後の参照解除は、ローカル・キャッシュ・アクセスになり、ネットワークのラウンドトリップが発生しないため、より高速になります。キャッシュ内のオブジェクトにアクセス中であることをオブジェクト・キャッシュに通知するために、アプリケーションはオブジェクトを確保し、オブジェクトの処理が終わった時点で確保解除します。オブジェクト・キャッシュは、キャッシュの各オブジェクトの確保カウントをメンテナンスします。確保カウントは、確保コール（ピン・コール）で増加し、確保解除コール（アンピン・コール）で減少します。確保カウントが 0 になると、アプリケーションがそのオブジェクトを必要としなくなったことを意味します。オブジェクト・キャッシュでは最低使用頻度（LRU）アルゴリズムを使用して、キャッシュのサイズが管理されます。キャッシュが最大サイズに達すると、LRU アルゴリズムによって確保カウントが 0 の候補オブジェクトが解放されます。

オブジェクトを操作する OCI プログラムの作成

オブジェクトを操作する OCI プログラムを作成するときは、次の一般的なステップを実行する必要があります。

1. アプリケーション・オブジェクトに対応するオブジェクト型を定義します。
2. SQL DDL 文を実行して、必要なオブジェクト型をデータベースに定義します。
3. オブジェクト型をホスト言語形式で表します。

たとえば、C プログラムでオブジェクト型のインスタンスを操作するには、それらの型を C 言語形式で記述する必要があります。そのためには、オブジェクト型を C 構造体で表します。Oracle のオブジェクト・タイプ・トランスレータ（OTT）というツールを使用すると、オブジェクト型に対応する C の構造体を生成できます。OTT は、等価な C 構造体をヘッダー・ファイル（*.h）に作成します。これらの *.h ファイルを、アプリケーションを実装する C 関数を含む *.c ファイルにインポートします。

4. アプリケーションの *.c ファイルを OCI ライブラリとともにコンパイルおよびリンクすることによって、アプリケーションの実行可能ファイルを作成します。

参照： 6-4 ページの「[オブジェクトに対する OCI のヒントおよび技法](#)」を参照してください。

Pro*C/C++

Oracle Pro*C/C++ プリコンパイラを使用すると、プログラマはオブジェクト型を C プログラムおよび C++ プログラムで使用できます。

Pro*C 開発者は、オブジェクト・タイプ・トランスレータを使用して、Oracle オブジェクト型およびコレクション型を、Pro*C アプリケーションで使用できる C のデータ型にマップできます。

Pro*C では、コンパイル時のオブジェクト型とコレクション型のタイプ・チェック、およびデータベースのデータ型から C のデータ型への自動型変換が行われます。

Pro*C には、オブジェクトを作成および破棄するための EXEC SQL 構文が含まれており、次の 2 つを使用してサーバーにあるオブジェクトにアクセスできます。

- Pro*C プログラム内に埋め込まれた SQL 文および PL/SQL ファンクションまたはプロシージャ。
- オブジェクト・キャッシュへのインタフェース (3-2 ページの「[Oracle Call Interface \(OCI\)](#)」を参照)。この場合、オブジェクトにはポインタを介してアクセスし、サーバー上で変更および更新できます。

参照： Pro*C プリコンパイラの詳細は、『Oracle8i Pro*C/C++ プリコンパイラ・プログラマーズ・ガイド』を参照してください。

Pro*C/C++ での結合アクセス

結合アクセスのバックグラウンド情報については、3-3 ページの「[OCI プログラムにおける結合アクセス](#)」を参照してください。

Pro*C/C++ では、オブジェクトの結合アクセスに対して次の機能が提供されます。

- オブジェクト・キャッシュに割り当てられたオブジェクトの一時コピーのサポート
- 埋込み SQL の INSERT、UPDATE、DELETE 文、または SELECT 文の WHERE 句の入力ホスト変数として参照されるオブジェクトの一時コピーのサポート
- 埋込み SQL の SELECT および FETCHS 文で出力ホスト変数として参照されるオブジェクトの一時コピーのサポート
- オブジェクトの型およびスキーマ情報を取得するために DESCRIBE 文を介してオブジェクト型を参照する、ANSI 動的 SQL 文のサポート

Pro*C/C++ でのナビゲーション・アクセス

ナビゲーション・アクセスのバックグラウンド情報については、3-4 ページの「[OCI プログラムのナビゲーション・アクセス](#)」を参照してください。

Pro*C/C++ では、オブジェクトに対してさらにオブジェクト指向のインタフェースをサポートするために、次の機能が提供されます。

- 埋込み SQL の OBJECT Deref 文を使用した、オブジェクト・キャッシュでのオブジェクトの参照解除、確保およびロック（オプション）のサポート
- オブジェクトが更新または削除された場合、または不要になった場合に、Pro*C/C++ ユーザーが埋込み SQL の OBJECT UPDATE、OBJECT DELETE および OBJECT RELEASE 文を使用してオブジェクト・キャッシュに通知できる機能
- 埋込み SQL の OBJECT CREATE 文を使用して、オブジェクト・キャッシュに新しい参照可能オブジェクトを作成するための機能
- オブジェクト・キャッシュでの変更を、埋込み SQL の OBJECT FLUSH 文を使用してサーバーにフラッシュするための機能

Oracle のオブジェクト型と C 言語のデータ型の変換

オブジェクト・タイプ・トランスレータ（OTT）によって生成されるオブジェクトの C 表現では、スカラー属性に対して OCISString や OCINumber などの、内部詳細が非表示にされた OCI 型が使用されます。コレクション型およびオブジェクト参照も、OCITable、OCIArray および OCIRef 型を使用して同様に表されます。この不透明な型を使用すると、これらの型の内部形式の変更が見えなくなるため、C または C++ アプリケーションでこれらの型を使用することはお薦めしません。Pro*C/C++ では、C および C++ アプリケーションで OCI 型を簡単に使用できるように、次の拡張が行われています。

- 埋込み SQL の OBJECT GET 文を使用することで、オブジェクト属性を取得し暗黙的に C のデータ型に変換可能。
- 埋込み SQL の OBJECT SET 文を使用することで、オブジェクト属性を C のデータ型で設定可能。
- 埋込み SQL の COLLECTION GET 文を使用することで、コレクションをホスト配列にマップ可能。さらに、コレクションがスカラー型で構成されている場合、OCI 型は、暗黙的に C のデータ型に変換されます。
- 埋込み SQL の COLLECTION SET 文を使用することで、ホスト配列でコレクションの要素を更新可能。COLLECTION GET 文と同様、コレクションがスカラー型で構成されている場合、C のデータ型は、暗黙的に OCI 型に変換されます。

オブジェクト・タイプ・トランスレータ (OTT)

オブジェクト・タイプ・トランスレータ (OTT) は、オブジェクト型に対応する C の構造体を自動的に生成するプログラムです。OTT は、Pro*C プリコンパイラおよび OCI サーバー・アクセス・パッケージを使用しやすくします。

参照： OTT の詳細は、『Oracle8i コール・インタフェース・プログラマーズ・ガイド』および『Oracle8i Pro*C/C++ プリコンパイラ・プログラマーズ・ガイド』を参照してください。

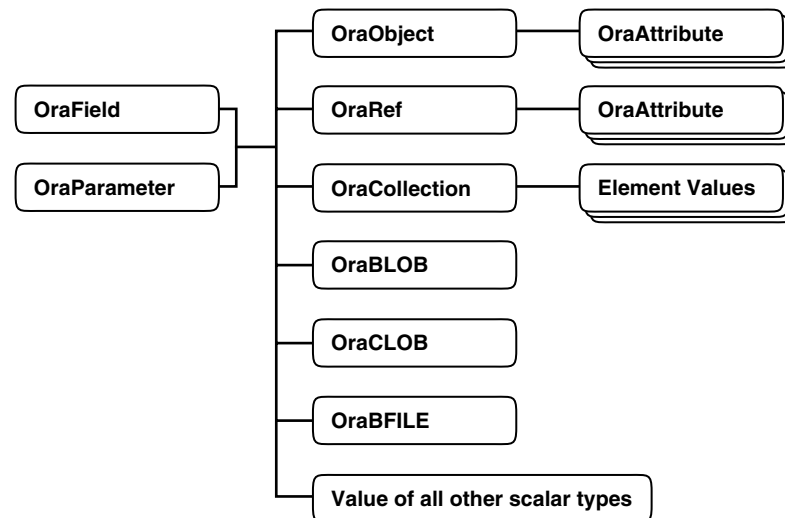
Oracle Objects For OLE (Visual Basic、Excel、ActiveX、Active Server Page 対応)

Oracle Objects for OLE (OO4O) では、Oracle データベース・サーバーにある REF のインスタンス、値インスタンス、可変長配列 (VARRAY) および NESTED TABLE へのアクセスおよび操作が完全にサポートされます。

参照： OO4O で Oracle オブジェクトを使用する場合の詳細は、OO4O のオンライン・ヘルプを参照してください。

図 3-1 に、OO4O におけるすべての型の値インスタンスの抑制階層を示します。

図 3-1 サポートされている Oracle データ型



これらの型のインスタンスは、データベースからフェッチできるか、または SQL 文および (ストアド・プロシージャおよびファンクションを含む) PL/SQL ブロックに対する入力変数または出力変数として渡すことができます。すべてのインスタンスは、属性の動的アクセスおよび操作のメソッドを提供する、COM オートメーション・インタフェースにマップされます。このインタフェースは、次のものから取得できます。

- Dynaset の OraField オブジェクトの値プロパティ

- SQL 文または PL/SQL ブロックで入力または出力パラメータとして使用される OraParameter オブジェクトの値プロパティ
- オブジェクトの属性 (REF)
- コレクションの要素 (VARRAY または NESTED TABLE)

Visual Basic でのオブジェクトの表現形式 (OraObject)

OraObject インタフェースは、Oracle 埋込みオブジェクトまたは値インスタンスの表現形式です。これには、値インスタンスの個々の属性にアクセスおよび操作（更新および挿入）するためのコレクション・インタフェース (OraAttributes) が含まれます。OraAttributes コレクション・インタフェースの個々の属性には、添字または属性の名前を使用して、アクセスできます。

次の Visual Basic の例では、person_tab 表の Address オブジェクトの属性にアクセスする方法を示します。

```
Dim Address OraObject
Set Person = OraDatabase.CreateDynaset("select * from person_tab", 0&)
Set Address = Person.Fields("Addr").Value
msgbox Address.Zip
msgbox Address.City
```

Visual Basic での REF の表現形式 (OraRef)

OraRef インタフェースは、Oracle オブジェクト参照 (REF) を表すと同時に、クライアント・アプリケーションの参照可能なオブジェクトを表します。このオブジェクト属性には、OraObject インタフェースによって表されるオブジェクトの属性と同じ方法でアクセスすることができます。OraRef は、COM での抑制メカニズムを介して、OraObject インタフェースから導出されます。REF オブジェクトは、Dynaset など派生元となるコンテキストからは独立して、更新および削除できます。OraRef インタフェースでは、6-9 ページの「[関連オブジェクトのプリフェッチ \(複合オブジェクト検索\)](#)」で説明する OCI の複合オブジェクト検索機能 (COR) を利用して、オブジェクト間の関連をたどる機能のカプセル化も行います。

Visual Basic での VARRAY および NESTED TABLE の表現形式 (OraCollection)

OraCollection インタフェースでは、OO4O において Oracle コレクション型、つまり可変長配列 (VARRAY) および NESTED TABLE にアクセスおよび操作するためのメソッドが提供されます。コレクションに含まれる要素には、添字でアクセスできます。

次の Visual Basic の例では、department 表の EnameList オブジェクトの属性にアクセスする方法を示します。

```
Dim EnameList OraCollection
Set Person = OraDatabase.CreateDynaset("select * from department", 0&)
set EnameList = Department.Fields("Enames").Value
'access all elements of the EnameList VArray
for I=1 to I=EnameList.Size
    msgbox EnameList(I)
Next I
```

Java: JDBC、Oracle SQLJ および JPublisher

Java は、開発者に簡単で効率的でポータブルかつ安全なアプリケーション開発プラットフォームを提供する強力な最新のオブジェクト指向言語として登場しました。Oracle では、Oracle オブジェクト機能を Java と統合する 2 つの方法として JDBC および Oracle SQLJ が提供されています。次の項では、これらの環境について詳細に説明します。

JDBC を利用した Oracle オブジェクト・データへのアクセス

JDBC (Java Database Connectivity) は、Oracle Server に接続するための Java インタフェースの集合です。Oracle では、オブジェクトと JDBC の間の緊密な統合が提供されます。SQL 型は Java クラスにマップでき、このマップ方法について、かなりの柔軟性が提供されています。

Oracle の JDBC:

- Java プログラム内において動的 SQL を利用することで、(データベースで定義された) オブジェクト型およびコレクション型にアクセスできます。
- デフォルトまたはカスタマイズ可能なマッピングを介して、データベースで定義されたデータ型を Java クラスに変換します。

JDBC 2.0 では、ユーザー定義型 (オブジェクト型) などのオブジェクト・リレーショナル構造がサポートされています。JDBC は、Oracle オブジェクトを特定の Java クラスのインスタンスとして具体化します。JDBC を使用して Oracle オブジェクトにアクセスすると、Oracle オブジェクトに対する Java クラスの作成、およびこれらのクラスへのデータの代入ができます。たとえば、次のような処理を行うことができます。

- JDBC にオブジェクトを STRUCT として具体化させます。この場合、JDBC は属性に対するクラスを作成し、それにデータを代入します。
- Oracle オブジェクトと Java クラスの間のマッピングを直接指定します。つまり、Java クラスをオブジェクト・データ用にカスタマイズします。ドライバは、カスタマイズされた指定の Java クラスに代入でき、これが Java クラスに対する一連の制約となります。これらの制約に従うために、クラスを SQLData インタフェースまたは CustomDatum インタフェースのどちらに合わせて定義するかを選択できます。

参照： JDBC の詳細は、『Oracle8i JDBC 開発者ガイドおよびリファレンス』を参照してください。

SQLJ を利用した Oracle オブジェクト・データへのアクセス

SQLJ では、Java コードに埋め込んだ SQL 文を使用して、サーバー・オブジェクトへアクセスできます。

- Java プログラムでオブジェクト型を使用可能です。
- JPublisher を使用して、オブジェクト型およびコレクション型を、アプリケーションで
使用できる Java クラスにマップします。
- SQL 文のオブジェクト型およびコレクション型は、コンパイル時にチェックされます。

参照： SQLJ の詳細は、『Oracle8i Java 開発者ガイド』を参照してください。

データ・マッピングの選択

Oracle SQLJ では、イテレータまたはホスト式で使用するために、強力な型指定または緩い型指定の Java 表現のいずれかによるオブジェクト型、参照型 (REF) およびコレクション型 (VARRAY および NESTED TABLE) がサポートされます。

強力な型指定の表現では、特定のオブジェクト型、参照型またはコレクション型に対応するカスタム Java クラスを使用し、インタフェース `oracle.sql.CustomDatum` を実装する必要があります。Oracle JPublisher ユーティリティでは、そのようなカスタム Java クラスを自動的に生成します。

緩い型指定の表現では、クラス `oracle.sql.STRUCT` (オブジェクト用)、`oracle.sql.REF` (参照用) または `oracle.sql.ARRAY` (コレクション用) を使用します。

参照： これらのサンプル・コードについては、8-39 ページの「[Java を使用したオブジェクトの操作](#)」を参照してください。

JPublisher を使用した JDBC および SQLJ プログラム用 Java クラスの作成

Oracle では、オブジェクト型、参照型およびコレクション型を Java クラスにマップすることで、強力な型指定によるすべての利点を得ることができます。

- Oracle JPublisher を使用してカスタム Java クラスを自動的に生成し、それらのクラスを変更なしで使用します。
- JPublisher によって生成されたクラスをサブクラス化し、独自の専用 Java クラスを作成します。
- 『Oracle8i SQLJ 開発者ガイドおよびリファレンス』に記述されている要件をクラスが満たすように、JPublisher を使用せず、カスタム Java クラスを手動でコーディングします。

JPublisher を使用して生成されたクラスが不十分な場合にサブクラス化することをお勧めします。

JPublisher が生成するもの

オブジェクト型に対して JPublisher を実行する場合、次のものが自動的に作成されます。

- ユーザーの Oracle オブジェクト型に対応する型定義として動作するカスタム・オブジェクト・クラス

このクラスには、各属性に対する取得メソッドおよび設定メソッドが含まれます。メソッド名は、属性 `foo` の場合は、`getFoo()` および `setFoo()` の形式をとります。

また、オプションで、サーバーで実行される Oracle オブジェクト・メソッドを起動するラッパー・メソッドをクラスに作成するように指定できます。

- ユーザーの Oracle オブジェクト型に対するオブジェクト参照用のカスタム・リファレンス・クラス

このクラスには、ユーザーのカスタム・オブジェクト・クラスのインスタンスを戻す `getValue()` メソッドおよびデータベースのオブジェクト値を更新する `setValue()` メソッドが含まれます。これらのメソッドは、入力としてカスタム・オブジェクト・クラスのインスタンスをとります。

コレクション型に対して JPublisher を実行する場合、次のものが自動的に作成されます。

- ユーザーの Oracle コレクション型に対応する型定義として動作するカスタム・コレクション・クラス

このクラスには、コレクション全体を取得または更新する、オーバーロード型の `getArray()` メソッドおよび `setArray()` メソッド、コレクションの個々の要素を取得または更新する `getElement()` メソッドおよび `setElement()` メソッド、その他のユーティリティ・メソッドが含まれます。

これらのどのカテゴリでも、JPublisher が生成するカスタム Java クラスでは、`CustomDatum` インタフェース、`CustomDatumFactory` インタフェースおよび `getFactory()` メソッドが実装されます。

参照： JPublisher の使用方法の詳細は、『Oracle8i JPublisher ユーザーズ・ガイド』を参照してください。

オブジェクト・モデルのリレーショナル・データへの適用

この章では、基礎となるリレーショナル・データの構造を変更することなく、オブジェクト指向アプリケーションを作成する方法を示します。

- オブジェクト・ビュー使用の利点
- オブジェクト・ビューの定義
- アプリケーションにおけるオブジェクト・ビューの使用
- オブジェクト・ビューにおけるオブジェクトのネスト
- オブジェクト・ビューにおける NULL オブジェクトの識別
- オブジェクト・ビューにおける NESTED TABLE および VARRAY の使用
- オブジェクト・ビューに対するオブジェクト識別子の指定
- オブジェクト・ビューが提供するオブジェクトへの参照の作成
- オブジェクト・ビューを利用した逆関連のモデル化
- オブジェクト・ビューの更新
- オブジェクト・モデルのリモート表への適用
- オブジェクト・ビューにおける複雑な関連の定義

オブジェクト・ビュー使用の利点

ビューが仮想表であるように、オブジェクト・ビューは仮想オブジェクト表です。オブジェクト・ビューの各行は、1つのオブジェクトであるため、ドット表記法を使用してこのオブジェクトのメソッドをコールし属性にアクセスすることができます。また、これを指す REF を作成することもできます。

オブジェクト・ビューは、オブジェクト指向アプリケーションのプロトタイピングやオブジェクト指向アプリケーションへ切り替える場合に役立ちます。ビューの中のデータはリレーショナル表から取り出すことができ、オブジェクト表として定義されているかのようにアクセスすることができるためです。したがって、既存の表を別の物理構造に変換しなくても、オブジェクト指向アプリケーションを実行できます。

オブジェクト・ビューは、オブジェクト・データに適用される従来のビューと同様の機能を提供します。たとえば、機密性の高いデータを含む属性を持たず、削除方法がない従業員の表のオブジェクト・ビューを作成することも可能です。

オブジェクト・ビューの使用によって、パフォーマンスが向上します。オブジェクト・ビューの1行を構成するリレーショナル・データは、1単位としてネットワークを横断するため、ラウンドトリップが大幅に削減されます。

リレーショナル・データは、クライアント側のオブジェクト・キャッシュにフェッチすることができ、C の構造体、C++ または Java クラスにマップすることもできます。そのため、3GL アプリケーションで固有のクラスのように操作することができます。また、複雑なオブジェクト検索のようなオブジェクト機能をリレーショナル・データとともに使用することもできます。

- リレーショナル・データからオブジェクトを合成することによって、新しい方法でデータを問い合わせることができます。複数の表との複雑な結合を記述するかわりに、オブジェクトの参照解除を利用することで複数の表からデータを表示することができます。
- このビューのオブジェクトは、クライアント上ではなくサーバー内で処理されるため、SQL 文の数およびネットワーク通信量を大幅に削減できます。
- オブジェクト・ビューのオブジェクト・データは、クライアント側のオブジェクト・キャッシュに確保し使用することができます。オブジェクト・キャッシュ上に確保された、これらの合成されたオブジェクトを、特殊化されたオブジェクト検索メカニズムを使用して取り出すことで、ネットワーク通信量を削減できます。
- モデル開発が継続できるビューの内部にオブジェクト・モデルを作成する場合は、大幅な柔軟性が得られます。オブジェクト型を変更する必要がある場合、無効なビューを新しい定義に簡単に置換できます。
- ビューの中のオブジェクトを使用することで、基礎となる記憶域メカニズムの特性が制限されることはまったくありません。同様に、現行のテクノロジーによって制限されることもありません。たとえば、パラレル化およびパーティション化されたリレーショナル表からオブジェクトを合成することもできます。
- 基礎となる同じデータから異なる複合データ・モデルを作成できます。

参照：

- SQL 構文および使用方法の詳細は、『Oracle8i SQL リファレンス』を参照してください。
 - PL/SQL 機能の詳細は、『Oracle8i PL/SQL ユーザーズ・ガイドおよびリファレンス』を参照してください。
 - Java の詳細は、『Oracle8i Java ストアド・プロシージャ開発者ガイド』を参照してください。
 - これらの機能の詳細は、『Oracle8i コール・インタフェース・プログラマーズ・ガイド』を参照してください。
-

オブジェクト・ビューの定義

オブジェクト・ビューを定義する手順は次のとおりです。

1. 既存のリレーショナル表の列にそれぞれ対応するデータ型の属性を持つオブジェクト型を定義します。
2. リレーショナル表からデータを抽出する問合せを作成します。オブジェクト型の属性と同じ順序で、リレーショナル表の列を指定します。
3. 基礎となるデータの属性に基づいて、一意の値を指定します。これは、ビュー内のオブジェクトを指すポインタ (REF) を作成するためのオブジェクト識別子として動作します。既存の主キーを使用することもできます。

オブジェクト・ビューを更新するには、さらに別の手順が必要です。オブジェクト型の属性が既存の表の列に正確に対応していない場合、次のことを行ってください。

4. **INSTEAD OF** トリガー (4-11 ページの「[オブジェクト・ビューの更新](#)」を参照) を作成します。アプリケーション・プログラムがオブジェクト・ビューの中でデータを更新しようすると、Oracle がこれを実行します。

これで、オブジェクト・ビューをオブジェクト表とまったく同じように使用できます。

たとえば、次の SQL 文は、ビューの各行が **EMPLOYEE_T** 型のオブジェクトであるオブジェクト・ビューを定義します。

```
CREATE TABLE emp_table (  
    empnum    NUMBER (5),  
    ename     VARCHAR2 (20),  
    salary    NUMBER (9, 2),  
    job       VARCHAR2 (20) );
```

```
CREATE TYPE employee_t (  

```

```
empno    NUMBER (5),
ename    VARCHAR2 (20),
salary   NUMBER (9, 2),
job       VARCHAR2 (20) );

CREATE VIEW emp_view1 OF employee_t
WITH OBJECT IDENTIFIER (empno) AS
  SELECT  e.empnum, e.ename, e.salary, e.job
    FROM    emp_table e
   WHERE   job = 'Developer';
```

リレーショナル表の EMPNUM 列に相当するデータにアクセスするには、オブジェクト型の EMPNO 属性にアクセスします。

アプリケーションにおけるオブジェクト・ビューの使用

オブジェクト・ビューの行にあるデータは、2 つ以上の表から生成されている可能性があります。オブジェクトは 1 つの操作のみでネットワークを横断します。インスタンスがクライアント側のオブジェクト・キャッシュにある場合、C または C++ の構造体として、または PL/SQL オブジェクト変数としてプログラマに示されます。このインスタンスを、他のすべてのシステム固有の構造体と同じように操作することもできます。

SQL 文の中で、オブジェクト・ビューはオブジェクト表と同じ方法で参照できます。たとえば、オブジェクト・ビューは、SELECT 文、UPDATE-SET 句または WHERE 句の中で使用できます。

また、オブジェクト・ビューの上に、オブジェクト・ビューを定義することもできます。

オブジェクト表からオブジェクトに対して使用するものと同じ OCI コールを使用して、クライアント側のオブジェクト・ビューのデータにアクセスできます。たとえば、REF を確保するために `OCIObjectPin()` を、また、オブジェクトをサーバーにフラッシュするために `OCIObjectFlush()` を使用することができます。オブジェクト・ビューのオブジェクトを更新またはサーバーにフラッシュした場合、Oracle は、オブジェクト・ビューを更新します。

参照： OCI コールの詳細は、『Oracle8i コール・インタフェース・プログラマーズ・ガイド』を参照してください。

オブジェクト・ビューにおけるオブジェクトのネスト

1 つのオブジェクト型の属性の 1 つがオブジェクト型である場合、リレーショナル・データから適切な列オブジェクトを抽出する必要があります。列オブジェクトは、リレーショナル表にすでに存在している列オブジェクトから選択でき、適切なタイプ・コンストラクタを使用して、関連する列の集合から合成することもできます。

たとえば、部門表 dept を考えてみます。

```
CREATE TABLE dept
(
    deptno      NUMBER PRIMARY KEY,
    deptname    VARCHAR2(20),
    deptstreet  VARCHAR2(20),
    deptcity    VARCHAR2(10),
    deptstate   CHAR(2),
    deptzip     VARCHAR2(10)
);
```

部門オブジェクトが住所をオブジェクトとして持つように、オブジェクト・ビューを作成します。これによって、住所オブジェクト用の再使用可能なメソッドを定義し、すべての種類のアドレスに対して使用できます。

1. アドレス・オブジェクト用の型を作成します。

```
CREATE TYPE address_t AS OBJECT
(
    street  VARCHAR2(20),
    city    VARCHAR2(10),
    state   CHAR(2),
    zip     VARCHAR2(10)
);
/
```

2. 住所をオブジェクトとして持つ部門オブジェクト用の型を作成します。

```
CREATE TYPE dept_t AS OBJECT
(
    deptno      NUMBER,
    deptname    VARCHAR2(20),
    address     address_t
);
/
```

3. 部門番号、名前および住所を含むビューを作成します。住所は、リレーショナル表のいくつかの列で構成されます。

```
CREATE VIEW dept_view OF dept_t WITH OBJECT IDENTIFIER (deptno) AS
SELECT d.deptno, d.deptname,
       address_t(d.deptstreet,d.deptcity,d.deptstate,d.deptzip) AS deptaddr
FROM   dept d;
```

オブジェクト・ビューにおける NULL オブジェクトの識別

オブジェクトのコンストラクタが NULL を返すことはありません。したがって、前述のビューにおいて、リレーショナル表の中の city、street などのすべての列が NULL であったとしても、アドレス・オブジェクトが NULL になることはありません。リレーショナル表には、部門の住所が NULL かどうかを指定する列はありません。NULL である deptstreet 列が、すべての住所が NULL であることを示すための規則を定義した場合、DECODE 関数またはその他の関数を使用して、NULL または構成されたオブジェクトのどちらかを返すことができます。

```
CREATE VIEW dept_view AS
  SELECT d.deptno, d.deptname,
         DECODE(d.deptstreet, NULL, NULL,
                address_t(d.deptstreet, d.deptcity, d.deptstate, d.deptzip)) AS deptaddr
  FROM dept d;
```

このような方法では、リレーショナル表の中の列に直接対応していないため、ビューを介して部門の住所を直接更新できなくなります。そのかわり、ビュー上に INSTEAD-OF トリガーを定義し、この列の更新をハンドルします。

オブジェクト・ビューにおける NESTED TABLE および VARRAY の使用

NESTED TABLE および VARRAY のコレクションはどちらも、ビューの列として使用することができます。これらのコレクションは、基礎となるコレクション列から選択するか、または副問合せを使用して合成できます。CAST-MULTISET 演算子によって、これらのコレクションを合成します。

もう一度、前述の例を考えてみます。次の構造を持つ emp リレーショナル表の中で各従業員を表示します。

```
CREATE TABLE emp
(
  empno    NUMBER PRIMARY KEY,
  empname  VARCHAR2(20),
  salary   NUMBER,
  deptno   NUMBER REFERENCES dept(deptno)
);
```

このリレーショナル表を使用すると、部門番号、名前、住所および部門に所属する従業員のコレクションを持つ dept_view を構成できます。

1. 従業員型を定義し、その従業員型を利用して NESTED TABLE 型を定義します。

```
CREATE TYPE employee_t AS OBJECT
(
  eno NUMBER,
```

```

    ename VARCHAR2(20),
    salary NUMBER
);

CREATE TYPE employee_list_t AS TABLE OF employee_t;

```

2. 部門番号、名前、住所および部門に所属する従業員のコレクションを属性として持つ部門型を定義します。

```

CREATE TYPE dept_t AS OBJECT
(
    deptno      NUMBER,
    deptname    VARCHAR2(20),
    deptaddr    address_t,
    emp_list    employee_list_t
);
/

```

3. ここで dept_view が定義できます。

```

CREATE VIEW dept_view OF dept_t WITH OBJECT IDENTIFIER (deptno) AS
    SELECT d.deptno, d.deptname,
           address_t(d.deptstreet,d.deptcity,d.deptstate,d.deptzip) AS deptaddr,
           CAST( MULTISET (
                       SELECT e.empno, e.empname, e.salary
                       FROM emp e
                       WHERE e.deptno = d.deptno)
              AS employee_list_t)
              AS emp_list
    FROM   dept d;

```

CAST-MULTISSET ブロック内の SELECT 副問合せが、現行の部門に所属する従業員のリストを選択します。MULTISSET キーワードは、これが単一の値とは対照的なリストであることを示します。CAST 演算子は、結果セットを適切な型にキャスト（型変換）します。この場合は、NESTED TABLE 型 employee_list_t にキャストします。

このビューへの問合せでは、部門番号、名前、住所オブジェクトおよびその部門に所属する従業員のコレクションを含む各部門行のリストが取得されます。

オブジェクト・ビューに対するオブジェクト識別子の指定

オブジェクト・ビューの中で、行オブジェクトを指すポインタ（REF）を作成できます。ビューのデータは永続的に格納されるわけではないため、オブジェクト識別子として使用される値を指定する必要があります。オブジェクト識別子によって、オブジェクト・ビューにおけるオブジェクトを、オブジェクト・キャッシュ上で参照し、確保することができます。

このビューがオブジェクト表またはオブジェクト・ビューに基づいている場合には、すでに各行に対応付けられたオブジェクト識別子があり、これを再使用できます。WITH OBJECT IDENTIFIER 句を省略するか、または WITH OBJECT IDENTIFIER DEFAULT 句を指定します。

ただし、行オブジェクトがリレーショナル・データから合成されている場合、オブジェクト識別子として利用する値を選択する必要があります。

Oracle では、主キーに基づいたオブジェクト識別子を指定できます。行オブジェクトを識別する一意のキーの集合は、そのオブジェクトの識別子になります。値が重複すると、オブジェクト参照時に問題が発生することもあるため、これらの値はビューから選択された行内で一意である必要があります。

WITH OBJECT IDENTIFIER 句を使用して作成されたオブジェクト・ビューは、指定した列の値から導出されたオブジェクト識別子を持ちます。WITH OBJECT IDENTIFIER DEFAULT 句が指定されている場合、オブジェクト識別子は、基礎となる表またはビュー定義によって、システム生成または主キー・ベースです。

この部門表の例を使用して、部門番号をオブジェクト識別子として使用する dept_view オブジェクト・ビューを作成できます。

この部門型 dept_t の場合の、行のオブジェクト型を定義します。

```
CREATE TYPE dept_t AS OBJECT
(
    dno          NUMBER,
    dname        VARCHAR2(20),
    deptaddr     address_t,
    emplist      employee_list_t
);
```

基礎となるリレーショナル表が deptno を主キーとして持つため、各部門行は一意の部門番号を持ちます。このビューにおいて、deptno 列はオブジェクト型の dno 属性になります。ビュー・オブジェクト内で dno が一意であることがわかれば、これをオブジェクト識別子として指定できます。

```
CREATE VIEW dept_view OF dept_t WITH OBJECT IDENTIFIER(dno)
AS SELECT d.deptno, d.deptname,
        address_t(d.deptstreet,d.deptcity,d.deptstate,d.deptzip),
        CAST( MULTISET (
                SELECT e.empno, e.empname, e.salary
                FROM emp e
                WHERE e.deptno = d.deptno)
        AS employee_list_t)
FROM    dept d;
```

参照： 6-3 ページの「[オブジェクト識別子](#)」を参照してください。

オブジェクト・ビューが提供するオブジェクトへの参照の作成

これまでの例では、dept_view ビューから選択された各オブジェクトは、部門番号の値から導出された一意のオブジェクト識別子を持っています。リレーショナルの場合、従業員表 emp の外部キー deptno は、部門表 dept の主キー値 deptno と一致します。dept_view では、オブジェクト識別子の作成に主キー値を使用します。したがって、dept_view における主キー値の参照の作成に、emp_view における外部キーを使用できます。

このためには、MAKE_REF 演算子を使用して、主キー・オブジェクト参照を合成します。この演算子は、参照がポイントするビューまたは表の名前および外部キー値のリストをとり、参照されたビューの中のある特定のオブジェクトと一致するような参照のオブジェクト識別子部分を作成します。

従業員の番号、名前、給料および所属部門への参照を持つ emp_view ビューを作成するには、従業員型 emp_t およびその型に基づいたビューを作成する必要があります。

```
CREATE TYPE emp_t AS OBJECT
(
    eno      NUMBER,
    ename    VARCHAR2(20),
    salary   NUMBER,
    deptref  REF dept_t
);

CREATE VIEW emp_view OF emp_t WITH OBJECT IDENTIFIER(en)
AS SELECT e.empno, e.empname, e.salary,
        MAKE_REF(dept_view, e.deptno)
FROM emp e;
```

ビューの中の deptref 列は、部門への参照を保持します。サンフランシスコ市にある部門のすべての従業員を判断するには、次の簡単な問合せを作成します。

```
SELECT e.eno, e.salary, e.deptref.dno
FROM emp_view e
WHERE e.deptref.deptaddr.city = 'San Francisco';
```

dept_view オブジェクトを参照するために REF 修飾子を使用できるように注意してください。

```
CREATE VIEW emp_view OF emp_t WITH OBJECT IDENTIFIER(en)
AS SELECT e.empno, e.empname, e.salary, REF(d)
```

```
FROM emp e, dept_view d
WHERE e.deptno = d.dno;
```

この場合、dept_view および emp 表を deptno キーで結合します。REF 修飾子のかわりに MAKE_REF 演算子を使用するのは、この演算子を使用すると、循環参照を作成できるという利点があるためです。たとえば、所属部門への参照を持つ従業員のビューを作成することができます。また、部門のビューは、その部門に所属している従業員への参照のリストを持つことができます。

オブジェクト・ビューが主キーに基づくオブジェクト識別子を持つ場合、このようなビューへの参照は主キーに基づくことに注意してください。一方、システム生成のオブジェクト識別子を持つビューへの参照は、システム生成のオブジェクト参照になります。この違いは、OCI オブジェクト・キャッシュでオブジェクト・インスタンスを作成した場合にのみ関連し、新しく作成したオブジェクトを参照するために必要になります。これについては後の項で説明します。

合成オブジェクトの場合と同様に、持続的に格納される参照をビュー列として選択し、これらを問合せで透過的に使用できます。ただし、ビュー・オブジェクトへのオブジェクト参照は、持続的には格納されません。

オブジェクト・ビューを利用した逆関連のモデル化

オブジェクトを持つビューは、逆関連をモデル化するために使用できます。

1 対 1 関連

1 対 1 関連は、逆オブジェクト参照でモデル化できます。たとえば、各従業員がそれぞれコンピュータを持っており、そのコンピュータはその従業員のみが使用するとします。リレーショナル・モデルは、コンピュータ表から従業員表へ、またはその逆へ外部キーを使用して 1 体 1 関連を獲得します。ビューを使用するとオブジェクトをモデル化することができ、その結果従業員からコンピュータ・オブジェクトへのオブジェクト参照、およびコンピュータ・オブジェクトから従業員への参照を持つことができます。

1 対多および多対 1 関連

1 対多関連（または多対 1 関連）は、オブジェクト参照を使用するか、オブジェクトを埋め込むことでモデル化できます。1 対多関連は、オブジェクトのコレクションまたはオブジェクト参照のコレクションを持つことでモデル化できます。多対 1 関連は、オブジェクト参照を使用してモデル化できます。

部門対従業員の例を考えてみます。基礎となるリレーショナル・モデルでは、外部キーが従業員表の中にあります。ビューにおいてコレクションを使用すると、部門と従業員の関連をモデル化できます。

部門ビューは従業員のコレクションを持つことができ、従業員ビューは部門への参照を持つ（または部門値をインライン化する）ことができます。これによって、前方関連（従業員から部門へ）および逆関連（部門から従業員リストへ）の両方が獲得されます。また、部門ビューは、従業員オブジェクトを埋め込むかわりに、従業員オブジェクトへの参照コレクションを持つこともできます。

オブジェクト・ビューの更新

オブジェクト表と同じ SQL DML を使用して、オブジェクト・ビューのデータを更新、挿入および削除できます。矛盾がなければ、Oracle はオブジェクト・ビューのベース表を更新します。

ビューの問合せが結合、集合演算子、集計関数、GROUP BY または DISTINCT を含む場合、このビューは更新できません。ビューの問合せが疑似列または式を含む場合、対応するビュー列は更新できません。オブジェクト・ビューに結合が含まれることはよくあります。

これらの問題を解決するために、Oracle は INSTEAD OF トリガーを提供しています。Oracle は実際の DML 文のかわりにトリガー本体を実行するため、INSTEAD OF トリガーといわれます。

INSTEAD OF トリガーは、オブジェクト・ビューまたはリレーショナル・ビューを更新するための透過的な方法を提供します。オブジェクト表と同じ SQL DML (INSERT、DELETE および UPDATE) 文を実行すると、Oracle が SQL 文のかわりに適切なトリガーを起動し、トリガー本体の中で指定された処理が行われます。

ビューにおける NESTED TABLE 列の更新

NESTED TABLE は、新しい要素を挿入したり、既存の要素を更新または削除することで変更することができます。ビューにおいて仮想または合成された NESTED TABLE 列は、通常は更新できません。これを解決するために、Oracle では、INSTEAD OF トリガーをこの列の上に作成できます。

(ビューの) NESTED TABLE 列に対して定義された INSTEAD OF トリガーは、列が修正された場合に起動します。コレクション全体が（親行の更新によって）更新された場合、INSTEAD OF トリガーは起動されないので注意してください。

変更および妥当性チェックを制御する INSTEAD-OF トリガーの使用

INSTEAD-OF トリガーを利用することで、他の方法では更新できない複雑なビューを更新することができます。また制約を施行し、権限をチェックし、DML の妥当性チェックを行うためにも使用することができます。これらのトリガーを使用すると、挿入、更新および削除による、オブジェクト・ビューを介して作成されたオブジェクトの変更を制御できます。

たとえば、ある部門の従業員の数が 10 を越えることができないという条件を施行する場合を考えてみます。これを施行するために、従業員ビューのための INSTEAD-OF トリガーを作成できます。ビューは更新できるため、DML の実行にトリガーは必要ありません。ただし、制約を施行するにはトリガーが必要です。

次のコードを使用して、トリガーを実装します。

```
CREATE TRIGGER emp_instr INSTEAD OF INSERT ON emp_view
FOR EACH ROW
DECLARE
    dept_var dept_t;
    emp_count integer;
BEGIN
    -- Enforce the constraint...!
    -- First get the department number from the reference
    UTIL_REF.SELECT_OBJECT(:NEW.deptref,dept_var);

    SELECT COUNT(*) INTO emp_count
    FROM emp
    WHERE deptno = dept_var.dno;

    IF emp_count < 9 THEN
        -- let us do the insert
        INSERT INTO emp VALUES (:NEW.eno,:NEW.ename,:NEW.salary,dept_var.dno);
    END IF;
END;
```

オブジェクト・モデルのリモート表への適用

リモート表は、オブジェクト表のように直接アクセスすることはできませんが、オブジェクト・ビューを利用することで、オブジェクト表と同じようにリモート表にアクセスできます。

2 つの支店をワシントン D.C. およびシカゴに持つ会社を例に考えてみます。各支社は従業員表を持ちます。ワシントンの本社には、すべての部門のリストを持つ部門表があります。組織全体のビューを作成するには、各リモート表でビューを作成し、次に組織全体のビューを作成します。

まず、各従業員表のオブジェクト・ビューを作成します。

```
CREATE VIEW emp_washington_view (eno,ename,salary)
AS SELECT e.empno, e.empname, e.salary
```

```

FROM emp@washington_link e;

CREATE VIEW emp_chicago_view
AS SELECT e.eno, e.name, e.salary
FROM emp_tab@chicago_link e;

```

ここで、グローバルなビューを作成できます。

```

CREATE VIEW orgnzn_view OF dept_t WITH OBJECT IDENTIFIER (dno)
AS SELECT d.deptno, d.deptname,
address_t(d.deptstreet,d.deptcity,d.deptstate,d.deptzip),
CAST( MULTISSET (
SELECT e.eno, e.ename, e.salary
FROM emp_washington_view e)
AS employee_list_t)
FROM dept d
WHERE d.deptcity = 'Washington'
UNION ALL
SELECT d.deptno, d.deptname,
address_t(d.deptstreet,d.deptcity,d.deptstate,d.deptzip),
CAST( MULTISSET (
SELECT e.eno, e.name, e.salary
FROM emp_chicago_view e)
AS employee_list_t)
FROM dept d
WHERE d.deptcity = 'Chicago';

```

このビューは、各部門のすべての従業員のリストを持ちます。複数の部門に所属する従業員はいないため、UNION ALL を使用します。複数の部門に所属する従業員がいる場合は、行の UNION を使用できます。ただし、UNION 演算子を使用する場合は、2つのコレクションの比較が適切に実行されるように、CAST-MULTISSET 式内に ORDER BY 演算子を導入する必要があります。

オブジェクト・ビューにおける複雑な関連の定義

MAKE_REF 演算子を使用して、オブジェクト・ビューに循環参照を定義できます。循環参照では、view_A は、view_A を参照する view_B を参照できます。これによって、オブジェクト・ビューは、リレーショナル・データから複雑な構造体を合成できます。

たとえば、部門および従業員の例では、部門オブジェクトには、現在、従業員のリストが含まれています。領域を節約するためには、部門オブジェクト内のすべての従業員を具体化するかわりに、従業員オブジェクトへの参照を部門オブジェクト内に入れることができます。従業員オブジェクトへの参照を構成（確保）し、後でドット表記法を使用してこの参照に従い、従業員情報を抽出できます。

従業員オブジェクトはすでに従業員の所属部門への参照を持っているため、このモデルでのオブジェクト・ビューには、部門ビューと従業員ビューの間の循環参照が含まれます。

次の2つの方法で、オブジェクト・ビュー間の循環参照を作成できます。

方法 1:

1. ビュー B への参照をまったく持たないビュー A を作成します。
2. ビュー A への参照を含むビュー B を作成します。
3. ビュー B への参照を含む新しい定義を持つビュー A に置換します。

方法 2:

1. FORCE キーワードを使用して、ビュー B への参照を持つビュー A を作成します。
2. ビュー A への参照を持つビュー B を作成します。ビュー A が使用されている場合、これは妥当性チェックおよび再コンパイルされています。

方法 2 は少ないステップでできますが、ビューの作成で使用する FORCE キーワードにエラーがあるかもしれません。ビュー作成中にエラーが発生したかどうかを確認するには、USER_ERRORS カタログ・ビューを問い合わせる必要があります。ビュー作成文の中にエラーがないことが保証されている場合にのみ、この方法を使用してください。

また、使用時にエラーのためにビューを再コンパイルできない場合、ALTER VIEW COMPILE コマンドを使用し、ビューを手動で再コンパイルする必要があります。

両方の方法の実装についてみてみます。

循環参照を示す表および型

まず、リレーショナル表および対応付けられたオブジェクト型を設定します。表がオブジェクトを含んでいても、それはオブジェクト表ではありません。データ・オブジェクトにアクセスするために、後でオブジェクト・ビューを作成します。

emp 表は従業員情報を格納します。

```
CREATE TABLE emp
(
    empno    NUMBER PRIMARY KEY,
    empname  VARCHAR2(20),
    salary   NUMBER,
    deptno   NUMBER
);
```

emp_t 型は部門への参照を含みます。*emp_t* 型を正常に作成するには、ダミーの部門型が必要です。

```
CREATE TYPE dept_t;  
/
```

従業員型は部門への参照を含みます。

```
CREATE TYPE emp_t AS OBJECT  
(  
    eno NUMBER,  
    ename VARCHAR2(20),  
    salary NUMBER,  
    deptref REF dept_t  
);  
/
```

従業員への参照のリストを NESTED TABLE として表します。

```
CREATE TYPE employee_list_ref_t AS TABLE OF REF emp_t;  
/
```

部門表は、通常のリレーショナル表です。

```
CREATE TABLE dept  
(  
    deptno          NUMBER PRIMARY KEY,  
    deptname        VARCHAR2(20),  
    deptstreet      VARCHAR2(20),  
    deptcity        VARCHAR2(10),  
    deptstate       CHAR(2),  
    deptzip         VARCHAR2(10)  
);
```

オブジェクト・ビューを作成するには、リレーショナル表から列にマップするオブジェクト型が必要です。

```
CREATE TYPE address_t AS OBJECT  
(  
    street          VARCHAR2(20),  
    city            VARCHAR2(10),  
    state           CHAR(2),  
    zip             VARCHAR2(10)  
);  
/
```

前に不完全な型を作成しましたが、ここで再定義します。

```
CREATE OR REPLACE TYPE dept_t AS OBJECT
(
    dno          NUMBER,
    dname        VARCHAR2(20),
    deptaddr     address_t,
    empreflist   employee_list_ref_t
);
/
```

循環参照を持つオブジェクト・ビューの作成

基礎となるリレーショナル表の定義ができたので、次は、この上にオブジェクト・ビューを作成します。

方法1: ビューをコンパイルする

まず、*deptref* 列に NULL を持つ従業員ビューを作成します。後で、この列を参照にします。

```
CREATE VIEW emp_view OF emp_t WITH OBJECT IDENTIFIER(eno)
AS SELECT e.empno, e.empname, e.salary,
        NULL
FROM emp e;
```

次に、従業員オブジェクトへの参照を含む部門ビューを作成します

```
CREATE VIEW dept_view OF dept_t WITH OBJECT IDENTIFIER(dno)
AS SELECT d.deptno, d.deptname,
        address_t(d.deptstreet,d.deptcity,d.deptstate,d.deptzip),
        CAST( MULTISET (
            SELECT MAKE_REF(emp_view, e.empno)
            FROM emp e
            WHERE e.deptno = d.deptno)
        AS employee_list_ref_t)
FROM dept d;
```

従業員全体のオブジェクトのかわりに、従業員オブジェクトへの参照のリストを作成します。ここで、部門ビューへの参照を持つ従業員ビューを再作成します。

```
CREATE OR REPLACE VIEW emp_view OF emp_t WITH OBJECT IDENTIFIER(eno)
AS SELECT e.empno, e.empname, e.salary,
        MAKE_REF(dept_view, e.deptno)
FROM emp e;
```

これでビューが作成されました。

方法 2：ビューを使用時にコンパイルする

ビューの作成文に構文エラーがないことが保証される場合、FORCE キーワードを使用することで、参照先のビューが存在しなくても、ビューを作成できます。

まず、この時点で存在しない部門ビューへの参照を含む従業員ビューを作成します。このビューは、部門ビューが適切に作成されるまで問い合わせることができません。

```
CREATE FORCE VIEW emp_view OF emp_t WITH OBJECT IDENTIFIER(eno)
AS SELECT e.empno, e.empname, e.salary,
        MAKE_REF(dept_view, e.deptno)
FROM emp e;
```

次に、従業員オブジェクトへの参照を含む部門ビューを作成します。emp_view がすでに存在するため、ここで FORCE キーワードを使用する必要はありません。

```
CREATE VIEW dept_view OF dept_t WITH OBJECT IDENTIFIER(dno)
AS SELECT d.deptno, d.deptname,
        address_t(d.deptstreet,d.deptcity,d.deptstate,d.deptzip),
        CAST( MULTISET (
                SELECT MAKE_REF(emp_view, e.empno)
                FROM emp e
                WHERE e.deptno = d.deptno)
        AS employee_list_ref_t)
FROM dept d;
```

これによって部門ビューに対し問合せを実行し、NESTED TABLE 型の列 empreflist に格納された従業員オブジェクトに対する参照を解除することで、従業員オブジェクトを獲得できます。

```
SELECT Deref(e.COLUMN_VALUE)
FROM TABLE( SELECT e.empreflist FROM dept_view e WHERE e.dno = 100) e;
```

COLUMN_VALUE は、スカラー NESTED TABLE でスカラー値を表す特別な名前です。この場合、COLUMN_VALUE は NESTED TABLE empreflist の従業員オブジェクトへの参照を示します。

名前が「John」の、すべての従業員の従業員番号にのみアクセスすることもできます。

```
SELECT e.COLUMN_VALUE.eno
FROM TABLE(SELECT e.empreflist FROM dept_view e WHERE e.dno = 100) e
WHERE e.COLUMN_VALUE.ename like 'John%';
```

表形式に出力するには、部門表を NESTED TABLE の項目に結合し、参照のリストのネストを解除します。

```
SELECT d.dno, e.COLUMN_VALUE.eno, e.COLUMN_VALUE.ename
FROM dept_view d, TABLE(d.empreflist) e
WHERE e.COLUMN_VALUE.ename like 'John%'
AND d.dno = 100;
```

最後に、dept_view のかわりに emp_view を使用するために前述の問い合わせを再作成し、1つのビューから他のビューへナビゲートする方法を表示できます。

```
SELECT e.deptref.dno, Deref(f.COLUMN_VALUE)
FROM emp_view e, TABLE(e.deptref.empreflist) f
WHERE e.deptref.dno = 100
AND f.COLUMN_VALUE.ename like 'John%';
```

Oracle オブジェクトの設計上の考慮点

この章では、Oracle のオブジェクト・リレーショナル・モデルの実装およびパフォーマンス特性について説明します。ここで説明する内容は、論理データ・モデルを Oracle の物理的な実装にマップしたり、オブジェクト指向機能を使用するアプリケーションを開発する場合に役立ちます。

内容は次のとおりです。

- 列または行としてのオブジェクトの表現
- オブジェクト識別子 (OID) の記憶域上の考慮点
- ネストを解除する問合せを使用したリレーショナル形式でのオブジェクト・データの表示
- メソッド関数に対する言語の選択

この章を読む前に、Oracle オブジェクトの背景となる基本概念について理解しておいてください。

関連項目： Oracle オブジェクトの概念的な説明は、『Oracle8i 概要』を参照してください。Oracle オブジェクトを使用するための SQL 構文の詳細は、『Oracle8i SQL リファレンス』を参照してください。

列または行としてのオブジェクトの表現

オブジェクトは、列オブジェクトとしてリレーショナル表の列に格納するか、または行オブジェクトとしてオブジェクト表に格納できます。そのオブジェクトが含まれているリレーショナル・データベース・オブジェクトの外で意味を持つオブジェクト、または複数のリレーショナル・データベース・オブジェクトの間で共有されるオブジェクトは、行オブジェクトとして参照できるようにする必要があります。つまり、そのようなオブジェクトは、リレーショナル表の列に格納するかわりに、オブジェクト表に格納する必要があります。

たとえば、オブジェクト型 CUSTOMER のオブジェクトは、特定の発注書の外で意味を持ち、他から参照できるようにする必要があります。したがって、CUSTOMER オブジェクトは、オブジェクト表の行オブジェクトとして格納する必要があります。ただし、オブジェクト型 ADDRESS のオブジェクトは、特定の発注書の外ではあまり意味を持たず、発注書内の 1 つの属性になります。したがって、ADDRESS オブジェクトは、リレーショナル表またはオブジェクト表の列に列オブジェクトとして格納する必要があります。そのため、ADDRESS は、CUSTOMER 行オブジェクトの中の列オブジェクトになる場合があります。

列オブジェクトの記憶域

列オブジェクトの記憶域は、1 つの集合としてオブジェクトを形成する、同等のスカラー列集合の記憶域と同じです。違いは、オブジェクトのアトミック NULL 値およびその埋込みオブジェクト属性をメンテナンスするオーバーヘッドが加わることのみです。これらの値は、それぞれの列オブジェクトが NULL かどうか、およびその埋込みオブジェクト属性が NULL かどうかを指定するため、「NULL 標識」といわれます。ただし、NULL 標識は、列オブジェクトのスカラー属性が NULL かどうかは指定しません。Oracle では、別の方法でスカラー属性が NULL かどうかを判断します。

ある組織の構成員の識別番号、名前、住所および電話番号を持つ表を考えてみます。名前、住所および電話番号を保持するために、異なる 3 つのオブジェクト型を作成できます。まず、次の SQL 文を入力して、name_objtyp オブジェクト型を作成します。

```
CREATE TYPE name_objtyp AS OBJECT (  
  first      VARCHAR2(15),  
  middle     VARCHAR2(15),  
  last       VARCHAR2(15));
```

図 5-1 name_objtyp 型のオブジェクト・リレーショナル表現

NAME_OBJTYP型		
FIRST	MIDDLE	LAST
テキスト VARCHAR2(15)	テキスト VARCHAR2(15)	テキスト VARCHAR2(15)

次に、次の SQL 文を入力して、address_objtyp オブジェクト型を作成します。

```
CREATE TYPE address_objtyp AS OBJECT (  
  street      VARCHAR2(200),  
  city        VARCHAR2(200),  
  state       CHAR(2),  
  zipcode     VARCHAR2(20));
```

図 5-2 address_objtyp 型のオブジェクト・リレーショナル表現

ADDRESS_OBJTYP型			
STREET	CITY	STATE	ZIPCODE
テキスト VARCHAR2(200)	テキスト VARCHAR2(200)	テキスト CHAR(2)	数値 VARCHAR2(20)

最後に、次の SQL 文を入力して、phone_objtyp オブジェクト型を作成します。

```
CREATE TYPE phone_objtyp AS OBJECT (  
  location    VARCHAR2(15),  
  num         VARCHAR2(14));
```

図 5-3 phone_objtyp 型のオブジェクト・リレーショナル表現

PHONE_OBJTYP型	
LOCATION	NUM
テキスト VARCHAR2(15)	数値 VARCHAR2(14)

各構成員が複数の電話番号を持っている場合があるため、phone_objtyp オブジェクト型に基づいて NESTED TABLE 型 phone_ntabtyp を作成します。

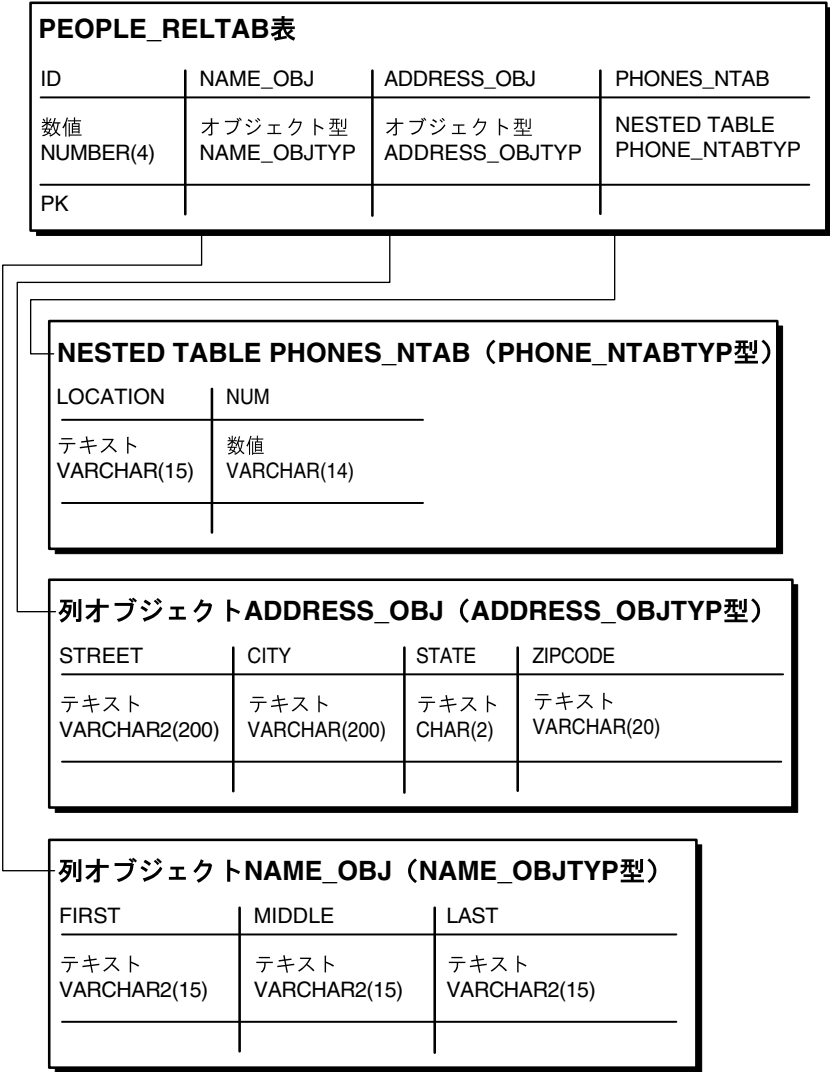
```
CREATE TYPE phone_ntabtyp AS TABLE OF phone_objtyp;
```

参照： NESTED TABLE の詳細は、5-14 ページの「[NESTED TABLE](#)」を参照してください。

これらのオブジェクト型が準備できたら、次の SQL 文を使用して、組織の構成員についての情報を保持する表を作成できます。

```
CREATE TABLE people_reltab (  
  id          NUMBER(4)    CONSTRAINT pk_people_reltab PRIMARY KEY,  
  name_obj    name_objtyp,  
  address_obj address_objtyp,  
  phones_ntab phone_ntabtyp)  
  NESTED TABLE phones_ntab STORE AS phone_store_ntab;
```

図 5-4 people_reltab リレーショナル表の表現



people_reltab 表には、name_obj、address_obj および phones_ntab の 3 つの列オブジェクトがあります。phones_ntab 列オブジェクトは、NESTED TABLE でもあります。

注意： name_obj オブジェクト、address_obj オブジェクト、phones_ntab NESTED TABLE および people_reltab 表は、この章の他の例でも使用します。

people_reltab 表に格納される各オブジェクトの記憶域は、オブジェクトの属性の記憶域と同じです。たとえば、name_obj オブジェクトに必要な記憶域は、NULL 標識のオーバーヘッドを除いて、first 属性、middle 属性、last 属性の組合せに対する記憶域と同じです。

COMPATIBLE パラメータが 8.1.0 以降に設定された場合、オブジェクトの NULL 標識およびその埋込みオブジェクト属性には、それぞれ 1 ビットが割り当てられます。したがって、(すべてのネスト・レベルのオブジェクトを含めて) n 個の埋込みオブジェクト属性を持つオブジェクトには、記憶域のオーバーヘッドが $\text{CEIL}(n/8)$ バイトあります。たとえば、people_reltab 表では、各行に対する NULL 情報のオーバーヘッドは、 $\text{CEIL}(3/8)$ つまり $\text{CEIL}(.375)$ になり、これは 1 バイトに丸められるため、1 バイトになります。この場合、各行には、name_obj、address_obj および phones_ntab の 3 つのオブジェクトがあります。

ただし、COMPATIBLE パラメータが 8.1.0 より下、たとえば 8.0.0 に設定された場合、記憶域は、次の計算によって決定されます。

$$\text{CEIL}(n/8) + 6$$

ここで、 n はオブジェクト内のすべての属性（スカラーおよびオブジェクト）の合計数です。したがって、たとえば people_reltab 表では、各行に対する NULL 情報のオーバーヘッドは次の計算から 7 バイトになります。

$$\text{CEIL}(4/8) + 6 = 7$$

$\text{CEIL}(4/8)$ は $\text{CEIL}(.5)$ で、これは 1 バイトに丸められます。この場合、各行の 3 つのオブジェクトと 1 つのスカラーがあります。

したがって、列オブジェクトを操作する際の記憶域のオーバーヘッドおよびパフォーマンスは、同等のスカラー列集合の場合と似ています。コレクション属性の記憶域は、5-12 ページの「[ネストを解除する問合せを使用したリレーショナル形式でのオブジェクト・データの表示](#)」を参照してください。

参照： CEIL の詳細は、『Oracle8i SQL リファレンス』を参照してください。

オブジェクト表の行オブジェクトの記憶域

行オブジェクトは、オブジェクト表に格納されます。オブジェクト表は、オブジェクトを格納し、このオブジェクト属性に対してリレーショナル表と同様にアクセスできるリレーショナル・ビューを提供する特殊な表です。オブジェクト表は、オブジェクト表に格納された、オブジェクトの最上位の属性に対応するデータ型の列を持つリレーショナル表と、論理的および物理的に似ています。主な違いは、オブジェクト表には、追加のオブジェクト識別子 (OID) の列および索引をオプションで含むことができることです。

オブジェクト識別子 (OID) の記憶域および OID 索引 Oracle では、デフォルトで、すべての行オブジェクトに OID と呼ばれる一意で不変のオブジェクト識別子が割り当てられます。OID を利用することで、対応する行オブジェクトを、他のオブジェクトまたはリレーショナル表から参照できます。このような参照は、REF と呼ばれる組込みデータ型によって表されます。REF によって、指定されたオブジェクト型の行オブジェクトへの参照がカプセル化されます。

デフォルトでは、オブジェクト表にはシステムによって生成される OID 列が含まれるため、各行オブジェクトには、グローバルに一意な OID が割り当てられます。この OID 列は、効率的な OID ベースの検索ができるように自動的に索引付けされます。OID 列は、16 バイトの主キー列を別に持つと同じ効果があります。

主キー・ベースの OID 主キー列が使用可能な場合、16 バイトの OID 列およびその索引をメンテナンスする際の記憶域およびパフォーマンスのオーバーヘッドを回避できます。システムによって生成される OID を使用するかわりに、CREATE TABLE 文を使用して、システムが主キー列を表のオブジェクトの OID に使用するように指定できます。したがって、既存の列をオブジェクトの OID として使用するか、または Oracle によって生成される 16 バイトのグローバルに一意な OID よりも小さいアプリケーション生成の OID を使用できます。

オブジェクト比較のパフォーマンス

オブジェクト型に定義されたマップ・メソッドまたはオーダー・メソッドを起動することによって、オブジェクトを比較できます。マップ・メソッドは、オブジェクトの順序付けを行う際にスカラー値に変換します。オブジェクトをスカラー値にマップすると、システムによって効率よく順序付けできるため、スカラー値への変換が可能な場合、これをお勧めします。

ORDER BY または GROUP BY 処理のためにオブジェクトのソートが必要な場合、オブジェクトのマップ方法がパフォーマンスに大きく影響します。

これは、オブジェクトが他のオブジェクトと何度も比較される必要があり、最初にオブジェクトをスカラー値にマップできると、より効率よく比較できるためです。比較セマンティクスが非常に複雑な場合、またはオブジェクトを比較用のスカラー値にマップできない場合、指定された2つのオブジェクトに対して、オブジェクト作成者によって決定された順序を戻すオーダー・メソッドを定義できます。オーダー・メソッドは、マップ・メソッドほど効率的でないため、オーダー・メソッドを使用するとパフォーマンスが低下する場合があります。どのオブジェクト型でも、マップ・メソッドまたはオーダー・メソッドのどちらか一方を実装できますが、両方はできません。

ここでもう一度、4つの文字属性、STREET、CITY、STATE および ZIPCODE で構成されるオブジェクト型 ADDRESS を考えてみます。ここで、各オブジェクトは簡単にスカラー値に変換できるため、最も効率的な比較メソッドはマップ・メソッドです。たとえば、州によってすべてのオブジェクトを順序付けるマップ・メソッドを定義できます。

一方、イメージなどのバイナリ・オブジェクトを比較する場合を考えてみます。この場合、比較セマンティクスが複雑すぎてマップ・メソッドは使用できない可能性があります。その場合は、オーダー・メソッドを使用して比較を実行できます。たとえば、各イメージの明度またはピクセル数によってイメージを比較するオーダー・メソッドを作成できます。

マップ・メソッドもオーダー・メソッドもないオブジェクト型のオブジェクトでは、等価比較のみが許可されます。この場合、Oracle では、対応するオブジェクト属性のフィールド同士が定義順に比較されます。比較が失敗した時点で、FALSE 値が戻されます。すべての点で比較が一致すると、TRUE 値が戻されます。ただし、オブジェクトが LOB 属性のコレクションを持つ場合、フィールド・ベースのオブジェクトの比較は行われません。このようなオブジェクトの比較を実行するためにはマップ・メソッドまたはオーダー・メソッドが必要です。

オブジェクト識別子 (OID) の記憶域上の考慮点

REF は、オブジェクト識別子 (OID) を使用してオブジェクトを指します。システムによって生成される OID または主キー・ベースの OID のどちらかを使用できます。この2種類の OID の違いについては、5-7 ページの「[オブジェクト表の行オブジェクトの記憶域](#)」を参照してください。オブジェクト表に対して、システムが生成する OID を使用する場合、Oracle がこれらの OID を格納する列の索引をメンテナンスします。索引には記憶領域が必要で、個々の行オブジェクトには、システムが生成する OID があります。OID には、行あたり 16 バイトの記憶域が必要です。

システムによって生成される OID のかわりに、オブジェクト識別子として主キーを使用することで、必要な記憶領域の増加を回避できます。リレーショナル表の外部キーと同様の方法で、主キー・ベースのオブジェクト識別子を持つ行オブジェクトへの参照を格納する列に、参照整合性を施行できます。

ただし、主キーのサイズが 16 バイト以上ある場合に、主キー・ベースの OID を使用すると、システムによって生成される OID より多くの領域が必要になる場合があります。さらに、個々の主キー・ベースの OID は、ローカルに一意です（グローバルに一意である必要はありません）。グローバルに一意の識別子が必要な場合は、主キーがグローバルに一意であることを保証するか、またはシステムによって生成される OID を使用する必要があります。

REF の記憶域サイズ

REF には、次の 3 つの論理コンポーネントが含まれます。

- 参照されるオブジェクトの OID。システムによって生成される OID の長さは 16 バイトです。主キー・ベースの OID のサイズは、主キー列のサイズに依存します。
- 参照されるオブジェクトが含まれている表またはビューの OID。この長さは 16 バイトです。
- ROWID ヒント。この長さは 10 バイトです。

REF 列での整合性制約

REF 列での参照整合性制約によって、REF が指し示す行オブジェクトが確実に存在することを保障できます。REF に関する参照整合性制約では、リレーショナル・データで主キー / 外部キー関連を指定するのと同じ関連が作成されます。一般に、参照整合性制約は、REF に対して行オブジェクトが存在することを保証する唯一の方法であるため、できるだけ参照整合性制約を使用してください。ただし、NESTED TABLE にある REF では、参照整合性制約を指定できません。

SCOPED REF のパフォーマンスおよび記憶域上の考慮点

SCOPED REF は、指定したオブジェクト表への参照のみを含むように制限されます。列のデータ型、コレクション要素またはオブジェクト型属性が REF になるように宣言する場合、SCOPED REF を指定できます。一般に、SCOPED REF はより効率よく格納されるため、特別な理由がないかぎり、常に、SCOPED REF を使用します。SCOPED REF は、OID のみがディスクに格納されるため、長さは 16 バイトです。サイズが小さい上に、SCOPED REF を参照解除する問合せは、オプティマイザによって効率のよい結合に最適化できることがあります。SCOPED REF 以外の REF にこの最適化はできません。これは、問合せの最適化時に、SCOPED REF 以外の REF に含まれている表をオプティマイザが判断できないためです。

ただし、参照整合性制約とは異なり、SCOPED REF では参照される行オブジェクトの存在は保証されません。保証されるのは、参照されるオブジェクト表が存在することのみです。

したがって、行オブジェクトに対して SCOPED REF を指定した後でその行オブジェクトを削除した場合は、参照対象となるオブジェクトが存在しなくなるため、SCOPED REF は参照先がない REF になります。

注意： 参照整合性制約には、暗黙的に有効範囲が付きます。

アプリケーション設計上、参照されるオブジェクトが複数の表に分散している場合は、SCOPED REF 以外の REF が役立ちます。ROWID ヒントは SCOPED REF に対して無視されるため、この後「[WITH ROWID オプションを使用したオブジェクト・アクセスの高速化](#)」で説明されるように、ROWID ヒントによるパフォーマンスの向上の方が、SCOPED REF を使用した場合の記憶域の節約および問合せの最適化よりも重視される場合は、SCOPED REF 以外の REF を使用してください。

SCOPED REF の索引作成

CREATE INDEX コマンドを使用して、SCOPED REF 列に索引を作成できます。それによって、その索引を使用して、SCOPED REF を参照解除する問合せを効率よく評価できます。このような問合せは、暗黙的に結合を利用した問合せに変換されます。Oracle では、ある問合せに対して、SCOPED REF 列の索引を使用して結合を効率よく評価できます。

たとえば、オブジェクト型 address_objtyp を使用して address_objtyp という名前のオブジェクト表を作成するとします。

```
CREATE TABLE address_objtab OF address_objtyp ;
```

次に、住所に対して REF が使用されること以外は、5-2 ページの「[列オブジェクトの記憶域](#)」で説明した people_reltab 表と同じ定義を持つ people_reltab2 表を作成できます。

```
CREATE TABLE people_reltab2 (
  id          NUMBER(4)    CONSTRAINT pk_people_reltab2 PRIMARY KEY,
  name_obj    name_objtyp,
  address_ref REF address_objtyp SCOPE IS address_objtab,
  phones_ntab phone_ntabtyp)
  NESTED TABLE phones_ntab STORE AS phone_store_ntab2 ;
```

これで address_ref 列に索引を作成できます。

```
CREATE INDEX address_ref_idx ON people_reltab2 (address_ref) ;
```

次の問合せで address_ref が参照解除されます。

```
SELECT id FROM people_reltab2 p
WHERE p.address_ref.state = 'CA' ;
```

この問合せが実行されるとき、効率的に評価を行うために address_ref_idx 索引が使用されます。ここで、address_ref は、address_objtab オブジェクト表に格納される住所への参照を格納した SCOPED REF 列です。前述の問合せは、結合を持つ問合せに暗黙的に変換されます。

```
SELECT p.id FROM people_reltab2 p, address_objtab a
WHERE p.address_ref = ref(a) AND a.state = 'CA' ;
```

Oracle のオプティマイザは、address_objtab を外部表としてネステッド・ループ・ジョインを実行し、SCOPED REF 列 address_ref の索引を使用して、一致する住所を検索する計画を作成することがあります。

WITH ROWID オプションを使用したオブジェクト・アクセスの高速化

REF 列に対して WITH ROWID オプションを指定する場合、REF で参照されるオブジェクトの ROWID がメンテナンスされます。そして、REF に含まれる ROWID を使用することで、OID 索引から ROWID をフェッチしなくても、Oracle は、参照されるオブジェクトを直接検索できます。したがって、ROWID ヒントを指定するには、WITH ROWID オプションを使用します。ROWID で REF の記憶域要件が 10 バイト増えるため、これをメンテナンスするにはより多くの記憶領域が必要です。

OID 索引検索をバイパスすると、アプリケーションでの REF の横断（ナビゲーション・アクセス）のパフォーマンスが向上します。実際のパフォーマンス向上は、次の要因から、アプリケーションによって異なります。

- OID 索引の大きさ
- OID 索引がバッファ・キャッシュにキャッシュされているかどうか
- アプリケーションが実行する REF 横断の数

WITH ROWID オプションを使用する場合、Oracle は、REF 内の OID で行オブジェクトの OID をチェックするため、このオプションは単なるヒントにすぎません。2 つの OID が一致しない場合、かわりに OID 索引が使用されます。ROWID ヒントは、SCOPED REF、参照整合性制約付き REF および主キー・ベースの REF に対しては利用できません。

ネストを解除する問合せを使用したリレーショナル形式でのオブジェクト・データの表示

コレクションに対してネストを解除する問合せを実行することで、データをフラットな（リレーショナルな）形式で表示できます。ネストを解除する問合せは、NESTED TABLE および VARRAY の両方で実行できます。この項では、ネストを解除する問合せの例を示します。

NESTED TABLE は、次の例のように、TABLE 構文を使用して問合せ用にネストを解除できます。

```
SELECT p.name_obj, n.num
      FROM people_reltab p, TABLE(p.phones_ntab) n ;
```

ここで、phones_ntab は、NESTED TABLE の属性 phones_ntab を指定します。子行を持たない親行も確実に取り出されるようにするには、次のように外部結合構文を使用します。

```
SELECT p.name_obj, n.num
      FROM people_reltab p, TABLE(p.phones_ntab) (+) n ;
```

最初の場合、問合せが（FROM 句の NESTED TABLE の列以外に）親表の列を参照しない場合、問合せは記憶表に対してのみ実行されるよう最適化されます。

TABLE 構文を使用して VARRAY を問い合わせることもできます。たとえば、NESTED TABLE の属性 phones_ntab が、かわりに phones_var という名前の VARRAY になるとします。この場合も、次の例のように、TABLE 構文を使用して VARRAY を問い合わせることができます。

```
SELECT p.name_obj, n.num
      FROM people_reltab p, TABLE(p.phones_var) n ;
```

ネストを解除する問合せ構文は、VARRAY および NESTED TABLE のどちらに対しても同じです。

ネストを解除する問合せでのプロシージャおよびファンクションの使用

ネストを解除する問合せを実行するためのプロシージャおよびファンクションを作成できます。たとえば、location が「home」である電話番号のみを戻す、home_phones() というファンクションを作成できます。home_phones() ファンクションを作成するには、次の例のようなコードを入力します。

```
CREATE OR REPLACE FUNCTION home_phones(allphones IN phone_ntabtyp)
RETURN phone_ntabtyp IS
  homephones phone_ntabtyp := phone_ntabtyp();
  indx1      number;
  indx2      number := 0;
BEGIN
  FOR indx1 IN 1..allphones.count LOOP
```

```

IF
    allphones(indx1).location = 'home'
THEN
    homephones.extend;    -- extend the local collection
    indx2 := indx2 + 1;    -- extend the local collection
    homephones(indx2) := allphones(indx1);
END IF;
END LOOP;

RETURN homephones;
END;
/

```

ここで、人のリストおよびその自宅の電話番号を問い合わせるには、次のように入力します。

```

SELECT p.name_obj, n.num
FROM people_reltab p, table(
    CAST(home_phones(p.phones_ntab) AS phone_ntabtyp)) n ;

```

自宅の電話番号がリストされていない人も含めて、人のリストおよびその自宅の電話番号を問い合わせるには、次のように入力します。

```

SELECT p.name_obj, n.num
FROM people_reltab p,
    TABLE(CAST(home_phones(p.phones_ntab) AS phone_ntabtyp))(+) n ;

```

参照： TABLE 構文の使用の詳細は、『Oracle8i SQL リファレンス』を参照してください。

VARARRAY の記憶域上の考慮点

格納される VARARRAY のサイズは、VARARRAY 内の現行の要素数にのみ依存し、保持できる最大要素数には関係しません。VARARRAY の記憶域には、NULL 情報などのオーバーヘッドが少しかかります。したがって、格納される VARARRAY のサイズは、(要素のサイズ) × (要素数) より少し大きくなります。

VARARRAY は、RAW または BLOB として列に格納されます。VARARRAY が定義されるときに、宣言された VARARRAY の LIMIT (要素の上限値) を使用して計算される VARARRAY の最大許容サイズを基にして、VARARRAY の格納方法が決定されます。サイズが約 4000 バイトを超える場合、VARARRAY は BLOB に格納されます。4000 バイトを超えない場合、VARARRAY は RAW として列自体に格納されます。さらに、Oracle ではインライン LOB がサポートされます。

したがって、(LOB ロケータ用に何バイトか予約された) 大きな VARRAY の最初の 4000 バイトに相当する要素は、その行自体の列に格納されます。

VARRAY および NESTED TABLE のパフォーマンス

アプリケーションでコレクション全体が 1 つの単位として操作される場合、NESTED TABLE よりも VARRAY の方が効率よく機能します。VARRAY はまとめて格納され、NESTED TABLE とは異なり、データを取り出すための結合は必要ありません。

VARRAY の問合せ

ネストを解除する構文を、NESTED TABLE へのアクセスの場合と同様の方法で、VARRAY 列へのアクセスにも使用できます。

参照： 詳細は、5-12 ページの「[ネストを解除する問合せを使用したリレーショナル形式でのオブジェクト・データの表示](#)」を参照してください。

VARRAY の更新

VARRAY の要素単位の更新は、サポートされていません。このため、VARRAY が更新される場合、古いコレクション全体が新しいコレクションで置き換えられます。

NESTED TABLE

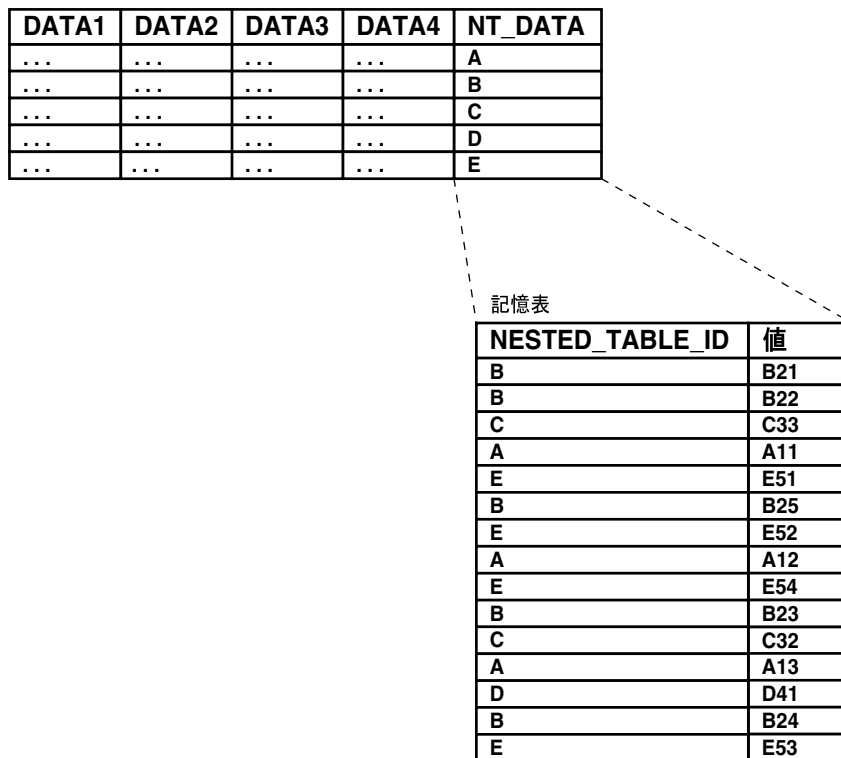
次の項では、NESTED TABLE を使用するための設計上の考慮点について説明します。

NESTED TABLE の記憶域

Oracle では、NESTED TABLE の行は別の記憶表に格納されます。システムが生成する NESTED_TABLE_ID (長さ 16 バイト) によって、親行およびそれに対応する記憶表の行が関連付けられます。

[図 5-5](#) に、記憶表の動作を示します。記憶表には、NESTED TABLE 列内の NESTED TABLE ごとの値が含まれています。この個々の値が、記憶表の 1 行に相当します。記憶表は、NESTED_TABLE_ID を使用して、個々の値に対する NESTED TABLE を追跡します。したがって、[図 5-5](#) では、NESTED TABLE A に属するすべての値が識別され、それに続いて NESTED TABLE B に属するすべての値も同様に識別されます。

図 5-5 NESTED TABLE 記憶域

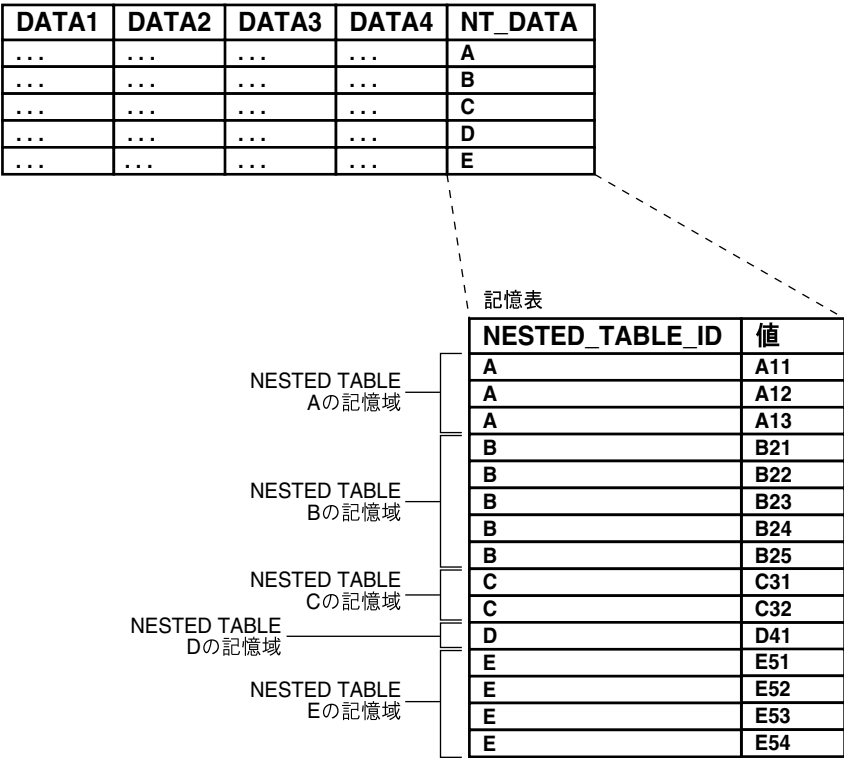


索引構成表 (IOT) における NESTED TABLE

NESTED TABLE が主キーを持つ場合、この表を索引構成表 (IOT) として構成できます。NESTED_TABLE_ID 列が、指定された親行に対する主キーの第一要素 (プレフィックス) である場合、その子行は物理的に 1 つにクラスタ化されます。このため、親行がアクセスされるときに、そのすべての子行を効率よく取り出せます。親行のみがアクセスされる場合でも、子行が親行と混ざりあうことがないため、同等の効率が維持されます。

図 5-6 に、NESTED TABLE が IOT である場合の記憶表の動作を示します。記憶表では、NESTED TABLE 内では NESTED_TABLE_ID ごとに値がグループ化されます。図 5-6 では、親表の NT_DATA 列にある NESTED TABLE ごとに、記憶表においてデータがグループ化されています。したがって、NESTED TABLE A のすべての値がグループ化され、それに続いて、NESTED TABLE B のすべての値も同様にグループ化されます。

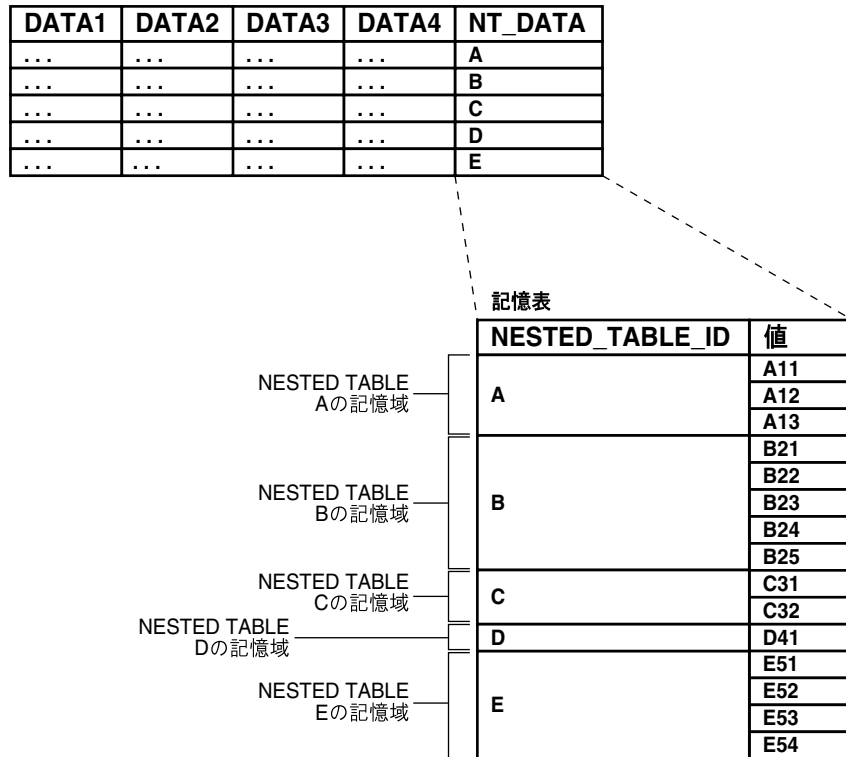
図 5-6 IOT 記憶域の NESTED TABLE



さらに、COMPRESS 句によって、IOT でプレフィックスを圧縮することが可能です。この圧縮では、それぞれの子行にある親のキーの重複部分を取り除きます。つまり、親キーが個々の子行で繰り返されることがないため、記憶域を大幅に節約することができます。

つまり、COMPRESS 句を使用して NESTED TABLE の圧縮を指定すると、記憶表に必要な領域の合計サイズが削減されます。これは、同じグループの各値に対して同一の NESTED_TABLE_ID が繰り返されないためです。かわりに、図 5-7 に示すように、NESTED_TABLE_ID はグループにつき 1 回のみ格納されます。

図 5-7 IOT 記憶域における圧縮付きの NESTED TABLE



オラクル社では、一般に、NESTED_TABLE_ID 列を主キーのプレフィックスとして使用して、NESTED TABLE を IOT に格納することをお勧めします。さらに、IOT でプレフィックス圧縮を有効にする必要があります。ただし、通常は NESTED TABLE を 1 つの単位として取り出すことがなく、子行をクラスタ化しない場合は、NESTED TABLE を IOT に格納しないでください。また、圧縮も指定しないでください。

NESTED TABLE の索引

(IOT でなく) ヒープ表に格納された NESTED TABLE に対しては、記憶表の NESTED_TABLE_ID 列に索引を作成する必要があります。対応する親表の ID 列の索引は、表の作成時に自動的に作成されます。NESTED_TABLE_ID 列に索引を作成することによって、NESTED TABLE の子行にさらに効率よくアクセスできるようになります。

これは、Oracle が `NESTED_TABLE_ID` 列を使用して、親表と NESTED TABLE の間に結合を実行する必要があるためです。

NESTED TABLE のロケータ

大規模な子集団の場合は、親行、および子集団に対するロケータを戻して、オンデマンドで子行にアクセスできるようにすることができます。子集団をフィルタすることもできます。NESTED TABLE のロケータを使用すると、すべての親に対して子行が不必要に転送されることを回避できます。

次のいずれかを実行して、NESTED TABLE のロケータを使用して子行にアクセスできます。

- OCI コレクション関数をコールします。このアクションは、`OCIColl*` 関数などのクライアント側コードでコレクション要素にアクセスするときに暗黙的に発生します。最初のアクセスで、コレクション全体が暗黙的に取り出されます。

参照： OIC コレクション関数の詳細は、『Oracle8i コール・インタフェース・プログラマーズ・ガイド』を参照してください。

- SQL を使用して、NESTED TABLE に対応する行を取り出します。このアクションの説明は、8-27 ページの「オブジェクト表 `PurchaseOrder_objtab`」で説明されています。

セット・メンバーシップ問合せの最適化

NESTED TABLE で特定の項目を検索する場合、セット・メンバーシップ問合せが役立ちます。たとえば、次の問合せは、子集団のメンバーシップ、特に、NESTED TABLE `phones_ntab`（親表 `people_reltab` にあります）に `home` という `location` があるかどうかをテストします。

```
SELECT * FROM people_reltab p
WHERE 'home' IN (SELECT location FROM TABLE(p.phones_ntab)) ;
```

Oracle では、子集団のメンバーシップをテストする問合せを、内部的にセミ・ジョインに変換することでより効率的に実行しています。ただし、この最適化は、`ALWAYS_SEMI_JOIN` 初期化パラメータが設定されている場合にのみ行われます。セミ・ジョインを実行する場合、このパラメータに対して有効な値は `MERGE` および `HASH` です。これらのパラメータ値によって、使用する結合メソッドが示されます。

注意： 前述の例では、home および location は子集団要素です。子集団要素がオブジェクト型である場合は、セット・メンバーシップ問合せを実行するマップ・メソッドまたはオーダー・メソッドが必要です。

NESTED TABLE での DML 操作

NESTED TABLE では DML 操作を実行できます。適切な SQL コマンドを使用することによって、NESTED TABLE への行の挿入、NESTED TABLE からの行の削除、および NESTED TABLE の既存の行の更新が可能になります。これらの操作では、NESTED TABLE は TABLE 副問合せによって識別されます。次の例では、NESTED TABLE 型の列 phones_ntab への電話番号の挿入も含めて、people_reltab 表に新しい人を挿入します。

```
INSERT INTO people_reltab values (
    0001,
    name_objtyp(
        'john', 'william', 'foster'),
    address_objtyp(
        '111 Maple Road', 'Fairfax', 'VA', '22033'),
    phone_ntabtyp(
        phone_objtyp('home', '650.331.1222'),
        phone_objtyp('work', '650.945.4389')));
```

次の例では、people_reltab 表にある識別番号が 0001 の既存の人物に対して、NESTED TABLE phones_ntab に電話番号を挿入します。

```
INSERT INTO TABLE(SELECT p.phones_ntab FROM people_reltab p WHERE p.id = '0001')
VALUES ('cell', '650.331.9337');
```

特定の NESTED TABLE を削除するには、次の例のように、親行で NESTED TABLE 列を NULL に設定します。

```
UPDATE people_reltab SET phones_ntab = NULL WHERE id = '0001';
```

NESTED TABLE を一度削除すると、再作成するまで値を挿入できません。識別番号が 0001 である人物に対して、NESTED TABLE の列オブジェクト phones_ntab に NESTED TABLE を再作成するには、次の SQL 文を入力します。

```
UPDATE people_reltab SET phones_ntab = phone_ntabtyp() WHERE id = '0001';
```

NESTED TABLE を再作成するときに、値を挿入することもできます。

```
UPDATE people_reltab
```

```
SET phones_ntab = phone_ntabtyp(phone_objtyp('home', '650.331.1222'))
WHERE id = '0001' ;
```

NESTED TABLE に対する DML 操作によって、親行はロックされます。したがって、NESTED TABLE の他の行に対する変更であっても、特定の NESTED TABLE のデータへの変更は、一度に 1 つしかできません。ただし、同時変更をサポートする必要があるのは NESTED TABLE の一部のデータのみで、NESTED TABLE の他のデータではこのサポートが必要でない場合は、同時変更の必要なデータへの REF の使用を検討する必要があります。

たとえば、発注書処理するアプリケーションがある場合、この発注書には顧客情報および明細項目が含まれている可能性があります。この場合、顧客情報が頻繁に変更されることはないため、このデータに対して同時変更をサポートする必要はありません。一方、明細項目は頻繁に変更される可能性があります。同じ発注書にある明細項目について同時更新をサポートするには、この明細項目を別のオブジェクト表に格納して、これを NESTED TABLE に格納した REF で参照します。

他のコレクション内でのコレクションのネスト

コレクションの属性は、コレクション型（VARRAY または NESTED TABLE）にはできません。つまり、コレクション内にコレクションを持つことはできません。Oracle では、コレクションの直接のネストは 1 レベルしか許可されません。ただし、コレクションの属性は、コレクション属性を持つオブジェクトへの参照にできます。したがって、REF を使用することによって、間接的に複数レベルのコレクションを持つことができます。

たとえば、5-2 ページの「[列オブジェクトの記憶域](#)」で説明された name_objtyp、address_objtyp および phone_ntabtyp オブジェクト型を使用して、person_objtyp と呼ばれる新しいオブジェクト型を作成するとします。1 人が複数の電話番号を持つことがあるため、phone_ntabtyp オブジェクト型が NESTED TABLE になっていることに注意してください。

person_objtyp オブジェクト型を作成するには、次の SQL 文を実行します。

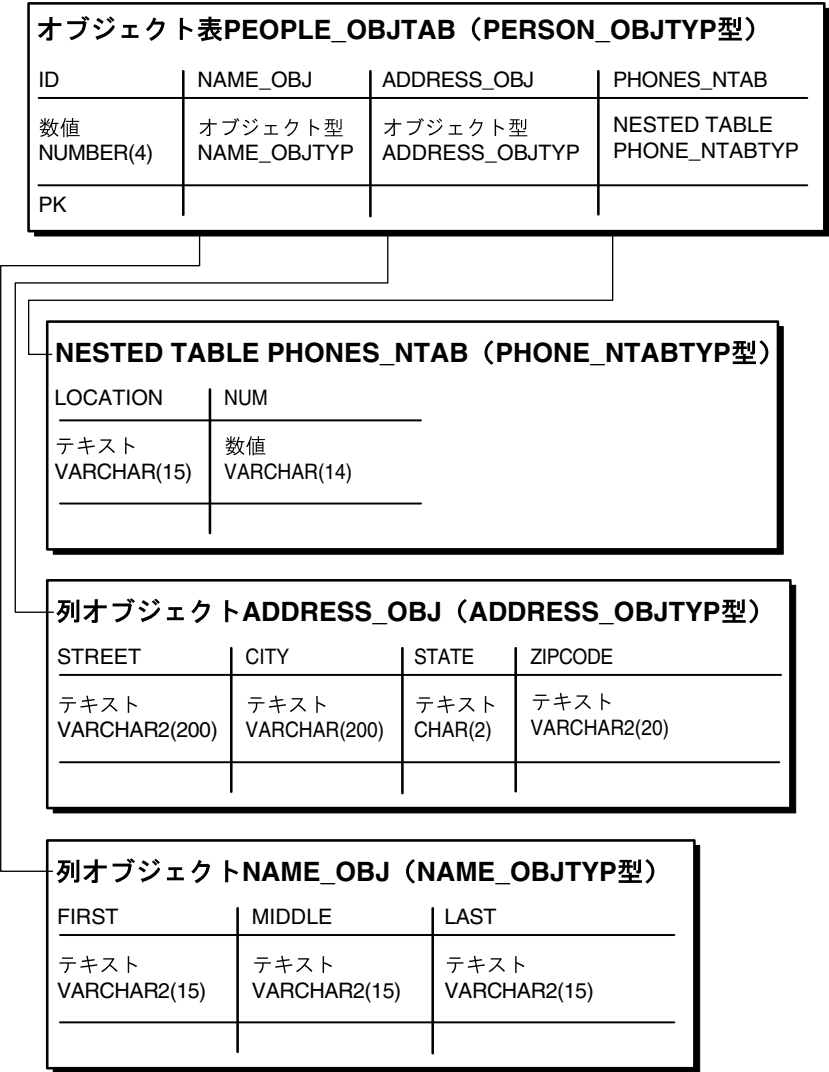
```
CREATE TYPE person_objtyp AS OBJECT (
    id          NUMBER(4),
    name_obj    name_objtyp,
    address_obj address_objtyp,
    phones_ntab phone_ntabtyp);
```

person_objtyp オブジェクト型の people_objtab というオブジェクト表を作成するには、次の SQL 文を実行します。

```
CREATE TABLE people_objtab OF person_objtyp (id PRIMARY KEY)
  NESTED TABLE phones_ntab STORE AS phones_store_ntab ;
```

people_objtab 表は、5-2 ページの「[列オブジェクトの記憶域](#)」で説明した people_reltab 表と同じ属性を持ちます。違いは、people_objtab が行オブジェクトを持つオブジェクト表であり、people_reltab 表が3つの列オブジェクトを持つリレーショナル表であるということです。

図 5-8 people_objtab オブジェクト表のオブジェクト・リレーショナル表現



ここで、people_objtab オブジェクト表にある行オブジェクトを他の表から参照できます。たとえば、次のものが含まれる projects_objtab 表を作成するとします。

- 各プロジェクトのプロジェクト識別番号
- 各プロジェクトのタイトル
- 各プロジェクトのプロジェクト・リード
- 各プロジェクトの説明
- 各プロジェクトに割り当てられたチームのメンバーを格納するコレクション型 NESTED TABLE

プロジェクト・リードに対しては people_objtab への REF を使用でき、チームに対しては REF の NESTED TABLE コレクションを使用できます。まず、person_objtyp オブジェクト型に基づいて、personref_ntabtyp という NESTED TABLE オブジェクト型を作成します。

```
CREATE TYPE personref_ntabtyp AS TABLE OF REF person_objtyp;
```

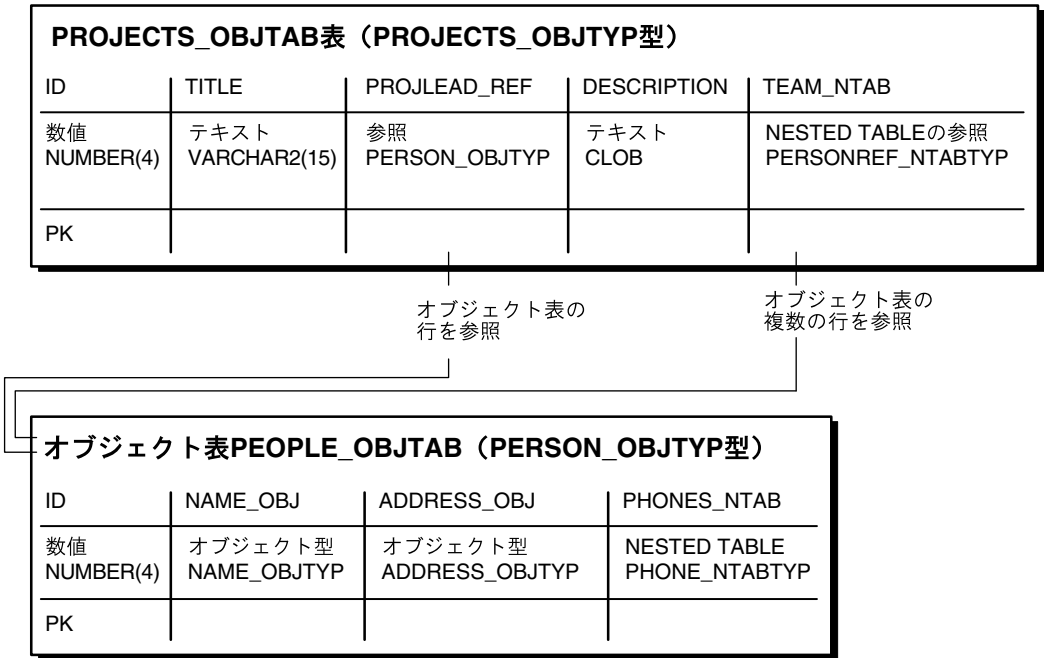
これで、オブジェクト表 projects_objtab を作成する準備ができました。まず、次の SQL 文を実行して、オブジェクト型 projects_objtyp を作成します。

```
CREATE TYPE projects_objtyp AS OBJECT (
  id          NUMBER(4),
  title       VARCHAR2(15),
  proglead_ref REF person_objtyp,
  description  CLOB,
  team_ntab   personref_ntabtyp);
```

次に、projects_objtyp に基づいて、オブジェクト表 projects_objtab を作成します。

```
CREATE TABLE projects_objtab OF projects_objtyp (id PRIMARY KEY)
  NESTED TABLE team_ntab STORE AS team_store_ntab ;
```

図 5-9 projects_objtab オブジェクト表のオブジェクト・リレーショナル表現



一度、people_objtab オブジェクト表および projects_objtab オブジェクト表が作成されると、ネストしたコレクションを間接的に持つことになります。つまり、projects_objtab 表には、people_objtab 表の人を指す REF の NESTED TABLE コレクションが含まれ、people_objtab 表の人には電話番号の NESTED TABLE コレクションがあります。

次のようにして、people_objtab 表に値を挿入できます。

```
INSERT INTO people_objtab VALUES (  
    0001,  
    name_objtyp('JOHN', 'JACOB', 'SCHMIDT'),  
    address_objtyp('1252 Maple Road', 'Fairfax', 'VA', '22033'),  
    phone_ntabtyp(  
        phone_objtyp('home', '650.339.9922'),  
        phone_objtyp('work', '510.563.8792')));
```

```
INSERT INTO people_objtab VALUES (
    0002,
    name_objtyp('MARY', 'ELLEN', 'MILLER'),
    address_objtyp('33 Spruce Street', 'McKees Rocks', 'PA', '15136'),
    phone_ntabtyp(
        phone_objtyp('home', '415.642.6722'),
        phone_objtyp('work', '650.891.7766')));
```

```
INSERT INTO people_objtab VALUES (
    0003,
    name_objtyp('SARAH', 'MARIE', 'SINGER'),
    address_objtyp('525 Pine Avenue', 'San Mateo', 'CA', '94403'),
    phone_ntabtyp(
        phone_objtyp('home', '510.804.4378'),
        phone_objtyp('work', '650.345.9232'),
        phone_objtyp('cell', '650.854.9233')));
```

さらに、次の例のように、REF 演算子を使用して people_objtab オブジェクト表から選択することによって、projects_objtab リレーショナル表に挿入できます。

```
INSERT INTO projects_objtab VALUES (
    1101,
    'Demo Product',
    (SELECT REF(p) FROM people_objtab p WHERE id = 0001),
    'Demo the product, show all the great features.',
    personref_ntabtyp(
        (SELECT REF(p) FROM people_objtab p WHERE id = 0001),
        (SELECT REF(p) FROM people_objtab p WHERE id = 0002),
        (SELECT REF(p) FROM people_objtab p WHERE id = 0003)));
```

```
INSERT INTO projects_objtab VALUES (
    1102,
    'Create PRODDb',
    (SELECT REF(p) FROM people_objtab p WHERE id = 0002),
    'Create a database of our products.',
    personref_ntabtyp(
        (SELECT REF(p) FROM people_objtab p WHERE id = 0002),
        (SELECT REF(p) FROM people_objtab p WHERE id = 0003)));
```

注意： この例では、REF を格納するために NESTED TABLE を使用していますが、REF を VARRAY に格納することもできます。つまり、REF の VARRAY を持つことができます。

メソッド関数に対する言語の選択

メソッド関数は、Oracle でサポートされる任意の言語（PL/SQL、Java、C など）で実装できます。特定のアプリケーション用に言語を選択する場合は、次の要因を考慮します。

- 使用しやすさ
- SQL コール
- 実行速度
- 同一 / 異なるアドレス空間

一般に、アプリケーションで主として計算を実行する場合には、C が適していますが、比較的多くのデータベース・コールを実行する場合は、PL/SQL または Java が適しています。

C で実装されるメソッドは、外部ルーチンを使用して、サーバーとは別のプロセスで実行されます。それに対して、Java または PL/SQL で実装されるメソッドは、サーバーと同じプロセスで実行されます。

メソッドの実装例

この項で説明される例には、異なる言語で実装されたメソッドを持つオブジェクト型が含まれています。この例では、オブジェクト型 `ImageType` は、ID 属性（一意に識別される `NUMBER`）および `IMG` 属性（イメージを格納する `BLOB`）を持ちます。オブジェクト型 `ImageType` は、次のメソッドを持ちます。

- メソッド `get_name()`。データベース内でイメージの名前を検索してフェッチします。このメソッドは、PL/SQL で実装されています。
- メソッド `rotate()`。イメージを回転します。このメソッドは、C で実装されています。
- メソッド `clear()`。指定された色の新しいイメージを戻します。このメソッドは、Java で実装されています。

C でメソッドを実装する場合は、外部 C ルーチンが含まれるライブラリを指す、`LIBRARY` オブジェクトを定義する必要があります。Java でメソッドを実装する場合、この例では、メソッドを持つ Java クラスがコンパイルされ、Oracle にアップロードされていると仮定します。

オブジェクト型の定義およびそのメソッドは次のとおりです。

```
CREATE TYPE ImageType AS OBJECT (  
    id    NUMBER,  
    img   BLOB,  
    MEMBER FUNCTION get_name() return VARCHAR2,  
    MEMBER FUNCTION rotate() return BLOB,  
    STATIC FUNCTION clear(color NUMBER) return BLOB  
);  
  
CREATE TYPE BODY ImageType AS  
    MEMBER FUNCTION get_name() RETURN VARCHAR2  
    AS  
        imgname VARCHAR2(100);  
    BEGIN  
        SELECT name INTO imgname FROM imgtab WHERE imgid = id;  
        RETURN imgname;  
    END;  
  
    MEMBER FUNCTION rotate() RETURN BLOB  
    AS LANGUAGE C  
    NAME "Crotate"  
    LIBRARY myCfuncs;  
  
    STATIC FUNCTION clear(color NUMBER) RETURN BLOB  
    AS LANGUAGE JAVA  
    NAME 'myJavaClass.clear(color oracle.sql.NUMBER) RETURN oracle.sql.BLOB';  
  
END;  
/
```

制限事項： タイプ・メソッドは、静的 Java メソッドのみにマップできません。

参照：

- 詳細は、『Oracle8i Java ストアド・プロシージャ開発者ガイド』を参照してください。
 - 言語の選択の詳細は、[第3章「Oracle プログラム環境のオブジェクト・サポート」](#)を参照してください。
-
-

静的メソッド

静的メソッドは、SELF 値が第 1 パラメータとして渡されない点で、メンバー・メソッドと異なります。SELF の値が問題ではないメソッドは、静的メソッドとして実装する必要があります。静的メソッドは、ユーザー定義コンストラクタに使用できます。

次の例は、コンストラクタに似たメソッドで、明示的な入力パラメータに基づいてその型のインスタンスを作成し、指定された表にそのインスタンスを挿入します。

```
CREATE OR REPLACE TYPE atype AS OBJECT(a1 NUMBER,
    STATIC PROCEDURE newa (
        p1          NUMBER,
        tabname     VARCHAR2,
        schname     VARCHAR2));

CREATE OR REPLACE TYPE BODY atype AS
    STATIC PROCEDURE newa (p1 NUMBER, tabname VARCHAR2, schname VARCHAR2)
    IS
        sqlstmt VARCHAR2(100);
    BEGIN
        sqlstmt := 'INSERT INTO '||schname||'.'||tabname|| ' VALUES (atype(:1))';
        EXECUTE IMMEDIATE sqlstmt USING p1;
    END;
END;
/

CREATE TABLE atab OF atype;
BEGIN
    atype.newa(1, 'atab', 'scott');
END;
```

実行者権限を使用した再使用コードの作成

任意のスキーマで使用できる汎用オブジェクト型を作成するには、`CREATE OR REPLACE TYPE` の `AUTHID CURRENT_USER` オプションを介して、実行者権限を使用する型を定義する必要があります。一般に、次の条件がいずれも真である場合に実行者権限を使用します。

- データにアクセスし操作するタイプ・メソッドがある
- これらのタイプ・メソッドを定義していないユーザーによって使用される必要がある

たとえば、SCOTT によって 5-28 ページの「静的メソッド」で作成された型 `atype` に対する実行権限を、ユーザー SARA に付与し、その後この型に基づいて表 `atab` を作成できます。

```
GRANT EXECUTE ON atype TO SARA ;
CONNECT SARA/TPK101 ;
CREATE TABLE atab OF scott.atype ;
```

ここで、ユーザー SARA が、次の文で `atype` を使用するとします。

```
BEGIN
    scott.atype.new(1, 'atab', 'SARA'); -- raises an error
END;
/
```

型定義者 (SCOTT) には、`new` プロシージャで挿入を実行するために必要な権限がないため、この文ではエラーが戻されます。このエラーは、実行者権限を使用して `atype` を定義することによって回避できます。ここで、まず両方のスキーマの `atab` 表を削除し、実行者権限を使用して `atype` を再作成します。

```
DROP TABLE atab ;
CONNECT SCOTT/TIGER ;
DROP TABLE atab ;
```

```
CREATE OR REPLACE TYPE atype AUTHID CURRENT_USER AS OBJECT(a1 NUMBER,
    STATIC PROCEDURE new(p1 NUMBER, tabname VARCHAR2, schname VARCHAR2));
```

```
CREATE OR REPLACE TYPE BODY atype AS
  STATIC PROCEDURE newa(p1 NUMBER, tabname VARCHAR2, schname VARCHAR2)
  IS
    sqlstmt VARCHAR2(100);
  BEGIN
    sqlstmt := 'INSERT INTO '||schname||'.'||tabname||' VALUES
      (scott.atype(:1))';
    EXECUTE IMMEDIATE sqlstmt USING p1;
  END;
END;
/
```

これで、ユーザー SARA が再度 atype を使用しようとした場合、文は正常に実行されます。

```
GRANT EXECUTE ON atype TO SARA ;
CONNECT SARA/TPK101 ;
CREATE TABLE atab OF scott.atype;

BEGIN
  scott.atype.newa(1, 'atab', 'SARA'); -- executes successfully
END;
/
```

プロシージャは定義者（SCOTT）権限でなく実行者（SARA）権限で実行されるため、このとき、文は正常に実行されます。

タイプ・メソッドの戻り値に関するファンクション索引

タイプ・メソッドの戻り値に関して、ファンクション索引を作成できます。次の例では、atype2 型のメソッド afun() にファンクション索引を作成します。

```
CREATE TYPE atype2 AS OBJECT
(
  a NUMBER,
  MEMBER FUNCTION afun RETURN NUMBER DETERMINISTIC
);

CREATE OR REPLACE TYPE BODY atype2 IS
  MEMBER FUNCTION afun RETURN NUMBER IS
  BEGIN
    RETURN self.a * 100;
  END;
END;
/
```

```
CREATE TABLE atab2 OF atype2 ;  
CREATE INDEX atab2_afun_idx ON atab2 x (x.afun()) ;
```

メソッドによっては、ファンクション索引を使用して SQL でのメソッド起動のパフォーマンスを向上させることができます。

制限事項： 入力として、LOB、REF、NESTED TABLE または VARRAY 型の引数をとるタイプ・メソッド、またはこれらの属性が含まれるオブジェクト型の引数をとるタイプ・メソッドに関しては、索引を作成できません。

参照： ファンクション索引の使用の詳細は、『Oracle8i SQL リファレンス』を参照してください。

リリース 8.1 での新しいオブジェクト形式

リリース 8.1 では、オブジェクトは、以前の形式よりも使用する記憶領域が小さく、パフォーマンス特性の優れた新しい形式で格納されます。パフォーマンスは、さらに効率的な転送プロトコルによっても向上しています。COMPATIBLE パラメータが 8.1.0 以上に設定された場合、新しく作成するすべてのオブジェクトは、自動的にリリース 8.1 形式で格納および転送されます。

リリース 8.0 データベースで作成されたオブジェクトをリリース 8.1 形式に変換するには、次のステップを実行します。

1. CREATE TABLE...AS SELECT... 文を使用して表を再作成します。
2. 表のデータをエクスポート / インポートします。

参照： 互換性および COMPATIBLE 初期化パラメータの詳細は、『Oracle8i 移行ガイド』を参照してください。

オブジェクト表およびオブジェクト列のレプリケーション

オブジェクト列およびオブジェクト表のレプリケーションは、まだサポートされていません。レプリケーションが必要な場合は、オブジェクト・ビューを使用してアプリケーション・オブジェクトをリレーショナル表に格納し、これをレプリケートします。オブジェクト・ビューを使用すると、オブジェクト・モデルおよびレプリケートされるデータの両方をデータベースに保存できます。

Oracle の継承の実装結果

継承を利用することで、スーパータイプを様々なレベルでカプセル化することができます。スーパータイプを他のオブジェクトに公開してはいけない場合、スーパータイプが表示されないようにするために必要なメソッドおよび属性を、サブタイプに含める必要があります。継承の実装結果を理解するには、Oracle8i が強力な型指定を持つシステムであることを認識しておくことも重要です。強力な型指定を持つシステムでは、属性を宣言するときに、属性の型を宣言する必要があります。宣言された型の値のみが、属性に格納できます。たとえば、Oracle8i コレクションは、強力に型指定されています。Oracle8i では、異種コレクション（複数の種類のコレクション）の実装は許可されません。

継承のシミュレート

Oracle タイプ・モデルでは、継承は直接サポートされません。ただし、現行の Oracle オブジェクト型を Java クラスにマップした後で、Java に固有の継承機能を導入できます。

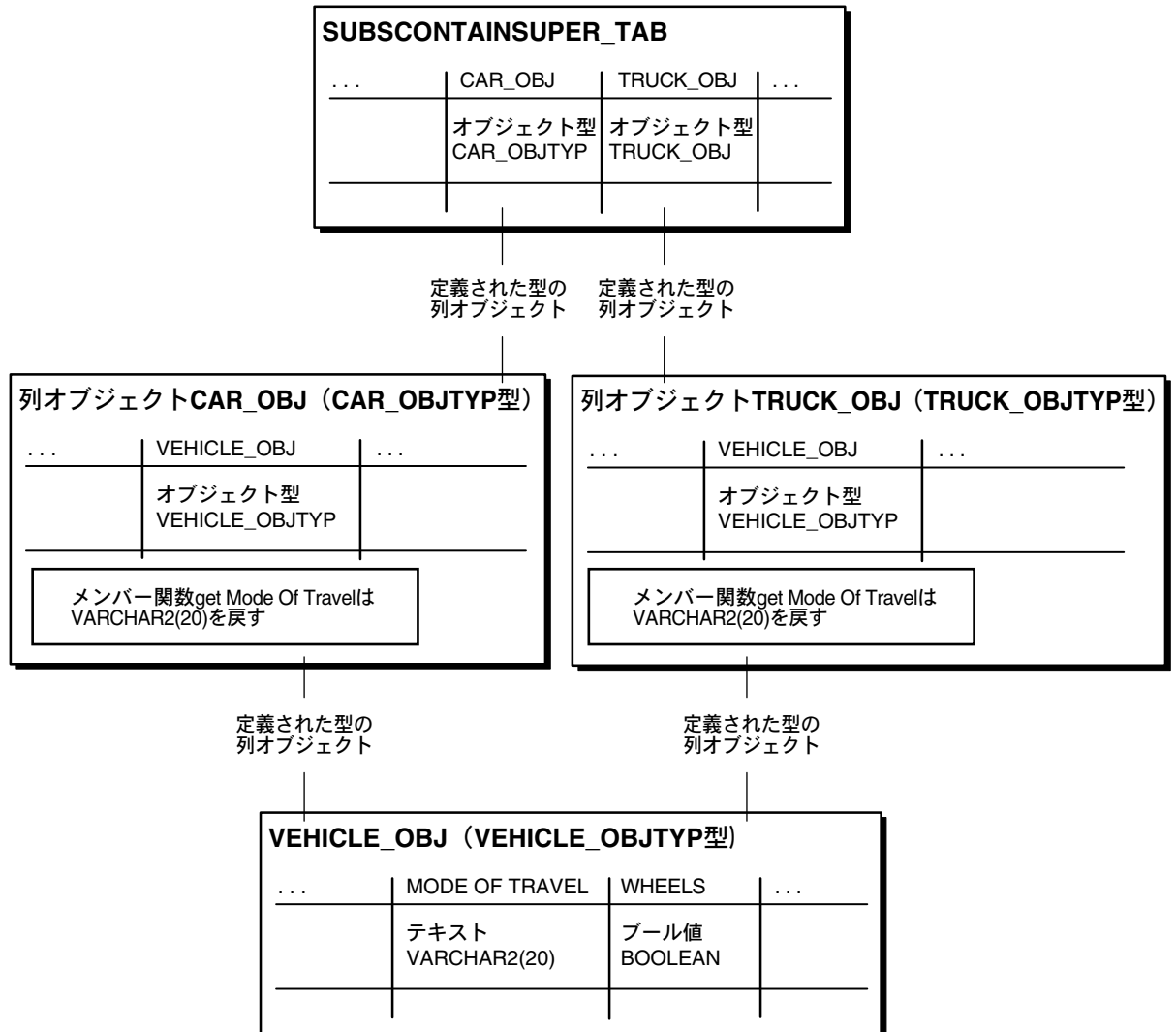
参照： Oracle オブジェクトの Java クラスへのマップの詳細は、『Oracle8i JDBC 開発者ガイドおよびリファレンス』および『Oracle8i SQLJ 開発者ガイドおよびリファレンス』を参照してください。

さらに、Oracle では継承をシミュレートできます。たとえば、次のいずれかを使用して、継承をシミュレートできます。

- スーパータイプを含むサブタイプ
- すべてのサブタイプを含むか、または参照するスーパータイプ
- デュアル・サブタイプ / スーパータイプの参照

スーパータイプを含むサブタイプ

図 5-10 オブジェクト・リレーショナル・スキーマ – スーパータイプを含むサブタイプ



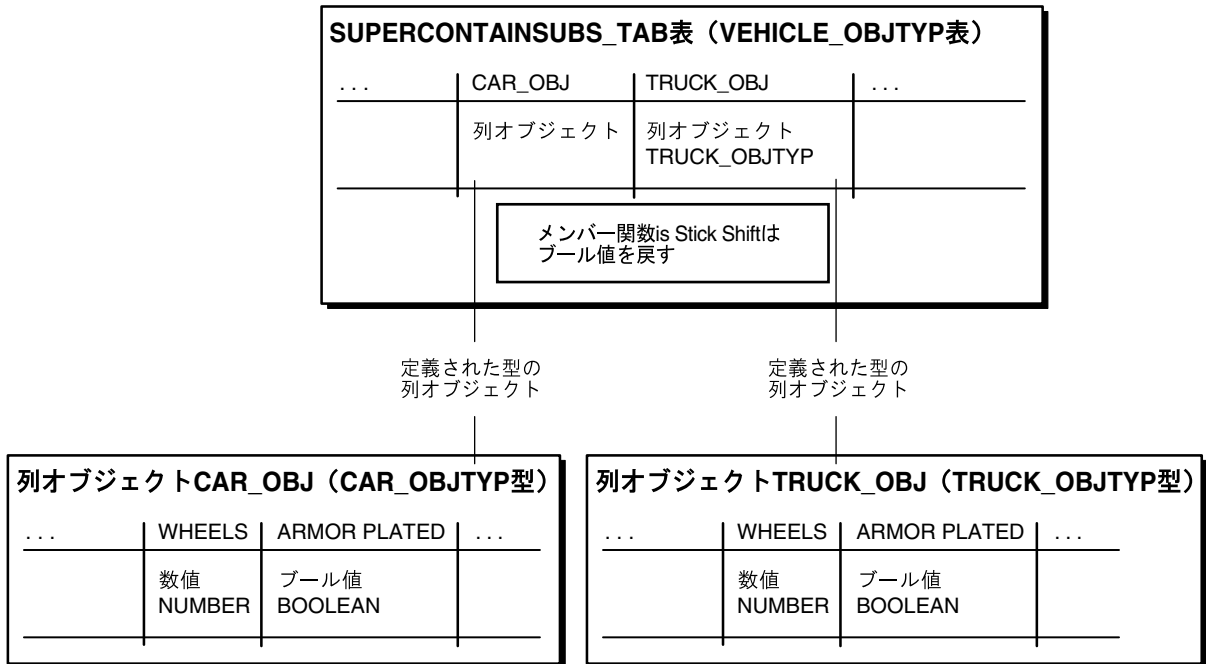
「スーパータイプを含むサブタイプ」の方法では、サブタイプに対する抽象化 / 汎用化の実装は非表示になります。各サブタイプは、オブジェクト・モデルの他の型に公開されます。スーパータイプは、他の型に公開されません。継承をシミュレートするために、設計オブジェクト・モデルのスーパータイプは、オブジェクト型として作成されます。また、サブタイプもオブジェクト型として作成されます。スーパータイプは、サブタイプの埋込み属性として定義されます。サブタイプおよびそのスーパータイプに対して実行できるすべてのメソッドは、サブタイプで定義する必要があります。

「スーパータイプを含むサブタイプ」の方法は、個々のサブタイプがオブジェクト・モデルの他のオブジェクトに対して特定の関連を持つ場合に使用されます。たとえば、Customer というスーパータイプは、Private Customer および Corporate Customer のサブタイプを持つことができます。Private Customers は、Personal Banking オブジェクトと関連を持ち、一方、Corporate Customers は、Commercial Banking オブジェクトと関連を持ちます。この環境では、Customer スーパータイプは、オブジェクト・モデルに含まれるその他のオブジェクトから参照できません。

Vehicle-Car/Truck の例では、Vehicle (スーパータイプ) は、サブタイプ Car および Truck に埋め込まれます。

すべてのサブタイプを含むスーパータイプ

図 5-11 オブジェクト・リレーショナル・スキーマ – すべてのサブタイプを含むスーパータイプ



「すべてのサブタイプを含むスーパータイプ」の方法では、サブタイプの実装は非表示になり、スーパータイプのみが公開されます。継承をシミュレートするために、設計オブジェクト・モデルにある指定されたスーパータイプに対するすべてのサブタイプは、オブジェクト型として作成されます。また、スーパータイプもオブジェクト型として作成されます。スーパータイプは、各サブタイプに対する属性を宣言します。また、スーパータイプもサブタイプ属性に対して 1 対 1 のルールを施行する制約を宣言します。サブタイプに対して実行できるすべてのメソッドは、スーパータイプで定義される必要があります。

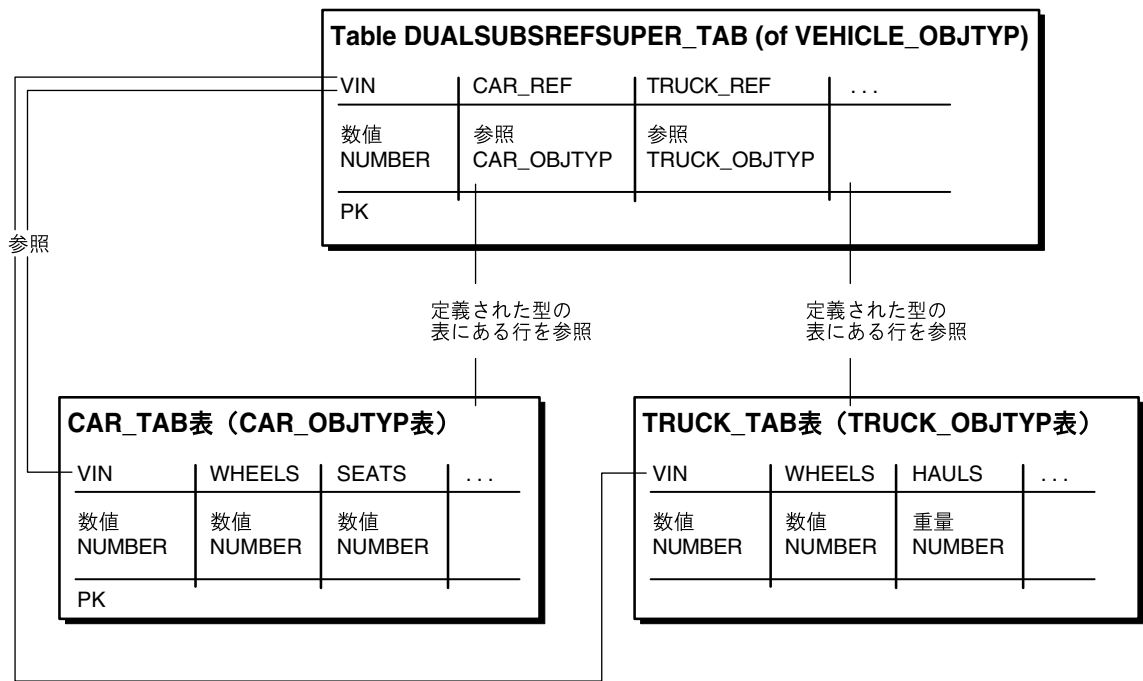
「すべてのサブタイプを含むスーパータイプ」の方法は、オブジェクト間の関連が主に 1 対多の場合に使用されます。たとえば、Customer は多くの Accounts を持つことができ、Bank は多くの Accounts を持つことができます。「スーパータイプを含むサブタイプ」の方法を使用する場合、各サブタイプに対するコレクションで多くの関連が必要です。Account がスーパータイプで、Checking および Savings がサブタイプである場合、Bank および Customer の両方に、Checking および Savings のコレクション（合計 4 つのコレクション）を実装する必要があります。

新しいアカウント・サブタイプを追加するには、Customer および Bank の両方に新しいアカウント・サブタイプをサポートするコレクションを追加する必要があります（1つの追加あたり2つのコレクション）。「すべてのサブタイプを含むスーパータイプ」の方法を使用することは、Customer および Bank が Account のコレクションを持つことを表します。Accounts へのサブタイプの追加は、アカウントのみの変更を表します。

Vehicle と Car/Truck の場合、Vehicle は Vehicle の埋込み属性としての Car および Truck とともに作成されます。

デュアル・サブタイプ/スーパータイプの参照

図 5-12 オブジェクト・リレーショナル・スキーマ デュアル・サブタイプ/スーパータイプの参照



オブジェクト・モデルにおいて、スーパータイプが多重度として対多の複数オブジェクト関連に関係しており、サブタイプが特定の関連を持っている場合、継承は、2つの継承方法を組み合わせて実装されます。スーパータイプは、オブジェクト型として実装されます。各サブタイプは、オブジェクト型として実装されます。スーパータイプは、各サブタイプに対する参照属性を実装します（0（ゼロ）参照関連）。スーパータイプは、サブタイプ属性のグループに対する OR 結合も実装します。

各サブタイプは、スーパータイプに対する参照属性を実装します (1 参照関連)。このようにして、スーパータイプおよびサブタイプの両方が、残りのオブジェクト・モデルから参照できます。

Vehicle と Car/Truck の場合、Vehicle は型として作成されます。Car および Truck は型として作成されます。Vehicle 型は、Car 属性に Truck 属性の OR 制約を使用して、Car および Truck の両方への参照を実装します。Car は、Vehicle を参照する属性を実装します。Truck は、Vehicle を参照する属性を実装します。

オブジェクトでの制約

Oracle では、型指定内での制約およびデフォルトはサポートされません。ただし、表を作成するときに、制約およびデフォルトを指定できます。

```
CREATE OR REPLACE TYPE customer_type AS OBJECT(  
    cust_id INTEGER);  
  
CREATE OR REPLACE TYPE department_type AS OBJECT(  
    deptno INTEGER);  
  
CREATE TABLE customer_tab OF customer_type (  
    cust_id default 1 NOT NULL);  
  
CREATE TABLE department_tab OF department_type (  
    deptno PRIMARY KEY);  
  
CREATE TABLE customer_tab1 (  
    cust customer_type DEFAULT customer_type(1)  
    CHECK (cust.cust_id IS NOT NULL),  
    some_other_column VARCHAR2(32));
```

型の発展

列オブジェクトまたは行オブジェクト、あるいはその両方の形式において、依存データを持つ型の定義は変更できません。ただし、列オブジェクトを持つ表の場合は、標準のリレーショナル表と同様の方法で列を削除および追加することで変更できます。

行オブジェクトを含む表は、列の削除、追加または変更によって変更することはできません。行オブジェクトを含む表を変更する必要がある場合、解決策として次のことを実行します。

1. 表のデータを一時表にコピーするか、または表のデータをエクスポートします。
2. 表を削除します。
3. 新しい定義で型を再作成します。
4. 表を再作成します。
5. 一時表から関連データをコピーするか、またはデータをインポートします。

型の発展が必要で、この解決策をとれない場合、列オブジェクトまたは行オブジェクトのかわりに、リレーショナル表に定義されたオブジェクト・ビューを使用します。こうすることで、オブジェクト型およびビューの定義を変更できます。

パフォーマンス・チューニング

アプリケーションのパフォーマンスの測定およびチューニングの詳細は、『Oracle8i パフォーマンスのための設計およびチューニング』を参照してください。特に重要なパフォーマンス要因は、次のとおりです。

- 統計を収集する ANALYZE コマンド
- SQL コマンドの実行をプロファイルする tkprof
- 問合せ計画を生成する EXPLAIN PLAN

Oracle オブジェクトでのパラレル問合せ

Oracle8i では、オブジェクトでのパラレル問合せを実行できます。ただし、次の制限があります。

- (ORDER BY、GROUP BY および SET 操作を使用して) 結合およびソートをパラレルに実行する問合せを行うには、MAP ファンクションが必要です。MAP ファンクションがない場合、問合せは自動的にシリアルになります。
- NESTED TABLE でのパラレル問合せはサポートされません。表に対するパラレル・ヒントまたはパラレル属性がある場合でも、問合せはシリアルになります。
- パラレル DML およびパラレル DDL は、オブジェクトでサポートされません。DML および DDL は、常にシリアルに実行されます。

Oracle オブジェクトの高度なトピック

このマニュアルの他の章では、Oracle オブジェクトで作業を開始するために必要なトピックについて説明しています。この章では、オブジェクト・リレーショナル技法を大規模なアプリケーションまたは複雑なスキーマに適用する場合について説明します。

この章で使用されている用語に詳しくない場合、またはその重要点を理解できない場合は、[第1章「Oracle オブジェクトの概要」](#) および [第7章「Oracle オブジェクトを使用したプログラミングに関する FAQ」](#) を参照してください。

オブジェクトの記憶域

Oracle は、オブジェクト型の複合構造を、単純な構造の表に自動的にマップします。

リーフ・レベル属性

オブジェクト型はツリー構造に似ており、ブランチが属性を表します。それ自体がオブジェクトである属性は、それ自体の属性に対してサブブランチを発生させます。

最終的に、各ブランチの終わりは、組込み型 (NUMBER、VARCHAR2、REF など) またはコレクション型 (VARRAY、NESTED TABLE など) の属性になります。元のオブジェクト型のこのようなリーフ・レベル属性は、それぞれ表の列に格納されます。

コレクション型ではないリーフ・レベル属性は、オブジェクト型のリーフ・レベル・スカラー属性といわれます。

列にまたがって分割される行オブジェクト

オブジェクト表では、すべてのリーフ・レベル・スカラー属性または REF 属性に対するデータが、個別の列に格納されます。各 VARRAY も、大きすぎない限り、列に格納されます (6-3 ページの「[VARRAY の内部レイアウト](#)」を参照)。

NESTED TABLE 型のリーフ・レベル属性は、オブジェクト表に対応付けられた個別の表に格納されます。これらの表は、オブジェクト表宣言の一部として宣言する必要があります (6-3 ページの「[NESTED TABLE の内部レイアウト](#)」を参照)。

オブジェクト表にあるオブジェクトの属性を取り出したり、変更したりする場合、対応する操作が表の列で実行されます。オブジェクト自体の値にアクセスすると、オブジェクト表の列を引数として使用し、その型のデフォルトのコンストラクタを起動することによって、オブジェクトのコピーが生成されます。

システムによって生成されたオブジェクト識別子は、非表示列に格納されます。Oracle は、オブジェクト識別子を使用して、そのオブジェクトに対する REF を作成します。

列オブジェクトを持つ表の非表示列

表がオブジェクト型の列とともに定義されている場合、オブジェクト型のリーフ・レベル属性に対する表に、非表示列が追加されます。各列オブジェクトには、オブジェクトの NULL 情報 (トップ・レベルのオブジェクトおよびネストしたオブジェクトのアトミック NULL) を格納するための非表示列があります。

REF

行オブジェクトに対して REF が作成される場合、その作成された REF は、オブジェクト識別子 (オブジェクト表のいくつかのメタデータおよび ROWID (オプション)) で構成されます。

REF 型の列にある REF のサイズは、その列に対応付けられた記憶域プロパティに依存します。たとえば、列が REF WITH ROWID として宣言されている場合、ROWID は REF 列に格納されます。ROWID のヒントは、制約付き REF 列にあるオブジェクト参照では無視されます。

列が SCOPED REF として宣言されている場合は、オブジェクト表のメタデータおよび ROWID を省略することによって、その列は小さくなります。SCOPED REF の長さは、16 バイトです。

オブジェクト識別子が主キー・ベースの場合、主キーを導出する列の数に基づいて、主キーの値を格納するための 1 つ以上の内部列が作成されます。

注意： REF 列が、オブジェクト識別子が主キーから導出される行オブジェクトを参照する場合、この REF 列を、主キー・ベース REF または pkREF といいます。pkREF を含む列は、有効範囲付きであるかまたは参照制約が指定されている必要があります。

NESTED TABLE の内部レイアウト

NESTED TABLE の行は、個別の記憶表に格納されます。各 NESTED TABLE の列には、行ごとに 1 つの記憶表ではなく、対応付けられた 1 つの記憶表があります。記憶表は、その列にあるすべての NESTED TABLE に対するすべての要素を保持しています。記憶表には、システムによって生成される値を持つ非表示 NESTED_TABLE_ID 列があります。この値によって、NESTED TABLE の要素が適切な行にマップされます。

記憶表を索引構成表にすることによって、コレクション全体を取り出す問合せを高速化できます。ORGANIZATION INDEX 句を STORE AS 句の中に挿入します。

NESTED TABLE 型は、オブジェクトまたはスカラーを含むことができます。

- 要素がオブジェクトである場合、記憶表はオブジェクト表のようになります。つまり、オブジェクト型の最上位の属性が、記憶表の列になります。ただし、NESTED TABLE の行にはオブジェクト識別子の列がないため、NESTED TABLE のオブジェクトに対する REF を作成することはできません。
- 要素がスカラーである場合、記憶表には、スカラー値を含む COLUMN_VALUE という単一の列が含まれます。

詳細は、5-14 ページの「[NESTED TABLE の記憶域](#)」を参照してください。

VARRAY の内部レイアウト

VARRAY のすべての要素は、単一の列に格納されます。配列のサイズに基づいて、VARRAY はインラインまたは BLOB に格納されます。詳細は、5-13 ページの [VARRAY の記憶域上の考慮点](#) を参照してください。

オブジェクト識別子

オブジェクト表にあるすべての行オブジェクトには、対応付けられた論理オブジェクト識別子 (OID) があります。デフォルトでは、システムによって生成された一意の OID (16 バイト長) が、各行オブジェクトに割り当てられます。Oracle では、オブジェクト識別子のドキュメントまたはオブジェクト識別子の内部構造へのアクセスが提供されていません。この構造はいつでも変更できます。

オブジェクト表の OID 列は非表示列です。オブジェクト表の OID 列が一度設定されると、これを無視できます。かわりに、オブジェクト参照を介してのオブジェクトのフェッチおよびナビゲートに集中できます。

行オブジェクトに対する OID は、オブジェクト表にある行オブジェクトを一意に識別します。オブジェクト表の OID 列に、索引が暗黙のうちに作成され、メンテナンスされます。分散環境およびレプリケート環境では、システムによって生成された一意の識別子によって、オブジェクトが一意に識別されます。

主キー・ベースのオブジェクト識別子 システムによって生成される、グローバルに一意な識別子が必要でない環境では、各オブジェクトが余分に 16 バイト消費し、そのオブジェクト上の索引をメンテナンスすることが効果的でない場合があります。行オブジェクトの主キー値をそのオブジェクト識別子として再使用することで、領域を節約することができます。

また、主キー・ベースの識別子を利用することで、オブジェクト表へのデータのロードを高速かつ容易に行うことができます。それに対して、システムによって生成されるオブジェクト識別子は、それらのオブジェクト識別子への参照が永続的に格納される場合、ユーザーが指定したキーを使用して再度マップする必要があります。

オブジェクトに対する OCI のヒントおよび技法

次の項では、オブジェクトを使用する OCI プログラムで実行される一般的な操作を示し、OCI を効果的に使用するためのヒントおよび技法を紹介します。

オブジェクト・モードでの OCI プログラムの初期化

オブジェクト操作を使用可能にするには、OCI プログラムをオブジェクト・モードで初期化する必要があります。次の OCI コードで、プログラムをオブジェクト・モードで初期化します。

```
err = OCIInitialize(OCI_OBJECT, 0, 0, 0, 0);
```

プログラムがオブジェクト・モードで初期化される場合、オブジェクト・キャッシュが初期化されます。この時点では、キャッシュのメモリーは割り当てられません。キャッシュのメモリーは、必要に応じて割り当てられます。

新しいオブジェクトの作成

OCIObjectNew() 関数によって、一時オブジェクトまたは永続オブジェクトが作成されます。一時オブジェクトの存続期間は、それが作成されたセッションの存続期間です。永続オブジェクトとは、データベースのオブジェクト表に格納されるオブジェクトです。OCIObjectNew() 関数は、キャッシュ内に作成されたオブジェクトに対するポインタを戻すため、アプリケーションは属性値を直接設定して、新しいオブジェクトを初期化する必要があります。オブジェクトは、この時点でまだデータベースに作成されていません。キャッシュからフラッシュされるときに、作成され、データベースに格納されます。

OCIObjectNew() を利用してキャッシュ内にオブジェクトを作成する場合、すべての属性が NULL に設定されます。属性 NULL 標識情報は、パラレル NULL 標識構造体に記録されます。アプリケーションが属性値を設定したにもかかわらず、パラレル NULL 構造体への NULL 標識情報の設定に失敗すると、オブジェクトのフラッシュ時に、オブジェクト属性がデータベース内で NULL に設定されます。

Oracle8i では、オブジェクト作成時にすべての属性を NOT NULL に設定するかわりに、OCIAttrSet() 関数を使用して環境ハンドルの OCI_OBJECT_NEW_NOTNULL 属性を使用できます。この属性が設定されると、NULL 属性を持たないオブジェクトが作成されます。つまり、すべての属性が、Oracle によって提供されるデフォルト値に設定され、パラレル NULL 標識構造体におけるその NULL ステータス情報は、NOT NULL に設定されます。この属性を使用すると、標識構造体を変更する追加ステップが不要になります。Oracle によって提供されるデフォルト値は変更できません。かわりに、オブジェクトの作成直後に、オブジェクトに独自のデフォルト値を代入できます。

OCIObjectNew() を使用して永続オブジェクトが作成されると、コール側は、新しく作成されたオブジェクトを挿入するデータベース表を識別する必要があります。コール側は、表オブジェクトを使用して表を識別します。スキーマ名および表名が指定されると、OCIObjectPinTable() 関数は、表オブジェクトに対するポインタを戻します。OCIObjectPinTable() への各コールは、表オブジェクト情報をフェッチするためのサーバーへのコールになります。必要な表オブジェクトがすでにキャッシュに確保されている場合でも、サーバーへのコールが発生します。アプリケーションが、同一データベース表に挿入するオブジェクトを複数作成している場合は、表オブジェクトを一度確保し、その表オブジェクトへのポインタを将来の使用に備えて保存しておくことをお勧めします。これによって、アプリケーションのパフォーマンスが向上します。

オブジェクトの更新

オブジェクトを更新する前に、そのオブジェクトをキャッシュに確保する必要があります。オブジェクトが確保されると、アプリケーションは必要な属性を直接更新できます。オブジェクトが更新されたことを示すには、OCIObjectMarkUpdate() 関数へのコールを行う必要があります。更新済としてマークされたオブジェクトは、使用済リストに置かれ、キャッシュのフラッシュ時またはトランザクションがコミットされるときに、サーバーにフラッシュされます。

オブジェクトの削除

OCIObjectMarkDelete() 関数または OCIObjectMarkDeleteByRef() 関数をコールすることによって、オブジェクトを削除できます。

オブジェクト・キャッシュ・サイズの制御

次の 2 つの OCI 環境ハンドル属性を使用して、オブジェクト・キャッシュのサイズを制御できます。

- OCI_ATTR_CACHE_MAX_SIZE: 最大キャッシュ・サイズを制御します。
- OCI_ATTR_CACHE_OPT_SIZE: 最適なキャッシュ・サイズを制御します。

OCIAttrGet() または OCIAttrSet() 関数を使用して、これらの OCI 属性を取得または設定できます。メモリーがキャッシュに割り当てられるたびに、最大キャッシュ・サイズに達したかどうかを判断するチェックが行われます。最大キャッシュ・サイズに達した場合、キャッシュは、最も前に使用されたオブジェクトを、確保カウントを 0（ゼロ）で自動的に解放（エージ・アウト）します。キャッシュでのメモリー使用量が最適なサイズになるか、または解放できるオブジェクトがなくなるまで、キャッシュはこのようなオブジェクトの解放を続けます。オブジェクト・キャッシュには、最大キャッシュ・サイズの制限がありません。メモリー割当て要求のサービスによって、キャッシュ・サイズが、指定された最大キャッシュ・サイズを超える可能性があります。前述の 2 つのパラメータを利用することで、キャッシュからのオブジェクトのエージングの頻度をアプリケーションから制御できます。

クライアント・キャッシュへのオブジェクトの取出し（確保）

確保とは、サーバーからクライアント・キャッシュにオブジェクトを取り出し、メモリーにオブジェクトを置き、アプリケーションが操作できるようにオブジェクトにポインタを指定し、オブジェクトに使用中のマークを付ける処理のことです。OCIObjectPin() 関数は、指定された REF を参照解除して、対応するオブジェクトをキャッシュに確保します。確保されたオブジェクトへのポインタは、コール側に戻されます。また、このポインタは、オブジェクトがキャッシュに確保されている限り有効です。オブジェクトが確保解除された後は、そのオブジェクトはエージ・アウトされて、オブジェクト・キャッシュ内に存在しなくなる場合があるため、このポインタを使用しないでください。

OCIObjectPin() および OCIObjectUnpin() コールの例を次に示します。

```
status = OCIObjectPin(envh, errh, empRef, (OCIComplexObject*)0,
                     OCI_PIN_RECENT, OCI_DURATION_TRANSACTION,
                     OCI_LOCK_NONE, (dvoid**)&emp);
/* manipulate emp object */
status = OCIObjectUnpin(envh, errh, emp);
```

OCIObjectPin() の引数である empRef パラメータは、必要な従業員オブジェクトに対する REF を指定します。キャッシュ内の従業員オブジェクトへのポインタは、emp パラメータを介して戻されます。

OCIObjectPinArray() 関数を使用して、オブジェクトの配列を 1 回のコールで確保できます。この関数は、REF の配列を参照解除して、対応するオブジェクトをキャッシュに確保します。オブジェクトがキャッシュ上にキャッシュされていない場合には、1 回のネットワーク・ラウンドトリップでサーバーから取り出されます。したがって、OCIObjectPinArray() をコールしてオブジェクトの配列を確保すると、アプリケーションのパフォーマンスが向上します。また、確保するオブジェクトの配列の型は異なってもかまいません。

取り出すオブジェクトのバージョンの指定

オブジェクトを確保するときに、PIN オプション引数を使用して、オブジェクトの現在のバージョン、最新バージョンまたは任意のバージョンのどれが必要かを指定できます。有効なオプションの詳細は、次のとおりです

- OCI_PIN_RECENT PIN オプション。現行トランザクションでキャッシュにロードされているオブジェクトを戻すように、オブジェクト・キャッシュに指示します。つまり、オブジェクトが現行トランザクション以前にロードされていた場合、オブジェクト・キャッシュは、このオブジェクトをデータベースからの最新バージョンでリフレッシュする必要があります。同一トランザクション内でオブジェクトの確保に成功すると、キャッシュされているコピーが戻され、その結果データベース・アクセスは発生しません。ほとんどの場合は、この PIN オプションを使用してください。
- OCI_PIN_LATEST PIN オプション。常にオブジェクトの最新のコピーを取得するように、オブジェクト・キャッシュに指示します。オブジェクトがすでにキャッシュ内にあり、ロックされていない場合、オブジェクト・コピーは、データベースからの最新のコピーでリフレッシュされます。一方、キャッシュ内のオブジェクトがロックされている場合、これが最新のコピーと想定され、キャッシュされているコピーが戻されます。オブジェクトの最新のコピーを表示する必要があるアプリケーション（株式相場、当座預金残高などを表示するアプリケーションなど）には、このオプションを使用する必要があります。
- OCI_PIN_ANY PIN オプション。最も効率のよい方法でオブジェクトをフェッチするように、オブジェクト・キャッシュに指示します。戻されるオブジェクトのバージョンは問題ではありません。このオプションは、製品情報、部品情報など、頻繁には変更されないオブジェクトの場合に適しています。また、読取り専用のオブジェクトにも適しています。

オブジェクトの確保時間の指定

オブジェクトを確保するとき、オブジェクトがキャッシュ内に確保される有効期限を指定できます。有効期限が過ぎると、オブジェクトは自動的にキャッシュから確保解除されます。オブジェクトの確保有効期限が終了した後は、アプリケーションでオブジェクト・ポインタを使用しないでください。オブジェクトは、OCIObjectUnpin() 関数を明示的にコールすることによって、有効期限切れになる前に確保解除できます。Oracle では、次の 2 つの事前定義済みの確保有効期限がサポートされます。

- セッション確保有効期限 (OCI_DURATION_SESSION) 存続期間は、データベース接続の有効期限です。トランザクションをまたがってキャッシュ内に常に必要なオブジェクトは、セッション有効期限で確保する必要があります。
- トランザクション確保有効期限 (OCI_DURATION_TRANS) 存続期間は、データベース・トランザクションの有効期限です。つまり、トランザクションがロールバックまたはコミットされた時点で有効期限が終了します。

サーバー上でオブジェクトをロックするかどうかの指定

オブジェクトを確保するとき、コール側は、ロック・オプションを介してオブジェクトをロックするかどうかを指定できます。オブジェクトがロックされると、サーバー側のロックが行われ、他のユーザーがオブジェクトを変更できなくなります。トランザクションがコミットまたはロールバックされた時点で、ロックが解除されます。使用可能なロック・オプションは次のとおりです。

- OCI_LOCK_NONE ロック・オプション。ロックしないでオブジェクトを確保するように、キャッシュに指示します。
- OCI_LOCK_X ロック・オプション。ロックを取得した後にのみオブジェクトを確保するように、キャッシュに指示します。オブジェクトが、現在、他のユーザーによってロックされている場合、このオプションを持つ確保コールは、ロックを取得できるまで待機した後、コール側に戻ります。OCI_LOCK_X ロック・オプションを使用することは、SELECT FOR UPDATE 文を実行することと同等です。
- OCI_LOCK_X_NOWAIT ロック・オプション。ロックを取得した後にのみオブジェクトを確保するように、キャッシュに指示します。OCI_LOCK_X オプションと異なり、OCI_LOCK_X_NOWAIT オプションを持つ確保コールは、オブジェクトが現在別のユーザーによってロックされている場合は待機しません。OCI_LOCK_X_NOWAIT ロック・オプションを使用することは、SELECT FOR UPDATE WITH NOWAIT 文を実行することと同等です。

ロック技法の選択方法

オブジェクトが更新される頻度に基づいて、前述の項で説明したどのロック・オプションを使用するかを選択できます。

オブジェクトが頻繁に更新される場合は、悲観的ロック構造を使用できます。この構造は、更新アクセスの競合が頻繁に発生することを想定しています。オブジェクトは、キャッシュにあるオブジェクトが変更される前にロックされ、ロックを所有しているトランザクションがコミットまたはロールバックを実行するまで、他のどのユーザーもそのオブジェクトを変更できないようにします。適切なロック・オプションを選択することによって、オブジェクトは、確保の時点でロックできます。確保の時点でロックされていなかったオブジェクトも、OCIObjectLock() 関数によってロックできます。Oracle8i では、新しいロック関数 OCIObjectLockNoWait() が追加されています。名前が示すように、この関数は、別のユーザーがオブジェクトのロックを保持している場合は、ロックの取得を待機しません。

オブジェクトがあまり頻繁に更新されない場合は、楽観的ロック構造を使用できます。この構造は、更新アクセスの競合がほとんどないことを想定しています。オブジェクトは、ロックを取得せずに、キャッシュ内でフェッチおよび変更されます。オブジェクトがサーバーにフラッシュされる場合にのみ、ロックが取得されます。楽観的ロックでは、悲観的ロックより高度な同時アクセスが可能です。楽観的ロックを効果的に使用するために、Oracle8i のオブジェクト・キャッシュは、オブジェクトがキャッシュにフェッチされてから、他のユーザーによって変更されたかどうかを検出します。

オブジェクト変更検出モードにすると、オブジェクトがキャッシュにフェッチされてから他のユーザーによって変更されていない場合にのみ、そのオブジェクトの変更が持続的に行われます。このモードは、OCIAttrSet() 関数を使用して環境ハンドルの OCI_OBJECT_DETECTCHANGE 属性を設定することによってアクティブになります。

オブジェクト・キャッシュからのオブジェクトのフラッシュ

オブジェクト・キャッシュ内のオブジェクトに行われた変更は、オブジェクト・キャッシュがフラッシュされるまで、データベースに送信されません。OCICacheFlush() 関数は、クライアントとサーバー間の 1 回のネットワーク・ラウンドトリップで、すべての変更をフラッシュします。変更には、適切なオブジェクト表への新しいオブジェクトの挿入、オブジェクト表でのオブジェクトの更新およびオブジェクト表からのオブジェクトの削除が含まれます。OCITransCommit() 関数をコールすることによってアプリケーションがトランザクションをコミットする場合、オブジェクト・キャッシュは、トランザクションをコミットする前に、キャッシュのフラッシュを自動的に実行します。

関連オブジェクトのプリフェッチ（複合オブジェクト検索）

複合オブジェクト検索（COR）によって、複雑に関連付けられた複数のオブジェクトを操作するアプリケーションのパフォーマンスを大幅に向上できます。COR によって、アプリケーションは一連の関連オブジェクトを 1 回のネットワーク・ラウンドトリップでプリフェッチできるため、パフォーマンスが向上します。OCIObjectPin() または OCIObjectPinArray() を使用してルート・オブジェクトを確保する場合、ルートとともに関連オブジェクトをプリフェッチするように指定できます。プリフェッチ・オブジェクトはキャッシュには確保されません。かわりに、LRU リストに入れます。その結果、これらのオブジェクトへの後続の確保コールはキャッシュ・ヒットするため、サーバーへのラウンドトリップが回避されます。

アプリケーションは、次の情報を指定することによって、一連の関連オブジェクトがプリフェッチされるように指定します。

- ルート・オブジェクトへの REF
- プリフェッチされるオブジェクトの内容および境界を指定するための、オブジェクト型と深さ情報の 1 つ以上の組。タイプ情報は、参照解除する必要がある REF 属性およびプリフェッチする必要がある結果オブジェクトを示します。深さは、プリフェッチされるオブジェクトの境界を定義します。深さレベルは、ルート・オブジェクトから関連オブジェクトに到達するまでに横断する必要がある最少の参照数です。

たとえば、次のプロパティを持つ発注書システムを考えてみます。

- 各発注情報オブジェクトには、発注書番号、顧客オブジェクトへの REF および明細項目オブジェクトを指す REF のコレクションが含まれています。

- 各顧客オブジェクトには、顧客の名前や住所など、顧客についての情報が含まれています。
- 各明細項目オブジェクトには、在庫品目への参照および発注の数量が含まれています。
- 各在庫品目オブジェクトには、在庫品目の名前、価格およびその品目についてのその他の情報が含まれています。

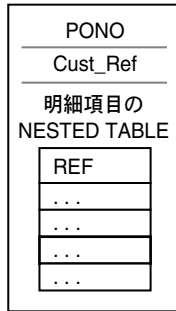
特定の発注書の合計コストを計算する場合を考えてみます。効率を最大にするには、計算に必要なオブジェクトのみをサーバーからクライアント・キャッシュにフェッチし、サーバーへのコール数をできるだけ少なくして、これらのオブジェクトをフェッチします。

COR を使用しない場合、アプリケーションは、必要なオブジェクトをすべて取り出すためにサーバー・コールを数回行う必要があります。ただし、COR を使用すると、取り出すオブジェクトを指定して、その他の不要なオブジェクトを除外できます。発注書の合計コストを計算するには、発注情報オブジェクト、関連明細項目オブジェクトおよび関連在庫品目オブジェクトが必要ですが、顧客オブジェクトは必要ありません。

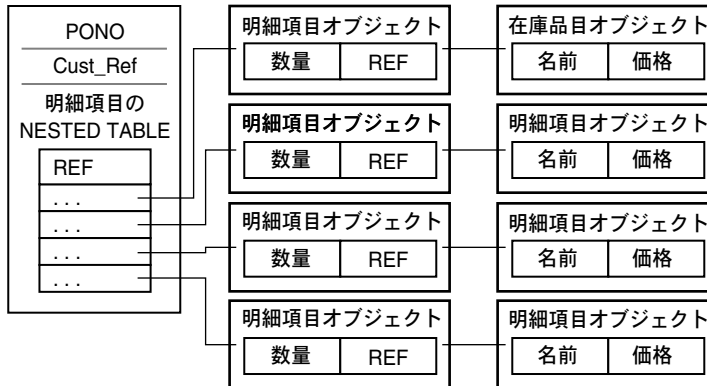
したがって、[図 6-1](#) に示すように、COR を使用すると、最も効率のよい方法で計算に必要な情報を取り出すことができます。COR を使用しないで発注情報オブジェクトを確保する場合、そのオブジェクトのみが取り出されます。COR を使用して発注情報オブジェクトを確保すると、発注情報オブジェクト、関連明細項目オブジェクトおよび在庫品目オブジェクトが取り出されます。ただし、関連顧客オブジェクトは計算には不要なため、取り出されません。

図 6-1 COR を使用した場合と COR を使用しない場合のオブジェクトの取出しの違い

COR を使用しないで発注情報オブジェクトを確保する



COR を使用して発注情報オブジェクトを確保する



OCI および Oracle オブジェクトのデモ

Oracle オブジェクトで OCI を使用方法のデモは、`$ORACLE_HOME/rdbms/demo` にある `cdemocor1.c` ファイルを参照してください。

オブジェクト・ビューが提供するオブジェクトでの OCI オブジェクト・キャッシュの使用

オブジェクト表の場合と同様の方法で、オブジェクト・ビューから合成されたオブジェクトを OCI オブジェクト・キャッシュ上で確保およびナビゲートできます。また、そのキャッシュから、その新しいビュー・オブジェクトを作成、更新、削除およびフラッシュすること

もできます。フラッシュによって、ビューに適切な DML（新しく作成されたオブジェクトの挿入、属性の変更に対する更新など）が実行されます。これによって、ビュー上のすべての INSTEAD-OF トリガーが起動され、オブジェクトが持続的に格納されます。

オブジェクト・キャッシュに新しく作成されたインスタンスへの参照の取得に関して、2つのアプローチの間には少し違いがあります。

主キー・ベースの参照を持つオブジェクト・ビューの場合は、OCIObjectGetObjectRef コールがオブジェクト上でコールされ、オブジェクト参照が取得される前に、オブジェクトに対する識別子を形成する属性が初期化される必要があります。たとえば、発注情報オブジェクトの OCI オブジェクト・キャッシュに新しいオブジェクトを作成するには、次のステップを行う必要があります。

```
.. /* Initialize all the settings including creating a connection, getting a
   environment handle etc. We do not check for error conditions to make
   the example easier to read. */
OCIType *purchaseOrder_tdo = (OCIType *) 0; /* This is the type object for the
   purchase order */
dvoid * purchaseOrder_viewobj = (dvoid *) 0; /* This is the view object */

/* The purchaseOrder struct is a structure that is defined to have the same
   attributes as that of PurchaseOrder_objtyp type. This can be created by the user or
   generated automatically using the OTT generator. */
purchaseOrder_struct *purchaseOrder_obj;

/* This is the null structure corresponding to the purchase order object's
   attributes */
purchaseOrder_nullstruct *purchaseOrder_nullobj;

/* This is the variable containing the purchase order number that we need to create
   */
int PONo = 1003;

/* This is the reference to the purchase order object */
OCIRef *purchaseOrder_ref = (OCIRef *)0;

/* Pin the object type first */
OCITypeByName( envhp, errhp, svchp,
               (CONST text *) "", (ub4) strlen( "" ) ,
               (CONST text *) "PURCHASEORDER_OBJTYP" ,
               (ub4) strlen("PURCHASEORDER_OBJTYP"),
               (CONST char *) 0, (ub4)0,
               OCI_DURATION_SESSION, OCI_TYPEGET_ALL,
               &purchaseOrder_tdo);

/* Pin the table object - in this case it is the purchase order view */
OCIObjectPinObjectTable(envhp, errhp, svchp, (CONST text *) "",
```

```
(ub4) strlen( "" ),
(CONST text *) "PURCHASEORDER_OBJVIEW",
(ub4 ) strlen("PURCHASEORDER_OBJVIEW"),
(CONST OCIRef *) NULL,
OCI_DURATION_SESSION,
&purchaseOrder_viewobj);

/* Now create a new object in the cache. This is a purchase order object */
OCIObjectNew(envhp, errhp, svchp, OCI_TYPECODE_OBJECT, purchaseOrder_tdo,
             purchaseOrder_viewobj, OCI_DURATION_DEFAULT, FALSE,
             (dvoid **) &purchaseOrder_obj);

/* Now we can initialize this object, and use it as a regular object. But before
getting the reference to this object we need to initialize the PONO attribute of the
object which makes up its object identifier in the view */

/* Initialize the null identifiers */
OCIObjectGetInd( envhp, errhp, purchaseOrder_obj, purchaseOrder_nullobj);

purchaseOrder_nullobj->purchaseOrder = OCI_IND_NOTNULL;
purchaseOrder_nullobj->PONO = OCI_IND_NOTNULL;

/* This sets the PONO attribute */
OCINumberFromInt( errhp, (CONST dvoid *) &PONO, sizeof(PONO), OCI_NUMBER_SIGNED,
                  &( purchaseOrder_obj->PONO));

/* Create an object reference */
OCIObjectNew( envhp, errhp, svchp, OCI_TYPECODE_REF, (OCITYPE *) 0,
             (dvoid *) 0, (dvoid *) 0, OCI_DURATION_DEFAULT, TRUE,
             (dvoid **) &purchaseOrder_ref);

/* Now get the reference to the newly created object */
OCIObjectGetObjectRef(envhp, errhp, (dvoid *) purchaseOrder_obj, purchaseOrder_ref);

/* This reference may be used in the rest of the program .... */
...
/* We can flush the changes to the disk and the newly instantiated purchase order
object in the object cache will become permanent. In the case of the purchase order
object, the insert will fire the INSTEAD-OF trigger defined over the purchase order
view to do the actual processing */

OCICacheFlush( envhp, errhp, svchp, (dvoid *) 0, 0, (OCIRef **) 0);
...
```

Oracle オブジェクトを持つ表のパーティション化

パーティション化は、大規模な表や索引を、パーティションと呼ばれる小さく管理しやすい単位に分解することで効率良くサポート可能にする機能です。Oracle8i では、オブジェクト、REF、VARRAY および NESTED TABLE を含む表をパーティション化できるように、パーティション化の機能を拡張しています。LOB に格納された VARRAY は、LOB と同様の方法で同一レベル・パーティション化されます。

次の例では、列オブジェクト ShipToAddr の属性である zip コード (ToZip) に従って、発注書の表をパーティション化します。この例の目的上、パーティション化された VARRAY の記憶域を示すために、NESTED TABLE の属性 LineItemList は VARRAY にされています。

制限事項： NESTED TABLE を含む表をパーティション化することはできませんが、NESTED TABLE に対応付けられた記憶表はパーティション化されません。

LineItemList は VARRAY として定義されていると仮定します。

```
CREATE TYPE LineItemList_vartyp as varray(10000) of LineItem_objtyp;
```

```
CREATE TYPE PurchaseOrder_typ AS OBJECT (  
    PONo          NUMBER,  
    Cust_ref      REF Customer_objtyp,  
    OrderDate     DATE,  
    ShipDate      DATE,  
    OrderForm     BLOB,  
    LineItemList  LineItemList_vartyp,  
    ShipToAddr    Address_objtyp,
```

```
    MAP MEMBER FUNCTION  
        ret_value RETURN NUMBER,
```

```
    MEMBER FUNCTION  
        total_value RETURN NUMBER
```

```
);
```

```
CREATE TABLE PurchaseOrders_tab of PurchaseOrder_typ  
    LOB (OrderForm) store as (nocache logging)  
    PARTITION BY RANGE (ShipToAddr.zip)  
        (PARTITION PurOrderZone1_part  
            VALUES LESS THAN ('59999')  
            LOB (OrderForm) store as (
```

```

        storage (INITIAL 10 MINEXTENTS 10 MAXEXTENTS 100))
    VARRAY LineItemList store as LOB (
        storage (INITIAL 10 MINEXTENTS 10 MAXEXTENTS 100)),
    PARTITION PurOrderZone6_part
        VALUES LESS THAN ('79999')
        LOB (OrderForm) store as (
            storage (INITIAL 10 MINEXTENTS 10 MAXEXTENTS 100))
    VARRAY LineItemList store as LOB (
        storage (INITIAL 10 MINEXTENTS 10 MAXEXTENTS 100)),
    PARTITION PurOrderZone0_part
        VALUES LESS THAN ('99999')
        LOB (OrderForm) store as (
            storage (INITIAL 10 MINEXTENTS 10 MAXEXTENTS 100))
    VARRAY LineItemList store as LOB (
        storage (INITIAL 10 MINEXTENTS 10 MAXEXTENTS 100)));

```

オブジェクト・ビューでのパラレル問合せ

パラレル問合せは、ビューから合成されたオブジェクトでサポートされます。

(ORDER BY、GROUP BY および SET の操作を使用して) 結合およびソートがパラレルに含まれる問合せを実行するには、MAP 関数が必要です。MAP 関数がない場合、問合せは自動的にシリアルになります。

NESTED TABLE の列でのパラレル問合せは、サポートされません。ビューに対してパラレル・ヒントまたはパラレル属性が存在しても、問合せに NESTED TABLE の列が含まれる場合は、シリアルになります。

パラレル DML は、INSTEAD-OF トリガーを持つビューではサポートされません。ただし、トリガー内の個々の文は、パラレル化できます。

ロケータで NESTED TABLE のパフォーマンスを向上させる方法

Oracle8i では、コレクション型の値は、C++ や Java などの言語でのシステム固有の型または構造へ直接マップしません。これらの言語を使用しているアプリケーションは、OCI などの Oracle インタフェースを介して、コレクションの内容にアクセスする必要があります。

一般に、クライアントが (オブジェクトをフェッチすることによって) NESTED TABLE に明示的または暗黙的にアクセスすると、コレクション値全体がクライアント・プロセスに戻されます。パフォーマンス上の理由から、クライアントは、コレクションの内容全体を取り出すのを遅延または回避する場合があります。Oracle では、実際の NESTED TABLE 値ではなくロケータを使用することによって、これに対処します。コレクションの内容に実際にアクセスする場合は、これらの内容はクライアントに自動的に転送されます。

NESTED TABLE のロケータは、コレクション値へのハンドルに似ています。このロケータは、検索実行時のデータベース・スナップショットを確保することで、NESTED TABLE の値またはコピー・セマンティクスを保ちます。スナップショットによって、コレクション要素がロケータを使用してフェッチされたときに、データベースが NESTED TABLE 値の正しいインスタンスを取り出せるようになります。ロケータの有効範囲は 1 つのセッションに限られ、複数のセッションにまたがって使用することはできません。データベース・スナップショットが使用されているため、NESTED TABLE 上の更新率が高い場合、「snapshot too old」というエラーが発生する場合があります。LOB ロケータとは異なり、NESTED TABLE のロケータは純粋なロケータであり、コレクション・インスタンスを変更するためには使用できません。

Oracle オブジェクトを使用したプログラミングに関する FAQ

この章では、Oracle のオブジェクト・リレーショナル機能に関して、ユーザーからよくある質問およびその答えについて説明します。内容は次のとおりです。

- [Oracle オブジェクトに関する一般的な質問](#)
- [オブジェクト型](#)
- [オブジェクト・メソッド](#)
- [オブジェクト参照](#)
- [コレクション](#)
- [オブジェクト・ビュー](#)
- [オブジェクト・キャッシュ](#)
- [ラージ・オブジェクト \(LOB\)](#)
- [ユーザー定義オペレータ](#)

この章は、入門情報およびこのマニュアルの他の章を読んだ後にまだ疑問が残っている場合の参照用の情報を記載しています。

Oracle オブジェクトに関する一般的な質問

オブジェクト・リレーショナル機能は、別のオプションですか？

現在では、そうではありません。Oracle8i では、オブジェクト・リレーショナル機能は、サーバー製品の基本機能の一部です。

Oracle8i のオブジェクト・リレーショナルおよび拡張テクノロジーの設計目標は何ですか？

Oracle8i のオブジェクトおよび拡張テクノロジーの設計目標は、次のとおりです。

- 型システムを拡張してユーザー定義型をサポートすることによって、ユーザーがビジネス・オブジェクトをデータベース内にモデル化できるようにします。これらの型はアプリケーション・オブジェクトを綿密にモデル化し、データベース・サーバーにおいて、数値や文字などの組み込み型と同様に扱えるようにするためのものです。
- Oracle データベース内に格納されたデータへのオブジェクト・ベースのアクセスを容易にするための構造基盤を提供します。また、この構造基盤によって、アプリケーションで 사용되는データ・モデルとデータベースがサポートするデータ・モデルとの不一致を最小にします。
- マルチメディア、財務および空間的アプリケーションに必要な新しいデータ型に対する組み込みサポートを提供します。
- データベースの拡張性のフレームワークを提供し、新しいマルチメディア・データ型および複合データ型をサポートし、データベース内でシステム固有の管理ができるようにします。このフレームワークでは、データ・カートリッジを介してサード・パーティが行う、データ・サーバーの拡張に必要な構造基盤を提供します。

このマニュアルでは、オブジェクト・リレーショナル・テクノロジーについて説明しています。拡張性の詳細は、『Oracle8i データ・カートリッジ開発者ガイド』を参照してください。

Oracle8i のオブジェクト・リレーショナル・テクノロジーの主な機能は何ですか？

Oracle8i オブジェクトとは、Oracle データ・サーバーの標準機能の上に構成された、ユーザー定義オブジェクト型の作成および操作を可能にする追加の機能の集合です。

オブジェクト型システム

Oracle8i では、新しいメタモデルを介して、データベース内で新しい型を定義する性能が提供されています。これを、Oracle8i Type System といいます。この型システムによって、Oracle リレーショナル型が拡張され、ユーザーはオブジェクトをモデル化する型およびメソッドを定義できます。Oracle8i では、ユーザー定義型のメタデータは、SQL、PL/SQL、

Java および他の公開されたインタフェースで利用できるスキーマ内に格納されます。ユーザーは、どのような組込みデータベース型または既知のオブジェクト型、オブジェクト参照およびコレクション型（あるいはその両方）を使用しても、新しいオブジェクト型を作成できます。

オブジェクト・ビュー

オブジェクト型をシステム固有にサーバーに格納することに加えて、Oracle8i では、オブジェクト・ビュー・メカニズムを介して、既存のリレーショナル・データ上にオブジェクトを作成できます。オブジェクト・ビューに属するオブジェクトは、SQL または他のコール・インタフェースを介して行オブジェクトにアクセスするのと同様の方法でアクセスできます。そのようなデータ・アクセスをサポートするため、Oracle8i サーバーでは、リレーショナル・スキーマおよび表に格納されたデータからユーザー定義型のオブジェクトをインスタンス化することができます。このビュー・メカニズムを利用することで、既存のデータベース・スキーマを変更することなく、新しいオブジェクト指向のアプリケーションを開発できます。

SQL のオブジェクトの拡張

オブジェクトの新しい機能をサポートするために、SQL（新しい DDL を含む）には、オブジェクト型を作成、変更または削除し、オブジェクトを表へ格納し、あるいはオブジェクト・ビューを作成または削除するための拡張機能が追加されています。オブジェクト型、参照およびコレクションをサポートするために、DML および問合せが拡張されています。

PL/SQL のオブジェクト拡張

PL/SQL は、SQL と密接に統合した Oracle のデータベース・プログラミング言語です。Oracle8i で導入されているユーザー定義型および他の SQL 型に加えて、ユーザー定義型をシームレスに操作するために、PL/SQL が拡張されています。したがって、アプリケーション開発者は、PL/SQL を使用して、データベース・サーバー内で実行するユーザー定義型に対して論理および操作を実装できます。

Oracle8i オブジェクトに対する Java のサポート

Oracle8i の Java VM は、RDBMS と密接に統合されており、JDBC（動的 SQL）および SQLJ（静的 SQL）へのオブジェクト拡張を介しての Oracle8i オブジェクトへのアクセスをサポートしています。したがって、アプリケーション開発者は、Java を使用して、データベース・サーバー内で実行するユーザー定義型に対して論理および操作を実装できます。

外部プロシージャ

オブジェクト型のデータベース・ファンクション、プロシージャまたはメンバー・メソッドは、PL/SQL、Java または外部プロシージャとして C で実装できます。外部プロシージャは、マシン精度の計算においてより効率的な、C などの低レベル言語でより高速で簡単にできる作業に最も適しています。外部プロシージャは、常に、RDBMS サーバーのアドレス空間外で、セーフ・モードで実行されます。

オブジェクト・タイプ・トランスレータ

オブジェクトで利用できるオブジェクト・タイプ・トランスレータ (OTT) は、Oracle データ・ディクショナリからのスキーマ情報を使用して、C の構造体と標識を含むヘッダー・ファイルを生成することによって、オブジェクト型スキーマへのクライアント側のマッピングを提供します。生成されたこれらのヘッダー・ファイルは、ホスト言語アプリケーションで、データベース・オブジェクトへ透過的にアクセスするために使用されます。

クライアント側キャッシュ

Oracle8i では、データベースに格納された永続オブジェクトに、効率的にアクセスするためのオブジェクト・キャッシュが提供されています。オブジェクトのコピーは、オブジェクト・キャッシュに置かれます。一度データがクライアントにキャッシュされると、アプリケーションは、これらのデータに対してメモリー・スピードでアクセスできます。キャッシュ内で行われたオブジェクトの変更は、Oracle Call Interface のオブジェクト拡張を使用して、データベースに反映できます。

複合オブジェクトの取出し

Oracle8i では、効率的な複合オブジェクトの取出しもサポートされています。つまり、サーバーからオブジェクトをフェッチする 1 回の要求で、フェッチしたオブジェクトにオブジェクト参照を利用して関連付けられている他のオブジェクトも、クライアントとサーバーの間の 1 回のラウンドトリップで取り出すことができます。このような機能によって、一連の関連オブジェクトを 1 つの単位としてフェッチおよび処理することが容易になります。

OCI のオブジェクト拡張

Oracle8i の Oracle Call Interface では、Oracle8i のオブジェクト機能を使用する必要があるアプリケーション開発者およびツール開発者に、わかりやすいアプリケーション・プログラミング・インタフェースが提供されています。Oracle Call Interface では、Oracle8i サーバーに接続して、サーバー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機能を持った、ランタイム環境を提供しています。これによって、アプリケーション開発者は、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およびそれらの属性に対してオブジェクト参照を利用して関連付けられたオブジェクトをナビゲーションに、または SQL DML を介してデータの特徴を指定して結合的に、アクセスおよび操作することができます。また、Oracle Call Interface では、サーバー内で定義されたオブジェクト型に関して、実行時にメタデータ情報にアクセスするための多数の機能が提供されています。それらの一連の機能によって、データベースに格納されたオブジェクト・メタデータおよび実オブジェクト・データへの動的アクセスが容易になります。

PRO*C/C++ のオブジェクト拡張

Oracle8i Pro*C プリコンパイラでは、埋込み SQL プログラム・インタフェースが提供されており、Oracle Call Interface より高レベルの抽象化を実行できます。Oracle Call Interface と同様に、Pro*C プリコンパイラでも、アプリケーション開発者は、Oracle8i のクライアント側のオブジェクト・キャッシュおよびオブジェクト・タイプ・トランスレータ・ユーティリ

ティを使用できます。Pro*C では、Oracle8i オブジェクト型に対する C バインド変数の使用がサポートされています。さらに、新しく単純化された構文が提供されており、これを使用して SQL 型オブジェクトの割当ておよび解放を行い、SQL DML またはナビゲーションなインタフェースを介してそれらにアクセスできます。したがって、アプリケーション開発者には、サーバーのスキーマに対する（クライアント側の）バインド変数のコンパイル時のタイプ・チェック、プログラムのバインド変数に対する Oracle8i サーバーのオブジェクト・データの自動マッピングおよびクライアント・プロセスでのデータベース・オブジェクトの簡単な管理方法および処理方法を含む、多くの利点があります。

0040 のオブジェクト拡張

Oracle8i Oracle Objects For OLE (OO4O) は、一連の COM オートメーション・インタフェース / オブジェクトで、Oracle8i データベース・サーバーへの接続、問合せの実行および結果の管理を行います。OO4O のオートメーション・インタフェースでは、Oracle8i に固有の機能に簡単に効率的にアクセスできます。また、Microsoft COM オートメーション・テクノロジーをサポートするほとんどのプログラミング言語またはスクリプティング言語から使用できます。これらの言語としては、Visual Basic、Visual C++、Excel の VBA、VBScript および IIS Active Server Pages の JavaScript があります。

リレーショナル機能との統合

Oracle8i オブジェクトでは、問合せ (SELECT...FROM...WHERE)、高速コミット、バックアップおよびリカバリ、スケーラブルな接続性、行レベルのロック、読取り一貫性、パーティション表、パラレル問合せ、Parallel Server、エクスポート / インポート、ロードなど、標準のリレーショナル・データベース機能が引き続きサポートされています。

Oracle8i の新しいオブジェクト・リレーショナル機能にはどのようなものがありますか？

Oracle8i では、複合オブジェクトをモデル化するための基盤が提供されています。次にオブジェクト・リレーショナル機能を、Oracle 8i の新機能とともに示します。

型システム

スカラ、LOB、オブジェクト、参照、コレクション

実行環境

PL/SQL メソッド、外部プロシージャ、Java メソッド

問合せプロセス

トリガー、制約、オブジェクト・ビュー、ユーザー定義オペレータ

データ索引

ソート、ハッシュ、ビットマップ、索引構成表、拡張索引作成機能

問合せ最適化

オブジェクト問合せ最適化、拡張可能オプティマイザ

操作の完全性

エクスポート / インポート、ローダー、パラレル問合せ、パーティション化のオブジェクト・サポート

言語インタフェース

OCI、C++ (ODD)、Pro*C、JDBC、OO4O でのオブジェクト・サポート

オブジェクト型

構造化データとは何ですか？

SQL 92 標準では、ほとんどのデータベース・プログラミングで使用される 19 のアトミック・データ型が定義されています。これらの種類のデータは、単純構造化データといわれます。

Oracle オブジェクトでは、REF およびコレクションの概念が導入されています。これらの種類のデータは、複合構造化データといわれます。

LOB には、情報を格納するための別の方法があります。それらは、非構造化データといわれます。

ユーザー定義型、ユーザー定義ファンクションおよび抽象データ型はどこにありますか？

Oracle において、ユーザー定義型や抽象データ型は、オブジェクト型に相当します。

また、Oracle において、ユーザー定義ファンクションは、オブジェクト型メソッドに相当します。

このような用語を使用するのは、これらの意味が業界共通の使用法とは異なるためです。たとえば、Oracle オブジェクトには NULL を指定できますが、抽象データ型には NULL を指定できません。

オブジェクト型とは何ですか？

Oracle8i では、オブジェクト型といわれるユーザー定義データ型の形式がサポートされています。

オブジェクト型は、実社会のエンティティを抽象化したものです。オブジェクト型は、次のコンポーネントを持つスキーマ・オブジェクトです。

- スキーマ内でオブジェクト型を一意に識別する名前
- 実社会のエンティティの構造および状態をモデル化する属性
- 実社会のエンティティの動作を実装するメソッド

オブジェクト型はなぜ便利なのですか？

オブジェクト型は、C++ および Java でサポートされているクラス・メカニズムに似ています。オブジェクトを再利用できるため、より速く、効率的にデータベース・アプリケーションを開発できます。オブジェクトのサポートによって、複雑な実社会のビジネス・エンティティおよび論理のモデル化が簡単になります。データベース内でオブジェクト型をシステム固有にサポートすることによって、Oracle8i では、オブジェクト指向プログラミング言語とデータベースとの間のデータ型式の不一致が排除されます。そのため、アプリケーション開発者は、クライアント側オブジェクトとデータベース・オブジェクトとの間のマッピング・レイヤーを記述する必要がなくなります。オブジェクトの抽象化およびカプセル化によって、アプリケーションの理解およびメンテナンスが容易になります。これは、エンタープライズ・データベース・アプリケーション開発にとっては重要な考慮点です。

Oracle8i では、オブジェクト・データはどのように格納および管理されますか？

オブジェクトは、データ・サーバーによってシステム固有に管理されます。オブジェクト型は、列の型（列オブジェクト）、またはオブジェクト表内の各行の型（行オブジェクト）として使用できます。列オブジェクトとして使用された場合、オブジェクト型はリレーショナル表のフィールドとして動作します。各行オブジェクトには、オブジェクト識別子（OID）といわれる一意の識別情報が付きます。

オブジェクトは、データベース・コンポーネントと完全に統合されます。オブジェクトには、索引付けおよびパーティション化を行うことができます。たとえば、オブジェクトを含む問合せはパラレル化でき、統計情報を使用したコストベースのオプティマイザによって最適化されます。

定評ある Oracle8i データ・サーバーを基盤として作成されているため、オブジェクトは、リレーショナル・データと同様の信頼性、可用性およびスケーラビリティで管理されます。

Oracle8i では、継承はサポートされていますか？

直接はサポートされていません。継承には、単一継承と複数継承の違いなど、各言語に個別の意味があるため、Oracle では各言語の継承動作をまねた方法が使用されます。

Oracle8i では、C++ および Java マッピングを介してのクライアント側の継承がサポートされています。C++ の場合、Oracle Designer に付属の Object Database Designer を使用して、統一モデリング言語（UML）の図に基づいた DDL および C++ コードを生成します。Java の場合、Oracle JDBC ドライバの CustomDatum 機能を使用します。

サーバー側メソッド継承は、Oracle8i Java VM によって Java に提供されます。Oracle8i では、現在、SQL において継承や型およびそのサブタイプを格納および問い合わせる機能はサポートされていません。

オブジェクト・メソッド

オブジェクト・メソッドはどの言語を使用して記述できますか？

メソッドは、PL/SQL、Java、C または C++ で実装できます。Oracle8i では、C および C++ は、外部プロシージャ機能を介してサポートされます。一方、PL/SQL メソッドおよび Java メソッドは、サーバーのアドレス空間内で実行されます。オブジェクト型が様々なプログラミング言語で実装可能であっても、SQL のメソッド仕様部をその実装から分離すると、オブジェクト型にメソッドを起動する方法が1つになります。Oracle8i では、PL/SQL メソッド、Java メソッドおよび外部 C プロシージャをサーバーから起動するための、安全で保護された環境が提供されています。ユーザー・メソッドにプログラミング・エラーがあっても、サーバーがクラッシュしたりデータベースが破損したりしません。そのため、ミッション・クリティカルな環境におけるサーバーの信頼性および可用性が保証されます。

オブジェクト・メソッドに PL/SQL と Java のどちらを使用するかは、どのように判断しますか？

Oracle8i では、PL/SQL および Java はサーバー・プログラミング言語として相互変換して使用できます。PL/SQL は、SQL のシームレスな拡張で、データベース内で SQL 集約操作を行うために適した言語です。Java は、データベース・サーバーを含む、複数層で実行する移植可能なアプリケーションを記述するために適した発展段階の言語です。

外部プロシージャはいつ使用する必要がありますか？

外部プロシージャは、通常、C などの低レベル言語で記述されるのが最適な計算集中型操作に使用されます。また、データ・サーバー内で実行するために Java または PL/SQL で簡単に書き直すことができない既存のライブラリにあるルーチンを起動することにも役立ちます。

外部プロシージャをコールする IPC（プロセス間通信）オーバーヘッドは、PL/SQL プロシージャまたは Java プロシージャをコールする IPC オーバーヘッドより大きくなります。ただし、外部プロシージャで行われる計算が、複雑で何万もの指示からなる場合、外部プロシージャをコールするオーバーヘッドが重要でなくなります。

定義者権限および実行者権限とは何ですか？

定義者権限と実行者権限の区別は、オブジェクト以外にも適用されます。オブジェクト指向のプログラムには、通常、再使用可能なモジュールが含まれるため、実行者権限は特に役立つ場合があります。

オブジェクト・メソッドは、メソッド定義に基づいて、所有者の権限（定義者権限）またはカレント・ユーザーの権限（実行者権限）で実行されます。実行者権限は、再使用可能なオブジェクトを記述する場合に役立ちます。これは、これらのオブジェクトのユーザーは、オブジェクトの作成者に対して表へのアクセス権限を付与する必要がないためです。定義者権限は、オブジェクトの実装の一部として、オブジェクト・メソッドがオブジェクトの作成者によってメンテナンスされるメタデータにアクセスする必要がある場合に役立ちます。メタデータにアクセスするメソッドは、オブジェクトの作成者が所有のメタデータをユーザーに公開する必要がないように、定義者権限を使用して実行されます。

オブジェクト参照

オブジェクト参照とは何ですか？

オブジェクト参照（REF）は、オブジェクト表に格納された行オブジェクト、またはオブジェクト・ビューから作成されたオブジェクトを、一意に識別します。通常、REF 値は、オブジェクトの一意の識別子、オブジェクト表に対応付けられた一意の識別子、およびそのオブジェクトが格納されているオブジェクト表の行の ROWID から導出されます。オブションの ROWID は、オブジェクトへの高速アクセスのヒントとして使用されます。

オブジェクト参照はいつ使用する必要がありますか？外部キーとの違いは何ですか？

外部キーと同様に、オブジェクト参照は、複雑な関連をモデル化するのに役立ちます。複雑な関連をモデル化する場合、次の理由で、オブジェクト参照の方が外部キーより柔軟性があります。

- オブジェクト参照は、強力に型指定されており、コンパイル時のタイプ・チェックが向上します。
- オブジェクト参照のコレクションを使用して、1 対多関連をモデル化できます。
- アプリケーションにおいて、SQL 文を作成しなくても、オブジェクト参照を使用して簡単にオブジェクトをナビゲートし取り出すことができます。

- SQL の REF ナビゲーションによって、複雑な結合を行う必要がなくなります。
- オブジェクト参照によって、アプリケーションは、サーバーへの 1 回の要求で、REF によって関連付けられたオブジェクトを取り出せます。

主キーに基づいてオブジェクト参照を作成できますか？

できます。外部キーに基づいてオブジェクト参照を作成し、次のオブジェクトを参照できます。

- オブジェクト・ビュー: オブジェクト・ビューを使用してリレーショナル表からオブジェクトを作成する場合、作成されたオブジェクトの OID は、通常、基礎となるリレーショナル表上の主キーに基づきます。
- 主キー・ベースの OID があるオブジェクト表: オブジェクト表を定義する場合、Oracle8i では、システムによって生成される OID ではなく、行オブジェクトの OID として主キーを指定するオプションが提供されています。

SCOPED REF とは何ですか？また、いつ使用する必要がありますか？

一般的に、列には、オブジェクトが格納されているオブジェクト表に関係なく、特定の宣言された型へのオブジェクト参照が含まれる場合があります。ただし、REF 型の列は、指定したオブジェクト表からのオブジェクトへの参照のみを含むよう有効範囲（制約）を指定できます。SCOPED REF の表識別子はシステムに格納されないため、SCOPED REF は標準の REF よりディスクの占有サイズが小さくて済みます。そのため、できるだけ SCOPED REF を使用してください。また、SCOPED REF のナビゲーションを含む問合せは、適切な場合、結合に最適化されます。

PL/SQL および Java でオブジェクト参照を使用してオブジェクトを操作できますか？

できます。PL/SQL および Java の両方ともオブジェクト参照をサポートしています。PL/SQL では、UTL_REF パッケージにオブジェクト参照を与えることで、オブジェクトの取出しおよび更新ができます。Java では、get メソッドおよび set メソッドを使用してオブジェクト参照を参照クラスにマップし、オブジェクトの取出しおよび更新ができます。

コレクション

Oracle8i ではどのような種類のコレクションがサポートされていますか？

Oracle8i では、可変配列（VARRAY）および NESTED TABLE という 2 種類のコレクションがサポートされています。表のオブジェクト型および列の属性は、コレクション型に指定できます。可変配列（VARRAY）および NESTED TABLE を使用することによって、アプリ

ケーションは、1 対多関連および多対多関連をデータベース・スキーマ内でシステム固有にモデル化できます。

コレクションのモデル化に VARRAY と NESTED TABLE のどちらを使用するかは、どのように判断すればよいですか？

コレクション要素の順序を保持する必要がある場合には、VARRAY が役立ちます。VARRAY は、常にコレクション全体を 1 つの単位として操作し、コレクションの個々の要素に問合せを要求しない場合に、非常に効率的です。VARRAY のサイズが小さい場合は、含んでいる行とともにインラインで格納されます。また、VARRAY のサイズが一定のしきい値より大きい場合は、自動的に LOB として格納されます。

コレクション要素間に順序がなく、個々の要素へ効率的に問合せを行うことが重要である場合には、NESTED TABLE が役立ちます。コレクション型列の要素は、リレーション・スキーマ内の親子表に似た、個別の表に格納されます。

Oracle8i オブジェクトでは、コレクション内のコレクションがサポートされていますか？

コレクションには、別のコレクション属性を含めることはできません。ただし、ネストしたコレクションは、コレクション属性を持つオブジェクトへの参照を使用してモデル化できます。したがって、アプリケーションは間接的に REF を利用することでネストしたコレクションをモデル化できます。

コレクション・ロケータとは何ですか？

コレクション・ロケータによって、アプリケーションは、メモリー内にコレクションを具体化することなく、大きなコレクションを取り扱うことができます。これによって、大きいコレクションをインタフェース間で効率的に転送できます。コレクションは、アプリケーションが最初にその要素にアクセスしたときに透過的に具体化されます。また、アプリケーションは、ロケータを使用してコレクションのサブセットに対する問合せおよび取出しができます。

コレクション・ロケータを扱うためには、CREATE および ALTER TABLE DDL で指定します。コレクションへのアクセスはカプセル化されているため、アプリケーションは、値として戻されるように指定された NESTED TABLE の場合と同じインタフェースを使用して、ロケータとして戻されるように指定された NESTED TABLE を取り出します。

コレクション・アンネスティングとは何ですか？

コレクション・アンネスティングによって、アプリケーションは、指定した行の一連のコレクションに対して効率的に問合せができます。この問合せは、リレーショナル・スキーマ内における指定した親行の子行に対する問合せに似ています。コレクション・アンネスティン

グを利用することで、アプリケーションは、コレクション・フォーム内またはフラット親子フォーム内の 1 対多関連を柔軟に表示できます。

オブジェクト・ビュー

オブジェクト・ビューとオブジェクト表の違いは何ですか？

リレーショナル・ビューとリレーショナル表に類似点があるように、オブジェクト・ビューには、次のようなオブジェクト表に似たプロパティがあります。

- オブジェクトを行に含みます。ビューの列は、オブジェクト型の最上位の属性にマップします。
- 各オブジェクトには、対応付けられた識別子があります。識別子は、ビュー定義者によって指定されます。ほとんどの場合、ベースとなる表の主キーが識別子として動作します。

オブジェクト・ビューは更新可能ですか？

すべての属性が表の実列にマップされている場合、オブジェクト・ビューは簡単に更新できます。CAST-MULTISET などのより複雑な技法を使用して属性を導出するビューには、INSTEAD-OF トリガーを使用して更新できます。そのようなビューを更新すると（あるいは挿入または削除すると）、システムは、ベース表を暗黙的に変更するのではなく、ビューに指定された INSTEAD-OF トリガーを起動します。トリガー本体に含める必要があるすべての更新セマンティクスはカプセル化できます。

オブジェクト・キャッシュ

なぜオブジェクト・キャッシュが必要なのですか？

オブジェクト・キャッシュを使用することで、アプリケーションには次のような利点があります。

- データベース・オブジェクトをメモリー内のホスト言語オブジェクトに透過的にマップできます。
- 永続オブジェクトに対して透過的で効率的なメモリー管理ができます。アプリケーションがデータベース・オブジェクトにアクセスするためのメモリーの割当てを意識する必要はありません。
- クライアント側オブジェクトに対してトランザクション・セマンティクスが使用できます。オブジェクト・キャッシュで変更された永続オブジェクトは、クライアントとサーバー間の 1 回のラウンドトリップで、データベースに反映されます。

- オブジェクトのナビゲーション・アクセスが可能です。オブジェクト・キャッシュによって、ナビゲーション・スタイルでオブジェクトにアクセスできます。OCI のオブジェクト機能を使用すると、REF を確保することによって、オブジェクトがオブジェクト・キャッシュにフェッチされます。ナビゲーション・スタイルでのオブジェクトへのアクセスは、REF を介して相互接続されているオブジェクトを操作する場合により適しています。
- 複合オブジェクトの取出しが可能です。つまり、オブジェクトをサーバーからフェッチする 1 回の要求で、REF を介してそのオブジェクトに関連付けられている他のオブジェクトを含めて、クライアント・サーバー間の 1 回のラウンドトリップで取り出すことができます。

オブジェクト・ロックはオブジェクト・キャッシュでサポートされますか？

オブジェクト・キャッシュでは、悲観的ロック構造および楽観的ロック構造の両方がサポートされます。

- 悲観的ロック構造では、オブジェクトは、キャッシュにあるオブジェクトが変更される前にロックされ、ロックを所有しているトランザクションがコミットまたはロールバックを実行するまで、他のどのユーザーもそのオブジェクトを変更できないようにします。
- 楽観的ロック構造では、オブジェクトは、ロックを取得せずに、キャッシュ内でフェッチおよび変更されます。オブジェクトがサーバーにフラッシュされる場合にのみ、ロックが取得されます。楽観的ロックでは、悲観的ロックより高度な同時アクセスが可能です。楽観的ロックを効果的に使用するために、オブジェクト・キャッシュは、オブジェクトがキャッシュにフェッチされてから、他のユーザーによって変更されたかどうかを検出する機能を提供します。オブジェクト変更検出モードにすると、オブジェクトがキャッシュにフェッチされてから他のユーザーによって変更されていない場合にのみ、そのオブジェクトの変更が持続的に行われます。

ラージ・オブジェクト (LOB)

Oracle8i を使用してラージ・オブジェクトを管理するにはどのようにすればいいですか？

テキスト、イメージ、オーディオ、ビデオなどのマルチメディア・データ型をサポートするには、バイナリおよび文字データに対する強固なサポートが必要です。このような種類のデータは大きくなる傾向があり、様々なバイナリ・データに直接アクセスする必要があります。そのため、Oracle8i では、大規模のバイナリ・データおよび文字データのサポートが大幅に改善されています。ここでラージ・オブジェクト型 (LOB) が導入され、イメージ、オーディオ・ファイル、テキストおよび空間データを含む、様々なドメインからのドメイン固有の大きなデータを格納するために使用されます。

Oracle8i では、バイナリ、文字ベースおよびファイル・ベースの 3 つのラージ・データ・オブジェクトがサポートされています。LOB を作成する機能に加えて、Oracle8i サーバーでは、バイナリ・データを管理するために改善された点もいくつかあります。これらの改善点は、次のとおりです。

- 表に 1 つ以上の LOB 列を定義するためのサポート
- LOB データへのランダムでピース単位のアクセス
- LOB データを単一のストリームとして転送するためのサポート
- LOB データに対するロギングまたはキャッシュ（あるいはその両方）を禁止するためのサポート
- LOB をインライン行記憶域から別のセグメントまたは別の表領域内の行外記憶域に透過的に移動するためのサポート

LOB の詳細は、『Oracle8i アプリケーション開発者ガイド ラージ・オブジェクト』を参照してください。

ユーザー定義オペレータ

ユーザー定義オペレータとは何ですか？

Oracle8i では、オブジェクト指向アプリケーションの開発者は、ユーザー定義オペレータと呼ばれるドメイン固有の演算子（たとえば、Contains、Within_Distance、Similar）を持つ組込み関係演算子（たとえば、+、-、/、*、LIKE）のリストを拡張できます。ユーザー定義オペレータは、Select 構文のリストまたは where 句など、組込み演算子が使用できる場所ではどこでも使用できます。組込み演算子と同様に、ユーザー定義オペレータは別の型の引数をサポートできます。また、索引を使用して評価できます。

ユーザー定義オペレータの詳細は、『Oracle8i SQL リファレンス』の「CREATE OPERATOR」または『Oracle8i データ・カートリッジ開発者ガイド』を参照してください。

ユーザー定義オペレータはなぜ役に立つのですか？

組込み演算子と同様に、ユーザー定義オペレータでは、オブジェクト・データに対して効果的な内容ベースの問合せおよびソートができます。たとえば、一定の資格を含む履歴書を検索するには、Contains オペレータを SQL の where 句の一部として指定できます。オプティマイザは、関係演算子を評価するために B ツリー索引を使用すると同様に、履歴書の列にテキスト索引を使用するように選択して、問合せを効率的に実行します。

オブジェクト・リレーショナル機能を使用した サンプル・アプリケーション

この章では、ユーザー定義データ型（Oracle オブジェクト）の使用方法的応用例について説明します。この例では、リレーショナル・モデルがどのようにオブジェクト・リレーショナル・モデルに変換されるかを示します。オブジェクト・リレーショナル・モデルでは、アプリケーションによって管理される実社会のエンティティがさらによく表現されます。

内容は次のとおりです。

- [概要](#)
- [発注書の例](#)
 - [リレーショナル・モデルでのアプリケーションの実装](#)
 - [オブジェクト・リレーショナル・モデルでのアプリケーションの実装](#)
- [Java を使用したオブジェクトの操作](#)
- [Oracle Objects for OLE でのオブジェクトの操作](#)

概要

ユーザー定義型とは、アプリケーション内のデータ構造および操作を形式化したスキーマ・オブジェクトのことです。

この章の例では、ユーザー定義型の定義および使用における最も重要な点を説明します。ユーザー定義型の使用で重要な点の1つは、オブジェクトに対して操作を実行するメソッドを作成することです。この例では、オブジェクト型メソッドの定義で PL/SQL 言語を使用します。型の定義など、ユーザー定義型の使用にかかわるそれ以外の部分では SQL を使用します。

PL/SQL および Java を使用すると、特にコレクション要素のアクセスおよび操作の点で、この章に示す以上の機能が実現できます。

Oracle Call Interface (OCI)、Pro*C/C++ または Oracle Objects for OLE (OO4O) を使用するクライアント・アプリケーションでは、その広範な機能を利用してオブジェクトおよびコレクションにアクセスし、それらをクライアント上で操作できます。

参照：

- ユーザー定義型の使用の概要および使用方法の詳細は、『Oracle8i 概要』を参照してください。
 - ユーザー定義型の SQL 構文および使用方法の詳細は、『Oracle8i SQL リファレンス』を参照してください。
 - PL/SQL の機能の詳細は、『Oracle8i PL/SQL ユーザーズ・ガイドおよびリファレンス』を参照してください。
 - Java の詳細は、『Oracle8i Java ストアド・プロシージャ開発者ガイド』を参照してください。
 - 『Oracle8i コール・インタフェース・プログラマーズ・ガイド』も参照してください。
 - 『Oracle8i Pro*C/C++ プリコンパイラ・プログラマーズ・ガイド』も参照してください。
 - 『Oracle Objects for OLE/ActiveX Programmer's Guide』も参照してください。
-
-

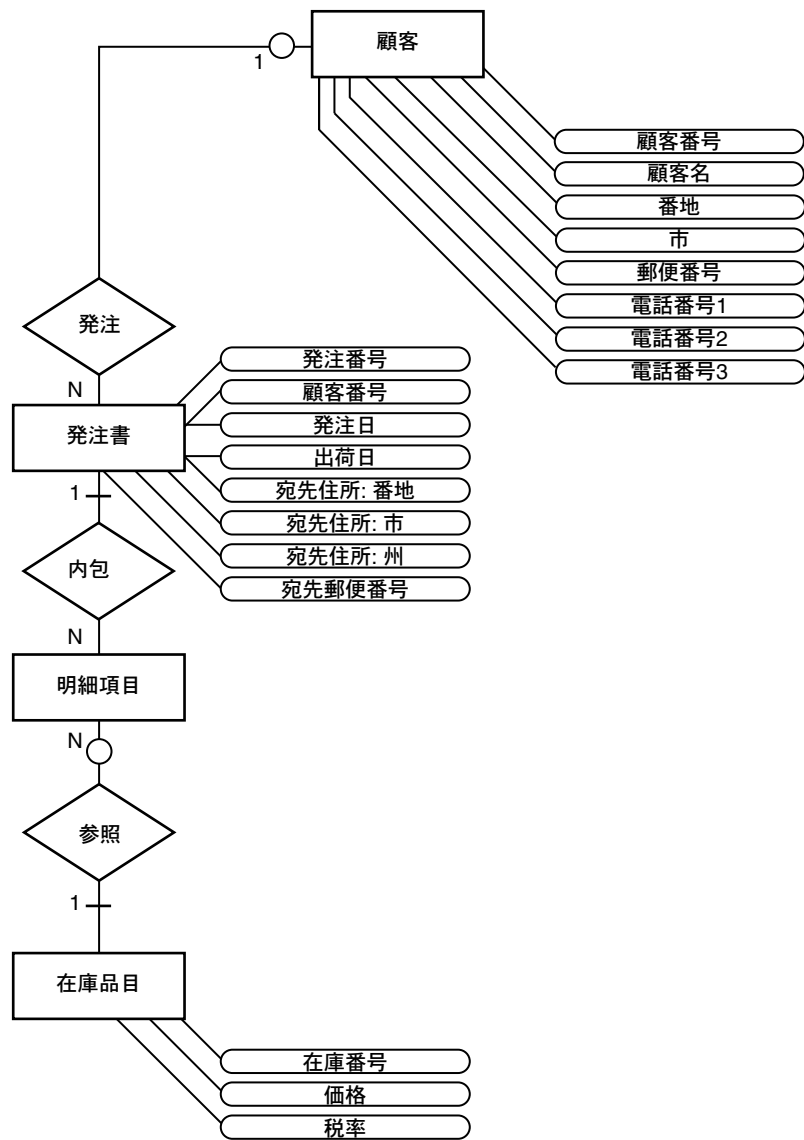
発注書の例

この例は、顧客からの注文の管理という、典型的な業務活動に基づいています。アプリケーションがリレーショナルからオブジェクト・リレーショナルへと発展する様子、および純粋なオブジェクト指向のアプローチを使用して、初めからアプリケーションを作成する方法を示します。

- **「リレーショナル・モデルでのアプリケーションの実装」**では、Oracle の組み込みデータ型のみを使用してスキーマを実装する方法を示します。第 4 章で説明したオブジェクト・ビューを使用して、このリレーショナル・スキーマ上にオブジェクト指向アプリケーションを作成できます。
- **「オブジェクト・リレーショナル・モデルでのアプリケーションの実装」**では、Oracle のオブジェクト型を使用して、アプリケーション・ドメインのエンティティおよび関連を表します。このアプローチでは、オブジェクト表を使用して、基礎となるデータを保持し、オブジェクトの動作をメソッド・ファンクションでカプセル化します。

リレーショナル・モデルでのアプリケーションの実装

図 8-1 発注アプリケーションの E-R 図



エンティティおよび関連

この例の基本エンティティは、次のとおりです。

- 顧客
- 販売製品の在庫
- 発注書

図 8-1 からわかるように、顧客には連絡用の情報があり、住所および一連の電話番号はその顧客専用です。このアプリケーションでは、異なる顧客が同一の住所または電話番号と対応付けられることはありません。顧客が住所を変更した場合、以前の住所はなくなります。顧客でなくなった場合は、対応付けられた住所は削除されます。

顧客と発注書は 1 対多関連になります。1 人の顧客が複数の発注を行うことはできますが、ある 1 つの発注書を発行できる顧客は 1 人のみです。発注を行う前に顧客が定義されるため、1 対多関連は必須でなく任意のものです。

同様に、発注書と在庫品目は多対多関連になります。多対多の関連では、どの在庫品目がどの発注書に記載されているかが示されないため、E-R には明細項目という概念があります。発注書には、明細項目が 1 つ以上含まれる必要があります。各明細項目は、1 つの発注書のみに対応付けられています。

明細項目と在庫品目の関連は、在庫品目がどの明細項目にも記載されなくても、1 つの明細項目または多くの明細項目に記載されてもかまいませんが、各明細項目が参照する在庫品目は 1 つのみです。

リレーショナル・モデルでの表の作成

リレーショナル・アプローチでは、すべてを正規化し、表にします。表名は、Customer_reltab、PurchaseOrder_reltab および Stock_reltab です。

住所の各部は、Customer_reltab 表の列になります。

電話番号を列として構造化することによって、顧客の電話番号の数に任意の制限を設定します。

リレーショナル・アプローチでは、発注書と明細項目を切り離し、PurchaseOrder_reltab および LineItems_reltab という名前の独自の表に組み込みます。図 8-1 で示されるように、明細項目は、発注および在庫品目の両方に対して関連を持ちます。これらは、PurchaseOrder_reltab および Stock_reltab への外部キーを持つ、LineItems_reltab 表の列として実装されます。

注意： この項では、リレーショナル表の名前には接尾辞 _reltab を付ける規則に従っています。このような記述法を定めることによって、コードのメンテナンスが簡単になります。

これが表 (_tab) と型 (_typ) の区別に役立ちます。ただし、任意の名前を選択できます。オブジェクト・リレーショナル手法の主な長所の 1 つは、実社会のオブジェクトを忠実にモデル化する名前を、ソフトウェア・エンティティに付けることができることです。

リレーショナル・アプローチの表は、次のようになります。

Customer_reltab

Customer_reltab 表の定義は、次のようになります。

```
CREATE TABLE Customer_reltab (  
  CustNo          NUMBER NOT NULL,  
  CustName        VARCHAR2(200) NOT NULL,  
  Street          VARCHAR2(200) NOT NULL,  
  City            VARCHAR2(200) NOT NULL,  
  State           CHAR(2) NOT NULL,  
  Zip             VARCHAR2(20) NOT NULL,  
  Phone1          VARCHAR2(20),  
  Phone2          VARCHAR2(20),  
  Phone3          VARCHAR2(20),  
  PRIMARY KEY (CustNo)  
);
```

この Customer_reltab 表には、顧客に関するすべての情報が格納されます。これは、(NOT NULL 制約で定義される) 顧客に固有の情報、および必須ではないすべての情報が含まれることを意味します。この表定義によると、このアプリケーションではすべての顧客が出荷先住所を持つことが必須です。

E-R 図では発注を行う顧客が示されていましたが、表では、顧客と発注書間の関連は考慮されていません。

これは、この関連を発注書によって管理する必要があることを示しています。

PurchaseOrder_reltab

PurchaseOrder_reltab 表の定義は、次のようになります。

```
CREATE TABLE PurchaseOrder_reltab (
  PONo          NUMBER, /* purchase order no */
  Custno        NUMBER references Customer_reltab, /* Foreign KEY referencing
                                                    customer */

  OrderDate     DATE, /* date of order */
  ShipDate      DATE, /* date to be shipped */
  ToStreet      VARCHAR2(200), /* shipto address */
  ToCity        VARCHAR2(200),
  ToState       CHAR(2),
  ToZip         VARCHAR2(20),
  PRIMARY KEY (PONo)
) ;
```

このように、PurchaseOrder_reltab は外部キー（FK）列 CustNo によって、顧客と発注書間の関連を管理します。CustNo は、Customer_reltab の CustNo キーを参照します。この表では、発注書と明細項目の間の関連は考慮されないため、明細項目のリストで処理する必要があります。

LineItems_reltab

LineItems_reltab 表の定義は、次のようになります。

```
CREATE TABLE LineItems_reltab (
  LineItemNo     NUMBER,
  PONo           NUMBER REFERENCES PurchaseOrder_reltab,
  StockNo        NUMBER REFERENCES Stock_reltab,
  Quantity       NUMBER,
  Discount       NUMBER,
  PRIMARY KEY (PONo, LineItemNo)
) ;
```

注意： LineItems_reltab 表を作成する前に、8-8 ページの「[Stock_reltab](#)」で説明される Stock_reltab 表を作成してください。

表名は、複数形 LineItems_reltab であり、コードを読む他のユーザーに対して、この表が明細項目のコレクションを保持することを強調しています。

E-R 図で示すように、明細項目のリストは発注書および在庫品目の両方に関連を持ちます。これらの関連は、LineItems_reltab で次の 2 つの外部キー列によって管理されます。

- PONo: PurchaseOrder_reltab の PONo 列を参照します。
- StockNo: Stock_reltab の StockNo 列を参照します。

Stock_reltab

Stock_reltab 表の定義は、次のようになります。

```
CREATE TABLE Stock_reltab (  
  StockNo      NUMBER PRIMARY KEY,  
  Price        NUMBER,  
  TaxRate      NUMBER  
);
```

リレーショナル・モデルでの値の挿入

このアプリケーションでは、次の文で表にデータを挿入します。

在庫品目録を作成する

```
INSERT INTO Stock_reltab VALUES(1004, 6750.00, 2) ;  
INSERT INTO Stock_reltab VALUES(1011, 4500.23, 2) ;  
INSERT INTO Stock_reltab VALUES(1534, 2234.00, 2) ;  
INSERT INTO Stock_reltab VALUES(1535, 3456.23, 2) ;
```

顧客を登録する

```
INSERT INTO Customer_reltab  
VALUES (1, 'Jean Nance', '2 Avocet Drive',  
       'Redwood Shores', 'CA', '95054',  
       '415-555-1212', NULL, NULL) ;  
  
INSERT INTO Customer_reltab  
VALUES (2, 'John Nike', '323 College Drive',  
       'Edison', 'NJ', '08820',  
       '609-555-1212', '201-555-1212', NULL) ;
```

発注する

```
INSERT INTO PurchaseOrder_reltab  
VALUES (1001, 1, SYSDATE, '10-MAY-1997',  
       NULL, NULL, NULL, NULL) ;
```

```
INSERT INTO PurchaseOrder_reltab
VALUES (2001, 2, SYSDATE, '20-MAY-1997',
      '55 Madison Ave', 'Madison', 'WI', '53715') ;
```

明細項目の細目

```
INSERT INTO LineItems_reltab VALUES(01, 1001, 1534, 12, 0) ;
INSERT INTO LineItems_reltab VALUES(02, 1001, 1535, 10, 10) ;
INSERT INTO LineItems_reltab VALUES(01, 2001, 1004, 1, 0) ;
INSERT INTO LineItems_reltab VALUES(02, 2001, 1011, 2, 1) ;
```

リレーショナル・モデルでのデータの問合せ

このアプリケーションでは、次の問合せを実行できます。

特定の発注書に対する顧客データおよび明細項目データを取得する

```
SELECT  C.CustNo, C.CustName, C.Street, C.City, C.State,
        C.Zip, C.phone1, C.phone2, C.phone3,
        P.PONo, P.OrderDate,
        L.StockNo, L.LineItemNo, L.Quantity, L.Discount
FROM    Customer_reltab C,
        PurchaseOrder_reltab P,
        LineItems_reltab L
WHERE   C.CustNo = P.CustNo
        AND      P.PONo = L.PONo
        AND      P.PONo = 1001 ;
```

発注書の合計値を取得する

```
SELECT  P.PONo, SUM(S.Price * L.Quantity)
FROM    PurchaseOrder_reltab P,
        LineItems_reltab L,
        Stock_reltab S
WHERE   P.PONo = L.PONo
        AND      L.StockNo = S.StockNo
GROUP BY P.PONo ;
```

特定の在庫番号で識別される在庫品目を使用する明細項目に対する発注書データおよび明細項目データを取得する

```
SELECT  P.PONo, P.CustNo,
        L.StockNo, L.LineItemNo, L.Quantity, L.Discount
FROM    PurchaseOrder_reltab P,
```

```
        LineItems_reltab      L
WHERE    P.PONo = L.PONo
AND      L.StockNo = 1004 ;
```

リレーショナル・モデルでのデータの更新

このアプリケーションでは、次の文を実行してデータを更新できます。

発注書 1001 の在庫品目 1534 の数量を更新する

```
UPDATE LineItems_reltab
SET      Quantity = 20
WHERE    PONo      = 1001
AND      StockNo   = 1534 ;
```

リレーショナル・モデルでのデータの削除

このアプリケーションでは、次の文を実行してデータを削除できます。

発注書 1001 を削除する

```
DELETE
FROM    LineItems_reltab
WHERE    PONo = 1001 ;

DELETE
FROM    PurchaseOrder_reltab
WHERE    PONo = 1001 ;
```

純粋なリレーショナル・モデルの制限事項

リレーショナル・データベース管理システム（RDBMS）は、強力で効率的な情報管理形態です。それでは、なぜ別のアプローチを考慮する必要があるのでしょうか。リレーショナル・モデルで開発されたアプリケーションを、アプリケーション・ドメインの実社会と比較すると、ある欠点が明らかになります。

データ（構造体）および操作（動作）のカプセル化の制限事項

データベース表は、関連の構造をモデル化するには優れていますが、実社会のオブジェクトが、そのデータに関する操作と自然に結び付けられている状態を獲得できていません。たとえば、実社会で発注書进行处理する場合、明細項目を合計してその顧客に対する合計金額がわかると期待します。同様に、名前、参照番号、住所などの発注元の顧客についての情報を取り出せることも期待します。さらに、顧客の購買履歴および支払いパターンを判断する場合もあります。

RDBMS では、データの格納および取出しに非常に洗練された構造体が提供されますが、個々のアプリケーション開発者は、それぞれのアプリケーションに必要な操作を手作りする必要があります。これは、同一企業内のアプリケーションですでにコーディングされた操作とほとんど同じ操作であっても、再コーディングが必要な場合がよくあることを意味します。

コンポジット操作の制限事項

リレーショナル表では、合成されたものは取得できません。たとえば、住所は番地、通り、市、州および郵便番号のコンポジットの場合がありますが、リレーショナル表では、個々の列が組み合わされた構造体としての住所の表記は、獲得されません。

集計操作の制限事項

リレーショナル表では、複雑な部分対全体の関連の処理は困難です。1 個のピストンおよび 1 個のエンジンは、`Stock_reltab` の列として同じ状態にありますが、主キーと外部キーの関連を持つ複数の表を作成せずにピストンがエンジンの一部である事実を表す簡単な方法はありません。同様に、コレクション同士の複雑な内部関連を実装する簡単な方法もありません。

汎用化 / 特殊化の処理の制限事項

汎用化 / 特殊化の関連（継承）を獲得する簡単な方法はありません。発注書の基本的な要件を抽象化し、関連を取得するコードを記述する場合、このコードを使用する発注書を作成し、異なるドメインに対してさらにコードを特殊化する方法はありません。かわりに、発注書のすべての実装においてコードを複製します。

オブジェクト・リレーショナル・データベース・システムの発展

第3世代言語（3GL）を使用してアプリケーションを作成しない理由を考えてみます。

まず、RDBMS では、レプリケートするために何百万人時もの工数がかかるような機能が提供されています。

第2に、3GL を使用した情報管理には、持続性がないという問題があります。持続性があったとしても、必要なパフォーマンスを得るために同一のアドレス空間にアプリケーション論理およびデータ論理を置くため、セキュリティを犠牲にしまいます。持続性およびセキュリティの両方が基本要件である RDBMS のユーザーにとっては、どちらの要件も欠かせません。

このため、リレーショナル・モデルを使用しているアプリケーション開発者は、ある形式で SQL にマッピングすることによって複合型をシミュレートする必要があります。この方法では、効率が悪だけでなく、実装に関する深刻な問題があります。たとえば、次のような問題です。

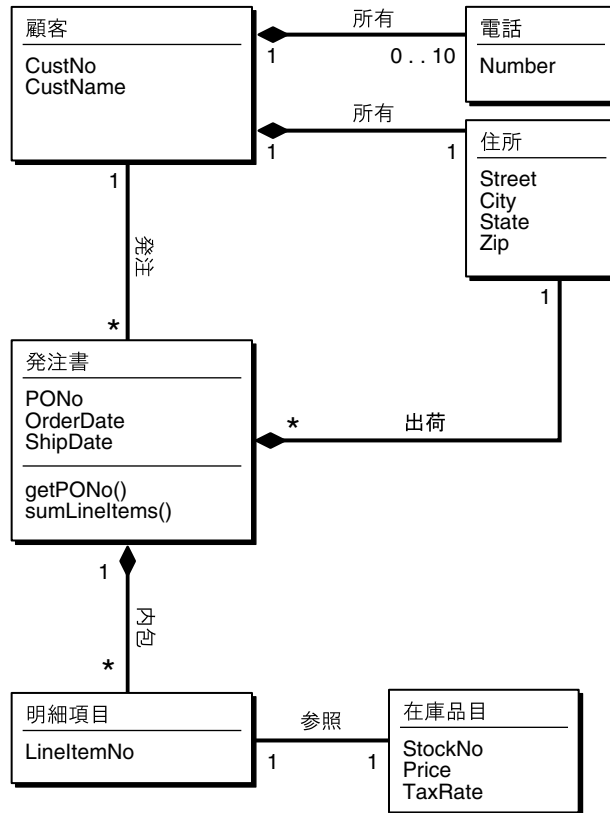
- 書き込み時のアプリケーション論理からデータ論理への変換
- 読み込み時のデータ論理からアプリケーション論理への変換

これでは、クライアントとサーバーのアドレス空間の間の通信量が大量になるため、パフォーマンスが低下します。クライアントおよびサーバーが別々のマシン上にある場合、ネットワーク・ラウンドトリップで多くのオーバーヘッドが発生する可能性があります。

これらの問題は、オブジェクト・リレーショナル（O-R）・テクノロジーによって解決されます。この章の後半では、この新機能の実装例について説明します。

オブジェクト・リレーショナル・モデルでのアプリケーションの実装

図 8-2 発注アプリケーションのクラス図



オブジェクト・リレーショナル (O-R)・アプローチは、8-5 ページの「[エンティティおよび関連](#)」と同じ E-R から始まります。これを前述のクラス図のようにオブジェクト指向の観点で見ることによって、実社会の構造をより忠実にデータベース・スキーマに変換できます。

O-R アプローチでは、住所または複数の電話番号をリレーショナル表の関連のない列に分けて入れるのではなく、それらを表す型を定義します。また、明細項目を切り離して別の表に入れるのではなく、それぞれの発注書と一緒にして、NESTED TABLE を作成します。

メイン・エンティティである顧客、在庫および発注書が、オブジェクトになります。オブジェクト参照で、オブジェクト間の関連を表します。コレクション型で、複数値属性をモデル化します。

オブジェクト・リレーショナルの実装に対して、次の2つのアプローチがあります。

- オブジェクト表を作成してデータを移入します。
- オブジェクト・ビューを使用して、既存のリレーショナル・データから仮想的なオブジェクト表を表現します。

この章の後半では、O-R スキーマを示し、それをオブジェクト表で実装する方法を説明します。第4章「オブジェクト・モデルのリレーショナル・データへの適用」では、オブジェクト・ビューを使用して同じスキーマを実装します。

型の定義

次の文で、先送り型定義を行い、アプリケーションの準備をします。

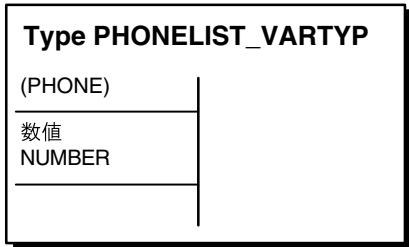
```
CREATE TYPE StockItem_objtyp;  
CREATE TYPE LineItem_objtyp;  
CREATE TYPE PurchaseOrder_objtyp;
```

先送り型定義を行うと、Oracle では、完全な定義が後で行われると解釈されます。Oracle では、このような不完全な型を参照する型がコンパイルされます。C などのプログラム言語のプロトタイプのようなものです。

次の文で、VARRAY 型を定義します。

```
CREATE TYPE PhoneList_vartyp AS VARRAY(10) OF VARCHAR2(20);
```

図 8-3 PHONELIST_VARTYP 型のオブジェクト・リレーショナル表現



前述の文は、PHONELIST_VARTYP 型を定義します。PHONELIST_VARTYP 型のすべてのデータ単位は、最大 10 件の電話番号の VARRAY であり、各番号は VARCHAR2 型のデータ項目で表されます。

電話番号のリストには、VARRAY または NESTED TABLE を使用できます。この場合、リストは顧客ごとの連絡用電話番号の集合です。次の理由から、NESTED TABLE より VARRAY の方をお薦めします。

- 番号の順序が重要な場合があります。NESTED TABLE は順序付けられませんが、VARRAY は順序付けられます。
- 特定の顧客の電話番号の数は多くありません。VARRAY では、要素の最大数（この場合は 10）を事前に指定する必要があります。そのため、サイズ制限のない NESTED TABLE よりも、記憶域を効率よく使用できます。
- 電話番号リストに対して問合せを行うことはないため、NESTED TABLE の形式には利点がありません。

通常、順序付けおよび記憶域制限が設計上重要な問題でないときは、次のように VARRAY か NESTED TABLE かを決定できます。つまり、コレクションを問い合わせる必要がある場合は NESTED TABLE を使用し、コレクション全体を 1 つとして取り出す場合は VARRAY を使用します。

参照： VARRAY および NESTED TABLE の設計の詳細は、[第 5 章「Oracle オブジェクトの設計上の考慮点」](#)を参照してください。

次の文で、住所を表すオブジェクト型 ADDRESS_OBJTYP を定義します。

```
CREATE TYPE Address_objtyp AS OBJECT (  
  Street      VARCHAR2(200),  
  City        VARCHAR2(200),  
  State       CHAR(2),  
  Zip         VARCHAR2(20)  
)  
/
```

図 8-4 ADDRESS_OBJTYP 型のオブジェクト・リレーショナル表現

ADDRESS_OBJTYP型			
STREET	CITY	STATE	ZIPCODE
テキスト VARCHAR2(200)	テキスト VARCHAR2(200)	テキスト CHAR(2)	数値 VARCHAR2(20)

住所のすべての属性は文字列で、住所を単純化して最小限の部分に分割したものです。

次の文で、オブジェクト型 CUSTOMER_OBJTYP を定義します。この型は、他のユーザー定義型を組み込みブロックとして使用します。

```
CREATE TYPE Customer_objtyp AS OBJECT (  
  CustNo      NUMBER,  
  CustName    VARCHAR2(200),  
  Address_obj Address_objtyp,  
  PhoneList_var PhoneList_vartyp,  
  
  ORDER MEMBER FUNCTION  
    compareCustOrders(x IN Customer_objtyp) RETURN INTEGER  
)  
/
```

CUSTOMER_OBJTYP 型のインスタンスは、特定の顧客についての情報ブロックを表すオブジェクトです。CUSTOMER_OBJTYP オブジェクトの属性は、数値、文字列、ADDRESS_OBJTYP オブジェクトおよび PHONELIST_VARTYP 型の VARRAY です。

すべての CUSTOMER_OBJTYP オブジェクトには、2 種類の比較メソッドのうちの 1 つ、オーダー・メソッドが対応付けられています。Oracle では、2 つの CUSTOMER_OBJTYP オブジェクトの比較が必要になると、compareCustOrders メソッドが起動されます。

注意： 比較メソッドを実装する PL/SQL は、8-22 ページの「[compareCustOrders メソッド](#)」にあります。

2 種類の比較メソッドとは、マップ・メソッドおよびオーダー・メソッドです。このアプリケーションでは、説明のため、両方のメソッドをそれぞれ使用します。

オーダー・メソッドは、比較する 2 つのオブジェクトごとにコールする必要があるのに対して、マップ・メソッドは、各オブジェクトにつき 1 回のみコールします。一般に、オブジェクトの集合をソートする場合、オーダー・メソッドのコール回数は、マップ・メソッドより多くなります。

参照：

- オーダー・メソッドおよびマップ・メソッドの設計上の考慮事項の詳細は、[第 5 章「Oracle オブジェクトの設計上の考慮点」](#)を参照してください。
 - プラグマ宣言の使用の詳細は、『Oracle8i PL/SQL ユーザーズ・ガイドおよびリファレンス』を参照してください。
-

次の文は、この項の始めに宣言した不完全なオブジェクト型 LINEITEM_OBJTYP の定義を完成させます。

```
CREATE TYPE LineItem_objtyp AS OBJECT (  
  LineItemNo    NUMBER,  
  Stock_ref     REF StockItem_objtyp,  
  Quantity      NUMBER,  
  Discount      NUMBER  
)  
/
```

図 8-5 LINEITEM_OBJTYP 型のオブジェクト・リレーショナル表現

LINEITEM_OBJTYP型			
LINEITEMNO	STOCK_REF	QUANTITY	DISCOUNT
数値 NUMBER	参照 STOCKITEM_OBJTYP	数値 NUMBER	数値 NUMBER

LINEITEM_OBJTYP 型のインスタンスは、明細項目を表すオブジェクトです。3つの数値属性および1つの REF 属性で構成されます。LineItem_objtyp では、明細項目エンティティをモデル化しており、対応する在庫オブジェクトへのオブジェクト参照が含まれます。

次の文で、NESTED TABLE 型の LINEITEMLIST_NTABTYP を定義します。この型は、発注書内の明細項目の任意の集合を表します。

```
CREATE TYPE LineItemList_ntabtyp AS TABLE OF LineItem_objtyp  
/
```

この型のデータ単位は NESTED TABLE で、各行には LINEITEM_OBJTYP 型のオブジェクトが含まれます。複数値明細項目リストを表すには、次の理由から、LINEITEM_OBJTYP オブジェクトの VARRAY より明細項目の NESTED TABLE をお勧めします。

- 明細項目の内容の問合せは、ほとんどのアプリケーションで必要です。ただし、VARRAY の記憶域表現は表の表現とは異なるため、この問合せは VARRAY には適していません。
- アプリケーションで明細項目データの索引付けが必要な場合、NESTED TABLE では索引付けできますが、VARRAY ではできません。

- 明細項目の順序は、通常は、重要ではなく、必要なときには明細項目番号で識別できます。
- 発注書の明細項目の数には、実質的な上限はありません。VARRAY を使用すると、要素数の任意の上限を指定する必要があります。

次の文は、この項の始めに宣言した不完全なオブジェクト型 PURCHASEORDER_OBJTYP の定義を完成させます。

```
CREATE TYPE PurchaseOrder_objtyp AUTHID CURRENT_USER AS OBJECT (  
    PONo                NUMBER,  
    Cust_ref            REF Customer_objtyp,  
    OrderDate           DATE,  
    ShipDate            DATE,  
    LineItemList_ntab   LineItemList_ntabtyp,  
    ShipToAddr_obj      Address_objtyp,  
  
    MAP MEMBER FUNCTION  
        getPONo RETURN NUMBER,  
  
    MEMBER FUNCTION  
        sumLineItems RETURN NUMBER  
);  
/
```

図 8-6 PURCHASEORDER_OBJTYP のオブジェクト・リレーショナル表現

PURCHASEORDER_OBJTYP型					
PONO	CUST_REF	ORDERDATE	SHIPDATE	LINEITEMLIST_NTAB	SHIPTOADDR_OBJ
数値 NUMBER	参照 CUSTOMER_ OBJTYP	日付 DATE	日付 DATE	NESTED TABLE LINEITEMLIST_ NTABTYP	オブジェクト型 ADDRESS_ OBJTYP
PK	FK				
メンバー関数getPONoによりNUMBERが戻される メンバー関数SumLineItemsによりNUMBERが戻される					

前述の文で、オブジェクト型 PURCHASEORDER_OBJTYP が定義されます。この型のインスタンスは発注書を表すオブジェクトです。これには、CUSTOMER_OBJTYP への REF、ADDRESS_OBJTYP オブジェクトおよび LINEITEM_OBJTYP 型を基にしている LINEITEMLIST_NTABTYP 型の NESTED TABLE を含む 6 つの属性があります。

PURCHASEORDER_OBJTYP 型のオブジェクトには、getPONo および sumLineItems の 2 つのメソッドがあります。そのうちの 1 つ getPONo はマップ・メソッドで、2 種類の比較メソッドの 1 つです。マップ・メソッドは、オブジェクトのレコード順序内における指定レコードの相対位置を戻します。そのため、Oracle では、2 つの PURCHASEORDER_OBJTYP オブジェクトの比較が必要になると、getPONo メソッドが明示的にコールされます。

2 つのプラグマ宣言で、2 つのメソッドでデータベースに対してどのようなアクセスが必要かが PL/SQL に通知されます。

前述の文には、メソッド getPONo および sumLineItems を実装する、実際の PL/SQL プログラムは含まれていません。実際のプログラムは、8-21 ページの「メソッドの定義」にあります。

次の文は、この項の始めに宣言した 3 つの不完全なオブジェクト型の最後の 1 つ STOCKITEM_OBJTYP の定義を完成させます。

```
CREATE TYPE StockItem_objtyp AS OBJECT (  
    StockNo    NUMBER,  
    Price      NUMBER,  
    TaxRate    NUMBER  
)  
/
```

図 8-7 STOCKITEM_OBJTYP のオブジェクト・リレーショナル表現

STOCKITEM_OBJTYP型		
STOCKNO	PRICE	TAXRATE
数値 NUMBER	数値 NUMBER	数値 NUMBER
PK		

STOCKITEM_OBJTYP 型のインスタンスは、顧客が発注する在庫品目を表すオブジェクトです。これには、3 つの数値属性があります。

メソッドの定義

この項では、PURCHASEORDER_OBJTYP オブジェクト型および CUSTOMER_OBJTYP オブジェクト型のメソッドを指定する方法を示します。次の文で、PURCHASEORDER_OBJTYP オブジェクト型の本体（メソッドを実装する PL/SQL プログラム）を定義します。

```
CREATE OR REPLACE TYPE BODY PurchaseOrder_objtyp AS

MAP MEMBER FUNCTION getPONo RETURN NUMBER is
BEGIN
    RETURN PONo;
END;

MEMBER FUNCTION sumLineItems RETURN NUMBER is
    i          INTEGER;
    StockVal    StockItem_objtyp;
    Total       NUMBER := 0;

BEGIN
    FOR i in 1..SELF.LineItemList_ntab.COUNT LOOP
        UTL_REF.SELECT_OBJECT(LineItemList_ntab(i).Stock_ref,StockVal);
        Total := Total + SELF.LineItemList_ntab(i).Quantity * StockVal.Price;
    END LOOP;
    RETURN Total;
END;
END;
/
```

getPONo メソッド

getPONo メソッドは単純です。これを使用して、対応付けられた PURCHASEORDER_OBJTYP オブジェクトの発注番号を戻します。このような取得メソッドを使用すると、オブジェクトの内部表現が変更された場合に、そのオブジェクトを使用するコードを再操作せずに済みます。

sumLineItems メソッド

sumLineItems メソッドでは、多くの O-R 機能を使用します。

- 前述のとおり、sumLineItems メソッドの基本的な機能は、それに対応付けられた PURCHASEORDER_OBJTYP オブジェクトの明細項目の請求額の合計を戻すことです。キーワード SELF は、すべての関数のパラメータとして暗黙的に作成され、そのオブジェクトを参照します。
- キーワード COUNT は、PL/SQL 表または配列にある要素の総数を表します。ここで LOOP を組み合わせると、アプリケーションは、コレクションのすべての要素（この場合、発注書の項目）で処理を繰り返します。

このように、SELF.LineItemList_ntab.COUNT は NESTED TABLE 内の要素の数を表し、SELF で表される PURCHASEORDER_OBJTYP オブジェクトの LineItemList_ntab 属性と一致します。

- 実装にあたって、UTL_REF パッケージのメソッドが使用されます。Oracle では、PL/SQL プログラム内の REF の暗黙的な参照解除がサポートされていないため、UTL_REF メソッドが必要です。UTL_REF パッケージによって、オブジェクト参照を操作するメソッドが提供されます。ここで、Stock_ref に対応する STOCKITEM_OBJTYP オブジェクトを取得するために、SELECT_OBJECT メソッドがコールされます。
- AUTHID CURRENT USER 構文は、実行者権限を使用して PURCHASEORDER_OBJTYP が定義されていることを指定します。したがって、このメソッドは、型を定義したユーザーの権限でなく、カレント・ユーザーの権限で実行されます。
- PL/SQL 変数 StockVal は、STOCKITEM_OBJTYP 型です。UTL_REF.SELECT_OBJECT は、これを次のような参照を持つオブジェクトに設定します。

```
(LineItemList_ntab(i).Stock_ref)
```

このオブジェクトは、現在選択されている明細項目で参照される、実際の在庫品目です。

- 対象の在庫品目を取り出すと、次にそのコストが計算されます。このプログラムでは、在庫品目のコストが、StockVal.Price という STOCKITEM_OBJTYP オブジェクトの Price 属性として参照されます。ただし、項目のコストを計算するには、品目の発注量を知っておく必要があります。このアプリケーションでは、LineItemList_ntab(i).Quantity という項で、現在選択されている LINEITEM_OBJTYP オブジェクトの Quantity 属性が表されます。

メソッドの残りのソースコードは、明細項目の請求額を合計するループです。メソッドは、その合計を値として戻します。

compareCustOrders メソッド

次の文で、CUSTOMER_OBJTYP オブジェクト型の compareCustOrders メソッドを定義します。

```
CREATE OR REPLACE TYPE BODY Customer_objtyp AS
  ORDER MEMBER FUNCTION
  compareCustOrders (x IN Customer_objtyp) RETURN INTEGER IS
  BEGIN
    RETURN CustNo - x.CustNo;
  END;
END;
/
```

前述したように、オーダー・メソッド `compareCustOrders` 操作は、2つの顧客の発注書と比較します。別の `Customer_objtyp` オブジェクトを入力引数として受け取り、2つの `CustNo` 数値の差を返します。戻り値は次のとおりです。

- 負数: 内部にあるオブジェクトの `CustNo` の値が小さい場合
- 正数: 内部にあるオブジェクトの `CustNo` の値が大きい場合
- 0 (ゼロ): 2つのオブジェクトが持つ `CustNo` の値が同じ場合。この場合、自分自身を参照します。

戻り値が正、負または0 (ゼロ) であるかによって、顧客番号の相対順序が示されます。たとえば、小さい番号が大きい番号より先に作成されます。オーダー・メソッドの入力引数 (SELF でかつ明示的) のどちらかが NULL である場合、Oracle によってオーダー・メソッドがコールされることなく、結果が NULL として扱われます。

以上が、発注書アプリケーションで使用するユーザー定義型の定義です。ここまでの宣言では、表の作成またはデータ記憶域の予約を行っていません。

オブジェクト表の作成

ここまでは、オブジェクト表を作成してデータを移入する場合も、8-4 ページの「[リレーショナル・モデルでのアプリケーションの実装](#)」で示したリレーショナル表の一番上のオブジェクト・ビューを使用して、アプリケーションを実装する場合も、例は同じです。この章の残りの部分では、引き続きオブジェクト表を使用した例を紹介します。[第4章「オブジェクト・モデルのリレーショナル・データへの適用」](#)では、この点を取り上げ、続けてオブジェクト・ビューの例を説明します。

一般に、オブジェクトとオブジェクト表の関連は、次のように考えられます。

- オブジェクト表へのクラス (エンティティを表す) のマップ
- 列への属性のマップ
- 行へのオブジェクトのマップ

この方法では、各オブジェクト表は、オブジェクト (特定の行) がそれぞれ同じ属性 (列値) を持つ暗黙的な型です。ユーザー定義データ型およびオブジェクト表を明示的に作成すると、新しいレベルの機能性が導入されます。

オブジェクト表 `Customer_objtab`

次の文で、`CUSTOMER_OBJTYP` 型のオブジェクトを保持するオブジェクト表 `Customer_objtab` を定義します。

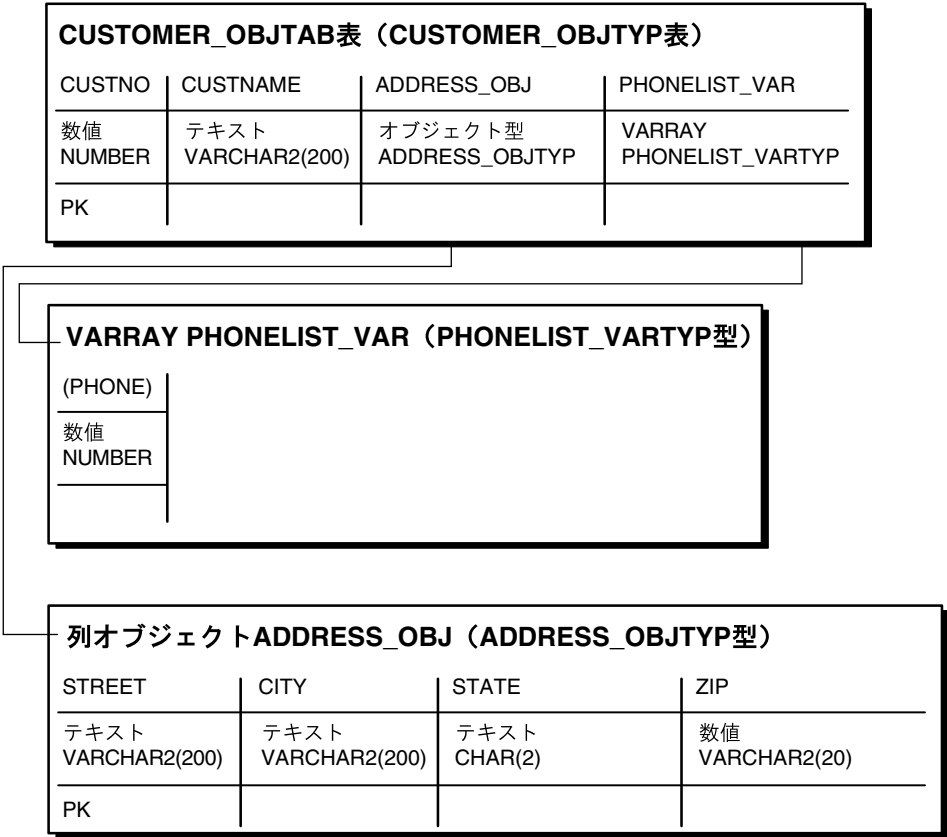
```
CREATE TABLE Customer_objtab OF Customer_objtyp (CustNo PRIMARY KEY)
  OBJECT ID PRIMARY KEY ;
```

この文からもわかるように、OF という項で、オブジェクト表とリレーショナル表の CREATE 文を区別します。事前に定義した CUSTOMER_OBJTYP オブジェクトの属性は、次のとおりです。

CustNo	NUMBER
CustName	VARCHAR2(200)
Address_obj	Address_objtyp
PhoneList_var	PhoneList_vartyp

これは、オブジェクト表 Customer_objtab が、CustNo、CustName、Address_obj および PhoneList_var の各列を持ち、また、各列は CUSTOMER_OBJTYP 型のオブジェクトであることを意味します。行オブジェクトのこの表記法は、機能性において著しく進歩しています。

図 8-8 Customer_objtab 表のオブジェクト・リレーショナル表現



オブジェクト表のテンプレートとしてのオブジェクトのデータ型

CUSTOMER_OBJTYP という型があるため、同じ型のオブジェクト表を多数作成できます。たとえば、CUSTOMER_OBJTYP 型のオブジェクト表 Customer_objtab2 を作成することもできます。この機能がない場合、それぞれの表を個別に定義する必要があります。

複数の表を作成する場合は、バリエーションを導入できます。Customer_objtab を作成した文によって、CustNo 列に主キー制約が定義されました。この制約は、このオブジェクト表のみに適用されます。同じ型の別のオブジェクト表には、この制約がない場合があります。

オブジェクト識別子および参照

Customer_objtab には、行オブジェクトとして表される顧客オブジェクトが含まれています。Oracle では、行オブジェクトが参照できます。これは、他の行オブジェクトまたはリレーショナル行が、行オブジェクトのオブジェクト識別子 (OID) を使用して、行オブジェクトを参照できることを意味します。たとえば、発注書の行オブジェクトは、顧客行オブジェクトに対するオブジェクト参照を使用して、顧客行オブジェクトを参照できます。オブジェクト参照は、REF 型で表されるシステム生成の不透明な値で、行オブジェクトの一意の OID で構成されます。

Oracle では、すべての行オブジェクトには一意の OID があることが必須です。一意の OID 値がシステムによって生成されるように指定することも、行オブジェクトの主キーが一意の OID として機能するように指定することもできます。CREATE TABLE 文を実行するときに、OBJECT ID PRIMARY KEY または OBJECT ID SYSTEM GENERATED (デフォルト) を指定することによって、これを指定します。主キーの値が 16 バイト (システムによって生成される識別子のデフォルト値) より小さい場合、オブジェクト識別子として主キーを選択すると、より効率的です。例では、行オブジェクト識別子として主キーが選択されています。

埋込みオブジェクト付きのオブジェクト表

Customer_objtab の定義を調べると、Address_obj 列に ADDRESS_OBJTYP オブジェクトが含まれていることがわかります。つまり、オブジェクト型は、それ自体がオブジェクト型である属性を持つことができます。これらの埋込みオブジェクトは、コンポジット値または構造体の値を表しており、同時に列オブジェクトとしても参照されます。これらは他を参照することはできず、NULL 値をとることができる点で、行オブジェクトとは異なります。

ADDRESS_OBJTYP オブジェクトには、CUSTOMER_OBJTYP のリーフ・レベルのスカラー属性であることを表す組込み型の属性があります。Oracle では、ADDRESS_OBJTYP オブジェクトの列およびその属性が、オブジェクト表 Customer_objtab に作成されます。これらの列は、ドット表記法を使用して参照できます。たとえば、zip 列に索引を作成する場合、これを Address.Zip として参照できます。

PhoneList_var 列には、VARRAY 型 PHONELIST_VARTYP が含まれます。PHONELIST_VARTYP 型の各オブジェクトは、最大 10 件の電話番号の VARRAY として定義されており、各番号は VARCHAR2 型のデータ項目で表されます。

```
CREATE TYPE PhoneList_vartyp AS VARRAY(10) OF VARCHAR2(20)
/
```

PHONELIST_VARTYP 型の各 VARRAY は、最大 200 文字 (10 x 20) と多少のオーバーヘッドしか含むことができないため、Oracle は、VARRAY を単一のデータ単位として PhoneList_var 列に格納します。Oracle では、4000 バイト未満の VARRAY はインライン BLOB に格納されます。つまり、VARRAY 値の一部は、表外に格納される可能性があります。

オブジェクト表 Stock_objtab

次の文は、STOCKITEM_OBJTYP オブジェクトのオブジェクト表を作成します。

```
CREATE TABLE Stock_objtab OF StockItem_objtyp (StockNo PRIMARY KEY)
  OBJECT ID PRIMARY KEY ;
```

表の各行は、STOCKITEM_OBJTYP オブジェクトで、それぞれ次の 3 つの数値属性があります。

```
StockNo    NUMBER
Price      NUMBER
TaxRate    NUMBER
```

Oracle では、各属性に列が割り当てられ、CREATE TABLE 文によって、StockNo 列に主キー制約が設定されます。また、主キーが行オブジェクトの識別子として使用されるよう指定されます。

オブジェクト表 PurchaseOrder_objtab

次の文は、PURCHASEORDER_OBJTYP オブジェクトのオブジェクト表を定義します。

```
CREATE TABLE PurchaseOrder_objtab OF PurchaseOrder_objtyp ( /* Line 1 */
  PRIMARY KEY (PONo), /* Line 2 */
  FOREIGN KEY (Cust_ref) REFERENCES Customer_objtab) /* Line 3 */
  OBJECT ID PRIMARY KEY /* Line 4 */
  NESTED TABLE LineItemList_ntab STORE AS PoLine_ntab ( /* Line 5 */
    (PRIMARY KEY (NESTED_TABLE_ID, LineItemNo)) /* Line 6 */
    ORGANIZATION INDEX COMPRESS) /* Line 7 */
  RETURN AS LOCATOR /* Line 8 */
/
```

前述の CREATE TABLE 文では、PurchaseOrder_objtab オブジェクト表が作成されます。各行の重要点は、次のとおりです。

行 1:

```
CREATE TABLE PurchaseOrder_objtab OF PurchaseOrder_objtyp (
```

この行は、表の各行が PURCHASEORDER_OBJTYP オブジェクトであることを示します。
PURCHASEORDER_OBJTYP オブジェクトの属性は、次のとおりです。

PONo	NUMBER
Cust_ref	REF Customer_objtyp
OrderDate	DATE
ShipDate	DATE
LineItemList_ntab	LineItemList_ntabtyp
ShipToAddr obj	Address_objtyp

図 8-9 PurchaseOrder_objtab 表のオブジェクト・リレーショナル表現

PONO	CUST_REF	ORDERDATE	SHIPDATE	LINEITEMLIST_NTAB	SHIPTOADDR_OBJ
数値 NUMBER	参照 CUSTOMER_ OBJTYP	日付 DATE	日付 DATE	NESTED TABLE LINEITEMLIST_ NTABTYP	オブジェクト型 ADDRESS_ OBJTYP
PK	FK				

メンバー関数getPONOによりNUMBERが戻される
 メンバー関数SumLineItemsにより NUMBERが戻される

表の
行への
参照

CUSTNO	CUSTNAME	ADDRESS_OBJ	PHONELIST_VAR
数値 NUMBER	テキスト VARCHAR2(200)	オブジェクト型 ADDRESS_OBJTYP	VARRAY PHONELIST_VARTYP
PK			

行 2:

PRIMARY KEY (PONo),

この行は、PONo 属性が表の主キーであることを指定します。

行 3:

```
FOREIGN KEY (Cust_ref) REFERENCES Customer_objtab)
```

この行は、Cust_ref 列についての参照制約を指定します。この参照制約は、リレーショナル表に対して指定されたものと似ています。制約がない場合、REF 列によって任意の行オブジェクトを参照できます。ただし、この場合、Cust_ref REF では Customer_objtab オブジェクト表の行オブジェクトしか参照できません。

行 4:

```
OBJECT ID PRIMARY KEY
```

この行は、PurchaseOrder_objtab オブジェクト表の主キーが行の OID として使用されることを示します。

行 5 ～ 8:

```
NESTED TABLE LineItemList_ntab STORE AS PoLine_ntab (  
    (PRIMARY KEY (NESTED_TABLE_ID, LineItemNo))  
    ORGANIZATION INDEX COMPRESS)  
RETURN AS LOCATOR
```

これらの行は、NESTED TABLE の LineItemList_ntab 列の記憶域仕様およびプロパティを示します。NESTED TABLE の行は、別の記憶表に格納されます。この記憶表に対して、ユーザーが直接問合せを実行することはできませんが、メンテナンスの目的で DDL 文で参照できます。記憶表には、NESTED_TABLE_ID という非表示列があり、対応する親行の行と一致します。特定の親に属する NESTED TABLE のすべての要素に、同一の NESTED_TABLE_ID 値があります。たとえば、PurchaseOrder_objtab の任意の行にある NESTED TABLE のすべての要素は、NESTED_TABLE_ID の値が同じです。PurchaseOrder_objtab の別の行に属する NESTED TABLE の要素には、異なる NESTED_TABLE_ID の値があります。

前述の CREATE TABLE の例で、行 5 は、NESTED TABLE LineItemList_ntab の行が PoLine_ntab という名前の（記憶表として参照される）別の表に格納されることを示します。STORE AS 句によって、記憶表に対する制約および記憶域仕様を指定することもできます。この例で、行 7 は、記憶表が索引構成表（IOT）であることを示します。一般に、1 つの IOT に NESTED TABLE 行を格納すると効果的です。これによって、同じ親に属している行がクラスタ化されます。IOT で COMPRESS を指定すると、記憶領域が節約されます。これは、COMPRESS を指定しないと、IOT のキーの NESTED_TABLE_ID 部分が、親の行オブジェクトのすべての行に対して繰り返されるためです。ただし、COMPRESS を指定すると、NESTED_TABLE_ID は親の行オブジェクトの各行に 1 回のみ格納されます。

CREATE TABLE 文では、REF への SCOPE FOR 制約は指定できません。したがって、Stock_ref がオブジェクト表 Stock_objtab のみを参照できるように指定するには、PoLine_ntab 記憶表に次の ALTER TABLE 文を発行します。

```
ALTER TABLE PoLine_ntab
  ADD (SCOPE FOR (Stock_ref) IS stock_objtab) ;
```

この文は、親表でなく記憶表に対して実行されることに注意してください。

参照： NESTED TABLE を IOT として構成して NESTED TABLE の圧縮を指定する利点、および NESTED TABLE の格納の詳細は、5-14 ページの「[NESTED TABLE の記憶域](#)」を参照してください。

行 6 で、NESTED_TABLE_ID および LineItemNo 属性を記憶表の主キーとして指定することには、目的が 2 つあります。第 1 に、これは IOT のキーとして機能します。第 2 に、親表の各行内にある NESTED TABLE の列 (LineItemNo) の一意性を施行します。キーに LineItemNo 列を組み込むことによって、この文では、各発注書の LineItemNo 列の値は、確実に一意になります。

行 8 は、NESTED TABLE LineItemList_ntab が取り出される際に、ロケータ形式で戻されることを示します。LOCATOR を指定しないとデフォルトは VALUE となり、NESTED TABLE に対してロケータのみが戻されるかわりに、NESTED TABLE 全体が戻されることを示します。NESTED TABLE コレクションに多くの要素が含まれる場合、含まれる行オブジェクトまたは列が選択されるたびに NESTED TABLE 全体を戻すのは、あまり効率的ではありません。

NESTED TABLE のロケータが戻されるように指定すると、実際のコレクション値に対するロケータのみがクライアントに戻されます。アプリケーションは、OCICollIsLocator または UTL_COLL.IS_LOCATOR をコールすることで、フェッチされた NESTED TABLE がロケータにあるか、値形式であるかがわかります。ロケータが戻されていることがわかると、アプリケーションはロケータを使用して問い合わせ、NESTED TABLE 内の必要な行要素のサブセットのみをフェッチできます。このような NESTED TABLE の行の、ロケータベースの取出しは、元の文のスナップショットに基づいており、NESTED TABLE の値またはコピー・セマンティクスを保存します。

つまり、NESTED TABLE の行要素のサブセットをフェッチするためにロケータが使用される場合、NESTED TABLE のスナップショットは、ロケータが最初に取り出された時点の NESTED TABLE を反映しています。

8-21 ページの「[メソッドの定義](#)」で説明した、PurchaseOrder_objttyp の sumLineItems メソッドの実装を考えてみます。この実装では、NESTED TABLE LineItemList_ntab が VALUE として戻されることを想定しています。大きい NESTED TABLE をより効率的に処理し、PurchaseOrder_objttyp の NESTED TABLE がロケータとして戻されることを利用するには、sumLineItems メソッドを次のように書き直す必要があります。

```
CREATE OR REPLACE TYPE BODY PurchaseOrder_objttyp AS

    MAP MEMBER FUNCTION getPONo RETURN NUMBER is
    BEGIN
        RETURN PONo;
    END;

    MEMBER FUNCTION sumLineItems RETURN NUMBER IS
        i            INTEGER;
        StockVal     StockItem_objttyp;
        Total        NUMBER := 0;

    BEGIN
        IF (UTL_COLL.IS_LOCATOR(LineItemList_ntab)) -- check for locator
        THEN
            SELECT SUM(L.Quantity * L.Stock_ref.Price) INTO Total
            FROM   TABLE(CAST(LineItemList_ntab AS LineItemList_ntabtyp)) L;
        ELSE
            FOR i in 1..SELF.LineItemList_ntab.COUNT LOOP
                UTL_REF.SELECT_OBJECT(LineItemList_ntab(i).Stock_ref,StockVal);
                Total := Total + SELF.LineItemList_ntab(i).Quantity *
                                StockVal.Price;
            END LOOP;
        END IF;
        RETURN Total;
    END;
END;
/
```

書き直された sumLineItems メソッドでは、UTL_COLL.IS_LOCATOR 関数を使用して、NESTED TABLE の属性 LineItemList_ntab がロケータとして戻されているかどうかをチェックされます。条件が TRUE と評価された場合、TABLE 式を使用して、NESTED TABLE のロケータが問い合わせられます。

注意： 現在このような TABLE 式では、CAST 式が必要です。これは、SQL コンパイル・エンジンが問合せを正常にコンパイルできるように、SQL コンパイル・エンジンにコレクション属性（またはパラメータ、変数）の実際の型を知らせることを目的としています。

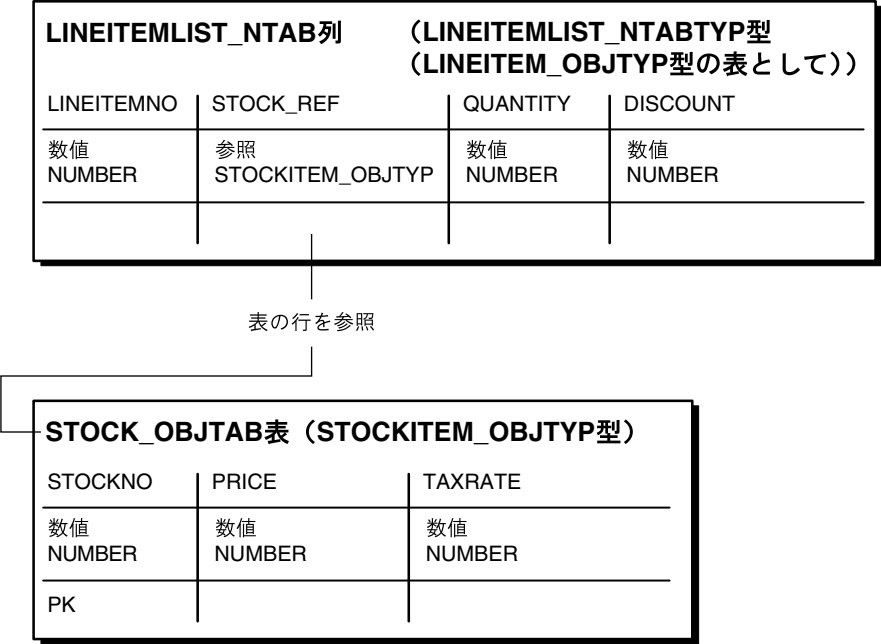
NESTED TABLE のロケータを問い合わせることで、発注書の膨大な明細項目リストの処理がより効率的になります。LineItemList_ntab で繰返しを行う前述のコードは、NESTED TABLE が VALUE として戻される場合を処理するために保持されています。

表が作成された後、次の ALTER TABLE 文が発行されます。

```
ALTER TABLE PoLine_ntab
  ADD (SCOPE FOR (Stock_ref) IS stock_objtab);
```

この文は、NESTED TABLE の Stock_ref 列の有効範囲が Stock_objtab に限られることを指定します。これは、この列に格納される値が、Stock_objtab の行オブジェクトへの参照である必要があることを示します。SCOPE 制約は参照されるオブジェクトに無関係という点で、参照制約とは異なります。たとえば、Stock_objtab 内の参照される行オブジェクトは、NESTED TABLE の Stock_ref 列で参照されている場合でも削除できます。このような削除を行うと、NESTED TABLE の対応する参照は、DANGLING REF になります。

図 8-10 NESTED TABLE LineItemList_ntab のオブジェクト・リレーショナル表現



Oracle では、記憶表に対して参照制約は指定できません。この場合は、REF 列に対して SCOPE 句を指定します。一般に、REF 列に有効範囲または参照制約を指定すると、いくつか利点があります。

- Oracle によって、行オブジェクトの一意識別子のみが列の REF 値として格納されるようになるため、記憶領域が節約されます。
- 記憶表の REF 列に索引が作成できるようになります。
- Oracle によって、これらの REF の参照解除を含む問合せが、参照表を含む結合として再作成できるようになります。

ここまで、発注書アプリケーションのすべての表が設定されました。次の項では、これらの表の操作方法を示します。

図 8-11 PurchaseOrder_objtab 表のオブジェクト・リレーショナル表現

PURCHASEORDER_OBJTAB表 (PURCHASEORDER_OBJTYP型)					
PONO	CUST_REF	ORDERDATE	SHIPDATE	LINEITEMLIST_NTAB	SHIPTOADDR_OBJ
数値 NUMBER	参照 CUSTOMER_ OBJTYP	日付 DATE	日付 DATE	NESTED TABLE LINEITEMLIST_ NTABTYP	オブジェクト型 ADDRESS_ OBJTYP
PK	FK				

定義された型の
列オブジェクト

列オブジェクトSHIPTOADDR_OBJ (ADDR_OBJTYP型)			
STREET	CITY	STATE	ZIP
テキスト VARCHAR2(200)	テキスト VARCHAR2(200)	テキスト CHAR(2)	数値 VARCHAR2(20)

値の挿入

この項では、事前にリレーショナル表で行ったように、オブジェクト表に同じデータを挿入する方法を説明します。値のどれだけが、実際に、オブジェクト型のコンストラクタへのコールであるかに注意してください。

Stock_objtab

```
INSERT INTO Stock_objtab VALUES (1004, 6750.00, 2) ;
INSERT INTO Stock_objtab VALUES (1011, 4500.23, 2) ;
INSERT INTO Stock_objtab VALUES (1534, 2234.00, 2) ;
INSERT INTO Stock_objtab VALUES (1535, 3456.23, 2) ;
```

Customer_objtab

```
INSERT INTO Customer_objtab
VALUES (
    1, 'Jean Nance',
    Address_objtyp('2 Avocet Drive', 'Redwood Shores', 'CA', '95054'),
    PhoneList_vartyp('415-555-1212')
) ;

INSERT INTO Customer_objtab
VALUES (
    2, 'John Nike',
    Address_objtyp('323 College Drive', 'Edison', 'NJ', '08820'),
    PhoneList_vartyp('609-555-1212', '201-555-1212')
) ;
```

PurchaseOrder_objtab

```
INSERT INTO PurchaseOrder_objtab
SELECT 1001, REF(C),
       SYSDATE, '10-MAY-1999',
       LineItemList_ntabtyp(),
       NULL
FROM   Customer_objtab C
WHERE  C.CustNo = 1 ;
```

前述の文で、次の属性を持つ PURCHASEORDER_OBJTYP オブジェクトが構成されます。

PONo	1001
Cust_ref	REF to customer number 1
OrderDate	SYSDATE
ShipDate	10-MAY-1999
LineItemList_ntab	an empty LineItem_ntabtyp
ShipToAddr_obj	NULL

この文は、問合せを使用して、CustNo 値が 1 である Customer_objtab オブジェクト表の行オブジェクトに対する REF を構成します。

次の文は、TABLE 式を使用して、NESTED TABLE (PONo 値が 1001 である PurchaseOrder_objtab 表の行オブジェクトの LineItemList_ntab 列にある NESTED TABLE) を挿入先として識別します。

注意： Oracle リリース 8.0 では、NESTED TABLE を識別するために、フラット化した副問合せまたは THE (副問合せ) 式がサポートされています。リリース 8.1 では、この構成より、次の TABLE 式が使用されています。

```
INSERT INTO TABLE (  
  SELECT P.LineItemList_ntab  
  FROM   PurchaseOrder_objtab P  
  WHERE  P.PONo = 1001  
)  
SELECT  01, REF(S), 12, 0  
FROM    Stock_objtab S  
WHERE   S.StockNo = 1534 ;
```

前述の文で、TABLE 式で識別された NESTED TABLE に明細項目が挿入されます。挿入される明細項目には、StockNo 値が 1534 であるオブジェクト表 Stock_objtab の行オブジェクトに対する REF があります。

次の文は、前述の文と同じパターンです。

```
INSERT INTO PurchaseOrder_objtab  
  SELECT  2001, REF(C),  
          SYSDATE, '20-MAY-1997',  
          LineItemList_ntabtyp(),  
          Address_objtyp('55 Madison Ave', 'Madison', 'WI', '53715')  
  FROM    Customer_objtab C  
  WHERE   C.CustNo = 2 ;  
  
INSERT INTO TABLE (  
  SELECT P.LineItemList_ntab  
  FROM   PurchaseOrder_objtab P  
  WHERE  P.PONo = 1001  
)  
SELECT  02, REF(S), 10, 10  
FROM    Stock_objtab S  
WHERE   S.StockNo = 1535 ;
```

```

INSERT INTO TABLE (
  SELECT  P.LineItemList_ntab
    FROM   PurchaseOrder_objtab P
   WHERE  P.PONo = 2001
)
SELECT  10, REF(S), 1, 0
  FROM   Stock_objtab S
  WHERE  S.StockNo = 1004 ;

INSERT INTO TABLE (
  SELECT  P.LineItemList_ntab
    FROM   PurchaseOrder_objtab P
   WHERE  P.PONo = 2001
)
VALUES (11, (SELECT REF(S)
  FROM   Stock_objtab S
  WHERE  S.StockNo = 1011), 2, 1) ;

```

問合せ

次の問合せ文では、比較メソッドが暗黙的にコールされます。ここでは、PURCHASEORDER_OBJTYP 型のオブジェクトを、その型の比較メソッドを使用して Oracle でどのように順序付けるかを示します。

```

SELECT  p.PONo
  FROM   PurchaseOrder_objtab p
 ORDER BY VALUE(p) ;

```

選択内のそれぞれの PURCHASEORDER_OBJTYP オブジェクトに対し、マップ・メソッド getPONo が起動します。このメソッドは、オブジェクトの PONo 属性を戻すため、選択は発注書番号が昇順に並んだリストを生成します。

次の問合せは、リレーショナル・モデルで実行された問合せに対応しています。

発注書 1001 の顧客および明細項目データ

```

SELECT  Deref(p.Cust_ref), p.ShipToAddr_obj, p.PONo,
        p.OrderDate, LineItemList_ntab
  FROM   PurchaseOrder_objtab p
  WHERE  p.PONo = 1001 ;

```

各発注書の合計金額

```
SELECT  p.PONo, p.sumLineItems()
FROM    PurchaseOrder_objtab p ;
```

在庫品目 1004 の発注書および明細項目

```
SELECT  po.PONo, po.Cust_ref.CustNo,
        CURSOR (
            SELECT  *
            FROM    TABLE (po.LineItemList_ntab) L
            WHERE   L.Stock_ref.StockNo = 1004
        )
FROM    PurchaseOrder_objtab po ;
```

前述の間合せでは、NESTED TABLE から選択された一連の LineItem_obj オブジェクトに対するネスト・カーソルが戻されます。アプリケーションは、ネスト・カーソルからフェッチして、個々の LineItem_obj オブジェクトを取得できます。前述の間合せは、出力結果に関して、ネスト・セットのネストを解除することによって、次のように表すこともできます。

```
SELECT  po.PONo, po.Cust_ref.CustNo, L.*
FROM    PurchaseOrder_objtab po, TABLE (po.LineItemList_ntab) L
WHERE   L.Stock_ref.StockNo = 1004 ;
```

前述の間合せは、結果セットをフラット化したフォーム（または第 1 正規形）として戻します。このような間合せは、ODBC などのリレーショナル・ツールおよび API から Oracle コレクション列にアクセスする場合に役立ちます。前述のネストを解除する例では、LineItemList_ntab 行を持つ PurchaseOrder_objtab オブジェクト表の行のみが戻されます。対応する LineItemList_ntab に行があるかどうかにかかわらず、PurchaseOrder_objtab 表のすべての行をフェッチするには、(+) 演算子が必要です。

```
SELECT  po.PONo, po.Cust_ref.CustNo, L.*
FROM    PurchaseOrder_objtab po, TABLE (po.LineItemList_ntab) (+) L
WHERE   L.Stock_ref.StockNo = 1004 ;
```

発注書のすべての明細項目に渡る値引きの平均

この要求には、すべての PurchaseOrder_objtab 行のすべての NESTED TABLE LineItemList_ntab にある行の間合せが必要です。さらに、ネストの解除が必要です。

```
SELECT  AVG(L.DISCOUNT)
FROM    PurchaseOrder_objtab po, TABLE (po.LineItemList_ntab) L ;
```

削除

次の例は、リレーショナルの例で必要とされた2つの削除文と同じ結果になります（8-10ページの「[リレーショナル・モデルでのデータの削除](#)」を参照）。この場合、Oracle では、削除される発注書に属するすべての明細項目が自動的に削除されます。リレーショナルの場合には、もう1文の命令を発行する必要があります。

発注書 1001 の削除

```
DELETE
FROM   PurchaseOrder_objtab
WHERE  PONo = 1001 ;
```

Java を使用したオブジェクトの操作

発注書の例で定義済のスキーマを使用して、Java Database Connectivity (JDBC) API を使用するか、または埋込み SQL を SQLJ で使用して、データベース内のオブジェクトを操作できます。この例では JDBC を使用しますが、両方のコーディングは似ており、オブジェクト指向プログラムにはどちらの手法も使用できます。

まず、データベース内のオブジェクト型をどれほど忠実に Java クラスにマップするかを決定する必要があります。次の項では、2つの選択肢について説明します。

oracle.sql.* クラス（緩い型指定）の使用

この例では、次のことを行います。

- データ型およびオブジェクトを、顧客表から、ORACLE.SQL パッケージで提供されている事前定義されたオブジェクト・クラスにマップします。
- オブジェクト型ごとに1つのクラスを作成するのではなく、すべてのアプリケーション論理で1つのクラスのみを作成します。
- オブジェクトを汎用型 `oracle.sql.STRUCT` として、コレクション型を `oracle.sql.ARRAY` として、およびスカラー値を `oracle.sql.NUMBER` などの事前定義型として扱います。
- `STRUCT` クラスから属性を動的に取り出し、それを単一の配列に引き込みます。最初の属性は数値であることなど、クラスの内部詳細を認識し、配列の各要素を権限タイプのオブジェクトにキャストする必要があります。

特定のクラスの定義が同じであれば、この手法によって、そのクラスと簡単に相互作用できるプロシージャ Java プログラムを作成できます。

```
import java.sql.*;
import oracle.sql.*;

public class DefaultMappingDemo
{
    public static void main(String[] args)
    {
        System.out.println("*** JAVA OBJECTS DEMO ***");

        try {
            DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

            Connection conn = DriverManager.getConnection
                ("jdbc:oracle:thin:@stpc90.us.oracle.com:1521:stpc90",
                 "scott", "tiger") ;

            Statement stmt = conn.createStatement();

            ResultSet rs = stmt.executeQuery
                ("select value(c) from CUSTOMER_TAB c order by value(c)");

            while (rs.next ())
            {

                // retrieve the STRUCT
                oracle.sql.STRUCT cust_struct = (STRUCT)rs.getObject(1);

                oracle.sql.Datum cust_attrs[] = cust_struct.getOracleAttributes();
                oracle.sql.NUMBER num = (NUMBER)cust_attrs[0];

                // string attribute in Object
                oracle.sql.CHAR name = (CHAR) cust_attrs[1];

                // embedded object
                oracle.sql.STRUCT address_struct = (STRUCT)cust_attrs[2];
                oracle.sql.Datum address_attrs[] = address_struct.getOracleAttributes();
                oracle.sql.CHAR street = (CHAR) address_attrs[0];
                oracle.sql.CHAR city   = (CHAR) address_attrs[1];
                oracle.sql.CHAR state  = (CHAR) address_attrs[2];
                oracle.sql.CHAR zip    = (CHAR) address_attrs[3];

                System.out.println("Number: " + num.stringValue() + ", Name: " + name +
```

```

        ", Address: " + street + ", " + city + ", " + state +
        ", " + zip);

//embedded array
oracle.sql.ARRAY phone_list = (ARRAY)cust_attrs[3];
}
rs.close();
stmt.close();
}
catch (SQLException exn)
{
    System.out.println("SQLException: "+exn);
}
}
}
}

```

強力な型指定（SQLData または CustomDatum）の使用

複数の Java クラスを使用してデータベース・オブジェクト型をモデル化する場合は、強力に型指定されたモデルを作成できます。すべてのクラスは、いくつかの共通動作を実装して、基礎となるデータベース処理を実行します。ここで、クラスを JDBC 2.0 API（SQLData インタフェース）上でモデル化するか、または Oracle の API（CustomDatum インタフェース）上でモデル化するかを選択します。

SQLData インタフェースは業界標準に準拠しており、異なるデータベース・システム間の移植が可能です。CustomDatum インタフェースは JDBC から導出されますが、さらに拡張があります。REF、コレクション型および JDBC ではサポートされていない、他のオブジェクト指向機能をカプセル化できます。

異なるオプションで JPublisher を使用して、どちらのインターフェースにもラッパー・クラスを生成できます。

JPublisher を使用したラッパー・クラスの生成

強力に型指定されたモデルでは、スキーマ内の各オブジェクト型に Java クラスが必要です。これらのクラスを取得する最も簡単な方法は、Oracle にデータベースから型定義を読み込ませ、Java コードを生成させることです。このためには、JPublisher Tool への入力として、次のファイルを使用できます。

```

SQL SCOTT."ADDRESS_OBJTYP" AS JAddress
SQL SCOTT."CUSTOMER_OBJTYP" AS JCustomerInfo
SQL SCOTT."LINEITEMLIST_NTABTYP" AS JLineItemList

```

```
SQL SCOTT."LINEITEM_OBJTYP" AS JLineItem
SQL SCOTT."PHONELIST_VARTYP" AS JPhoneList
SQL SCOTT."PURCHASEORDER_OBJTYP" AS JPurchaseOrder
SQL SCOTT."STOCKITEM_OBJTYP" AS JStockInfo
```

ラッパー・クラスの使用法

すべてのラッパー・クラスは、次の JCustomer に似ています。JCustomer は、データベース・スキーマ内の CUSTOMER_INFO_T 型に対応しています。例では、JCustomer の属性の 1 つが JAddress オブジェクトであるため、JAddress ラッパー・クラスも必要です。

正規の Java I/O ストリームを使用して、この型のインスタンスを読み込みまたは書き込みできます。追加のメンバー関数を実装するには、JCustomer をサブクラス化して、そのクラスが再生されるたびにコードが保存されるようにします。

```
import java.sql.*;
import oracle.jdbc2.*;
import oracle.sql.*;

public class JCustomer implements SQLData
{
    private String sql_type;
    public int custNo;
    public String custName;
    public JAddress address;
    public Array phoneList;

    public String getSQLTypeName() throws SQLException { return sql_type; }

    public void readSQL (SQLInput stream, String typeName) throws SQLException
    {
        sql_type = typeName;
        custNo   = stream.readInt();
        custName = stream.readString();
        address  = (JAddress) stream.readObject();
        phoneList= stream.readArray();
    }
    public void writeSQL (SQLOutput stream) throws SQLException
    {
        stream.writeInt(custNo);
        stream.writeString(custName);
        stream.writeObject(address);
    }
}
```

```

        stream.writeArray(phoneList);
    }
}

```

この例では、データベース型のメソッド関数から導出されているメンバー関数を示します。そのようなメンバー関数をコールすると、データベース・サーバーとの間でオブジェクト・データが移動されるため、通信量が増加します。また、入力および出力パラメータの特定の規則に従う必要があります。この問題の詳細は、『Oracle8i SQLJ 開発者ガイドおよびリファレンス』（オブジェクトおよびコレクション）および『Oracle8i JDBC 開発者ガイドおよびリファレンス』（Oracle オブジェクト型での作業）を参照してください。

SQLData インタフェースを使用したサンプル・プログラム

次のプログラムでは、次のことを行います。

- JPublisher によって生成された JCustomer および JAddress のクラスを使用して作業します。JAddress は、JCustomer の属性の 1 つの型であるため必要です。
- どの Java クラスがどのオブジェクト型に対応するかを、データベースに認識させます。たとえば、JCustomer は CUSTOMER_INFO_T に対応します。その情報によって、Oracle では、例にある INSERT などの SQL 文にオブジェクト・データを代入できるようになります。
- 一度、結果セットから JCustomer にオブジェクトをキャストすると、他の Java クラスの場合と同様に、そのデータおよび関数にアクセスできます。
- Java のオブジェクトを更新してから、データベースを更新する SQL 文に Java オブジェクトを代入します。

```

import java.sql.*;
import oracle.sql.*;
import oracle.jdbc.driver.*;
import oracle.jdbc2.*;
import java.util.*;

public class SQLDataDemo
{
    public static void main(String[] args) throws Exception, SQLException
    {
        System.out.println("*** JAVA OBJECTS DEMO : USING SQLData INTERFACE ***");

        DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
    }
}

```

```
OracleConnection conn = (OracleConnection) DriverManager.getConnection
    ("jdbc:oracle:thin:@stpc90.us.oracle.com:1521:stpc90",
     "scott", "tiger");

Statement stmt = conn.createStatement();

//put an entry in the typemap
try
{
    Dictionary map = conn.getTypeMap();

    map.put("CUSTOMER_INFO_T", Class.forName("JCustomer"));
    map.put("ADDRESS_T", Class.forName("JAddress"));
}
catch (ClassNotFoundException exn) { }

ResultSet rs = stmt.executeQuery("select VALUE (p) from CUSTOMER_TAB p");

while (rs.next())
{
    //retrieve the object using standard API
    JCustomer jc = (JCustomer) rs.getObject(1);
    int custNo = jc.custNo;
    String custName = jc.custName;

    jc.custName = "Geoff Lee";
    PreparedStatement pstmt = conn.prepareStatement
        ("INSERT INTO CUSTOMER_TAB VALUES (?)");
    pstmt.setObject(1, jc);
    pstmt.executeUpdate();

    rs.close();
    stmt.close();
}
}
```

Oracle Objects for OLE でのオブジェクトの操作

Windows システム上では、Oracle Objects For OLE (OO4O) を使用して、Visual Basic、または Excel などの COM プロトコルをサポートする他の環境で、オブジェクト指向データベース・プログラムを作成できます。

次のすべての例は、データベースに接続する類似したヘッダー・セクションで始まり、それぞれはオブジェクト・データ上の異なる操作の実行方法を示します。

データの選択

次に、SELECT 操作を実行するボタンのイベント・ハンドラを示します。

- データベースから行集合を取得します。各行には、いくつかのリレーショナル列およびオブジェクトであるいくつかの列が含まれています。
- CUSTREF 列の名前を使用して、オブジェクトであるその値を取り出します。
- ここで、ドット表記法を使用して、オブジェクトの属性にアクセスできます。汎用オブジェクト型 `OraObject` として変数を定義します。`OraObject` が実際のオブジェクトでインスタンス化されると、対応するオブジェクト型のプロパティをとります。

```
Private Sub obj_select_Click()
    Dim OO4OSession As OraSession
    Dim InvDB As OraDatabase
    Dim PurchaseOrder As OraDynaset
    Dim CustomerInfo As OraRef
    Dim LineItemsList As OraCollection
    Dim LineItem As OraObject
    Dim ShipToAddr As OraObject
    Dim StockInfo As OraRef
    Dim CustomerAddr As OraObject

    'Create the OraSession Object.
    Set OO4OSession = CreateObject("OracleInProcServer.XOraSession")

    'Create the OraDatabase Object by opening a connection to Oracle.
    Set InvDB = OO4OSession.OpenDatabase("exampledb", "scott/tiger", 0&)

    'Select from purchase_tab
    Set PurchaseOrder = InvDB.CreateDynaset("select * from purchase_tab", 0&)

    'Get the custref attribute from PurchaseOrder
    Set CustomerInfo = PurchaseOrder.Fields("custref").Value
```

```
' Accessing attributes CustomerInfo object

'Display custno,custname,phonelist attributes of CustomerInfo
MsgBox CustomerInfo.custno
MsgBox CustomerInfo.custname

'Get address and phonelist attributes of CustomerInfo
Set CustomerAddr = CustomerInfo.Address

'Display all the attributes of CustomerAddr
MsgBox CustomerAddr.Street
MsgBox CustomerAddr.State
MsgBox CustomerAddr.Zip

' Accessing elements of LineItemsList Object

'Get line_item_list attribute from PurchaseOrder
Set LineItemsList = PurchaseOrder.Fields("line_item_list").Value

'Get LineItem object element from LineItemList collection
Set LineItem = LineItemsList(1)

'Display lineitemno,quantity,discount attributes
MsgBox LineItem.lineitemno
MsgBox LineItem.quantity
MsgBox LineItem.discount

'Access stockref attribute of LineItem
Set StockInfo = LineItem.Stockref

'Display stockno,cost,tax_code of StockInfo
MsgBox StockInfo.stockno
MsgBox StockInfo.cost
MsgBox StockInfo.tax_code

End Sub
```

データの挿入

次に、データベースから行集合を取り出して、新しい行を追加するプログラムを示します。

- 適切なオブジェクト型のオブジェクトをいくつか作成します。

- オブジェクトにサンプル値を移入します。
- 発注書表に新しい行を作成し、その列に値を指定します。オブジェクトではない列は、直接設定できます。オブジェクトである列は、VALUE フィールドを使用して設定する必要があります。

```

Dim OO4OSession As OraSession
Dim InvDB As OraDatabase
Dim PurchaseOrder As OraDynaset
Dim CustomerInfo As OraRef
Dim LineItemsList As OraCollection
Dim LineItem As OraObject
Dim ShipToAddr As OraObject
Dim StockInfo As OraRef
Dim CustomerAddr As OraObject

'Create the OraSession Object.
Set OO4OSession = CreateObject("OracleInProcServer.XOraSession")

'Create the OraDatabase Object by opening a connection to Oracle.
Set InvDB = OO4OSession.OpenDatabase("exampledb", "scott/tiger", 0&)

'Select from purchase_tab
Set PurchaseOrder = InvDB.CreateDynaset("select * from purchase_tab", 0&)

' Step 1 - Creating CustomerInfo ref object

'select a ref from customer_tab for custono 2
Set CustomerDyn = InvDB.CreateDynaset("select REF(C) from customer_tab c
where c.custno = 2", 0&)

'get the CustomerInfo ref object
Set CustomerInfo = CustomerDyn.Fields(0).Value

' Step 2 - Creating LineItemsList object

' Create a new line_items_list object
Set LineItemsList = InvDB.CreateOraObject("line_item_list_t")

' Create a new line_items object
Set LineItem = InvDB.CreateOraObject("line_item_t")

'set attributes of LineItem object
LineItem.lineitemno = 2
LineItem.quantity = 15

```

```
LineItem.discount = 30
LineItem.Stockref = Null

'set the LineItem to first element of LineItemList
LineItemsList(1) = LineItem

' Step 3 - Creating ShipToAddr object

' create a shiptoaddr object
Set ShipToAddr = InvDB.CreateOraObject("address_t")

'set the attributes of ShipToAddr Object
ShipToAddr.city = "Belmont"
ShipToAddr.Street = "Continental way"
ShipToAddr.Zip = "94002"
ShipToAddr.State = "CA"

' Start the AddNew operation on PurchaseOrder dynaset

PurchaseOrder.AddNew

PurchaseOrder.Fields("pono").Value = 1002
PurchaseOrder.Fields("orderdate").Value = "5/15/99"
PurchaseOrder.Fields("shipdate").Value = "6/15/99"

'set the custref field to CustomerInfo object created in step1
PurchaseOrder.Fields("custref").Value = CustomerInfo

'set the line_item_list field to LineItemslist object created in step2
PurchaseOrder.Fields("line_item_list").Value = LineItemsList

'set the shiptoaddr field to ShipToAddr object created in step3
PurchaseOrder.Fields("shiptoaddr").Value = ShipToAddr

' Call the update method on Purchaseorder Dynaset which inserts a new row
' in purchase_tab table

PurchaseOrder.Update
```

データの更新

次に、データベースから行をいくつか取り出して、特定のデータを更新するプログラムを示します。

- 単一行を戻す問合せを使用して、発注書を選択します。
- 個々のデータ項目を取得して、その他の表および元の発注書から操作します。
- 更新のために発注書の行をロックし、新しい値を入れます。

```

Dim OO4OSession As OraSession
Dim InvDB As OraDatabase
Dim PurchaseOrder As OraDynaset
Dim CustomerInfo As OraRef
Dim LineItemsList As OraCollection
Dim LineItem As OraObject
Dim ShipToAddr As OraObject
Dim StockInfo As OraRef
Dim CustomerAddr As OraObject

'Create the OraSession Object.
Set OO4OSession = CreateObject("OracleInProcServer.XOraSession")

'Create the OraDatabase Object by opening a connection to Oracle.
Set InvDB = OO4OSession.OpenDatabase("exampledb", "scott/tiger", 0&)

'Select from purchase_tab for pono 1002
Set PurchaseOrder = InvDB.CreateDynaset("select * from purchase_tab where
pono = 1002", 0&)

'Create a StockInfo from stock_tab for stockno 1535
Set StockDyn = InvDB.CreateDynaset("select REF(s) from stock_tab s where
s.stockno = 1535", 0&)
Set StockInfo = StockDyn.Fields(0).Value

'Get line_item_list attribute from PurchaseOrder
Set LineItemsList = PurchaseOrder.Fields("line_item_list").Value

'Get LineItem object element from LineItemsList collection
Set LineItem = LineItemsList(1)

'Start the edit operation on PurchaseOrder dynaset
PurchaseOrder.Edit

' Set the StockInfo object created in Step1 to stockref attribute
' of LineItem
LineItem.Stockref = StockInfo
PurchaseOrder.Update

```

メソッド関数のコール

次に、発注書を取り出してメンバー関数 TOTAL VALUE をコールし、発注書の一部である明細項目のコストを合計するプログラムを示します。

- 発注書表から 1 行を選択します。結果がオブジェクトとして戻るように、VALUE を選択していることに注意してください。
- 発注書オブジェクト（結果行の 0（ゼロ）列目）へのポインタを取得します。このポインタは、後で PL/SQL ストアド・プロシージャに渡され、Java メソッドまたは C++ メソッドの SELF ポインタをシミュレートします。
- 暗黙的な自己パラメータおよびメソッド関数の戻り値に対応するパラメータのリストを作成します。各リストに、バインド変数、値、モードおよび型を指定します。
- メソッド関数に対応するストアド・プロシージャをコールして、結果を TOTALVALUE バインド変数に格納します。
- 結果を使用するには、パラメータ・リストから戻り値を取り出します。

```
Dim OO4OSession As OraSession
Dim InvDB As OraDatabase
Dim PurchaseOrderObj As OraDynaset

'Create the OraSession Object.
Set OO4OSession = CreateObject("OracleInProcServer.XOraSession")

'Create the OraDatabase Object by opening a connection to Oracle.
Set InvDB = OO4OSession.OpenDatabase("exampledb", "scott/tiger", 0&)

'Select from purchase_tab
Set PurchaseOrderDyn = InvDB.CreateDynaset("select VALUE(p) from
purchase_tab p where p.pono = 1001", 0&)

'Get the PurchaseOrderObj
Set PurchaseOrderObj = PurchaseOrderDyn.Fields(0).Value

'Create a OraParameter object for purchase_order_t object and set it to
PurchaseOrder
InvDB.Parameters.Add "PURCHASEORDER", PurchaseOrderObj, ORAPARM_BOTH,
ORATYPE_OBJECT, "PURCHASE_ORDER_T"

'Create a parameter for total_value return
InvDB.Parameters.Add "TOTALVALUE", "", ORAPARM_OUTPUT

'Execute a member method
```

```
InvDB.ExecuteSQL ("BEGIN :TOTALVALUE :=  
PURCHASE_ORDER_T.TOTAL_VALUE(:PURCHASEORDER); END;")
```

```
'Display the totalvalue  
MsgBox InvDB.Parameters("TOTALVALUE").Value
```

索引

A

Active Server Page, 3-9
ActiveX, 3-9
ADMIN OPTION
 EXECUTE ANY TYPE, 2-11
ALTER ANY TYPE 権限, 2-10
 「権限」を参照
ASP, 3-9

C

COMPRESS 句
 NESTED TABLE, 5-16
CONNECT ロール
 ユーザー定義型, 2-11
CREATE ANY TYPE 権限, 2-10
 「権限」を参照
CREATE INDEX 文
 オブジェクト型, 2-5
CREATE TABLE 文
 例
 NESTED TABLE, 1-10
 オブジェクト表, 1-5, 1-10, 2-4, 2-7
 列オブジェクト, 1-11, 2-7
CREATE TRIGGER 文
 例
 オブジェクト表, 2-5
CREATE TYPE 権限, 2-10
 「権限」を参照
CREATE TYPE コマンド
 NESTED TABLE, 1-9, 1-11
CREATE TYPE 文
 NESTED TABLE, 2-3
 VARRAY, 1-9, 8-15

オブジェクト型, 1-10, 2-2, 2-3, 2-7, 8-14
オブジェクト・ビュー, 4-3
不完全型, 2-14
CREATE VIEW 文
 例
 オブジェクト・ビュー, 4-4

D

DBA ロール
 ユーザー定義型, 2-11
DROP ANY TYPE 権限, 2-11
 「権限」を参照
DROP TYPE 文
 FORCE オプション, 2-15

E

Excel, 3-9
EXECUTE ANY TYPE 権限, 2-11
 「権限」を参照
EXECUTE 権限
 「権限」を参照
 ユーザー定義型, 2-11
EXECUTE 権限の GRANT オプション, 2-11
EXECUTE ユーザー定義型, 2-11

F

FAQ
 Oracle オブジェクト, 7-1
 Oracle オブジェクトに関する, 7-1
FORCE オプション
 オブジェクト型依存性, 2-15

I

INSTEAD OF トリガー
 NESTED TABLE, 4-11

J

Java
 Oracle JDBC および Oracle オブジェクト, 3-12
 Oracle SQLJ および Oracle オブジェクト, 3-12
 Oracle オブジェクト, 3-12

JDBC
 「Oracle JDBC」を参照

N

NESTED TABLE, 1-9, 5-14
 COMPRESS 句, 5-16
 DML 操作, 5-19
 INSTEAD OF トリガー, 4-11
 VARRAY との比較, 8-15, 8-18
 一意性, 8-30
 記憶域, 5-14, 8-29
 索引, 2-4
 索引構成表における, 5-15
 索引の作成, 5-17
 問合せ, 8-18
 ビューにおける更新, 4-11
 ロケータとして戻す, 5-18, 8-30
NESTED TABLE を戻す, 8-30
NESTED_TABLE_ID, 5-17, 8-29
NULL
 アトミック, 2-2
 オブジェクト型, 2-2

O

OCI
 OCIObjectFlush, 4-4
 OCIObjectPin, 4-4
 Oracle オブジェクト
 プログラムの作成, 3-5
 新しいオブジェクトの作成, 6-4
 オブジェクト・キャッシュ, 3-4, 6-11
 オブジェクトのフラッシュ, 6-9
 オブジェクト操作の初期化, 6-4
 オブジェクトの確保および確保解除, 6-6

オブジェクトの更新, 6-5
オブジェクトの削除, 6-5
結合アクセス, 3-3
ナビゲーション・アクセス, 3-4
複合オブジェクト検索 (COR), 6-9
ロック・オプション, 6-8

OID

「オブジェクト識別子」を参照

Oracle Call Interface
 オブジェクト・キャッシュ・サイズの制御, 6-5

Oracle JDBC
 Oracle オブジェクト・データのアクセス, 3-12

Oracle Objects for OLE
 OraCollection インタフェース, 3-11
 OraObject インタフェース, 3-10
 OraRef インタフェース, 3-10

Oracle SQLJ
 JPublisher, 3-13
 Oracle オブジェクトのサポート, 3-12
 カスタム Java クラスの作成, 3-13

Oracle オブジェクト
 「オブジェクト・リレーショナル・モデル」を参照

OraCollection インタフェース, 3-11
OraObject インタフェース, 3-10
OraRef インタフェース, 3-10
OTT, 3-8

P

pkREF, 6-2
PL/SQL
 オブジェクト・ビュー, 4-4
 バインド変数
 ユーザー定義型, 3-2
 ユーザー定義データ型, 3-2

Pro*C/C++
 Oracle タイプと C 型の間の変換, 3-7
 結合アクセス, 3-6
 ナビゲーション・アクセス, 3-7
 ユーザー定義データ型, 3-2

R

REF, 1-6
 SCOPED, 1-7, 5-9, 6-2
 WITH ROWID オプション, 5-11
 暗黙的な参照解除, 8-22

オブジェクト識別子, 8-26
オブジェクト識別子から作成, 6-2
オブジェクト・ビューの行のための, 4-3
確保, 2-13, 4-4
記憶域, 5-9
サイズ, 6-2
索引, 2-4, 5-10
参照解除, 1-7, 8-22
参照先がない, 1-7
制約, 5-9
相互依存型, 2-14
表別名の使用, 2-7
RESOURCE ロール
 ユーザー定義型, 2-11
REVOKE コマンド
 オブジェクト型および依存性, 2-15
REVOKE 文
 FORCE オプション, 2-15

S

SCOPE FOR 制約, 8-30, 8-32
SCOPED REF, 1-7, 6-2
SQL
 ユーザー定義データ型, 3-1
 OCI, 3-2
 埋込み SQL, 3-6
SQLJ
 「Oracle SQLJ」を参照
STORE AS 句, 8-29

T

TABLE 構文, 5-12

V

VARRAY, 1-8
 NESTED TABLE との比較, 8-15, 8-18
 アクセス, 5-14
 記憶域, 5-13
 更新, 5-14
 問合せ, 5-14
 「配列」、「コレクション」を参照
Visual Basic, 3-9

W

WITH OBJECT IDENTIFIER 句, 4-4

あ

アトミック NULL, 2-2
暗黙的な参照解除, 8-22

い

依存性
 オブジェクト型定義, 2-14, 2-15
インポート・ユーティリティ
 ユーザー定義型, 2-15

う

内側獲得, 2-7

え

エクスポート・ユーティリティ
 ユーザー定義型, 2-15

お

オーダー・メソッド, 1-5, 5-7, 8-17, 8-23
オブジェクト
 オブジェクト参照, 4-9
 行オブジェクトおよびオブジェクト識別子, 4-6
 コレクション・オブジェクト, 4-6
 列の中の, 4-4
オブジェクト型, 1-2
 オブジェクト型トランスレータ, 3-8
 キャッシュでのロック, 3-3
 行オブジェクト, 1-6
 コンストラクタ・メソッド, 1-4, 6-2
 実行者権限, 5-29
 相互依存, 2-14
 属性, 1-3
 発注書の例, 1-10
 比較メソッド, 1-4, 8-20
 表別名の使用, 2-7
 不完全, 2-14, 8-14
 メソッド, 1-3, 8-21
 PL/SQL, 3-2

- メソッド・コール, 2-8
- 列オブジェクト, 1-6
- 索引, 2-4
- 列オブジェクトおよび行オブジェクト, 5-1
- オブジェクト型トランスレータ (OTT), 3-8
- オブジェクト型のコンパイル, 2-14
- オブジェクト・キャッシュ
 - OCI, 3-2
 - Pro*C, 3-6
 - オブジェクトのフラッシュ, 6-9
 - オブジェクト・ビュー, 4-4
 - 権限, 2-13
- オブジェクト識別子, 8-26
 - REF, 5-8
 - WITH OBJECT IDENTIFIER 句, 4-4
 - オブジェクト型に対する, 6-2
 - 記憶域, 5-7
 - 主キー・ベースの, 5-7
- オブジェクト・ビュー, 4-1 ~ 4-18
 - INSTEAD OF トリガーによる更新, 4-11
 - NESTED TABLE, 4-11
 - 定義, 4-3
 - 利点, 4-2
- オブジェクト表, 1-5, 5-7, 8-23
 - 値の削除, 8-39
 - 値の挿入, 8-34
 - 仮想のオブジェクト表, 4-2
 - 行オブジェクト, 1-6
 - 索引, 2-4
 - 制約, 2-3
 - 問合せ, 8-37
 - トリガー, 2-5
- オブジェクト表に対する DELETE 権限, 2-12, 2-13
- オブジェクト表に対する INSERT 権限, 2-12, 2-13
- オブジェクト表に対する SELECT 権限, 2-12, 2-13
- オブジェクト表に対する UPDATE 権限, 2-12, 2-13
- オブジェクト・リレーショナル・モデル, 8-1
 - 新しいオブジェクト形式, 5-31
 - 埋込みオブジェクト, 8-26
 - オブジェクトの比較, 5-7
 - オブジェクト表での実装, 8-14
 - 型の発展, 5-38
 - 継承, 1-10
 - 制約, 5-37
 - 設計上の考慮点, 5-1
 - パーティション化, 6-14
 - プログラム環境, 3-1

- メソッド, 1-3
- リレーショナル・モデルの制限事項, 8-11
- レプリケーション, 5-31

か

- 外部キー
 - 多対1のE-Rを表す, 8-7
- 獲得回避規則, 2-7
- 型
 - 「データ型」、「オブジェクト型」を参照
- 型の発展, 5-38
- カッコ、メソッド・コールでの使用, 2-8

き

- キー
 - 外部キー, 8-7
- 記憶域
 - NESTED TABLE, 6-3
 - REF, 6-2
 - オブジェクト表, 6-1
- キャッシュ
 - オブジェクト・キャッシュ, 2-13, 3-2, 3-6
 - オブジェクト・ビュー, 4-4
- 行
 - 行オブジェクト, 1-6
- 行オブジェクト, 1-6
 - 記憶域, 5-7

け

- 継承, 1-10
 - スーパータイプを含むサブタイプ, 5-33
 - すべてのサブタイプを含むスーパータイプ, 5-35
 - デュアル・サブタイプ / スーパータイプの参照, 5-36
- 権限
 - システム
 - ユーザー定義型, 2-10
 - ユーザー定義型
 - ADMIN OPTION を含む EXECUTE ANY TYPE, 2-11
 - ALTER ANY TYPE, 2-10
 - CREATE ANY TYPE, 2-10
 - CREATE TYPE, 2-10
 - DELETE, 2-12, 2-13

- DROP ANY TYPE, 2-11
- EXECUTE, 2-11
- EXECUTE ANY TYPE, 2-11
- GRANT オプションを含む EXECUTE, 2-11
- INSERT, 2-12, 2-13
- SELECT, 2-12, 2-13
- UPDATE, 2-12, 2-13
- オブジェクト表の列レベル, 2-14
- 確保時にチェック, 2-13
- システム権限, 2-10
- ロールによって取得, 2-11
- 使用方法, 2-11, 2-15

権限付与

- ユーザー定義型の実行, 2-11

こ

更新

- オブジェクト・ビュー, 4-11

コレクション

- NESTED TABLE, 1-9

- 問合せ, 5-12

- ネスト, 5-20

- 変数配列 (VARRAY), 1-8

コレクション型の COUNT 属性, 8-22

コンストラクタ・メソッド, 1-4, 6-2

- ～のリテラル起動, 2-3

さ

索引

- REF, 2-4

- ユーザー定義型, 2-4

索引構成表

- NESTED TABLE を格納, 5-15

参照解除, 1-7, 8-22

- 暗黙的, 8-22

参照先がない REF, 1-7

し

システム権限

- ADMIN OPTION, 2-11

- 「権限」を参照

- ユーザー定義型, 2-10

実行者権限

- オブジェクト型, 5-29

主キー・ベース REF, 6-2

す

スキーマ

- ユーザー定義データ型, 1-2, 3-2

スキーマ名

- 列名を修飾, 2-7

せ

制約, 8-25

- Oracle オブジェクトでの, 5-37

- REF, 5-9

- SCOPE FOR 制約, 8-30, 8-32

- オブジェクト表, 2-3

そ

属性

- オブジェクト型, 1-3

- リーフ・レベル, 6-1

- リーフ・レベル・スカラー, 6-1

た

ダンプ・ファイル

- エクスポートおよびインポート, 2-15

て

データ型

- NESTED TABLE, 1-9

- オブジェクト型, 1-2

- 配列型, 1-8

データベース管理者 (DBA)

- DBA ロール, 2-11

デフォルト値

- ユーザー定義型, 2-3

と

問合せ

- VARRAY, 5-14

- セット・メンバーシップ, 5-18

- ネストを解除する, 5-12

ドット表記法, 1-3

トリガー
 INSTEAD OF トリガー
 オブジェクト・ビューと、4-11
 ユーザー定義型、2-5

ね

ネストを解除する問合せ、5-12

は

パーティション化
 Oracle オブジェクトを持つ表、6-14
配列、8-26
 VARRAY のサイズ、1-8
 変数 (VARRAY)、1-8
バインド変数
 ユーザー定義型、3-2
パラレル問合せ
 Oracle オブジェクトに対する制限事項、5-38

ひ

比較メソッド、1-4、8-20
表
 NESTED TABLE、1-9
 索引、2-4
 オブジェクト
 「オブジェクト表」を参照
 オブジェクト表、1-5
 仮想、4-2
 索引、2-4
 制約、2-3
 トリガー、2-5
 列名を修飾、2-6、2-7

ふ

ファイル
 エクスポート / インポート・ダンプ・ファイル、
 2-15
ファンクション索引
 タイプ・メソッドの戻り値、5-30
不完全オブジェクト型、2-14、8-14
複合オブジェクト検索
 Oracle Call Interface 用、6-9
プラグマ RESTRICT_REFERENCES、8-20

プログラム環境
 Oracle オブジェクト、3-1 ~ 3-12

へ

変数
 オブジェクト変数、4-4
 バインド変数
 ユーザー定義型、3-2

ま

マップ・メソッド、1-5、5-7、8-17

め

メソッド、1-3
 オーダー、5-7、8-17、8-23
 オブジェクト型、1-3、8-21
 PL/SQL、3-2
 オーダー・メソッド、1-5
 空のカッコの使用、2-8
 起動の自己参照的スタイル、1-4
 権限の実行、2-11
 コンストラクタ・メソッド、6-2
 マップ・メソッド、1-5
 言語の選択、5-26
 コンストラクタ・メソッド、1-4
 リテラル起動、2-3
 静的、5-28
 比較、8-20
 比較メソッド、1-4
 ファンクション索引、5-30
 マップ、5-7、8-17
メソッド起動の自己参照的スタイル、1-4

ゆ

ユーザー定義データ型、2-1 ~ 2-16
 エクスポートおよびインポート、2-15
 オブジェクト型、1-2
 表別名の使用、2-7
 「オブジェクト・リレーショナル・モデル」を参照
記憶域、6-1
権限、2-10
コレクション
 NESTED TABLE、1-9

変数配列 (VARRAY), 1-8
不完全な型, 2-14

り

リーフ・レベル・スカラー属性, 6-1
リーフ・レベル属性, 6-1
リテラル起動
 コンストラクタ・メソッド, 2-3

れ

列
 問合せでの修飾, 2-6
列オブジェクト, 1-6
 索引, 2-4
列名
 問合せでの修飾, 2-7
列オブジェクト
 行オブジェクト, 5-1

ろ

ロール
 CONNECT ロール, 2-11
 DBA ロール, 2-11
 RESOURCE ロール, 2-11
ロケータ, 8-30
 NESTED TABLE を戻す, 5-18
ロック
 オブジェクト・レベル・ロック, 3-3

