Oracle9*i*

XML データベース開発者ガイド - Oracle XML DB

リリース 2（9.2）

2003 年 2 月

部品番号 : J06289-02

ORACLE®

Oracle9*i* XML データベース開発者ガイド - Oracle XML DB, リリース 2（9.2）

部品番号 : J06289-02

原本名 ：Oracle9*i* XML Database Developer's Guide - Oracle XML DB, Release 2 (9.2)

原本部品番号：A96620-02

原本著者：Shelley Higgins

グラフィック・デザイナ：Valarie Moore

原本協力者：Nipun Agarwal、Abhay Agrawal、Omar Alonso、Sandeepan Banerjee、Mark Bauer、Ravinder Booreddy、Yuen Chan、Sivasankaran Chandrasekar、Vincent Chao、Mark Drake、Fei Ge、Wenyun He、Thuvan Hoang、Sam Idicula、Neema Jalali、Bhushan Khaladkar、Viswanathan Krishnamurthy、Muralidhar Krishnaprasad、Wesley Lin、Annie Liu、Anand Manikutty、Jack Melnick、Nicolas Montoya、Steve Muench、Ravi Murthy、Eric Paapanen、Syam Pannala、John Russell、Eric Sedlar、Vipul Shah、Cathy Shea、Tarvinder Singh、Simon Slack、Muralidhar Subramanian、Asha Tarachandani、Randy Urbano、Priya Vennapusa、James Warner、Harish Akali、Deanna Bradshaw、Paul Brandenstein、Lisa Eldridge、Geoff Lee、Susan Kotsovolos、Sonia Kumar、Roza Leyderman、Diana Lorentz、Yasuhiro Matsuda、Bhagat Nainani、Visar Nimani、Sunitha Patel、Denis Raphaely、Rebecca Reitmeyer、Ronen Wolf

目次

2 Oracle XML DB を使用する前に

3 Oracle XML DB の使用

第 II 部　Oracle XML DB からの XML データの格納および取出し

4　XMLType の使用

5 XMLType の構造化されたマッピング

6 XMLType データの変換および検証

7 Oracle Text を使用した XML データの検索

第 III 部　XMLType API を使用した XML データの操作

8　PL/SQL API for XMLType

9 Java API for XMLType

第 IV 部　既存データの XML 表示

10　データベースからの XML データの生成

11 XMLType ビュー

12 URL を介したデータの作成およびアクセス

第 V 部　Oracle XML DB Repository: フォルダリング、セキュリティおよびプロトコル

13　Oracle XML DB Foldering

14　Oracle XML DB Versioning

15 RESOURCE_VIEW および PATH_VIEW

16 Oracle XML DB Resource API for PL/SQL（DBMS_XDB）

17 Oracle XML DB Resource API for Java

18 Oracle XML DB リソースのセキュリティ

19 FTP、HTTP および WebDAV プロトコルの使用

20 Java での Oracle XML DB アプリケーションの作成

第 VI 部　Oracle XML DB をサポートする Oracle のツール製品

21　Oracle Enterprise Manager を使用した Oracle XML DB の管理

22　Oracle XML DB への XML データのロード

23 XMLType 表のインポートおよびエクスポート

第 VII 部 Advanced Queueing を使用した XML データの交換

24 Advanced Queuing（AQ）および Oracle Streams を使用した XML データの変換

第 VIII 部　Oracle XML DB の事例

25　Oracle XML DB の事例 : Web サービスによる XML 文書の取出しおよび表示

26　Oracle XML DB Basic Demo

A Oracle XML DB のインストールおよび構成

B XML Schema の手引き

C XPath および Namespace の手引き

D XSLT の手引き

E Java DOM API for XMLType および Resource API for Java: クイック・リファレンス

F Oracle XML DB の XMLType API、PL/SQL API および Resource PL/SQL API: クイック・リファレンス

G 設定スクリプトの例および Oracle XML DB によって提供される XML Schema

用語集

索引

はじめに

このマニュアルでは、Oracle XML DB（Oracle9*i* XML Database）について説明します。Oracle XML DB を使用して、データベースで eXtensible Markup Language（XML）データの格納、生成、操作、管理および問合せを実行する方法を説明します。

XMLType フレームワークおよび Oracle XML DB Repository という Oracle XML DB の主要部分の概要を説明した後、Oracle XML DB アプリケーションを計画するときに考慮する設計基準について簡単に説明します。Oracle XML DB の使用方法および使用可能な場所の例も示します。

次に、Oracle XML DB を使用した XML データの格納および取出し方法、XMLType データを操作するための API、および既存の XML データの表示、生成、変換および検索方法も説明します。また、Oracle XML リポジトリの使用方法（バージョニングやセキュリティを含む）、プロトコル、Structured Query Language（SQL）、PL/SQL または Java を使用してリポジトリのリソースにアクセスおよび操作する方法、および Oracle Enterprise Manager を使用して Oracle XML DB アプリケーションを管理する方法についても説明します。さらに、XML メッセージ機能およびアドバンスト・キューイング（AQ）による XMLType のサポートについても説明します。

ここでは、次の項目について説明します。

対象読者

このマニュアルは、Oracle9*i* データベース上で XML アプリケーションを構築する開発者を対象としています。

前提知識

このマニュアルを使用するには、XML、XML Schema、XPath および eXtensible Stylesheet Language（XSL）を理解していることが理想です。

このマニュアルに示されている多くの例は、SQL、Java または PL/SQL で記述されています。そのため、このうちの 1 つ以上の言語の実用経験があることを前提としています。

このマニュアルの構成

このマニュアルは、次のように構成されています。

第 I 部「Oracle XML DB の概要」

第 I 部では、XMLType やリポジトリを含む、Oracle XML DB のコンポーネントおよびアーキテクチャの概要を説明します。いくつかの基本的な設計問題について説明し、Oracle XML DB の使用方法および使用可能な場所を総合的に示す一連の例も示します。

第 1 章「Oracle XML DB の概要」

この章では、Oracle XML DB のコンポーネントおよびアーキテクチャの概要を説明します。Oracle XML DB を使用するメリット、主な機能、サポートされている標準および Oracle XML DB の実行要件について説明します。このマニュアルを通して使用される Oracle XML DB 関連の用語をリストします。

第 2 章「Oracle XML DB を使用する前に」

この章では、Oracle XML DB のインストール方法、互換性と移行、および基本的なアプリケーション計画の問題について説明します。

第 3 章「Oracle XML DB の使用」

この章では、Oracle XML DB の使用可能な場所および使用方法を説明します。Oracle XML DB を使用した XML データの格納、アクセス、更新および検証の例を示します。

第 II 部「XML データの格納および取出し」

第 II 部では、Oracle9*i* データベースのネイティブ XMLType API を使用して、XML データを格納、取出し、検証および変換する方法を説明します。

第 4 章「XMLType の使用」

この章では、XMLType 表の作成方法、および XML Schema に基づかない XMLType 表および列に対する XML データの操作および問合せ方法を説明します。

第 5 章「XMLType の構造化されたマッピング」

この章では、SQL と XML 間での Oracle XML DB マッピングの使用方法を説明します。また、XML Schema の登録方法の概要、および Oracle XML DB のデフォルト・マッピングの使用方法、独自のマッピングの指定方法も説明します。さらに、Oracle XML DB での Ordered Collections in Tables（OCT）の使用方法も説明します。

第 6 章「XMLType データの変換および検証」

この章では、SQL 関数を使用して、データベースに格納され、データベースから取出しまたは生成された XML データを変換する方法を説明します。また、SQL 関数を使用して、データベースに入力されている XML データを検証する方法も説明します。

第 7 章「Oracle Text を使用した XML データの検索」

この章では、DBUriType（または Oracle XML DB の URIType）列に Oracle Text 索引を作成する方法、および Oracle Text の CONTAINS() 関数および XMLType の existsNode() 関数を使用して XML データを検索する方法を説明します。XML データの XPath 問合せに CTXXPATH 索引を使用する方法も説明します。

第 III 部「XMLType API を使用した XML データの操作」

第 III 部では、PL/SQL および Java 用の XMLType API の概要および使用方法を説明します。

第 8 章「PL/SQL API for XMLType」

この章では、PL/SQL DOM API for XMLType、PL/SQL Parser API for XMLType および PL/SQL XSLT Processor API for XMLType の概要を説明します。この章には、例およびコール順序図が含まれます。

第 9 章「Java API for XMLType」

この章では、Java（JDBC）API for XMLType の使用方法を説明します。この章には、例およびコール順序図が含まれます。

第 IV 部「既存データの XML 表示」

第 IV 部では、既存のデータを XML として表示する方法の概要を説明します。

第 10 章「データベースからの XML データの生成」

この章では、XML を生成するための SQLX 関数、Oracle SQLX 拡張関数および SQL 関数について説明します。SQLX 関数には、`XMLElement()` および `XMLForest()` が含まれます。Oracle SQLX 拡張関数には、`XMLColAttValue()` が含まれます。SQL 関数には、`SYS_XMLGEN()`、`XMLSEQUENCE()` および `SYS_XMLAGG()` が含まれます。この章では、`DBMS_XMLGEN`、XSQL Pages パブリッシング・フレームワークおよび XML SQL Utility（XSU）を使用して、データベースに格納されたデータから XML データを生成する方法も説明します。

第 11 章「XMLType ビュー」

この章では、XML 生成関数、オブジェクト型または `XMLType` 表の変換に基づいて `XMLType` ビューを作成する方法を説明します。また、`XMLType` ビューで XML データを操作する方法も説明します。

第 12 章「URL を介したデータの作成およびアクセス」

この章では、Oracle9*i* データベースでの URI および URL の処理方法を説明します。`URIType` および関連するサブタイプ（`DBUriType`、`HTTPUriType` および `XDBUriType`）を使用してデータベースのデータを作成したり、URL を介してデータにアクセスする方法を説明します。また、`URIFactory` パッケージを使用して `URIType` のインスタンスを作成する方法、SQL 関数 `SYS_DBURIGEN()` の使用方法、および `DBUri` サーブレットを使用して URL をデータベース問合せに変換する方法も説明します。

第 V 部「Oracle XML DB Repository: フォルダリング、セキュリティおよびプロトコル」

第 V 部では、Oracle XML DB Repository の概要、その背景の概念、およびバージョニング、アクセス制御リスト（ACL）・セキュリティ、プロトコル・サーバーおよび関連する様々な Oracle XML DB Resource API の使用方法を説明します。

第 13 章「Oracle XML DB Foldering」

この章では、階層的な索引付けおよびフォルダリングについて説明します。Oracle XML DB Resource View API、バージョニング、Oracle XML DB Resource API for PL/SQL、Oracle XML DB Resource API for Java などの様々な Oracle XML DB Repository のコンポーネントの概要を説明します。

第 14 章「Oracle XML DB Versioning」

この章では、バージョン管理された Oracle XML DB リソース（VCR）を作成する方法および VCR にアクセスおよび更新する方法を説明します。

第 15 章「RESOURCE_VIEW および PATH_VIEW」

この章では、Oracle XML DB Resource View API を使用して、Oracle XML DB Repository に格納されたデータに SQL を介してアクセスする方法を説明します。また、その他の Oracle XML DB Resource API の機能との比較も示します。

第 16 章「Oracle XML DB Resource API for PL/SQL（DBMS_XDB）」

この章では、Oracle XML DB Resource API for PL/SQL について説明します。

第 17 章「Oracle XML DB Resource API for Java」

この章では、Oracle XML DB Resource API for Java の概要、およびそれを使用して Oracle XML DB Repository のデータにアクセスする方法を説明します。

第 18 章「Oracle XML DB リソースのセキュリティ」

この章では、Oracle XML DB リソースおよび ACL セキュリティの使用方法、ACL の共有方法および ACL 情報の取出し方法を説明します。

第 19 章「FTP、HTTP および WebDAV プロトコルの 使用」

この章では、Oracle XML DB Protocol Server の概要、および Oracle XML DB での FTP、HTTP および WebDAV の使用方法を説明します。

第 20 章「Java での Oracle XML DB アプリケーションの作成」

この章では、Oracle XML DB アプリケーションを Java で作成する方法を説明します。データベース内外で使用可能な Java API、Oracle XML DB HTTP Servlet の作成のヒント、構成ファイル `/xdbconfig.xml` 内でサーブレットを構成するために使用するパラメータ、および HTTP のリクエスト処理について説明します。

第 VI 部「Oracle XML DB の開発をサポートする Oracle のツール製品」

第 VI 部では、Oracle XML DB アプリケーションを構築および管理するためのツール製品について説明します。

第 21 章「Oracle Enterprise Manager を使用した Oracle XML DB の管理」

この章では、Oracle Enterprise Manager を使用して XML Schema を登録する方法、リソース、`XMLType` の表、ビューおよび列を作成する方法、ACL セキュリティを管理する方法、Oracle XML DB を構成する方法、およびファンクション索引を作成する方法を説明します。

第 22 章「Oracle XML DB への XML データのロード」

この章では、SQL*Loader を使用して `XMLType` データをロードする方法を説明します。

第 23 章「XMLType 表のインポートおよびエクスポート」

この章では、XMLType 表をロードするためのインポート / エクスポート・ユーティリティのサポートについて説明します。

第 VII 部「AQ を使用した XML データの交換」

第 VII 部では、XML および XMLType メッセージ機能で使用する Oracle Advanced Queuing（Oracle AQ）のサポートについて説明します。

第 24 章「Advanced Queuing（AQ）および Oracle Streams を使用した XML データの変換」

この章では、アドバンスト・キューイングを使用して XML データを交換する方法を説明します。Oracle Streams、Internet Data Access Presentation（iDAP）、AQ XML サーブレットを使用したメッセージのエンキューおよびデキュー方法、iDAP および AQ XML Schema の使用方法の概要を説明します。

第 VIII 部「Oracle XML DB の事例」

第 VIII 部では、2 つの XML DB ベースのアプリケーションについて説明します。

第 25 章「Oracle XML DB の事例 : Web サービスによる XML 文書の取出しおよび表示」

この章では、XML DB の Web サービスベースの発注書アプリケーションを作成するためのコール順序およびコードを示します。

第 26 章「Oracle XML DB Basic Demo」

この章では、XML DB を使用して発注書 XML 文書を格納、アクセスおよび操作する方法の多くの例を示します。

付録 A「Oracle XML DB のインストールおよび構成」

この付録では、Oracle XML DB のインストールおよび構成方法を説明します。

付録 B「XML Schema の手引き」

この付録では、World Wide Web Consortium（W3C）の XML Schema 勧告の概要を説明します。

付録 C「XPath および Namespace の手引き」

この付録では、W3C の XPath 勧告、Namespace 勧告および情報セットの概要を説明します。

付録 D「XSLT の手引き」

この付録では、W3C の XSL および XSLT 勧告の概要を説明します。

付録 E「Java DOM API for XMLType および Resource API for Java: クイック・リファレンス」

この付録では、Oracle XML DB の Java API に関するクイック・リファレンスを示します。

付録 F「Oracle XML DB の XMLType API、PL/SQL API および Resource PL/SQL API: クイック・リファレンス」
この付録では、Oracle XML DB の PL/SQL API に関するクイック・リファレンスを示します。

付録 G「設定スクリプトの例および Oracle XML DB によって提供される XML Schema」
この付録では、第 3 章の例で使用される設定スクリプトについて説明します。また、RESOURCE_VIEW および PATH_VIEW 構造について説明し、Oracle XML DB で提供されるサンプル・リソースの XML Schema をリストします。

用語集

関連文書

詳細は、次の Oracle ドキュメントを参照してください。

- 『Oracle9*i* データベース新機能』
  Oracle9*i* データベースと Oracle9*i* データベース Enterprise Edition の違い、および使用可能な機能とオプションについて説明します。また、Oracle9*i* データベース リリース 2（9.2）の新機能についても説明します。
- 『Oracle9*i* XML API リファレンス - XDK および Oracle XML DB』
- 『Oracle9*i* XML Developer's Kit ガイド - XDK』
- 『Oracle9*i* ケース・スタディ - XML アプリケーション』
  XML Developer's Kit（XDK）の例を示します。今回のリリースの Oracle XML DB の例は含まれません。
- 『Oracle9*i* データベース・エラー・メッセージ』
- 『Oracle Text アプリケーション開発者ガイド』
- 『Oracle Text リファレンス』
- 『Oracle9*i* データベース概要』
- 『Oracle9*i* Java Developer's Guide』
- 『Oracle9*i* アプリケーション開発者ガイド - 基礎編』
- 『Oracle9*i* アプリケーション開発者ガイド - アドバンスト・キューイング』
- 『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』

このマニュアルのいくつかの例で、Oracle のインストール時にデフォルトとしてインストールされるシード・データベースのサンプル・スキーマを使用しています。スキーマの作成方法および使用方法の詳細は、『Oracle9*i* サンプル・スキーマ』を参照してください。

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

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

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

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

表記規則

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

- 本文中の表記規則
- コード例中の表記規則

本文中の表記規則

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

規則	意味	例
太字	太字は、本文中で定義されている用語または用語集に記載されている用語（あるいはその両方）を示します。	この句を指定すると、**索引構成表**が作成されます。
固定幅フォントの大文字	固定幅フォントの大文字は、システムが提供する要素を示します。このような要素には、パラメータ、権限、データ型、Recovery Manager キーワード、SQL キーワード、SQL*Plus またはユーティリティ・コマンド、パッケージおよびメソッドが含まれます。また、システムが提供する列名、データベース・オブジェクト、データベース構造、ユーザー名およびロールも含まれます。	NUMBER 列に対してのみに、この句を指定できます。 BACKUP コマンドを使用して、データベースのバックアップを取ることができます。 USER_TABLES データ・ディクショナリ・ビュー内の TABLE_NAME 列を問い合せます。 DBMS_STATS.GENERATE_STATS プロシージャを使用します。

規則	意味	例
固定幅フォントの小文字	固定幅フォントの小文字は、実行可能ファイル、ファイル名、ディレクトリ名およびユーザーが提供する要素のサンプルを示します。このような要素には、コンピュータ名およびデータベース名、ネット・サービス名および接続識別子が含まれます。また、ユーザーが提供するデータベース・オブジェクトとデータベース構造と列名、パッケージとクラス、ユーザー名とロール、プログラム・ユニットおよびパラメータ値も含まれます。 **注意：**大文字と小文字を組み合せて使用するプログラム要素もあります。これらの要素は、記載されているとおり入力してください。	sqlplus と入力して、SQL*Plus をオープンします。 パスワードは、orapwd ファイルで指定します。 /disk1/oracle/dbs ディレクトリ内のデータ・ファイルおよび制御ファイルのバックアップを取ります。 hr.departments 表には、department_id、department_name および location_id 列があります。 QUERY_REWRITE_ENABLED 初期化パラメータを true に設定します。 oe ユーザーとして接続します。 JRepUtil クラスが次のメソッドを実装します。
固定幅フォントの小文字のイタリック	固定幅フォントの小文字のイタリックは、プレースホルダまたは変数を示します。	*parallel_clause* を指定できます。 U*old_release*.SQL を実行します。ここで、*old_release* とはアップグレード前にインストールしたリリースを示します。

コード例中の表記規則

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

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

次の表に、コード例で使用される表記規則を説明し、その使用例を示します。

規則	意味	例
[ ]	大カッコは、任意に選択する 1 つ以上の項目を囲みます。大カッコは、入力しないでください。	DECIMAL (*digits* [, *precision*])
{ }	中カッコは、2 つ以上の項目を囲み、そのうちの 1 つの項目は必須です。中カッコは、入力しないでください。	{ENABLE | DISABLE}
|	縦線は、大カッコまたは中カッコ内の 2 つ以上のオプションの選択項目を表します。オプションのうちの 1 つを入力します。縦線は、入力しないでください。	{ENABLE | DISABLE} [COMPRESS | NOCOMPRESS]

規則	意味	例
...	水平の省略記号は、次のいずれかを示します。	
	■ 例に直接関連しないコードの一部が省略されている。	`CREATE TABLE... AS subquery;`
	■ コードの一部を繰り返すことができる。	`SELECT col1, col2,... , coln FROM employees;`
. . .	垂直の省略記号は、例に直接関連しない複数の行が省略されていることを示します。	
その他の句読点	大カッコ、中カッコ、縦線および省略記号以外の句読点は、表示されているとおり入力する必要があります。	`acctbal NUMBER(11,2);` `acct CONSTANT NUMBER(4) := 3;`
イタリック体	イタリック体は、特定の値を指定する必要があるプレースホルダや変数を示します。	`CONNECT SYSTEM/system_password` `DB_NAME = database_name`

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

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

Oracle XML DB の新機能

ここでは、Oracle9*i* データベース リリース 2（9.2.0.2）および Oracle9*i* データベース リリース 2（9.2.0.1）の一機能としての Oracle XML DB に追加された新機能、拡張機能、API および製品統合について説明します。

Oracle XML DB: Oracle9*i* データベース リリース 2（9.2.0.2）：拡張機能

この項では、パッチ・リリースの Oracle9*i* データベース リリース 2（9.2.0.2）で提供された Oracle XML DB の拡張機能の概要を示します。

> **参照：** 製品に付属しているリリース・ノート関連文書を参照してください。

XML データのエクスポートおよびインポート

Oracle9*i* データベース リリース 2（9.2.0.2）では、拡張されたインポート / エクスポート・ユーティリティを使用して Oracle XML DB に XML データをロードできます。第 23 章「XMLType 表のインポートおよびエクスポート」を参照してください。

XMLAgg() SQLX 関数

今回のリリースでは、`XMLAgg()` で ORDER BY 句がサポートされています。10-17 ページの「XMLAgg() 関数」を参照してください。

updateXML() XMLType 関数

`updateXML()` の項に、より多くの総合的な例が含まれています。

グローバリゼーション・サポート：マルチバイト・キャラクタ

Oracle XML DB は、クライアントのキャラクタ・セットがデータベースのキャラクタ・セットと同じである場合、マルチバイト・キャラクタを処理できます。

更新された Oracle XML DB で提供される XML Schema

付録 G「設定スクリプトの例および Oracle XML DB によって提供される XML Schema」の終わりに、更新された Oracle XML DB で提供される 3 つの XML Schema（`XDBResource.xsd`、`acl.xsd` および `xdbconfig.xsd`）を示します。

リリース 2（9.2.0.1）からリリース 2（9.2.0.2）への移行

リリース 2（9.2.0.1）からリリース 2（9.2.0.2）への移行方法についての注意事項を追加しています。付録 A「Oracle XML DB のインストールおよび構成」を参照してください。

RESOURCE_VIEW の PATH 演算子

この項には、複数の相関が存在したり、pathname 引数が指定されている場合にパスを決定するための追加の例および説明を記載しています。15-9 ページの「PATH」を参照してください。

新しい構成パラメータ

新しいチューニング・パラメータ resource-view-cache-size が追加されています。今回のリリースでは、大規模な RESOURCE_VIEW を問い合せる場合、xdbconfig ファイル内の resource-view-cache-size パラメータをチューニングできます。15-15 ページの「高速問合せのための XML DB のチューニング」を参照してください。

HTTP/WebDAV パラメータ default-url-charset が追加されています。これは、受信 URL が UTF-8 またはリクエストの Content-Type フィールドの Charset パラメータに指定されたキャラクタ・セットにエンコードされない場合に、HTTP プロトコル・サーバーが想定するキャラクタ・セットです。19-6 ページの「HTTP または WebDAV 固有の構成パラメータ（サーブレット・パラメータを除く）」を参照してください。

PL/SQL DOM API for XMLType: 新しいメソッド

今回のリリースでは、いくつかの新しい DBMS_XMLDOM メソッドが PL/SQL DOM API for XMLType でサポートされます。第 8 章「PL/SQL API for XMLType」を参照してください。今回のリリースでは、いくつかのメソッドがサポートされないことに注意してください。

Java DOM API for XMLType: サポートされないメソッド

今回のリリースでは、XDBDocument、XDBNode および XDBDOMImplementation クラスのいくつかのメソッドがサポートされていません。第 9 章「Java API for XMLType」を参照してください。

Oracle Text の INPATH および HASPATH 演算子での XML 文書に対するハイライト表示のサポート

今回のリリースでは、INPATH または HASPATH 問合せ要素をハイライト表示させ、CTX_DOC.MARKUP または HIGHLIGHT プロシージャを使用して、Oracle Text 内の XML 文書をハイライト表示できます。7-49 ページの「INPATH/HASPATH テキスト演算子に対するハイライト表示のサポート」を参照してください。

Oracle XML DB の事例の記載

第 25 章「Oracle XML DB の事例 : Web サービスによる XML 文書の取出しおよび表示」および第 26 章「Oracle XML DB Basic Demo」を参照してください。

> **参照：** Oracle XML DB の最新の更新情報および注意事項については、http://otn.oracle.com/tech/xml/content.html を参照してください。

Oracle XML DB: Oracle9*i* データベース リリース 2（9.2.0.1）：XMLType の拡張

XMLType データ型は、Oracle9*i* データベースで最初に導入されました。このデータ型は、Oracle9*i* データベース リリース 2（9.2.0.1）で大幅に拡張されています。このリリースは、通常、リリース 2（9.2）と示されます。次の項で、これらの拡張について説明します。

> **参照：**『Oracle9*i* XML API リファレンス - XDK および Oracle XML DB』を参照してください。

XMLType 表

今回のリリースでは、XMLType データ型を使用して、XMLType 表を作成できます。これによって、オブジェクトと同様に列、表または表全体のいずれにも XML を格納できます。

XMLType コンストラクタ

今回のリリースでは、いくつかの XMLType コンストラクタが追加されています。createXML() 関数に加えて、ユーザー定義コンストラクタを使用して XMLType を構成することもできます。

W3C の XML Schema のサポート

今回のリリースでは、Oracle XML DB に XML Schema の拡張サポートが追加されています。次の操作を行うことができます。

- XML Schema に基づく XMLType のオブジェクト型を構成し、継続して XML Schema の検証が行えます。
- XML Schema に基づく XMLType 表を作成できます。この機能によって、適切な格納構造が自動的に作成され、XML Schema に基づく XML 文書の格納が最適化されます。SQL のデータ定義言語（DDL）とは異なり、この処理を行うためにすべての列のデータ型および定義を理解しておく必要はありません。
- DBMS_XMLSCHEMA パッケージを使用して注釈付きの XML Schema を登録し、記憶域および型定義を共有できます。登録された XML Schema はすべてのデータベース・ユーザー間で共有でき、インスタンス全体で共通の XML 文書の定義が可能になります。登録処理では、オプションでデフォルト表を作成できます。XML Schema の注釈を使用すると、SQL 型やデフォルト記憶表などの様々なオブジェクト型を指定できます。
- 受信 XML 文書を事前解析し、その文書をデフォルト記憶表に自動的に転送できます。これによって、FTP や WebDAV などのプロトコルが構造化された XML 文書を受け入れ、それらの文書をオブジェクト・リレーショナル表に格納できます。
- Oracle XML DB に XML 文書またはインスタンスが追加された場合、W3C の XML Schema に対してその XML 文書またはインスタンスを自動的に検証できます。

- XMLType に IsSchemaValid() メソッドを使用して、XML Schema に対して XML 文書および XMLType のインスタンスを明示的に検証できます。
- extractValue 演算子を使用して、データ型を認識する方法で XML 文書の一部を抽出できます。

SQLX 関数および Oracle 拡張関数

今回のリリースでは、既存のリレーショナル表およびオブジェクト・リレーショナル表から XML を生成するための SQLX 操作がサポートされています。これは、XML 関連仕様（SQL/XML）の ISO ANSI 草案（ISO/IEC 9075 Part 14 and ANSI）に基づいています。この草案では、データベース言語 SQL を XML と組み合せて使用する方法が定義されています。

たとえば、SQLX 標準機関で定義されている XMLElement()、XMLForest()、XMLConcat() および XMLAgg() 関数がサポートされています。Oracle XML DB では、XMLColAttVal()、XMLSequence()、SYS_XMLGEN()、SYS_XMLAGG() などの関数を使用する SQLX 操作も拡張されています。

抽出、条件確認および更新に対する W3C の XPath サポート

Oracle9*i* データベース リリース 1（9.0.1）では、XMLType オブジェクトに対して extract() および existsNode() 関数が提供されています。これらの関数によって、XML 文書に対して XPath を使用した問合せを実行できます。今回のリリースでは、次のサポートが追加されています。

- 名前空間の操作を実行する extract()、existsNode() および extractValue()
- 軸演算子を含む完全な XPath 関数セットに対する extract()、existsNode() および extractValue()
- XPath をロケータとして使用することによって XMLType の DOM の一部を置き換える updateXML() 関数（新規）

ToObject メソッド

ToObject メソッドを使用すると、コール側が XMLType オブジェクトを PL/SQL オブジェクト型に変換できます。

XMLType ビュー

今回のリリースでは、XMLType ビューがサポートされています。これらのビューを使用すると、データベース内のすべてのデータを XML として表示できます。XMLType ビューは、XML Schema に基づくビューにも基づかないビューにもできます。第 11 章「XMLType ビュー」を参照してください。

W3C の XSLT サポート

今回のリリースでは、新しい関数 `XMLTransform()` が導入され、これを使用するとメモリー内またはディスク上の XML 文書をデータベースにネイティブに変換できます。第 6 章「XMLType データの変換および検証」を参照してください。

XMLType に対する JDBC のサポート

Oracle XML DB では、データベース・クライアントが `XMLType` をバインドおよび定義できます。JDBC サポートには機能が豊富な `XMLType` クラスが含まれています。このクラスでは、（Thick JDBC 用に）ネイティブな XML 機能がサポートされています。第 9 章「Java API for XMLType」を参照してください。

C ベースの PL/SQL DOM、パーサーおよび XSLT API

今回のリリースでは、データベース・コードに統合されているネイティブな PL/SQL DOM、パーサーおよび XSLT API が含まれています。これらの PL/SQL API は、Oracle9*i* データベース リリース 1（9.0.1）以上で XDK for PL/SQL の一部として付属している Java ベースの PL/SQL API と互換性があります。第 8 章「PL/SQL API for XMLType」を参照してください。

Oracle XML DB: Oracle9*i* データベース リリース 2（9.2.0.1）：リポジトリ

今回のリリースでは、Oracle XML DB Repository によってデータベースにフォルダリングおよびセキュリティの拡張メカニズムが追加されています。Oracle XML DB Repository は、すべてのデータベース・データに対して、新しいファイル・システムのようなアクセスを提供する新機能です。リポジトリでは、次の操作を実行できます。

- リソースを含むファイル・システム（通常はファイルおよびフォルダ）としてデータベースおよびそのコンテンツを表示する。
- パス名ベースの SQL および Java API を介してリソースにアクセスし操作する。
- FTP、HTTP および WebDAV 用のネイティブな組込みプロトコル・サーバーを介してリソースにアクセスし操作する。
- Oracle XML DB リソースに対する ACL ベースのセキュリティを施行する。

Oracle XML DB Resource API（PL/SQL）: DBMS_XDB

`DBMS_XDB` パッケージでは、Oracle XML DB リソースにアクセスし、操作するメソッドが提供されます。第 16 章「Oracle XML DB Resource API for PL/SQL（DBMS_XDB）」を参照してください。

Oracle XML DB Resource View API（SQL）

Resource View は、パブリック XMLType ビューです。Resource View を使用すると、データベース・インスタンス内のすべてのリソースに対してパス名ベースの問合せを実行できます。このビューでは、パスベースの問合せを、リレーショナル表、リレーショナル・ビュー、オブジェクト・リレーショナル表およびオブジェクト・リレーショナル・ビューに対する問合せとマージできます。第 15 章「RESOURCE_VIEW および PATH_VIEW」を参照してください。

リリース 2（9.2.0.2）では、XDBconfig ファイルにチューニング・パラメータ resource-view-cache-size が含まれており、これによって高速問合せが可能になります。第 15 章「RESOURCE_VIEW および PATH_VIEW」の終わりを参照してください。

Oracle XML DB Versioning: DBMS_XDB_VERSION

DBMS_XDB_VERSION パッケージでは、Oracle XML DB リソースのバージョニングを実行するメソッドが提供されます。第 14 章「Oracle XML DB Versioning」を参照してください。

Oracle XML DB の ACL セキュリティ

DBMS_XDB パッケージには、ACL ベースのセキュリティを実装するメソッドが含まれています。それらのメソッドを使用すると、すべての XMLType オブジェクトに対する高パフォーマンスなアクセス制御リストを作成できます。第 18 章「Oracle XML DB リソースのセキュリティ」を参照してください。

Oracle XML DB Protocol Server

プロトコル・サーバーによって、FTP、HTTP および WebDAV を介してフォルダリングされた任意の XMLType 行にアクセスできます。XMLType は、すべてのファイル形式の任意のバイナリ・データを管理できます。第 19 章「FTP、HTTP および WebDAV プロトコルの 使用」を参照してください。

XDBUriType

今回のリリースでは、URIType に Oracle XML DB 内のパス名を表す新しいサブタイプ XDBUriType が含まれています。第 12 章「URL を介したデータの作成およびアクセス」を参照してください。

Oracle XML DB に対する Oracle ツール製品の拡張機能

Oracle Enterprise Manager

Oracle Enterprise Manager では、グラフィカル・インタフェースを使用して Oracle XML DB を管理および構成できます。第 21 章「Oracle Enterprise Manager を使用した Oracle XML DB の管理」を参照してください。

Oracle Text の拡張機能

今回のリリースでは、次の Oracle Text の拡張機能が提供されています。

- Oracle Text を使用して、Oracle9*i* データベース内で XMLType 型の列をネイティブに索引付けできます。
- CONTAINS() は、XPath 問合せにおける ora:contains として、また、existsNode() 関数の一部として使用する新しい関数です。
- CTXXPATH は、XPath 検索を高速化するために existsNode() とともに使用する新しい索引タイプです。

第 7 章「Oracle Text を使用した XML データの検索」を参照してください。

Oracle Advanced Queuing（AQ）のサポート

今回のリリースでは、アドバンスト・キューイング（AQ）の Internet Data Access Presentation（iDAP）が拡張されています。iDAP によって、インターネット上で AQ を簡単に使用できます。AQ XML サーブレットを使用して、HTTP および Simple Object Access Protocol（SOAP）を介して Oracle9*i* AQ にアクセスできます。

また、今回のリリースでは、iDAP は AQ 操作に Simple Object Access Protocol（SOAP）を実装しています。iDAP は、SOAP リクエストのボディで使用される XML メッセージ構造を定義します。

今回のリリースでは、XMLType を Oracle オブジェクト型の属性として埋め込まず、XMLType を AQ のペイロード型として使用できます。

参照：

- 第 24 章「Advanced Queuing（AQ）および Oracle Streams を使用した XML データの変換」を参照してください。
- 『Oracle9*i* アプリケーション開発者ガイド - アドバンスト・キューイング』も参照してください。

XMLType に対する Oracle XDK サポート

XDK for Java サポート

今回のリリースでは、XSQL Servlet および XML SQL Utility（XSU）for Java で XMLType がサポートされています。getClobVal() などの XMLType オブジェクトに対する多くのメソッドは、XSU for Java で使用できます。

XDK for PL/SQL サポート

今回のリリースでは、XML SQL Utility（XSU）for PL/SQL で `XMLType` がサポートされています。

参照：

- 10-51 ページの「XSQL Pages パブリッシング・フレームワークを使用した XML の生成」および 10-54 ページの「XML SQL Utility（XSU）を使用した XML の生成」を参照してください。
- 『Oracle9*i* XML Developer's Kit ガイド - XDK』も参照してください。

第I部

Oracle XML DB の概要

第 I 部では、Oracle XML DB の概要を説明します。第 I 部に含まれる章は、次のとおりです。

- 第 1 章「Oracle XML DB の概要」
- 第 2 章「Oracle XML DB を使用する前に」
- 第 3 章「Oracle XML DB の使用」

1

Oracle XML DB の概要

この章では、Oracle XML DB のメリット、機能、アーキテクチャなど、Oracle XML DB の概要を説明します。この章の内容は次のとおりです。

Oracle XML DB の概要

この章では、Oracle XML DB の概要を説明します。Oracle9*i* データベース上で XML アプリケーションを構築するために使用可能な機能について説明します。

XML は、本質的に自己記述型および動的拡張性という主な特長を持っています。これによって、様々なアプリケーション間でメッセージを転送するために必要な柔軟性が得られ、分散業務処理を疎結合できます。

また、XML は言語およびプラットフォームに依存しません。XML がブラウザ、アプリケーション・サーバーおよびデータベースで標準サポートされたことによって、企業は XML を使用してレガシー・アプリケーションを Web に統合し、様々な独自のファイル交換テンプレートおよびドキュメント交換テンプレートを XML に変換することを希望しています。

最近では、XML Schema などの新世代の XML 標準によって、構造化データとドキュメントの両方を処理できる統一されたデータ・モデルを使用できます。XML Schema は、XML としてマークアップされた文書をデータベース内に移動可能にすることによって、文書のコンテンツをデータと同じ厳密さで管理するための主要な新技術として登場しました。

Oracle XML DB には、XML に対応した高パフォーマンスな格納および取出しテクノロジが組み込まれています。Oracle XML DB によって、World Wide Web Consortium（W3C）の XML データ・モデルが Oracle9*i* データベースに完全に取り入れられ、XML をナビゲートおよび問い合せるための新しい標準アクセス方法の提供が可能になりました。これによって、リレーショナル・データベース・テクノロジおよび XML テクノロジのすべてのメリットが同時に得られます。Oracle XML DB を使用すると、XML を格納、問合せ、更新、変換または処理すると同時に、同じ XML データに SQL でアクセスできます。

個別のデータベース・サーバーではない Oracle XML DB

Oracle XML DB は、いわゆる個別のデータベース・サーバーではなく、ユーザーが Oracle データベースで使用する、高パフォーマンスな XML の格納および取出しに関連するテクノロジ・グループの名称です。Oracle XML DB は、高度に相互運用可能な方法で SQL と XML データ・モデルの両方を網羅し、XML をネイティブにサポートする拡張 Oracle データベースとも考えられます。

Oracle XML DB は、Oracle XML Developer's Kit（XDK）と組み合せて使用します。XDK では、Oracle9*i*AS または Oracle9*i* データベースの中間層で実行できる共通の開発時ユーティリティが提供されます。

> **参照：** XDK の詳細は、『Oracle9*i* XML Developer's Kit ガイド - XDK』を参照してください。

Oracle XML DB のメリット

通常、アプリケーションでは、構造化データを表として、また非構造化データをファイルまたはラージ・オブジェクト（LOB）として管理します。このため、開発者は様々な種類のデータを管理するために異なるパラダイムを適用する必要があります。アプリケーション開発は、次のいずれかに分類されます。

- **非構造化 :** 通常、ドキュメント・アクセスが透過的になり、表アクセスが複雑になります。
- **構造化 :** 通常、ドキュメント・アクセスが複雑になり、表アクセスが透過的になります。

Oracle XML DB のメリットは次のとおりです。

- W3C の同一の標準 XML データ・モデル（XML Schema）に従って、構造化データと非構造化データの両方を格納および管理できます。
- XML と SQL データ・ビュー間の完全な透過性および互換性が実現します。
- 高度なリポジトリ機能（フォルダリング、アクセス制御、FTP と WebDAV プロトコルのサポート、バージョニング）がサポートされます。これによって、アプリケーションが Oracle に入力された XML データを操作する場合に、ファイルの抽象的概念を保持できます。そのため、XML をデータベースに格納（問合せ可能になるようにレンダリング）すると同時に、一般的なデスクトップ・ツールを使用して XML にアクセスできます。
- 次の機能がサポートされることによって、非構造化データの管理が改善されます。
  - ピース単位更新
  - XML の索引付け
  - Oracle Text を使用した XML テキスト検索
  - データに対する複数のビュー（SQL アクセスに対するリレーショナル・ビューを含む）
  - XML 文書の文書内および文書間関係の規定
- 現在、複雑な XML の格納および取出しのパフォーマンスは高くありません。Oracle XML DB によって、XML 操作のパフォーマンスおよび拡張性が向上します。これは、XML 固有のデータ・キャッシュやメモリー管理、XML に対する問合せの最適化、XML リポジトリの特別な階層索引などに関連する多くの最適化が固有に行われるためです。
- たとえば、Oracle Gateway や外部表を介して異なるシステムのデータおよびドキュメントにアクセスし、標準のデータ・モデルに組み込むことができます。この統合によって、異なるシステムに格納されたデータを処理する必要があるアプリケーション開発の複雑さが低減されます。

Oracle XML DB の主要な機能

表 1-1 に、Oracle XML DB の機能を示します。この表には、Oracle9*i* データベース リリース 1（9.0.1）以上で使用可能な XML 機能が含まれています。

表 1-1 Oracle XML DB の機能

Oracle XML DB の機能	説明
XMLType	データベースにネイティブな XMLType データ型は、XML の格納および操作に有効です。複数の記憶域オプション（キャラクタ・ラージ・オブジェクト（CLOB）、構造化 XML 記憶域）が XMLType で使用可能であり、管理者は要件を満たす記憶域を選択できます。CLOB 記憶域は、元の XML のイメージに類似した、分解されていない記憶域です。 データベースにネイティブな構造化 XML 記憶域は、SQL 問合せの実行性を改善するために、XML を基礎となるオブジェクト・リレーショナル構造（Oracle によって自動的に作成および管理される）に細かく分解したものです。 XMLType を使用すると、次のような SQL 操作を実行できます。 ■ XML 操作、および XML データに対する問合せや OLAP ファンクションの起動など ■ SQL データに対する XPath 検索や XSLT 変換など XMLType に対して通常の SQL 索引または Oracle Text 索引を作成して、アプリケーションの様々なパフォーマンスを向上させることができます。第 4 章「XMLType の使用」を参照してください。
DOM 再現性	ドキュメント・オブジェクト・モデル（DOM）は、XML 文書の標準プログラム表現です。Oracle XML DB は、XML 文書を分解すると同時に、DOM 再現性を保持する方法で XML 文書を格納（構造化 XML 記憶域に）できます。格納する DOM は取得する DOM と同じになります。DOM 再現性とは、取得した XML データと完全に同じデータをプログラムが操作でき、格納処理が要素の順序や名前空間の存在などに影響しないことを意味します。ただし、DOM 再現性は、空白の保持などは保証しません。空白を含む XML の完全なレイアウトを保持する必要がある場合は、CLOB 記憶域を使用します。第 5 章「XMLType の構造化されたマッピング」を参照してください。
ドキュメントの再現性	空白文字を含め、元のドキュメントの完全な再現性を保持して XML を格納する必要があるアプリケーションでは、CLOB 記憶域オプションを使用します。
XML Schema	Oracle XML DB を使用すると、XML 文書を XML Schema に制約できます。W3C の標準 XML Schema に基づいて、表および型を自動的に作成できます。格納中の XML 文書のスキーマが妥当であることも規定できます。これは、すべての構造化および非構造化データの標準データ・モデルが存在し、データベースを使用してこのデータ・モデルを施行できることを意味します。第 5 章「XMLType の構造化されたマッピング」を参照してください。

表 1-1 Oracle XML DB の機能（続き）

Oracle XML DB の機能	説明
DOM 再現性を保持した XML Schema 格納	構造化記憶域（オブジェクト・リレーショナル形式）列、VARRAY、ネストした表および LOB を使用して、DOM 再現性を保持（格納された DOM が取り出される DOM と同じ）したまま、XML Schema にすべての要素または要素のサブツリーを格納します。第 5 章「XMLType の構造化されたマッピング」を参照してください。 **注意：** Oracle9*i* データベース リリース 1（9.0.1）以上で `XMLType` に使用可能な CLOB 記憶域オプションを使用すると、空白を保持できます。
XML Schema の検証	Oracle XML DB に XML 文書を格納中に、文書の構造が特定の XML Schema に準拠するかどうか（妥当かどうか）を確認できます（オプション）。第 6 章「XMLType データの変換および検証」を参照してください。
XML のピース単位更新	XPath を使用して、文書全体を再書込みせずに、更新中に文書の個々の要素および属性を指定できます。これは、特に大規模な XML 文書の場合により効率的です。第 5 章「XMLType の構造化されたマッピング」を参照してください。
XPath 検索	XPath 構文（SQL 文に埋め込まれた構文、または HTTP リクエストの一部）を使用して、データベース内の XML コンテンツを問い合せることができます。第 4 章「XMLType の使用」および第 7 章「Oracle Text を使用した XML データの検索」を参照してください。
XML の索引付け	XPath を使用して文書の一部を指定し、XPath 検索用の索引を作成します。これによって、XML 文書への高速アクセスが可能になります。第 4 章「XMLType の使用」を参照してください。
SQLX 演算子	（その場で XML 要素を作成するための）`XMLElement` などの新しい ANSI SQLX 標準に準拠する新しい SQL メンバー関数によって、XML 問合せおよびその場での XML 生成が簡単になります。これらの関数によって、SQL および XML の隠喩が相互運用可能になります。第 10 章「データベースからの XML データの生成」を参照してください。
XMLType の XSLT 変換	XSLT を使用して、SQL 演算子を介して XML 文書を変換します。データベース内で実行する高パフォーマンスの XSLT 変換です。第 6 章「XMLType データの変換および検証」および付録 D「XSLT の手引き」を参照してください。
XML の遅延ロード	Oracle XML DB は仮想的な DOM を提供します。これは、要求されたデータ行のみをロードし、メモリーの使用量が大きくなりすぎた場合に、文書内の参照済の部分を廃棄します。多くの同時ユーザーが大規模な XML 文書を処理している場合にこれを使用すると、高い拡張性を得ることができます。仮想的な DOM は、クライアントの Java 実行環境内またはサーバーで実行している Java インタフェースを介して使用できます。第 8 章「PL/SQL API for XMLType」を参照してください。
XML ビュー	XML ビューを作成して、XML 文書の様々なフラグメントまたはリレーショナル表の永続集計を作成できます。Oracle Gateway を使用して、異機種間データ・ソースのビューを作成することもできます。第 11 章「XMLType ビュー」を参照してください。

表 1-1 Oracle XML DB の機能（続き）

Oracle XML DB の機能	説明
PL/SQL および OCI インタフェース	XML データへのアクセスおよび操作に DOM およびその他の API を使用します。XML への静的および動的アクセスが可能になります。第 8 章「PL/SQL API for XMLType」を参照してください。
スキーマ・キャッシュ	構造情報（要素タグ、データ型、格納場所など）は、特別なスキーマ・キャッシュに保持され、アクセス時間および格納コストが最小化されます。第 5 章「XMLType の構造化されたマッピング」を参照してください。
XML の生成	`SYS_XMLGEN` や `SYS_XMLAGG` などの SQL 演算子によって、SQL 問合せからのネイティブで高パフォーマンスな XML 生成が可能になります。XML 表および要素をその場で作成するための `XMLElement()` などの新しい演算子によって、XML の生成がより柔軟になります。第 10 章「データベースからの XML データの生成」を参照してください。これらの演算子は、新しい ANSI SQLX 標準に準拠しています。
Oracle XML DB Repository	組込み XML リポジトリです。このリポジトリはフォルダリングに使用でき、それによって Oracle XML DB に格納されている XML コンテンツをディレクトリに類似したフォルダの階層として表示できます。第 13 章「Oracle XML DB Foldering」を参照してください。 ■ リポジトリでは、すべての `XMLType` オブジェクトに対するアクセス制御リスト（ACL）がサポートされており、特定のシステム定義の権限を付与することに加えて、独自の権限を定義することができます。第 18 章「Oracle XML DB リソースのセキュリティ」を参照してください。 ■ リポジトリを使用すると、XML コンテンツを多くの一般的なクライアント・ツールおよびデスクトップ・ツールを介してナビゲート可能なディレクトリとして表示できます。リポジトリで管理される項目を、リソースといいます。 ■ リポジトリでは、階層的な索引付けが可能です。Oracle XML DB では、フォルダ検索を高速化するための特別な階層索引が提供されます。また、リレーショナル表内の階層データをフォルダに自動的にマップできます（この場合、階層は、`CONNECT BY` などを使用して既存のリレーショナル情報によって定義されます）。
SQL リポジトリ検索	SQL を使用して、XML リポジトリを検索できます。UNDER_PATH や DEPTH などの演算子を使用すると、アプリケーションでフォルダ、XML ファイル・メタデータ（所有者や作成日など）および XML ファイルのコンテンツを検索できます。第 15 章「RESOURCE_VIEW および PATH_VIEW」を参照してください。
WebDAV、HTTP および FTP アクセス	WebDAV および FTP を使用して、フォルダリングされたすべての `XMLType` 行にアクセスできます。Oracle9*i* データベース内の XML データを操作している場合は、HTTP API を使用できます。第 19 章「FTP、HTTP および WebDAV プロトコルの 使用」を参照してください。
バージョニング	Oracle XML DB には、XML リポジトリで管理されているリソースのバージョニングおよびバージョン管理機能があります。第 14 章「Oracle XML DB Versioning」を参照してください。

Oracle XML DB および XML Schema

XML Schema によって、ドキュメントとデータのモデル化が統合されます。Oracle XML DB では、XML Schema を使用して自動的に表および型を作成できます。そのため、すべてのデータ（構造化、非構造化、疑似 / 半構造化）用に標準データ・モデルを開発および使用できます。今回のリリースでは、Oracle9*i* データベースを使用して、すべてのデータに対してこのデータ・モデルを施行できます。

XML Schema に基づく `XMLType` 表および列を作成できます。また、オプションで、次のように指定することもできます。

- XMLType 表および列が事前登録済の XML Schema に準拠する
- XMLType 表および列が XML Schema で指定されている構造化記憶域形式で格納され、DOM 再現性を保持する

また、`XMLType` ビューを使用して、既存のリレーショナル・データおよびオブジェクト・リレーショナル・データを XML 形式にラップすることもできます。

`XMLType` オブジェクトは、XML Schema に基づくオブジェクトまたは XML Schema に基づかないオブジェクトとして格納できます。

- XML Schema に基づくオブジェクト : Oracle XML DB に LOB として格納されるか、または表、列またはビュー内の構造化記憶域に格納されます（オブジェクト・リレーショナル形式）。
- XML Schema に基づかないオブジェクト : Oracle XML DB に LOB として格納されます。

XML インスタンスから構造化または LOB 記憶域にマップできます。マッピングは XML Schema 内で指定でき、XML Schema は Oracle XML DB に登録しておく必要があります。これは、XML Schema に基づくインスタンス・ドキュメントを格納する前に実行する必要がある手順です。XML Schema は、一度登録すると、URL を使用して参照できます。

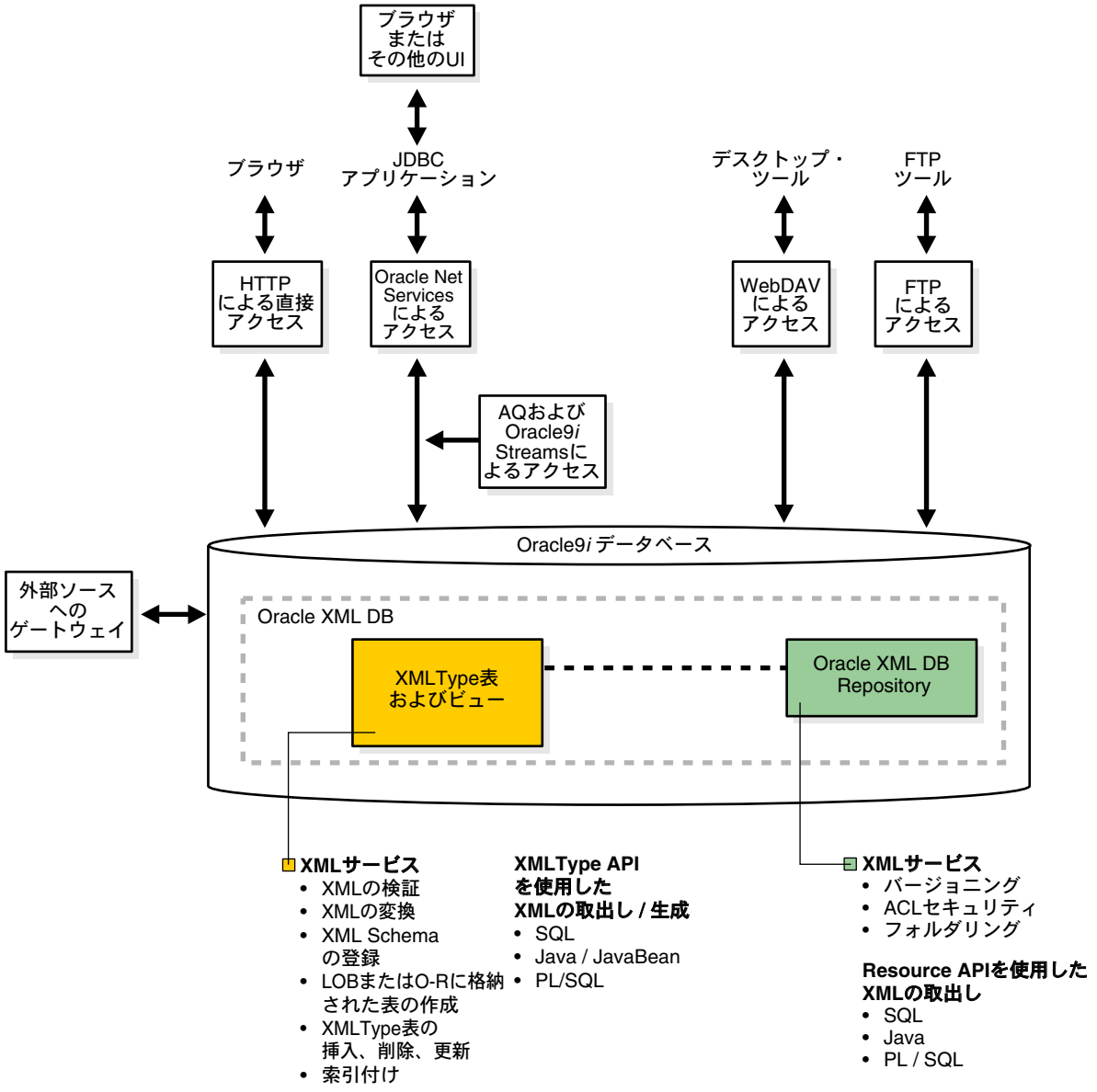
Oracle XML DB のアーキテクチャ

図 1-1 に、Oracle XML DB のアーキテクチャを示します。Oracle XML DB アーキテクチャの 2 つの主な機能は次のとおりです。

- **`XMLType` 表およびビュー記憶域**（`XMLType` 表およびビューの記憶域を含む）
- **Oracle XML DB Repository**（このマニュアルでは、XML リポジトリまたはリポジトリともいう）

図 1-1 の後の項では、アーキテクチャの詳細を説明します。

図 1-1 Oracle XML DB のアーキテクチャ : XMLType 記憶域およびリポジトリ



XMLType 表およびビュー記憶域

Oracle XML DB 内の「XMLType 表およびビュー記憶域」には、SQL と緊密に統合されたデータベースにネイティブな XML の格納および取出し機能があります。

XML Schema 定義ファイルを含む XML データは、LOB または構造化記憶域（オブジェクト・リレーショナル形式）に格納するか、あるいは LOB と構造化記憶域の両方を組み合せた任意のハイブリッド記憶域を使用して格納できます。第 3 章「Oracle XML DB の使用」および第 4 章「XMLType の使用」を参照してください。

サポートされている XML アクセス API

- PL/SQL API for XMLType および Java API for XMLType: これらの API を使用して次の操作を実行します。
  - XMLType 表、列およびビューの作成
  - XML データの問合せおよび取出し
- XMLElement() や XMLForest() などの SQL 関数 : アプリケーションでは、標準の SQL、および SQLX 標準に準拠した SQL メンバー関数を使用して、データベース内の XML データを問い合せることができます。

  参照 : 第 IV 部「既存データの XML 表示」を参照してください。

サポートされている XML サービス

Oracle XML DB では、XML データへのアクセスまたは生成の他に、データに対して様々な操作を実行できます。

- **PL/SQL API for XMLType および Java API for XMLType:** これらの API を使用すると、XML データの更新、削除、挿入などの XMLType データの操作を実行できます。
- **索引付け :** これによって、XPath 機能が重要でない場合のデータ検索が高速化されます。これは、LOB に格納された XML データに最適です。
- **XML データの変換 :** XMLType の XMLTransform() 関数、XDK の XSLT プロセッサまたは XSQL Servlet Pages パブリッシング・フレームワークを使用して、その他の XML や HTML などへ変換します。第 6 章「XMLType データの変換および検証」および第 10 章「データベースからの XML データの生成」を参照してください。
- **XML データの検証 :** XML データがデータベース内に格納されている場合に、XML Schema に対してその XML データを検証します。

  参照 : 1-11 ページの「XMLType 記憶域のアーキテクチャ」を参照してください。

Oracle XML DB Repository

Oracle XML DB Repository（XML リポジトリまたはリポジトリ）は、XML データの処理用に最適化された Oracle9*i* データベース内の XML データ・リポジトリです。Oracle XML DB Repository の主要部分は、Oracle XML DB Foldering モジュールです。

> **参照：** 第 13 章「Oracle XML DB Foldering」を参照してください。

Oracle XML DB Repository のコンテンツをリソースといいます。これらはコンテナ（ディレクトリやフォルダ）またはファイルのいずれかになります。すべてのリソースは、パス名で識別され、ユーザーが定義した実際のコンテンツに加えて、所有者や作成日などの（拡張可能な）一連の（メタデータ）プロパティを持ちます。

サポートされている XML アクセス API

図 1-1 には、Oracle XML DB でサポートされている次の XML アクセスおよび操作 API を示しています。

- **Oracle XML DB Resource API:** この API を使用して、フォルダリングされた `XMLType` およびその他のデータ（Oracle XML DB の階層的に索引付けされたリポジトリを使用してアクセスされるデータ）にアクセスします。API が使用可能な言語は次のとおりです。
  - SQL（RESOURCE_VIEW および PATH_VIEW API を使用）
  - PL/SQL（`DBMS_XDB`）API
  - Java API

  > **参照：** 第 V 部「Oracle XML DB Repository: フォルダリング、セキュリティおよびプロトコル」を参照してください。

- **Oracle XML DB Protocal Server:** Oracle XML DB では、データベースの `XMLType` 表および列内に格納された XML データに高速アクセスするために、JDBC に加えて FTP、HTTP および WebDAV プロトコルがサポートされています。第 19 章「FTP、HTTP および WebDAV プロトコルの 使用」を参照してください。

サポートされている XML サービス

XML リポジトリは、XML およびその他のデータにアクセスおよび操作するための API の他に、次のサービスをサポートします。

- **バージョニング :** Oracle XML DB は、リソースのバージョニングをサポートしています。PL/SQL パッケージ `DBMS_XDB_VERSION` は、リソースをバージョン管理するためのファンクションを実装します。リソースに対する後続のすべての更新では、前のバージョンに対応するデータが保持されたまま、新しいバージョンが作成されます。

- **ACL セキュリティ :** Oracle XML DB リソースへのアクセスのセキュリティは、アクセス制御リスト（ACL）のメカニズムに基づいています。Oracle XML DB 内のすべてのリソースには、権限をリストした ACL が関連付けられています。リソースがアクセスまたは操作されるたびに、これらの ACL によって操作が正当かどうかが判断されます。
- **フォルダリング :** XML リポジトリのフォルダリング・モジュールによって、コンテナ（フォルダまたはディレクトリ）およびリソースの永続階層が管理されます。プロトコル・サーバー、Schema Manager、Oracle XML DB の RESOURCE_VIEW API など、その他の Oracle XML DB モジュールでは、フォルダリング・モジュールを使用してパス名がリソースにマップされます。

XMLType 記憶域のアーキテクチャ

図 1-2 に、XMLType 表およびビュー記憶域のアーキテクチャの詳細を示します。

XMLType 表、XMLType 列を含む表およびビューでは、それらが XML Schema に基づき、XML Schema が Oracle XML DB に登録されている場合、XML 要素はデータベース表にマップされます。これらの要素は、XML リポジトリで簡単に表示およびアクセスできます。

XMLType 表および XMLType 列を含む表内のデータは、キャラクタ・ラージ・オブジェクト（CLOB）に格納するか、または構造化 XML 記憶域にネイティブに格納できます。

XMLType ビュー内のデータは、ローカル表または DBLink を使用してアクセスされるリモート表に格納できます。

XMLType 表とビューは両方とも、B* ツリー、Oracle Text、ファンクション索引またはビットマップ索引を使用して索引付けできます。

XML リポジトリ内のデータにアクセスするためのオプションには、次のものが含まれます。

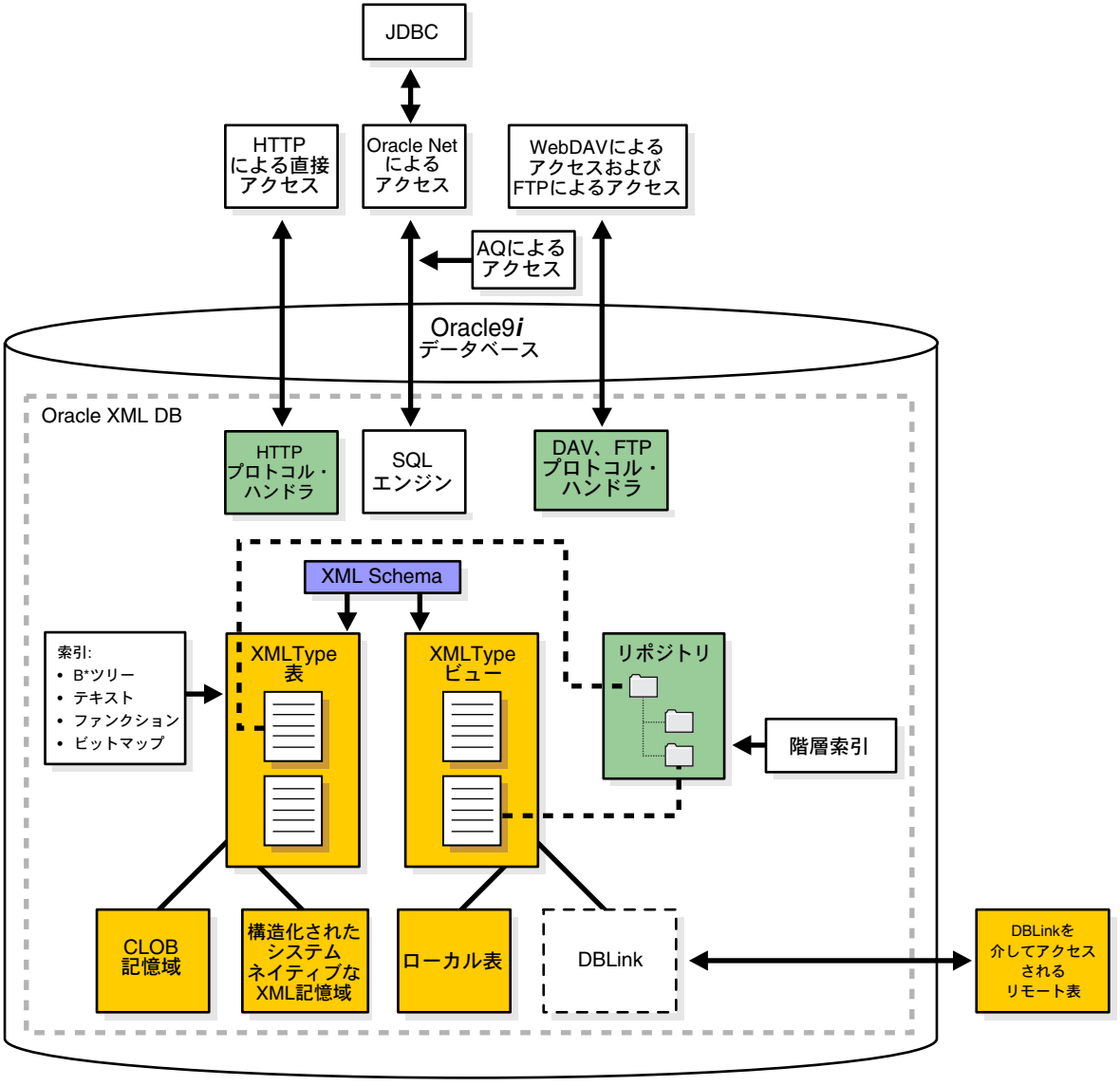
- HTTP プロトコル・ハンドラを介した HTTP
- WebDAV および FTP プロトコル・サーバーを介した WebDAV および FTP
- JDBC を含む Oracle Net Services を介した SQL

  Oracle XML DB では、アドバンスト・キューイング（AQ）および SOAP を使用した XML データのメッセージ機能もサポートされています。

  参照 : 次の部または章を参照してください。

  - 第 II 部「Oracle XML DB からの XML データの格納 および取出し」
  - 第 19 章「FTP、HTTP および WebDAV プロトコルの 使用」
  - 第 21 章「Oracle Enterprise Manager を使用した Oracle XML DB の管理」
  - 第 24 章「Advanced Queuing（AQ）および Oracle Streams を使用した XML データの変換」

図 1-2 Oracle XML DB:XMLType の格納および取出しのアーキテクチャ



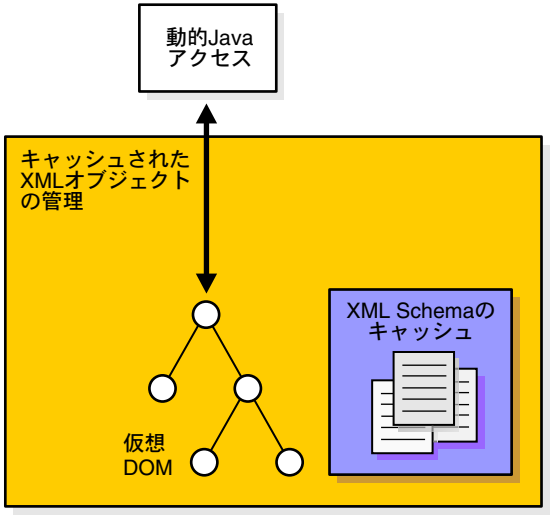
キャッシュされた XML オブジェクト管理のアーキテクチャ

図 1-3 に、XMLType インスタンスへのプログラム・アクセスに関連した、Oracle XML DB のキャッシュされた XML オブジェクト管理のアーキテクチャを示します。Oracle XML DB のキャッシュは、クライアント（Oracle JDBC OCI Driver を使用）またはサーバー内に配置できます。このキャッシュによって、次のものが提供されます。

- 格納された XMLType からの遅延ロードされた仮想的な DOM。これらのノードは、必要に応じてフェッチされます。
- XML Schema のキャッシュ。

これによって、メモリー内で XML DOM 全体をロードせずに、XML に動的アクセスできます。これは、コンパイル中に DOM 内のノードへのオフセットを計算することによって実現します。

図 1-3 キャッシュされた XML オブジェクト管理のアーキテクチャ



XML リポジトリのアーキテクチャ

図 1-4 に、Oracle XML DB Repository（XML リポジトリ）のアーキテクチャを示します。

リソースは、Oracle XML DB で管理される任意のコンテンツであり、それに対してファイルまたはフォルダの隠喩を保持または表示する必要があります。

各リソースには、名前、リソースを参照できるユーザーを決定するためのアクセス制御リスト、特定の静的プロパティ、およびアプリケーションによって拡張可能ないくつかの追加プロパティがあります。リポジトリを使用するアプリケーションでは、親子関係にあるフォルダの論理ビューが取得されます。リポジトリは、RESOURCE_VIEW を使用してデータベース内で（SQL アクセスなどに）使用できます。

Oracle9*i* データベース内の RESOURCE_VIEW は、問合せ可能なリソース名を含むリソース（それ自体は `XMLType`）、ACL およびプロパティ（静的または拡張可能）で構成されます。

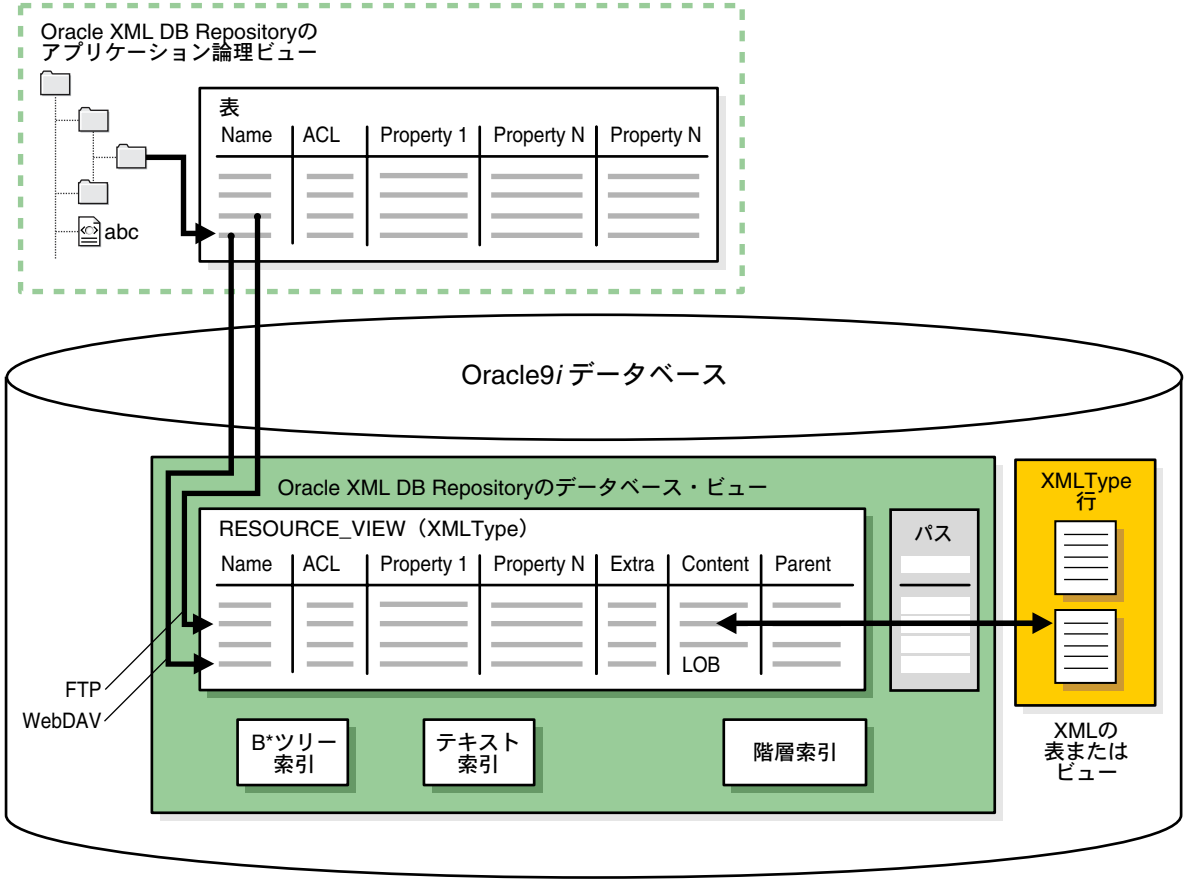
- リソースを構成するコンテンツが XML（`XMLType` 表またはビュー内に格納されている）である場合、RESOURCE_VIEW はそのコンテンツを格納する `XMLType` 行を指します。
- コンテンツが XML でない場合、RESOURCE_VIEW はそれを LOB として格納します。

フォルダ間の親子関係（階層を構築するために必要）は、階層索引を使用して効率的に保持および検索されます。テキスト索引はリソースのプロパティの検索に使用でき、名前および ACL に対する内部 B* ツリー索引によって、リソース `XMLType` のこれらの属性に対するアクセスが高速化されます。

RESOURCE_VIEW には、リソース情報に加えて、各リソースへのパスを保持するパス列も含まれます。

> **参照：** 次の章を参照してください。
>
> - 第 13 章「Oracle XML DB Foldering」
> - 第 15 章「RESOURCE_VIEW および PATH_VIEW」

図 1-4 Oracle XML DB: リポジトリのアーキテクチャ



Oracle XML DB を使用するメリット

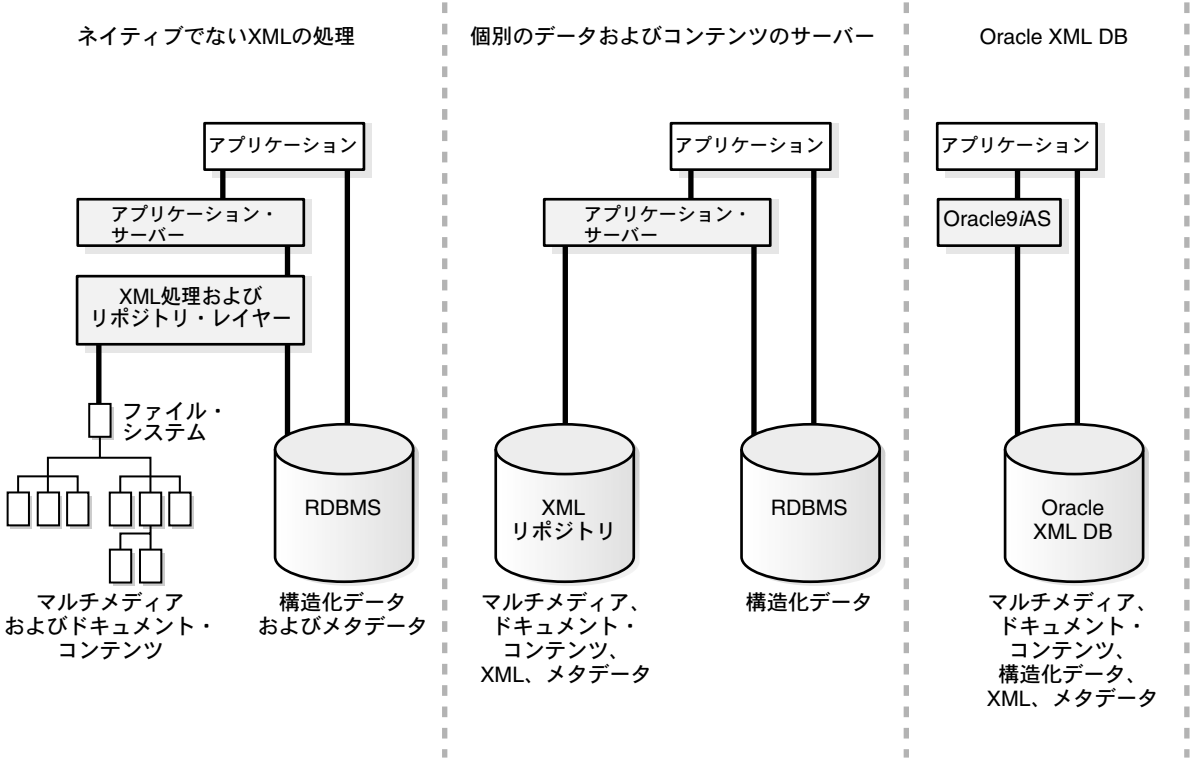
次の項では、XML データベース・アプリケーションの構築における Oracle XML DB のメリットについて説明します。主なメリットは次のとおりです。

- Oracle XML DB を使用したデータおよびコンテンツの統合
- Oracle XML DB による複雑な XML 文書の高速格納および取出し
- Oracle XML DB によるアプリケーションの統合支援
- データが XML でない場合に使用可能な XMLType ビュー

Oracle XML DB を使用したデータおよびコンテンツの統合

多くのアプリケーションのデータおよび Web コンテンツは、リレーショナル・データベースまたはファイル・システム内、あるいはそれらを組み合せたものに格納されます。多くの場合、XML は転送に使用され、データベースまたはファイル・システムから生成されます。転送される XML のボリュームが大きくなると、これらの XML 文書を再生成するコストが増加し、XML コンテンツの保存にはこれらの格納方法は効率的ではありません。図 1-5 を参照してください。Oracle XML DB は、XML コンテンツを効率的に保存します。また、XML に対するネイティブな拡張サポートを提供します。

図 1-5 データおよびコンテンツの統合 : 一般的な XML アーキテクチャ



通常、企業は構造化データと非構造化データを異なる方法で管理しています。

- 表内の非構造化データの場合、ドキュメント・アクセスが透過的になり、表アクセスが複雑になります。
- 通常はバイナリ・ラージ・オブジェクト（BLOB）の構造化データの場合、ドキュメント・アクセスが複雑になり、表アクセスが透過的になります。

Oracle XML DB では、標準データ・モデル、標準 SQL および標準 XML を使用して、構造化、非構造化、疑似または半構造化のすべてのデータを格納および管理できます。

Oracle XML DB によって、XML と SQL 間の完全な透過性および互換性が実現します。次の両方の操作を行うことができます。

- オブジェクト・リレーショナル（表など）・データに対する XML 操作
- XML 文書に対する SQL 操作

これによって、データベースでの XML 形式のデータ・コンテンツ対するアクセス性が向上します。

データベース機能の使用

以前のリリースでは、データベースでの XML サポートが強力ではなかったため、多くの場合、XML データをファイルまたは CLOB などの非構造化記憶域に格納していました。XML データをファイルに格納するか、CLOB に格納するかによって、使用できなかった Oracle データベースの機能がありました。

- **索引付けおよび検索 :** アプリケーションでは、「2002 年の 3 月～ 4 月に作成されたすべての製品定義を検索する」のような問合せが実行されます。この問合せは、日付列の B* ツリー索引でサポートされています。以前は、コンテンツ管理ベンダーは、独自の問合せ API を構築してこの問題を処理する必要がありました。Oracle XML DB を使用すると、XML データに対する効率的な構造化検索ができるようになります。第 4 章「XMLType の使用」、第 10 章「データベースからの XML データの生成」および第 7 章「Oracle Text を使用した XML データの検索」を参照してください。
- **更新およびトランザクション処理 :** 商用のリレーショナル・データベースでは、レコードのサブパーツを高速更新することによって、更新を試行しているユーザー間の競合が最小化されています。従来、ドキュメント中心のデータは、XML を介して協調環境を構築しているため、この要件はより重要になります。ファイル記憶域または CLOB 記憶域では、Oracle XML DB で行われる細分化された並列性制御を行うことができません。第 4 章「XMLType の使用」を参照してください。
- **関係の管理 :** 通常、なんらかの構造を持つデータには外部キー制約があります。現在、XML データを格納するシステムにはこの機能がないため、アプリケーション・コードで任意の制約を実装する必要があります。Oracle XML DB を使用すると、XML Schema の定義に基づいて XML データを制約できるため、構造化データが持つ関係を制御できます。第 5 章「XMLType の構造化されたマッピング」、および第 4 章「XMLType の使用」の最後に示す発注書の例を参照してください。

- **データの複数のビュー :** 多くの企業アプリケーションでは、別々のモジュール用に、それぞれ異なる方法でデータをグループ化する必要があります。そのため、これらの複数の方法でデータを組み合せることができるように、リレーショナル・ビューが必要になります。Oracle XML DB では、XML でビューを使用することによって、様々なアプリケーションで使用する場合など、XML に別々の論理的な抽象概念が作成されます。第 11 章「XMLType ビュー」を参照してください。
- **パフォーマンスおよび拡張性 :** データは、高速な格納、取出しおよび問合せが可能である必要があります。通常、ファイルまたは CLOB のロードおよび解析は、リレーショナル・データのアクセスより遅くなります。Oracle XML DB では、XML の格納および取出しが大幅に高速化されます。第 2 章「Oracle XML DB を使用する前に」および第 3 章「Oracle XML DB の使用」を参照してください。
- **簡単な開発 :** データベースは、個々のデータ要素を簡単に操作、変換および変更するための標準的な手段を提供する、最も重要なアプリケーション・プラットフォームです。通常の XML パーサーでは、XML データに対する標準的な読込みアクセスが提供されますが、個々の XML 要素を簡単に変更および格納するための手段は提供されません。Oracle XML DB では、XML Schema、XPath、DOM および Java を使用してデータを格納、変更および取り出すための多くの標準的な手段がサポートされています。

  参照 : 次の章を参照してください。

  - 第 9 章「Java API for XMLType」
  - 第 15 章「RESOURCE_VIEW および PATH_VIEW」
  - 第 16 章「Oracle XML DB Resource API for PL/SQL（DBMS_XDB）」

XML 機能の活用

XML ファイル記憶域のデメリットが原因で、XML をデータベース表および列に分割する必要がある場合でも、XML には次のメリットがあります。

- **構造に非依存 :** XML のオープン・コンテンツ・モデルは、表と列のみの構造には簡単に取得されません。XML Schema によって、コンテナのみでなく、グローバルな要素宣言が可能になります。そのため、アプリケーションの進化に伴って XML 文書内でデータ項目がどこに移動するかにかかわらず、特定のデータ項目を検索できます。第 5 章「XMLType の構造化されたマッピング」を参照してください。
- **記憶域に非依存 :** リレーショナル設計を使用する場合、クライアント・プログラムでは、データの格納場所、格納形式、格納先の表および表間の関係が認識されている必要があります。`XMLType` を使用すると、それらの情報を使用せずにアプリケーションを作成でき、DBA は構造化データを物理表および物理列の記憶域にマップできます。第 5 章「XMLType の構造化されたマッピング」および第 13 章「Oracle XML DB Foldering」を参照してください。

- **簡単な表示 :** XML は、ブラウザ、多くの一般的なデスクトップ・アプリケーションおよびほぼすべてのインターネット・アプリケーションでネイティブに認識されます。一般的に、リレーショナル・データは、アプリケーションから直接アクセスできず、一般のクライアントがアクセスできるようにプログラミングする必要があります。Oracle XML DB では、データは XML として格納および取り出されるため、データベースのコンテンツを表示するためのプログラミングは必要ありません。次の章およびマニュアルを参照してください。
  - 第 6 章「XMLType データの変換および検証」
  - 第 10 章「データベースからの XML データの生成」
  - 第 11 章「XMLType ビュー」
  - 『Oracle9*i* XML Developer's Kit ガイド - XDK』の「XSQL Pages パブリッシング・フレームワーク」の章。この章には、`XMLType` の例があります。
- **簡単な交換 :** XML は、Business-to-Business（B2B）のデータ交換で使用される言語です。通常、XML を任意の表構造に格納する必要がある場合は、XML の内容を独自の方法で解釈し、分解します。言語を分解すると、情報が失われ、交換が困難になります。Oracle XML DB では、XML をネイティブに認識し、格納および取出し処理での DOM 再現性を提供することによって、完全な交換が可能になります。次の章を参照してください。
  - 第 6 章「XMLType データの変換および検証」
  - 第 11 章「XMLType ビュー」

Oracle XML DB による複雑な XML 文書の高速格納および取出し

現在、ユーザーは、複雑で大規模な多くの XML 文書を格納および取り出す場合のパフォーマンスの問題に直面しています。Oracle XML DB では、XML 操作のパフォーマンスおよび拡張性が大幅に向上されています。主なパフォーマンス機能は次のとおりです。

- ネイティブな `XMLType`。第 4 章「XMLType の使用」を参照してください。
- 遅延評価された仮想的な DOM のサポート。第 8 章「PL/SQL API for XMLType」を参照してください。
- データベースに統合された非定型の XPath および XSLT のサポート。このサポートについては、第 4 章「XMLType の使用」や第 6 章「XMLType データの変換および検証」などの章を参照してください。
- XML Schema キャッシュのサポート。第 5 章「XMLType の構造化されたマッピング」を参照してください。
- CTXPATH のテキスト索引付け。第 7 章「Oracle Text を使用した XML データの検索」を参照してください。
- リポジトリに対する階層索引。第 13 章「Oracle XML DB Foldering」を参照してください。

Oracle XML DB によるアプリケーションの統合支援

Oracle XML DB を使用すると、異なるシステムのデータにゲートウェイを介してアクセスし、1 つの共通のデータ・モデルとして組み合せることができます。これによって、異なるストアのデータを処理する必要があるアプリケーション開発の複雑さが低減されます。

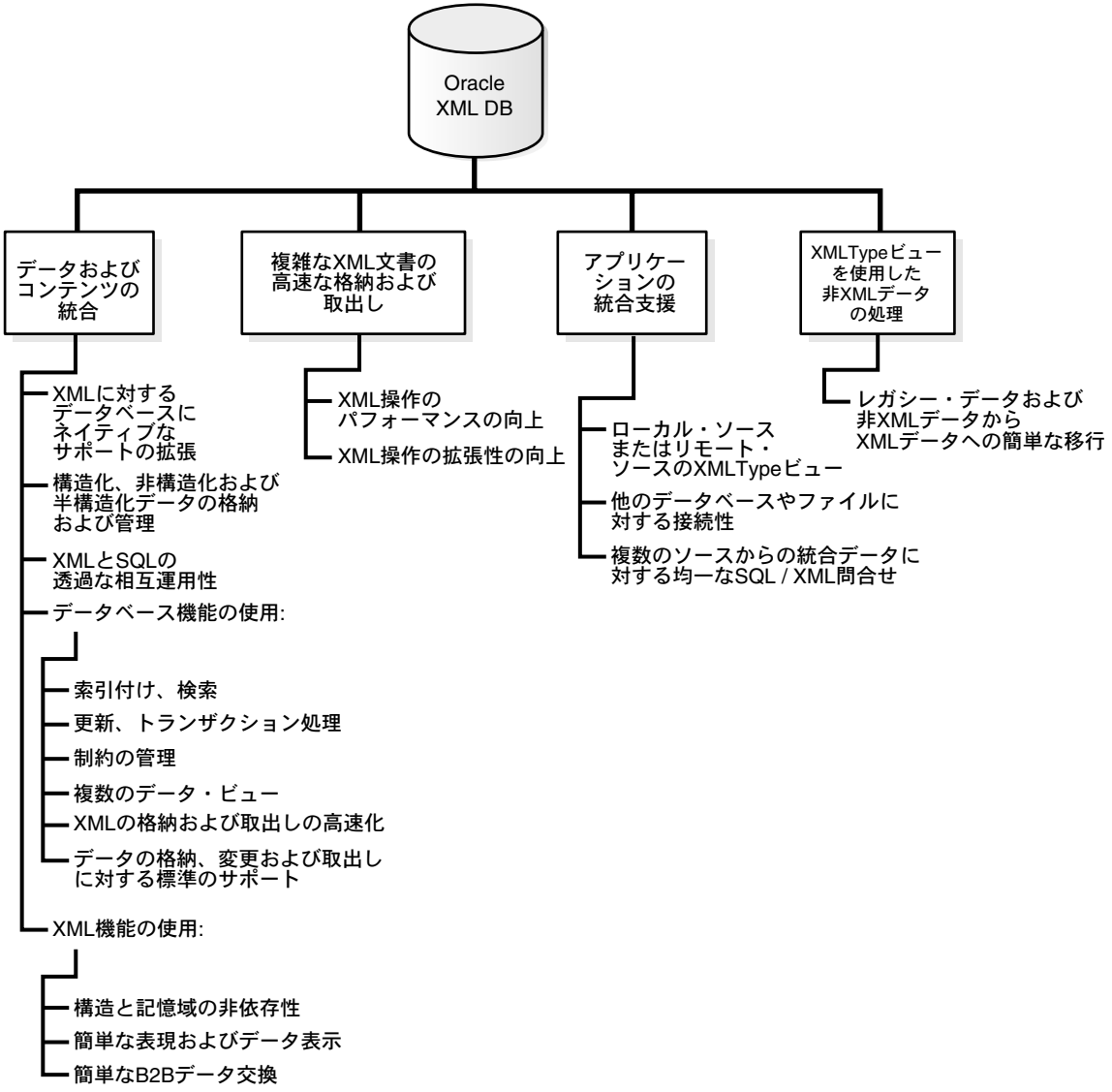
データが XML でない場合に使用可能な XMLType ビュー

XMLType ビューでは、既存のリレーショナル・データおよびオブジェクト・リレーショナル・データを XML 形式にラップする方法が提供されます。これは、XML でないレガシー・データを XML 形式に移行する必要がある場合などに特に有効です。XMLType ビューを使用すると、アプリケーション・コードを変更する必要がありません。

> **参照：** 第 11 章「XMLType ビュー」を参照してください。

XMLType ビューを使用するには、まず XML と SQL オブジェクト型の間のマッピングを表す注釈が付いた XML Schema を登録する必要があります。その後、適切な SQL オブジェクト型のインスタンスを構築する基礎となる問合せを発行することによって、この XML Schema（マッピング）に準拠する XMLType ビューを作成できます。図 1-6 に、Oracle XML DB のメリットの概要を示します。

図 1-6 Oracle XML DB のメリット



Oracle Text を使用した CLOB に格納された XML データの検索

Oracle では、XML での特別な索引付け（セクション検索用の Oracle Text 索引を含む）、XML を処理するための特別な演算子、XML の集計、および XML を伴う問合せの特別な最適化が可能です。

キャラクタ・ラージ・オブジェクト（CLOB）に格納された XML データ、または構造化記憶域内の XMLType 列に（オブジェクト・リレーショナル形式で）格納された XML データは、Oracle Text を使用して索引付けできます。HASPATH() および INPATH() 演算子は、XML データの検索を最適化するように設計されています。この場合、XML テキストにおける部分文字列一致検索が可能です。

Oracle9*i* データベース リリース 2（9.2）では、次の機能も提供されます。

- CONTAINS() 関数。これは、existsNode() とともに、XPath を使用した検索に使用できます。この関数は、XPath 問合せで ora:contains 関数として、また、existsNode() 関数の一部として使用します。
- URIType および XDBUriType 列に対する索引作成機能。
- 新しい索引タイプ CTXXPATH。これを使用すると、existsNode() を使用した Oracle XML DB での XPath 検索のパフォーマンスが向上します。

参照： 次の章またはマニュアルを参照してください。

- 第 7 章「Oracle Text を使用した XML データの検索」
- 『Oracle Text アプリケーション開発者ガイド』
- 『Oracle Text リファレンス』

アドバンスト・キューイングを使用した Oracle XML DB の XML メッセージ・アプリケーションの構築

今回のリリースでは、アドバンスト・キューイングで次のものを使用できます。

- XML Schema に基づく XMLType を含む、メッセージ型およびペイロード型としての XMLType
- XMLType メッセージのキューおよびデキュー

参照：

- Oracle Advanced Queuing での XMLType の使用については、『Oracle9*i* アプリケーション開発者ガイド - アドバンスト・キューイング』を参照してください。
- 第 24 章「Advanced Queuing（AQ）および Oracle Streams を使用した XML データの変換」も参照してください。

Oracle Enterprise Manager を使用した Oracle XML DB アプリケーションの管理

Oracle Enterprise Manager（Enterprise Manager）を使用すると、Oracle XML DB アプリケーションを管理および構成できます。Enterprise Manager の Graphical User Interface（GUI）を使用すると、次のタスクを簡単に実行できます。

- 構成
  - プロトコル・サーバーの構成を含む、Oracle XML DB の構成
  - Oracle XML DB 構成パラメータの表示および編集
  - XML Schema の登録
- リソースの作成
  - リソース ACL 定義の編集などのリソース・セキュリティの管理
  - リソース権限の付与および取消し
  - リソース索引の作成および編集
  - Oracle XML DB の階層リポジトリの表示およびナビゲート
- XML Schema に基づく表およびビューの作成
  - XML Schema に基づく格納インフラストラクチャの作成
  - XML Schema の編集
  - XMLType 表および XMLType 列を含む表の作成
  - XML Schema に基づくビューの作成
  - XPath 式に基づくファンクション索引の作成

参照： 第 21 章「Oracle Enterprise Manager を使用した Oracle XML DB の管理」を参照してください。

Oracle XML DB の実行要件

Oracle XML DB は、Oracle9*i* データベース リリース 2（9.2）で使用可能です。

参照：

- Oracle XML DB に関する最新のニュースおよびホワイト・ペーパーについては、`http://otn.oracle.co.jp/` を参照してください。
- 第 2 章「Oracle XML DB を使用する前に」も参照してください。

Oracle XML DB でサポートされている標準

Oracle XML DB では、XML、SQL、Java、インターネットなど、すべての主要な標準がサポートされています。

- W3C の XML Schema 1.0 勧告。XML Schema の登録、XML Schema に対する格納された XML コンテンツの検証、または XML Schema に対するサーバーに格納された XML の制約を行うことができます。
- W3C の XPath 1.0 勧告。HTTP リクエストまたは SQL のいずれかから、XPath を使用してデータベース内に格納された XML を検索できます。
- XML 関連仕様（SQL/XML）の ISO ANSI 草案（ISO/IEC 9075 Part 14 and ANSI）。新しい ANSI SQLX 関数を使用して、SQL から XML を問い合せることができます。
- Java Database Connectivity（JDBC）API。Java プログラマは、XML へ JDBC アクセスできます。
- W3C の XSL 1.0 勧告。XSLT を使用して、サーバーで XML 文書を変換できます。
- W3C の DOM レベル 1.0 および 2.0 コア勧告。動的アクセスのために、サーバーに格納された XML を XML DOM として取り出すことができます。
- プロトコル・サポート。HTTP、FTP、IETF WebDAV などの標準プロトコルおよび Oracle Net を使用して、Oracle XML DB に XML データを格納したり、Oracle XML DB から XML データを取り出すことができます。第 19 章「FTP、HTTP および WebDAV プロトコルの 使用」を参照してください。
- Java サーブレットのバージョン 2.2（ただし、Servlet WAR ファイル、`web.xml` はサポートされず、現在サポートされているのは、1 つの `ServletContext` および 1 つの Web アプリケーションのみです。ステートフル・サーブレットはサポートされていません）。第 20 章「Java での Oracle XML DB アプリケーションの作成」を参照してください。

- Simple Object Access Protocol（SOAP）。SOAP リクエストからサーバーに格納された XML にアクセスできます。WSDL および UDDI を介して、Oracle XML DB および Oracle9*i*AS を使用して Web サービスを構築、公開または検索できます。Oracle Advanced Queuing iDAP（キューイング操作に対する SOAP 仕様）を Oracle9*i* データベースに格納された XML に使用できます。第 24 章「Advanced Queuing（AQ）および Oracle Streams を使用した XML データの変換」および『Oracle9*i* アプリケーション開発者ガイド - アドバンスト・キューイング』を参照してください。

Oracle XML DB の技術サポート

オラクル社カスタマ・サポート・センターの通常のサポート・チャネルの他に、Oracle の XML 対応テクノロジの技術サポートが、OTN-J の「会議室」から無償で利用できます。

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

OTN-J の登録ユーザーでなくても、OTN-J の掲示板で XML に関連する質問を送信したり、質問に回答することができます。OTN-J の掲示板を利用するには、次の手順に従います。

1. OTN-J サイトの左側のナビゲーション・バーで、「Services」の「掲示板」を選択します。
2. 「テクノロジー」のセクションまでスクロールし、「XML の部屋」を選択します。
3. そこで、質問、コメント、リクエストまたはエラー・レポートを送信します。

このマニュアルで使用する用語

表 1-2 に、このマニュアルで使用する用語の説明を示します。

参照：「用語集」を参照してください。

表 1-2 このマニュアルで使用する用語

このマニュアルで使用する用語	説明
XML Schema	それに準拠するインスタンス・ドキュメントの構造およびその他の様々なセマンティクスを記述するために使用できるスキーマ定義言語（それ自体も XML）です。付録 B「XML Schema の手引き」を参照してください。 Oracle XML DB は、注釈付き XML Schema（Oracle XML DB で定義された追加の属性を含む XML Schema）を使用します。Oracle XML DB の属性によって、XML の構造化とデータベース・スキーマに対する XML のマッピングの両方を決定するメタデータが指定されます。XML Schema を登録し、適切な XML Schema の URL を使用して、XMLType 表および列を作成しながら、XMLType ビューを定義することもできます。次の章または付録を参照してください。 ■ 第 5 章「XMLType の構造化されたマッピング」 ■ 付録 B「XML Schema の手引き」
XPath	XML 文書の一部をアドレス指定するための言語で、XSLT および XPointer で使用されます。XPath では、ディレクトリ検索構文を使用して XML 文書が検索されます。これには、検索するノード上の述語式を指定する構文も含まれます。XPath 検索の結果は XML フラグメントになります。付録 C「XPath および Namespace の手引き」を参照してください。
XSL	XML 文書を HTML や XML などの言語に変換するために使用されるスタイルシート言語です。付録 D「XSLT の手引き」を参照してください。
DOM	HTML ドキュメントおよび XML 文書に対する Application Program Interface（API）です。ドキュメントの論理構造、およびドキュメントへのアクセス方法および操作方法を定義します。DOM 仕様では、「ドキュメント」は広い意味で使用される用語です。 XML は、様々なシステムに格納される様々な情報を表す方法として、ますます需要が高まっています。従来、この情報のほとんどは、ドキュメントではなくデータとして考えられてきましたが、XML では、データをドキュメントとして表し、このデータを管理するために DOM が使用されます。 DOM を使用すると、ドキュメントを構築したり、それらの構造をナビゲートしたり、要素およびコンテンツを追加、変更または削除することができます。いくつかの例外はありますが、HTML ドキュメントまたは XML 文書内のすべてのものは、DOM を使用してアクセス、変更、削除または追加できます。DOM は、すべてのプログラミング言語で使用できるように設計されています。 Oracle XML DB では、PL/SQL DOM、Java DOM および C DOM（OCI クライアント用）を含む様々なクライアント API を使用して XMLType インスタンス上で操作するために、DOM API が実装されます。次の章を参照してください。 ■ 第 5 章「XMLType の構造化されたマッピング」 ■ 第 8 章「PL/SQL API for XMLType」

表 1-2 このマニュアルで使用する用語（続き）

このマニュアルで使用する用語	説明
Oracle XML DB Repository	第 3 章「Oracle XML DB の使用」を参照してください。
リソース	URL で識別されるオブジェクトです。HTTP および WebDAV 標準に準拠して、`displayname` や `creationdate` などの一連のシステム・プロパティを持ちます。常に、リソースは参照件数を保持し、それにバインドされた最後の URL が削除された場合に、関連付けられたすべてのデータを廃棄します。リソースは、アクセス制御リスト（ACL）および所有者を保持します。Oracle XML DB リソースは、これらのプロパティを含むパス名にマップされた `XMLType` です。第 15 章「RESOURCE_VIEW および PATH_VIEW」を参照してください。
リポジトリ	Oracle XML DB Repository は、すべての Oracle XML DB リソースのセットです。リポジトリは、階層的に編成された一連の `XMLType` オブジェクトであり、それぞれがオブジェクトを識別するパス名を持ちます。Oracle XML DB Repository は、ファイルではなくオブジェクトのファイル・システムです。このリポジトリには、それぞれがパス名を持つリソース・セットを含む 1 つのルート（「/」）があります。その他のリソースを含むリソースは、フォルダといいます（次の「フォルダ」を参照）。 Oracle XML DB オブジェクトには、多くのパス名を指定できます（リソースは複数のフォルダに格納できます）。すべてのデータベース・オブジェクトはパス名にマップできるため、データベース自体がリポジトリと考えられます。ただし、Oracle XML DB はリポジトリを使用して、すべてのスキーマでパス名にマップされたデータベース・オブジェクト・セットを参照します。第 13 章「Oracle XML DB Foldering」を参照してください。
フォルダ	Oracle XML DB Repository 内の非リーフ・ノード・オブジェクト、またはそのようなノードになる可能性があるオブジェクトです。Oracle XML DB には、最適化の理由で、コレクションに特別な格納セマンティクスがあります。Oracle XML DB は、コレクションの階層をナビゲートするために使用される特別な階層索引をメンテナンスし、階層内のパス名を形成するために使用される `name` というプロパティを定義します。コレクションには、フォルダやディレクトリなど多くの名前があります。すべての XML 要素型は、Oracle XML DB スキーマ内で `isFolder` 属性を指定することによって、フォルダにすることができます。第 13 章「Oracle XML DB Foldering」を参照してください。
パス名	階層的な名前は、ルート要素（最初の /）、要素セパレータ（/）および様々なサブ要素（またはパス要素）で構成されます。パス要素は、後続のバックスラッシュ（\）またはスラッシュ（/）を除いて、データベース・キャラクタ・セットのすべての文字で構成できます。Oracle XML DB では、スラッシュは、パス名内のデフォルトの名前セパレータです。
リソース名	ここでのリソースは、Oracle XML DB Repository に格納されたすべてのデータベース・オブジェクトを意味します。リソース名は、親フォルダ内のリソースの名前です。リソース名は、パス要素（フォルダ内のファイル名）です。リソース名は、フォルダ内で一意である必要があります（大 / 小文字が区別されない場合もあります）。

表 1-2 このマニュアルで使用する用語（続き）

このマニュアルで使用する用語	説明
コンテンツ	リソースの本体は、ファイルのようなリソースを処理する場合にそのコンテンツを要求すると戻されるものです。
XDBBinary	バイナリ・データを含む Oracle XML DB スキーマで定義された XML 要素です。XDBBinary 要素は、完全な非構造化バイナリ・データが Oracle XML DB にアップロードされた場合に、リポジトリに格納されます。
ACL 用語	第 18 章「Oracle XML DB リソースのセキュリティ」を参照してください。
アクセス制御リスト（ACL）	オブジェクトへのアクセスを制限します。Oracle XML DB は、ACL を使用して任意の Oracle XML DB リソース（Oracle XML DB ファイル・システムの階層にマップされたすべての XMLType オブジェクト）へのアクセスを制限します。
プロトコル用語	第 19 章「FTP、HTTP および WebDAV プロトコルの 使用」および第 3 章「Oracle XML DB の使用」を参照してください。
FTP	ファイル転送プロトコルです。RFC959 でインターネット標準（STD009）として定義されています。Oracle XML DB は、この標準を実装します。FTP は、オペレーティング・システム・レベルの専用クライアント、ファイル・システムのエクスプローラ・クライアントおよびブラウザで実装されています。一般的に、FTP は、バルク・ファイルのアップロードおよびダウンロード、リポジトリのメンテナンス・スクリプトの作成に使用されます。FTP は、HTTP に類似したモードで使用でき、「非アクティブ」モードで、ブラウザから頻繁にセッションを確立または終了できます。
HTTP	HyperText Transfer Protocol の省略形です。Oracle XML DB は、RFC2616 に定義されている HTTP 1.1 を実装します。今回のリリースでは、Oracle XML DB は、Cookie、Basic 認証および HTTP/1.1（RFC2616、2109 および 2965）を実装します。
WebDAV	Web Distributed Authoring and Versioning の省略形です。今回のリリースでは、Oracle XML DB は、RFC2518 およびアクセス制御をサポートします。
サーブレット	Sun 社は、プロトコル・リクエストの結果として Java コードを起動したり、リクエストに対してパラメータを渡すために、広範囲に使用される標準を開発しました。サーブレットは、HTTP で最も一般的に実装されています。多くの Java サービスは、JSP（JavaServer Pages）や SOAP（Simple Object Access Protocol）などの（Java で実装された）メカニズムを介して、サーブレットとして実装されています。そのため、サーブレットは、Web アプリケーション開発の大部分を占めるアーキテクチャの基礎を形成します。 Oracle XML DB は、Oracle Services（Net Services）以外のプロトコル上で Java ストアド・プロシージャを起動するためのメソッドを提供します。Oracle XML DB は、多くのサーブレット標準を実装します。第 20 章「Java での Oracle XML DB アプリケーションの作成」を参照してください。

このマニュアルで使用する Oracle XML DB の例

このマニュアルには、Oracle の XML および XMLType の使用例を記載しています。例は、多くのデータベース・スキーマ、サンプル XML 文書およびサンプル XML Schema に基づいています。多くの場合、各章の例を使用して例のインフラストラクチャを説明します。

参照：

- 付録 G「設定スクリプトの例および Oracle XML DB によって提供される XML Schema」を参照してください。
- 更新された例のリストは、http://otn.oracle.com/tech/xml/doc.html を参照してください。

2

Oracle XML DB を使用する前に

この章では、Oracle XML DB のソリューションを計画する場合に考慮する基本的な設計基準について説明します。この章の内容は次のとおりです。

Oracle XML DB を使用する前に

Oracle XML DB のインストール

Oracle XML DB は、Oracle9*i* データベース リリース 2（9.2）データベースに付属する汎用データベースの一部としてインストールされます。Oracle XML DB の手動インストールまたは削除の詳細は、付録 A「Oracle XML DB のインストールおよび構成」を参照してください。

Oracle XML DB を使用する場合

Oracle XML DB は、アプリケーションで処理される一部またはすべてのデータが XML で表現されているすべてのアプリケーションに適しています。Oracle XML DB では、XML データの収集、格納、処理および取出しが高パフォーマンスで実行されます。また、既存のリレーショナル・データから XML を迅速かつ簡単に生成できます。

Oracle XML DB が特に適しているアプリケーションには、次のものが含まれます。

- Business-to-Business（B2B）および Application-to-Application（A2A）統合
- インターネット・アプリケーション
- コンテンツ管理アプリケーション
- メッセージ機能
- Web サービス

典型的な Oracle XML DB アプリケーションには、次のうち 1 つ以上の要件および特長があります。

- 多くの XML 文書を収集または生成する必要がある。
- 大規模な XML 文書を処理または生成する必要がある。
- 文書内と大規模な文書コレクション間の両方で、検索が高パフォーマンスで実行される。
- 高いレベルのセキュリティが得られる。セキュリティのファイングレイン・コントロールが実行される。
- データ処理は XML 文書に含まれ、データは従来のリレーショナル表に含まれる必要がある。
- SQL、XML、XPath、XSLT などのオープン標準をサポートする Java などの言語が使用される。
- FTP、HTTP、WebDAV、JDBC などの標準インターネット・プロトコルを使用して情報にアクセスする。
- SQL を介して完全に問合せ可能であり、分析機能と統合されている。
- XML 文書の検証が重要。

XML アプリケーションの設計

Oracle XML DB を使用すると、Oracle9*i* データベース内での XML 文書の格納および処理手順を微調整できます。開発しているアプリケーションの性質に応じて、XML 記憶域には次のうち 1 つ以上の機能が必要です。

- XML 文書の高パフォーマンスな収集および取出し
- XML 文書の高パフォーマンスな索引付けおよび検索
- XML 文書のセクションの更新
- 構造化または非構造化（あるいはその両方）の XML 文書の高度な管理

Oracle XML DB の設計の問題 : 概要

この項では、Oracle XML DB アプリケーションの計画時に考慮できる基本的な設計基準について説明します。図 2-1 に、Oracle XML DB アプリケーションの構築のための主な設計オプションの概要を示します。

a. データ

データは高度に構造化されている（主に XML）か、半構造化（疑似構造化）されているか、非構造化であるか？高度に構造化されている場合、表は XML Schema に基づくか、XML Schema に基づかないか？2-5 ページの「Oracle XML DB アプリケーションの設計 : a. データの構造化」および第 3 章「Oracle XML DB の使用」を参照してください。

b. アクセス

他のアプリケーションおよびユーザーが、どのように XML およびその他のデータにアクセスするか？アクセスにどの程度のセキュリティ・レベルが必要か？バージョニングが必要か？2-7 ページの「Oracle XML DB アプリケーションの設計 : b. アクセス・モデル」を参照してください。

c. アプリケーション言語

アプリケーションはどのプログラミング言語で作成するか？2-8 ページの「Oracle XML DB アプリケーションの設計 : c. アプリケーション言語」を参照してください。

d. 処理

XML を生成する必要があるか？第 10 章「データベースからの XML データの生成」を参照してください。

XML 文書はどれほどの頻度でアクセス、更新および操作されるか？更新の対象は文書のフラグメントか、または文書全体か？

XML を HTML や WML などの他の言語に変換する必要があるか？また、アプリケーションによって XML がどのように変換されるか？第 6 章「XMLType データの変換および検証」を参照してください。

アプリケーションを主にデータベースに常駐させる必要があるか？またはデータベースと中間層の両方で動作させる必要があるか？

アプリケーションは、データ中心、ドキュメント / コンテンツ中心、またはその統合（データ中心かつドキュメント中心）のいずれであるか？2-9 ページの「Oracle XML DB アプリケーションの設計 : d. 処理モデル」を参照してください。

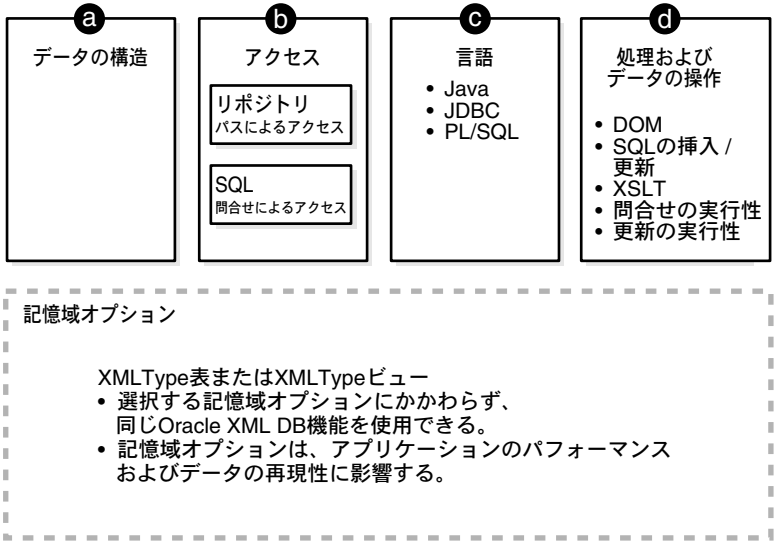
ゲートウェイを介して、XML データを他のアプリケーションと交換するか？アドバンスト・キューイング（AQ）または SOAP に準拠させる必要があるか？第 24 章「Advanced Queuing（AQ）および Oracle Streams を使用した XML データの変換」を参照してください。

記憶域

データ、XML データ、XML Schema などをどの方法でどの場所に格納するか？2-10 ページの「Oracle XML DB の設計 : 記憶域モデル」を参照してください。

注意： 通常、前述の a ～ d の 4 つのカテゴリで選択するモデルは、相互に関連しています。ただし、記憶域モデルの選択は、他の設計モデルの選択とは関連がありません。つまり、他の設計モデル・オプションに対する選択は、選択した記憶域モデル・オプションに依存しません。

図 2-1 Oracle XML DB の設計オプション



Oracle XML DB アプリケーションの設計 : a. データの構造化

図 2-2 に、次のデータ構造のカテゴリおよび推奨する関連記憶域オプションを示します。

- **構造化データ :** データが高度に構造化されているか ? つまり、データが主に XML データか ?
- **半構造化または疑似構造化データ :** データが半構造化または疑似構造化されているか ? つまり、いくつかの XML データが含まれているか ?
- **非構造化データ :** データが非構造化データであるか ? データが主に非 XML データか ?

XML Schema に基づく場合または XML Schema に基づかない場合

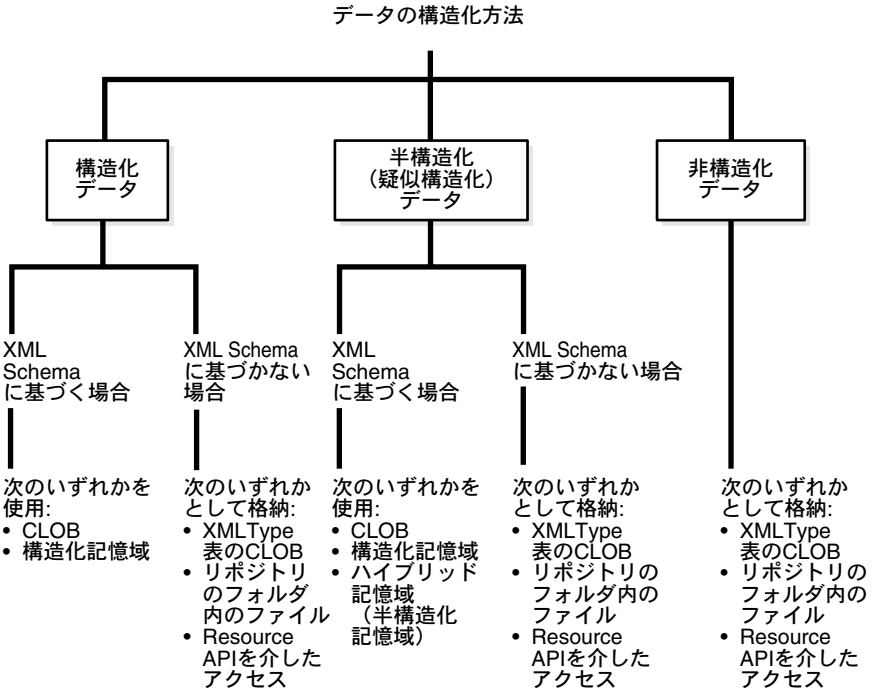
次のデータ・モデリングの問題も考慮します。

- アプリケーションが XML Schema に基づく場合 :
  - 構造化データでは、キャラクタ・ラージ・オブジェクト（CLOB）または構造化記憶域のいずれかを使用できます。
  - 半構造化または疑似構造化データでは、CLOB、構造化記憶域またはハイブリッド記憶域のいずれかを使用できます。この場合、XML Schema はより疎結合になります。2-10 ページの「Oracle XML DB の設計 : 記憶域モデル」も参照してください。

– 非構造化データでは、XML Schema 設計は適用されません。

- アプリケーションが XML Schema に基づかない場合 : 構造化、半構造化、疑似構造化および非構造化データでは、データを XMLType 表またはビュー内の CLOB、またはリポジトリのフォルダ内のファイルのいずれかに格納できます。この設計では、リソース・ビューを介したパスベースや問合せベースのアクセスを含む、多くのアクセス・オプションを選択できます。

図 2-2 データ記憶域モデル : データの構造化



Oracle XML DB アプリケーションの設計 : b. アクセス・モデル

図 2-3 に、Oracle XML DB アプリケーションの設計時に考慮する 2 つの主なデータ・アクセス・モードを示します。

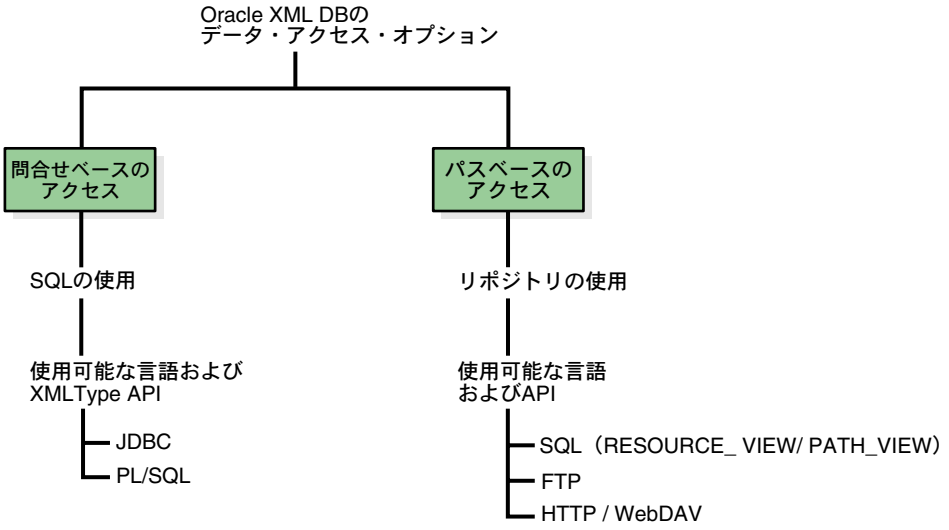
- **ナビゲーションベースまたはパスベースのアクセス :** これは、コンテンツ / ドキュメント指向とデータ指向アプリケーションの両方に適しています。Oracle XML DB では、次の言語およびアクセス API が提供されています。
  - リソース・ビューまたはパス・ビューを介した SQL アクセス。第 15 章「RESOURCE_VIEW および PATH_VIEW」を参照してください。
  - DBMS_XDB を介した PL/SQL アクセス。第 16 章「Oracle XML DB Resource API for PL/SQL（DBMS_XDB）」を参照してください。
  - Java アクセス。第 17 章「Oracle XML DB Resource API for Java」を参照してください。
  - コンテンツ指向のアプリケーションに最適な HTTP、WebDAV または FTP を使用したプロトコルベースのアクセス。第 19 章「FTP、HTTP および WebDAV プロトコルの 使用」を参照してください。
- **問合せベースのアクセス :** これは、データ指向アプリケーションに最適です。Oracle XML DB では、SQL 問合せを使用する、次の API を介したアクセスが提供されています。
  - Java（JDBC を介した）アクセス。第 9 章「Java API for XMLType」を参照してください。
  - PL/SQL アクセス。第 8 章「PL/SQL API for XMLType」を参照してください。

リポジトリ・データにアクセスするためのこれらのオプションについては、第 13 章「Oracle XML DB Foldering」を参照してください。

次のアクセス・モデル基準についても考慮します。

- どの程度のセキュリティ・レベルが必要か？第 18 章「Oracle XML DB リソースのセキュリティ」を参照してください。
- アプリケーションに最適な索引付けの形式は何か？Oracle Text の索引付けおよび問合せを使用する必要があるか？第 4 章「XMLType の使用」および第 7 章「Oracle Text を使用した XML データの検索」を参照してください。
- データのバージョニングが必要か？必要な場合、第 14 章「Oracle XML DB Versioning」を参照してください。

図 2-3 データ・アクセス・モデル : ユーザーまたはアプリケーションがデータにアクセスする方法



Oracle XML DB アプリケーションの設計 : c. アプリケーション言語

Oracle XML DB アプリケーションは、次の言語でプログラミングできます。

- Java（JDBC、Java サーブレット）

  参照 : 次の章または付録を参照してください。

  - 第 9 章「Java API for XMLType」
  - 第 17 章「Oracle XML DB Resource API for Java」
  - 第 20 章「Java での Oracle XML DB アプリケーションの作成」
  - 付録 E「Java DOM API for XMLType および Resource API for Java: クイック・リファレンス」

- PL/SQL

  参照 : 次の章または付録を参照してください。

  - 第 8 章「PL/SQL API for XMLType」
  - 第 16 章「Oracle XML DB Resource API for PL/SQL（DBMS_XDB）」
  - 付録 F「Oracle XML DB の XMLType API、PL/SQL API および Resource PL/SQL API: クイック・リファレンス」

Oracle XML DB アプリケーションの設計 : d. 処理モデル

次の処理オプションは、Oracle XML DB アプリケーションの設計時に使用可能であり、考慮する必要があります。

- XSLT: XML を HTML や WML などの他の言語に変換する必要があるか ? また、アプリケーションによって XML がどのように変換されるか ? Oracle XML DB に XML 文書を格納中に、構造が特定の XML Schema に準拠するかどうか（妥当かどうか）をオプションで確認できます。第 6 章「XMLType データの変換および検証」を参照してください。
- DOM: 第 8 章「PL/SQL API for XMLType」を参照してください。オブジェクト・リレーショナル列、VARRAY、ネストした表および LOB を使用して、DOM 再現性を保持（格納された DOM が取り出される DOM と同じ）したまま、XML Schema に任意の要素または要素のサブツリーを格納します。Oracle9*i* データベース リリース 1（9.0.1）以上で `XMLType` に使用可能な CLOB 記憶域オプションを使用すると、空白を保持できます。XML Schema を使用する場合の DOM 再現性については、第 5 章「XMLType の構造化されたマッピング」を参照してください。
- XPath 検索 : XPath 構文（SQL 文に埋め込まれた構文、または HTTP リクエストの一部）を使用して、データベース内の XML コンテンツを問い合せることができます。第 4 章「XMLType の使用」、第 7 章「Oracle Text を使用した XML データの検索」、第 13 章「Oracle XML DB Foldering」および第 15 章「RESOURCE_VIEW および PATH_VIEW」を参照してください。
- XML の生成および `XMLType` ビュー : XML を生成または再生成する必要があるか ? 必要な場合は、第 10 章「データベースからの XML データの生成」を参照してください。

XML 文書はどれほどの頻度でアクセス、更新および操作されるか ? 第 4 章「XMLType の使用」および第 15 章「RESOURCE_VIEW および PATH_VIEW」を参照してください。

更新の対象は文書のフラグメントか、文書全体か ? XPath を使用して、文書全体を再書込みせずに、更新中に文書の個々の要素および属性を指定できます。これは、特に大規模な XML 文書の場合により効率的です。第 5 章「XMLType の構造化されたマッピング」.

アプリケーションは、データ中心、ドキュメント / コンテンツ中心、またはその統合（データ中心かつドキュメント中心）のいずれであるか ? 第 3 章「Oracle XML DB の使用」を参照してください。

メッセージ機能オプション

アドバンスト・キューイング（AQ）では、XML および XMLType アプリケーションがサポートされています。XMLType 属性を含むペイロードを持つキューを作成できます。これらのペイロードは、XML 文書を含むメッセージの転送および格納に使用できます。XMLType 属性を持つ Oracle オブジェクトを定義すると、次の操作を実行できます。

- 複数の型の XML 文書を同じキュー内に格納します。この文書は、CLOB として内部的に格納されます。
- existsNode() や extract() などの演算子を使用して、XMLType 属性を持つメッセージを選択的にデキューします。
- 変換を定義して、Oracle オブジェクトを XMLType に変換します。
- existsNode() や extract() などの XMLType 演算子を使用して、メッセージ内容を問い合せるルールベースのサブスクライバを定義します。

参照 : 次の章またはマニュアルを参照してください。

- 第 24 章「Advanced Queuing（AQ）および Oracle Streams を使用した XML データの変換」
- 『Oracle9*i* アプリケーション開発者ガイド - アドバンスト・キューイング』

Oracle XML DB の設計 : 記憶域モデル

図 2-4 に、XMLType 表またはビューの使用に関連する Oracle XML DB の記憶域オプションの概要を示します。既存のリレーショナル・データまたはレガシー・リレーショナル・データが存在する場合は、XMLType ビューを使用します。

Oracle XML DB アプリケーションに選択した記憶域オプションにかかわらず、Oracle XML DB では同じ機能が提供されます。ただし、選択したオプションは、アプリケーションのパフォーマンスおよびデータの再現性（データの精度）に影響します。

現在、Oracle XML DB アプリケーションには、主に次の 3 つの記憶域オプションがあります。

- **LOB ベースの記憶域 :** LOB ベースの記憶域では、空白を含むテキストの完全な再現性が保証されます。これは、XML 文書を CLOB に格納した場合、XML 文書を取り出したときにデータが損失しないことを意味します。データの整合性は高く、再生成のコストは低くなります。
- **構造化記憶域 :** 構造化記憶域では、空白情報は失われますが、XML DOM 再現性（格納された DOM は取り出される DOM と同じであること）は保持されます。これには、次のメリットがあります。
  - SQL 問合せの実行性およびパフォーマンスが向上します。
  - ピース単位更新が可能になります。

- **ハイブリッドまたは半構造化記憶域 :** ハイブリッド記憶域は特別な構造化記憶域で、XML データの一部が構造化された形式に分割され、残りのデータが CLOB として格納されます。

記憶域オプションは、次の基準とは関係がありません。

- データの問合せの実行性および更新の実行性（データの問合せおよび更新方法および頻度）。
- データのアクセス方法。これは、アプリケーション処理の要件によって決まります。
- アプリケーションで使用される言語。これも、アプリケーション処理の要件によって決まります。

参照 : 次の項を参照してください。

- 3-23 ページの「XML の格納 : 構造化記憶域または非構造化記憶域」、3-27 ページの「構造化記憶域 : XMLType の XML Schema に基づく記憶域」および 3-38 ページの「リソース用の記憶域オプション」
- 4-4 ページの「Oracle XML DB への XMLType データの格納」
- 5-19 ページの「DOM 再現性」

XMLType 表の使用

XMLType 表を使用している場合、データを次の記憶域に格納できます。

- CLOB（非構造化）記憶域
- 構造化記憶域
- ハイブリッドまたは半構造化記憶域

XMLType ビューの使用

既存のリレーショナル・データが存在する場合、XMLType ビューを使用します。次のオプションを使用して、XMLType ビューを定義できます。

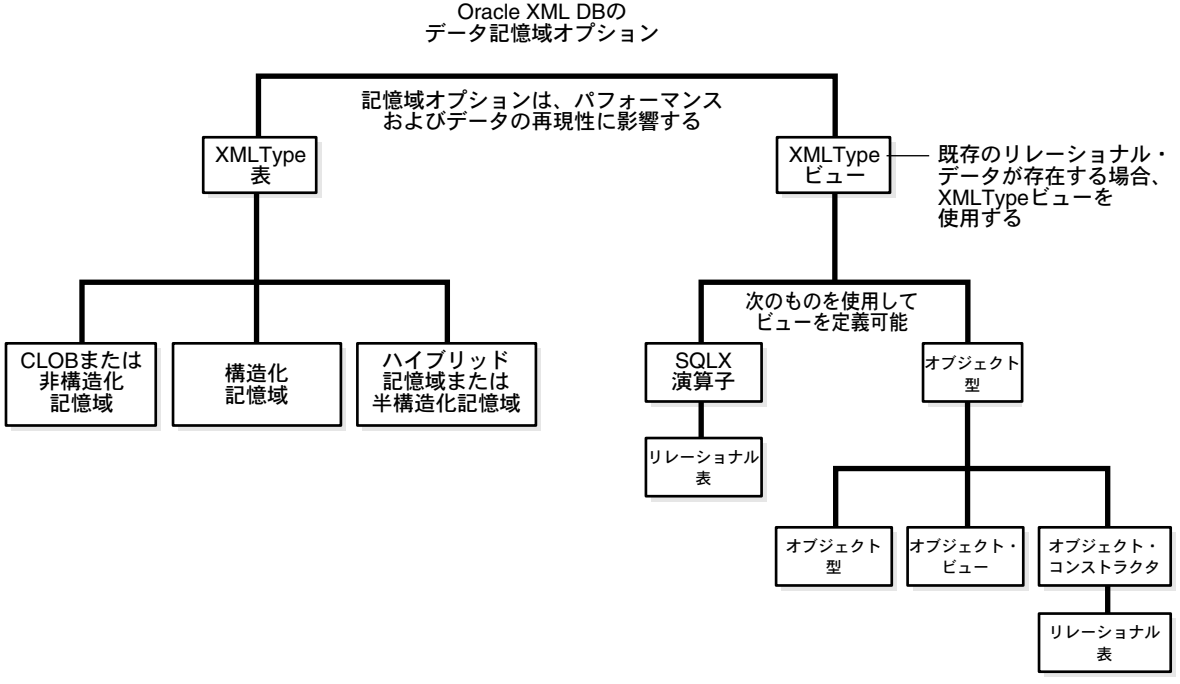
- SQLX 演算子

  これらの演算子を使用して、データをリレーショナル表に格納し、XML を生成および再生成することもできます。第 10 章「データベースからの XML データの生成」を参照してください。

- オブジェクト型
  - オブジェクト表
  - オブジェクト・コンストラクタ

    オブジェクト・コンストラクタを使用して、データをリレーショナル表に格納できます。
  - オブジェクト・ビュー

図 2-4 構造化記憶域オプション



3

Oracle XML DB の使用

この章では、Oracle XML DB の使用可能な場所および使用方法について説明します。また、XMLType データの格納やアクセスを含む Oracle XML DB の一般的な使用例、データの更新と検証、および XPath と XML Schema の必要性についても説明します。また、リポジトリを使用して、様々なクライアントから標準プロトコルを介してデータベース・データを格納、アクセスおよび操作する方法も示します。

また、XML Schema に基づく文書を格納したり、XDBUriType を使用して XML Schema に基づかないコンテンツにアクセスするためのデフォルトの XML 表の定義方法も説明します。

この章の内容は次のとおりです。

- XML Schema に基づく文書に対するデフォルト表の記憶域の定義
- XML Schema に基づくコンテンツへのアクセス
- XDBUriType を使用した XML Schema に基づかないコンテンツへのアクセス
- Oracle XML DB Protocol Server

参照： Oracle XML DB の使用方法および使用可能な場所の例については、次の章を参照してください。

- 第 25 章「Oracle XML DB の事例 : Web サービスによる XML 文書の取出しおよび表示」
- 第 26 章「Oracle XML DB Basic Demo」

XMLType 列または XMLType 表へのデータの格納

XML 文書を Oracle9*i* データベースに格納するには、次のような方法があります。

- XML 文書を Oracle9*i* データベース以外で解析し、XML 文書のデータを 1 つ以上の表に行として格納します。この場合、データベースには XML コンテンツを管理しているという認識はありません。
- CLOB 列または VARCHAR 列を使用して、XML 文書を Oracle9*i* データベースに格納します。この場合でも、データベースには XML コンテンツを管理しているという認識はありませんが、プログラムによって XDK を使用して、XML 操作を実行できます。
- XMLType データ型を使用して、XML 文書を Oracle9*i* データベースに格納します。この場合、次の 2 つのオプションを使用できます。
  - XML 文書を XMLType 列に格納する
  - XMLType 表を使用して XML 文書を格納する

  これらのオプションは、データベースが XML コンテンツを管理していることを認識していることを意味します。この方法を選択すると、データベースによって XML コンテンツを効率的に処理できる一連の機能が提供されるため、多くのメリットがあります。

例 3-1 XMLType 列を含む表の作成

```
CREATE TABLE Example1
(
    KEYVALUE  varchar2(10)  primary key,
    XMLCOLUMN xmltype
);
```

例 3-2 XMLType 表の作成

```
CREATE TABLE XMLTABLE OF XMLType;
```

例 3-3 XML 文書を格納するための getDocument() を使用した XMLType インスタンスの作成

XML 文書を XMLType 表または列に格納するには、まず XML 文書を XMLType インスタンスに変換する必要があります。これは、XMLType データ型によって提供される様々なコンストラクタを使用して行われます。たとえば、getCLOBDocument() という PL/SQL ファンクションについて考えてみます。

```
create or replace function getClobDocument(
filename in varchar2,
 charset in varchar2 default NULL)
 return CLOB deterministic
 is
    file            bfile := bfilename('DIR',filename);
    charContent     CLOB := ' ';
```

```
    targetFile      bfile;
    lang_ctx        number := DBMS_LOB.default_lang_ctx;
    charset_id      number := 0;
    src_offset      number := 1 ;
    dst_offset      number := 1 ;
    warning         number;
 begin
   if charset is not null then
       charset_id := NLS_CHARSET_ID(charset);
   end if;
   targetFile := file;
   DBMS_LOB.fileopen(targetFile, DBMS_LOB.file_readonly);
   DBMS_LOB.LOADCLOBFROMFILE(charContent, targetFile,
           DBMS_LOB.getLength(targetFile), src_offset, dst_offset,
           charset_id, lang_ctx,warning);
   DBMS_LOB.fileclose(targetFile);
   return charContent;
 end;
 /
-- create XMLDIR directory
-- connect system/manager
-- create directory XMLDIR as '<location_of_xmlfiles_on_server>';
-- grant read on directory xmldir to public with grant option;

-- you can use getCLOBDocument() to generate a CLOB from a file containin
-- an XML document. For example, the following statement inserts a row into the
-- XMLType table Example2 created earlier:

INSERT INTO XMLTABLE
VALUES(XMLTYPE(getCLOBDocument('purchaseorder.xml')));
```

「charset」というパラメータが使用されていることに注意してください。このパラメータは、指定されたファイルのキャラクタ・セットを識別するために使用されます。このパラメータが省略されている場合、現行のデータベースのデフォルトのキャラクタ・セット ID が使用されます。

たとえば、invoice.xml というファイルで韓国語キャラクタ・セットの 1 つである KO16KSC5601 が使用されている場合、次のとおり XMLDOC という XMLType 表にロードできます。

```
insert into xmldoc
   values(xmltype(getClobDocument('invoice.xml','KO16KSC5601')));
```

次の例では、UTF8 ファイル形式を使用します。

```
insert into xmldoc
   values(xmltype(getClobDocument('invoice.xml','UTF8')));
```

次の例では、データベースとファイルの両方で、同じキャラクタ・セット（UTF8 など）を使用します。

```
insert into xmldoc values(xmltype(getClobDocument('invoice.xml')));
```

> **注意：** Oracle XML DB は、クライアントのキャラクタ・セットがデータベースのキャラクタ・セットと同じである場合、マルチバイト・キャラクタを処理できます。

XMLType 列または XMLType 表のデータへのアクセス

XML 文書のコレクションを XMLType 表または列として格納した後、それらの文書を取り出すことができるようにします。XML 文書のコレクションを使用する場合、次の 2 つの基本タスクを実行する必要があります。

- 使用可能な文書のサブセットを選択する方法の決定
- 文書内に含まれるノードの特定のサブセットにアクセスする最適な方法の判断

Oracle9*i* データベースおよび XMLType データ型は、これらのタスクを簡単に実行できる多数の関数を提供します。これらの関数は、W3C の XPath 勧告を使用して、XML 文書のコレクション全体およびコレクション内をナビゲートします。

> **参照：** W3C の XPath 勧告については、付録 C「XPath および Namespace の手引き」を参照してください。

Oracle XML DB での XPath の使用

Oracle XML DB によって提供される多くの関数は、W3C の XPath 勧告に基づいています。スラッシュで区切った要素名および属性名のリストによってナビゲートする要素を指定すると、XPath はネストした XML 要素を全検索します。XPath を使用して、XML 文書内および XML 文書間での問合せを定義します。Oracle XML DB では、標準に準拠した一般的な方法で、XML 文書に対する階層問合せを表現できます。

Oracle XML DB では、XPath は、主に extract()、extractValue() および existsNode() 関数とともに使用されます。

existsNode() 関数は、指定された文書に W3C の XPath 式と一致するノードが含まれているかどうかを評価します。existsNode() 関数は、関数に指定された XPath 式によって指定されたノードが文書に含まれている場合、TRUE（1）を戻します。また、existsNode() 関数によって提供されるこの機能は、XMLType データ型の existsNode() メソッドを介しても使用できます。

参照： 次の章を参照してください。

- 第 4 章「XMLType の使用」
- 第 10 章「データベースからの XML データの生成」

XML 文書 PurchaseOrder

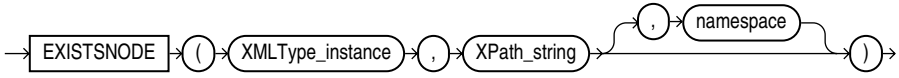
この項では、次の XML 文書 PurchaseOrder を使用した例を示します。

```
<PurchaseOrder
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="http://www.oracle.com/xdb/po.xsd">
  <Reference>ADAMS-20011127121040988PST</Reference>
  <Actions>
    <Action>
      <User>SCOTT</User>
      <Date>2002-03-31</Date>
    </Action>
  </Actions>
  <Reject/>
  <Requestor>Julie P. Adams</Requestor>
  <User>ADAMS</User>
  <CostCenter>R20</CostCenter>
  <ShippingInstructions>
    <name>Julie P. Adams</name>
    <address>Redwood Shores, CA 94065</address>
    <telephone>650 506 7300</telephone>
  </ShippingInstructions>
  <SpecialInstructions>Ground</SpecialInstructions>
  <LineItems>
    <LineItem ItemNumber="1">
      <Description>The Ruling Class</Description>
      <Part Id="715515012423" UnitPrice="39.95" Quantity="2"/>
    </LineItem>
    <LineItem ItemNumber="2">
      <Description>Diabolique</Description>
      <Part Id="037429135020" UnitPrice="29.95" Quantity="3"/>
    </LineItem>
    <LineItem ItemNumber="3">
      <Description>8 1/2</Description>
      <Part Id="037429135624" UnitPrice="39.95" Quantity="4"/>
    </LineItem>
  </LineItems>
</PurchaseOrder>
```

existsNode() の使用

図 3-1 に、existsNode() の構文を示します。

図 3-1 existsNode() の構文



例 3-4 XPath 式と一致するノードを検索する existsNode() の例

この XML 文書の例では、次の existsNode() 演算子は TRUE（1）を戻します。

```
SELECT existsNode(value(X),'/PurchaseOrder/Reference')
   FROM XMLTABLE X;

SELECT existsNode(value(X),
       '/PurchaseOrder[Reference="ADAMS-20011127121040988PST"]')
   FROM XMLTABLE X;

SELECT existsNode(value(X),
       '/PurchaseOrder/LineItems/LineItem[2]/Part[@Id="037429135020"]')
   FROM XMLTABLE X;

SELECT existsNode(value(X),
       '/PurchaseOrder/LineItems/LineItem[Description="8 1/2"]')
   FROM XMLTABLE X;
```

例 3-5 XPath 式と一致するノードを検索しない existsNode() の例

次の existsNode() 演算子は、XPath 式と一致するノードを検索せず、すべての場合に FALSE（0）を戻します。

```
SELECT existsNode(value(X),'/PurchaseOrder/UserName')
   FROM XMLTABLE X;

SELECT existsNode(value(X),
       '/PurchaseOrder[Reference="ADAMS-XXXXXXXXXXXXXXXXXXXX"]')
   FROM XMLTABLE X;

SELECT existsNode(value(X),
       '/PurchaseOrder/LineItems/LineItem[3]/Part[@Id="037429135020"]')
   FROM XMLTABLE X;

SELECT existsNode(value(X),
       '/PurchaseOrder/LineItems/LineItem[Description="Snow White"]')
   FROM XMLTABLE X;
```

existsNode() の最も一般的な使用方法は、SQL の SELECT 文、UPDATE 文または DELETE 文の WHERE 句で使用することです。この場合、existsNode() 関数に渡された XPath 式は、表に格納されたどの XML 文書が SQL 文によって処理されるかを決定するために使用されます。

例 3-6 WHERE 句での existsNode() の使用

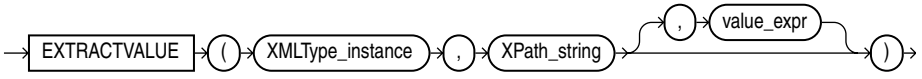
```
SELECT count(*)
   FROM XMLTABLE x
   WHERE existsNode(value(x),'/PurchaseOrder[User="ADAMS"]') = 1;

DELETE FROM XMLTABLE x
   WHERE existsNode(value(x),'/PurchaseOrder[User="ADAMS"]') = 1;
commit;
```

extractValue() 関数は、XMLType として格納された XML 文書から、XPath 式に関連付けられたテキスト・ノードまたは属性の値を戻すために使用されます。この関数は、スカラー・データ型を戻します。

extractValue() の使用

図 3-2 に、extractValue() の構文を示します。

図 3-2 extractValue() の構文



次に、extractValue() の例を示します。

例 3-7 extractValue() の有効な使用

```
SELECT extractValue(value(x),'/PurchaseOrder/Reference')
  FROM XMLTABLE X;
```

次の結果が戻されます。

```
EXTRACTVALUE(VALUE(X),'/PURCHASEORDER/REFERENCE')
--------------------------------------------------------------------------------
ADAMS-20011127121040988PST

SELECT extractValue(value(x),
       '/PurchaseOrder/LineItems/LineItem[2]/Part/@Id')
  FROM XMLTABLE X;
```

次の結果が戻されます。

```
EXTRACTVALUE(VALUE(X),'/PURCHASEORDER/LINEITEMS/LINEITEM[2]/PART/@ID')
--------------------------------------------------------------------------------
037429135020
```

extractValue() は、単一ノードの値または属性値のみを戻すことができます。たとえば、次の例は、extractValue() を無効な方法で使用しています。最初の例では、XPath 式は文書内の 3 つのノードと一致し、2 つ目の例では、XPath 式は 1 つのテキスト・ノード値または属性値ではなく、1 つのノード・ツリーを識別します。

例 3-8 extractValue() の無効な使用

```
SELECT extractValue(value(X),
                    '/PurchaseOrder/LineItems/LineItem/Description')
   FROM XMLTABLE X;

-- FROM XMLTABLE X;
--       *
-- ERROR at line 3:
-- ORA-19025: EXTRACTVALUE returns value of only one node

SELECT extractValue(value(X),
                    '/PurchaseOrder/LineItems/LineItem[1]')
  FROM XMLTABLE X;

-- FROM XMLTABLE X
--    *
-- ERROR at line 3:
-- ORA-19025: EXTRACTVALUE returns value of only one node
```

例 3-9 WHERE 句での extractValue() の使用

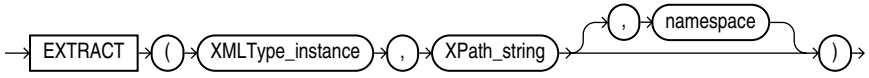
extractValue() も、SELECT 文、UPDATE 文または DELETE 文の WHERE 句で使用できます。これによって、XMLType 表または XMLType 列を含む表と他のリレーショナル表または XMLType 表の間の結合が可能になります。次の問合せでは、SELECT 構文のリストと WHERE 句の両方で extractValue() を使用します。

```
SELECT extractValue(value(x),'/PurchaseOrder/Reference')
    FROM XMLTABLE X, SCOTT.EMP
    WHERE extractValue(value(x),'/PurchaseOrder/User') = EMP.ENAME
    AND EMP.EMPNO = 7876;

-- This returns:
-- EXTRACTVALUE(VALUE(X),'/PURCHASEORDER/REFERENCE')
-- -------------------------------------------------
-- ADAMS-20011127121040988PST
```

extract() の使用

図 3-3 に、extract() の構文を示します。

図 3-3 extract() の構文



extract() は、XPath 式によってノードのコレクションが戻される場合に使用されます。ノードは、XMLType の 1 つのインスタンスとして戻されます。extract() の結果は、XML 文書または XML 文書のフラグメントになります。この抽出機能は、XMLType データ型の extract() メソッドを介して使用することもできます。

例 3-10 XML フラグメントを戻すための extract() の使用

次の extract() 文は、Description ノードが出現する XML 文書のフラグメントを含む XMLType を戻します。これらは、次に示す指定された XPath 式と一致します。

> **注意：** この場合、XML は複数のルート・ノードを含むため、整形式ではありません。

```
set long 20000

SELECT extract(value(X),
               '/PurchaseOrder/LineItems/LineItem/Description')
  FROM XMLTABLE X;

-- This returns:
-- EXTRACT(VALUE(X),'/PURCHASEORDER/LINEITEMS/LINEITEM/DESCRIPTION')
-- ----------------------------------------------------------------
-- <Description>The Ruling Class</Description>
-- <Description>Diabolique</Description>
-- <Description>8 1/2</Description>
```

例 3-11 XPath 式と一致するノード・ツリーを戻すための extract() の使用

この例では、extract() は指定された XPath 式と一致するノード・ツリーを戻します。

```
SELECT extract(value(X),
               '/PurchaseOrder/LineItems/LineItem[1]')
  FROM XMLTABLE X;
```

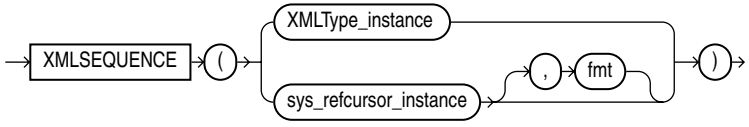
次の結果が戻されます。

```
EXTRACT(VALUE(X),'/PURCHASEORDER/LINEITEMS/LINEITEM[1]')
--------------------------------------------------------------------------------
<LineItem ItemNumber="1">
  <Description>The Ruling Class</Description>
  <Part Id="715515012423" UnitPrice="39.95" Quantity="2"/>
</LineItem>
```

XMLSequence() の使用

図 3-4 に、XMLSequence() の構文を示します。

図 3-4 XMLSequence() の構文



XML 文書のフラグメントは、XMLSequence() 関数を使用して、一連の XMLType に変換できます。XMLSequence() は、1 つの文書のフラグメントを含む XMLType を取り、XMLType オブジェクトのコレクションを戻します。そのコレクションには、フラグメント内のルート・レベル・ノードごとに 1 つの XMLType が含まれます。その後、SQL の TABLE 関数を使用して、コレクションを一連の行に変換できます。

例 3-12 XMLSequence() および TABLE() を使用した XML 文書からの Description ノードの抽出

次の例では、XMLSequence() および Table() を使用して、purchaseorder 文書から Description ノードの集合を抽出します。

```
set long 10000
set feedback on
SELECT extractValue(value(t),'/Description')
  FROM XMLTABLE X,
    TABLE ( xmlsequence (
           extract(value(X),
                 '/PurchaseOrder/LineItems/LineItem/Description')
           )
  ) t;
```

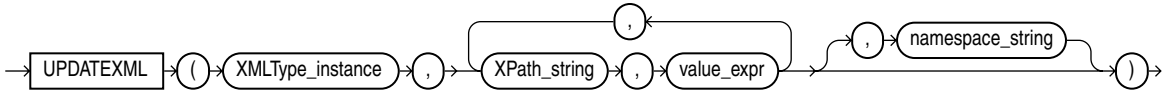
次の結果が戻されます。

```
EXTRACTVALUE(VALUE(T),'/DESCRIPTION')
--------------------------------------------------------------------------------
The Ruling Class
Diabolique
8 1/2
```

updateXML() を使用した XML 文書の更新

図 3-5 に、updateXML() の構文を示します。

図 3-5 updateXML() の構文



XML 文書は、updateXML() 関数を使用して更新できます。updateXML() は、属性値、ノード、テキスト・ノードまたはノード・ツリーを更新します。更新操作のターゲットは、XPath 式を使用して識別されます。次の例では、updateXML() を使用して、XMLType として格納された XML 文書のコンテンツを変更します。

例 3-13 updateXML() を使用した、XPath 式によって識別されるテキスト・ノード値の更新

次の例では、updateXML() を使用して、XPath 式 '/PurchaseOrder/Reference' によって識別されるテキスト・ノードの値を更新します。

```
UPDATE XMLTABLE t
    SET value(t) = updateXML(value(t),
                             '/PurchaseOrder/Reference/text()',
                             'MILLER-20020331120000000PST')
    WHERE existsNode(value(t),
       '/PurchaseOrder[Reference="ADAMS-20011127121040988PST"]') = 1;
```

次の結果が戻されます。

```
1 row updated.

SELECT value(t)
   FROM XMLTABLE t;
```

次の結果が戻されます。

```
VALUE(T)
```

```
--------------------------------------------------------------------------------
<PurchaseOrder xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:noNamespaceSchemaLocation="http://www.oracle.com/xdb/po.xsd">
  <Reference>MILLER-20020331120000000PST</Reference>
...
</PurchaseOrder>
```

例 3-14 updateXML() を使用した、XPath 要素に関連付けられたノード・ツリーのコンテンツの置換

次の例では、updateXML() は、XPath 式 '/PurchaseOrders/LineItems/LineItem[2]' によって識別される要素に関連付けられたノード・ツリーのコンテンツを置換します。

> **注意：** この例では、置換値は 1 つのノード・ツリーであるため、updateXML() 関数の 3 番目の引数は XMLType データ型の 1 つのインスタンスとして指定されます。

```
UPDATE XMLTABLE t
  SET value(t) =
  updateXML(value(t),
            '/PurchaseOrder/LineItems/LineItem[2]',
            xmltype('<LineItem ItemNumber="4">
                       <Description>Andrei Rublev</Description>
                       <Part Id="715515009928" UnitPrice="39.95"
                             Quantity="2"/>
                     </LineItem>'
             )
  )
  WHERE existsNode(value(t),
        '/PurchaseOrder[Reference="MILLER-20020331120000000PST"]'
  ) = 1;
```

次の結果が戻されます。

```
1 row updated.

SELECT value(t)
  FROM XMLTABLE t;
```

次の結果が戻されます。

```
VALUE(T)
--------------------------------------------------------------------------------
<PurchaseOrder xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNames
```

```
paceSchemaLocation="http://www.oracle.com/xdb/po.xsd">
  <Reference>MILLER-20020311200000000PST</Reference>
...
  <LineItems>
    <LineItem ItemNumber="1">
      <Description>The Ruling Class</Description>
      <Part Id="715515012423" UnitPrice="39.95" Quantity="2"/>
    </LineItem>
    <LineItem ItemNumber="4">
      <Description>Andrei Rublev</Description>
      <Part Id="715515009928" UnitPrice="39.95" Quantity="2"/>
    </LineItem>
    <LineItem ItemNumber="3">
      <Description>8 1/2</Description>
      <Part Id="037429135624" UnitPrice="39.95" Quantity="4"/>
    </LineItem>
  </LineItems>
</PurchaseOrder>
```

W3C の XSLT 勧告の概要

W3C の XSLT 勧告は、XML 文書の形式を別の形式に変換する方法を指定するための XML 言語を定義します。変換には、1 つの XML Schema から別の XML Schema へのマッピング、または XML から HTML や WML などの他の形式へのマッピングが含まれる場合があります。

> **参照：** W3C の XSL 勧告および XSLT 勧告の概要は、付録 D「XSLT の手引き」を参照してください。

例 3-15 XSLT スタイルシートの例 : PurchaseOrder.xsl

次の PurchaseOrder.xsl の例は、XSLT スタイルシートのサンプル・フラグメントです。

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns:xdb="http://xmlns.oracle.com/xdb"
                xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <xsl:template match="/">
    <html>
      <head/>
      <body bgcolor="#003333" text="#FFFFCC" link="#FFCC00"
            vlink="#66CC99" alink="#669999">
        <FONT FACE="Arial, Helvetica, sans-serif">
          <xsl:for-each select="PurchaseOrder"/>
          <xsl:for-each select="PurchaseOrder">
            <center>
              <span style="font-family:Arial; font-weight:bold">
                <FONT COLOR="#FF0000">
                  <B>Purchase Order </B>
                </FONT>
              </span>
            </center>
            <br/>
          ...
                        <FONT FACE="Arial, Helvetica, sans-serif"
                              COLOR="#000000">
                          <xsl:for-each select="Part">
                            <xsl:value-of select="@Quantity*@UnitPrice"/>
                          </xsl:for-each>
                        </FONT>
                      </td>
                    </tr>
                  </tbody>
                </xsl:for-each>
              </xsl:for-each>
            </table>
          </xsl:for-each>
        </FONT>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

参照 : この XSLT スタイルシートの完全なリストについては、付録 D「XSLT の手引き」を参照してください。

Oracle XML DB での XSL/XSLT の使用

Oracle XML DB は、データベースでの XSLT 変換をサポートすることによって、W3C の XSL 勧告および XSLT 勧告に準拠しています。Oracle XML DB では、XSLT 変換は次のいずれかを使用して実行できます。

- XMLTransform() 関数
- XMLType データ型の transform() メソッド

XSLT スタイルシートは妥当な XML 文書であるため、XSLT スタイルシートが XMLType データ型のインスタンスとして提供されている場合は、どちらの方法も適用できます。XSLT 変換の結果も、1 つの XMLType として戻されます。

データの変換はメモリー上で行われるため、Oracle XML DB は、変換を実行するために必要なメモリー使用量、I/O 操作、ネットワーク通信量などの機能を最適化できます。

> **参照：** 第 6 章「XMLType データの変換および検証」を参照してください。

例 3-16 transform() を使用した XSL の変換

次の例では、transform() によって XSLT を XSLT スタイルシート PurchaseOrder.xsl に適用して、PurchaseOrder.xml 文書を変換します。

```
SELECT value(t).transform(xmltype(getDocument('purchaseOrder.xsl')))
    from XMLTABLE t
    where existsNode(value(t),
                     '/PurchaseOrder[Reference="MILLER-20020311200000000PST"]'
    ) = 1;
```

次の結果が戻されます。

```
VALUE(T).TRANSFORM(XMLTYPE(GETDOCUMENT('PURCHASEORDER.XSL')))
--------------------------------------------------------------------------------
<html>
  <head/>
  <body bgcolor="#003333" text="#FFFFCC" link="#FFCC00" vlink="#66CC99" alink="#
669999">
    <FONT FACE="Arial, Helvetica, sans-serif">
      <center>
...
    </FONT>
  </body>
</html>
```

XSLT を使用して変換された文書は、XMLType のインスタンス内に存在していたと考えられるため、そのソースはデータベース表であったと判断できます。

その他の XMLType メソッド

次に、その他の XMLType メソッドについて説明します。

- createXML()。XMLType インスタンスを作成するためのスタティック・メソッド。異なるシグネチャを使用すると、XML 文書を含む多くの異なるソースから XMLType を作成できます。Oracle9*i* データベース リリース 2（9.2）では、大部分が XMLType コンストラクタに置き換えられています。
- isFragment()。XMLType に文書のフラグメントが含まれている場合、TRUE（1）を戻します。文書のフラグメントは、ルート・ノードを持たない XML 文書です。通常、extract() 関数およびメソッドを使用して生成されます。
- getClobVal()。XMLType のコンテンツに基づいて、XML 文書を含む CLOB を戻します。
- getRootElement()。XMLType に含まれる XML 文書のルート要素の名前を戻します。
- getNameSpace()。XMLType に含まれる XML 文書のルート要素の名前を戻します。

W3C の XML Schema 勧告の概要

XML Schema は、一連の XML 文書の予測されるコンテンツを定義する標準化された方法を提供します。XML Schema は、メタデータを定義する XML 文書です。このメタデータは、ドキュメント・クラスのメンバーのコンテンツを指定します。ドキュメント・クラスのメンバーは、インスタンス・ドキュメントともいいます。

XML Schema 定義は、単に XML Schema（http://www.w3.org/2001/XMLSchema）によって定義されるクラスに準拠した XML 文書であるため、XML Schema は、メモ帳、vi などの単純なテキスト・エディタ、Oracle9*i* JDeveloper ツール製品に含まれている XML エディタなどのスキーマを認識するエディタ、または Altova 社の XMLSpy などの明示的な XML Schema 作成ツールを使用して作成できます。XMLSpy などのツールを使用するメリットは、これらのツールを使用すると、開発者が直観的に使用できるグラフィカル・エディタを使用して XML Schema を開発できることです。このグラフィカル・エディタでは、XML Schema 定義の詳細な部分はほとんど表示されません。

例 3-17 XML Schema の例 : PurchaseOrder.xsd

次の PurchaseOrder.xsd の例は、W3C の標準 XML Schema のサンプルです。XML 形式で書かれており、この XML Schema 自体が XML 文書です。

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
        <xs:complexType name="ActionsType" >
                <xs:sequence>
                        <xs:element name="Action" maxOccurs="4" >
                                <xs:complexType >
                                        <xs:sequence>
                                                <xs:element ref="User"/>
```

```
                                                <xs:element ref="Date"/>
                                        </xs:sequence>
                                </xs:complexType>
                        </xs:element>
                </xs:sequence>
        </xs:complexType>
        <xs:complexType name="RejectType" >
                <xs:all>
                        <xs:element ref="User" minOccurs="0"/>
                        <xs:element ref="Date" minOccurs="0"/>
                        <xs:element ref="Comments" minOccurs="0"/>
                </xs:all>
        </xs:complexType>
        <xs:complexType name="ShippingInstructionsType" >
                <xs:sequence>
                        <xs:element ref="name"/>
                        <xs:element ref="address"/>
                        <xs:element ref="telephone"/>
                </xs:sequence>
...
....
                <xs:complexType>
                        <xs:attribute name="Id" >
                                <xs:simpleType>
                                        <xs:restriction base="xs:string">
                                            <xs:minLength value="12"/>
                                            <xs:maxLength value="14"/>
                                        </xs:restriction>
                                </xs:simpleType>
                        </xs:attribute>
                        <xs:attribute name="Quantity" type="money"/>
                        <xs:attribute name="UnitPrice" type="quantity"/>
                </xs:complexType>
        </xs:element>
</xs:schema>
```

参照： 完全な PurchaseOrder.xsd については、付録 B「XML Schema の手引き」を参照してください。

Oracle XML DB での XML Schema の使用

Oracle XML DB は、W3C の XML Schema の使用を次の 2 つの方法でサポートします。

- インスタンス・ドキュメントの自動検証
- 記憶域モデルの定義

W3C の XML Schema を Oracle XML DB で使用するには、XML Schema 文書をデータベースに登録する必要があります。XML Schema を登録すると、その XML Schema にバインドする XMLType 表および列を作成できます。

XML Schema を登録するには、2 つの項目を指定する必要があります。1 つは XML Schema 文書で、もう 1 つは、XML 文書がその XML Schema に準拠していることを示すために使用される URL です。この URL は、W3C の XML Schema 勧告で定義されているとおり、noNamespaceSchemaLocation 属性または schemaLocation 属性を使用して、インスタンス・ドキュメントのルート要素で指定されます。

XML Schema は、PL/SQL パッケージ DBMS_XMLSCHEMA によって提供されるメソッドを使用して登録されます。スキーマは、グローバル・スキーマまたはローカル・スキーマとして登録できます。グローバル・スキーマとローカル・スキーマの違いについては、第 5 章「XMLType の構造化されたマッピング」を参照してください。

Oracle XML DB では、スキーマ登録処理の一部としてデフォルトのデータベース・オブジェクトおよび Java クラスを自動的に生成するための多くのオプションが提供されます。これらのオプションの一部を、この項の後半で示します。

例 3-18 registerSchema() を使用した、PurchaseOrder.xsd のローカル XML Schema としての登録

次の例では、registerSchema() メソッドを使用して、前述の XML Schema PurchaseOrder.xsd をローカル XML Schema として登録します。

```
begin
    dbms_xmlschema.registerSchema(
                    'http://www.oracle.com/xsd/purchaseOrder.xsd',
                    getDocument('PurchaseOrder.xsd'),
                    TRUE, TRUE, FALSE, FALSE
    );
end;
/

--This returns:
-- PL/SQL procedure successfully completed.
```

registerSchema() プロシージャを使用すると、Oracle XML DB によって次の操作が実行されます。

- XML Schema の解析および検証
- XML Schema を記述する Oracle データ・ディクショナリ内の一連のエントリの作成
- XML Schema で定義された complexType に基づく一連の SQL オブジェクト定義の作成

XML Schema を Oracle XML DB に登録すると、XMLType 列を含む表を定義する場合、または XMLType 表を作成する場合にその XML Schema を参照できます。

例 3-19 XML Schema に準拠した XMLType 表の作成

次の例では、http://www.oracle.com/xsd/purchaseOrder.xsd で登録された XML Schema の PurchaseOrder 要素の定義に準拠した XML 文書のみを含むことができる XMLType 表を作成します。

```
CREATE TABLE XML_PURCHASEORDER of XMLType
   XMLSCHEMA "http://www.oracle.com/xsd/purchaseOrder.xsd"
   ELEMENT "PurchaseOrder";
```

次の結果が戻されます。

```
Table created.

DESCRIBE XML_PURCHASEORDER
```

次の結果が戻されます。

```
Name                                      Null?    Type
----------------------------------------- -------- ----------------------------
TABLE of SYS.XMLTYPE(XMLSchema "http://www.oracle.com/xsd/purchaseOrder.xsd" Element
"PurchaseOrder") STORAGE Object-relational TYPE "PurchaseOrder538_T"
```

XMLSchema-instance 名前空間

Oracle XML DB は、XML Schema に基づく表または列に挿入された XML 文書が、XML Schema によって定義されたドキュメント・クラスの有効なメンバーであることを認識する必要があります。XML 文書は、XML Schema またはそれが関連付けられた XML Schema を正しく識別する必要があります。

これは、文書のルート要素の開始タグに適切な属性を追加することによって、文書で使用される名前空間ごとに XML Schema が識別される必要があることを意味します。これらの属性は、W3C の XML Schema 勧告によって定義され、W3C の XMLSchema-instance 名前空間の一部です。したがって、これらの属性を定義するには、まず文書で XMLSchema-instance 名前空間を宣言する必要があります。この名前空間は、xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance と宣言されます。

XMLSchema-instance 名前空間が宣言され、名前空間接頭辞が指定されると、XML Schema を識別する属性をインスタンス・ドキュメントのルート要素に追加できます。特定の文書を 1 つ以上の XML Schema に関連付けることができます。前述の例では、XMLSchema-instance 名前空間の名前空間接頭辞が xsi と定義されています。

noNameSpaceSchemaLocation 属性 非修飾要素に関連付けられた XML Schema は、noNamespaceSchemaLocation 属性を使用して定義されます。XML Schema PurchaseOrder.xsd の場合、正しい定義は次のとおりです。

```
<PurchaseOrder
xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
  xsi:noNamespaceSchemaLocation="http://www.oracle.com/xsd/purchaseOrder.xsd">
```

複数の名前空間の使用 : schemaLocation 属性 XML 文書で複数の名前空間を使用する場合、各名前空間は schemaLocation 属性によって識別される必要があります。たとえば、Purchaseorder 文書で名前空間 PurchaseOrder が使用され、PurchaseOrder 名前空間に po という接頭辞が与えられていると想定します。この場合、PurchaseOrder 文書のルート要素の定義は次のとおりです。

```
<po:PurchaseOrder
xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
xmlns:po="PurchaseOrder"
  xsi:schemaLocation="PurchaseOrder http://www.oracle.com/xsd/purchaseOrder.xsd">
```

XML Schema を使用した XML 文書の検証

デフォルトでは、Oracle XML DB はインスタンス・ドキュメントの格納時に最小限の検証を実行します。この最小限の検証によって、XML 文書の構造が XML Schema で指定された構造に準拠していることが確認されます。

例 3-20 XML Schema PurchaseOrder に準拠した XMLType 表への請求書 XML 文書の挿入の試行

次の例は、XML Schema PurchaseOrder に準拠した文書の格納時に定義された XMLType 表に、請求書を含む XML 文書を挿入しようとした場合に発生する状況です。

```
INSERT INTO XML_PURCHASEORDER
   values (xmltype(getDocument('Invoice.xml')))
  values (xmltype(getDocument('Invoice.xml')))
        *
```

次の結果が戻されます。

```
ERROR at line 2:
ORA-19007: スキーマと要素が一致しません。
```

インスタンスの完全な検証が自動的に実行されない理由は、データベースへの XML 文書の挿入が試行される前にスキーマベースの検証が実行済である可能性が高いと考えられる場合が多いためです。

スキーマベースの検証が実行済でない場合、次のいずれかを使用して、インスタンスの完全な検証を有効にできます。

- 表レベルの CHECK 制約
- PL/SQL の BEFORE INSERT トリガー

例 3-21 XMLType 表での CHECK 制約の使用

次の例は、XMLType 表に対して CHECK 制約を使用する方法と、その表に無効な文書を挿入しようとした場合の結果です。

```
ALTER TABLE XML_PURCHASEORDER
   add constraint VALID_PURCHASEORDER
   check (XMLIsValid(sys_nc_rowinfo$)=1);

-- This returns:
-- Table altered

INSERT INTO XML_PURCHASEORDER
   values (xmltype(getDocument('InvalidPurchaseOrder.xml')));
INSERT INTO XML_PURCHASEORDER;
*
-- This returns:
-- ERROR at line 1:
-- ORA-02290: チェック制約（DOC92.VALID_PURCHASEORDER）に違反しました
```

例 3-22 BEFORE INSERT トリガーを使用した、XMLType 表に挿入されるデータの検証

次の例では、BEFORE INSERT トリガーを使用して、XMLType 表に挿入されるデータが、指定されたスキーマに準拠していることを検証します（エラー・メッセージの string には文字列が入ります）。

```
CREATE TRIGGER VALIDATE_PURCHASEORDER
    before insert on XML_PURCHASEORDER
    for each row
    declare
      XMLDATA xmltype;
    begin
       XMLDATA := :new.sys_nc_rowinfo$;
       xmltype.schemavalidate(XMLDATA);
    end;
/

-- This returns:
-- Trigger created.

insert into XML_PURCHASEORDER
   values (xmltype(getDocument('InvalidPurchaseOrder.xml')));
```

```
-- values (xmltype(getDocument('InvalidPurchaseOrder.xml')))
--        *
-- ERROR at line 2:
-- ORA-31154: 無効な XML 文書
-- ORA-19202: XML 処理中 string にエラーが発生しました
-- LSX-00213: 0 状態変化 ( パーティクル "User") のみです。最小値は 1 です
-- ORA-06512: "SYS.XMLTYPE" 行 0
-- ORA-06512: "DOC92.VALIDATE_PURCHASEORDER" 行 5
-- ORA-04088: トリガー 'DOC92.VALIDATE_PURCHASEORDER' の実行中にエラーが発生しました
```

前述のとおり、どちらの方法でも、妥当な XML 文書のみが XMLType 表に格納されることが保証されます。

- **表 CHECK 制約 :** 表制約を使用する方法のメリットは、コーディングがより単純であることです。表制約のデメリットは、この方法が isSchemaValid() メソッドに基づいているため、インスタンス・ドキュメントが妥当であるかどうかのみを示すことです。この方法では、インスタンス・ドキュメントが妥当でない場合にその理由についての情報が提供されません。
- **BEFORE INSERT トリガー :** BEFORE INSERT トリガーの場合、必要なコーディングが表制約よりわずかに多くなります。このトリガーのメリットは、schemaValidate() メソッドに基づいていることです。この方法では、インスタンス・ドキュメントが妥当でない場合にそのインスタンス・ドキュメントの問題点についての情報が提供されます。また、トリガーが適切な処置を行うことができるというメリットもあります。

XML の格納 : 構造化記憶域または非構造化記憶域

Oracle XML DB アプリケーションを設計する場合、まず、構造化記憶域または非構造化記憶域のどちらに XMLType 列および表を格納するかを決定する必要があります。

表 3-1 に、構造化記憶域と非構造化記憶域における XML の格納の比較を示します。

表 3-1 XML の構造化記憶域と非構造化記憶域の比較

機能	XML の非構造化記憶域	XML の構造化記憶域
格納方法	XMLType 列および表のコンテンツが、CLOB データ型を使用して格納されます。	XMLType 列および表のコンテンツが、SQL オブジェクトのコレクションとして格納されます。デフォルトでは、XML Schema に基づく XMLType 列および表の基礎となる記憶域モデルは構造化記憶域です。
XML Schema に基づかない表を格納できるかどうか	XML Schema に関連付けられていない XMLType 表および列に対してのみ使用できます。	XMLType 列または表が XML Schema に基づく場合にのみ使用できます。これは、インスタンス・ドキュメントが基礎となる XML Schema に準拠している必要があることを意味します。
パフォーマンス : 格納速度および取出し速度	格納および取出し操作中に解析および分解に関連するオーバーヘッドが回避されるため、より高速な収集および取出しが可能です。	文書の収集中に断片化が発生し、取出し前に再構成されるため、収集および取出し操作中にわずかなオーバーヘッドが発生します。
パフォーマンス : 操作速度	構造化記憶域の場合より遅くなります。	XML Schema を登録すると、Oracle XML DB では、XML Schema で定義された complexType に対応する一連の SQL オブジェクトが生成されます。Oracle XML DB 関数に送信された XPath 式は、基礎となるオブジェクトに対して直接操作を実行する SQL 文に変換されます。 このオブジェクト・リレーショナル SQL 文への XMLType 操作の再作成によって、非構造化記憶域を使用して格納された XML 文書に対して同じ操作を実行した場合と比較して、パフォーマンスが大幅に向上します。
柔軟性 : 様々なコンテンツを簡単に処理できるかどうか	処理される文書の柔軟性が高いため、XML 文書に様々なコンテンツが含まれている場合に適しています。	Oracle9*i* データベースのオブジェクト・リレーショナル機能を使用します。

表 3-1 XML の構造化記憶域と非構造化記憶域の比較 （続き）

機能	XML の非構造化記憶域	XML の構造化記憶域
メモリー使用量 : XML 文書の解析が必要かどうか	Oracle XML DB は、XML 文書に対して検証、XSLT 変換または XPath 操作を実行する前に、その文書全体を解析し、それをメモリー内 DOM 構造にロードする必要があります。	Oracle XML DB は、次の機能を使用することによって、メモリー使用量を最小化し、XMLType 表および列に対する DOM ベースの操作のパフォーマンスを最適化できます。 ■ **実体化の遅延（LM）**: Oracle XML DB が XML 文書に基づいて DOM 構造を構成するときに発生します。文書へのアクセス時に DOM 全体を構成するかわりに、LM を使用すると、Oracle XML DB は直後の操作を実行するために必要なノードのみをインスタンス化します。文書の他の部分が必要なときは、適切なノード・ツリーが動的に DOM にロードされます。 ■ **最低使用頻度（LRU）**: DOM 内の最近アクセスされていないノードを廃棄するための方針です。
更新処理	格納時に、文書に対して更新操作を行うと、CLOB 全体が再書込みされます。 updateXML() を使用して文書の一部を更新した場合、その文書全体を CLOB からフェッチおよび更新し、CLOB に再書込みする必要があります。	XML 文書全体を再書込みせずに、文書内の個々の要素、属性またはノードを更新できます。 updateXML() 操作を、XPath 式によって参照される列またはオブジェクトを操作する SQL の UPDATE 文に再作成できます。
索引付け	XPath 式の評価に基づく B* ツリー索引、または Oracle Text の逆向きのリスト索引を使用できます。 非構造化記憶域では、コレクション内で使用されている要素または属性の値に基づく B* ツリー索引を作成できません。	B* ツリー索引および Oracle Text の逆向きのリスト索引を使用できます。 コレクションの管理方法をチューニングすると、文書内の任意の要素や属性（コレクション内で使用されている要素や属性など）に対する索引を作成できます。

表 3-1 XML の構造化記憶域と非構造化記憶域の比較 （続き）

機能	XML の非構造化記憶域	XML の構造化記憶域
必要な領域	大きい領域が必要な場合があります。	XML Schema に基づくため、Oracle XML DB は、XML 文書のコンテンツの格納時に XML タグ名を格納する必要がありません。これによって、必要な記憶域を大幅に削減できます。
データ整合性	--	XML 文書のコンテンツをデータベース内の別の場所に保存されている情報に対して検証できる一連のデータベース整合性制約を使用できます。
チューニング : オブジェクトのファイングレイン・コントロール	なし	XML Schema から生成された SQL オブジェクトの集合に対するファイングレイン・コントロールおよびこれらのオブジェクトのデータベースへの格納方法について、XML Schema に注釈を付けることができます。 コレクションの管理方法、表領域使用の定義、SQL オブジェクトの格納および管理に使用される 1 つ以上の表のパーティション化を制御できます。これによって、アプリケーションのニーズを満たすために Oracle XML DB のパフォーマンスを微調整できるようになります。 その他の注釈によって、単純な要素および属性を SQL 列にマップする方法が制御されます。

データ操作言語（DML）の非依存性

Oracle XML DB では、Oracle XML DB 関数に基づくすべてのデータ操作言語（DML）操作が一貫した結果を戻すことが保証されます。Oracle XML DB は、`XMLType` データ型を使用して記憶域モデルを抽象化し、XPath を使用して XML 文書に対する操作を実行する一連の演算子によって、構造化記憶域と非構造化記憶域の切替えを可能にしています。また、アプリケーションに影響することなく異なる形式の構造化記憶域を試してみることができます。

構造化記憶域および非構造化記憶域における DOM 再現性

DOM 再現性を保持するには、格納された XML 文書の表現から生成された DOM が元の XML 文書から生成された DOM と同じであることがシステムによって保証される必要があります。DOM 整合性を保持することによって、XML 文書に含まれるすべての情報が文書の格納後に失われないことが保証されます。

DOM 整合性を保持するうえでの問題は、XML 文書に、要素値および属性値に含まれているデータの他に、多くの情報が含まれている場合があることです。この情報には、コメントおよび処理命令を使用して明示的に指定されるものもあります。または、次のように暗黙的に指定されるものがあります。

- コレクション内の要素の順序付け
- 親内の子要素の順序付け
- コメントおよび処理命令の相対的な位置

従来のリレーショナル・モデルを使用して XML 文書のコンテンツを管理する場合に、アプリケーション開発者が直面する一般的な問題の 1 つは、この情報の保存方法です。表 3-2 に、構造化記憶域と非構造化記憶域における DOM 再現性の比較を示します。

表 3-2 DOM 再現性 : 非構造化記憶域および構造化記憶域

非構造化記憶域での DOM 再現性	構造化記憶域での DOM 再現性
リレーショナル・システムは、暗黙的な順序付け、およびコメントや処理命令などのバンド外データを簡単に保存できるようにする柔軟性を提供しません。一般的なリレーショナル・データベースの場合、DOM 再現性を保持するには、非構造化記憶域方法を使用してソース・ドキュメントを格納する必要があります。	Oracle XML DB は、構造化記憶域を使用した場合でも DOM 再現性を保持できます。XML 文書を断片化し、構造化記憶域方法を使用して格納した場合、コメント、処理命令、およびソース・ドキュメント内のすべての暗黙的な順序付け情報は、その文書が断片化されたときに作成された SQL オブジェクトの一部として保存されます。この情報は、文書の取出し時に、生成された XML 文書に再度取り込まれます。

構造化記憶域 : XMLType の XML Schema に基づく記憶域

論理的に、XML 文書は要素および属性のコレクションで構成されます。要素は、次のいずれかです。

- `complexType`（子要素および属性を含む）
- `simpleType`（スカラー値を含む）

XML Schema は、XML 文書の特定のクラスに存在可能な要素および属性の集合、およびそれらの間の関係を定義します。

XML Schema の登録中、Oracle XML DB によって、その XML Schema で定義された各 `complexType` 用の SQL オブジェクト型が生成されます。SQL オブジェクトの定義には、`complexType` の定義が反映されます。

`complexType` によって定義された各子要素および属性は、SQL オブジェクト型の属性にマップされます。

- `complexType` の子要素自体が `complexType` である場合、対応する SQL 属性のデータ型は該当する SQL 型になります。
- W3C の XML Schema 勧告によって定義されたスカラー・データ型のいずれかに基づいて、子要素が `simpleType` または属性である場合、対応する SQL 属性のデータ型は該当する基本 SQL データ型になります。

XML Schema 名および Oracle XML DB 名前空間の定義

デフォルトでは、XML Schema の登録時に生成された SQL オブジェクトの名前は、システムによって生成されます。ただし、Oracle XML DB では、XML Schema に注釈を付けることによって、ユーザーが SQL オブジェクトの名前を指定できます。XML Schema に注釈を付けるには、まず、次のとおり定義される Oracle XML DB 名前空間を XMLSchema タグに含める必要があります。

```
http://xmlns.oracle.com/xdb
```

したがって、Oracle XML DB の注釈を使用する XML Schema では、XMLSchema タグに次の属性が含まれている必要があります。

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns:xdb="http://xmlns.oracle.com/xdb" >
...
</xs:schema>
```

Oracle XML DB 名前空間を定義すると、Oracle XML DB によって定義された注釈を使用できます。

例 3-23 complexType から生成された SQL オブジェクトの名前の定義

次の例では、xdb:SQLType を使用して、complexType である PurchaseOrder から生成された SQL オブジェクトの名前を XML_PURCHASEORDER_TYPE と定義します。

```
<xs:element name="PurchaseOrder">
    <xs:complexType type="PurchaseOrderType"
                    xdb:SQLType="XML_PURCHASEORDER_TYPE">
        <xs:sequence>
            <xs:element ref="Reference"/>
            <xs:element name="Actions" type="ActionsType"/>
            <xs:element name="Reject" type="RejectType" minOccurs="0"/>
            <xs:element ref="Requestor"/>
            <xs:element ref="User"/>
            <xs:element ref="CostCenter"/>
            <xs:element name="ShippingInstructions"
                        type="ShippingInstructionsType"/>
            <xs:element ref="SpecialInstructions"/>
            <xs:element name="LineItems" type="LineItemsType"/>
       </xs:sequence>
    </xs:complexType>
</xs:element>
```

次の文を実行します。

```
DESCRIBE XML_PURCHASEORDER_TYPE;
```

次の構造が戻されます。

```
XML_PURCHASEORDER_TYPE is NOT FINAL
Name                           Null?    Type
------------------------------ -------- ----------------------------
SYS_XDBPD$                              XDB.XDB$RAW_LIST_T
Reference                               VARCHAR2(26)
Actions                                 XML_ACTIONS_TYPE
Reject                                  XML_REJECTION_TYPE
Requestor                               VARCHAR2(128)
User                                    VARCHAR2(10)
CostCenter                              VARCHAR2(4)
ShippingInstructions                    XML_SHIPPINGINSTRUCTIONS_TYPE
SpecialInstructions                     VARCHAR2(2048)
LineItems                               XML_LINEITEMS_TYPE
```

注意： 前述の例では、xdb:SQLType 注釈は、complexType である ActionsType、ShippingInstructionsType および LineItemsType に対応する SQL 型に名前を割り当てるためにも使用されています。

xdb:SQLName を使用したデフォルト名のオーバーライド

Oracle XML DB は、事前定義アルゴリズムを使用して、XML Schema で定義された XML 要素、属性および型の名前から有効な SQL 名を生成します。xdb:SQLName 注釈を使用して、デフォルトのアルゴリズムをオーバーライドし、これらの項目に明示的に名前を指定できます。

xdb:SQLType を使用したデフォルトのマッピングのオーバーライド

Oracle XML DB は、XML Schema 勧告によって定義されたスカラー・データ型と SQL によって定義された基本データ型間のデフォルトのマッピングも提供します。SQL データ型のサイズが、XML データ型に定義された制限事項から導出される場合もあります。必要に応じて、xdb:SQLType 注釈を使用して、このデフォルトのマッピングをオーバーライドできます。

例 3-24 xdb:SQLType および xdb:SQLName を使用した、complexType から生成されたオブジェクトの名前およびマッピングの指定

次の例では、SpecialInstructions 要素に対して使用される名前および型をオーバーライドする方法、および生成された SQL オブジェクト型にこれらの変更が及ぼす影響を示します。

注意： SpecialInstructions 要素名のオーバーライドは、要素が定義された場所ではなく、要素が使用される PurchaseOrderType 内で適用されます。

```
<xs:element name="SpecialInstructions" xdb:SQLType="CLOB"  >
    <xs:simpleType>
        <xs:restriction base="xs:string">
            <xs:minLength value="0"/>
            <xs:maxLength value="2048"/>
        </xs:restriction>
    </xs:simpleType>
</xs:element>

<xs:element name="PurchaseOrder">
    <xs:complexType type="PurchaseOrderType"
                    xdb:SQLType="XML_PURCHASEORDER_TYPE">
        <xs:sequence>
            <xs:element ref="Reference"/>
            <xs:element name="Actions" type="ActionsType"/>
            <xs:element name="Reject" type="RejectType" minOccurs="0"/>
            <xs:element ref="Requestor"/>
            <xs:element ref="User"/>
            <xs:element ref="CostCenter"/>
            <xs:element name="ShippingInstructions"
                        type="ShippingInstructionsType"/>
            <xs:element ref="SpecialInstructions"
                        xdb:SQLName="SPECINST"/>
            <xs:element name="LineItems" type="LineItemsType"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

次の文を実行します。

```
DESCRIBE XML_PURCHASEORDER_TYPE;
```

次の構造が戻されます。

```
XML_PURCHASEORDER_TYPE is NOT FINAL
Name                           Null?    Type
------------------------------ -------- ----------------------------
SYS_XDBPD$                              XDB.XDB$RAW_LIST_T
Reference                               VARCHAR2(26)
Actions                                 XML_ACTIONS_TYPE
Reject                                  XML_REJECTION_TYPE
Requestor                               VARCHAR2(128)
User                                    VARCHAR2(10)
CostCenter                              VARCHAR2(4)
ShippingInstructions                    XML_SHIPPINGINSTRUCTIONS_TYPE
SPECINST                                CLOB
LineItems                               XML_LINEITEMS_TYPE
```

構造化記憶域 : complexType コレクションの格納

構造化記憶域を選択する場合に考慮する必要があることに、コレクションの管理方法があります。様々な方法を使用でき、それぞれにメリットがあります。通常、コレクションは次の 5 つの方法で処理できます。

- **CLOB:** complexType が xdb:SQLType="CLOB" で定義されている場合、その型およびすべての子要素は、非構造化記憶域方法で格納されます。
- インライン **VARRAY:** 複数回出現する complexType に対して他の情報が与えられない場合、コレクションのメンバーは、親要素用の SQL オブジェクトの一部として、シリアル化オブジェクトの集合として表内に格納されます。コレクションの一部である要素または属性に B* ツリー索引は作成できません。
- **ネストしたオブジェクト表 :** コレクションのメンバーは、ネストしたオブジェクト表に格納されます。SQL オブジェクトは、前述のオプションの場合と同様に VARRAY 型の属性を含んでいますが、表として格納されます。親である行には、一意の setid（セット識別子）値が含まれています。この値は、対応するネストした表の行に関連付けるために使用されます。
- **個別の XMLType 表 :** コレクションのメンバーは、個別の XMLType 表として格納されます。コレクションの各メンバーは、表内の行として格納されます。親 SQL オブジェクトには、この親に属する子表内の行を指す REF の配列が含まれます。すべてのデータは XMLType です。
  - XML Schema に基づく複数の XMLType 列の作成
  - 子から対応する親へのリンク
- **リンク表を含む個別の XMLType 表 :** コレクションのメンバーは、個別の XMLType 表として格納されます。子表内のどのメンバーが親表のどのメンバーにリンクされているかを相互参照するリンク表が作成されます。すべてのデータは、XMLType として参照で

きます。子から親への逆リンクも可能ですが、XML Schema に基づく複数の XMLType 列を作成するうえで問題があります。

> **参照：** 第 5 章「XMLType の構造化されたマッピング」を参照してください。

構造化記憶域 : データ整合性および制約チェック

構造化記憶域では、スキーマの検証に加えて、XMLType 列および表に従来のリレーショナル制約を導入できます。データベース整合性チェックを使用すると、XML Schema に基づく検証で実現可能な範囲を超えるインスタンスの検証を実行できます。

W3C の XML Schema 勧告では、インスタンス・ドキュメントで値の相互参照に基づく検証のみが可能です。データベース整合性チェックを使用すると、ドキュメントのコレクション全体で要素または属性の一意性を規定したり、データベース内の他の場所に格納された情報に対して要素または属性の値を検証するなど、他の種類の検証を施行できます。

注意： Oracle9*i* データベース リリース 2（9.2）では、制約はオブジェクト・リレーショナル構文を使用して指定する必要があります。

例 3-25 PurchaseOrder 表への一意の参照制約の追加

次の例では、PurchaseOrder 表に一意の参照制約を導入します。

```
XMLDATA.SQLAttributeName
alter table XML_PURCHASEORDER
add constraint REFERENCE_IS_UNQIUE
-- unique(extractValue('/PurchaseOrder/Reference'))
unique (xmldata."Reference");
```

次に示すとおり、要素 /PurchaseOrder/Reference の重複値を含む XML 文書を表に挿入しようとすると、データベースはその挿入が一意制約に違反することを検出し、該当するエラーを戻します。

```
insert into xml_purchaseorder values (
   xmltype(getDocument('ADAMS-20011127121040988PST.xml'))
   );
```

次の結果が戻されます。

```
1 row created.

insert into xml_purchaseorder values (
  xmltype(getDocument('ADAMS-20011127121040988PST.xml'))
  );

insert into xml_purchaseorder values (
*
```

次の結果が戻されます。

```
ERROR at line 1:
ORA-00001: 一意制約（DOC92.REFERENCE_IS_UNQIUE）に反しています
```

例 3-26 Oracle9i データベースが参照制約 User_Is_Valid を規定する方法

次の例では、データベースが参照制約 USER_IS_VALID を規定します。

```
alter table XML_PURCHASEORDER
add constraint USER_IS_VALID
-- foreign key extractValue('/PurchaseOrder/User') references
SCOTT.EMP(ENAME)
foreign key (xmldata."User") references SCOTT.EMP(ENAME);
```

この制約は、SQLAttribute xmldata.user" に変換される要素 /PurchaseOrder/User の値が SCOTT.EMP 内の ENAME の値のいずれかと一致する必要があることを示します。

```
insert into xml_purchaseorder values (
   xmltype(getDocument('HACKER-20011127121040988PST.xml'))
   );

insert into xml_purchaseorder values (
*
```

次の結果が戻されます。

```
ERROR at line 1:
ORA-02291: 整合性制約 (SCOTT.USER_IS_VALID) に違反しました - 親キーがありません
```

Oracle XML DB Repository

XML 文書は階層構造です。XML 文書に含まれる情報は、要素、子要素および属性の階層によって表されます。また、XML 文書はそれ自体以外のものを階層として参照します。XML 文書は、別の XML 文書または他の種類のドキュメントを参照する場合、URL を使用します。URL は、相対 URL または絶対 URL のいずれかです。どちらの場合も、URL はターゲット・ドキュメントへのパスを定義します。このパスは、フォルダ階層で表されます。

Oracle XML DB Repository を使用すると、データベースに格納されたすべての XML コンテンツをファイル / フォルダ隠喩を使用して表示できます。Oracle XML DB Repository は、ファイルやフォルダの作成などの基本操作の他に、バージョン管理やアクセス制御などのより高度な機能もサポートします。

Oracle XML DB Repository は、SQL を介して完全にアクセス、問合せおよび更新できます。また、Oracle XML DB Repository には、HTTP、WebDAV、FTP などの業界標準のプロトコルを介して直接アクセスすることもできます。

参照： 第 13 章「Oracle XML DB Foldering」を参照してください。

IETF WebDAV 標準の概要

WebDAV は、コンテンツの分散オーサリングおよびバージョニングのための Internet Engineering Task Force（IETF）標準です。この標準は、HTTP プロトコルの拡張によって実装され、Web サーバーが分散環境でファイル・サーバーとして機能できるようにします。

WebDAV に基づく Oracle XML DB Repository

Oracle XML DB Repository は、WebDAV 標準によって定義されたモデルに基づいています。Oracle XML DB Repository は、WebDAV リソース・モデルを使用して、リポジトリに格納された各ドキュメント用に保持される基本メタデータを定義します。WebDAV プロトコルは、XML を使用して、クライアントとサーバー間でメタデータを転送します。

そのため、標準ツールを使用して、Oracle XML DB Repository に格納されたドキュメントの作成や編集、およびドキュメントへのアクセスを簡単に行うことができます。たとえば、次のツールを使用できます。

- Microsoft Web フォルダ
- WebDAV が使用可能なその他の製品（Microsoft Office、Macromedia、Adobe 社のオーサリング・ツールなど）

WebDAV では、ファイルやフォルダを定義するために、「リソース」という用語が使用されます。WebDAV は、リソースに対して実行できる一連の基本操作を定義します。これらの操作を実行するには、WebDAV サーバーが各リソースの一連の基本メタデータを保持する必要があります。Oracle XML DB は、このメタデータを次の形式の一連の XML 文書として公開します。

例 3-27 Oracle XML DB による XML 文書としての WebDAV リソース・メタデータの公開

```
<Resource xmlns="http://xmlns.oracle.com/xdb/XDBResource.xsd"
          Hidden="false" Invalid="false" Container="false"
          CustomRslv="false">
  <CreationDate> 2002-02-14T16:01:01.066324000</CreationDate>
  <ModificationDate> 2002-02-14T16:01:01.066324000</ModificationDate>
  <DisplayName>testFile.xml</DisplayName>
  <Language>us english</Language>
  <CharacterSet>utf-8</CharacterSet>
  <ContentType>text/xml</ContentType>
  <RefCount>1</RefCount>
  <ACL>
    <acl description="/sys/acls/all_all_acl.xml"
         xmlns="http://xmlns.oracle.com/xdb/acl.xsd"
         xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
        xsi:schemaLocation="http://xmlns.oracle.com/xdb/acl.xsd
                            http://xmlns.oracle.com/xdb/acl.xsd">
      <ace>
        <grant>true</grant>
        <privilege>
```

```
          <all/>
        </privilege>
        <principal>PUBLIC</principal>
      </ace>
    </acl>
  </ACL>
  <Owner>DOC92</Owner>
  <Creator>DOC92</Creator>
  <LastModifier>DOC92</LastModifier>
  <SchemaElement>
     http://xmlns.oracle.com/xdb/XDBSchema.xsd#binary
   </SchemaElement>
  <Contents>
    <binary>02C7003802C77B7000081000838B1C240000000002C71E7C</binary>
  </Contents>
</Resource>
```

Oracle XML DB Repository への問合せベースのアクセス

Oracle XML DB は、リポジトリを 2 つのビューとして SQL 開発者に公開します。

- RESOURCE_VIEW
- PATH_VIEW

また、Oracle XML DB は、リポジトリ操作を実行するための一連の SQL 関数および PL/SQL パッケージも提供します。

> **参照：** 第 15 章「RESOURCE_VIEW および PATH_VIEW」を参照してください。

RESOURCE_VIEW の使用

RESOURCE_VIEW は、Oracle XML DB Repository を問い合せるための主要な方法です。RESOURCE_VIEW には、リポジトリに格納されたドキュメントごとに 1 つのエントリがあります。RES 列にはドキュメントのリソース・エントリが含まれ、ANY_PATH エントリはルートからリソースまでの有効なフォルダ・パスを指定します。

RESOURCE_VIEW の定義は次のとおりです。

```
SQL> describe RESOURCE_VIEW

 Name                           Null?    Type
------------------------------- -------- ----------------------------
 RES                                     SYS.XMLTYPE
 ANY_PATH                                VARCHAR2(4000)
```

PATH_VIEW の使用

PATH_VIEW には、リポジトリ内の各パスのエントリが含まれます。リソースは複数のフォルダにリンクできるため、PATH_VIEW には、リポジトリ内のすべての使用可能なパスおよびそれらのパスが指すリソースが表示されます。PATH_VIEW の定義は次のとおりです。

```
SQL> describe PATH_VIEW

 Name                           Null?    Type
------------------------------- -------- ----------------------------
 PATH                                    VARCHAR2(1024)
 RES                                     SYS.XMLTYPE
 LINK                                    SYS.XMLTYPE
```

新しいフォルダおよびドキュメントの作成

DBMS_XDB パッケージによって提供されるメソッドを使用して、新しいフォルダおよびドキュメントを作成できます。たとえば、createFolder() プロシージャを使用して新しいフォルダを作成し、createResource() を使用してそのフォルダにファイルをアップロードできます。次の例は、この方法を示します。

例 3-28 リポジトリのリソースおよびフォルダの作成

```
SQL> declare
  2    result boolean;
  3  begin
  4    result := dbms_xdb.createFolder('/public/testFolder');
  5  end;
  6  /

PL/SQL procedure successfully completed.

SQL> declare
  2    result boolean;
  3  begin
4    result := dbms_xdb.createResource(
5                        '/public/testFolder/testFile.xml',
6                        getDocument('testFile.xml')
7    );
  8  end;
  9  /

PL/SQL procedure successfully completed.
```

リソース・ドキュメントの問合せ

RESOURCE_VIEW は、他のビューと同様に問い合せることができます。Oracle XML DB は、新しい演算子 UNDER_PATH を提供します。この演算子は、問合せを RESOURCE_VIEW 内の特定のフォルダ・ツリーに制限する方法を提供します。

extractValue() および existsNode() は、RESOURCE_VIEW および PATH_VIEW リソース・ドキュメントの問合せ時にリソース・ドキュメントに対して使用できます。

リソースの更新

リソースは、updateXML() を使用して更新できます。

例 3-29 リポジトリのリソースの更新

次の問合せでは、前述の例で作成されたドキュメントの所有者および名前を更新します。

```
update RESOURCE_VIEW
  set RES=updateXML(RES,
                    '/Resource/DisplayName/text()','RenamedFile',
                    '/Resource/Owner/text()','SCOTT'
 )
where any_path = '/public/testFolder/testFile.xml';

-- 1 row updated.

select r.res.getClobVal()
   from RESOURCE_VIEW r
   where ANY_PATH = '/public/testFolder/testFile.xml'
/

-- Results in:
-- R.RES.GETCLOBVAL()
-- --------------------------------------------------------------------------
-- <Resource xmlns="http://xmlns.oracle.com/xdb/XDBResource.xsd"
--           Hidden="false" Invalid="false" Container="false"
--           CustomRslv="false">
--   <CreationDate> 2002-02-14T16:01:01.066324000</CreationDate>
--   <ModificationDate> 2002-02-14T21:36:39.579663000</ModificationDate>
--   <DisplayName>RenamedFile</DisplayName>
--   <Language>us english</Language>
--   <CharacterSet>utf-8</CharacterSet>
--   <ContentType>text/xml</ContentType>
-- <RefCount>1</RefCount>
-- <ACL>
-- ...
-- </ACL>
```

```
--    <Owner>SCOTT</Owner>
--    <Creator>DOC92</Creator>
--   <LastModifier>DOC92</LastModifier>
-- </Resource>
```

リソースの削除

リソースは、deleteResource() を使用して削除できます。リソースがフォルダである場合、フォルダを削除する前に、そのフォルダを空にする必要があります。

例 3-30 リポジトリのリソースの削除

次の例では、deleteResource() プロシージャを使用します。

```
call dbms_xdb.deleteResource('/public/testFolder')
   /
call dbms_xdb.deleteResource('/public/testFolder')
     *
ERROR at line 1:
ORA-31007: 空ではないコンテナ /public//testFolder を削除しようとしました
ORA-06512: 151 行 XDB.DBMS_XDB
ORA-06512: 1 行

call dbms_xdb.deleteResource('/public/testFolder/testFile.xml')
 /
Call completed.

call dbms_xdb.deleteResource('/public/testFolder')
/
Call completed.
```

リソース用の記憶域オプション

RESOURCE_VIEW および PATH_VIEW は、Oracle XML DB データベース・スキーマに格納された表に基づいています。RESOURCE_VIEW および PATH_VIEW を介して公開されたメタデータは、Oracle XML DB が提供する XML Schema である XDBSchema.xsd 内で定義されている一連の表を使用して格納および管理されます。ファイルのコンテンツは、この XML Schema 内に BLOB 列または CLOB 列として格納されます。

参照： G-18 ページの「xdbconfig.xsd: Oracle XML DB を構成するための XML Schema」を参照してください。

XML Schema に基づく文書に対するデフォルト表の記憶域の定義

XML Schema に基づく XML 文書を格納する場合、この記憶域パラダイムには例外があります。XML Schema を Oracle XML DB に登録するときに、その XML Schema で定義された各ルート要素のデフォルト表の記憶域を定義できます。

xdb:defaultTable 属性を最上位要素の定義に追加すると、ユーザー独自のデフォルト表の記憶域を定義できます。XML Schema の登録時に、Oracle XML DB によって、ユーザー独自の XML Schema で定義されたデフォルト表とリポジトリ間のリンクが確立されます。デフォルト表は、XML Schema 登録の一部として生成されるように選択できます。

リポジトリと連動する XMLType 表であるデフォルト表

デフォルト表は XMLType 表であり、XMLType データ型に基づくオブジェクト表です。ユーザー独自のデフォルト表のルート要素および XML Schema と一致するルート要素および XML Schema を含む XML 文書をリポジトリに挿入すると、XML コンテンツは指定したデフォルト表内の行として格納されます。デフォルト表内の該当する行への参照を含むリソースが作成されます。

XMLType 表の特長の 1 つに、リポジトリと連動することがあります。XML Schema 登録の一部として作成されたデフォルト表は、自動的にリポジトリと連動します。表がリポジトリと連動する場合、デフォルト表に対する DML 操作が実行されると、それに対応する操作が Oracle XML DB Repository に対して実行される場合があります。たとえば、デフォルト表から行が削除されると、その行を参照するリポジトリ内のすべてのエントリが削除されます。

例 3-31 XML Schema の要素定義への xdb:defaultTable 属性の追加

次の例は、xdb:defaultTable 属性を XML Schema 定義の PurchaseOrder 要素に追加し、gentables 引数を TRUE に設定して XML Schema を登録した場合の結果です。

```
<xs:element name="PurchaseOrder" xdb:defaultTable="XML_PURCHASEORDER">
    <xs:complexType type="PurchaseOrderType"
                    xdb:SQLType="XML_PURCHASEORDER_TYPE">
        <xs:sequence>
            <xs:element ref="Reference"/>
            <xs:element name="Actions" type="ActionsType"/>
            <xs:element name="Reject" type="RejectType" minOccurs="0"/>
            <xs:element ref="Requestor"/>
            <xs:element ref="User"/>
            <xs:element ref="CostCenter"/>
            <xs:element name="ShippingInstructions"
                        type="ShippingInstructionsType"/>
            <xs:element ref="SpecialInstructions"
                        xdb:SQLName="SPECINST"/>
            <xs:element name="LineItems" type="LineItemsType"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

```
SQL> begin
  2    dbms_xmlschema.registerSchema(
  3                    'http://www.oracle.com/xsd/purchaseOrder.xsd',
  4                    getDocument('purchaseOrder3.xsd'),
  5                    TRUE, TRUE, FALSE, TRUE
  6         );
  7
  8  end;
  9  /

PL/SQL procedure successfully completed.

SQL> describe XML_PURCHASEORDER

 Name                           Null?    Type
 ------------------------------ -------- ----------------------------
TABLE of SYS.XMLTYPE(XMLSchema
http://www.oracle.com/xsd/purchaseOrder.xsd Element "PurchaseOrder")
STORAGE Object-relational TYPE "XML_PURCHASEORDER_TYPE"
```

例 3-32 Oracle XML DB Repository への XML 文書の挿入によって発生する表への行の挿入

次の例では、XML Schema を登録し、デフォルト表が作成された場合に、XML 文書を Oracle XML DB Repository に挿入すると、指定したデフォルト表に行が挿入されます。

```
select count(*) from XML_PURCHASEORDER;
```

次の結果が戻されます。

```
  COUNT(*)
 ----------
          0

-- create testFolder
declare
 result boolean;
begin
 result := dbms_xdb.createFolder('/public/testFolder');
end;
/

declare
      result boolean;
    begin
      result := dbms_xdb.createResource(
                        '/public/testFolder/purchaseOrder1.xml',
                       getDocument('purchaseOrder1.xml')
```

```
              );
    end;
/

-- PL/SQL procedure successfully completed.

commit;

-- Commit complete.

select count(*) from XML_PURCHASEORDER;
```

次の結果が戻されます。

```
   COUNT(*)
 ----------
          1
```

例 3-33 行の削除によって発生するリポジトリからの対応するエントリの削除

次の例では、リポジトリと連動するデフォルト表から行を削除すると、対応するエントリがその階層から削除されます。

```
select extractValue(res,'Resource/DisplayName') "Filename"
   from RESOURCE_VIEW where under_path(res,'/public/testFolder') = 1;
/
```

次の結果が戻されます。

```
Filename
--------------------------------------------------------------------------------
purchaseOrder1.xml

delete from XML_PURCHASEORDER;
1 row deleted.

SQL> commit;
Commit complete.

select extractValue(res,'Resource/DisplayName') "Filename"
   from RESOURCE_VIEW where under_path(res,'/public/testFolder') = 1
/
```

次の結果が戻されます。

```
no rows selected
```

XML Schema に基づくコンテンツへのアクセス

リソースがデフォルト表に格納された XML コンテンツを記述する場合、そのリソース・エントリ自体には、デフォルト表内の該当する行への参照のみが含まれます。この参照は、リソースとそのコンテンツ間の結合操作を実行するために使用できます。この例を次に示します。

XDBUriType を使用した XML Schema に基づかないコンテンツへのアクセス

XDBUriType を使用すると、論理パスを使用してリポジトリに格納されたファイルのコンテンツにアクセスできます。次の例では、JPEG ファイルに関連付けられたリソースにアクセスします。JPEG ファイルは、リポジトリに挿入済です。この例では、Oracle *inter*Media の ordsys.ordimage クラスを使用して、JPEG ファイルに関連付けられたメタデータを抽出します。

```
create or replace function getImageMetaData (uri varchar2)
return xmltype deterministic
is
  resType xmltype;
  resObject xdb.xdb$resource_t;
  attributes CLOB;
  xmlAttributes xmltype;
begin
   DBMS_LOB.CREATETEMPORARY(attributes, FALSE, DBMS_LOB.CALL);
   -- ordsys.ordimage.getProperties(xdburitype(uri).getBlob(),
   --                               attributes);
   select res into resType from resource_view where any_path = uri;
   resType.toObject(resObject);
   ordsys.ordimage.getProperties(resObject.XMLLOB,attributes);
   xmlAttributes := xmltype(attributes);
   DBMS_LOB.FREETEMPORARY(attributes);
   return xmlAttributes;
end;
/
```

Oracle XML DB Protocol Server

Oracle XML DB には、3 つのプロトコル・サーバーが含まれます。これらのサーバーを介して、標準のファイルベースのアプリケーションから直接リポジトリにアクセスできます。

> **参照：** 第 19 章「FTP、HTTP および WebDAV プロトコルの 使用」を参照してください。

FTP プロトコル・サーバーの使用

FTP プロトコル・サーバーを使用すると、標準 FTP クライアントは、通常の FTP サーバーのコンテンツの場合と同様に、リポジトリに格納されたコンテンツにアクセスできます。FTP プロトコル・サーバーは、次に示す標準 FTP クライアントとともに動作します。

- コマンドライン・クライアント（UNIX および Windows のコマンド・プロンプトに付属のコマンドライン・クライアントなど）
- グラフィカル・クライアント（WS-FTP など）
- FTP プロトコルをサポートする Web ブラウザ

図 3-6、図 3-7、図 3-8 および図 3-9 に、様々な標準 FTP クライアントを使用してリポジトリのルート・レベルにアクセスする方法を示します。

図 3-6 DOS コマンド・プロンプトのコマンドラインからリポジトリのルート・レベルへのアクセス

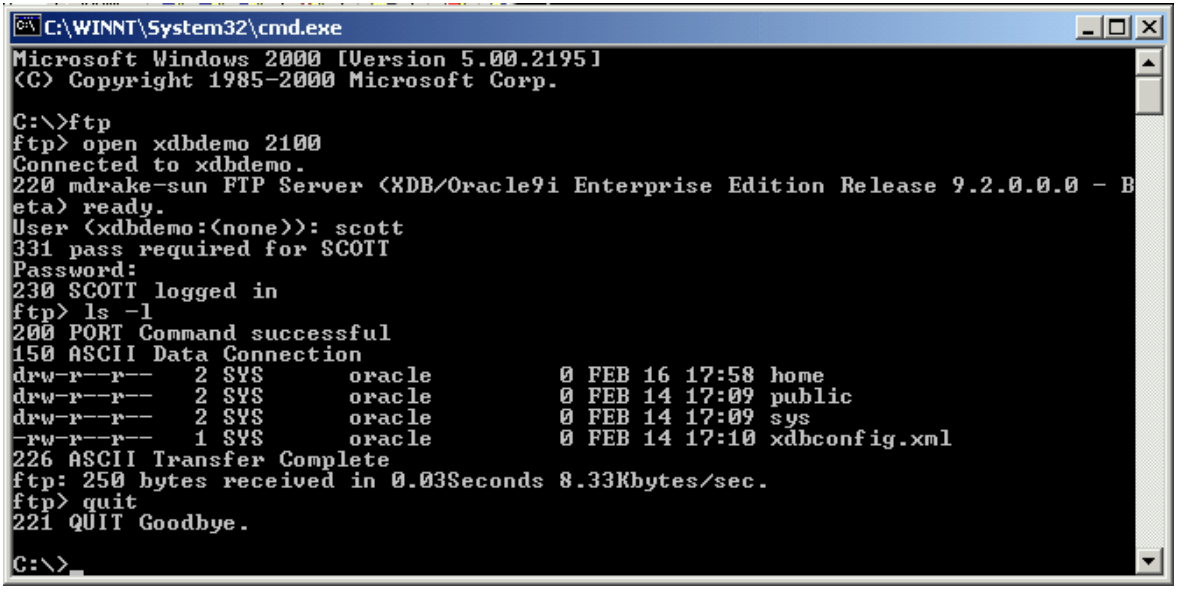


図 3-7 IE ブラウザの Web フォルダ・メニューからリポジトリのルート・レベルへのアクセス

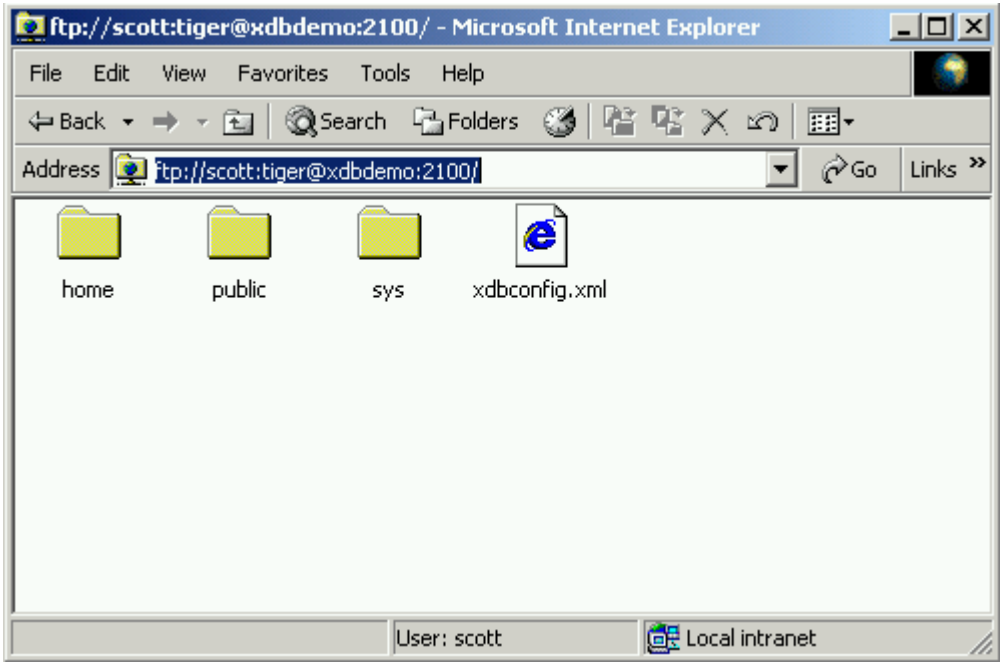


図 3-8 WS_FTP95LE FTP インタフェース・プログラムからリポジトリのルート・レベルへのアクセス

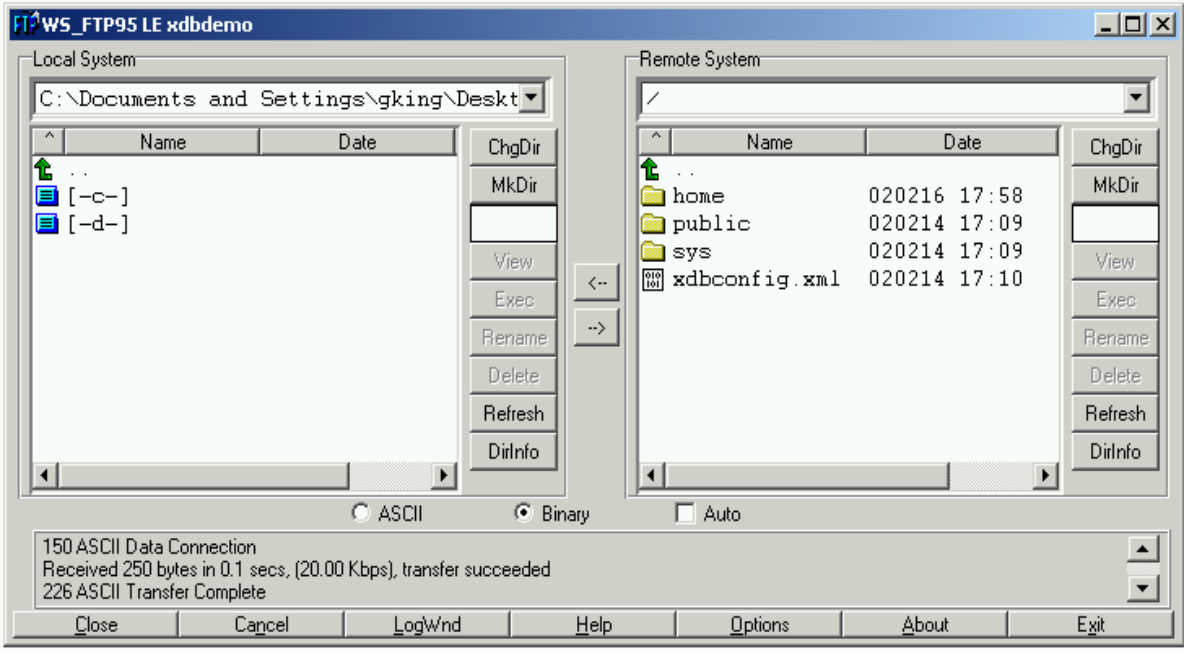
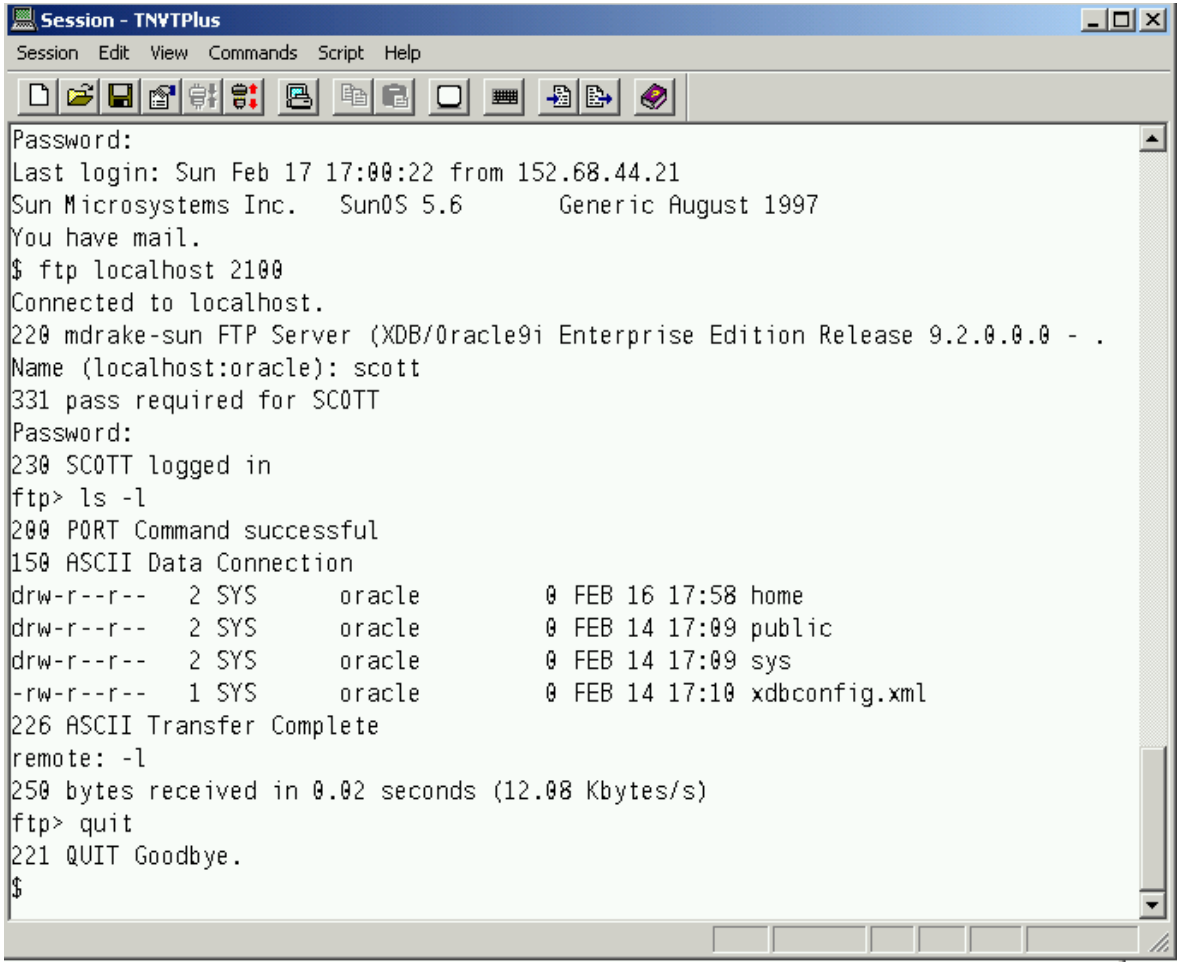


図 3-9 Telnet セッションからリポジトリのルート・レベルへのアクセス



HTTP/WebDAV プロトコル・サーバーの使用

Oracle XML DB Repository には、HTTP および WebDAV を使用してアクセスすることもできます。WebDAV のサポートによって、Microsoft Web フォルダ・クライアント、Microsoft Office、Macromedia 社の Dreamweaver などのアプリケーションで、直接 Oracle XML DB Repository にアクセスできます。図 3-10 および図 3-11 に、HTTP および WebDAV を使用してリポジトリにアクセスする例を示します。

図 3-10 HTTP/WebDAV を使用した Microsoft Windows エクスプローラからリポジトリへのアクセス

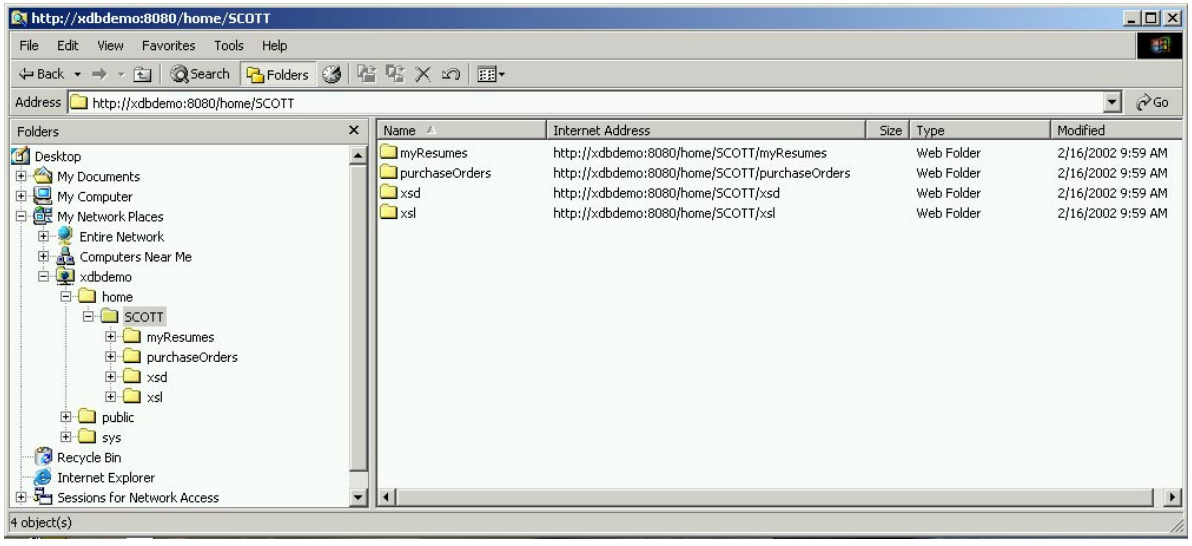
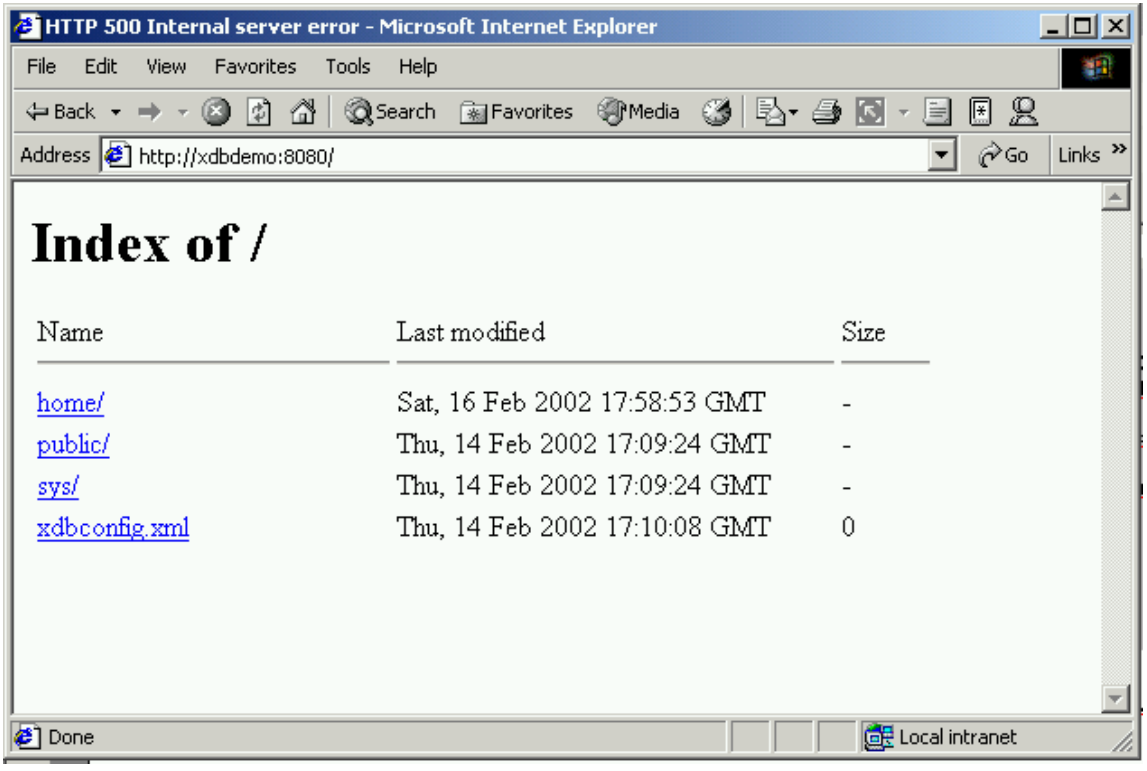


図 3-11 HTTP/WebDAV プロトコル・サーバーを使用した Microsoft Web フォルダ・クライアントからリポジトリへのアクセス



Oracle XML DB は、業界標準のプロトコルをサポートするため、一般的な標準インタフェースを使用して、Oracle9*i* データベースに格納されたデータおよびドキュメントに対してアップロードおよびアクセスできます。

第 II 部

Oracle XML DB からの XML データの格納および取出し

第 II 部では、Oracle XML DB を使用して XML データを格納、取出し、検証および変換する方法を説明します。第 II 部に含まれる章は、次のとおりです。

- 第 4 章「XMLType の使用」
- 第 5 章「XMLType の構造化されたマッピング」
- 第 6 章「XMLType データの変換および検証」
- 第 7 章「Oracle Text を使用した XML データの検索」

4

XMLType の使用

この章では、XMLType データ型の使用方法、XMLType 表および列の作成および操作方法、XMLType 表および列の問合せ方法について説明します。この章の内容は次のとおりです。

注意：

- **XML Schema に基づかない XMLType:** この章に示す XMLType 表および列は、XML Schema に基づきません。ただし、XMLType 表および列用に選択した記憶域オプションに関係なく、この章に示す方法および例を使用できます。記憶域の推奨項目の詳細は、第 3 章「Oracle XML DB の使用」を参照してください。
- **XML Schema に基づく XMLType:** XML Schema に基づく XMLType 表および列で作業する方法については、第 5 章「XMLType の構造化されたマッピング」および付録 B「XML Schema の手引き」を参照してください。

XMLType の概要

Oracle9*i* データベース リリース 1（9.0.1）では、新しい XMLType データ型が導入され、データベース内の XML データのネイティブな処理が簡単になりました。XMLType の概要を次に示します。

- XMLType は、PL/SQL ストアド・プロシージャのパラメータ、戻り値および変数として使用できます。
- XMLType は、XML 文書を SQL で（XMLType の）インスタンスとして表すことができます。
- XMLType には、XML コンテンツを操作する組込みメンバー関数が含まれています。たとえば、XMLType 関数を使用して、Oracle9*i* データベースに格納された XML データの作成、抽出および索引付けを行うことができます。
- 機能は、PL/SQL および Java で提供される一連の Application Program Interface（API）を介して使用することもできます。

XMLType およびこれらの機能を使用すると、SQL 開発者は、XML での作業中にリレーショナル・データベースの機能を使用できます。同様に、XML 開発者は、リレーショナル・データベースでの作業中に XML 標準の機能を使用できます。

XMLType データ型は、表の列およびビューのデータ型として使用できます。XMLType の変数は、PL/SQL ストアド・プロシージャのパラメータ、戻り値などとして使用できます。また、XMLType は、SQL、PL/SQL および（JDBC を介して）Java でも使用できます。

注意： Oracle9*i* データベース リリース 1（9.0.1）では、XMLType は、SQL、PL/SQL および Java を介してサーバーでのみサポートされていました。Oracle9*i* データベース リリース 2（9.2）では、XMLType は、SQL、Java および FTP や HTTP/WebDAV などのプロトコルを介してクライアント側でもサポートされています。

XML コンテンツを操作する多くの有効な関数が提供されています。これらの関数の多くは、XMLType の SQL 関数およびメンバー関数として提供されています。たとえば、extract() 関数は、XMLType インスタンスから特定のノード（複数の場合もあります）を抽出します。

SQL 問合せ内の XMLType は、システムの他のユーザー定義データ型と同じ方法で使用できます。

参照：

- 1-19 ページの「Oracle XML DB による複雑な XML 文書の高速格納および取出し」を参照してください。
- 第 26 章「Oracle XML DB Basic Demo」も参照してください。
- 『Oracle9*i* SQL リファレンス』の付録 D の「SQL 文での XML の使用方法」も参照してください。

XMLType データ型および API のメリット

XMLType データ型および API は、多くのメリットを提供します。このデータ型は、XML コンテンツに対する SQL 操作および SQL コンテンツに対する XML 操作を可能にします。

- **汎用 API:** XMLType には組込み関数、索引付けサポート、ナビゲーションなどが含まれているため、アプリケーション開発用の汎用 API が含まれています。
- **XMLType および SQL:** XMLType は、他の列およびデータ型と組み合せて SQL 文で使用できます。たとえば、XMLType 列を問い合せ、抽出の結果をリレーショナル列と結合できます。Oracle は、これらの問合せを実行するための最適な方法を判断できます。
- **XMLType を使用した最適化された評価 :** XMLType は、必要な場合以外は、XML データがツリー構造に生成されないように最適化されています。したがって、SQL 問合せで XMLType インスタンスを選択すると、シリアル化形式のみが関数を越えて交換されます。これらのインスタンスは、extract() や existsNode() などの操作が実行されるときのみ、ツリー形式に分解されます。XMLType の内部構造も、DOM に類似した最適化されたツリー構造です。

- **索引付け :** Oracle Text 索引は、XMLType 列をサポートするように拡張されています。また、existsNode() および extract() にもファンクション索引を作成して、問合せの評価を高速化できます。

  参照 : 第 10 章「データベースからの XML データの生成」を参照してください。

XMLType を使用する場合

XMLType は、次の操作を実行する必要がある場合に使用します。

- XML 文書の一部または全体の SQL 問合せ : existsNode() 関数および extract() 関数を使用すると、XML 文書に必要な SQL 問合せ関数を実行できます。
- SQL 文および PL/SQL ファンクションでの厳密な型指定 : 厳密な型指定とは、渡される値が常に XML 値であり、任意のテキスト文字列ではないことを意味します。
- extract() 関数および existsNode() 関数で提供される XPath 機能 : XMLType は、組込みの C 言語で書かれた XML パーサーおよび XSLT プロセッサを使用するため、サーバー内部で使用されるとパフォーマンスおよび拡張性が向上します。
- 文書の XPath 検索の索引付け : XMLType には、ファンクション索引を作成して検索を最適化するためのメンバー関数があります。
- 記憶域モデルからのアプリケーションの保護 : CLOB またはリレーショナル記憶域のかわりに XMLType を使用すると、アプリケーションでの問合せまたは DML 文に影響することなく、後でアプリケーションが様々な代替の記憶域に効率的に移行できます。
- 将来の最適化への準備 : XML の新機能は、XMLType をサポートします。Oracle9*i* データベースは、XMLType が XML データを格納できることをネイティブに認識するため、より優れた最適化および索引付けの方法を実現できます。XMLType を使用するアプリケーションを作成することによって、アプリケーションを作成しなおさなくても、これらの最適化および拡張を将来のリリースで簡単に実現および保持できます。

Oracle XML DB への XMLType データの格納

XMLType データは、次に示す 2 つの方法のいずれかまたはそれらを組み合せて格納できます。

- **ラージ・オブジェクト（LOB）内 :** LOB 記憶域では、元の XML（空白を含むすべて）に対するコンテンツの精度が維持されます。この場合、XML 文書は文書全体（ファイルなど）として構成され、格納されます。今回のリリースでは、XMLType は XML Schema に基づかない記憶域用に CLOB 記憶域オプションを提供します。将来のリリースでは、BLOB や NCLOB など、他の記憶域オプションが提供される可能性もあります。また、XML Schema に基づく記憶域用に CLOB ベースの記憶域を作成することもできます。

XML Schema 仕様を使用せずに XMLType 列を作成すると、XML データを格納するために自動的に非表示 CLOB 列が作成されます。XMLType 列自体は、この非表示 CLOB 列上の仮想列になります。CLOB 列には直接アクセスできませんが、XMLType STORAGE 句を使用してその列の記憶特性を設定できます。

- **構造化記憶域内（表およびビュー内）:** 構造化記憶域では、ドキュメント・オブジェクト・モデル（DOM）に対する再現性が保持されます。この場合、XML 文書はオブジェクト・リレーショナル表またはビューに分解されます。XMLType は、SQL オブジェクトや Java オブジェクトが通常提供しない次のような情報を維持することによって、DOM 再現性を実現します。
  - 子要素および属性の順序付け
  - 要素と属性の区別
  - スキーマで宣言された非構造化コンテンツ（content="mixed"、<any> 宣言など）
  - 処理命令、コメント、名前空間宣言などのインスタンス・ドキュメント内の宣言されていないデータ
  - SQL では使用できない基本 XML データ型のサポート（BOOLEAN、QNAME など）
  - 列挙リストなど、SQL によって直接サポートされない XML 制約（ファセット）のサポート

ネイティブな XMLType インスタンスには、SQL オブジェクト・モデルに完全には収まらないこの追加情報を格納する非表示列が含まれます。この情報には、extractNode() などのメンバー関数を使用して、SQL または Java で API を介してアクセスできます。

データベースのインポートおよびエクスポートを使用すると、XMLType 記憶域を構造化記憶域と LOB 記憶域の間で切り替えることができます。アプリケーション・コードを変更する必要はありません。各記憶域オプションにはメリットがあるため、アプリケーションをチューニングするときに XML 記憶域オプションを変更できます。

Oracle XML DB の XML 記憶域オプションのメリットとデメリット

表 4-1 に、Oracle XML DB の記憶域オプションを選択する場合に考慮する必要があるメリットとデメリットの概要を示します。

表 4-1 Oracle XML DB の XML 記憶域オプション

機能	LOB 記憶域（Oracle Text 索引付き）	構造化記憶域（B* ツリー索引付き）
データベース・スキーマに対する柔軟性	スキーマの変更に対して柔軟です。	スキーマの変更に対する柔軟性が制限されます。ALTER TABLE 制限と同様です。
データ整合性および精度	元の XML がバイト単位で保持されます。これは一部のアプリケーションでは重要です。	後続の新しい行、タグ内の空白、および非 STRING（文字列）データ型のデータ・フォーマットは失われます。ただし、DOM 再現性は保持されます。
パフォーマンス	一般的な DML パフォーマンスを示します。	高い DML パフォーマンスを示します。
SQL へのアクセス	SQL 機能へのある程度のアクセス性を示します。	制約、索引などの既存の SQL 機能への高いアクセス性を示します。
必要な領域	大量の領域を消費する場合があります。	特に Oracle XML DB に登録された XML Schema とともに使用した場合、必要な領域が少なくなります。

XMLType に CLOB 記憶域を使用する場合

次の場合は、`XMLType` に CLOB 記憶域を使用します。

- XML を文書全体としてデータベースに格納し、文書全体として取り出す必要がある場合。
- XML 文書に対してピース単位更新を実行する必要がない場合。

注意： XMLType および VARRAY:

- VARRAY は表に格納されるときには CLOB をサポートしないため、`XMLType` の VARRAY を作成して、データベースに格納することはできません。
- `XMLType` を含む VARRAY 型の列は作成できません。これは、Oracle が VARRAY 内の LOB ロケータをサポートしないためです。

参照：

- 第 2 章「Oracle XML DB を使用する前に」を参照してください。
- 3-23 ページの「XML の格納 : 構造化記憶域または非構造化記憶域」も参照してください。
- XMLType データの生成方法については、第 10 章「データベースからの XML データの生成」を参照してください。

XMLType メンバー関数

Oracle9*i* データベース リリース 1（9.0.1）では、XMLType 値を操作するいくつかの SQL 関数および XMLType メンバー関数が導入されています。Oracle9*i* データベース リリース 2（9.2）では、機能が拡張されています。リリース 2（9.2）では、いくつかの新しい SQL 関数および XMLType メンバー関数が提供されています。

参照：

- 付録 F「Oracle XML DB の XMLType API、PL/SQL API および Resource PL/SQL API: クイック・リファレンス」を参照してください。
- すべての XMLType 関数とメンバー関数、およびそれらの構文と説明は、『Oracle9*i* XML API リファレンス - XDK および Oracle XML DB』を参照してください。

すべての XMLType 関数は、組込み C 言語で書かれた XML パーサーおよび XSLT プロセッサを使用して、XML データを解析して妥当性を検証し、そのデータに XPath 式を適用します。また、最適化されたメモリー内 DOM ツリーを使用して、XML 文書や XML フラグメントの抽出などの処理を実行します。

参照： 付録 C「XPath および Namespace の手引き」を参照してください。

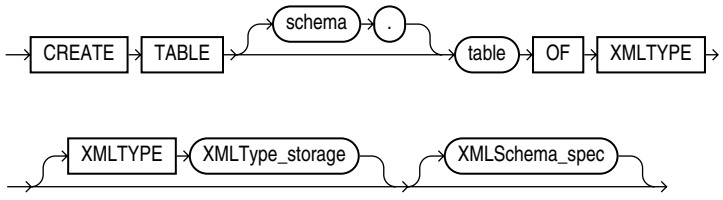
XMLType API の使用方法

XMLType API を使用すると、表および列を作成できます。XMLType API の createXML() 静的関数を使用すると、XMLType インスタンスを作成して挿入できます。XML 文書を XMLType として格納することによって、標準の SQL 問合せを使用して簡単に XML コンテンツを検索できます。

図 4-1 に、XMLType 表を作成するための構文を示します。

```
CREATE TABLE [schema.] table OF XMLTYPE
  [XMLTYPE XMLType_storage] [XMLSchema_spec];
```

図 4-1 XMLType 表の作成



この項では、XMLType 列を作成し、それを SQL 文で使用する方法、および XMLType 表を作成する方法を示す簡単な例を示します。

XMLType 列の作成、追加および削除

次に、XMLType 列を作成、追加および削除する例を示します。

例 4-1 XMLType の作成 : XMLType 列の作成

XMLType 列は、他のユーザー定義型の列と同様に作成できます。

```
CREATE TABLE warehouses(
  warehouse_id NUMBER(4),
  warehouse_spec XMLTYPE,
  warehouse_name VARCHAR2(35),
  location_id NUMBER(4));
```

例 4-2 XMLType の作成 : XMLType 列の作成

前述のとおり、XMLType を単にデータ型として使用することによって、XMLType 列を作成できます。次の文は、XMLType の発注書列 poDoc を作成します。

```
CREATE TABLE po_xml_tab(
  poid number,
  poDoc XMLTYPE);

CREATE TABLE po_xtab of XMLType; -- this creates a table of XMLType. The default
                                 -- is CLOB based storage.
```

例 4-3 XMLType 列の追加

表を変更して XMLType 列を追加することもできます。この方法は、他のデータ型と同様です。次の文は、表に新しい顧客ドキュメント列を追加します。

```
ALTER TABLE po_xml_tab add (custDoc XMLType);
```

例 4-4 XMLType 列の削除

他のデータ型の方法と同様に、表を変更して XMLType 列を削除できます。次の文は、custDoc 列を削除します。

```
ALTER TABLE po_xml_tab drop (custDoc);
```

XMLType 列への値の挿入

XMLType 列に値を挿入するには、XMLType インスタンスをバインドする必要があります。

例 4-5 XMLType() コンストラクタを使用した XMLTYpe への挿入

XMLType インスタンスは、XMLType() コンストラクタを使用して、VARCHAR またはキャラクタ・ラージ・オブジェクト（CLOB）から簡単に作成できます。

```
INSERT INTO warehouses VALUES
   (       100, XMLType(
               '<Warehouse whNo="100">
                <Building>Owned</Building>
                </Warehouse>'), 'Tower Records', 1003);
```

この例では、文字列リテラルから XMLType インスタンスを作成しています。XMLType() コンストラクタへの入力には、VARCHAR2 または CLOB を戻す、すべての式を使用できます。XMLType() コンストラクタは、入力 XML が整形式かどうかも確認します。

SQL 文での XMLType の使用

次の単純な SELECT 文は、SQL 文で XMLType を使用する方法を示します。

例 4-6 SELECT 文での XMLType の使用

```
SELECT
  w.warehouse_spec.extract('/Warehouse/Building/text()').getStringVal()
     "Building"
  FROM warehouses w;
```

この場合の warehouse_spec は、メンバー関数 extract() によって操作される XMLType 列です。この単純な問合せの結果は、次の文字列（VARCHAR2）になります。

```
Building
-----------------
Owned
```

参照： 4-7 ページの「XMLType API の使用方法」を参照してください。

XMLType 列の更新

XMLType の XML 文書は CLOB 内に格納できます。その後、更新して既存の文書全体を置換する必要があります。

例 4-7 XMLType の更新

XML 文書を更新するには、標準的な SQL の UPDATE 文を実行します。次に示すとおり、XMLType インスタンスをバインドする必要があります。

```
UPDATE warehouses SET warehouse_spec = XMLType
                  ('<Warehouse whono="200">
                    <Building>Leased</Building>
                    </Warehouse>');
```

この例では、文字列リテラルから XMLType インスタンスを作成し、warehouse_spec 列を新しい値で更新します。

注意： UPDATE 文ではすべてのトリガーが起動され、そのトリガー内で XML 値を確認および変更できます。

XMLType 列を含む行の削除

XMLType 列を含む行を削除する方法は、他のデータ型を含む行を削除する場合と同じです。

例 4-8 XMLType 列を含む行の削除

extract() 関数および existsNode() 関数を使用しても、削除する行を識別できます。たとえば、ウェアハウスの建物が Leased になっているすべての warehouse 行を削除するには、次の文を使用します。

```
DELETE FROM warehouses e
   WHERE e.warehouse_spec.extract('//Building/text()').getStringVal()
           = 'Leased';
```

注意： 今回のリリースでは、Oracle は XMLType を sys.XMLType のパブリック・シノニムとしてサポートします。XMLType は、一連のユーザー定義コンストラクタ（createXML 静的関数を反映したもの）もサポートします。次に例を示します。

- Oracle9*i* データベース リリース 1（9.0.1）では、次の構文を使用できました。
  ```
  sys.XMLType.createXML('<Warehouse whNo="100">...)
  ```
- Oracle9*i* データベース リリース 2（9.2）では、次の省略した形式を使用できます。
  ```
  XMLType('<Warehouse whNo="100">...)
  ```

XMLType 表および列を使用する場合のガイドライン

次の項では、XML データを XMLType 表および列に格納する場合のガイドラインについて説明します。

XMLType 表および列の定義

XMLType 表および列を定義します。表定義および列定義にはオプションの記憶特性を含めることができます。

注意： Oracle の今回のリリースでは、XMLType 表の作成がサポートされています。これらの表へのオブジェクト参照（REF）を作成し、オブジェクト・キャッシュで使用できます。

XMLType インスタンスの作成

XML データを列および表に挿入する前に、XMLType コンストラクタを使用して、XMLType インスタンスを作成します。他にも、XMLType を戻す様々な関数があります。

参照： 例については、10-48 ページの「SYS_XMLGEN(): XMLType インスタンスの変換」を参照してください。

特定の XMLType インスタンスの選択または抽出

XMLType インスタンスは列から選択できます。XMLType では、メンバー関数 extract() を使用して特定のノードを抽出したり、メンバー関数 existsNode() を使用してノードが存在するかどうかを確認することができます。『Oracle9*i* XML API リファレンス - XDK および Oracle XML DB』の XMLType メンバー関数の表を参照してください。

参照： 次の項を参照してください。

- 4-17 ページの「getClobVal() を使用した XMLType 列の選択」
- 4-29 ページの「extract() を使用した XMLType からのフラグメントの抽出」

Oracle Text 索引の定義

XMLType 列に Oracle Text 索引を定義できます。これによって、その列で CONTAINS、HASPATH、INPATH およびその他のテキスト演算子を使用できます。LOB 列を操作するすべての Oracle Text 演算子および索引関数は、XMLType 列も操作します。

XPath 索引 CTXXPATH の定義

今回のリリースでは、新しい Oracle Text 索引タイプ CTXXPATH が導入されています。この索引タイプは、existsNode() による索引付けの実装に有効で、述語内での existsNode() の評価を最適化します。

参照： 次の項、章またはマニュアルを参照してください。

- 4-37 ページの「XMLType 列の索引付け」
- 第 7 章「Oracle Text を使用した XML データの検索」
- 第 10 章「データベースからの XML データの生成」
- 『Oracle9*i* アプリケーション開発者ガイド - ラージ・オブジェクト』

XMLType 列に対する記憶特性の指定

XMLType 列の XML データは CLOB 列として格納できます。したがって、その列には、LOB の記憶特性を指定することもできます。4-8 ページの例「XMLType の作成 : XMLType 列の作成」では、warehouse_spec 列は XMLType 列です。

例 4-9 XMLType 表作成時の記憶域の指定

次のとおり、表を作成するときにこの列の記憶特性を指定できます。

```
CREATE TABLE po_xml_tab(
     poid NUMBER(10),
     poDoc XMLTYPE
     )
     XMLType COLUMN poDoc
        STORE AS CLOB (
            TABLESPACE lob_seg_ts
            STORAGE (INITIAL 4096 NEXT 4096)
            CHUNK 4096 NOCACHE LOGGING
         );
```

表に列を追加するときは、STORE AS 句もサポートされます。

例 4-10 XMLType 列の追加および記憶域の指定

この表に新しい XMLType 列を追加し、その列に STORAGE 句を指定するには、次の SQL 文を使用します。

```
ALTER TABLE po_xml_tab  add(
     custDoc XMLTYPE
   )
   XMLType COLUMN custDoc
      STORE AS CLOB (
           TABLESPACE lob_seg_ts
           STORAGE (INITIAL 4096 NEXT 4096)
           CHUNK 4096 NOCACHE LOGGING
         );
```

XMLData を使用した XMLType 列の記憶域オプションの変更

XML Schema に基づかない記憶域では、XMLDATA を使用して、XMLType 列の記憶特性を変更できます。

例 4-11 XMLDATA を使用した XMLType 列の記憶特性の変更

たとえば、表 foo_tab について考えてみます。

```
CREATE TABLE foo_tab (a xmltype);
```

foo_tab の LOB 列 a の記憶特性を変更するには、次の文を使用します。

```
ALTER TABLE foo_tab MODIFY LOB (a.xmldata) (storage (next 5K) cache);
```

XMLDATA は、この列の内部記憶域を識別します。CLOB ベースの記憶域の場合、これは CLOB 列に対応します。XML Schema に基づく記憶域の場合も同様です。XMLDATA を使用すると、構造化記憶域を調べ、その値を変更することができます。

> **注意：** 今回のリリースでは、XMLType の列の内部記憶域に記憶特性、制約などを直接指定できるように、XMLDATA 属性を使用してその列にアクセスできます。

STORAGE 句の他に、制約および索引で XMLDATA 属性を使用できます。

参照： LOB 記憶域オプションの詳細は、『Oracle9*i* アプリケーション開発者ガイド - ラージ・オブジェクト』および『Oracle9*i* SQL リファレンス』を参照してください。

XMLType 列に対する制約の指定

XMLType 列には NOT NULL 制約を指定できます。

例 4-12 XMLType 列に対する制約の指定

```
CREATE TABLE po_xml_tab (
  poid number(10),
  poDoc XMLType NOT NULL
);
```

これによって、次のような挿入が回避されます。

```
INSERT INTO po_xml_tab (poDoc) VALUES (null);
```

例 4-13 ALTER TABLE を使用した XMLType 列の NOT NULL の変更

ALTER TABLE 文を使用して、他の型の列の場合と同じ方法で XMLType 列の NOT NULL 情報を変更することもできます。

```
ALTER TABLE po_xml_tab MODIFY (poDoc NULL);
ALTER TABLE po_xml_tab MODIFY (poDoc NOT NULL);
```

XMLType 列には CHECK 制約を定義することもできます。このデータ型では、他のデフォルト値はサポートされていません。

XMLType 列 / 表の XML データの操作

XMLType はユーザー定義のデータ型で、ファンクションが定義されているため、XMLType にファンクションをコールし、結果を取得することができます。XMLType は、表の列、ビュー、トリガー本体、型定義などのユーザー定義型を使用している場所では常に使用できます。

XMLType 列および表の XML データでは、次のデータ操作言語（DML）操作を実行できます。

- XMLType 列 / 表への XML データの挿入
- XML データの選択および問合せ
- XML インスタンスおよび表と列のデータの更新
- XML データの削除

XMLType 列 / 表への XML データの挿入

次の方法で、データを XMLType 列に挿入できます。

- INSERT 文（SQL、PL/SQL および Java）を使用する方法。
- SQL*Loader を使用する方法。第 22 章「Oracle XML DB への XML データのロード」を参照してください。

XMLType 列には整形式の XML 文書のみ格納できます。XMLType 列には、フラグメントおよびその他の非整形式の XML は格納できません。

INSERT 文の使用

INSERT 文を使用して XML データを XMLType に挿入するには、最初に XML 文書を作成し、それを使用して挿入を実行する必要があります。次の方法で、挿入可能な XML 文書を作成できます。

- XMLType コンストラクタを使用する方法。SQL、PL/SQL および Java で実行できます。
- XMLElement()、XMLConcat()、XMLAGG() などの SQL 関数を使用する方法。SQL、PL/SQL および Java で実行できます。

例 4-14 CLOB で XMLType() コンストラクタを使用した XML データの挿入

次の例では、INSERT...SELECT および XMLType コンストラクタを使用して XML 文書を作成し、その文書を XMLType 列に挿入します。XML 文書を格納する CLOB 型の poClob 列を含む表 po_clob_tab について考えてみます。

```
CREATE TABLE po_clob_tab
(
  poid number,
  poClob CLOB
);

-- some value is present in the po_clob_tab
INSERT INTO po_clob_tab
     VALUES(100, '<?xml version="1.0"?>
                    <PO pono="1">
                       <PNAME>Po_1</PNAME>
                       <CUSTNAME>John</CUSTNAME>
                       <SHIPADDR>
                          <STREET>1033, Main Street</STREET>
                          <CITY>Sunnyvalue</CITY>
                          <STATE>CA</STATE>
                       </SHIPADDR>
                    </PO>');
```

例 4-15 XMLType インスタンスを使用した XML データの挿入

別の po_clob_tab に格納されている CLOB データから XML インスタンスを作成するのみで、発注書の XML 文書を po_xml_tab 表に挿入できます。

```
INSERT INTO po_xml_tab
       SELECT poid, XMLType(poClob)
       FROM po_clob_tab;
```

> **注意：** CLOB 値は、一時 CLOB を作成できる関数、または他の表やビューから CLOB を選択できる関数を含むすべての式から取得できます。

例 4-16 文字列で XMLType() を使用した XML データの挿入

次の例では、XMLType コンストラクタを使用して po_tab 表に発注書を挿入します。

```
INSERT INTO po_xml_tab
   VALUES(100, XMLType('<?xml version="1.0"?>
                       <PO pono="1">
                          <PNAME>Po_1</PNAME>
                          <CUSTNAME>John</CUSTNAME>
                          <SHIPADDR>
                             <STREET>1033, Main Street</STREET>
                             <CITY>Sunnyvalue</CITY>
                             <STATE>CA</STATE>
                          </SHIPADDR>
                       </PO>'));
```

例 4-17 XMLElement() を使用した XML データの挿入

次の例では、SQL 関数 XMLElement() を使用して生成された発注書を、po_xml_tab 表に挿入します。発注書は、発注情報オブジェクトを含むオブジェクト・ビューと想定します。発注書ビューの完全な定義については、10-34 ページの「DBMS_XMLGEN: XML 形式でのデータベースからの発注書の生成」を参照してください。

```
INSERT INTO po_xml_tab
  SELECT XMLelement("po", value(p))
       FROM po p
       WHERE p.pono=2001;
```

XMLElement() は発注情報オブジェクトから XMLType を作成し、作成された XMLType は po_xml_tab 表に挿入されます。SYS_XMLGEN() を INSERT 文で使用することもできます。

XML データの選択および問合せ

次の方法で、XMLType 列から XML データを問い合せることができます。

- SQL、PL/SQL または Java を介して XMLType 列を選択する方法。
- 直接 XMLType 列を問い合せ、extract() および existsNode() を使用する方法。
- Oracle Text 演算子を使用して XML コンテンツを問い合せる方法。4-37 ページの「XMLType 列の索引付け」および第 7 章「Oracle Text を使用した XML データの検索」を参照してください。

XML データを操作するための SQL 関数

existsNode()、extract()、XMLTransform()、updateXML() などの SQL 関数は、SQL 内で XML データを操作します。XMLType データ型は、これらのほとんどの SQL 関数をメンバー関数としてサポートします。自己参照的に起動することも、SQL 関数を使用して起動することもできます。

XML データの選択

XMLType データは、PL/SQL または Java を使用して選択できます。また、getClobVal()、getStringVal() または getNumberVal() 関数を使用して、XML をそれぞれ CLOB、VARCHAR または NUMBER として取得することもできます。

例 4-18 getClobVal() を使用した XMLType 列の選択

次の例では、SQL*Plus を使用して XMLType 列を選択します。

```
SET long 2000

SELECT e.poDoc.getClobval() AS poXML
   FROM po_xml_tab e;



POXML
---------------------
<?xml version="1.0"?>
<PO pono="2">
   <PNAME>Po_2</PNAME>
   <CUSTNAME>Nance</CUSTNAME>
   <SHIPADDR>
      <STREET>2 Avocet Drive</STREET>
      <CITY>Redwood Shores</CITY>
      <STATE>CA</STATE>
   </SHIPADDR>
</PO>
```

XML データの問合せ

existsNode() 関数および extract() 関数を使用して、XMLType データを問い合せて、その一部を抽出できます。これらの関数は、W3C の XPath 勧告のサブセットを使用して、ドキュメントをナビゲートします。

XML 文書を検索するための XPath 式の使用

XPath は、XML 文書をナビゲートするための W3C 勧告です。XPath は、XML 文書をノードのツリーとしてモデル化します。XPath は、ツリー内を移動し、述語およびノード・テスト関数を適用するための、豊富な操作を提供します。XPath 式を XML 文書に適用すると、ノードの集合が戻されます。たとえば、/PO/PONO は、文書の「PO」ルート要素の下にあるすべての「PONO」子要素を選択します。

表 4-2 に、XPath で使用する共通構造体を示します。

表 4-2 XPath の共通構造体

XPath 構造体	説明
/	XPath 式のツリーのルートを示します。たとえば、/PO は、名前が「PO」であるルート・ノードの子を示します。
/	任意のノードの子ノードを識別するための区切りとしても使用します。たとえば、/PO/PNAME は、ルート要素の子である発注書名要素を示します。
//	現行ノードのすべての子孫を識別するために使用します。たとえば、PO//ZIP は、「PO」要素の下にあるすべての「ZIP」要素と一致します。
*	任意の子ノードと一致させるためのワイルド・カードとして使用します。たとえば、/PO/*/STREET は、「PO」要素の孫であるすべての「STREET」要素と一致します。
[ ]	述語式を示すために使用します。XPath は OR、AND、NOT など、様々なバイナリ演算子をサポートしています。たとえば、/PO[PONO=20 and PNAME="PO_2"]/SHIPADDR は、発注書番号が 20 で、発注書名が「PO_2」であるすべての発注書の出荷先要素を選択します。[ ] は、リスト内の索引を示す場合にも使用します。たとえば、/PO/PONO[2] は「PO」ルート要素の下の 2 番目の発注書番号要素を示します。

XPath は単一または一連の要素、テキストまたは属性ノードを識別する必要があります。XPath の結果はブール式であってはなりません。

> **参照：** 付録 C「XPath および Namespace の手引き」を参照してください。

XMLType メンバー関数を使用した XML データの問合せ

XMLType データは、PL/SQL、OCI または Java を介して選択できます。また、getClobVal()、getStringVal() または getNumberVal() 関数を使用して、XML をそれぞれ CLOB、VARCHAR または NUMBER として取得することもできます。

例 4-19 getClobVal() および existsNode() を使用した CLOB としての XML 文書の取出し

次の例では、getClobVal() および existsNode() を使用して XMLType 列を選択します。

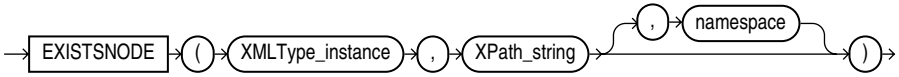
```
set long 2000

SELECT e.poDoc.getClobval() AS poXML
   FROM po_xml_tab e
   WHERE e.poDoc.existsNode('/PO[PNAME = "po_2"]') = 1;

POXML
--------------------
<?xml version="1.0"?>
<PO pono="2">
   <PNAME>Po_2</PNAME>
   <CUSTNAME>Nance</CUSTNAME>
   <SHIPADDR>
      <STREET>2 Avocet Drive</STREET>
      <CITY>Redwood Shores</CITY>
      <STATE>CA</STATE>
   </SHIPADDR>
</PO>
```

existsNode 関数

existsNode() 関数の構文は、図 4-2 および次に示すとおりです。

```
existsNode(XMLType_instance IN XMLType,
           XPath_string IN VARCHAR2, namespace_string IN varchar2 := null) RETURN
NUMBER
```

図 4-2 existsNode() の構文



XMLType に existsNode() 関数を使用すると、任意の XPath 評価の結果が、少なくとも単一の XML 要素またはテキスト・ノードになるかどうかが確認されます。単一の XML 要素またはテキスト・ノードになる場合は数値 1 が戻り、それ以外の場合は 0（ゼロ）が戻りま

す。namespace を使用すると、XPath_string に指定された接頭辞の、対応する名前空間へのマッピングを識別できます。

例 4-20 XMLType に対する existsNode() の使用

たとえば、次の XML 文書を考えてみます。

```
<PO>
  <PONO>100</PONO>
  <PNAME>Po_1</PNAME>
  <CUSTOMER CUSTNAME="John"/>
  <SHIPADDR>
     <STREET>1033, Main Street</STREET>
     <CITY>Sunnyvalue</CITY>
     <STATE>CA</STATE>
  </SHIPADDR>
</PO>
```

/PO/PNAME のような XPath 式の結果は、単一のノードになるため、existsNode() はその XPath に対して 1 を戻します。これは、結果が単一のテキスト・ノードになる /PO/PNAME/text() の場合も同じです。

/PO/POTYPE のような XPath 式はノードを戻さないため、この式では existsNode() は数値の 0（ゼロ）を戻します。

したがって、existsNode() メンバー関数は、問合せ内で使用したり、ファンクション索引を作成して問合せの評価を高速化するために使用することができます。

例 4-21 existsNode() を使用したノードの検索

次の例では、サンプル表 oe.warehouses の warehouse_spec 列の XML パス内に /Warehouse/Dock ノードが存在するかどうかをテストします。

```
SELECT warehouse_id, EXISTSNODE(warehouse_spec, '/Warehouse/Docks')
   "Loading Docks"
   FROM warehouses
   WHERE warehouse_spec IS NOT NULL;

WAREHOUSE_ID Loading Docks
------------ -------------
           1             1
           2             1
           3             0
           4             1
```

索引を使用した existsNode() の評価

existsNode() を使用してファンクション・ベース索引を作成すると、実行を高速化できます。また、CTXXPATH 索引を作成して、任意の XPath 検索を高速化することもできます。

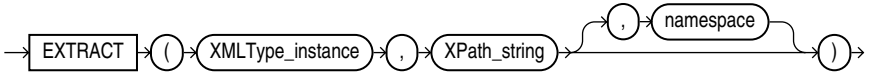
参照： 4-40 ページの「XMLType 列での XPath 索引の作成 : CTXXPATH 索引」を参照してください。

extract() 関数

extract() 関数は、existsNode() 関数と類似しています。この関数は、オプションの名前空間パラメータを含む XPath 文字列 VARCHAR2 を適用し、XML フラグメントを含む XMLType インスタンスを戻します。構文は、図 4-3 および次に示すとおりです。

```
extract(XMLType_instance IN XMLType, XPath_string IN VARCHAR2,
                namespace_string In varchar2 := null) RETURN XMLType;
```

図 4-3 extract() の構文



XMLType に extract() を使用すると、XPath 式によって識別された文書からノードまたはノードの集合が抽出されます。抽出されるノードは、要素、属性またはテキスト・ノードの場合があります。すべてのテキスト・ノードは、抽出時に単一のテキスト・ノード値に縮小されます。namespace を使用して、XPath 文字列内の接頭辞の名前空間情報を指定できます。

extract() を介して XPath を適用した結果の XMLType は、整形式の XML 文書である必要はありませんが、場合によってはノードの集合または単純なスカラー・データを含んでいる可能性があります。XMLType に getStringVal() または getNumberVal() メソッドを使用すると、このスカラー・データを抽出できます。

たとえば、XPath 式 /PO/PNAME は、前述の XML 文書内の PNAME 要素を識別します。一方、式 /PO/PNAME/text() は、PNAME 要素のテキスト・ノードを参照します。

注意： 後者の式でも、XMLType とみなされます。実際には XMLtype インスタンスにテキスト以外は含まれない場合でも、たとえば、extract(poDoc,'/PO/PNAME/text()') は XMLType インスタンスを戻します。getStringVal() を使用すると、VARCHAR2 の結果としてテキスト値を抽出できます。

---

getStringVal() または getNumberVal() を使用してテキスト・ノードを SQL データに変換する前に、text() ノード・テスト関数を使用して、要素内のテキスト・ノードを識別します。text() ノードを使用しない場合、XML フラグメントが生成されます。

たとえば、次の XPath 式を考えてみます。

- /PO/PNAME は、フラグメント <PNAME>PO_1</PNAME> を識別します。
- /PO/PNAME/text() は、テキスト値「PO_1」を識別します。

XML 文書内で要素が繰り返されている場合、索引作成メカニズムを使用して、個々の要素を識別できます。たとえば、次の XML 文書を考えてみます。

```
<PO>
  <PONO>100</PONO>
  <PONO>200</PONO>
</PO>
```

この場合、

- //PONO[1] を使用すると、最初の「PONO」要素（値は 100）を識別できます。
- //PONO[2] を使用すると、2 番目の「PONO」要素（値は 200）を識別できます。

extract() の結果は常に XMLType です。XPath を適用することによって空のセットが生成される場合、extract() は NULL 値を戻します。

したがって、extract() メンバー関数は、次のような多くの場合に使用できます。

- 数値を抽出し、その数値にファンクション索引を作成することによる処理の高速化
- SQL 文の FROM 句で使用するコレクション式の抽出
- 後で集計して別の文書を生成するためのフラグメントの抽出

例 4-22 extract() を使用したノード値の抽出

次の例では、表 oe.warehouses の列 warehouse_spec のノード /Warehouse/Docks の値を抽出します。

```
SELECT warehouse_name,
   extract(warehouse_spec, '/Warehouse/Docks').getStringVal()
   "Number of Docks"
   FROM warehouses
   WHERE warehouse_spec IS NOT NULL;

WAREHOUSE_NAME       Number of Docks
-------------------- --------------------
Southlake, Texas          <Docks>2</Docks>
San Francisco             <Docks>1</Docks>
New Jersey                <Docks/>
Seattle, Washington       <Docks>3</Docks>
```

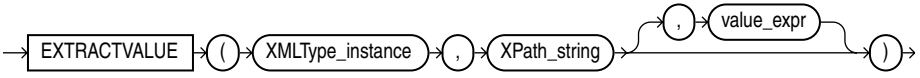
extractValue() 関数

extractValue() 関数は、引数として XMLType インスタンスおよび XPath 式を取ります。この関数は、XMLType インスタンスの XPath 評価の結果に対応するスカラー値を戻します。extractValue() の構文は、図 4-4 を参照してください。

- **XML Schema に基づく文書 :** XML Schema に基づく文書では、Oracle9*i* データベースが戻り値の型を推測できる場合、該当する型のスカラー値が戻されます。それ以外の場合、結果は VARCHAR2 型です。
- **XML Schema に基づかない文書 :** XML Schema 基づかない文書では、戻り型は常に VARCHAR2 です。

extractValue() は、文書の XML Schema から正しい戻り型を推測しようとします。XMLType が XML Schema 基づかない場合、または正しい戻り型を判断できない場合、Oracle XML DB は VARCHAR2 を戻します。

図 4-4 extractValue() の構文



ショートカット関数

extractValue() を使用すると、同等の extract 関数を使用した場合より簡単に必要な値を抽出できます。この関数は、簡単に使用できるショートカット関数です。したがって、この関数を次の関数のかわりに使用します。

```
extract(x,'path/text()').get(string|number)val()
```

extract().getStringVal() や extract().getnumberval() は、次のとおり extractValue() に置き換えることができます。

```
extractValue(x, 'path/text()')
```

extractValue() では、text() を省略できます。ただし、これは 'path' 部分が指すノードが 1 つの子のみを持ち、その子がテキスト・ノードである場合のみです。それ以外の場合は、エラーが発生します。

extractValue() の構文は extract() の構文と同じです。

extractValue() の特性

extractValue() には、次の特性があります。

- 常に NUMBER...VARCHAR2 などのスカラー・コンテンツのみを戻します。

- XML ノードまたは複合的なコンテンツを戻すことはできません。結果として XML ノードを取得した場合、コンパイル時または実行時にエラーを戻します。
- デフォルトでは、常に VARCHAR2 を戻します。ノードの値が 4KB より大きい場合、ランタイム・エラーが発生します。
- XML Schema 情報が存在する場合、`extractValue()` は問合せのコンパイル時にその情報を検出すると、コンパイル時にその XML Schema 情報に基づく適切なデータ型を自動的に戻します。たとえば、パス `/PO/POID` の XML Schema 情報がそれが数値であることを示す場合、`extractValue()` は NUMBER を戻します。
- XPath がノードを識別する場合、ノードは自動的にその子テキスト・ノードからスカラー・コンテンツを取得します。この場合、ノードが持つ子テキスト・ノードが 1 つのみである必要があります。次に例を示します。

  ```
  extractValue(xmlinstance, '/PO/PNAME')
  ```

  これによって、`PNAME` の子テキスト・ノードが抽出されます。これは次の場合と同等です。

  ```
  extract(xmlinstance, '/PO/PNAME/text()').getstringval()
  ```

例 4-23 extractValue() を使用した XML フラグメントのスカラー値の抽出

次の例は、入力として 4-21 ページの extract() 関数の例と同じ引数を取ります。この例は、`extract()` のように XML フラグメントを戻すのではなく、XML フラグメントのスカラー値を戻します。

```
SELECT warehouse_name,
   extractValue(e.warehouse_spec, '/Warehouse/Docks')
   "Docks"
   FROM warehouses e
   WHERE warehouse_spec IS NOT NULL;

WAREHOUSE_NAME       Docks
-------------------- ------------
Southlake, Texas     2
San Francisco        1
New Jersey
Seattle, Washington  3
```

`ExtractValue()` は、自動的に Docks 要素の子テキスト・ノードを抽出し、その値を戻しています。これは、`extract()` を使用して次のとおり書き込むこともできます。

```
extract(e.warehouse_spec, '/Warehouse/Docks/text()').getstringval()
```

XML を問い合せる他の SQL の例

次に、XML を問い合せる SQL の例を示します。

例 4-24 extract() および existsNode() を使用した XMLType の問合せ

発注書 ID および発注書 XML 列を含む po_xml_tab 表があり、この表に次の値を挿入すると想定します。

```
INSERT INTO po_xml_tab values (100,
   xmltype('<?xml version="1.0"?>
                  <PO>
                    <PONO>221</PONO>
                    <PNAME>PO_2</PNAME>
                  </PO>'));

INSERT INTO po_xml_tab values (200,
   xmltype('<?xml version="1.0"?>
                  <PO>
                    <PONAME>PO_1</PONAME>
                  </PO>'));
```

これで、extract() を使用して、発注書番号の数値を抽出できます。

```
SELECT e.poDoc.extract('//PONO/text()').getNumberVal() as pono
   FROM po_xml_tab e
   WHERE e.podoc.existsnode('/PO/PONO')  = 1 AND poid > 1;
```

この場合の extract() は、タグの内容である発注書番号「PONO」を抽出します。existsNode() は、「PO」の子として「PONO」が存在するノードを検出します。

注意： この場合、text() 関数を使用すると、テキスト・ノードのみが戻されます。getNumberVal() 関数は、テキスト値のみを数値に変換できます。

参照： 4-7 ページの「XMLType メンバー関数」を参照してください。

例 4-25 一時 XMLType データの問合せ

次の例は、XML データを選択して、それを PL/SQL 内で問い合せる方法を示します。発注書の表から一時インスタンスを作成し、それに対して抽出を実行します。po_xml_tab に、例 4-16「文字列で XMLType() を使用した XML データの挿入」に示すデータが、次のように変更されて含まれていると想定します。

```
set serverout on
declare
```

```
   poxml XMLType;
   cust XMLType;
   val VARCHAR2(200);
begin

 -- select the adt instance
  select poDoc into poxml
     from po_xml_tab p where p.poid = 100;

  -- do some traversals and print the output
  cust := poxml.extract('//SHIPADDR');

   -- do something with the customer XML fragment
  val := cust.getStringVal();
  dbms_output.put_line(' The customer XML value is '|| val);

end;
/
```

例 4-26 extract() を使用した XML 文書からのデータの抽出および表への挿入

次の例は、発注書 XML からデータを抽出し、そのデータを SQL リレーショナル表に挿入する方法を示します。次のリレーショナル表を考えてみます。

```
CREATE TABLE cust_tab
(
  custid number primary key,
  custname varchar2(20)
);

INSERT INTO cust_tab values (1001, 'John Nike');

CREATE TABLE po_rel_tab
(
  pono number,
  pname varchar2(100),
  custid number references cust_tab,
  shipstreet varchar2(100),
  shipcity varchar2(30),
  shipzip varchar2(20)
);
```

単純な PL/SQL ブロックを作成し、`extract()` を使用して、次の形式の XML をリレーショナル表に変換できます。

```
<?xml version = '1.0'?>
 <PO>
  <PONO>2001</PONO>
```

```
  <PNAME>Po_1</PNAME>
   <CUSTOMER CUSTNAME="John Nike"/>
   <SHIPADDR>
    <STREET>323 College Drive</STREET>
    <CITY>Edison</CITY>
    <STATE>NJ</STATE>
    <ZIP>08820</ZIP>
   </SHIPADDR>
 </PO>
```

次に SQL の例を示します（前述の例に示す XML が po_xml_tab 内に存在すると想定します）。

```
INSERT INTO po_rel_tab
SELECT p.poDoc.extract('/PO/PONO/text()').getnumberval() as pono,
   p.poDoc.extract('/PO/PNAME/text()').getstringval() as pname,
   -- get the customer id corresponding to the customer name
   ( SELECT c.custid
     FROM  cust_tab c
     WHERE c.custname = p.poDoc.extract('/PO/CUSTOMER/@CUSTNAME').getstringval()
    ) as custid,
    p.poDoc.extract('/PO/SHIPADDR/STREET/text()').getstringval() as shipstreetr,
    p.poDoc.extract('//CITY/text()').getstringval() as shipcity,
    p.poDoc.extract('//ZIP/text()').getstringval() as shipzip
FROM po_xml_tab p;
```

これで、po_tab 表は次の値を持ちます。

```
PONO    PNAME   CUSTID   SHIPSTREET         SHIPCITY   SHIPZIP
-------------------------------------------------------------
2001    Po_1    1001     323 College Drive  Edison     08820
```

> **注意：** 入力 XML 文書で、PO の下に PNAME という要素がなかったため、PNAME が NULL になっています。また、前述の例は、//CITY を使用してすべての階層で city 要素を検索しています。

例 4-27 PL/SQL ブロック内での extract() を使用した XML 文書からのデータの抽出および表への挿入

次に示すとおり、PL/SQL ブロック内で等式を使用しても、同じタスクを実行できます。

```
DECLARE
  poxml XMLType;
  cname varchar2(200);
  pono number;
  pname varchar2(100);
```

```
  shipstreet varchar2(100);
  shipcity varchar2(30);
  shipzip varchar2(20);

BEGIN

 -- select the adt instance
  SELECT poDoc INTO poxml FROM po_xml_tab p;

  cname := poxml.extract('//CUSTOMER/@CUSTNAME').getstringval();

  pono := poxml.extract('/PO/PONO/text()').getnumberval();
  pname := poxml.extract('/PO/PNAME/text()').getstringval();
  shipstreet := poxml.extract('/PO/SHIPADDR/STREET/text()').getstringval();
  shipcity := poxml.extract('//CITY/text()').getstringval();
  shipzip := poxml.extract('//ZIP/text()').getstringval();

  INSERT INTO po_rel_tab
    VALUES (pono, pname,
           (SELECT custid FROM cust_tab c WHERE custname = cname),
            shipstreet, shipcity, shipzip);
END;
/
```

例 4-28 extract() および existsNode() を使用した XML データの検索

extract() 関数および existsNode() 関数を使用すると、次に示すとおり、列に対して様々な検索操作を実行できます。

```
SELECT e.poDoc.extract('/PO/PNAME/text()').getStringVal() PNAME
   FROM po_xml_tab e
   WHERE e.poDoc.existsNode('/PO/SHIPADDR') = 1 AND
      e.poDoc.extract('//PONO/text()').getNumberVal() = 300 AND
      e.poDoc.extract('//@CUSTNAME').getStringVal() like '%John%';
```

この SQL 文は、出荷先を含み、発注書番号が 300 で、顧客名「CUSTNAME」に「John」という文字列が含まれているすべての XML 文書の発注書要素 PO から、発注書名「PNAME」を抽出します。

例 4-29 extractValue() を使用した XML データの検索

extractValue() を使用すると、前述の問合せを次のとおりリライトできます。

```
SELECT extractvalue(e.poDoc, '/PO/PNAME') PNAME
   FROM po_xml_tab e
   WHERE e.poDoc.existsNode('/PO/SHIPADDR') = 1 AND
         extractvalue(e.poDoc,'//PONO') = 300 AND
         extractvalue(e.poDoc,'//@CUSTNAME') like '%John%';
```

例 4-30 extract() を使用した XMLType からのフラグメントの抽出

extract() メンバー関数は、XPath 式によって識別されるノードを抽出し、フラグメントを含む XMLType を戻します。この場合、検索結果は、ノードの集合、単一のノードまたはテキスト値のいずれかになります。XMLType に isFragment() 関数を使用すると、結果がフラグメントかどうかを確認できます。次に例を示します。

```
SELECT e.poDoc.extract('/PO/SHIPADDR/STATE').isFragment()
    FROM po_xml_tab e;
```

> **注意：** フラグメントを XMLType 列に挿入することはできません。SYS_XMLGEN() を使用すると、囲みタグを追加することによってフラグメントを整形式の文書に変換できます。10-42 ページの「SYS_XMLGEN() 関数」を参照してください。ただし、様々な XMLType 関数を使用して、フラグメントに追加の問合せを実行することができます。

/PO/SHIPADDR/STATE を抽出すると、フラグメントではない単一の整形式ノードが戻るため、前述の SQL 文は、0（ゼロ）を戻します。

一方、/PO/SHIPADDR/STATE/text() などの XPath は、整形式の XML 文書ではないため、フラグメントとみなされます。

XML インスタンスおよび表と列のデータの更新

この項では、一時 XML インスタンスおよび表に格納された XML データの更新について説明します。

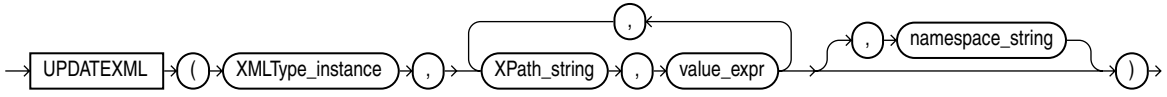
CLOB ベースの記憶域を使用する場合、今回のリリースでは、更新を行うと、文書全体が効果的に置換されます。XML 文書全体を更新するには、SQL の UPDATE 文を使用します。UPDATE の SET 句の右側には XMLType インスタンスを指定する必要があります。これは、XML インスタンスを戻す SQL 関数および XML コンストラクタか、または既存の XML インスタンスを変更およびバインドする PL/SQL DOM API for XMLType または Java DOM API for XMLType を使用して作成できます。

SQL 関数 updateXML()

updateXML() 関数は、元となる XMLType インスタンスおよび XPath と値の組を複数取ります。図 4-5 を参照してください。この関数は、指定された値を使用して更新された適切な XML ノードを含む、元の XMLType インスタンスで構成される新しい XML インスタンスを戻します。オプションの名前空間パラメータは、XPath パラメータ内の接頭辞の名前空間マッピングを指定します。

updateXML() を使用すると、要素、属性およびその他のノードを新しい値で更新および置換できます。この関数を使用して直接新しいノードを挿入したり、既存のノードを削除することはできません。かわりに、そのノードを含む親要素を新しい値で更新する必要があります。

図 4-5 updateXML() の構文



updateXML() は、メモリー内の一時 XML インスタンスのみを更新します。表に格納されたデータを更新するには、SQL の UPDATE 文を使用します。updateXML() の構文は次のとおりです。

```
UPDATEXML(xmlinstance, xpath1, value_expr1
                     [, xpath2, value_expr2]...[,namespace_string]);
```

例 4-31 UPDATE 文を使用した XMLType の更新

次の例は、UPDATE 文を使用して XMLType を更新します。この例では、発注書番号が 2001 のドキュメントのみが更新されます。

```
UPDATE po_xml_tab e
   SET  e.poDoc = XMLType(
   '<?xml version="1.0"?>
   <PO pono="2">
      <PNAME>Po_2</PNAME>
      <CUSTNAME>Nance</CUSTNAME>
      <SHIPADDR>
         <STREET>2 Avocet Drive</STREET>
         <CITY>Redwood Shores</CITY>
         <STATE>CA</STATE>
      </SHIPADDR>
   </PO>')
WHERE e.poDoc.EXTRACT('/PO/PONO/text()').getNumberVal() = 2001;
```

注意： XML Schema に基づかない XML 文書を更新すると、常に XML 文書全体が更新されます。

例 4-32 UPDATE および updateXML() を使用した XMLType の更新

新しい XML 文書を作成するのではなく、表内の XML 文書を更新するには、UPDATE 文の右側に updateXML() を使用すると、その文書を更新できます。

> **注意：** これによって、文書の一部のみでなく、文書全体も更新されます。

```
UPDATE po_xml_tab
SET poDoc =  UPDATEXML(poDoc,
   '/PO/CUSTNAME/text()', 'John');

1 row updated

SELECT e.poDoc.getstringval() AS newpo
   FROM po_xml_tab e;

NEWPO
--------------------------------------------------------------------
<?xml version="1.0"?>
<PO pono="2">
   <PNAME>Po_2</PNAME>
   <CUSTNAME>John</CUSTNAME>
   <SHIPADDR>
      <STREET>2 Avocet Drive</STREET>
      <CITY>Redwood Shores</CITY>
      <STATE>CA</STATE>
   </SHIPADDR>
</PO>
```

例 4-33 updateXML() を使用した列内の複数の要素の更新

単一の updateXML() 式内で複数の要素を更新できます。たとえば、前述の例に示す UPDATE 文と同じ UPDATE 文を使用して、発注書 po を更新できます。

```
UPDATE emp_tab e
   SET e.emp_col = UPDATEXML(e.emp_col,
      '/EMPLOYEES/EMP[EMPNAME="Joe"]/SALARY/text()',100000,
      '//EMP[EMPNAME="Jack"]/EMPNAME/text()','Jackson',
      '//EMP[EMPNO=217]',XMLTYPE.CREATEXML(
              '<EMP><EMPNO>217</EMPNO><EMPNAME>Jane</EMPNAME></EMP>'))
   WHERE EXISTSNODE(e.emp_col, '//EMP') = 1;
```

これによって、EMP 要素を持つすべての行が新しい値で更新されます。

例 4-34 updateXML() を使用した発注書の XML 文書内の顧客名の更新

次の例は、発注書の XML 文書 po 内の顧客名を更新します。

> **注意：** この例では、その文書のみを選択し、更新は一時 XMLType インスタンスに対して実行されます。元の文書は影響を受けません。

```
SELECT
   UPDATEXML(poDoc,
   '/PO/CUSTNAME/text()', 'John').getstringval() AS updatedPO
   FROM po_xml_tab;

UPDATEDPO
--------------------------------------------------------------------
<?xml version="1.0"?>
<PO pono="2">
   <PNAME>Po_2</PNAME>
   <CUSTNAME>John</CUSTNAME>
   <SHIPADDR>
      <STREET>2 Avocet Drive</STREET>
      <CITY>Redwood Shores</CITY>
      <STATE>CA</STATE>
   </SHIPADDR>
</PO>
```

例 4-35 updateXML() を使用した複数の一時 XML インスタンスの更新

updateXML() を使用して、複数の一時インスタンスを更新することもできます。たとえば、表 emp_tab の列 emp_col に格納された次の XML 文書を考えてみます。

```
<EMPLOYEES>
   <EMP>
    <EMPNO>112</EMPNO>
    <EMPNAME>Joe</EMPNAME>
    <SALARY>50000</SALARY>
  </EMP>
  <EMP>
    <EMPNO>217</EMPNO>
    <EMPNAME>Jane</EMPNAME>
   <SALARY>60000</SALARY>
   </EMP>
  <EMP>
    <EMPNO>412</EMPNO>
    <EMPNAME>Jack</EMPNAME>
    <SALARY>40000</SALARY>
  </EMP>
```

```
</EMPLOYEES>
```

Joe の給与を 100,000 に更新した新しい文書を生成し、Jack の名前を Jackson に更新し、217 の Employee 要素を変更して salary 要素を削除するとします。次のような問合せを書くことができます。

```
SELECT UPDATEXML(emp_col, '/EMPLOYEES/EMP[EMPNAME="Joe"]/SALARY/text()', 100000,
                          '//EMP[EMPNAME="Jack"]/EMPNAME/text()','Jackson',
                          '//EMP[EMPNO=217]',
            XMLTYPE.CREATEXML('<EMP><EMPNO>217</EMPNO><EMPNAME>Jane</EMPNAME>'))
   FROM emp_tab e;
```

これによって、次の更新済 XML が生成されます。

```
<EMPLOYEES>
   <EMP>
    <EMPNO>112</EMPNO>
    <EMPNAME>Joe</EMPNAME>
    <SALARY>100000</SALARY>
  </EMP>
  <EMP>
    <EMPNO>217</EMPNO>
    <EMPNAME>Jane</EMPNAME>
   </EMP>
  <EMP>
    <EMPNO>412</EMPNO>
    <EMPNAME>Jackson</EMPNAME>
    <SALARY>40000</SALARY>
  </EMP>
</EMPLOYEES>
```

updateXML() を使用した XML データのビューの作成

updateXML() を使用すると、XML データの新しいビューを作成できます。これは、特定ユーザーが SALARY などの機密データを参照できないようにする場合に有効です。

例 4-36 updateXML() を使用したビューの作成

次のようなビューを考えてみます。

```
CREATE VIEW new_emp_view
   AS SELECT
      UPDATEXML(emp_col, '/EMPLOYEES/EMP/SALARY/text()', 0) emp_view_col
   FROM emp_tab e;
```

このビューでは、ビュー new_emp_view を選択するユーザーは、すべての従業員の SALARY フィールドを参照できないことが保証されます。

updateXML() の最適化

ほとんどの場合、updateXML() では、メモリー内で入力 XML 文書全体が生成され、値が更新されます。ただし、この関数は、XML Schema に基づくオブジェクト・リレーショナル形式で格納された XMLType 表および列に対して UPDATE 文を実行するために最適化されています。そのため、列の値は直接更新されます。

リライト条件の詳細は、5-47 ページの「XML Schema に基づく構造化記憶域でのクエリー・リライト」を参照してください。すべてのリライト条件が満たされる場合、updateXML() は、オブジェクト・リレーショナル列を新しい値で直接更新するようにリライトされます。たとえば、次の UPDATE 文を考えてみます。

```
UPDATE po_xml_tab
SET poDoc =  UPDATEXML(poDoc,
   '/PO/CUSTNAME/text()', 'John');
```

リライト条件が満たされる場合、この文は custname 列を直接更新する UPDATE 文にリライトされます。

```
UPDATE po_xml_tab p
SET p.xmldata.CUSTNAME  =  'John';
```

updateXML() および NULL 値

XML 要素を NULL で更新する場合、Oracle によって要素の属性および子が削除され、要素が空になります。要素のタイプおよび名前空間プロパティは保持されます。要素を NULL 値で更新することは、要素を空に設定することと同じです。

要素のテキスト・ノードを NULL で更新する場合、Oracle によって要素のテキスト値が削除されます。要素自体は残りますが、この要素は空になります。たとえば、ノード '/empno/text()' を NULL 値で更新した場合、empno 要素のテキスト値が削除され、empno 要素が空になります。

属性を NULL に設定すると、同様に属性の値が空の文字列に設定されます。

updateXML() を使用して特定の要素や属性を追加または削除することはできません。その要素または属性を含む要素を新しい値で更新する必要があります。

注意： empno が属性を持たない単純なスカラー要素である場合、'empno' を NULL に設定すると、'empno/text()' を NULL に設定した場合と同じ結果になります。

例 4-37 updateXML() を使用した NULL での更新

次の XML 文書について考えてみます。

```
<PO>
  <pono>21</pono>
    <shipAddr gate="xxx">
       <street>333</street>
       <city>333</city>
  </shipAddr>
</PO>
```

次の句を想定します。

```
updateXML(xmlcol,'/PO/shipAddr',null)
```

この句は、次の文書を作成することと同じです。

```
<PO>
   <pono>21</pono>
   <shipAddr/>
</PO>
```

テキスト・ノードを NULL に更新することは、テキスト値のみを削除することと同じです。次に例を示します。

```
UPDATEXML(xmlcol,'/PO/shipAddr/street/text()', null)
```

次の結果が戻されます。

```
<PO>
   <pono>21</pono>
   <shipAddr>
       <street/>
       <city>333</city>
   </shipAddr>
</PO>
```

同じ XML ノードの複数回の更新

同じ XML ノードを `updateXML()` 文で複数回更新できます。たとえば、`/EMP[EMPNO=217]` と `/EMP[EMPNAME="Jane"]/EMPNO` の両方を更新できます。この場合、1 つ目の XPath はそれを含む `EMPNO` ノードも識別します。更新の順序は、左から順に、XPath 式の順序によって決定されます。各 XPath は、その前の XPath の更新結果に基づいて機能します。

XMLTransform() 関数

XMLTransform() 関数は、XMLType インスタンスおよび XSLT スタイルシートを取ります。この関数は、スタイルシートを XML 文書に適用し、変換された XML インスタンスを戻します。図 4-6 を参照してください。

図 4-6 XMLTransform() の構文

XMLTransform() の詳細は、第 6 章「XMLType データの変換および検証」を参照してください。

XML データの削除

XMLType 列を含む行での DELETE は、他のデータ型の場合と同じ方法で実行されます。

例 4-38 extract() を使用した行の削除

たとえば、発注書名が「Po_2」のすべての発注書行を削除するには、次の文を実行します。

```
DELETE FROM po_xml_tab e
    WHERE e.poDoc.extract('/PO/PNAME/text()').getStringVal()='Po_2';
```

トリガー内での XMLType の使用

トリガー内で NEW バインドおよび OLD バインドを使用して、XMLType 列値を読み込み、変更することができます。INSERT 文および UPDATE 文の場合、NEW 値を変更して、挿入する値を変更できます。

例 4-39 XMLType トリガーの作成

たとえば、発注書に納品先が記載されていない場合、トリガーを作成して、発注書を変更できます。

```
CREATE OR REPLACE TRIGGER po_trigger
  BEFORE INSERT OR UPDATE ON po_xml_tab FOR EACH ROW
declare
  pono Number;
begin
```

挿入する場合は次のとおりです。

```
    if :NEW.poDoc.existsnode('//SHIPADDR') = 0 then
```

```
      :NEW.poDoc := xmltype('<PO>INVALID_PO</PO>'); end if;
    end if;
```

更新時に、古い poDoc の発注書番号が新しい poDoc のものと異なる場合、それを無効な PO にします。

更新する場合は次のとおりです。

```
        if :OLD.poDoc.extract('//PONO/text()').getNumberVal() !=
            :NEW.poDoc.extract('//PONO/text()').getNumberVal() then

          :NEW.poDoc := xmltype('<PO>INVALID_PO</PO>');
        end if;
    end if;
end;
/
```

この例は、単なる説明用です。XMLType 値を使用して、トリガー内で有効な操作（XML 文書が準拠する必要のあるビジネス・ロジックや規則の検証、監査など）を実行できます。

XMLType 列の索引付け

XMLType を使用する場合、次の索引を作成できます。索引付けを行うと、問合せの評価が高速化されます。

XMLType 列でのファンクション索引の作成

existsNode()、または XML 文書の extract() 使用部分にファンクション索引を作成することによって、問合せを高速化できます。

例 4-40 extract() 関数でのファンクション索引の作成

たとえば、次の問合せで検索を高速化するとします。

```
SELECT * FROM po_xml_tab e
     WHERE e.poDoc.extract('//PONO/text()').getNumberVal()= 100;
```

この場合、次のとおり、extract() 関数でファンクション索引を作成できます。

```
CREATE INDEX city_index ON po_xml_tab
    (poDoc.extract('//PONO/text()').getNumberVal());
```

SQL 問合せは、行ごとに XML 文書を解析して XPath 式を評価するかわりに、ファンクション索引を使用して述語を評価します。

例 4-41 existsNode() 関数でのファンクション索引の作成

ビットマップ化されたファンクション索引を作成して、演算子の評価を高速化することもできます。existsNode() は、XPath で示されたノードが XML 文書内に存在するかどうかに応じて、数値 1 または 0（ゼロ）を戻すため最適です。

たとえば、XML 文書のいずれかのレベルに SHIPADDR という要素が含まれているかどうか検索する、次の問合せを高速化するとします。

```
SELECT * FROM po_xml_tab e
      WHERE e.poDoc.existsNode('//SHIPADDR') = 1;
```

この場合、次のとおり、existsNode() 関数でビットマップ化されたファンクション・ベース索引を作成できます。

```
CREATE BITMAP INDEX po_index ON po_xml_tab
     (poDoc.existsNode('//SHIPADDR'));
```

これによって、問合せ処理が高速化されます。

XMLType 列での Oracle Text 索引の作成

Oracle Text 索引は CLOB 列および VARCHAR 列で機能します。Oracle9*i* データベースでは、Oracle Text 索引は XMLType 列でも機能するように拡張されています。デフォルトでは、Oracle Text 索引が XMLType 列に定義されると、自動的に XML セクションが作成されます。また、XPath をサポートするように拡張されている CONTAINS 演算子も提供します。

通常、Oracle Text 索引は、他の CLOB 列または VARCHAR 列と同様に、INDEXTYPE を指定した SQL の CREATE INDEX 文を使用して作成できます。ただし、XMLType 列の Oracle Text 索引はファンクション索引として作成されます。

例 4-42 Oracle Text 索引の作成

```
CREATE INDEX po_text_index ON
      po_xml_tab(poDoc) indextype is ctxsys.context;
```

XMLType 列に、CONTAINS や SCORE などの Oracle Text 操作を実行できます。Oracle9*i* データベース リリース 1（9.0.1）では、CONTAINS 演算子は、新しい演算子 INPATH および HASPATH を使用する XPath をサポートするように拡張されています。

- INPATH は、指定のパス内に任意の文字列が存在するかどうかを確認します。
- HASPATH は、XML 文書内に任意の XPath が存在するかどうかを確認します。

例 4-43 HASPATH を使用した XML データの検索

次に例を示します。

```
SELECT * FROM po_xml_tab  w
      WHERE CONTAINS(w.poDoc,
           'haspath(/PO[./@CUSTNAME="John Nike"])') > 0;
```

不要になった QUERY_REWRITE 権限

Oracle9*i* データベース リリース 1（9.0.1）では、問合せで Oracle Text 索引を作成および使用するには、索引を作成する権限および Oracle Text 索引を作成する権限の他に、ファンクション索引を作成する次の権限を取得し、必要な設定を行う必要がありました。

- QUERY_REWRITE 権限: 自分のスキーマ内の XMLType 列に Text 索引を作成するための権限を取得している必要があります。
- GLOBAL_QUERY_REWRITE 権限: 他のスキーマや他のスキーマに存在する表の XMLType 列に Oracle Text 索引を作成する必要がある場合は、この権限を取得している必要があります。

XMLType 列に索引付けをする場合、Oracle Text 索引は、デフォルトのセクション・グループとして PATH_SECTION_GROUP を使用します。このデフォルトは、Oracle Text 索引の作成時にオーバーライドできます。

今回のリリースでは、Oracle Text 索引の作成時に、追加の QUERY_REWRITE 権限は必要ありません。

参照： 次の章またはマニュアルを参照してください。

- 第 7 章「Oracle Text を使用した XML データの検索」
- 第 10 章「データベースからの XML データの生成」
- 『Oracle Text リファレンス』
- 『Oracle Text アプリケーション開発者ガイド』

注意： XMLType 列で Oracle Text 索引や他のファンクション索引を作成する場合、QUERY_REWRITE_INTEGRITY および QUERY_REWRITE_ENABLED のセッション設定は必要ありません。

XMLType 列での XPath 索引の作成 : CTXXPATH 索引

SQL 関数 existsNode() は、CONTAINS 演算子とは異なり、Oracle Text 索引を使用してその評価を高速化することはできません。今回のリリースでは、existsNode() での XPath 検索のパフォーマンスを向上させるために、新しい索引タイプ CTXXPATH が導入されています。

CTXXPATH 索引は、Oracle Text によって提供される新しい索引タイプです。この索引は、existsNode() 処理の 1 次フィルタとして機能するように設計され、existsNode() 関数によって生成される結果のスーパーセットを生成します。その後、existsNode() 関数の実装が結果に適用され、適切な行セットが戻されます。

CTXXPATH 索引は、XPath のパス検索、ワイルド・カードおよび文字列等価述語を処理できます。

例 4-44 CTXXPATH 索引または existsNode() を使用した XPath の検索

```
CREATE INDEX po_text_index ON
      po_xml_tab(poDoc) indextype is ctxsys.ctxxpath;
```

たとえば、次の問合せを考えてみます。

```
SELECT *
   FROM po_xml_doc  w
   WHERE existsNode(w.poDoc,'/PO[@CUSTNAME="John Nike"]') = 1;
```

この問合せは、existsNode() 述語を満たすために CTXXPATH 索引を使用している可能性があります。

> **参照 :** 次の章を参照してください。
>
> - 第 7 章「Oracle Text を使用した XML データの検索」
> - 第 10 章「データベースからの XML データの生成」

CONTAINS と existsNode() / extract() の違い

CONTAINS を使用した場合の XPath サポートと、existsNode() 関数および extract() 関数での XPath サポートには、次の相違点があります。

- Oracle Text 索引は空白を無視するため、空白が重要な場合、XPath 式で正確な結果を得られない場合があります。
- Oracle Text 索引は文字列の等価性を含む特定の述語式もサポートしていますが、数値および範囲の比較はできません。

- XML 文書にテキストのないタグ名および属性名のみが含まれている場合、Oracle Text 索引は不正な結果を戻す場合があります。たとえば、次の XML 文書を考えてみます。

```
<A>
  <B>
      <C>
      </C>
  </B>
  <D>
      <E>
      </E>
   </D>
 </A>
```

  この場合、XPath 式の A/B/E は、この XML 文書と一致しますが、これは不正です。

- ファンクション索引および Oracle Text 索引は、いずれもナビゲーションをサポートします。そのため、Oracle Text 索引を 1 次フィルタとして使用して、基準と一致する可能性のあるすべての文書を効率的にフィルタし、その後に、残りの XML 文書に `existsNode()` や `extract()` 操作などの 2 次フィルタを適用することができます。

  参照： 7-38 ページの表 7-6「CONTAINS() および existsNode() を使用した XMLType データの検索」を参照してください。

5

XMLType の構造化されたマッピング

この章では、XML Schema の概要および Oracle XML DB アプリケーションでの XML Schema の使用方法について説明します。XML Schema を登録し、XML Schema に基づく XML を格納するための格納構造を作成する方法について説明します。XML データの DOM 再現性を保持する方法を含む、XML から SQL 格納型へのマッピングの詳細を示します。また、このマッピングに基づく XMLType 表および列に対する問合せを、クエリー・リライトを使用して最適化する方法についても説明します。さらに既存のオブジェクト型から XML Schema を生成するメカニズムについても説明します。

この章の内容は次のとおりです。

- XML Schema: SQL への complexType のマッピング
- Oracle XML DB の complexType の拡張および制限
- XML Schema に基づく XML 表を作成する場合の詳細なガイドライン
- XML Schema に基づく構造化記憶域でのクエリー・リライト
- XML Schema の登録時のデフォルト表の作成
- Ordered Collections in Tables（OCT）
- XML Schema 間での循環参照
- FAQ: XML DB および XML Schema に基づく問題

XML Schema の概要

XML Schema 勧告は、XML 文書のコンテンツおよび構造を XML で記述するために World Wide Web Consortium（W3C）で作成されました。XML Schema には、既存の Document Type Definition（DTD）を XML Schema に変換できるように、すべての DTD 機能が含まれています。また、XML Schema には DTD にはない追加機能があります。

参照： 付録 B「XML Schema の手引き」を参照してください。

XML Schema および Oracle XML DB

XML Schema は、XML で記述されたスキーマ定義言語です。準拠するインスタンス・ドキュメントの構造およびその他の様々なセマンティクスを定義するために使用できます。たとえば、次の XML Schema 定義 *po.xsd* は、発注書の XML 文書の構造およびその他のプロパティを定義しています。

このマニュアルでは、XML Schema とは XML Schema 定義を指します。

例 5-1 XML Schema 定義 po.xsd

次に、XML Schema 定義 `po.xsd` の例を示します。

```
<schema targetNamespace="http://www.oracle.com/PO.xsd"
xmlns:po="http://www.oracle.com/PO.xsd" xmlns="http://www.w3.org/2001/XMLSchema">
 <complexType name="PurchaseOrderType">
  <sequence>
   <element name="PONum" type="decimal"/>
   <element name="Company">
    <simpleType>
     <restriction base="string">
      <maxLength value="100"/>
     </restriction>
    </simpleType>
   </element>
   <element name="Item" maxOccurs="1000">
    <complexType>
     <sequence>
      <element name="Part">
       <simpleType>
        <restriction base="string">
         <maxLength value="1000"/>
        </restriction>
       </simpleType>
      </element>
      <element name="Price" type="float"/>
     </sequence>
    </complexType>
```

```
   </element>
  </sequence>
 </complexType>
 <element name="PurchaseOrder" type="po:PurchaseOrderType"/>
</schema>
```

例 5-2 XML Schema po.xsd に準拠する XML 文書 po.xml

次に、XML Schema po.xsd に準拠する XML 文書の例を示します。

```
<PurchaseOrder xmlns="http://www.oracle.com/PO.xsd"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://www.oracle.com/PO.xsd  http://www.oracle.com/PO.xsd">
     <PONum>1001</PONum>
     <Company>Oracle Corp</Company>
     <Item>
       <Part>9i Doc Set</Part>
       <Price>2550</Price>
     </Item>
</PurchaseOrder>
```

注意：

ここに示す URL「http://www.oracle.com/PO.xsd」は、データベース内で登録された XML Schema を一意に識別する名前であり、XML Schema 文書が存在する実際の URL である必要はありません。また、XML Schema のターゲットの名前空間は、XML Schema の場所を示す URL とは異なる URL です。この URL では、要素および型が宣言される抽象的な名前空間が指定されます。

XML Schema では、オプションで、ターゲットの名前空間の URL を指定できます。この属性が省略されている場合、XML Schema にターゲットの名前空間が定義されません。通常、ターゲットの名前空間は、XML Schema URL と同じであることに注意してください。

XML インスタンス・ドキュメントでは、ルート要素の名前空間（XML Schema のターゲットの名前空間と同じ）およびこのルート要素を定義する XML Schema の場所（URL）の両方が指定されている必要があります。場所は、xsi:schemaLocation 属性で指定されます。XML Schema にターゲットの名前空間が定義されていない場合、xsi:noNamespaceSchemaLocation 属性を使用して XML Schema URL を指定します。

Oracle XML DB および XML Schema の使用

Oracle XML DB では、注釈付き XML Schema（いくつかの Oracle XML DB 定義の属性が追加された標準的な XML Schema 定義）がメタデータとして使用されます。これらの属性は異なる名前空間に存在し、データベースへのインスタンス・ドキュメントのマッピングを制御します。これらの属性は、XML Schema の名前空間とは異なる名前空間に存在するため、注釈付き XML Schema は、正当な XML Schema 文書になります。

> **参照：** XML Schema の名前空間の構成については、次の URL を参照してください。
>
> `http://www.w3.org/2001/XMLSchema`

Oracle XML DB を使用する場合、まず XML Schema を登録する必要があります。これによって、`XMLType` 表、列およびビューの作成に、XML Schema URL を使用できます。

Oracle XML DB では、次のタスクに対して XML Schema のサポートが提供されます。

- W3C 準拠の XML Schema の登録
- 登録された XML Schema 定義に対する XML 文書の検証
- ローカルおよびグローバル XML Schema の登録
- オブジェクト型からの XML Schema の生成
- 他のユーザーが所有する XML Schema の参照
- 同じ名前を持つローカル XML Schema が存在する場合に行うグローバル XML Schema の明示的な参照
- XML Schema の登録時に XML Schema から行う構造化されたデータベース・マッピングの生成

  これには、SQL オブジェクト型、コレクション型およびデフォルト表の生成、および XML Schema 属性を使用するマッピング情報の取得が含まれます。
- 複数の正当なマッピングが存在する場合に行う特定の SQL 型マッピングの指定
- 登録された XML Schema に基づく `XMLType` 表、ビューおよび列の作成
- XML Schema に基づく `XMLType` 表に対する操作（DML）および問合せの実行
- FTP、HTTP/WebDAV プロトコルおよびその他の言語を使用して、スキーマベースの XML インスタンスが Oracle XML DB Repository に挿入される場合に行う、デフォルト表へのデータの自動挿入

> **参照：** 第 26 章「Oracle XML DB Basic Demo」を参照してください。

XML Schema のメリット

第 4 章「XMLType の使用」で説明するとおり、XMLType データ型によって、データベースの列および表に XML を簡単に格納できるようになります。XML Schema を使用すると、データベースへの XMLType 列および表の格納がさらに簡単になります。また、領域を節約し、パフォーマンスを向上するオプションの他に、XML データに対するより多くの格納およびアクセス・オプションが提供されます。

たとえば、XML Schema を使用して、格納または処理される XML 文書で使用可能な要素および属性、要素のネストの種類およびデータ型を宣言できます。

XML Schema による XML から SQL への柔軟なマッピング設定

Oracle XML DB で XML Schema を使用すると、XML の格納マッピングを柔軟に設定できます。たとえば、次のとおり設定できます。

- 高度に構造化されたデータ（主に XML）の場合、XML 文書の各要素を表の 1 つの列として格納できます。
- 非構造化データ（主に非 XML データ）の場合、キャラクタ・ラージ・オブジェクト（CLOB）で格納できます。

選択する格納方法は、データの使用方法、問合せの実行性、データの問合せおよび更新に対する要件によって異なります。XML Schema を使用すると、高度に構造化されたデータまたは非構造化データをより柔軟に格納できます。

XML Schema による XML インスタンスの検証

Oracle XML DB で XML Schema を使用するもう 1 つのメリットは、最適なパフォーマンスを実現するための Oracle XML リポジトリの要件に対して、XML Schema による XML インスタンスの検証を実施できることです。たとえば、XML Schema を使用して、受け取ったすべての XML 文書が、使用可能な構造、型、項目の出現回数、項目の長さなど、XML Schema で宣言された定義に準拠することを確認できます。

また、Oracle XML DB に XML Schema を登録することによって、FTP や HTTP などのプロトコルを使用して XML インスタンスを挿入および格納する場合に、XML Schema の情報によって XML インスタンスを効率的に挿入できます。

事前情報のない XML インスタンスを処理する必要がある場合、XML Schema を使用すると、最適な格納、再現性およびアクセスを予測できます。

Oracle XML DB における DTD のサポート

オブジェクト・リレーショナル記憶域への構造化マッピングを提供する XML Schema のサポートに加えて、Oracle XML DB では、XML インスタンス・ドキュメントでの DTD の指定もサポートされます。DTD はマッピングの導出には使用されませんが、XML プロセッサから DTD にアクセスし、解析することはできます。

インライン DTD の定義

XML インスタンス・ドキュメントにインライン DTD の定義が含まれる場合、その定義はドキュメントの解析時に使用されます。すべての DTD の検証およびエンティティ宣言の処理は、この時点で行われます。ただし、解析後、実体参照は実際の値で置換され、元の実体参照は失われます。

外部 DTD の定義

Oracle XML DB では、リポジトリに格納された外部 DTD の定義もサポートされます。「/public/flights.dtd」などの外部 DTD の定義を含む XML 文書を処理する必要があるアプリケーションでは、最初に、DTD ドキュメントが「/public/flights.xsd」というパスで Oracle XML DB に格納されていることを確認する必要があります。

DBMS_XMLSCHEMA の概要

Oracle XML DB の XML Schema 機能は、PL/SQL パッケージ DBMS_XMLSCHEMA を介して使用可能です。これは、Oracle XML DB アプリケーションによって XML Schema 定義の登録を処理するために使用されるサーバー側のコンポーネントです。

> **参照：**『Oracle9*i* XML API リファレンス - XDK および Oracle XML DB』を参照してください。

DBMS_XMLSCHEMA には、主に次の 2 つのファンクションがあります。

- **registerSchema()**: XML Schema を登録します。次の項目を指定する必要があります。
  - XML Schema ソース（文字列、LOB、XMLType、URIType など、様々な形式の場合がある）
  - XML Schema URL または XML Schema 名
- **deleteSchema()**: 登録済の XML Schema を削除します。削除する XML Schema は、XML Schema URL または XML Schema 名で指定します。

Oracle XML DB を使用する前の XML Schema の登録

Oracle XML DB のコンテキストで XML Schema を使用または参照するには、XML Schema を登録する必要があります。XML Schema は、DBMS_XMLSCHEMA.registerSchema() を使用して、次の項目を指定することによって登録します。

- VARCHAR、CLOB、XMLType または URIType の XML Schema のソース・ドキュメント。
- XML Schema URL。これは、準拠する XML Schema の場所を指定するために XML インスタンス・ドキュメント内で使用される XML Schema の名前です。

登録が完了すると、次の作業が可能になります。

- この XML Schema に準拠する XML 文書、および XML 文書内で XML Schema URL を使用してこの XML Schema を参照する XML 文書を、Oracle XML DB によって処理できます。
- この XML Schema によって定義されたルート XML 要素に対して表および列を作成し、準拠する XML 文書を格納できます。

DBMS_XMLSCHEMA を使用した XML Schema の登録

DBMS_XMLSCHEMA を使用して、XML Schema を登録します。これには、XML Schema 文書およびその URL（XML Schema の場所ともいう）を指定する必要があります。

例 5-3 DBMS_XMLSCHEMA を使用した、complexType を宣言する XML Schema の登録

次の XML Schema について考えてみます。この XML Schema では、PurchaseOrderType という complexType およびこの型の PurchaseOrder 要素が宣言されます。このスキーマは、PL/SQL 変数 doc で格納されます。XML Schema は、http://www.oracle.com/PO.xsd という URL に登録されます。

```
declare
        doc varchar2(1000) := '<schema
targetNamespace="http://www.oracle.com/PO.xsd"
xmlns:po="http://www.oracle.com/PO.xsd" xmlns="http://www.w3.org/2001/XMLSchema">
 <complexType name="PurchaseOrderType">
  <sequence>
   <element name="PONum" type="decimal"/>
   <element name="Company">
    <simpleType>
     <restriction base="string">
      <maxLength value="100"/>
     </restriction>
    </simpleType>
   </element>
   <element name="Item" maxOccurs="1000">
    <complexType>
     <sequence>
      <element name="Part">
       <simpleType>
        <restriction base="string">
         <maxLength value="1000"/>
        </restriction>
       </simpleType>
      </element>
      <element name="Price" type="float"/>
     </sequence>
    </complexType>
```

```
   </element>
  </sequence>
 </complexType>
 <element name="PurchaseOrder" type="po:PurchaseOrderType"/>
</schema>';
begin
        dbms_xmlschema.registerSchema('http://www.oracle.com/PO.xsd', doc);
end;
```

登録されたスキーマを使用して、XML Schema に基づく表または XML Schema に基づく列を作成できます。たとえば、次の文では、XML Schema に基づく列を持つ表が作成されます。

```
create table po_tab(
 id number,
 po sys.XMLType
)
  xmltype column po
    XMLSCHEMA "http://www.oracle.com/PO.xsd"
    element "PurchaseOrder";
```

次に、前述の XML Schema に準拠する XMLType インスタンスが前述の表に挿入される場合の例を示します。schemaLocation 属性によって、スキーマの URL が指定されます。

```
insert into po_tab values (1,
  xmltype('<po:PurchaseOrder xmlns:po="http://www.oracle.com/PO.xsd"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.oracle.com/PO.xsd
  http://www.oracle.com/PO.xsd">
     <PONum>1001</PONum>
       <Company>Oracle Corp</Company>
     <Item>
      <Part>9i Doc Set</Part>
       <Price>2550</Price>
     </Item>
     <Item>
       <Part>8i Doc Set</Part>
       <Price>350</Price>
     </Item>
 </po:PurchaseOrder>'));
```

> **参照：**『Oracle9*i* XML API リファレンス - XDK および Oracle XML DB』を参照してください。

ローカル XML Schema およびグローバル XML Schema

XML Schema は、ローカルまたはグローバルとして登録できます。

- **ローカル XML Schema:** ローカル・スキーマとして登録された XML Schema は、デフォルトでは、所有者のみが参照できます。
- **グローバル XML Schema:** グローバル・スキーマとして登録された XML Schema は、デフォルトでは、すべてのデータベース・ユーザーが参照および使用できます。

XML Schema の登録時、DBMS_XMLSCHEMA によって、XML Schema に対応する Oracle XML DB リソースが Oracle XML DB Repository に追加されます。次の規則に従って、XML Schema URL によって Oracle XML DB Repository のリソースのパス名が決定されます。

ローカル XML Schema

Oracle XML DB では、ローカル XML Schema のリソースは、/sys/schemas/<username> ディレクトリに作成されます。残りのパス名は、スキーマの URL から導出されます。

例 5-4 ローカル XML Schema

スキーマの URL が設定されたローカル XML Schema の例を次に示します。

```
http://www.myco.com/PO.xsd
```

このスキーマは、SCOTT によって登録され、次のパス名が設定されています。

```
/sys/schemas/SCOTT/www.myco.com/PO.xsd.
```

データベース・ユーザーが XML Schema をローカル XML Schema として登録するには、このパス名でリソースを作成するための適切な権限（ACL）が必要です。

> **参照：** 第 18 章「Oracle XML DB リソースのセキュリティ」を参照してください。

デフォルトでは、XML Schema が Oracle XML DB に登録されると、登録したユーザーがこの XML Schema の所有者になります。XML Schema 文書に対する参照は、Oracle XML DB Repository の次のディレクトリに格納されます。

```
/sys/schemas/<username>/....
```

たとえば、SCOTT によって前述の XML Schema が登録された場合、このスキーマは次のファイルにマップされます。

```
/sys/schemas/SCOTT/www.oracle.com/PO.xsd
```

この XML Schema は、ローカルとして参照されます。通常、これらのスキーマは、所有者であるユーザーのみが使用できます。

注意：通常、XMLType 表、列またはビューの定義、文書の検証などに XML Schema を使用できるのは、この XML Schema の所有者のみです。ただし、Oracle では、完全修飾された XML Schema URL をサポートしています。この URL は、次のとおり指定できます。

```
http://xmlns.oracle.com/xdb/schemas/SCOTT/www.oracle.com/PO.xsd
```

権限のあるユーザーは、この拡張 URL を使用して、他のユーザーが所有する XML Schema を指定できます。

グローバル XML Schema

ローカル・スキーマとは異なり、権限のあるユーザーは、DBMS_XMLSCHEMA 登録ファンクションで引数を指定することによって、XML Schema をグローバル XML Schema として登録できます。

グローバル・スキーマは、すべてのユーザーが参照できます。このスキーマは、Oracle XML DB Repository の /sys/schemas/PUBLIC/ ディレクトリに格納されます。

注意： このディレクトリへのアクセスはアクセス制御リスト（ACL）によって制御され、デフォルトでは、DBA によってのみ書込み可能です。グローバル・スキーマを登録するには、このディレクトリに対する書込み権限が必要です。

デフォルトの保護付き ACL によって保護されていると想定して、XDBAdmin ロールでは、このディレクトリへの書込み権限も提供されます。

権限および XDBAdmin ロールの詳細は、第 18 章「Oracle XML DB リソースのセキュリティ」を参照してください。

既存のグローバル・スキーマと同じ URL でローカル・スキーマを登録できます。ローカル・スキーマは、常に同じ名前（URL）を持つグローバル・スキーマを非表示にします。

例 5-5 グローバル XML Schema

たとえば、SCOTT によって次の URL でグローバル・スキーマが登録されます。

```
www.myco.com/PO.xsd
```

このスキーマは、Oracle XML DB Repository の次のファイルにマップされます。

```
/sys/schemas/PUBLIC/www.myco.com/PO.xsd
```

データベース・ユーザーが XML Schema をグローバルとして登録するには、このリソースを作成するための適切な権限（ACL）が必要です。

XML Schema の登録 : Oracle XML DB による格納およびアクセス・インフラストラクチャの設定

XML Schema の登録の一部として、Oracle XML DB では、この他にも XML Schema に準拠する XML インスタンスの格納、アクセスおよび操作を簡単にするいくつかの手順が実行されます。次の手順が含まれます。

- **型の作成 :** XML Schema の登録時、Oracle によって、この XML Schema に準拠する XML 文書の構造化記憶域を使用可能にする、適切な SQL オブジェクト型が作成されます。Oracle XML DB によって定義された属性を XML Schema 文書に使用して、これらのオブジェクト型の生成方法を制御できます。
- **デフォルト表の作成 :** XML Schema の登録の一部として、Oracle XML DB によって、すべてのルート要素に対するデフォルトの `XMLType` 表が生成されます。表の作成時に使用される列および表のレベルに対する制約を指定することもできます。

  参照 : 次の項または章を参照してください。

  - 5-23 ページの「SQLName、SQLType 属性での SQL オブジェクト型名の指定」
  - 第 3 章「Oracle XML DB の使用」

DBMS_XMLSCHEMA を使用した XML Schema の削除

`DBMS_XMLSCHEMA.deleteSchema` プロシージャを使用することによって、登録された XML Schema を削除できます。XML Schema の削除を試行すると、`DBMS_XMLSCHEMA` によって次のことが確認されます。

- 現行のユーザーが、Oracle XML DB Repository 内の XML Schema に対応するリソースを削除するための適切な権限（ACL）を取得していることが確認されます。XML Schema のリソースに適切な ACL を設定することによって、各ユーザーが削除できる XML Schema を制御できます。
- 依存性が確認されます。依存性が存在する場合、エラーが発生し、削除が失敗します。これを、XML Schema 削除の **RESTRICT** モードといいます。

FORCE モード

FORCE モード・オプションは、XML Schema の削除時に指定します。FORCE モード・オプションを指定すると、依存性を確認しなかった場合でも、XML Schema の削除が実行されます。このモードでは、XML Schema の削除ですべての依存性が無効であると判断されます。

CASCADE モード

CASCADE モード・オプションでは、XML Schema を登録するための前のコールの一部として生成されたすべての型およびデフォルト表が削除されます。

> **参照：**『Oracle9*i* XML API リファレンス - XDK および Oracle XML DB』の「DBMS_XMLSCHEMA」を参照してください。

例 5-6 DBMS_XMLSCHEMA を使用した XML Schema の削除

次の例では、XML Schema PO.xsd が削除されます。まず、依存表 po_tab が削除されます。次に、DBMS_XMLSCHEMA.DELETESCHEMA で FORCE および CASCADE モードを使用して、スキーマが削除されます。

```
drop table po_tab;

EXEC dbms_xmlschema.deleteSchema('http://www.oracle.com/PO.xsd',
            dbms_xmlschema.DELETE_CASCADE_FORCE);
```

登録された XML Schema を使用する場合のガイドライン

次の項では、Oracle XML DB に XML Schema を登録する場合のガイドラインについて説明します。

登録された XML Schema に依存するオブジェクト

次のオブジェクトは、登録された XML Schema に依存します。

- XML Schema のいくつかの要素に準拠する XMLType 列が含まれる表またはビュー。
- このスキーマが定義の一部として含まれるか、またはインポートされる XML Schema。
- XML Schema の名前を（DBMS_XMLGEN 演算子内などで）参照するカーソル。これらは、一時オブジェクトであることに注意してください。

XMLType 表、ビューまたは列の作成

XML Schema の登録後、次の項目を参照することによって、XML Schema に基づく XMLType 表、ビューおよび列を作成できます。

- 登録された XML Schema の XML Schema URL
- ルート要素の名前

例 5-7 XMLType 表の登録後の作成

たとえば、次のとおり XML Schema に基づく XMLType 表を作成できます。

```
CREATE TABLE po_tab OF XMLTYPE
      XMLSCHEMA "http://www.oracle.com/PO.xsd" ELEMENT "PurchaseOrder";
```

次の文によって、XML Schema 準拠のデータが挿入されます。

```
insert into po_tab values (
  xmltype('<PurchaseOrder xmlns="http://www.oracle.com/PO.xsd"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://www.oracle.com/PO.xsd  http://www.oracle.com/PO.xsd">
     <PONum>1001</PONum>
     <Company>Oracle Corp</Company>
     <Item>
       <Part>9i Doc Set</Part>
       <Price>2550</Price>
     </Item>
     <Item>
       <Part>8i Doc Set</Part>
       <Price>350</Price>
     </Item>
</PurchaseOrder>'));
```

XML Schema に対する XML インスタンスの検証 : schemaValidate()

いずれかの検証メソッドを使用することによって、登録された XML Schema に対して XMLType インスタンスを検証できます。

> **参照：** 第 6 章「XMLType データの変換および検証」を参照してください。

例 5-8 schemaValidate() を使用した XML の検証

次の PL/SQL の例では、XML Schema `PO.xsd` に対して XML インスタンスが検証されます。

```
declare
  xmldoc xmltype;
  begin

  -- populate xmldoc (by fetching from table)
  select value(p) into xmldoc from po_tab p;

  --   validate against XML schema
  xmldoc.schemavalidate();

  if xmldoc.isschemavalidated() = 1 then
                     dbms_output.put_line('Data is valid');
  else
                     dbms_output.put_line('Data is invalid');
```

```
  end if;
end;
```

完全修飾された XML Schema URL

デフォルトでは、XML Schema URL の名前は、常に現行のユーザーの範囲内で参照されます。そのため、データベース・ユーザーによって XML Schema URL が指定された場合、まず現行のユーザーが所有するローカル XML Schema の名前として解決されます。

- 対応するローカル XML Schema が存在しない場合、グローバル XML Schema の名前として解決されます。
- 対応するグローバル XML Schema が存在しない場合、Oracle XML DB でエラーが発生します。

ユーザーが参照できない XML Schema

これらの規則は、デフォルトでは、ユーザーが次の XML Schema を参照できないことを意味します。

- 別のデータベース・ユーザーが所有する XML Schema
- ローカル XML Schema と同じ名前を持つグローバル XML Schema

完全修飾された XML Schema URL による XML Schema URL の明示的な参照の許可

前述の場合に XML Schema の明示的な参照を可能にするため、Oracle XML DB では、完全修飾された XML Schema URL という概念をサポートしています。この構成では、XML Schema URL の一部として、XML Schema を所有するデータベース・ユーザーの名前も指定します。ただし、次のとおり XML Schema URL が Oracle XML DB の名前空間に属している場合は除きます。

```
http://xmlns.oracle.com/xdb/schemas/<database-user-name>/<schemaURL-minus-protocol>
```

例 5-9 完全修飾された XML Schema URL の使用

たとえば、次の URL が設定されたグローバル XML Schema について考えてみます。

```
http://www.example.com/po.xsd
```

データベース・ユーザー SCOTT が同じ名前のローカル XML Schema を所有していると想定します。

```
http://www.example.com/po.xsd
```

ユーザー JOE は、次のとおり SCOTT が所有するローカル XML Schema を参照できます。

```
http://xmlns.oracle.com/xdb/schemas/SCOTT/www.example.com/po.xsd
```

同様に、グローバル XML Schema の完全修飾された URL は次のとおりです。

```
http://xmlns.oracle.com/xdb/schemas/PUBLIC/www.example.com/po.xsd
```

XML Schema 登録のトランザクション動作

XML Schema の登録はトランザクション型ではなく、他の SQL の DDL 操作と同様に、次のとおり自動コミットされます。

- 登録が正常に終了した場合、操作は自動コミットされます。
- 登録が失敗した場合、データベースは登録の開始前の状態にロールバックされます。

XML Schema の登録では、オブジェクト型および表が作成される可能性があるため、エラー・リカバリでは作成されたすべての型および表が削除されます。そのため、XML Schema の登録の全体がアトミックであることが保証されます。つまり、登録が正常に終了するか、またはデータベースが登録の開始前の状態にリストアされます。

DBMS_XMLSCHEMA.generateSchema() を使用した XML Schema の生成

XML Schema は、デフォルトのマッピングを使用して、オブジェクト・リレーショナル型から自動で生成できます。`DBMS_XMLSCHEMA` パッケージの `generateSchema()` ファンクションおよび `generateSchemas()` ファンクションは、オブジェクト型の名前が含まれる文字列および Oracle XML DB の XML Schema が含まれる文字列を取ります。

- `generateSchema()` は、XML Schema を含む `XMLType` を戻します。オプションで、任意のオブジェクト型によって参照されるすべての型に対して、または最上位の型のみに制限して XML Schema を生成できます。
- `generateSchemas()` は、それぞれが異なる名前空間に対応する XML Schema の `XMLSequenceType` を戻すことを除き、同様に動作します。また、オプションで追加の引数を取り、優先される XML Schema の場所のルート URL を指定します。

  ```
  http://xmlns.oracle.com/xdb/schemas/<schema>.xsd
  ```

  オプションで、Oracle XML DB に XML Schema を登録するために使用できる注釈付き XML Schema を生成することもできます。

例 5-10 generateSchema() を使用した XML Schema の生成

たとえば、次のオブジェクト型が設定されているとします。

```
connect t1/t1
CREATE TYPE employee_t AS OBJECT
(
   empno NUMBER(10),
```

```
   ename VARCHAR2(200),
   salary NUMBER(10,2)
);
```

次のとおり、この型に対するスキーマを生成できます。

```
select  dbms_xmlschema.generateschema('T1', 'EMPLOYEE_T') from dual;
```

これによって、EMPLOYEE_T 型に対応するスキーマが戻されます。このスキーマでは、EMPLOYEE_T という名前の要素および EMPLOYEE_TType という complexType が宣言されます。このスキーマには、http://xmlns.oracle.com/xdb に存在する他の注釈が含まれます。

```
DBMS_XMLSCHEMA.GENERATESCHEMA('T1','EMPLOYEE_T')
--------------------------------------------------------------------------------
<xsd:schema targetNamespace="http://ns.oracle.com/xdb/T1" xmlns="http://ns.oracl
e.com/xdb/T1" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xdb="http://xml
ns.oracle.com/xdb" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:sch
emaLocation="http://xmlns.oracle.com/xdb http://xmlns.oracle.com/xdb/XDBSchema.x
sd">
  <xsd:element name="EMPLOYEE_T" type="EMPLOYEE_TType" xdb:SQLType="EMPLOYEE_T"
xdb:SQLSchema="T1"/>
  <xsd:complexType name="EMPLOYEE_TType">
    <xsd:sequence>
      <xsd:element name="EMPNO" type="xsd:double" xdb:SQLName="EMPNO" xdb:SQLTyp
e="NUMBER"/>
      <xsd:element name="ENAME" type="xsd:string" xdb:SQLName="ENAME" xdb:SQLTyp
e="VARCHAR2"/>
      <xsd:element name="SALARY" type="xsd:double" xdb:SQLName="SALARY" xdb:SQLT
ype="NUMBER"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

XMLType の XML Schema 関連メソッド

表 5-1 に、XMLType API の XML Schema 関連メソッドを示します。

表 5-1 XMLType API の XML Schema 関連メソッド

XMLType API のメソッド	説明
isSchemaBased()	XMLType インスタンスが XML Schema に基づく場合、TRUE が戻されます。それ以外の場合は、FALSE が戻されます。
getSchemaURL() getRootElement() getNamespace()	XML Schema に基づく XMLType インスタンスに対する XML Schema URL、ルート要素の名前および名前空間が戻されます。
schemaValidate() isSchemaValid() isSchemaValidated() setSchemaValidated()	validation メソッドを使用して、登録された XML Schema に対して XMLType インスタンスを検証できます。第 6 章「XMLType データの変換および検証」を参照してください。

XML Schema の管理および格納

XML Schema 文書自体は、XMLType インスタンスとして Oracle XML DB に格納されます。XML Schema 関連の XMLType 型および表は、Oracle XML DB のインストール・スクリプト catxdbs.sql の一部として作成されます。

ルート XML Schema XDBSchema.xsd

XML Schema に対する XML Schema を、ルート XML Schema XDBSchema.xsd といいます。XDBSchema.xsd は、Oracle XML DB によって登録可能な、すべての妥当な XML Schema 文書を定義します。Oracle XML DB Repository を介して、次の場所に存在する XDBSchema.xsd にアクセスできます。

```
/sys/schemas/PUBLIC/xmlns.oracle.com/xdb/XDBSchema.xsd
```

参照： 次の章または付録を参照してください。

- 第 21 章「Oracle Enterprise Manager を使用した Oracle XML DB の管理」
- 付録 A「Oracle XML DB のインストールおよび構成」

XML Schema に基づく XMLType 構造の格納方法

XML Schema に基づく XMLType 構造は、次のいずれかの方法で格納されます。

- **内部のオブジェクト型の列 :** デフォルトの格納メカニズムです。
  - オプションで、XML Schema の登録処理時に SQL オブジェクト型を作成できます。5-20 ページの「XML Schema に基づく XMLType 表および列の作成」を参照してください。
  - 5-23 ページの「SQLName、SQLType 属性での SQL オブジェクト型名の指定」を参照してください。
- **内部の単一 LOB 列 :** 次のように、CREATE TABLE 文の STORE AS 句で指定します。

```
CREATE TABLE po_tab OF xmltype
    STORE AS CLOB
    ELEMENT "http://www.oracle.com/PO.xsd#PurchaseOrder";
```

XML データを格納するための設計基準は、第 2 章「Oracle XML DB を使用する前に」および第 3 章「Oracle XML DB の使用」を参照してください。

格納メカニズムの指定

STORE AS 句を使用するのではなく、特定の XML Schema に基づくマッピングに従って表および列が格納されるように指定することもできます。マッピングに使用される XML Schema の URL を指定できます。

CLOB を使用すると、XML Schema に基づかない XML データを表に格納できます。ただし、索引付け、クエリー・リライトなどの機能は使用できません。

DOM 再現性

ドキュメント・オブジェクト・モデル（DOM）の再現性とは、DOM 検索のために、取得された XML 文書を元の XML 文書と同じ構造に保持するという概念です。DOM 再現性は、Oracle XML DB に格納された XML 文書の精度および整合性を保証するために必要です。

> **参照 :** 5-34 ページの「表外に格納するための SQLInLine 属性の FALSE への設定」を参照してください。

Oracle XML DB による XML Schema の DOM 再現性の保証方法

XML Schema で宣言されたすべての要素および属性は、対応する SQL オブジェクト型の個別の属性にマップされます。ただし、XML インスタンス・ドキュメントの次の情報は、これらの要素または属性によって直接表現されません。

- コメント
- 名前空間宣言

- 接頭辞情報

このデータの整合性および精度を保証するため、データベースに格納された XML 文書の再生成時などに、Oracle XML DB によって DOM 再現性というデータ整合性メカニズムが使用されます。

DOM 再現性は、特に DOM 検索のために、戻された XML 文書が元の XML 文書と同じ状態に保持されている程度を示します。

DOM 再現性および SYS_XDBPD$

DOM 再現性が保持されること、および DOM 検索のために戻された XML 文書が元の XML 文書と同じであることを保証するため、作成された各オブジェクト型に Oracle XML DB によってシステム・バイナリ属性 SYS_XDBPD$ が追加されます。

他の属性に格納できないすべての情報はこの位置指定記述子属性によって格納され、Oracle XML DB に格納されたすべての XML 文書の DOM 再現性が保証されます。この属性によって格納される情報には、注文情報、コメント、処理命令、名前空間の接頭辞などが含まれます。これは、位置指定記述子（PD）列にマップされます。

注意： PD 属性は、Oracle 内部で使用する属性です。この列に直接アクセスしたり、操作しないでください。

SYS_XDBPD$ の抑制方法

DOM 再現性が必要ない場合、maintainDOM=FALSE 属性を設定することによって、XML Schema 定義で SYS_XDBPD$ を抑制できます。

注意： このマニュアルの多くの例では、簡略化のため SYS_XDBPD$ 属性を省略しています。ただし、この属性は、XML Schema の登録処理によって生成されたすべての SQL オブジェクト型に、位置指定記述子（PD）列として常に存在します。

通常、PD 属性を抑制しないことをお薦めします。PD 列が存在しない場合、コメント、処理命令などの情報が失われる可能性があるためです。

XML Schema に基づく XMLType 表および列の作成

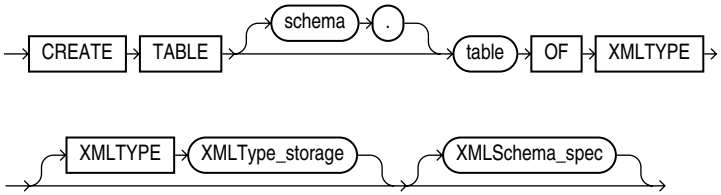
Oracle XML DB では、次の項目を参照することによって、XML Schema に基づく XMLType 表および列が作成されます。

- 登録された XML Schema の XML Schema URL
- ルート要素の名前

図 5-1 に、XMLType 表を作成するための構文を示します。

```
CREATE TABLE [schema.] table OF XMLTYPE
  [XMLTYPE XMLType_storage] [XMLSchema_spec];
```

図 5-1 XMLType 表の作成



次の例に示す XPointer 表記法のサブセットを使用して、XML Schema の場所および要素名を含む 1 つの URL を指定することもできます。

例 5-11 XML Schema に基づく XMLType 表の作成

この例では、任意の URL の XML Schema を使用して、XMLType 表 po_tab を作成します。

```
CREATE TABLE po_tab OF XMLTYPE
    XMLSCHEMA "http://www.oracle.com/PO.xsd" ELEMENT "PurchaseOrder";
```

次のように定義しても、同等の結果が戻されます。

```
CREATE TABLE po_tab OF XMLTYPE
    ELEMENT "http://www.oracle.com/PO.xsd#PurchaseOrder";
```

SQL オブジェクト・リレーショナル型による XML Schema に基づく XMLType 表の格納

XML Schema の登録時、Oracle XML DB によって、この XML Schema に基づく XML 文書の構造化記憶域を使用可能にする適切な SQL オブジェクト型が作成されます。デフォルトでは、すべての SQL オブジェクト型が、現在登録されている XML Schema に基づいて作成されます。

例 5-12 XMLType 表を格納するための SQL オブジェクト型の作成

たとえば、Oracle XML DB に PO.xsd が登録されると、次の SQL 型が作成されます。

注意： 型の名前は生成された名前であり、Itemxxx_t、Itemxxx_COLL および PurchaseOrderTypexxx_T（xxx は 3 桁の整数）に一致しない場合もあります。

```
 CREATE TYPE "Itemxxx_T" as object
(
    part varchar2(1000),
    price number
);

CREATE TYPE "Itemxxx_COLL" AS varray(1000) OF "Item_T";
CREATE TYPE "PurchaseOrderTypexxx_T" AS OBJECT
(
 ponum number,
 company varchar2(100),
 item Item_varray_COLL
);
```

注意： 前述の例のオブジェクト型および属性の名前は、システム生成できます。

- XML Schema にすでに値を持った SQLName、SQLType または SQLColType 属性が含まれている場合（詳細は、「SQLName、SQLType 属性での SQL オブジェクト型名の指定」を参照）、この名前はオブジェクト属性の名前として使用されます。
- XML Schema に SQLName 属性が含まれていない場合、長さまたは競合の問題のために使用できない場合を除き、名前は XML の名前から導出されます。

SQLSchema 属性が使用されている場合、Oracle XML DB によって、指定されたデータベース・スキーマを使用してオブジェクト型の作成が試行されます。現行のユーザーがこの操作を実行するには、必要な権限を取得している必要があります。

SQLName、SQLType 属性での SQL オブジェクト型名の指定

生成された SQL オブジェクトに特定の名前を指定するには、XML Schema を登録する前に、XML Schema 定義に SQLName および SQLType 属性を含めます。

- SQLName および SQLType の値を指定すると、Oracle XML DB は、これらの名前を使用して SQL オブジェクト型を作成します。
- これらの属性を指定しない場合、Oracle XML DB は、システム生成の名前を使用します。

> **注意：** これらのすべての属性に値を指定する必要はありません。Oracle XML DB によって、XML Schema の登録処理時に適切な値が指定されます。ただし、少なくとも最上位の SQL 型の名前を指定し、後で参照できるようにすることをお薦めします。

すべての注釈は、属性および要素の宣言内で指定できる属性の形式を取ります。これらの属性は、Oracle XML DB の名前空間 http://xmlns.oracle.com/xdb に属します。

表 5-2 に、要素および属性の宣言に指定できる Oracle XML DB の属性を示します。

表 5-2 要素に指定できる属性

属性	値	デフォルト	説明
SQLName	SQL 識別子	要素名	この XML 要素にマップされる SQL オブジェクト内での属性の名前を指定します。
SQLType	SQL 型の名前	要素名から生成された名前	この XML 要素の宣言に対応する SQL 型の名前を指定します。
SQLCollType	SQL コレクション型の名前	要素名から生成された名前	maxOccurs > 1 であるこの XML 要素に対応する SQL コレクション型の名前を指定します。
SQLSchema	SQL ユーザー名	XML Schema を登録するユーザー	SQLType によって指定された型を所有するデータベース・ユーザーの名前を指定します。
SQLCollSchema	SQL ユーザー名	XML Schema を登録するユーザー	SQLCollType によって指定された型を所有するデータベース・ユーザーの名前を指定します。
maintainOrder	TRUE | FALSE	TRUE	TRUE の場合、コレクションは VARRAY にマップされます。FALSE の場合、コレクションはネストした表にマップされます。

表 5-2 要素に指定できる属性（続き）

属性	値	デフォルト	説明
SQLInline	TRUE	FALSE	TRUE	TRUE の場合、この要素は埋込み属性（maxOccurs >1 の場合はコレクション）として表内に格納されます。FALSE の場合、REF（maxOccurs >1 の場合は REF のコレクション）が格納されます。SQL でインラインがサポートされない特定の状況（循環参照など）では、この属性は強制的に FALSE になります。
maintainDOM	TRUE	FALSE	TRUE	TRUE の場合、この要素のインスタンスは出力で DOM 再現性が保持されるように格納されます。これは、要素の順序の他に、すべてのコメント、処理命令、名前空間宣言などが保持されることを意味します。FALSE の場合、出力で入力と同じ DOM 動作が保持されない場合があります。
columnProps	有効な列 STORAGE 句	NULL	デフォルトの CREATE TABLE 文に挿入される列 STORAGE 句を指定します。主に、表にマップされる要素（最上位の要素の宣言および表外の要素の宣言）に有効です。
tableProps	有効な表の STORAGE 句	NULL	デフォルトの CREATE TABLE 文に追加される表 STORAGE 句を指定します。主に、グローバルな要素および表外の要素に有効です。
defaultTable	表の名前	要素名に基づきます。	このスキーマの XML インスタンスを格納する必要がある表の名前を指定します。表の名前が指定されていない API（FTP、HTTP など）を使用して XML を挿入する場合に最も有効です。
beanClassname	Java クラスの名前	要素名から生成されます。	要素の宣言内で使用できます。要素がグローバルな complexType に基づく場合、この名前は complexType の宣言内の beanClassname の値と同じである必要があります。ユーザーによって名前が指定されている場合、Bean を生成すると、要素名から名前が生成されるのではなく、この名前で Bean クラスが生成されます。

表 5-2 要素に指定できる属性（続き）

属性	値	デフォルト	説明
JavaClassname	Java クラスの名前	なし	対応する Bean クラスから導出された Java クラスの名前を指定するために使用し、Bean へのアクセス時にこのクラスのオブジェクトがインスタンス化されることを保証します。JavaClassname が指定されていない場合、Oracle XML DB によって Bean クラスのオブジェクトが直接インスタンス化されます。

表 5-3 グローバルな complexType を宣言する要素に指定できる属性

属性	値	デフォルト	説明
SQLType	SQL 型の名前	要素名から生成された名前	この XML 要素の宣言に対応する SQL 型の名前を指定します。
SQLSchema	SQL ユーザー名	XML Schema を登録するユーザー	SQLType によって指定された型を所有するデータベース・ユーザーの名前を指定します。
beanClassname	Java クラスの名前	要素名から生成されます。	要素の宣言内で使用できます。要素がグローバルな complexType に基づく場合、この名前は complexType 宣言内の beanClassname の値と同じである必要があります。ユーザーによって名前が指定される場合、Bean を生成すると、要素名から名前が生成されるのではなく、この名前で Bean クラスが生成されます。
maintainDOM	TRUE | FALSE	TRUE	TRUE の場合、この要素のインスタンスは出力で DOM 再現性が保持されるように格納されます。これは、要素の順序の他に、すべてのコメント、処理命令、名前空間の宣言などが保持されることを意味します。FALSE の場合、出力で入力と同じ DOM 動作が保持されない場合があります。

表 5-4 XML Schema の宣言に指定できる属性

属性	値	デフォルト	説明
mapUnboundedStringToLob	TRUE	FALSE	FALSE	TRUE の場合、デフォルトで、無制限の文字列が CLOB にマップされます。同様に、デフォルトで、無制限のバイナリ・データが BLOB にマップされます。FALSE の場合、無制限の文字列が VARCHAR2(4000) にマップされ、無制限のバイナリ・コンポーネントが RAW(2000) にマップされます。
storeVarrayAsTable	TRUE	FALSE	FALSE	TRUE の場合、VARRAY が表（OCT）として格納されます。FALSE の場合、VARRAY が LOB に格納されます。

登録時の XML Schema への SQL マッピングの指定

SQL マッピングについての情報は、XML Schema 文書に格納されます。5-29 ページの「DBMS_XMLSCHEMA を使用した型のマッピング」で説明するとおり、登録処理で SQL 型が生成され、マッピングについての情報を格納するために XML Schema 文書に注釈が追加されます。注釈は新しい属性の形式を取ります。

例 5-13 SQLType および SQLName 属性を使用した SQL マッピングの取得

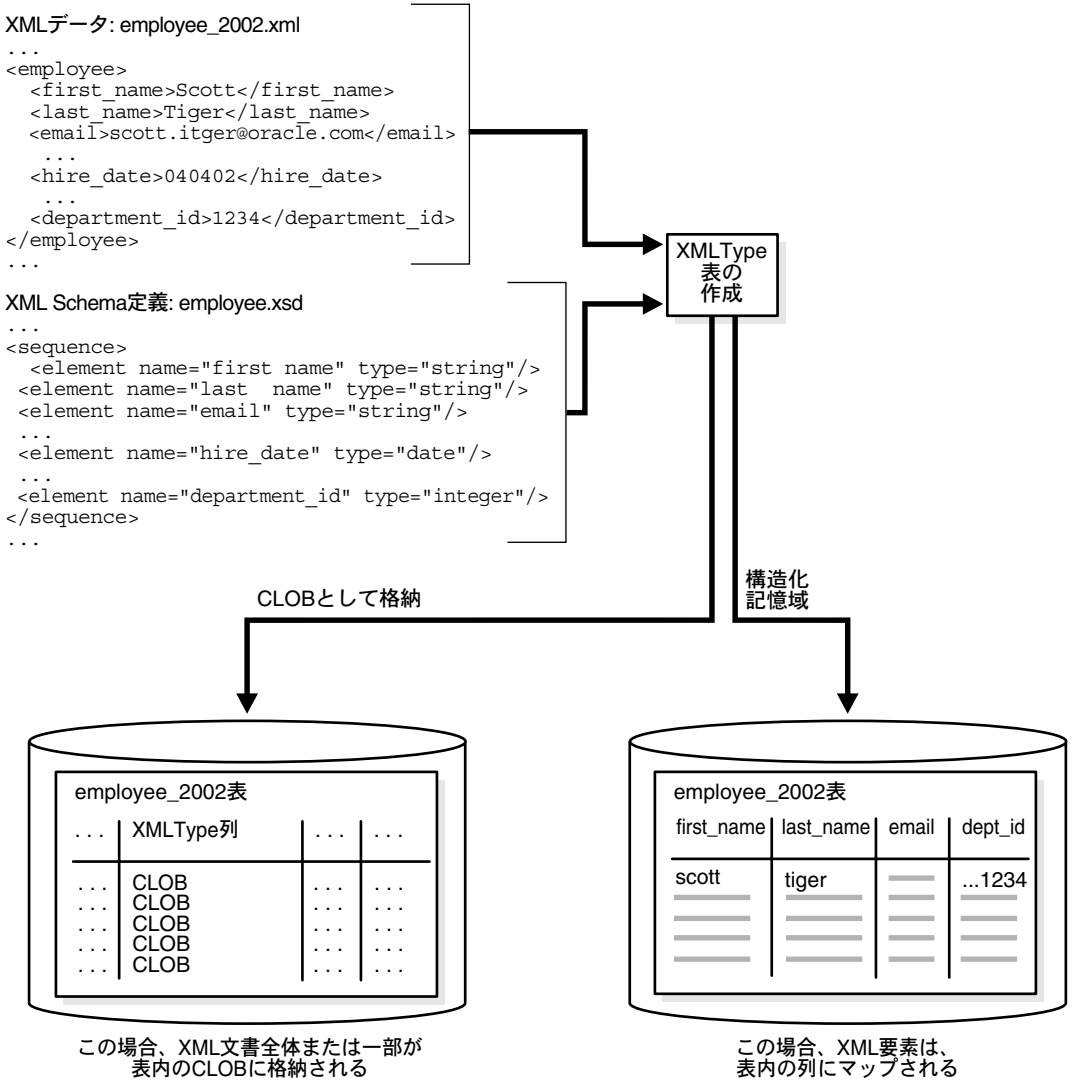
次の XML Schema 定義は、SQLType および SQLName 属性を使用して SQL マッピングについての情報を取得する方法を示します。

```
declare
     doc varchar2(3000) := '<schema targetNamespace="http://www.oracle.com/PO.xsd"
xmlns:po="http://www.oracle.com/PO.xsd" xmlns:xdb="http://xmlns.oracle.com/xdb"
xmlns="http://www.w3.org/2001/XMLSchema">
 <complexType name="PurchaseOrderType">
  <sequence>
   <element name="PONum" type="decimal" xdb:SQLName="PONUM" xdb:SQLType="NUMBER"/>
   <element name="Company" xdb:SQLName="COMPANY" xdb:SQLType="VARCHAR2">
    <simpleType>
     <restriction base="string">
      <maxLength value="100"/>
     </restriction>
    </simpleType>
   </element>
   <element name="Item"  xdb:SQLName="ITEM" xdb:SQLType="ITEM_T" maxOccurs="1000">
    <complexType>
     <sequence>
```

```
      <element name="Part"  xdb:SQLName="PART" xdb:SQLType="VARCHAR2">
       <simpleType>
        <restriction base="string">
         <maxLength value="1000"/>
        </restriction>
       </simpleType>
      </element>
      <element name="Price" type="float"  xdb:SQLName="PRICE" xdb:SQLType="NUMBER"/>
     </sequence>
    </complexType>
   </element>
  </sequence>
 </complexType>
 <element name="PurchaseOrder" type="po:PurchaseOrderType"/>
</schema>';
begin
   dbms_xmlschema.registerSchema('http://www.oracle.com/PO.xsd', doc);
end;
```

図 5-2 に、Oracle XML DB によって、XML 文書および XML Schema に指定されたマッピングを使用して、XML Schema に基づく XMLType 表を作成する方法を示します。まず XMLType 表が作成され、XML Schema に指定された格納方法によって、XML 文書が 1 つの CLOB として XMLType 列にマップおよび格納されるか、またはオブジェクト・リレーショナル形式で表のいくつかの列にまたがって格納されます。

図 5-2 Oracle XML DB による XML Schema に基づく XMLType 表のマッピング方法



まず XMLType 表が作成され、XML Schema に指定された格納方法によって、XML 文書が 1 つの CLOB として XMLType 列にマップおよび格納されるか、またはオブジェクト・リレーショナル形式で表のいくつかの列にまたがって格納されます。

DBMS_XMLSCHEMA を使用した型のマッピング

DBMS_XMLSCHEMA を使用して、属性および要素に対する型のマッピングの情報を設定します。

属性への型のマッピングについての情報の設定

属性の宣言には、次のいずれかを使用して型を指定できます。

- 基本型
- この XML Schema または外部の XML Schema 内で宣言されたグローバルな simpleType
- この XML Schema または外部の XML Schema 内で宣言されたグローバル属性に対する参照（ref=".."）
- ローカルな simpleType

メモリーのマッピングの情報と同様に、SQL 型および関連付けられた情報（長さおよび精度）も、属性の基礎となる simpleType から導出されます。

SQL 型のオーバーライド

入力 XML Schema 文書に SQLType の値を明示的に指定できます。この場合、指定された型に対して検証が行われます。これによって、次の特定の形式のオーバーライドが可能になります。

- デフォルトの型が STRING である場合、CHAR、VARCHAR または CLOB でオーバーライドできます。
- デフォルトの型が RAW である場合、RAW または BLOB でオーバーライドできます。

要素への型のマッピングについての情報の設定

要素の宣言では、次のいずれかを使用して型を指定できます。

- 属性の宣言に対して型を指定するいずれかの方法（5-29 ページの「属性への型のマッピングについての情報の設定」を参照）
- この XML Schema 文書または外部の XML Schema 内で指定されたグローバルな complexType
- この XML Schema 文書または外部の XML Schema 内に存在するグローバル要素（ref="..."）
- ローカルな complexType

SQL 型のオーバーライド

デフォルトでは、complexType に基づく要素は、各サブ要素および属性に対応する属性が含まれるオブジェクト型にマップされます。ただし、入力 XML Schema に SQLType 属性を明示的に指定することによって、このマッピングをオーバーライドできます。この場合、SQLType に対して次の値が使用できます。

- VARCHAR2
- RAW
- CLOB
- BLOB

これらの値によって、格納された XML はデータベース内でテキストまたは非分解形式で表されます。特別な場合の処理を次に示します。

- 要素の宣言に使用される complexType、および complexType 内で宣言された要素の処理の一部として循環が検出された場合、SQLInline 属性は強制的に FALSE に設定され、正しい SQL マッピングは REF XMLTYPE に設定されます。
- maxOccurs> 1 である場合、VARRAY 型を作成する必要がある場合があります。
  - SQLInline ="true" である場合、要素型が決定済の SQL 型である VARRAY 型が作成されます。
    * VARRAY のカーディナリティは、maxOccurs 属性の値に基づいて決定されます。
    * VARRAY 型の名前は、ユーザーが SQLCollType 属性を使用して明示的に指定するか、または要素名の加工によって取得します。
  - SQLInline="false" である場合、SQL 型は、XMLType に対する REF の配列を表す事前定義された型である XDB.XDB$XMLTYPE_REF_LIST_T に設定されます。
- 要素がグローバル要素である場合、または SQLInline="false" である場合、デフォルト表を作成する必要があります。これは、表作成コンテキストに追加されます。デフォルト表の名前は、ユーザーが指定するか、または要素名の加工によって導出されます。

XML Schema: SQL への simpleType のマッピング

この項では、XML Schema 定義によって XML Schema simpleType を SQL オブジェクト型にマップする方法を説明します。図 5-3 に例を示します。

表 5-5 ～表 5-8 に、XML Schema 定義に指定された、SQL への XML Schema simpleType のデフォルトのマッピングを示します。次に例を示します。

- XML 基本型は、最も近い SQL データ型にマップされます。たとえば、DECIMAL、POSITIVEINTEGER および FLOAT は、すべて SQL の NUMBER にマップされます。

- XML 列挙型は、1 つの RAW(n) 属性を持つオブジェクト型にマップされます。n の値は、列挙の宣言で使用可能な値の数によって決定されます。
- XML リストまたは UNION データ型は、SQL の文字列（VARCHAR2/CLOB）データ型にマップされます。

図 5-3 simpleType のマッピング : XML 文字列から SQL の VARCHAR2 または CLOB

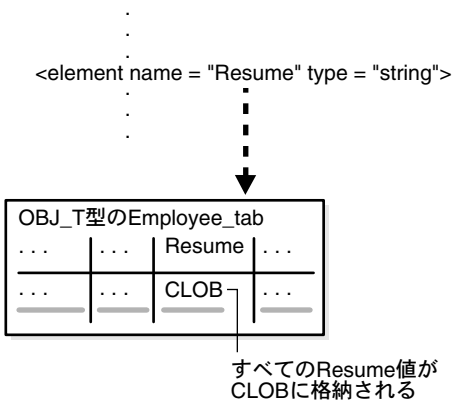


表 5-5 SQL への XML 文字列データ型のマッピング

XML 基本型	length または maxLength ファセット	デフォルトのマッピング	互換性のあるデータ型
string	n	n < 4000 の場合は VARCHAR2(n)、 それ以外の場合は VARCHAR2(4000)	CHAR、VARCHAR2、CLOB
string	--	mapUnboundedStringToLob="false" の場合は VARCHAR2(4000)、 それ以外の場合は CLOB	CHAR、VARCHAR2、CLOB

表 5-6 SQL への XML バイナリ・データ型（hexBinary/base64Binary）のマッピング

XML 基本型	length または maxLength ファセット	デフォルトのマッピング	互換性のあるデータ型
hexBinary、base64Binary	n	n < 2000 の場合は RAW(n)、 それ以外の場合は RAW(2000)	RAW、BLOB
hexBinary、base64Binary	-	mapUnboundedStringToLob="false" の場合は RAW(2000)、 それ以外の場合は BLOB	RAW、BLOB

表 5-7 SQL への XML 数値基本型のデフォルトのマッピング

XML simpleType	デフォルトの Oracle データ型	指定された totalDigits (m)、fractionDigits(n)	互換性のあるデータ型
float	NUMBER	NUMBER(m,n)	NUMBER、FLOAT、DOUBLE
double	NUMBER	NUMBER(m,n)	NUMBER、FLOAT、DOUBLE
decimal	NUMBER	NUMBER(m,n)	NUMBER、FLOAT、DOUBLE
integer	NUMBER	NUMBER(m,n)	NUMBER、FLOAT、DOUBLE
nonNegativeInteger	NUMBER	NUMBER(m,n)	NUMBER、FLOAT、DOUBLE
positiveInteger	NUMBER	NUMBER(m,n)	NUMBER、FLOAT、DOUBLE
nonPositiveInteger	NUMBER	NUMBER(m,n)	NUMBER、FLOAT、DOUBLE
negativeInteger	NUMBER	NUMBER(m,n)	NUMBER、FLOAT、DOUBLE
long	NUMBER(20)	NUMBER(m,n)	NUMBER、FLOAT、DOUBLE
unsignedLong	NUMBER(20)	NUMBER(m,n)	NUMBER、FLOAT、DOUBLE
int	NUMBER(10)	NUMBER(m,n)	NUMBER、FLOAT、DOUBLE
unsignedInt	NUMBER(10)	NUMBER(m,n)	NUMBER、FLOAT、DOUBLE
short	NUMBER(5)	NUMBER(m,n)	NUMBER、FLOAT、DOUBLE
unsignedShort	NUMBER(5)	NUMBER(m,n)	NUMBER、FLOAT、DOUBLE
byte	NUMBER(3)	NUMBER(m,n)	NUMBER、FLOAT、DOUBLE
unsignedByte	NUMBER(3)	NUMBER(m,n)	NUMBER、FLOAT、DOUBLE

表 5-8 SQL への XML 日付データ型のマッピング

XML 基本型	デフォルトのマッピング	互換性のあるデータ型
datetime	TIMESTAMP	DATE
time	TIMESTAMP	DATE
date	DATE	DATE
gDay	DATE	DATE
gMonth	DATE	DATE
gYear	DATE	DATE
gYearMonth	DATE	DATE
gMonthDay	DATE	DATE
duration	VARCHAR2(4000)	なし

表 5-9 SQL へのその他の XML 基本データ型のデフォルトのマッピング

XML simpleType	デフォルトの Oracle DataType	互換性のあるデータ型
boolean	RAW(1)	VARCHAR2
Language(string)	VARCHAR2(4000)	CLOB、CHAR
NMTOKEN(string)	VARCHAR2(4000)	CLOB、CHAR
NMTOKENS(string)	VARCHAR2(4000)	CLOB、CHAR
Name(string)	VARCHAR2(4000)	CLOB、CHAR
NCName(string)	VARCHAR2(4000)	CLOB、CHAR
ID	VARCHAR2(4000)	CLOB、CHAR
IDREF	VARCHAR2(4000)	CLOB、CHAR
IDREFS	VARCHAR2(4000)	CLOB、CHAR
ENTITY	VARCHAR2(4000)	CLOB、CHAR
ENTITIES	VARCHAR2(4000)	CLOB、CHAR
NOTATION	VARCHAR2(4000)	CLOB、CHAR
anyURI	VARCHAR2(4000)	CLOB、CHAR

表 5-9 SQL へのその他の XML 基本データ型のデフォルトのマッピング（続き）

XML simpleType	デフォルトの Oracle DataType	互換性のあるデータ型
anyType	VARCHAR2(4000)	CLOB、CHAR
anySimpleType	VARCHAR2(4000)	CLOB、CHAR
QName	XDB.XDB$QNAME	--

simpleType: SQL の VARCHAR2 または CLOB への XML 文字列のマッピング

XML Schema で、データ型が、maxLength の値が 4000 未満の文字列に指定されている場合、このデータ型は、指定された長さの VARCHAR2 属性にマップされます。ただし、XML Schema に maxLength が指定されていない場合は、LOB のみにマップできます。これは、ほとんどの文字列の値が小さく、LOB を必要とする値がわずかである場合は、あまり効率的ではありません。

> **参照：** 表 5-5「SQL への XML 文字列データ型のマッピング」を参照してください。

XML Schema: SQL への complexType のマッピング

XML Schema を使用して、次のとおり complexType が SQL オブジェクト型にマップされます。

- complexType 内で宣言された XML 属性は、オブジェクト属性にマップされます。XML 属性を定義する simpleType によって、対応する属性の SQL データ型が決定されます。
- complexType 内で宣言された XML 要素も、オブジェクト属性にマップされます。オブジェクト属性のデータ型は、XML 要素を定義する simpleType または complexType によって決定されます。

maxOccurs 属性 > 1 の場合に XML 要素が宣言されると、SQL の collection 属性にマップされます。collection は、VARRAY（デフォルト）またはネストした表（maintainOrder 属性が FALSE に設定されている場合）になります。また、VARRAY は、デフォルトでは、LOB ではなく Ordered Collections in Tables（OCT）に格納されます。storeAsLob 属性を TRUE に設定することによって、LOB 記憶域を選択できます。

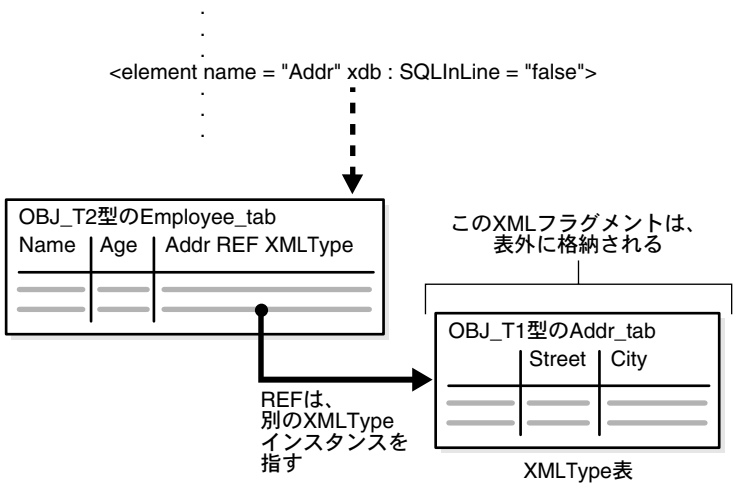
> **参照：** 5-65 ページの「Ordered Collections in Tables（OCT）」を参照してください。

表外に格納するための SQLInLine 属性の FALSE への設定

デフォルトでは、サブ要素は埋込みオブジェクト属性にマップされます。ただし、表外に格納することによってパフォーマンスが向上する場合もあります。この場合、SQLInline 属性を FALSE に設定し、Oracle XML DB によって REF 属性が埋め込まれたオブジェクト型を生成できます。REF は、表外に格納される XML フラグメントに対応する XMLType の他のインスタンスを指します。また、デフォルトの XMLType 表が作成され、この表に表外のフラグメントが格納されます。

図 5-4 に、表外に格納するための SQL への complexType のマッピングを示します。

図 5-4 表外に格納するための SQL への complexType のマッピング



例 5-14 Oracle XML DB の XML Schema: complexType のマッピング - 表外に格納するための SQLInLine 属性の FALSE への設定

この例では、Addr 要素の xdb:SQLInLine 属性が FALSE に設定されています。生成される OBJ_T2 オブジェクト型には、REF 属性が埋め込まれた XMLType 型の列が含まれています。REF 属性は、Addr_tab 表の OBJ_T1 オブジェクト型で作成された他の XMLType インスタンスを指します。Addr_tab には、Street 列および City 列が含まれています。後者の XMLType インスタンスが表外に格納されます。

```
declare
        doc varchar2(3000) := '<schema xmlns="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.oracle.com/emp.xsd"
xmlns:emp="http://www.oracle.com/emp.xsd"
xmlns:xdb="http://xmlns.oracle.com/xdb">
     <complexType name = "Employee" xdb:SQLType="OBJ_T2">
```

```
        <sequence>
           <element name = "Name" type = "string"/>
           <element name = "Age" type = "decimal"/>
           <element name = "Addr" xdb:SQLInline = "false">
              <complexType xdb:SQLType="OBJ_T1">
                 <sequence>
                    <element name = "Street" type = "string"/>
                    <element name = "City" type = "string"/>
                 </sequence>
              </complexType>
           </element>
        </sequence>
     </complexType>
  </schema>';
begin
        dbms_xmlschema.registerSchema('http://www.oracle.com/PO.xsd', doc);
end;
```

この XML Schema を登録すると、Oracle XML DB によって次の型および XMLType 表が生成されます。

```
CREATE TYPE OBJ_T1 AS OBJECT
(
  SYS_XDBPD$ XDB.XDB$RAW_LIST_T,
  Street VARCHAR2(4000),
  City VARCHAR2(4000)
);

CREATE TYPE OBJ_T2 AS OBJECT
(
  SYS_XDBPD$ XDB.XDB$RAW_LIST_T,
  Name VARCHAR2(4000),
  Age NUMBER,
  Addr REF XMLType
);
```

ラージ・オブジェクト（LOB）への XML フラグメントのマッピング

図 5-5 に示すとおり、複雑な要素に対する SQLType をキャラクタ・ラージ・オブジェクト（CLOB）またはバイナリ・ラージ・オブジェクト（BLOB）に指定できます。この場合、XML フラグメント全体が LOB 属性に格納されます。これは、XML 文書に問合せがほとんど行われない部分があり、多くの場合、これらの部分の取出しおよび格納が個別に行われている場合に有効です。XML フラグメントを LOB として格納することによって、解析、分解、再組立てのオーバーヘッドを軽減できます。

例 5-15 Oracle XML DB の XML Schema: LOB への complexType の XML フラグメントのマッピング

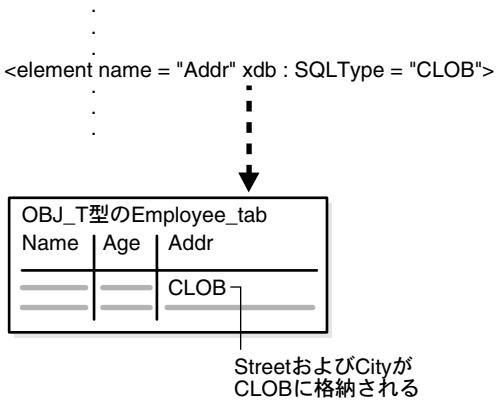
次の例では、XML Schema によって、XML フラグメントの Addr 要素で SQLType="CLOB" 属性が使用されるように指定します。

```
declare
       doc varchar2(3000) := '<schema xmlns="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.oracle.com/emp.xsd"
xmlns:emp="http://www.oracle.com/emp.xsd"
xmlns:xdb="http://xmlns.oracle.com/xdb">
     <complexType name = "Employee" xdb:SQLType="OBJ_T2">
        <sequence>
           <element name = "Name" type = "string"/>
           <element name = "Age" type = "decimal"/>
           <element name = "Addr" xdb:SQLType = "CLOB">
              <complexType >
                 <sequence>
                    <element name = "Street" type = "string"/>
                    <element name = "City" type = "string"/>
                 </sequence>
              </complexType>
           </element>
        </sequence>
     </complexType>
  </schema>';
begin
       dbms_xmlschema.registerSchema('http://www.oracle.com/PO.xsd', doc);
end;
```

この XML Schema を登録すると、Oracle XML DB によって次の型および XMLType 表が生成されます。

```
CREATE TYPE OBJ_T AS OBJECT
(
  SYS_XDBPD$ XDB.XDB$RAW_LIST_T,
  Name VARCHAR2(4000),
  Age NUMBER,
  Addr CLOB
);
```

図 5-5 キャラクタ・ラージ・オブジェクト（CLOB）への complexType の XML フラグメントのマッピング



Oracle XML DB の complexType の拡張および制限

XML Schema では、complexType は complexContent および simpleContent に基づいて宣言されます。

- simpleContent は、simpleType の拡張として宣言されます。
- complexContent は、次のいずれかとして宣言されます。
  - ベース型
  - complexType の拡張
  - complexType の制限

XML Schema での complexType の宣言 : 継承の処理

complexType の場合、次のとおり Oracle XML DB によって XML Schema で継承が処理されます。

- **他の complexType を拡張するように宣言された complexType の場合、**ベース型に対応する SQL 型が、現在の SQL 型のスーパータイプとして指定されます。サブ complexType に宣言された追加属性および要素のみが、サブ・オブジェクト型に属性として追加されます。
- **他の complexType を制限するように宣言された complexType の場合、**サブ complexType の SQL 型がベース型の SQL 型と同じになるように設定されます。これ

は、SQL では継承メカニズムを介したオブジェクト型の制限がサポートされないためです。すべての制約は、XML Schema での制限によるものです。

例 5-16 XML Schema での継承 : complexType の拡張としての complexContent

「Address」というベース complexType、および「USAddress」、「IntlAddress」という 2 つの拡張を定義する XML Schema について考えてみます。

```
declare
        doc varchar2(3000) := '<xs:schema
xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns:xdb="http://xmlns.oracle.com/xdb">
  <xs:complexType name="Address" xdb:SQLType="ADDR_T">
    <xs:sequence>
      <xs:element name="street" type="xs:string"/>
      <xs:element name="city" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="USAddress" xdb:SQLType="USADDR_T">
   <xs:complexContent>
    <xs:extension base="Address">
      <xs:sequence>
        <xs:element name="zip" type="xs:string"/>
      </xs:sequence>
    </xs:extension>
   </xs:complexContent>
  </xs:complexType>

  <xs:complexType name="IntlAddress" final="#all" xdb:SQLType="INTLADDR_T">
   <xs:complexContent>
    <xs:extension base="Address">
      <xs:sequence>
        <xs:element name="country" type="xs:string"/>
      </xs:sequence>
    </xs:extension>
   </xs:complexContent>
  </xs:complexType>
</xs:schema>';
begin
        dbms_xmlschema.registerSchema('http://www.oracle.com/PO.xsd', doc);
end;
```

> **注意：** 対応する complexType によって final 属性が指定されるため、INTLADDR_T 型は FINAL 型として作成されます。デフォルトでは、すべての complexType を他の型によって拡張および制限できるため、すべての SQL オブジェクト型が NOT FINAL 型として作成されます。

```
create type ADDR_T as object (
  SYS_XDBPD$ XDB.XDB$RAW_LIST_T,
  "street" varchar2(4000),
  "city" varchar2(4000)
) not final;

create type USADDR_T under ADDR_T (
 "zip" varchar2(4000)
) not final;

create type INTLADDR_T under ADDR_T (
  "country" varchar2(4000)
) final;
```

例 5-17 XML Schema での継承 : complexType の制限

ベース complexType の Address、および country 属性を指定できないようにする制限型 LocalAddress を定義する XML Schema について考えてみます。

```
declare
        doc varchar2(3000) := '<xs:schema
xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns:xdb="http://xmlns.oracle.com/xdb">
  <xs:complexType name="Address" xdb:SQLType="ADDR_T">
    <xs:sequence>
      <xs:element name="street" type="xs:string"/>
      <xs:element name="city" type="xs:string"/>
      <xs:element name="zip" type="xs:string"/>
      <xs:element name="country" type="xs:string" minOccurs="0" maxOccurs="1"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="LocalAddress" xdb:SQLType="USADDR_T">
    <xs:complexContent>
     <xs:restriction base="Address">
      <xs:sequence>
        <xs:element name="street" type="xs:string"/>
        <xs:element name="city" type="xs:string"/>
        <xs:element name="zip" type="xs:string"/>
        <xs:element name="country" type="xs:string"
```

```
            minOccurs="0" maxOccurs="0"/>
      </xs:sequence>
     </xs:restriction>
    </xs:complexContent>
  </xs:complexType>
</xs:schema>';
begin
        dbms_xmlschema.registerSchema('http://www.oracle.com/PO.xsd', doc);
end;
```

SQL での継承では制限の概念がサポートされないため、制限された complexType に対応する SQL 型は、親オブジェクト型の空のサブタイプになります。前述の XML Schema に対して、次の SQL 型が生成されます。

```
create type ADDR_T as object (
  SYS_XDBPD$ XDB.XDB$RAW_LIST_T,
  "street" varchar2(4000),
  "city" varchar2(4000),
  "zip" varchar2(4000),
  "country" varchar2(4000)
) not final;

create type USADDR_T under ADDR_T;
```

complexType のマッピング : simpleContent

simpleContent 宣言に基づく complexType は、XML 属性に対応する属性および本体の値に対応する追加の SYS_XDBBODY$ 属性を持つオブジェクト型にマップされます。本体属性のデータ型は、本体の型を定義する simpleType に基づきます。

例 5-18 XML Schema の complexType: simpleContent への complexType のマッピング

```
declare
        doc varchar2(3000) := '<schema xmlns="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.oracle.com/emp.xsd"
xmlns:emp="http://www.oracle.com/emp.xsd" xmlns:xdb="http://xmlns.oracle.com/xdb">
<complexType name="name" xdb:SQLType="OBJ_T">
  <simpleContent>
    <restriction base = "string">
    </restriction>
  </simpleContent>
</complexType>
</schema>';
begin
        dbms_xmlschema.registerSchema('http://www.oracle.com/emp.xsd', doc);
end;
```

この XML Schema を登録すると、Oracle XML DB によって次の型および XMLType 表が生成されます。

```
create type OBJ_T as object
(
  SYS_XDBPD$  xdb.xdb$raw_list_t,
  SYS_XDBBODY$ VARCHAR2(4000)
);
```

complexType のマッピング : any および anyAttribute

Oracle XML DB では、要素の宣言である any および属性の宣言である anyAttribute は、作成されたオブジェクト型の VARCHAR2 属性（またはオプションでラージ・オブジェクト（LOB））にマップされます。オブジェクト属性によって、any 宣言に一致する XML フラグメントのテキストが格納されます。

- コンテンツが指定された名前空間に属するように、namespace 属性を使用してコンテンツを制限できます。
- any 要素宣言内の processContents 属性は、any 宣言に一致するコンテンツに必要な検証レベルを示します。

例 5-19 Oracle XML DB の XML Schema: any/anyAttribute への complexType のマッピング

この XML Schema の例では、any 要素が宣言され、この要素が OBJ_T オブジェクト型の SYS_XDBANY$ 列にマップされます。また、この要素によって、processContents 属性が any 宣言に一致する検証対象のコンテンツをスキップすることが宣言されます。

```
declare
doc varchar2(3000) := '<schema xmlns="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.oracle.com/any.xsd"
xmlns:emp="http://www.oracle.com/any.xsd" xmlns:xdb="http://xmlns.oracle.com/xdb">
 <complexType name = "Employee" xdb:SQLType="OBJ_T">
  <sequence>
     <element name = "Name" type = "string" />
     <element name = "Age" type = "decimal"/>
     <any namespace = "http://www/w3.org/2001/xhtml" processContents = "skip"/>
  </sequence>
 </complexType>
</schema>';
begin
     dbms_xmlschema.registerSchema('http://www.oracle.com/emp.xsd', doc);
end;
```

結果として次の文が生成されます。

```
CREATE TYPE OBJ_T AS OBJECT
(
```

```
  SYS_XDBPD$  xdb.xdb$raw_list_t,
  Name VARCHAR2(4000),
  Age NUMBER,
  SYS_XDBANY$ VARCHAR2(4000)
);
```

XML Schema の complexType 間での循環の処理

オブジェクト型では循環が許可されていないため、オブジェクト型の生成時、循環が完了した時点で REF 属性を設定することによって、XML Schema での循環が使用不可になります。そのため、データの一部は表外に格納され、親である XML 文書の一部として取得されます。

例 5-20 XML Schema: complexType 間での循環

XML Schema では、complexType の定義間に循環を設定できます。図 5-6 に示す例では、complexType CT1 の定義によって別の complexType CT2 が参照され、CT2 の定義によって最初の型 CT1 が参照されます。

XML Schema では、complexType の定義間に循環を設定できます。長さが 2 である循環の例を示します。

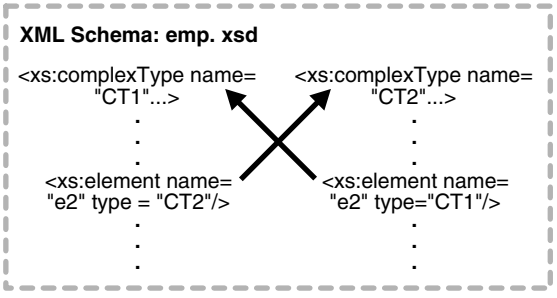
```
declare
doc varchar2(3000) := '<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns:xdb="http://xmlns.oracle.com/xdb">
  <xs:complexType name="CT1" xdb:SQLType="CT1">
    <xs:sequence>
      <xs:element name="e1" type="xs:string"/>
      <xs:element name="e2" type="CT2"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="CT2" xdb:SQLType="CT2">
    <xs:sequence>
      <xs:element name="e1" type="xs:string"/>
      <xs:element name="e2" type="CT1"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>';
begin
        dbms_xmlschema.registerSchema('http://www.oracle.com/emp.xsd', doc);
end;
```

SQL 型では、型の定義で循環を使用できません。ただし、REF（参照）属性を伴う循環など、弱い循環はサポートされます。そのため、必要に応じて強制的に SQLInline="false" を設定することによって循環が回避されるように、循環する XML Schema 定義は SQL オブジェクト型にマップされます。これによって、弱い循環が作成されます。

前述の XML Schema の場合、次の SQL 型が生成されます。

```
create type CT1 as object
(
  SYS_XDBPD$  xdb.xdb$raw_list_t,
  "e1" varchar2(4000),
  "e2" ref xmltype;
) not final;

create type CT2 as object
(
  SYS_XDBPD$  xdb.xdb$raw_list_t,
  "e1" varchar2(4000),
  "e2" CT1
) not final;
```

図 5-6 同じ XML Schema の異なる complexType 間での相互参照



例 5-21 XML Schema: complexType 間での循環（自己参照）

循環する complexType に、自己参照する complexType の宣言が含まれる場合があります。自己参照する <SectionT> 型の例を示します。

```
declare
        doc varchar2(3000) := '<xs:schema
xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:xdb="http://xmlns.oracle.com/xdb">
  <xs:complexType name="SectionT" xdb:SQLType="SECTION_T">
    <xs:sequence>
      <xs:element name="title" type="xs:string"/>
      <xs:choice maxOccurs="unbounded">
        <xs:element name="body" type="xs:string" xdb:SQLCollType="BODY_COLL"/>
        <xs:element name="section" type="SectionT"/>
      </xs:choice>
    </xs:sequence>
```

```
  </xs:complexType>
</xs:schema>';
begin
        dbms_xmlschema.registerSchema('http://www.oracle.com/section.xsd', doc);
end;
```

次の SQL 型が生成されます。

> **注意：** section 属性は、XMLType インスタンスに対する REF の VARRAY として宣言されます。複数の埋込みセクションが出現する場合があるため、属性が VARRAY になります。また、SQL オブジェクトの循環が構成されないようにするためにも、属性が XMLType に対する REF の VARRAY になります。

```
create type BODY_COLL as varray(32767) of VARCHAR2(4000);

create type SECTION_T as object
(
  SYS_XDBPD$  xdb.xdb$raw_list_t,
  "title" varchar2(4000),
  "body" BODY_COLL,
  "section" XDB.XDB$REF_LIST_T
) not final;
```

XML Schema に基づく XML 表を作成する場合の詳細なガイドライン

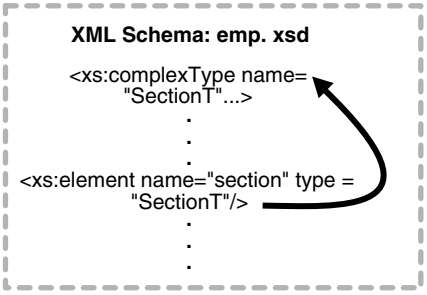
http://www.oracle.com/PO.xsd によって識別される XML Schema が登録されているとします。次のとおり XMLType 表 myPOs を作成し、この XML Schema の PurchaseOrder 要素に準拠するインスタンスをオブジェクト・リレーショナル形式で格納できます。

```
CREATE TABLE MyPOs OF XMLTYPE
   ELEMENT "http://www.oracle.com/PO.xsd#PurchaseOrder";
```

図 5-7 に、complexType によって自己参照および自己循環が行われる方法を示します。

> **参照：** 5-66 ページの「XML Schema 間での循環参照」を参照してください。

図 5-7 XML Schema 内での complexType による自己参照



非表示列が作成されます。これらの列は、PurchaseOrder 要素がマップされているオブジェクト型に対応しています。また、名前空間宣言など、最上位インスタンスのデータを格納するための XMLExtra オブジェクト列が作成されます。

注意： XMLDATA は、基礎となるオブジェクト列への直接アクセスを可能にする XMLType の擬似属性です。4-13 ページの「XMLData を使用した XMLType 列の記憶域オプションの変更」を参照してください。

XMLType の CREATE TABLE 文への STORAGE 句の指定

格納方法を指定するため、オブジェクト表記法または XML 表記法のいずれかを使用して、XMLType の STORAGE 句で基礎となる列を参照できます。

- **オブジェクト表記法 :** XMLDATA.<attr1>.<attr2>....

  次に例を示します。

```
CREATE TABLE  MyPOs OF XMLTYPE
   ELEMENT "http://www.oracle.com/PO.xsd#PurchaseOrder"
         lob (xmldata.lobattr) STORE AS (tablespace ...);
```

- **XML 表記法 :** extractValue(xmltypecol, '/attr1/attr2')

  次に例を示します。

```
CREATE TABLE MyPOs OF XMLTYPE
   ELEMENT "http://www.oracle.com/PO.xsd#PurchaseOrder"
         lob (ExtractValue(MyPOs, '/lobattr')) STORE AS (tablespace ...);
```

CREATE INDEX を使用した XMLType 列の参照

前述の例に示すとおり、XMLType 列の基礎となる列は、CREATE TABLE 文でオブジェクト表記法または XML 表記法のいずれかを使用して参照できます。CREATE INDEX 文の場合も同様です。

```
CREATE INDEX ponum_idx ON MyPOs (xmldata.ponum);
CREATE INDEX ponum_idx ON MyPOs p (ExtractValue(p, '/ponum');
```

XMLType 列に対する制約の指定

オブジェクト表記法または XML 表記法のいずれかを使用して、基礎となる XMLType 列に対して制約を指定することもできます。

- **オブジェクト表記法**

```
CREATE TABLE MyPOs OF XMLTYPE
ELEMENT "http://www.oracle.com/PO.xsd#PurchaseOrder" (unique(xmldata.ponum));
```

- **XML 表記法**

```
CREATE TABLE MyPOs P OF XMLTYPE
ELEMENT
"http://www.oracle.com/PO.xsd#PurchaseOrder"(unique(ExtractValue(p,'/ponum'));
```

XMLType 列への新しいインスタンスの挿入

次のとおり、XMLType 列に新しいインスタンスを挿入できます。

```
INSERT INTO MyPOs VALUES
     (xmltype.createxml('<PurchaseOrder>.....</PurchaseOrder>'));
```

XML Schema に基づく構造化記憶域でのクエリー・リライト

クエリー・リライトの概要

XMLType が XML Schema を使用して構造化記憶域に（オブジェクト・リレーショナル形式で）格納されており、XPath を使用する問合せが使用された場合、これらの問合せは、基礎となるオブジェクト・リレーショナル列に直接リライトされる可能性があります。また、この問合せのリライトは、XML Schema に基づかない特定の XMLType ビューに対して、XPath を使用する問合せが発行された場合にも実行される可能性があります。

これによって、B* ツリーまたは他の索引が列に存在する場合、これらをオプティマイザによる問合せの評価に使用できます。このクエリー・リライト・メカニズムは、existsNode()、extract()、extractValue()、updateXML() などの SQL 関数の XPath に対して使用されます。これによって、メモリー内で XML 文書を構成することなく、XML 文書に対して XPath を評価できます。

注意： リライトされるパス問合せは、サポートされる XPath 問合せのセットのサブセットです。できるだけ、クエリー・リライトの効果を最大限に引き出せるように問合せを記述してください。

例 5-22 クエリー・リライト

たとえば、次のような問合せが実行されるとします。

```
SELECT VALUE(p) FROM MyPOs p
    WHERE extractValue(value(p),'/PurchaseOrder/Company') = 'Oracle';
```

この問合せでは、Company 要素の値の取得およびリテラル 'Oracle' との比較が試行されます。MyPOs 表は XML Schema に基づく構造化記憶域で作成されているため、extractValue 演算子は、Purchaseorder についての企業情報が格納される基礎となるリレーショナル列にリライトされます。

前述の問合せは次の列にリライトされます。

```
SELECT VALUE(p) FROM MyPOs p
WHERE p.xmldata.company = 'Oracle';
```

参照： 第 4 章「XMLType の使用」を参照してください。

Company 列に対して通常の索引が作成されている場合の例を示します。

```
CREATE INDEX company_index ON MyPos e
      (extractvalue(value(e),'/PurchaseOrder/Company'));
```

この場合、前述の問合せでは、索引を使用して評価が行われます。

オブジェクト型 emp_t に対して、SYS_XMLGEN() を使用して作成された XMLType ビュー employee_xml について考えてみます。

```
CREATE TYPE Emp_t AS OBJECT (
  EMPNO NUMBER(4),
  ENAME VARCHAR2(10),
  MGR   NUMBER(4),
  HIREDATE DATE,
  SAL   NUMBER(7,2),
  COMM  NUMBER(7,2));

CREATE VIEW employee_xml OF XMLTYPE
  WITH OBJECT  ID
         (SYS_NC_ROWINFO$.EXTRACT('/ROW/EMPNO/text()').getnumberval()) AS
    SELECT SYS_XMLGEN(
         emp_t(e.empno, e.ename, e.job, e.mgr, e.hiredate, e.sal, e.comm))
      FROM emp e;
```

従業員番号が 7934 であるすべての従業員を取り出す次の問合せについて考えてみます。

```
SELECT VALUE(p) FROM employee_xml p
WHERE extractValue(value(p),'/ROW/EMPNO') = 7934;
```

同様に、この問合せは EMPNO の値を 12300 と一致させようとしています。この問合せは、次のようにリライトされます。

```
SELECT SYS_XMLGEN(
     emp_t(e.empno, e.ename, e.job, e.mgr, e.hiredate, e.sal, e.comm)) "value(p)"
     FROM emp e WHERE emp_t(e.empno, e.ename, e.job, e.mgr, e.hiredate, e.sal,
     e.comm).empno - 7934;
```

さらに、WHERE 句は最適化されて次の問合せになります。

```
SELECT SYS_XMLGEN(
     emp_t(e.empno, e.ename, e.job, e.mgr, e.hiredate, e.sal, e.comm)) "value(p)"
     FROM emp e where e.empno = 7934;
```

empno 列に作成された索引 empno_index が取り出されます。

```
CREATE INDEX empno_index ON emp (empno);
```

また、ビューをオブジェクト・ビュー上に定義する場合もあります。オブジェクト・ビューは、リレーショナル・データ上にすでに存在しているか、または作成されています。既存のリレーショナル・データ上に作成された次のオブジェクト・ビューについて考えてみます。

```
CREATE VIEW employee_ov OF emp_t WITH OBJECT ID (EMPNO) AS
    SELECT emp_t(e.empno, e.ename, e.job, e.mgr, e.hiredate, e.sal, e.comm)
    FROM emp e;
```

XML Schema に基づかない XMLType ビューが、次のように作成されます。

```
CREATE VIEW employee_xml_xv OF XMLTYPE WITH OBJECT  ID
    (SYS_NC_ROWINFO$.EXTRACT('/ROW/EMPNO/text()').getnumberval()) AS
     SELECT SYS_XMLGEN(value(p)) FROM employee_ov p;
```

クエリー・リライトが行われる場合

クエリー・リライトは、次の SQL 関数に対して行われます。

- extract()
- existsNode()
- extractValue
- updateXML

問合せ、DML または DDL 文のいずれかの式にこれらの SQL 関数が存在している場合に、リライトが行われます。たとえば、extractValue() を使用して、基礎となるリレーショナル列に対する索引を作成できます。

リライトされる XPath 式

ワイルド・カードまたは descendant 軸が含まれない単純な式を持つ XPath 式は、リライトされます。XPath によって要素または属性ノードが選択される場合もあります。述語がサポートされ、SQL 述語にリライトされます。また、単一で一意の XPath にマップできることが判断された場合に、ワイルド・カードおよび descendant 軸はリライトされ、XML Schema に基づく XMLType 表およびビューに対する制限付きでサポートされます。

表 5-10 に、今回のリリースで、基礎となる SQL 問合せに変換できる XPath 式の種類を示します。

表 5-10 基礎となる SQL 問合せへの変換がサポートされる XPath 式

変換される XPath 式	説明
単純な XPath 式 : /PurchaseOrder/@PurchaseDate /PurchaseOrder/Company	属性自体が単純なスカラー型またはオブジェクト型である場合、オブジェクト型の属性のみに対して全検索が行われます。サポートされる軸は、child 軸および attribute 軸のみです。
コレクション全検索式 : /PurchaseOrder/Item/Part	コレクション式の全検索が行われます。サポートされる軸は、child 軸および attribute 軸のみです。CREATE INDEX または updateXML() の実行時に SQL 演算子が使用される場合、コレクションの全検索はサポートされません。
述語 : [Company="Oracle"] /ROW[DEPTNAME="ACCOUNTING"]	XPath の述語は、SQL 述語にリライトされます。updateXML() の場合、述語はリライトされません。
リスト索引 : lineitem[1] /ROW/EMPLOYEES[1]	コレクションの n 番目の項目にアクセスするように索引がリライトされます。updateXML() の場合、索引はリライトされません。

クエリー・リライトがサポートされる XPath 構造体

次の XPath 構造体はリライトされます。

- 単純な XPath 全検索
- 述語
- descendant 軸（XML Schema に基づく場合のみ）: descendant 軸（//）に対するリライトは、次の条件が満たされる場合にサポートされます。
  - // の後に、XPath の子または属性へのアクセスが 1 回以上行われる。
  - 1 つの子孫のみが、// に続く XPath の子または属性名と一致する。複数の子孫が一致する可能性があることがスキーマに指定され、// が拡張できる一意のパスが存在しない場合、リライトは行われない。
  - 子孫が xsi:anyType 型の要素を持たない。
  - どの子孫にも同じ要素名を持つ代入可能なグループがない。
- ワイルド・カード（XML Schema に基づく場合のみ）: ワイルド・カード（/*）に対するリライトは、次の条件が満たされる場合にサポートされます。
  - /* の後に、XPath の子または属性へのアクセスが 1 回以上行われる。
  - 孫の 1 つのみが、// に続く XPath の子または属性名と一致する。複数の孫が一致する可能性があることがスキーマに指定され、/* が拡張できる一意のパスが存在しない場合、リライトは行われない。
  - /* の前に指定されたノードの子または孫が xsi:anyType 型の要素を持たない。
  - /* の前に指定されたノードのどの子にも同じ要素名を持つ代入可能なグループがない。

XPath 式のリライト : 型および項目のマッピング

次の項では、XPath 式による型および次の項目のマッピングについて説明します。

- 「単純な XPath のマッピング」
- 「スカラー・ノードのマッピング」
- 「述語のマッピング」
- 「コレクション述語のマッピング」
- 「コレクション全検索でのドキュメント内の順序の保持」
- 「コレクション索引」
- 「条件を満たさない XPath 式」
- 「名前空間の処理」

- 「日付書式の変換」

単純な XPath のマッピング 単純な XPath のリライトは、XPath 式に対応する属性へのアクセスを伴います。表 5-11 に、XPath マッピングを示します。

表 5-11 XML Schema purchaseOrder の単純な XPath マッピング

XPath 式	マップ先
/PurchaseOrder	XMLData 列
/PurchaseOrder/@PurchaseDate	XMLData."PurchaseDate" 列
/PurchaseOrder/PONum	XMLData."PONum" 列
/PurchaseOrder/Item	コレクション XMLData."Item" の要素
/PurchaseOrder/Item/Part	コレクション XMLData."Item" の属性 "Part"

スカラー・ノードのマッピング XPath 式は、XML 文書内のスカラー・コンテンツにマップされる `text()` 演算子を含むことができます。リライトによって、この式は基礎となるリレーショナル列に直接マップされます。

たとえば、XPath 式 `/PurchaseOrder/PONum/text()` は、SQL 列 `XMLData."PONum"` に直接マップされます。

`PONum` 列の NULL 値は、入力ドキュメントに `text` ノードが存在しないか、または要素自体が不明であるため、テキスト値が使用できないことを意味します。これは、`SYS_XBDPD$` 属性で要素の有無を確認する必要がないため、スカラー要素にアクセスする方法より効率的です。

たとえば、XPath `/PurchaseOrder/PONum` も SQL 属性 `XMLData."PONum"` にマップされます。

ただし、この場合、クエリー・リライトで、`XMLData` 列の `SYS_XDBPD$` を使用して要素自体の有無を確認する必要があります。

述語のマッピング 述語は、SQL 述語式にマップされます。

例 5-23 述語のマッピング

次に、XPath 式の述語の例を示します。

```
/PurchaseOrder[PONum=1001 and Company = "Oracle Corp"]
```

この述語は、次の SQL 述語にマップされます。

```
( XMLData."PONum" = 20 and XMLData."Company" = "Oracle Corp")
```

たとえば、次の問合せが構造化された（オブジェクト・リレーショナル型の）問合せにリライトされます。XPath の機能上の評価は必要ありません。

```
select extract(value(p),'/PurchaseOrder/Item').getClobval()
   from mypos p
   where existsNode(value(p),'/PurchaseOrder[PONum=1001 and Company = "Oracle
Corp"]') =1;
```

コレクション述語のマッピング XPath 式は、コレクション式の関係演算子を伴う場合があります。XPath 1.0 では、コレクションに関連する条件として、要素の有無が確認されます。コレクションに 1 つでも条件を満たす要素が含まれる場合、式は TRUE になります。

例 5-24 コレクション述語のマッピング

次に、XPath のコレクション述語の例を示します。

```
/PurchaseOrder[Items/Price > 200]
-- maps to a SQL collection expression:
EXISTS ( SELECT null
    FROM   TABLE (XMLDATA."Item") x
    WHERE  x."Price" > 200 )
```

たとえば、次の問合せが構造化された問合せにリライトされます。

```
select extract(value(p),'/PurchaseOrder/Item').getClobval()
   from mypos p
   where existsNode(value(p),'/PurchaseOrder[Item/Price > 400]') = 1;
```

コレクション < 条件 > コレクションが存在する場合、より複雑なリライトが行われます。この場合、これらの 2 つのコレクション引数のうち 1 つ以上のノードの組合せが条件を満たすと、述語が条件を満たすと判断されます。

例 5-25 existsNode() を使用したコレクション述語のマッピング

Purchaseorder にいくつかの部品番号と価格が同じである Item が存在するかどうかを確認する、架空の XPath について考えてみます。

```
/PurchaseOrder[Items/Price = Items/Part]
-- maps to a SQL collection expression:
   EXISTS ( SELECT null
            FROM   TABLE (XMLDATA."Item") x
            WHERE  EXISTS ( SELECT null
                            FROM  TABLE(XMLDATA."Item") y
                            WHERE  y."Part" = x."Price"))
```

たとえば、次の問合せが構造化された問合せにリライトされます。

```
select extract(value(p),'/PurchaseOrder/Item').getClobval()
    from mypos p
    where existsNode(value(p),'/PurchaseOrder[Item/Price = Item/Part]') = 1;
```

コレクション全検索でのドキュメント内の順序の保持 ほとんどのリライトでは、元のドキュメント内の順序が保持されます。ただし、SQL システムでは副問合せの結果で順序が保持されることが保証されないため、extract() 関数を使用してコレクションから要素を選択すると、ノードの順序がドキュメント内の順序と異なる場合があります。

例 5-26 コレクション全検索でのドキュメント内の順序の保持

次に例を示します。

```
SELECT extract(value(p),'/PurchaseOrder/Item[Price>2100]/Part')
FROM mypos p;
```

この問合せは、次のとおり副問合せを使用するようにリライトされます。

```
SELECT (SELECT XMLAgg( XMLForest(x."Part" AS "Part"))
        FROM   TABLE (XMLData."Item") x
        WHERE  x."Price" > 2100 )
        FROM po_tab p;
```

多くの場合、集計操作の結果はコレクション要素と同じ順序になりますが、必ずしも、結果がドキュメント内の順序と同じになるとはかぎりません。この制限は、将来のリリースで修正される予定です。

コレクション索引 XPath 式でコレクションの特定の索引にアクセスすることもできます。たとえば、/PurchaseOrder/Item[1]/Part が、コレクションの最初の Item を抽出し、この Item 内の Part 属性にアクセスするようにリライトされます。

コレクションが VARRAY として格納されている場合、この操作によって元のドキュメント内の順序でノードが取得されます。コレクションがネストした表にマップされている場合、順序は認識されません。VARRAY が Ordered Collection Table（OCT）として格納されている場合（storeVarrayAsTable="true" が設定されている場合に、スキーマ・コンパイラによって作成されるデフォルト表）、このようなコレクション索引へのアクセスは、VARRAY に存在する IOT 索引を使用するように最適化されます。

条件を満たさない XPath 式 XPath 式には、入力ドキュメントに含めることができないノードに対する参照が含まれる場合があります。式のこのような部分は、リライト時に SQL の NULL にマップされます。たとえば、/PurchaseOrder/ShipAddress という XPath 式は、PO.xsd という XML Schema に準拠するインスタンス・ドキュメントの条件を満たすことができません。XML Schema で PurchaseOrder の ShipAddress 要素が許可されないためです。したがって、この式は SQL の NULL リテラルにマップされます。

名前空間の処理 名前空間は、関数ベースの評価と同じ方法で処理されます。XML Schema に基づく文書の場合、関数（existsNode()、extract() など）によって名前空間パラメータが指定されないと、スキーマのターゲットの名前空間が XPath 式のデフォルトの名前空間として使用されます。

例 5-27 名前空間の処理

たとえば、SQL 関数によって名前空間の接頭辞およびマッピングが明示的に指定されない場合、XPath 式 /PurchaseOrder/PONum は、xmlns:a="http://www.oracle.com/PO.xsd" で /a:PurchaseOrder/a:PONum として処理されます。別の例を示します。

```
SELECT * FROM po_tab p
   WHERE  EXISTSNODE(value(p), '/PurchaseOrder/PONum') = 1;
```

これは、次の問合せと同等の結果を戻します。

```
SELECT * FROM po_tab p
   WHERE  EXISTSNODE(value(p),'/PurchaseOrder/PONum',
       'xmlns="http://www.oracle.com/PO.xsd') = 1;
```

クエリー・リライトを実行すると、特定の要素に対する名前空間は XML Schema 定義の名前空間と一致します。XML Schema に elementFormDefault="qualified" が含まれる場合、XPath 式の各ノードは名前空間をターゲットとする必要があります（これは、デフォルトの名前空間の指定を使用するか、または各ノードに名前空間の接頭辞を付けることによって実行できます）。

elementFormDefault が非修飾（デフォルト）である場合、名前空間を定義するノードのみに接頭辞が含まれる必要があります。たとえば、PO.xsd の要素形式が非修飾である場合、existsNode() 関数を次のとおりリライトする必要があります。

```
EXISTSNODE(value(p),'/a:PurchaseOrder/PONum',
                    'xmlns:a="http://www.oracle.com/PO.xsd") = 1;
```

注意： elementFormDefault が非修飾である場合、前述の例の SQL 関数 existsNode() で名前空間パラメータを省略すると、各ノードがデフォルトでターゲットの名前空間に設定されます。これは、XML Schema 定義に一致しないために、結果が戻されない場合があります。これは、関数がリライトされるかどうかに関係ありません。

日付書式の変換 デフォルトの日付書式は、XML Schema と SQL で異なります。したがって、日付の比較を伴う XPath 式をリライトする場合、XML 形式を使用する必要があります。

例 5-28 日付書式の変換

次に例を示します。

```
[@PurchaseDate="2002-02-01"]
```

この式は、次のようにリライトすることはできません。

```
XMLData."PurchaseDate" = "2002-02-01"
```

これは、SQL のデフォルトの日付書式は YYYY-MM-DD でないためです。したがって、クエリー・リライト時に、XML 形式の文字列が追加されてテキスト値が DATE データ型に正しく変換されます。前述の述語は、次のとおりリライトされます。

```
XMLData."PurchaseDate" = TO_DATE("2002-02-01","SYYYY-MM-DD");
```

同様に、これらの列を（extract() などに必要な）テキスト値に変換する場合、XML 形式の文字列が追加されて XML と同じ日付書式に変換されます。

existsNode() での XPath 式のリライト

existsNode() は、XPath によってノード（text() または element ノード）が戻されるかどうかを示す数値である 0（ゼロ）または 1 を戻します。前述のマッピングに基づいて、XPath が text() ノードまたは non-scalar ノードをターゲットとする場合はスカラー要素が非 NULL であるかどうか、そうでない場合は SYS_XDBPD$ を使用する要素が存在するかどうかが existsNode() によって確認されます。SYS_XDBPD$ 属性が存在しない場合、スカラー列の NULL 情報によって scalar ノードが存在するかどうかが判断されます。

ドキュメント内の順序が保持される existsNode のマッピング 表 5-12 に、ドキュメント内の順序が保持される（SYS_XDBPD$ が存在し、スキーマ・ドキュメントで maintainDOM="true" が設定されている）場合の、existsNode() での様々な XPath マッピングを示します。

表 5-12 ドキュメント内の順序が保持される existsNode() の XPath マッピング

XPath 式	マップ先
/PurchaseOrder	CASE WHEN XMLData IS NOT NULL THEN 1 ELSE 0 END
/PurchaseOrder/@PurchaseDate	CASE WHEN Check_Node_Exists(XMLData.SYS_XDBPD$, 'PurchaseDate') = 1 THEN 1 ELSE 0 END
/PurchaseOrder/PONum	CASE WHEN Check_Node_Exists(XMLData.SYS_XDBPD$, 'PONum') = 1 THEN 1 ELSE 0 END
/PurchaseOrder[PONum = 2100]	CASE WHEN XMLData."PONum" = 2100 THEN 1 ELSE 0
/PurchaseOrder[PONum = 2100]/@PurchaseDate	CASE WHEN XML Data."PONum" = 2100 AND Check_Node_ Exists(XMLData.SYS_XDBPD$, 'PurchaseDate') = 1 THEN 1 ELSE 0 END
/PurchaseOrder/PONum/text()	CASE WHEN XMLData."PONum" IS NOT NULL THEN 1 ELSE 0
/PurchaseOrder/Item	CASE WHEN EXISTS (SELECT NULL FROM TABLE (XMLData."Item") x WHERE value(x) IS NOT NULL) THEN 1 ELSE 0 END
/PurchaseOrder/Item/Part	CASE WHEN EXISTS (SELECT NULL FROM TABLE (XMLData."Item") x WHERE Check_Node_Exists(x.SYS_XDBPD$, 'Part') = 1) THEN 1 ELSE 0 END
/PurchaseOrder/Item/Part/text()	CASE WHEN EXISTS (SELECT NULL FROM TABLE (XMLData."Item") x WHERE x."Part" IS NOT NULL) THEN 1 ELSE 0 END

例 5-29 ドキュメント内の順序が保持される existsNode のマッピング

前述のマッピングを使用して、2100 という番号の purchaseorder に価格が 2000 を超える部品が含まれるかどうかを確認する問合せを実行します。

```
SELECT count(*)
FROM   mypos p
WHERE EXISTSNODE(value(p),'/PurchaseOrder[PONum=1001 and Item/Price > 2000]')= 1;
```

これは、次のとおり再作成されます。

```
SELECT count(*)
FROM   mypos p
```

```
WHERE  CASE WHEN
           p.XMLData."PONum" = 1001 AND
           EXISTS ( SELECT NULL
                    FROM   TABLE ( XMLData."Item") p
                    WHERE  p."Price" > 2000 )) THEN 1 ELSE 0 END  = 1;
```

CASE 式は、さらに定数関係式によって最適化されます。この問合せは、次のとおり再作成されます。

```
SELECT count(*)
FROM   mypos p
WHERE  p.XMLData."PONum" = 1001 AND
       EXISTS ( SELECT NULL
                FROM   TABLE ( p.XMLData."Item") x
                WHERE  x."Price" > 2000 );
```

Part 列および PONum 列にリレーショナル索引が存在する場合、これを使用して評価が行われます。

ドキュメント内の順序が保持されない existsNode のマッピング SYS_XDBPD$ が存在しない（XML Schema で maintainDOM="false" が設定されている）場合、NULL スカラー列が実在しないスカラー要素にマップされます。そのため、SYS_XDBPD$ 属性を使用してノードの有無を確認する必要はありません。表 5-13 に、SYS_XDBPD$ 属性が存在しない場合の existsNode() のマッピングを示します。

表 5-13 ドキュメント内の順序が保持されない existsNode の XPath マッピング

XPath 式	マップ先
/PurchaseOrder	CASE WHEN XMLData IS NOT NULL THEN 1 ELSE 0 END
/PurchaseOrder/@PurchaseDate	CASE WHEN XMLData.'PurchaseDate' IS NOT NULL THEN 1 ELSE 0 END
/PurchaseOrder/PONum	CASE WHEN XMLData."PONum" IS NOT NULL THEN 1 ELSE 0 END
/PurchaseOrder[PONum = 2100]	CASE WHEN XMLData."PONum" = 2100 THEN 1 ELSE 0 END
/PurchaseOrder[PONum = 2100]/@PurchaseOrderDate	CASE WHEN XMLData."PONum" = 2100 AND XMLData."PurchaseDate" NOT NULL THEN 1 ELSE 0 END
/PurchaseOrder/PONum/text()	CASE WHEN XMLData."PONum" IS NOT NULL THEN 1 ELSE 0 END

表 5-13 ドキュメント内の順序が保持されない existsNode の XPath マッピング （続き）

XPath 式	マップ先
/PurchaseOrder/Item	CASE WHEN EXISTS (SELECT NULL FROM TABLE (XMLData."Item") x WHERE value(x) IS NOT NULL) THEN 1 ELSE 0 END
/PurchaseOrder/Item/Part	CASE WHEN EXISTS (SELECT NULL FROM TABLE (XMLData."Item") x WHERE x."Part" IS NOT NULL) THEN 1 ELSE 0 END
/PurchaseOrder/Item/Part/text()	CASE WHEN EXISTS (SELECT NULL FROM TABLE (XMLData."Item") x WHERE x."Part" IS NOT NULL) THEN 1 ELSE 0 END

extractValue() のリライト

`extractValue()` を使用すると、`extract()` を使用した `text` ノードおよび属性の抽出、および `getStringVal()` または `getNumberVal()` を使用したスカラー・コンテンツの取得を簡単に行うことができます。`extractValue` によって、スカラー要素の `text` ノードまたは `attribute` ノードの値が戻されます。`extractValue()` では、複数の値または非スカラー要素を戻すことはできません。

表 5-14 に、extractValue() が実行された場合の様々な XPath 式のマッピングを示します。XPath 式が要素をターゲットとする場合、`extractValue` によって要素の `text` ノードの子が取得されます。そのため、`/PurchaseOrder/PONum` および `/PurchaseOrder/PONum/text()` という 2 つの XPath 式は `extractValue` によって同じ方法で処理され、両方によって `PONum` のスカラー・コンテンツが取得されます。

表 5-14 extractValue() の XPath マッピング

XPath 式	マップ先
/PurchaseOrder	非サポート - extractValue は、スカラーの要素および属性の値のみ取り出すことができます。
/PurchaseOrder/@PurchaseDate	XMLData."PurchaseDate"
/PurchaseOrder/PONum	XMLData."PONum"
/PurchaseOrder[PONum = 2100]	(SELECT TO_XML(x.XMLData) FROM Dual WHERE x."PONum" = 2100)
/PurchaseOrder[PONum = 2100]/@PurchaseDate	(SELECT x.XMLData."PurchaseDate") FROM Dual WHERE x."PONum" = 2100)
/PurchaseOrder/PONum/text()	XMLData."PONum"

表 5-14 extractValue() の XPath マッピング （続き）

XPath 式	マップ先
/PurchaseOrder/Item	非サポート - extractValue は、スカラーの要素および属性の値のみ取り出すことができます。
/PurchaseOrder/Item/Part	非サポート - extractValue は、複数のスカラー値を取り出すことができません。
/PurchaseOrder/Item/Part/text()	非サポート - extractValue は、複数のスカラー値を取り出すことができません。

例 5-30 extractValue() のリライト

たとえば、次のような SQL 問合せを実行するとします。

```
SELECT ExtractValue(value(p),'/PurchaseOrder/PONum')
   FROM   mypos p
   WHERE  ExtractValue(value(p),'/PurchaseOrder/PONum') = 1001;
```

これは、次のとおりリライトされます。

```
SELECT p.XMLData."PONum"
   FROM   mypos p
   WHERE  p.XMLData."PONum" = 1001;
```

これは単純なスカラー列にリライトされるため、PONum 属性に対する索引（存在する場合）が、問合せの条件を満たすために使用される場合があります。

索引の作成 extractValue は、索引式で使用できます。式がスカラー列にリライトされると、索引はファンクション索引ではなく B* ツリー索引に変換されます。

例 5-31 extract での索引の作成

次に例を示します。

```
create index my_po_index on mypos x
  (Extract(value(x),'/PurchaseOrder/PONum/text()').getnumberval());
```

これは、次のとおりリライトされます。

```
create index my_po_index on mypos x ( x.XMLData."PONum");
```

次に、通常の B* ツリー索引に変換されます。これは、ファンクション索引とは異なり、同じ索引で次のような列をターゲットとする問合せの条件を満たすことができるため有効です。

```
EXISTSNODE(value(x),'/PurchaseOrder[PONum=1001]') = 1;
```

extract() のリライト

extract() では、XPath の結果が XML で取得されます。extract() は、text ノードを伴う XPath 式の場合の extractValue() と同様の方法でリライトされます。

ドキュメント内の順序が保持される extract のマッピング 表 5-15 に、ドキュメント内の順序が保持される（SYS_XDBPD$ が存在し、スキーマ・ドキュメントで maintainDOM="true" が設定されている）場合の、extract() での様々な XPath マッピングを示します。

注意： 次の例で、XMLElement() および XMLForest() に空の別名文字列 "" が付いている場合、text 値のみを使用して XML インスタンスを作成することを示します。これは単なる説明用です。

表 5-15 ドキュメント内の順序が保持される extract() の XPath マッピング

XPath	マップ先
/PurchaseOrder	XMLForest(XMLData as "PurchaseOrder")
/PurchaseOrder/@PurchaseDate	CASE WHEN Check_Node_Exists(XMLData.SYS_XDBPD$, 'PurchaseDate') = 1 THEN XMLElement("" , XMLData."PurchaseDate") ELSE NULL END
/PurchaseOrder/PONum	CASE WHEN Check_Node_Exists(XMLData.SYS_XDBPD$, 'PONum') = 1 THEN XMLElement("PONum" , XMLData."PONum") ELSE NULL END
/PurchaseOrder[PONum = 2100]	(SELECT XMLForest(XMLData as "PurchaseOrder") from Dual where x."PONum" = 2100)
/PurchaseOrder[PONum = 2100]/@PurchaseDate	(SELECT CASE WHEN Check_Node_Exists(x.XMLData.SYS_XDBPD$,'PurchaseDate") = 1 THEN XMLElement("", XMLData."PurchaseDate") ELSE NULL END from Dual where x."PONum" = 2100)
/PurchaseOrder/PONum/text()	XMLElement("", XMLData.PONum)

表 5-15 ドキュメント内の順序が保持される extract() の XPath マッピング （続き）

XPath	マップ先
/PurchaseOrder/Item	(SELECT XMLAgg(XMLForest(value(p) as "Item")) from TABLE (x.XMLData."Item") p where value(p) IS NOT NULL)
/PurchaseOrder/Item/Part	(SELECT XMLAgg(CASE WHEN Check_Node_Exists(p.SYS_XDBPD$,'Part") = 1 THEN XMLForest(p."Part" as "Part") ELSE NULL END) from TABLE (x.XMLData."Item") p)
/PurchaseOrder/Item/Part/text()	(SELECT XMLAgg(XMLElement(" ", p."Part")) from TABLE (x.XMLData."Item") x)

例 5-32 ドキュメント内の順序が保持される extract() の XPath マッピング

表 5-15 に示すマッピングを使用して、purchaseorder に価格が 2000 を超える部品が含まれる PONum 要素を抽出する次の問合せを実行します。

```
SELECT Extract(value(p),'/PurchaseOrder[Item/Part > 2000]/PONum')
FROM   po_tab p;
```

これは、次のとおりリライトされます。

```
SELECT (SELECT CASE WHEN Check_Node_Exists(p.XMLData.SYS_XDBPD$, 'PONum')  = 1
                    THEN XMLElement("PONum", p.XMLData."PONum")
                    ELSE NULL END)
        FROM  DUAL
        WHERE  EXISTS( SELECT NULL
                       FROM   TABLE ( XMLData."Item") p
                       WHERE  p."Part" > 2000)
       )
FROM po_tab p;
```

`Check_Node_Exists` は、単なる説明用の内部機能です。

ドキュメント内の順序が保持されない extract のマッピング `SYS_XDBPD$` が存在しない（XML Schema で `maintainDOM="false"` が設定されている）場合、NULL スカラー列が実在しないスカラー要素にマップされます。そのため、`SYS_XDBPD$` 属性を使用してノードの有無を確認する必要はありません。表 5-16 に、`SYS_XDBPD$` 属性が存在しない場合の `existsNode()` のマッピングを示します。

表 5-16 ドキュメント内の順序が保持されない extract() の XPath マッピング

XPath	マップ先
/PurchaseOrder	XMLForest(XMLData AS "PurchaseOrder")
/PurchaseOrder/@PurchaseDate	XMLForest(XMLData."PurchaseDate" AS "")
/PurchaseOrder/PONum	XMLForest(XMLData."PONum" AS "PONum")
/PurchaseOrder[PONum = 2100]	(SELECT XMLForest(XMLData AS "PurchaseOrder") from Dual where x."PONum" = 2100)
/PurchaseOrder[PONum = 2100]/@PurchaseDate	(SELECT XMLForest(XMLData."PurchaseDate" AS "") from Dual where x."PONum" = 2100)
/PurchaseOrder/PONum/text()	XMLForest(XMLData.PONum AS "")
/PurchaseOrder/Item	(SELECT XMLAgg(XMLForest(value(p) as "Item") from TABLE (x.XMLData."Item") p where value(p) IS NOT NULL)
/PurchaseOrder/Item/Part	(SELECT XMLAgg(XMLForest(p."Part" AS "Part") from TABLE (x.XMLData."Item") p)
/PurchaseOrder/Item/Part/text()	(SELECT XMLAgg(XMLForest(p. "Part" AS "Part")) from TABLE (x.XMLData."Item") p)

updateXML() を使用した更新の最適化

`updateXML()` を使用する通常の更新では、XML 文書の値が更新され、次に文書全体が新しく更新された文書で置換されます。

`XMLType` がオブジェクト・リレーショナル形式で格納されている場合、XML Schema マッピングを使用して、文書のフラグメントが直接更新されるように最適化されます。たとえば、`PONum` 要素の値を更新する場合、メモリー内で文書全体を生成した後に更新を実行するのではなく、`XMLData.PONum` 列が直接更新されるようにリライトできます。

updateXML() で最適化を使用するには、次の条件を満たす必要があります。

- updateXML() に指定された XMLType 列は、SET 句で更新される列と同じである必要があります。次に例を示します。

```
UPDATE po_tab p SET value(p) = updatexml(value(p),...);
```

- XMLType 列は、Oracle XML DB の XML Schema マッピングを使用してオブジェクト・リレーショナル形式で格納されている必要があります。
- XPath 式が述語またはコレクション全検索を伴うことはできません。
- 複製スカラー式を含めることはできません。
- updateXML() 関数のすべての XPath 引数は、スカラー・コンテンツ（テキスト・ノードまたは属性）のみをターゲットとする必要があります。次に例を示します。

```
UPDATE po_tab p SET value(p) =
    updatexml(value(p),'/PurchaseOrder/@PurchaseDate','2002-01-02',
                       '/PurchaseOrder/PONum/text()', 2200);
```

前述のすべての条件が満たされる場合、updateXML は単純なリレーショナル更新にリライトされます。次に例を示します。

```
UPDATE po_tab p SET value(p) =
       updatexml(value(p),'/PurchaseOrder/@PurchaseDate','2002-01-02',
                          '/PurchaseOrder/PONum/text()', 2200);
```

これは、次のとおりリライトされます。

```
UPDATE po_tab p
  SET p.XMLData."PurchaseDate" = TO_DATE('2002-01-02','SYYYY-MM-DD'),
      p.XMLData."PONum" = 2100;
```

DATE の変換 DATE、gMONTH、gDATE などの DATE データ型の書式は、XML Schema と SQL で異なります。この場合、updateXML() にこれらの列に対する文字列値が含まれると、リライトによって XML 形式の文字列が自動で挿入され、文字列値が正しく変換されます。そのため、DATE 列に指定された文字列値は、XML の日付書式に一致し、SQL の日付書式には一致しません。

XML Schema の登録時のデフォルト表の作成

XML Schema の登録の一部として、デフォルト表を作成することもできます。デフォルト表は、この XML Schema に準拠する XML インスタンス・ドキュメントが、FTP や HTTP など、表が指定されない API を介して挿入される場合に最適です。この場合、XML インスタンスはデフォルト表に挿入されます。

defaultTable 属性に値を指定した場合、その名前で XMLType 表が作成されます。それ以外の場合は、内部生成された名前で作成されます。

また、tableProps および columnProps 属性を使用して指定されたすべてのテキストが、生成された CREATE TABLE 文に追加されます。

Ordered Collections in Tables（OCT）

通常、XML Schema での配列（maxOccurs > 1 の要素）は、VARRAY に格納されます。これは、ネストした表と同様に、ラージ・オブジェクト（LOB）または個別の格納表のいずれかに格納できます。

> **注意：** VARRAY の要素が個別の表に格納されている場合、VARRAY は Ordered Collection in Tables（OCT）と呼ばれます。次の項で使用する OCT は、格納表に索引構成表（IOT）格納が使用されていることを前提としています。

これによって、IOT に基づいて個別の表に VARRAY の要素を含めることができます。表の主キーは（NESTED_TABLE_ID, ARRAY_INDEX）です。NESTED_TABLE_ID は、要素とその要素に含まれる親をリンクするために使用されます。ARRAY_INDEX 列は、コレクション内で要素の場所を追跡するために使用されます。

OCT を使用した VARRAY の格納

OCT 格納を指定するには、2 つの方法があります。

- スキーマ属性 storeVarrayAsTable を使用する方法。デフォルトでは「false」で、VARRAY は LOB に格納されます。これが「true」に設定されている場合、すべての VARRAY（maxOccurs > 1 であるすべての要素）が OCT として格納されます。
- tableProps 属性を使用して格納方法を明示的に指定する方法。tableProps 属性の一部として、OCT を作成するために必要な SQL を使用できます。

```
"VARRAY xmldata.<array> STORE AS TABLE <myTable> ((PRIMARY KEY (NESTED_TABLE_ID,
ARRAY_INDEX)) ORGANIZATION INDEX)"
```

VARRAY の格納に OCT を使用するメリットによって、要素へのアクセスが高速化され、問合せ精度が改善されます。要素の属性に対して索引を作成することもできます。これによって、クエリー・リライトの実行が改善されます。

XML Schema 間での循環参照

XML Schema 文書間に循環型の依存性を設定できます。これによって、通常の方法で逐次登録が行われないようにできます。このような XML Schema の例を次に示します。

例 5-33 循環型の依存性

他の XML Schema を含む XML Schema は、含まれる XML Schema が存在しない場合は作成できません。

```
begin dbms_xmlschema.registerSchema('xm40.xsd',
'<schema xmlns="http://www.w3.org/2001/XMLSchema" xmlns:my="xm40"
targetNamespace="xm40">
  <include schemaLocation="xm40a.xsd"/>
  <!-- Define a global complextype here -->
  <complexType name="Company">
    <sequence>
      <element name="Name" type="string"/>
      <element name="Address" type="string"/>
    </sequence>
  </complexType>
  <!-- Define a global element depending on included schema -->
  <element name="Emp" type="my:Employee"/>
</schema>',
true, true, false, true); end;
/
```

ただし、FORCE オプションを使用すると作成できます。

```
begin dbms_xmlschema.registerSchema('xm40.xsd',

'<schema xmlns="http://www.w3.org/2001/XMLSchema" xmlns:my="xm40"
targetNamespace="xm40">
  <include schemaLocation="xm40a.xsd"/>
  <!-- Define a global complextype here -->
  <complexType name="Company">
    <sequence>
      <element name="Name" type="string"/>
      <element name="Address" type="string"/>
    </sequence>
  </complexType>
  <!-- Define a global element depending on included schema -->
  <element name="Emp" type="my:Employee"/>
</schema>',
true, true, false, true, true); end;
/
```

このスキーマを使用して、再コンパイルを実行しようとすると失敗します。

```
create table foo of sys.xmltype xmlschema "xm40.xsd" element "Emp";
```

FORCE オプションを使用して、2 つ目の XML Schema を作成します。これによって、最初の XML Schema も有効になります。

```
begin dbms_xmlschema.registerSchema('xm40a.xsd',
'<schema xmlns="http://www.w3.org/2001/XMLSchema" xmlns:my="xm40"
targetNamespace="xm40">
  <include schemaLocation="xm40.xsd"/>
  <!-- Define a global complextype here -->
  <complexType name="Employee">
    <sequence>
      <element name="Name" type="string"/>
      <element name="Age" type="positiveInteger"/>
      <element name="Phone" type="string"/>
    </sequence>
  </complexType>
  <!-- Define a global element depending on included schema -->
  <element name="Comp" type="my:Company"/>
</schema>',
true, true, false, true, true); end;
/
```

両方の XML Schema を使用して、表などを作成できます。

```
create table foo of sys.xmltype xmlschema "xm40.xsd" element "Emp";
create table foo2 of sys.xmltype xmlschema "xm40a.xsd" element "Comp";
```

相互に循環する依存性を持つこれらの 2 つの XML Schema を登録するには、次のとおり DBMS_XMLSCHEMA.registerSchema で FORCE パラメータを使用する必要があります。

1. FORCE モードで s1.xsd を登録します。

   ```
   dbms_xmlschema.registerSchema("s1.xsd", "<schema ...", ..., force => true)
   ```

   この時点では、s1.xsd は無効で使用できません。

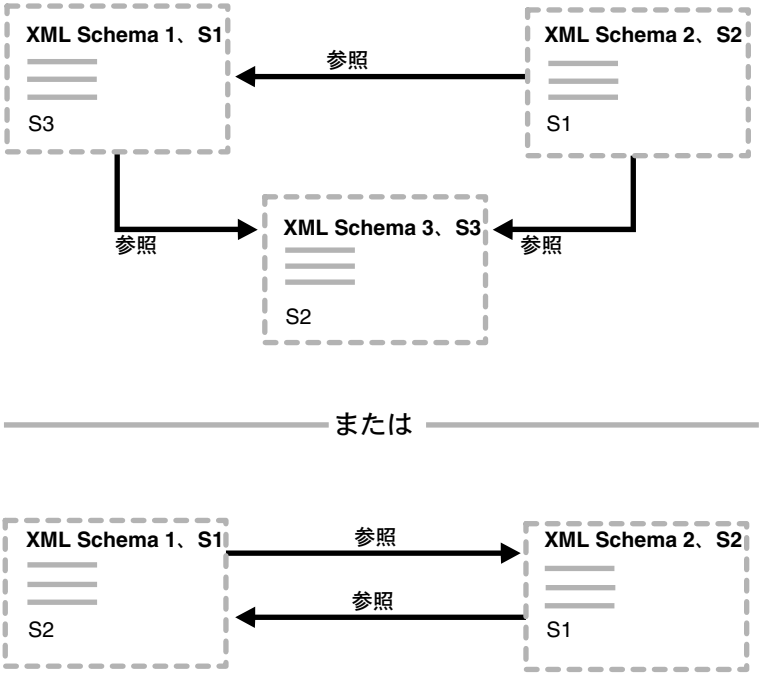
2. FORCE モードで s2.xsd を登録します。

   ```
   dbms_xmlschema.registerSchema("s2.xsd", "<schema ..", ..., force => true)
   ```

   これによって s1.xsd が自動的にコンパイルされ、両方の XML Schema が有効になります。

図 5-8 を参照してください。前述の例を、図の下半分に示します。

図 5-8 XML Schema 間での循環参照



FAQ: XML DB および XML Schema に基づく問題

XPath 式にバインド変数を使用するとメモリー・リークが発生するのはなぜですか？

次の単純な XML 文書があります。サイズは 3.6MB です。

```
<?xml version="1.0"?>
<PurchaseOrder xmlns="http://www.vector.com/po.xsd"
xmlns:xdb="http://xmlns.oracle.com/xdb"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.vector.com/po.xsd
http://www.vector.com/po.xsd">
 <PONum>1001</PONum>
   <Company>Oracle Corp</Company>
     <Item>
      <Part>9i Doc Set</Part>
```

```
      <Price>2550</Price>
    </Item>
    <Item>
      <Part>8i Doc Set</Part>
      <Price>350</Price>
    </Item>
    <Item>
      <Part>7i Doc Set</Part>
      <Price>50</Price>
    </Item>
 </PurchaseOrder>
```

このドキュメントをオブジェクト・リレーショナル形式で XMLType 表に格納します。XML Schema には注釈を付けていません。

```
 <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <xsd:complexType name="PurchaseOrderType">
   <xsd:sequence>
    <xsd:element name="PONum" type="xsd:decimal"/>
    <xsd:element name="Company">
     <xsd:simpleType>
      <xsd:restriction base="xsd:string">
       <xsd:maxLength value="100"/>
      </xsd:restriction>
     </xsd:simpleType>
    </xsd:element>
   <xsd:element name="Item" maxOccurs="2147483647">
     <xsd:complexType>
      <xsd:sequence>
...

Name                                      Null?    Type
----------------------------------------- -------- -------------------
FILENAME                                  NOT NULL VARCHAR2(20)
CONTENT                                   NOT NULL
XMLTYPE(XMLSchema "http://www.vector.com/po.xsd"
Element "PurchaseOrder") STORAGE Object-relational TYPE
 "PurchaseOrderType1627_T"
```

次の文を実行しました。

```
SQL> select existsnode(srp.content,'/PurchaseOrder/Item[Part="7i Doc Set"]')
     into :i from xmltable srp where filename='po6.xml';
```

この文を実行するには、ノートブック型コンピュータで約 6 秒かかりました。次のようなバインド変数を使用する文を実行しました。

```
SQL> var xpath varchar2(50)
```

```
SQL> exec :xpath:='/PurchaseOrder/Item[Part="7i Doc Set"]'

PL/SQL procedure successfully completed.
SQL> select existsnode(srp.content,:xpath) into :i from xmltable srp
     where filename='po6.xml';
```

完了するまで待ちましたが、ハングアップしました。CPU の使用率は 100%で、大量のメモリーが消費されています。

回答 : バインド変数を使用する場合、Oracle によるクエリー・リライトが行われません。最初の例ではリレーショナル・リライトが行われ、後の例では完全なファンクション・ベースの XPath が使用されています。

質問 2: SQL を共有するためにバインド変数を使用する必要があります。CURSOR_SHARING を FORCE に設定すると、どうなりますか ?

回答 2: 基本的に、クエリー・リライトとは、入力された XPath 式が Oracle によって基礎となる列に変更されることを意味します。つまり、指定された XPath には、内部的に参照される列や表などの特定のセットが存在します。参照する表や列などが共有カーソルによって正確に認識される必要があるため、Oracle による XPath 式の変更はコンパイル時の操作である必要があります。各行、またはカーソルの各インスタンスに対して異なる参照先を使用することはできません。

XPath 式自体がバインド変数である場合、カーソルの各インスタンスには完全に異なる XPath が使用される場合があるため、Oracle はリライトを行うことができません。これは、SQL 問合せで表と列の名前をバインドする場合に類似しています。たとえば、SELECT * FROM 表 (:1) などです。

注意 : 問合せの右側にバインド変数を指定すると、この問合せは適切に動作します。次に例を示します。

```
SELECT * FROM purchaseorder p WHERE
extractvalue(value(p),'/PurchaseOrder/LineItems/LineIte
m/ItemNumber') = :1;
```

この場合、Oracle での通常のバインド変数の共有が使用されます。

CURSOR_SHARING を FORCE に設定すると、デフォルトで、XPath を含むすべての文字列定数がバインド変数になります。このとき、extractvalue()、existsnode() などが使用されていると、Oracle によって XPath バインド変数が定数であるかどうかが確認されます。定数である場合、それらの定数を使用してクエリー・リライトが行われます。

このように、バインド変数の動作には使用方法によって大きな相違があります。

クエリー・リライトが正常に動作していることを確認する方法は？

XML Schema に基づくオブジェクト・リレーショナル記憶域でのクエリー・リライトについて質問があります。どのようにしてクエリー・リライトが正常に動作していることを確認できますか？SQL トレース、イベントなどを使用する必要がありますか？

回答 : クエリー・リライトの確認を行うには、2 つの方法があります。

- 実行計画を使用する方法 : この方法を使用すると、クエリー・リライトを正しく使用するための、索引の使用方法などが表示されます。
- イベントを使用する方法 : 次に例を示します。

```
Event 19027 - turns off query rewrite - no level information needed
Event 19021 - XML operations - general event. Use this with different levels
to get different behavior..
Level 0x1 - Turn off all functional evaluation..
Level 0x2 - Turn off functional evaluation of EXTRACT
Level 0x4 - Turn off functional evaluation of EXISTSNODE
Level 0x8 - Turn off functional evaluation of TRANSFORM
Level 0x10 - Turn off functional evaluation of EXTRACTVALUE
Level 0x20 - Turn off functional evaluation of UPDATEXML
```

2 番目の 19021 というイベントを使用すると、オプションで、これらの演算子の評価が選択された場合にエラーが発生するようにすることができます。次に例を示します。

```
ALTER SESSION SET EVENTS  '19021 trace name context forever, level 1';
```

この場合、前述のすべての XML 演算子の評価が無効になります。次のような問合せを発行するとします。

```
SELECT extract(value(x),'/purchaseorder/reference')
FROM purchaseorder_xml_tab
```

クエリー・リライトが行われない場合、extract() を実行すると、「ORA-19022: XML XPath 機能は使用できません」というエラーが発生します。

質問 2: アドバイスにしたがって、19021 というイベントを使用しました。使用したテストを次に示します。

```
1--set event
SQL> alter session set events '19021 trace name context forever, level 2';
Session altered.
2--extract function used
```

XML Schema に基づく場合および XML Schema に基づかない場合の両方でこのテストを実行しました。

XML Schema に基づくオブジェクト・リレーショナル記憶域の場合を示します。

```
SQL> SELECT value(x).extract('/a:PO/Company',
```

```
  2  'xmlns:a="http://www.oracle.com/PO"')
  3  FROM po_tab x;
```

次のエラーが発生しました。

```
ERROR:
ORA-19022: XML XPath機能は使用できません
ORA-06512: "SYS.XMLTYPE"、行0
ORA-06512: 行1
```

XML Schema に基づかない CLOB 記憶域の場合を示します。

```
SQL> SELECT extract(value(p),'/PO/PODATE')
  2  FROM po_tab p;
```

次のエラーが発生しました。

```
ERROR:
ORA-19022: XML XPath機能は使用できません
```

この結果は、クエリー・リライトが行われていないことを示しています。他にクエリー・リライトの確認を行う方法がありますか？

回答 2: この操作は間違っていません。イベント 19021 を設定すると、評価が無効になり、XMLType のすべての機能が無効になります。そのため、クエリー・リライトが行われない場合、「ORA-19022: XML XPath 機能は使用できません」というエラーが発生します。

2 番目の例（XML Schema に基づかない場合）が正常に動作しない理由は、オブジェクトに対して定義された、XML Schema に基づかない（NSB）XMLType ビュー（XV）に対してのみクエリー・リライトが行われるためです。XML Schema に基づかない XMLType 表では、格納が CLOB ベースであるため、クエリー・リライトは行われません。

最初の例（XML Schema に基づく場合）が正常に動作しない理由は、名前空間パラメータに原因がある可能性があります。

現在、XMLType メソッドである extract() または existsNode() に対して、クエリー・リライトは機能しません。ただし、XMLType メソッドのかわりに同等の演算子を使用できます。たとえば、xmltype.extract() メソッドのかわりに extract() 演算子を使用します。

作成した索引が XML DB の問合せで使用されないのはなぜですか？

他のすべてのスクリプトの実行を制御するデモ・スクリプトを実行しました。これによって、次のような索引が作成されました。

```
create index director_name on movies( extractValue(movieDoc,'/Movie/Director/Last')
);
```

次の問合せに対して実行計画を試行します。

```
SELECT extractValue(movieDoc,'/Movie/@Title')
FROM movies
 WHERE extractValue(movieDoc,'/Movie/Director/Last') = 'Minghella'
```

この場合、作成した索引が使用されません。Oracle9*i* JDeveloper の SQL ワークシートの実行計画を次に示します。

```
SELECT STATEMENT
  - Filter
    - Table Access (FULL) SCOTT.MOVIES
    - Collection Iterator (PICKLER FETCH)
```

これは、表に含まれる項目が少なく、オプティマイザによって、表の全表スキャンが最も高速な方法であると判断されたためでしょうか？また、次のとおり実行計画を試行しました。

```
SELECT /*+ INDEX(movies director_name) */
extractValue(movieDoc,'/Movie/@Title')
  FROMmovies
 WHERE extractValue(movieDoc,'/Movie/Director/Last') = 'Minghella'
```

この場合も、MOVIES の全表スキャンが行われます。

回答 : XMLType に対して XML Schema に基づかない索引を作成すると、この索引はファンクション索引になります。user_functional_indexes を確認してください。ファンクション索引の場合、文字列が完全に一致する必要があり、次のとおり ALTER SESSION を使用する必要があります。

```
ALTER SESSION SET query_rewrite_enabled=true
ALTER SESSION SET  query_Rewrite_integrity=trusted
```

これによって、索引が検出されます。

complexType の XML Schema の宣言で属性を指定する方法は？

回答 : グローバルな complexType に基づく要素が存在する場合、complexType 宣言に SQLType（および SQLSchema）属性を指定する必要があります。また、要素の宣言内に同じ SQLType および SQLSchema 属性を含めることもできます。

このように指定する理由は、グローバルな complexType に SQLType を指定しない場合、内部生成された名前を持つ SQLType が XML DB によって作成されるためです。このグローバルな型を参照する要素では、SQLType に異なる値を指定できません。つまり、次のとおり指定すると、正常に動作します。

```
<xsd:complexType name="PURCHASEORDERLINEITEM_TYPEType">
 <xsd:sequence>
  <xsd:element name="@LineNo" type="xsd:double" xdb:SQLName="@LineNo"
        xdb:SQLType="NUMBER"/>
  <xsd:element name="Decription" type="xsd:string"xdb:SQLName="Decription"
```

```
        xdb:SQLType="VARCHAR2"/>
  <xsd:element name="Part" type="PURCHASEORDERPART_TYPEType"xdb:SQLName="Part" />
</xsd:sequence>
</xsd:complexType>
  <xsd:complexType name="PURCHASEORDERPART_TYPEType" xdb:SQLSchema="XMLUSER"
        xdb:SQLType="PURCHASEORDERPART_TYPE">
   <xsd:sequence>
     <xsd:element name="@Id" type="xsd:string"
         xdb:SQLName="@Id"xdb:SQLType="VARCHAR2"/>
       <xsd:element name="@Quantity" type="xsd:double"xdb:SQLName="@Quantity"
           xdb:SQLType="NUMBER"/>
         <xsd:element name="@cost" type="xsd:double"
            xdb:SQLName="@cost"xdb:SQLType="NUMBER"/>
</xsd:sequence>
</xsd:complexType>
```

次のとおり指定しても正常に動作します。

```
<xsd:complexType name="PURCHASEORDERLINEITEM_TYPEType">
  <xsd:sequence>
    <xsd:element name="@LineNo" type="xsd:double" xdb:SQLName="@LineNo"
xdb:SQLType="NUMBER"/>
    <xsd:element name="Decription" type="xsd:string"xdb:SQLName="Decription"
xdb:SQLType="VARCHAR2"/>
    <xsd:element name="Part" type="PURCHASEORDERPART_TYPEType"xdb:SQLName="Part"
xdb:SQLSchema="XMLUSER"
xdb:SQLType="PURCHASEORDERPART_TYPE" />
  </xsd:sequence>
</xsd:complexType>
```

XML Schema と要素が一致しないのはなぜですか？

次の表定義があります。

```
SQL> describe "rechnung";

Name                      Null?        Type
------------------------- -----        ---------------
ID                        NOT NULL     NUMBER(10)
rechnung

SYS.XMLTYPE(XMLSchema "http://cczarski.de.oracle.com/Rec
hnung/Test001.xsd"
Element "rechnung") STORAGE Object-relational TYPE "RECHNUNG_T"
DATUM DATE
```

また、次の XML Schema があります。

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://cczarski.de.oracle.com/Rechnung/Test001.xsd"
            xmlns:xdb="http://xmlns.oracle.com/xdb"
xmlns:rechn="http://cczarski.de.oracle.com/Rechnung/Test001.xsd"
            elementFormDefault="qualified"
            version="1.0">
<!-- Zundchst wird der Kunde definiert --
```

次のドキュメントを挿入しました。

```
<rechnung xmlns="http://.../Test001.xsd" xmlns:xsi="
http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://cczarski.d
e.oracle.com/Rechnung/Test001.xsd">
        <kunde>
...
```

これによって、次のエラーが報告されるのはなぜですか？

```
ERROR at line 2:
ORA-19007: スキーマと要素が一致しません
```

回答 : xsi:schemaLocation には 2 つのパラメータ（NS SchemaURL）を指定します。

次のとおり試行してください。

```
xsi:schemaLocation="http://cczarski.de.oracle.com/Rechnung/Test001.xsd
http://cczarski.de.oracle.com/Rechnung/Test001.xsd">
```

RESOURCE_VIEW [S/MIME] からスタイルシートを取り出す方法は？

次の文を使用して RESOURCE_VIEW から保存されたスタイルシートを取り出そうとすると、問題が発生します。

```
SELECT EXTRACT
  (rtab.res,
   'r:Resource/r:Contents/node()/xsl:stylesheet',
   'xmlns:r="http://xmlns.oracle.com/xdb/XDBResource.xsd" '||
   'xmlns:xdb="http://xmlns.oracle.com/xdb" ' ||
   'xmlns:xsl="http://www.w3.org/1999/XSL/Transform"'
  ).getclobval()
  FROM resource_view rtab
  WHERE rtab.any_path =
   '/public/spec_proto/XDB_Stylesheet_Render_XML.xsl'
/
```

名前空間の設定が間違っているのでしょうか？

回答 : XML Schema を登録していますか？ 現時点では、登録済の XML Schema に含まれるコンテンツでないかぎり、RESOURCE_VIEW からコンテンツを抽出することはできません。

XML Parser で xmlns 属性を追加すると、selectSingleNode によって NULL が戻されるのはなぜですか？

作成したコードによって、ApplicationStructure という独自の XML Schema のインスタンスである XML ファイルを解析します。この解析は、最上位のタグに xmlns="http://www.oracle.com/JHeadstart/ApplicationStructure" を追加すると、正常に動作しません。次に selectSingleNode をコールすると、NULL が戻されます。xmlns 属性を削除すると、selectSingleNode によって必要なノードが戻されるようになります。次に使用したコードを示します。

```
import oracle.xml.parser.v2.*;
...
private XMLDocument mXmldoc;
mXmldoc = XMLLoader.getXMLDocument(mSource);
// Select Service node
XMLNode serviceNode = (XMLNode)mXmldoc.selectSingleNode("Service");
```

何が間違っているのでしょうか？

selectSingleNode の別のコンストラクタが存在し、これによって 2 番目の NSResolver というパラメータが受け入れられているようです。このコンストラクタの正しい使用方法を教えてください。また、xmlns 属性の有無に関係なく、XML ファイルに対してこのコンストラクタを使用可能にすることはできますか？

xmlns 属性を使用可能にして、登録済の独自の XML Schema に対して Oracle9*i* JDeveloper のコード・インサイトを使用できるようにする必要があります。

回答 : XPath には、デフォルトの名前空間で修飾された要素を検索するための構文がありません。XPath パターンの「foo」では、常に、名前空間が NULL である <foo> という要素が検索されます。そのため、構文としては、デフォルトの名前空間を使用して次のように記述することが可能です。

```
<foo xmlns="urn:mynamespace"/>
```

ただし、XML Parser では、内部的に、要素の名前が <{urn:mynamespace}:foo> に設定されます。次のようには設定されません。

```
<foo>
```

すなわち、デフォルトの名前空間を含む名前空間 URI を持つ要素を検索するには、次の項目を使用します。

- <foo xmlns="urn:mynamespace"/> の名前空間接頭辞。

- 「someprefix:foo」などの XPath パターン。この場合、「someprefix」という接頭辞は「urn:mynamespace」という名前空間 URI にマップされているとします。

次に例を示します。

```
package test;
import oracle.xml.parser.v2.*;
import org.w3c.dom.*;
import java.io.*;
public class Demo {
  private static final String URI =
"http://www.oracle.com/JHeadstart/ApplicationStructure";
  private static final String TESTDOC =
  "<foo xmlns='"+URI+"'/>";
  private static final NSResolver nsr = new MyNSResolver();
  public static void main(String[] args) throws Throwable {
    System.out.println("Document to parse is");
    System.out.println(TESTDOC);
    DOMParser dp = new DOMParser();
    dp.parse( new StringReader(TESTDOC));
    XMLDocument doc = dp.getDocument();
    Node n = doc.selectSingleNode("xxx:foo", nsr); // Provide NSResolver!
    System.out.println( "Found " + ((n!=null) ? " it! " : " nothing"));
  }

  static class MyNSResolver implements NSResolver {
    public String resolveNamespacePrefix(String pref) {
      if (pref.equals("xxx")) return URI;
      else return null;
    }
  }
}
```

「ORA-19007: スキーマと要素が一致しません」というエラーが発生するのはなぜですか？

次のスクリプトは、表にサンプル・データを挿入すると正常に実行されません。表にサンプル・データを挿入すると、「ORA-19007: スキーマと要素が一致しません」というエラーが発生します。これはなぜですか？

```
---- testPo.sql
set serverout on

drop table po_tab1;

declare
    urlvar  varchar2(100);
```

```
    xsdfile varchar2(2000);
begin
    urlvar := 'http://www.oracle.com/PO.xsd';

   --         xmlns:po="http://www.oracle.com/PO.xsd">
    xsdfile :=
    '<schema xmlns="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://www.oracle.com/PO.xsd"
        xmlns:po="http://www.oracle.com/PO.xsd">
        <complexType name="PurchaseOrderType">
          <sequence>
            <element name="PONum" type="decimal"/>
            <element name="Company" type="string"/>
            <element name="Item" maxOccurs="1000">
              <complexType>
                <sequence>
                  <element name="Part" type="string"/>
                  <element name="Price" type="decimal"/>
                </sequence>
              </complexType>
            </element>
          </sequence>
          <attribute name = "PurchaseDate" type = "date"/>
        </complexType>
        <element name="PurchaseOrder" type = "po:PurchaseOrderType"/>
      </schema>';

      begin

dbms_xmlschema.deleteschema(urlvar,dbms_xmlschema.delete_cascade_force);
      exception
         when others then null;
      end;

      dbms_xmlschema.registerschema(urlvar,xsdfile);
end;
/

set heading off
set pagesize 0
set long 10000
set maxdata 12000
set arraysize 1

select a.schema.getstringval() from user_xml_schemas a
where a.schema_url = 'http://www.oracle.com/PO.xsd';
```

```
CREATE TABLE po_tab1 OF XMLTYPE ELEMENT
"http://www.oracle.com/PO.xsd#PurchaseOrder";

insert into po_tab1 values (xmltype('
   <PurchaseOrder xmlns="http://www.oracle.com/PO.xsd"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.oracle.com/PO.xsd"
     PurchaseDate="1967-08-13">
   <PONum>1</PONum>
   <Company>The Business</Company>
   <Item>
    <Part>Part 1</Part>
    <Price>1000</Price>
   </Item>
</PurchaseOrder>'));

select * from po_tab1;
```

回答 : schemaLocation 属性は、<namespace> 値と <schemaloc> 値の組合せである必要があります。次に例を示します。

```
xsi:schemaLocation="http://www.oracle.com/PO.xsd
http://www.oracle.com/PO.xsd"
```

スキーマ用の XML Schema を登録することはできますか ?

類似したスキーマ用の XML Schema を登録しようとしています。XMLSchema.xsd、XMLSchema.dtd および datatypes.dtd をダウンロードし、XMLSchema.xsd の検証が可能であることを確認した後で、ローカルの Web サーバーでこれらを使用可能にしました。

```
begin
  dbms_xmlschema.registeruri(    schemaURL =>
'http://www.denmark.dk/MD/XMLSchema'
                , schemaDocUri => 'http://144.21.226.78/XMLSchema.xsd'
    , local => false
                );
end;
 /
declare
*
ERROR at line 1:
ORA-31011: XML 解析に失敗しました
ORA-19202: XML 処理中にエラーが発生しました
WPTG-00233: "xml" で始まる名前空間の接頭辞は予約済です。
Error at line 70
ORA-06512: "XDB.DBMS_XMLSCHEMA_INT"、行 0
ORA-06512: "XDB.DBMS_XMLSCHEMA"、行 160
ORA-06512: 行 34
```

すべての XML Schema を格納するための表が必要です。この表のコンテンツをスキーマ用の XML Schema に対して検証し、後でより複雑な分析を行うために、オブジェクト・リレーショナル構造内を検索する組込み機能が必要です。スキーマ用の XML Schema を登録できますか？

元の `XMLSchema.xsd` では、XML 名前空間が宣言されないため、XMLSpy での検証を行うことができませんでした。XMLSpy での検証を正常に行うために `xmlns:xml` という名前空間宣言を追加しましたが、エラーが発生します。

回答：「xml」で始まる名前空間を設定することはできません。「foo」、「xsd」、「xs」、「x」などの名前空間接頭辞を使用してください。

6

XMLType データの変換および検証

この章では、XSLT スタイルシートを使用して XMLType データを変換するための、SQL 関数および XMLType API について説明します。また、XML Schema に対して XMLType を検証するために使用可能な様々な関数および API についても説明します。この章の内容は次のとおりです。

XMLType インスタンスの変換

XML 文書には、構造はありますがフォーマットはありません。XML 文書にフォーマットを追加するには、eXtensible Stylesheet Language（XSL）を使用できます。XSL は、XML セマンティクスを表示する方法を提供します。XSLT を使用すると、XML 要素を HTML などの他のフォーマット言語またはマークアップ言語にマップできます。

Oracle XML DB では、Oracle9*i* データベースの XMLType 表、列またはビューに格納されている XMLType インスタンスまたは XML データは、XSLT スタイルシートおよび XMLType の関数である transform() を使用して、HTML、XML およびその他のマークアップ言語に（フォーマット）変換できます。このプロセスは、W3C の XSLT 1.0 勧告に準拠しています。

XMLType インスタンスは、次の方法で変換できます。

- データベースで SQL 関数 XMLTransform()（または XMLType の transform() メンバー関数）を使用する。
- XSLT Processor for Java などの中間層で XDK 変換オプションを使用する。

> **参照：** 次の項、付録およびマニュアルを参照してください。
>
> - 26-81 ページの「8.3 XSLT を使用した PurchaseOrder の変換」
> - 付録 D「XSLT の手引き」
> - 『Oracle9*i* XML Developer's Kit ガイド - XDK』の「XSQL Pages パブリッシング・フレームワーク」

XMLTransform() および XMLType.transform()

図 6-1 に、XMLTransform() 構文を示します。XMLTransform() 関数は、引数として XMLType インスタンスおよび XSLT スタイルシート（それ自体が XMLType インスタンス）を取ります。この関数では、スタイルシートがインスタンスに適用され、XMLType インスタンスが戻されます。

注意： XMLTYPE.transform() 構文も使用できます。これは、XMLtransform() と同じです。

図 6-2 に、渡された XML スタイルシートを使用して、XMLTransform() によって XML 文書が変換される方法を示します。この関数では、XSLT スタイルシートで指定されているとおり、処理された出力が XML、HTML などとして戻されます。通常、Oracle9*i* データベースに XMLType として格納された XML 文書の取出しまたは生成を行う場合、XMLTransform() を使用する必要があります。

> **参照：** 第 1 章「Oracle XML DB の概要」の図 1-1「Oracle XML DB のアーキテクチャ : XMLType 記憶域およびリポジトリ」を参照してください。

図 6-1 XMLTransform() の構文

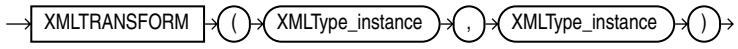
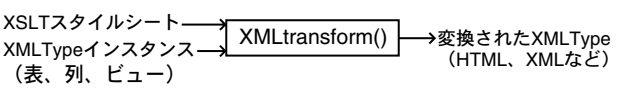


図 6-2 XMLTransform() の使用



XMLTransform() の例

次のコードを使用して、この章の例を実行するために必要な XML Schema および表を設定します。

```
--register schema
begin
dbms_xmlschema.deleteSchema('http://www.example.com/schemas/ipo.xsd',4);
end;
/
begin
dbms_xmlschema.registerSchema('http://www.example.com/schemas/ipo.xsd',
'<schema targetNamespace="http://www.example.com/IPO"
         xmlns="http://www.w3.org/2001/XMLSchema"
         xmlns:ipo="http://www.example.com/IPO">
  <!-- annotation>
   <documentation xml:lang="en">
    International Purchase order schema for Example.com
    Copyright 2000 Example.com. All rights reserved.
   </documentation>
  </annotation -->
  <element name="purchaseOrder" type="ipo:PurchaseOrderType"/>
  <element name="comment" type="string"/>
  <complexType name="PurchaseOrderType">
   <sequence>
    <element name="shipTo"     type="ipo:Address"/>
    <element name="billTo"     type="ipo:Address"/>
    <element ref="ipo:comment" minOccurs="0"/>
    <element name="items"      type="ipo:Items"/>
   </sequence>
```

```
   <attribute name="orderDate" type="date"/>
  </complexType>
  <complexType name="Items">
   <sequence>
    <element name="item" minOccurs="0" maxOccurs="unbounded">
     <complexType>
      <sequence>
       <element name="productName" type="string"/>
       <element name="quantity">
        <simpleType>
         <restriction base="positiveInteger">
          <maxExclusive value="100"/>
         </restriction>
        </simpleType>
       </element>
       <element name="USPrice"    type="decimal"/>
       <element ref="ipo:comment" minOccurs="0"/>
       <element name="shipDate"   type="date" minOccurs="0"/>
      </sequence>
      <attribute name="partNum" type="ipo:SKU" use="required"/>
     </complexType>
    </element>
   </sequence>
  </complexType>
  <complexType name="Address">
   <sequence>
    <element name="name"    type="string"/>
    <element name="street"  type="string"/>
    <element name="city"    type="string"/>
    <element name="state"   type="string"/>
    <element name="country" type="string"/>
    <element name="zip"     type="string"/>
   </sequence>
  </complexType>
  <simpleType name="SKU">
   <restriction base="string">
    <pattern value=" {3}-[A-Z]{2}"/>
   </restriction>
  </simpleType>
</schema>',
   TRUE, TRUE, FALSE);
end;
/

-- create table to hold XML instance documents
DROP TABLE po_tab;
CREATE TABLE po_tab (id number, xmlcol xmltype)
```

```
 XMLTYPE COLUMN xmlcol
 XMLSCHEMA "http://www.example.com/schemas/ipo.xsd"
 ELEMENT "purchaseOrder";

INSERT INTO po_tab VALUES(1, xmltype(
'<?xml version="1.0"?>
<ipo:purchaseOrder
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:ipo="http://www.example.com/IPO"
  xsi:schemaLocation="http://www.example.com/IPO
                        http://www.example.com/schemas/ipo.xsd"
  orderDate="1999-12-01">
    <shipTo xsi:type="ipo:Address">
        <name>Helen Zoe</name>
        <street>121 Broadway</street>
        <city>Cardiff</city>
        <state>Wales</state>
        <country>UK</country>
        <zip>CF2 1QJ</zip>
    </shipTo>
    <billTo xsi:type="ipo:Address">
        <name>Robert Smith</name>
        <street>8 Oak Avenue</street>
        <city>Old Town</city>
        <state>CA</state>
        <country>US</country>
        <zip>95819</zip>
    </billTo>
    <items>
        <item partNum="833-AA">
            <productName>Lapis necklace</productName>
            <quantity>1</quantity>
            <USPrice>99.95</USPrice>
            <ipo:comment>Want this for the holidays!</ipo:comment>
            <shipDate>1999-12-05</shipDate>
        </item>
    </items>
</ipo:purchaseOrder>'));
```

次の例では、XMLTransform() を使用して、XMLType として格納された XML データを、HTML、XML またはその他の言語に変換する方法を示します。

例 6-1 XSLT スタイルシートを取り出すための、XMLTransform() および DBUriType を使用した XMLType インスタンスの変換

```
DROP TABLE stylesheet_tab;
CREATE TABLE stylesheet_tab(id NUMBER, stylesheet xmltype);
INSERT INTO stylesheet_tab VALUES (1, xmltype(
'<?xml version="1.0" ?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="*">
    <td>
    <xsl:choose>
      <xsl:when test="count(child::*) > 1">
        <xsl:call-template name="nested"/>
      </xsl:when>
      <xsl:otherwise>
        <xsl:value-of select="name(.)"/>:<xsl:value-of select="text()"/>
      </xsl:otherwise>
    </xsl:choose>
    </td>
</xsl:template>
<xsl:template match="*" name="nested" priority="-1" mode="nested2">
    <b>
    <!-- xsl:value-of select="count(child::*)"/ -->
    <xsl:choose>
      <xsl:when test="count(child::*) > 1">
        <xsl:value-of select="name(.)"/>:<xsl:apply-templates mode="nested2"/>
      </xsl:when>
      <xsl:otherwise>
        <xsl:value-of select="name(.)"/>:<xsl:value-of select="text()"/>
      </xsl:otherwise>
    </xsl:choose>
    </b>
</xsl:template>
</xsl:stylesheet>'
));

SELECT XMLTransform(x.xmlcol,
  dburiType('/SCOTT/STYLESHEET_TAB/ROW[ID =
1]/STYLESHEET/text()').getXML()).getStringVal()
  AS result
   FROM po_tab x;

-- The preceding statement produces the following output:
-- RESULT
```

```
-- ------------------------------------------------------
-- <td>
--   <b>ipo:purchaseOrder:
--     <b>shipTo:
--       <b>name:Helen Zoe</b>
--       <b>street:100 Broadway</b>
--       <b>city:Cardiff</b>
--       <b>state:Wales</b>
--       <b>country:UK</b>
--       <b>zip:CF2 1QJ</b>
--     </b>
--     <b>billTo:
--       <b>name:Robert Smith</b>
--       <b>street:8 Oak Avenue</b>
--       <b>city:Old Town</b>
--       <b>state:CA</b>
--       <b>country:US</b>
--       <b>zip:95819</b>
--     </b>
--     <b>items:</b>
--   </b>
-- </td>
```

例 6-2 XSLT スタイルシートを取り出すための、XMLTransform() および副問合せ SELECT を使用した XMLType インスタンスの変換

次の例では、XMLType インスタンスを変換するために、格納されたスタイルシートを使用する方法を示します。前述の例と異なり、この例ではスカラー副問合せを使用して、格納されたスタイルシートを取り出します。

```
SELECT XMLTransform(x.xmlcol,
    (select stylesheet from stylesheet_tab where id = 1)).getStringVal()
     AS result
   FROM po_tab x;
```

例 6-3 一時スタイルシートおよび XMLTransform() を使用した XMLType インスタンスの変換

次の例では、一時スタイルシートを使用して XMLType インスタンスを変換する方法を示します。

```
SELECT x.xmlcol.transform(xmltype(
'<?xml version="1.0" ?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="*">
    <td>
    <xsl:choose>
```

```
      <xsl:when test="count(child::*) > 1">
        <xsl:call-template name="nested"/>
      </xsl:when>
      <xsl:otherwise>
        <xsl:value-of select="name(.)"/>:<xsl:value-of select="text()"/>
      </xsl:otherwise>
    </xsl:choose>
    </td>
</xsl:template>
<xsl:template match="*" name="nested" priority="-1" mode="nested2">
    <b>
    <!-- xsl:value-of select="count(child::*)"/ -->
    <xsl:choose>
      <xsl:when test="count(child::*) > 1">
        <xsl:value-of select="name(.)"/>:<xsl:apply-templates mode="nested2"/>
      </xsl:when>
      <xsl:otherwise>
        <xsl:value-of select="name(.)"/>:<xsl:value-of select="text()"/>
      </xsl:otherwise>
    </xsl:choose>
    </b>
</xsl:template>
</xsl:stylesheet>'
)).getStringVal()
FROM po_tab x;
```

XMLType インスタンスの検証

多くの場合、特定の XML 文書が整形式であることに加えて、特定の文書が特定の XML Schema に準拠するかどうか（特定の XML Schema に対して妥当であるかどうか）を認識する必要があります。

デフォルトでは、Oracle9*i* データベースによって、XMLType インスタンスが整形式かどうかが確認されます。また、スキーマベースの XMLType インスタンスの場合、Oracle9*i* データベースによっていくつかの基本的な妥当性チェックが実行されます。完全な XML Schema の検証（W3C で指定）は高コストな操作であるため、XMLType インスタンスが構成、格納または取得される場合、これらは完全に検証されません。

XML 文書の「validated」状態を検証または操作するために、次の関数および SQL 演算子が提供されています。

XMLIsValid()

XMLIsValid() は、SQL 演算子です。入力インスタンスが、指定された XML Schema に準拠するかどうかを確認します。XML インスタンスの検証状態は変更されません。XML Schema の URL が指定されておらず、XML 文書がスキーマベースである場合、XMLType インスタンス独自のスキーマに対して準拠しているかどうかが確認されます。いずれかの引数

が NULL に指定されている場合、結果は NULL になります。検証が失敗すると、0（ゼロ）が戻されますが、検証が失敗した理由を説明するエラーは通知されません。

構文

```
XMLIsValid ( XMLType_inst [, schemaurl [, elem]])
```

パラメータ：

- XMLType_inst - 指定された XML Schema に対して検証する XMLType インスタンス。
- schurl - 準拠を確認する XML Schema URL。
- elem - 指定されたスキーマの検証対象要素。複数の最上位要素を定義する XML Schema が存在し、このうちのいずれかの要素に対する準拠を確認する場合に有効です。

schemaValidate

schemaValidate は、メンバー・プロシージャです。XML Schema に対して XML インスタンスがまだ検証されていない場合に、検証を実行します。XML Schema に基づかない文書の場合は、エラーが発生します。検証が失敗するとエラーが発生しますが、それ以外の場合、文書の状態は VALIDATED に変更されます。

構文

```
MEMBER PROCEDURE schemaValidate
```

isSchemaValidated()

isSchemaValidated() は、メンバー関数です。XMLType インスタンスの検証状態を戻し、スキーマベースのインスタンスが、そのスキーマに対して実際に検証されたかどうかを通知します。スキーマに対してインスタンスが検証された場合は 1 を戻し、それ以外の場合は 0（ゼロ）を戻します。

構文

```
MEMBER FUNCTION isSchemaValidated return NUMBER deterministic
```

setSchemaValidated()

setSchemaValidated() は、メンバー関数です。入力 XML インスタンスの検証状態を設定します。

構文

```
MEMBER PROCEDURE setSchemaValidated(flag IN BINARY_INTEGER := 1)
```

パラメータ：

flag は、検証済の場合は 1、検証済でない場合は 0（ゼロ）です。デフォルト値は 1 です。

isSchemaValid()

isSchemaValid() は、メンバー関数です。入力インスタンスが、指定された XML Schema に準拠するかどうかを確認します。XML インスタンスの検証状態は変更されません。XML Schema URL が指定されておらず、XML 文書がスキーマベースである場合、XMLType インスタンス独自のスキーマに対して準拠しているかどうかが確認されます。検証が失敗すると例外が発生し、検証が失敗した理由が示されます。

構文

```
member function isSchemaValid(schurl IN VARCHAR2 := NULL, elem IN VARCHAR2 :=
    NULL) return NUMBER deterministic
```

パラメータ：

schurl - 準拠を確認する XML Schema URL。

elem - 指定されたスキーマの検証対象要素。複数の最上位要素を定義する XML Schema が存在し、このうちのいずれかの要素に対する準拠を確認する場合に有効です。

XMLType として格納された XML データの検証例

次の例では、isSchemaValid()、setSchemaValidated() および isSchemaValidated() を使用して、Oracle XML DB に XMLType として格納されている XML データを検証する方法を示します。

例 6-4 isSchemaValid() の使用

```
SELECT x.xmlcol.isSchemaValid('http://www.example.com/schemas/ipo.xsd',
              'purchaseOrder')
    FROM po_tab x;
```

例 6-5 isSchemaValid() を使用した XML の検証

次の PL/SQL の例では、XML Schema PO.xsd に対して XML インスタンスを検証します。

```
declare
  xmldoc xmltype;
  begin
  -- populate xmldoc (for example, by fetching from table)
  -- validate against XML schema
  xmldoc.isSchemaValid('http://www.oracle.com/PO.xsd');
  if xmldoc.isSchemaValid = 1 then  --
  else --
  end if;
end;
```

例 6-6 トリガー内での schemaValidate() の使用

XMLType の schemaValidate() メソッドを INSERT トリガーおよび UPDATE トリガー内で使用して、表に格納されたすべてのインスタンスが XML Schema に対して検証されたことを確認できます。

```
DROP TABLE po_tab;
CREATE TABLE po_tab OF xmltype
   XMLSchema "http://www.example.com/schemas/ipo.xsd" element "purchaseOrder";

CREATE TRIGGER emp_trig  BEFORE INSERT OR UPDATE ON po_tab FOR EACH ROW
DECLARE
   newxml xmltype;
BEGIn
   newxml := :new.sys_nc_rowinfo$;
   xmltype.schemavalidate(newxml);
END;
/
```

例 6-7 CHECK 制約内での XMLIsSchemaValid() の使用

この例では、XMLIsValid() を使用して、次の操作を実行します。

- XMLType インスタンスが指定された XML Schema に準拠していることの確認
- CHECK 制約を使用した、受信した XML 文書が妥当であることの確認

```
DROP TABLE po_tab;
CREATE TABLE po_tab OF XMLTYPe
   (CHECK (XMLIsValid(sys_nc_rowinfo$) = 1))
   XMLSchema "http://www.example.com/schemas/ipo.xsd" element "purchaseOrder";
```

> **注意：** 前述の項に示す検証関数および演算子を使用すると、簡単に妥当性チェックを実行できます。これらのうち、isSchemaValid() のみが、エラーの発生時に検証が失敗した理由を示します。

7

Oracle Text を使用した XML データの検索

この章では、Oracle Text 機能を使用した XML データの索引付けおよび問合せについて説明します。この章の内容は次のとおりです。

- Oracle Text の使用 : 高度な使用方法
- 例 : XML ベースの会議議事録の検索
- Oracle Text の FAQ

注意 : Oracle Text では、WITHIN または INPATH 演算子を使用できます。Oracle9*i* データベース リリース 1（9.0.1）では、XML 文書での XPath 検索を処理するための INPATH が導入されました。WITHIN 演算子を使用して実行可能なすべての操作は、INPATH を使用しても実行できます。Oracle9*i* データベース リリース 1（9.0.1）以上では、INPATH 構文を使用して XML データを検索することをお薦めします。

Oracle Text を使用した XML データの検索

この章では、次の Oracle Text 機能について説明します。

- XML 文書にセクション・グループおよび索引を作成する方法
- Oracle Text を使用して XML 問合せアプリケーションを構築し、XML 文書のデータを検索して取り出す方法
- Oracle Text を使用した XMLType データの検索

Oracle Text の概要

注意： Oracle Text は、サーバーベースの実装です。

参照： http://otn.oracle.com/products/text を参照してください。

Oracle Text（以前の *inter*Media Text）を使用して、XML 文書を検索できます。これは、Oracle に格納されたすべてのテキストまたはドキュメントを索引付けすることによって、Oracle9*i* データベースを拡張します。また、ファイル・システム内のドキュメントおよび URL も検索できます。

Oracle Text を使用すると、次の操作を行うことができます。

- 一般的な標準 SQL を使用したコンテンツ・ベースの問合せ（特定のワードを含むテキストおよびドキュメントの検索など）。
- Oracle9*i* データベースを使用して、従来の関連情報を使用した統合的な方法によって、テキストおよびドキュメントを管理するためのファイルベースのテキスト・アプリケーション。
- 英語ドキュメントの概念検索。
- テーマ / 要点パッケージを使用した英語ドキュメントのテーマ分析。
- 検索で一致した文字列のハイライト表示。Oracle Text を使用すると、ドキュメントを様々な方法でレンダリングできます。たとえば、問合せ文字列（英語の場合、ワード問合せのワードまたは ABOUT 問合せのテーマ）をハイライト表示させてドキュメントを表示できます。INPATH/HASPATH 問合せ要素をハイライト表示させて XML 文書を表示することもできます。これを行うには、CTX_DOC.MARKUP または HIGHLIGHT プロシージャを使用します。
- Oracle Text PL/SQL パッケージを使用したドキュメントの表示およびシソーラスのメンテナンス。

Oracle Text を使用しなくても、データベースに格納された XML データを直接問い合せることはできます。ただし、Oracle Text を使用すると、問合せのパフォーマンスが向上します。

参照： 次のマニュアルまたは Web サイトを参照してください。

- 『Oracle Text リファレンス』
- 『Oracle Text アプリケーション開発者ガイド』
- `http://otn.oracle.com/products/text`

Oracle Text へのアクセス

Oracle Text は、Oracle9*i* データベース Standard Edition、Enterprise Edition および Personal Edition のすべてのライセンスに付属する標準機能です。この機能は、インストール時に選択する必要があります。特別なインストール手順は不要です。

Oracle Text は、基本的に、CTXSYS が所有する一連のスキーマ・オブジェクトです。これらのオブジェクトは、Oracle カーネルにリンクされます。スキーマ・オブジェクトは、Oracle9*i* データベースのインストールに含まれます。

XMLType での Oracle Text サポート

今回のリリースでは、`XMLType` 列を含む表で Oracle Text 検索を実行できます。

Oracle Text の詳細な例

Oracle Text およびセクション・グループ索引の作成の詳細な例は、次の Web サイトを参照してください。

http://otn.oracle.com/products/text

この章の例に関する前提

XML テキストは、文字セマンティクスを持つ Oracle9*i* データベース表内の VARCHAR2 型または CLOB 型です。Oracle Text は、ファイル・システムまたは URL 上のドキュメントも処理できますが、この章ではこれらのドキュメント・タイプについては考慮しません。

例を簡略化するために、Oracle Text オプションの一部を使用し、次のことを前提とします。

- すべての XML データが、US-ASCII の 7 ビット・キャラクタ・セットを使用して表されます。
- 「*」などの文字を空白として処理するか、または単語の一部として処理するかについての問題は含まれません。
- Oracle Text 索引を実装する Oracle スキーマ・オブジェクトの記憶特性については、考慮しません。

- CREATE INDEX 文または ALTER INDEX 文内の SECTION GROUP パラメータに注目します。CREATE INDEX 文および ALTER INDEX 文に使用できるその他のパラメータ・タイプは、DATASTORE、FILTER、LEXER、STOPLIST および WORDLIST です。

  次に、CREATE INDEX 文で SECTION GROUP パラメータを使用した例を示します。

```
CREATE INDEX my_index
  ON my_table ( my_column )
  INDEXTYPE IS ctxsys.context
  PARAMETERS ( 'SECTION GROUP my_section_group' ) ;
```

- 特に、この例では、AUTO_SECTION_GROUP、XML_SECTION_GROUP および PATH_SECTION_GROUP を使用することに注目します。
- XML データ（タグ付けまたはマークアップされたデータ）の処理方法に注目します。Oracle Text は、他にも様々な種類のデータを処理します。

  参照：

  - 『Oracle Text アプリケーション開発者ガイド』を参照してください。
  - これらのパラメータ・タイプの詳細は、『Oracle Text リファレンス』を参照してください。

Oracle Text のユーザーおよびロール

Oracle Text では、次のユーザーおよびロールを使用できます。

- ユーザーを管理する CTXSYS ユーザー
- Oracle Text プリファレンスを作成および削除して Oracle Text PL/SQL パッケージを使用する CTXAPP ロール

CTXSYS ユーザー

CTXSYS ユーザーは、インストール時に作成されます。Oracle Text ユーザーをこのユーザーとして管理します。CTXSYS ユーザーには、次の権限があります。

- システム定義のプリファレンスを変更する権限
- 他のユーザーのプリファレンスを削除および変更する権限
- CTX_ADM PL/SQL パッケージのプロシージャをコールして、サーバーの起動およびシステム・パラメータの設定を行う権限
- ctxsrv サーバーを起動する権限
- すべてのシステム定義のビューを問い合せる権限
- CTXAPP ロールを使用してユーザーのすべてのタスクを実行する権限

CTXAPP ロール

すべてのユーザーは、Oracle Text 索引を作成したり、テキスト問合せを発行することができます。その他のタスクを実行する場合は、CTXAPP ロールを使用してください。これは、システム定義のロールです。このロールを使用して、次のタスクを実行できます。

- Oracle Text プリファレンスの作成および削除
- CTX_DDL パッケージなどの Oracle Text PL/SQL パッケージの使用

CONTAINS 演算子を使用した問合せ

Oracle Text の主な用途は、CONTAINS 演算子を使用することです。CONTAINS 演算子は、テキスト問合せ用の問合せ式を指定するために、SELECT 文の WHERE 句で使用できます。

CONTAINS 構文

CONTAINS 構文は次のとおりです。

```
...WHERE CONTAINS([schema.]column,text_query VARCHAR2,[label NUMBER])
```

パラメータは次のとおりです。

表 7-1 CONTAINS 演算子 : 構文の説明

構文	説明
[schema.] column	検索するテキスト列を指定します。この列には、それに対応付けられた Text 索引が必要です。
text_query	列内での検索を定義する問合せ式を指定します。
label	オプションで、CONTAINS 演算子によって生成されたスコアを識別するラベルを指定します。

CONTAINS は、選択された各行について、そのドキュメント行が問合せにどの程度関連しているかを示す 0 ～ 100 の数値を戻します。数値 0（ゼロ）は、Oracle がその行内で一致するデータを検索しなかったことを意味します。このスコアは、SCORE 演算子を使用して取得できます。

注意 : この数値を取得するには、ラベルを指定して SCORE 演算子を使用してください。

例 7-1 CONTAINS を使用した単純な SELECT 文の使用

次の例は、SELECT 構文で CONTAINS 演算子を使用する方法を示します。

```
SELECT id FROM my_table
   WHERE
   CONTAINS (my_column, 'receipts') > 0
```

CONTAINS 演算子の 'receipts' パラメータは、テキスト問合せ式ともいいます。

注意： CONTAINS 演算子を含む SQL 文を実行するには、Oracle Text 索引が必要です。

例 7-2 関連性を取得するラベルを指定した SCORE 演算子の使用

次の例は、テキスト列内で「Oracle」という文字列を含むすべてのドキュメントを検索します。各行のスコアは、SCORE 演算子で 1 というラベルを使用して選択されます。

```
SELECT SCORE(1), title from newsindex
          WHERE CONTAINS(text, 'oracle', 1) > 0 ORDER BY SCORE(1) DESC;
```

CONTAINS 演算子の後には、>0 の構文が続く必要があります。この構文は、CONTAINS 演算子が計算したスコアの値が 0（ゼロ）より大きい場合に、その行が選択されることを指定します。

SELECT 句などで SCORE 演算子がコールされると、演算子は例に示すとおり、ラベルの値を参照する必要があります。

問合せをドキュメント・セクションに限定する WITHIN 演算子の使用

HTML や XML のように内部構造を持つドキュメントの場合、索引付けを行う前に、埋込みタグを使用してドキュメント・セクションを定義できます。これによって、WITHIN 演算子を使用してセクション内で問合せできます。

注意： これは、XML_SECTION_GROUP のみで実行できます。AUTO_SECTION_GROUP または PATH_SECTION_GROUP では実行できません。

SECTION_GROUPS の概要

XML_SECTION_GROUP、AUTO_SECTION_GROUP または PATH_SECTION_GROUP のいずれかのセクション・グループ・タイプを使用して索引付けする場合、属性セクション内で問合せを行うことができます。次の XML 文書について考えてみます。

```
<book title="Tale of Two Cities">It was the best of times.</book>
```

XML_SECTION_GROUP

XML_SECTION_GROUP を使用する場合、次のいずれかのセクションを指定できます。

- ゾーン・セクション
- フィールド・セクション
- 属性セクション
- 特殊セクション

この章では、ゾーン・セクション、フィールド・セクションおよび属性セクションのみに注目します。特殊セクションの詳細は、『Oracle Text リファレンス』および『Oracle Text アプリケーション開発者ガイド』を参照してください。

ゾーン・セクション : CTX_DLL.ADD_ZONE_SECTION プロシージャ

構文は次のとおりです。

```
CTX_DDL.ADD_ZONE_SECTION(
  group_name     in    varchar2,
  section_name   in    varchar2,
  tag            in    varchar2);
```

chapter をゾーン・セクションとして定義するには、次のとおり XML_SECTION_GROUP を作成し、ゾーン・セクションを定義します。

```
EXEC ctx_ddl_create_section_group('myxmlgroup', 'XML_SECTION_GROUP');
EXEC ctx_ddl.add_zone_section('myxmlgroup', 'chapter', 'chapter');
```

ゾーン・セクションをこのように定義し、ドキュメント・セットを索引付けすると、次のとおり XML chapter ゾーン・セクションを問い合せることができます。

```
'Cities within chapter'
```

フィールド・セクション : CTX_DLL.ADD_FIELD_SECTION プロシージャ

構文は次のとおりです。

```
CTX_DDL.ADD_FIELD_SECTION(
  group_name      in    varchar2,
  section_name    in    varchar2,
  tag             in    varchar2);
```

abstract をフィールド・セクションとして定義するには、次のとおり XML_SECTION_GROUP を作成しフィールド・セクションを定義します。

```
EXEC ctx_ddl_create_section_group('myxmlgroup', 'XML_SECTION_GROUP');
EXEC ctx_ddl.add_field_section('myxmlgroup', 'abstract', 'abstract');
```

フィールド・セクションをこのように定義し、ドキュメント・セットを索引付けすると、次のとおり XML abstract フィールド・セクションを問い合せることができます。

```
'Cities within abstract'
```

属性セクション : CTX_DLL.ADD_ATTR_SECTION プロシージャ

構文は次のとおりです。

```
CTX_DDL.ADD_ATTR_SECTION(
  group_name      in    varchar2,
  section_name    in    varchar2,
  tag             in    varchar2);
```

booktitle 属性を属性セクションとして定義するには、次のとおり XML_SECTION_GROUP を作成し、属性セクションを定義します。

```
EXEC ctx_ddl_create_section_group('myxmlgroup', 'XML_SECTION_GROUP');
EXEC ctx_ddl.add_attr_section('myxmlgroup', 'booktitle', 'book@title');
```

属性セクションをこのように定義し、ドキュメント・セットを索引付けすると、次のとおり XML booktitle 属性テキストを問い合せることができます。

```
'Cities within booktitle'
```

属性セクションまたはフィールド・セクションの問合せに対する制約

次の制約が、属性セクションまたはフィールド・セクション内での問合せに適用されます。

- 通常の属性テキストの問合せは、WITHIN 句で修飾されていない場合、動作しません。次の XML 文書があると想定します。

  ```
  <book title="Tale of Two Cities">It was the best of times.</book>
  ```

  Tale の問合せは、'WITHIN title@book' で修飾されていない場合、動作しません。

- 属性セクションまたはフィールド・セクションは、ネストした WITHIN 問合せでは使用できません。
- 句は、属性テキストを無視します。たとえば、次のオリジナル・ドキュメントがあると想定します。

  ```
  ....Now is the time for all good <word type="noun"> men </word> to come to the
  aid......
  ```

  検索を行うと、通常の問合せによって「good men」が検出され、間に挿入されている属性テキストは無視されます。

WITHIN での AUTO_SECTION_GROUP または PATH_SECTION_GROUP

AUTO_SECTION_GROUP または PATH_SECTION_GROUP を使用して XML 文書を索引付けする場合、Oracle9*i* データベースでは自動的にセクションが作成されます。

属性セクション booktitle 内で Tale を検索するには、次のとおり、SELECT 文に WITHIN 句を含めます。

- AUTO_SECTION_GROUP を使用している場合

  ```
  ... WHERE CONTAINS ('Tale INPATH booktitle')>0;
  ```

- PATH_SECTION_GROUP を使用している場合

  ```
  ... WHERE CONTAINS ('Tale INPATH title@book')>0;
  ```

参照： 7-51 ページの「DOCTYPE 間のタグの区別」を参照してください。

ALTER INDEX を使用したセクションまたは停止セクションの動的追加

ALTER INDEX 文の構文は次のとおりです。

```
ALTER INDEX [schema.]index REBUILD [ONLINE] [PARAMETERS (paramstring)];
```

この場合の paramstring は次のとおりです。

```
paramstring = 'replace [datastore datastore_pref]
                       [filter filter_pref]
                       [lexer lexer_pref]
                       [wordlist wordlist_pref]
                       [storage storage_pref]
                       [stoplist stoplist]
                       [section group section_group]
                          [memory memsize]
|    ...
|    add zone section section_name tag tag
```

```
|    add field section section_name tag tag [(VISIBLE | INVISIBLE)]
|    add attr section section_name tag tag@attr
|    add stop section tag'
```

追加されたセクションは、この操作の後に索引付けされたドキュメントのみに適用されます。したがって、変更を有効にするには、タグを含む既存のドキュメントを手動で再索引付けする必要があります。この文では索引は再作成されません。

セクション問合せのための WITHIN 構文

セクションを問い合せる WITHIN 構文は次のとおりです。

```
...WHERE CONTAINS(text,'XML WITHIN title') >0;...
```

この構文は、セクション内で text 式を検索します。XML_SECTION_GROUP を使用する場合、事前定義されているゾーン、フィールドまたは属性セクションには次の制限事項が適用されます。

- **ゾーン・**セクションの場合、ゾーン・セクションまたは特殊セクションを指定した 1 つ以上の WITHIN 演算子（ネストした WITHIN）を式に含めることができます。
- **フィールド・**セクションまたは**属性**セクションの場合、式に含めることができる WITHIN 演算子は 1 つのみです。

WITHIN 句は、組み合せてネストすることができます。XML のセクションをより詳細に検索するには、CONTAINS を使用した SELECT 文内に WITHIN 句を指定します。

WITHIN 演算子の制限事項

WITHIN 演算子には、次の制限事項があります。

- WITHIN 句を句に埋め込むことはできません。たとえば、term1 WITHIN section term2 のような句を作成することはできません。
- WITHIN 演算子を「$」、「!」、「*」などの拡張演算子と組み合せることはできません。
- WITHIN 演算子は予約語であるため、検索するには、中カッコを使用してその文字列をエスケープする必要があります。

参照：『Oracle Text リファレンス』を参照してください。

XPath に類似した式を使用した INPATH 演算子または HASPATH 演算子の検索

Oracle Text を使用したパスの索引付けおよびパスの問合せ

Oracle Text リリース 9.2 には、XPath に類似した問合せ言語をサポートする、新しいセクション型および新しい問合せ演算子が導入されています。XML パス検索を使用した型コンテキストの索引によって、XML 文書で非常に複雑なセクション検索が可能になります。次の項では、パスの索引付けおよびパスの問合せの基本概念について説明します。

パスの索引付け

セクション検索は、セクション・グループを定義することによって実行できます。XML パス検索を使用するには、次のとおり、新しいセクション・グループの PATH_SECTION_GROUP を使用して、Oracle Text 索引を作成する必要があります。

```
begin
   ctx_ddl.create_section_group('mypathgroup','PATH_SECTION_GROUP');
end;
```

Oracle Text 索引を作成するには、次のコマンドを使用します。

```
create index order_idx on library_catalog(text)
 indextype is ctxsys.context
 parameters ('SECTION GROUP mypathgroup');
```

パスの問合せ

Oracle Text パスの問合せ言語は、W3C の XPath に基づいています。Oracle9*i* データベース リリース 1（9.0.1）以上の場合、INPATH および HASPATH 演算子を使用して、パスの問合せを表すことができます。

INPATH 演算子を使用した XML 文書でのパス検索

INPATH 演算子を使用して、XML 文書でパス検索を実行できます。表 7-2 に、パス検索に INPATH 演算子を使用する方法の概要を示します。

表 7-2 INPATH 演算子を使用した XML ドキュメントのパス検索

パス検索機能	構文	説明
単純なタグ検索	virginia INPATH (//STATE)	<STATE> と </STATE> の間に「virginia」という文字列があるすべての文書を検索します。文書構造のすべてのレベルにある STATE 要素が検索されます。
大 / 小文字の区別	virginia INPATH (STATE) virginia INPATH (State)	パス検索では、タグ名および属性名の大 / 小文字が区別されます。virginia INPATH(STATE) では、<STATE>virginia</STATE> は検索されますが、<State>virginia</State> は検索されません。後者を検索するには、virginia INPATH (State) として検索する必要があります。
最上位タグの検索	virginia INPATH (Legal) virginia INPATH (/Legal) たとえば、次の問合せでは、<order> と </order> 間の Quijote が検索されます。 select id from library_catalog where contains(text,'Quijote INPATH(order)') > 0; この場合、<order> は、最上位タグである必要があります。	最上位タグである Legal 要素に「virginia」があるすべての文書を検索します。「Legal」は、文書の最上位タグである必要があります。間に挿入されている他のタグに関係なく、「virginia」はこのタグのどこかにあります。次に例を示します。 <?xml version=″1.0″ standalone=″yes″?> <!-- <?xml-stylesheet type="text/xsl" href="./xsl/vacourtfiling(html).xsl"?> --> <Legal> <CourtFiling> <Filing ID="f001" FilingType="Civil"> <LeadDocument> <CaseCaption> <CourtInformation> <Location> <Address> <AddressState>VIRGINIA</AddressState> </Address>... </Legal>

表 7-2 INPATH 演算子を使用した XML ドキュメントのパス検索 （続き）

パス検索機能	構文	説明
任意のレベルのタグの検索	virginia INPATH (//Address) たとえば、ダブルスラッシュは、そのタグ以下の任意のレベルを示します。次の問合せでは、<title> タグ内の最上位以下のレベルにある Quijote が検索されます。 select id from library_catalog where contains(text,'Quijote INPATH(//title)') > 0;	「virginia」が「Address」タグのどこかにあります。他のタグの間にある場合もあります。次に例を示します。 <?xml version="1.0" standalone="yes"?> <!-- <?xml-stylesheet type="text/xsl" href="./xsl/vacourtfiling(html).xsl"?> --> <Legal> <CourtFiling> <Filing ID="f001" FilingType="Civil"> <LeadDocument> <CaseCaption> <CourtInformation> <Location> <Address> <AddressState> VIRGINIA </AddressState>... </Legal>
直接の親子関係のパス検索	virginia INPATH (//CourtInformation/Location) 次に例を示します。 select id from library_catalog where contains(text,'virginia INPATH(order/item)') > 0;	CourtInformation 要素の直接の子である Location 要素に「virginia」が含まれるすべての文書を検索します。次に例を示します。 <?xml version="1.0" standalone="yes"?> <!-- <?xml-stylesheet type="text/xsl" href="./xsl/vacourtfiling(html).xsl"?> --> <Legal> <CourtFiling> <Filing ID="f001" FilingType="Civil"> <LeadDocument> <CaseCaption> <CourtInformation> <Location> <Address> <AddressState>VIRGINIA</AddressState> </Address>... </CourtInformation>

表 7-2 INPATH 演算子を使用した XML ドキュメントのパス検索 （続き）

パス検索機能	構文	説明
シングルレベル・ワイルド・カード検索	virginia INPATH(A/*/B) 'virginia INPATH (//CaseCaption/*/Location)'	A 要素の孫である B 要素に「virginia」があるすべての文書を検索します。たとえば、<A><D><B>virginia</B></D></A> などです。中間要素は索引付けされた XML タグである必要はありません。次に例を示します。 <?xml version="1.0" standalone="yes"?> <!-- <?xml-stylesheet type="text/xsl" href="./xsl/vacourtfiling(html).xsl"?> --> <Legal> <CourtFiling> <Filing ID="f001" FilingType="Civil"> <LeadDocument> <CaseCaption> <CourtInformation> <Location> <Address> <AddressState> VIRGINIA </AddressState>... </Legal>

表 7-2 INPATH 演算子を使用した XML ドキュメントのパス検索 （続き）

パス検索機能	構文	説明
マルチレベル・ワイルド・カード検索	'virginia INPATH (Legal/*/Filing/*/*/CourtInformation)'	「Legal」が最上位タグで、タグ・レベルが「Legal」と「Filing」の間に 1 つのみ、「Filing」と「CourtInformation」の間に 2 つのみある必要があります。「virginia」は「CourtInformation」内のどこかに表示されています。次に例を示します。 <?xml version="1.0" standalone="yes"?> <!-- <?xml-stylesheet type="text/xsl" href="./xsl/vacourtfiling(html).xsl"?> --> <Legal> <CourtFiling> <Filing ID="f001" FilingType="Civil"> <LeadDocument> <CaseCaption> <CourtInformation> <Location> <Address> <AddressState>VIRGINIA</AddressState> </Address> </Location> <CourtName> IN THE CIRCUIT COURT OF LOUDOUN COUNTY </CourtName> </CourtInformation>....
子孫の検索	virginia INPATH(A//B)	A 要素の子孫（任意のレベル）である B 要素に「virginia」があるすべての文書を検索します。
属性の検索	virginia INPATH(A/@B)	A 要素の B 属性に「virginia」があるすべての文書を検索します。構文 <tag>/@<attribute> を使用して属性値内を検索できます。 select id from library_catalog where contains(text,'dvd INPATH(//item/@type)') > 0; AND and OR ブール AND および OR を使用して、テストの存在述語または等価述語を組み合せることができます。 select id from library_catalog where contains(text,'Levy or Cervantes INPATH(//title)') >0;

表 7-2 INPATH 演算子を使用した XML ドキュメントのパス検索 （続き）

パス検索機能	構文	説明
子孫 / 属性の存在テスト	virginia INPATH (A[B])	B 要素を直接の子とする A 要素に「virginia」があるすべての文書を検索します。
	任意のレベルのタグの検索を使用して、文書内を検索できます。 select id from library_catalog where contains (text,'Quijote INPATH(/order/title)') > 0; 「*」をシングルレベル・ワイルド・カードとして使用することもできます。* は、1 つのレベルのみと一致します。 select id from library_catalog where contains (text,'Cervantes INPATH(/order/*/author)') > 0;	■ virginia INPATH A[.//B] - B 要素を子孫（任意のレベル）とする A 要素に「virginia」があるすべての文書を検索します。 ■ virginia INPATH A[@B] - B 属性を持つ A 要素に「virginia」があるすべての文書を検索します。

表 7-2 INPATH 演算子を使用した XML ドキュメントのパス検索 （続き）

パス検索機能	構文	説明
属性値のテスト	virginia INPATH A[@B = "foo"]	値が「foo」である B 属性を持つ A 要素に「virginia」があるすべての文書を検索します。 ■ テストには等式のみサポートされています。範囲演算子および範囲関数はサポートされていません。 ■ 等式の左側には属性またはタグを指定する必要があります。リテラルは指定できません。 ■ 右側にはリテラルを指定する必要があります。タグおよび属性は指定できません。
内部等値	次のことを意味します。 デフォルトのレクサーおよびストップリストを使用すると、virginia INPATH (A[@B = "pot of gold"]) は次のいずれかに一致します。 ■ <A B="POT OF GOLD">virginia</A> デフォルトでは、レクサーは大 / 小文字を区別しないため、「pot」は「POT」に一致します。 ■ <A B="POT BLACK GOLD">virginia</A> デフォルトでは、「of」はストップワードであるため、問合せでは、その位置にある任意の文字列に一致します。	内部等値（7-12 ページの「INPATH 演算子を使用した XML 文書でのパス検索」を参照）は、テストの評価に使用されます。 空白は、テキスト索引付けではほとんの場合無視されます。レクサーは大 / 小文字を区別しません。 <A B="pot_of_gold">virginia</A> アンダースコアはアルファベット以外の文字であり、デフォルトでは結合文字ではありません。そのため、ほとんどの場合は空白として処理され、文字列は 3 つの単語に分割されます。 次に例を示します。 select id from library_catalog where contains(text,'(Bob the Builder) INPATH(//item[@type="dvd"])') > 0; 次の問合せでは、行が戻されません。 select id from library_catalog where contains(text,'(Bob the Builder) INPATH(//item[@type="book"])') > 0;

表 7-2 INPATH 演算子を使用した XML ドキュメントのパス検索 （続き）

パス検索機能	構文	説明
数値等値	virginia INPATH (A[@B = 5])	数値リテラルは、使用できますが、テキストとして処理されます。内部等値を使用して評価します。これは、問合せが一致しないことを意味します。 <A B=″5.0″>virginia</A> は、小数点付きの「5.0」が整数の 5 と同じであるとみなされないため、A[@B=5] に一致しません。
論理積のテスト	virginia INPATH (A[B AND C]) virginia INPATH (A[B AND @C = "foo"])...	述語を論理積として結合できます。
パスのテストとノードのテストの結合	virginia INPATH (A[@B = "foo"]/C/D) virginia INPATH(A//B[@C]/D[E])...	ノードのテストは、パス内のすべてのノードに適用できます。

HASPATH 演算子を使用した XML 文書でのパス検索

HASPATH 演算子を使用して、指定されたセクション・パスを含むすべての XML 文書を検索できます。HASPATH は、パスの存在を確認する場合に使用します。セクション等価性の検査にも有効です。order に item が含まれている XML 文書を検索するには、次のコマンドを実行します。

```
select id from library_catalog
    where contains(text,'HASPATH(order/item)') > 0;
```

この場合、最上位タグが order 要素で、item 要素が直接の子として含まれるすべての文書が戻されます。

Oracle Text リリース 9.2 には、XPath に類似した問合せ言語をサポートする、新しいセクション型および新しい問合せ演算子が導入されています。XML パス検索を使用した型コンテキストの索引によって、XML 文書で非常に複雑なセクション検索が可能になります。次に、INPATH および HASPATH を使用したパス問合せの例を示します。次の XML 文書について考えてみます。

```
<?xml version="1.0"?>
  <order>
    <item type="book">
       <title>Crypto</title>
       <author>Levi</author>
    </item>
    <item type="dvd">
       <title> Bob the Builder</title>
       <author>Auerbach</author>
    </item>
```

```
    <item type="book">
       <title>Don Quijote</title>
       <author>Cervantes</author>
    </item>
  </order>
```

通常、INPATH 演算子および HASPATH 演算子は、PATH_SECTION_GROUP を使用して作成した索引にのみ使用します。PATH_SECTION_GROUP を使用すると、パス検索ができます。パス検索は WITHIN 演算子の構文を拡張するため、セクション名のオペランド（右側）は、セクション名ではなく*パス*になります。

HASPATH 演算子を使用したパスの存在の検索

> **注意：** HASPATH 演算子の動作は、XMLType の Existsnode() の動作に類似しています。

HASPATH 演算子は、PATH_SECTION_GROUP を使用して索引を作成した場合にのみ使用します。HASPATH 演算子の構文は次のとおりです。

- **WHERE CONTAINS(column, 'HASPATH(path)'...):** HASPATH は、XML 文書セットを検索し、パスが存在するすべての文書に対してスコア値 100 を戻します。親および子であるパスは、A/B/C のように、スラッシュ（/）文字で区切ります。次の問合せを行うとします。

  ```
  ...WHERE CONTAINS (col,'HASPATH(A/B/C)')>0;
  ```

  この問合せは、検索した次の文書に対してスコア 100 を戻します。

  ```
  <A><B><C>Virginia</C></B></A>
  ```

  この場合、「Virginia」は参照されません。

- **WHERE CONTAINS(column, 'HASPATH(A="value")'...):** HASPATH 句は、XML 文書セットを検索し、コンテンツ値および値のみを含む要素 A があるすべての文書に対してスコア値 100 を戻します。HASPATH を使用して、等価性をテストできます。これは、HASPATH 演算子のセクション等価性の検査機能です。次の問合せを行うとします。

  ```
  ...WHERE CONTAINS virginia INPATH A
  ```

  この問合せは、<A>virginia</A> のみでなく <A>virginia state</A> も検索します。問合せを virginia という用語に限定するには、HASPATH 演算子によるセクション等価性の検査を使用します。次の問合せを行うとします。

  ```
  ... WHERE CONTAINS (col,'HASPATH(A="virginia")'
  ```

この問合せは <A>virginia</A> のみを検索してスコア値 100 を戻します。<A>virginia state</A> は検索しません。

タグ値の等価性の検査

HASPATH を使用して、タグ値の等価性の検査を行うことができます。

```
select id from library_catalog
       where CONTAINS(text,'HASPATH (//author="Auerbach")') >0;
```

Oracle Text を使用した問合せアプリケーションの構築

Oracle Text の問合せアプリケーションを構築するには、次の手順を実行します。

1. セクション・プリファレンス・グループを作成します。セクション・グループおよび Oracle Text 索引を作成する前に、必要なロールを決定し、適切な権限を付与する必要があります。7-5 ページの「Oracle Text のユーザーおよびロール」を参照し、適切な権限を付与してください。

   データの作成および準備後、次の手順に進みます。7-22 ページの「手順 1: セクション・グループ・プリファレンスの作成」を参照してください。

2. セクションまたは停止セクションを追加します。
3. 作成したセクション・グループに基づいて Oracle Text 索引を作成します。作成したセクション・プリファレンスを使用して、Oracle Text 索引を作成します。7-28 ページの「手順 3: 手順 2 で作成したセクション・プリファレンスを使用した索引の作成」を参照してください。
4. CONTAINS 演算子を使用して、問合せアプリケーションを構築します。これで、問合せアプリケーションの構築が完了します。7-30 ページの「手順 4: 問合せ構文の作成」を参照してください。

必要なロール

まず、必要なロールを決定します。『Oracle Text リファレンス』および 7-5 ページの「Oracle Text のユーザーおよびロール」を参照して、次のとおり、適切な権限を付与します。

```
CONNECT system/manager
GRANT ctxapp to scott;
CONNECT scott/tiger
```

手順 1: セクション・グループ・プリファレンスの作成

まず、プリファレンスを作成する必要があります。この項では、PATH_SECTION_GROUP、XML_SECTION_GROUP および AUTO_SECTION_GROUP を使用して、セクション・プリファレンスを作成する方法を説明します。表 7-3 に、グループおよびその機能の概要を示します。

表 7-3 Oracle Text のセクション・グループの比較

セクション・グループ	説明
XML_SECTION_GROUP	このグループ・タイプは、XML 文書を索引付けし、XML 文書内のセクションを定義するために使用します。
AUTO_SECTION_GROUP	このグループ・タイプは、XML 文書内の開始タグ / 終了タグの各組用のゾーン・セクションを自動的に作成するために使用します。XML タグから導出されるセクション名は、XML の場合と同様、大 / 小文字が区別されます。属性セクションは、属性を含む XML タグ用に自動的に作成されます。属性セクションには、「属性名 @ タグ名」という形式の名前が付けられます。停止セクション、空タグ、処理命令およびコメントは索引付けされません。自動セクション・グループには、次の制限事項が適用されます。 ■ 自動セクション・グループには、ゾーン、フィールドまたは特殊セクションを追加できません。 ■ 自動セクション作成では、XML 文書タイプ（ルート要素）は索引付けされません。ただし、停止セクションは、文書タイプで定義できます。 ■ 接頭辞および名前空間を含めた索引付きタグの長さは、64 文字以下である必要があります。64 文字より長いタグは、索引付けされません。
PATH_SECTION_GROUP	このグループ・タイプは、XML 文書を索引付けするために使用します。AUTO_SECTION_GROUP と同じように動作します。このセクション・グループを使用すると、INPATH 演算子および HASPATH 演算子によるパス検索を実行できます。問合せは、タグ名および属性名の大 / 小文字を区別します。 **AUTO_SECTION_GROUP との類似点** ■ ドキュメントは XML であると想定されます。 ■ デフォルトで、すべてのタグおよび属性が索引付けされます。 ■ 停止セクションを追加して、特定のタグが索引付けされないようにできます。 ■ 追加できるセクションは停止セクションのみです。ゾーン・セクション、フィールド・セクションおよび特殊セクションは追加できません。 ■ XML 文書のコレクションを索引付けする場合、セクションは自動的に定義されるため、ユーザーが明示的に定義する必要はありません。 **AUTO_SECTION_GROUP との相違点** ■ 新しい INPATH 演算子および HASPATH 演算子を使用して、問合せ時にパス検索を実行できます（7-57 ページの「例 : XML ベースの会議議事録の検索」および 7-12 ページの「INPATH 演算子を使用した XML 文書でのパス検索」を参照）。 ■ 問合せ時に、タグ名および属性名の大 / 小文字が区別されます。

注意： AUTO_SECTION_GROUP または PATH_SECTION_GROUP を使用して XML 文書のコレクションを索引付けする場合、セクションは索引付け中に自動的に定義されるため、ユーザーが明示的に定義する必要はありません。

使用するセクション・グループの決定

アプリケーションに最適なセクション・グループを決定する方法は、アプリケーションによって異なります。表 7-4 に、XML 文書の索引付けに使用するセクション・グループ（XML_、AUTO_ または PATH_）を決定するための一般的なガイドライン、およびそのセクション・グループが適切な理由を示します。

表 7-4 XML_、AUTO_ または PATH_ セクション・グループ選択のガイドライン

アプリケーションの条件	XML_section_...	AUTO_section_...	PATH_section_...
XPath 検索機能を使用している	--	--	使用可
XML 文書のレイアウトおよび構造を理解し、ユーザーが検索を行うと想定されるセクションを事前定義できる	使用可	--	--
ユーザーが検索するタグを特定できない	--	使用可	--
一般的な問合せのパフォーマンス	最も速い	XML_section_... より少し遅い	AUTO_section_... より少し遅い
一般的な索引付けのパフォーマンス	最も速い	XML_section_... より少し遅い	AUTO_section_... より少し遅い
索引サイズ	最も小さい	XML_section_... より少し大きい	AUTO_section_... より少し大きい
その他の機能	1 つ以上の DTD のタグを 1 つのセクションにマップするようにマッピングを定義できます。DTD の拡張およびデータ集計に適しています。	最も単純です。マッピングを定義する必要がありません。add_stop_section を使用していくつかのセクションを無視できます。	より高度な XPath に類似した問合せのために設計されています。

XML_SECTION_GROUP を使用したセクション・プリファレンスの作成

次のコマンドは、XML_SECTION_GROUP グループ・タイプを使用して、xmlgroup というセクション・グループを作成します。

```
EXEC ctx_ddl.create_section_group('myxmlgroup', 'XML_SECTION_GROUP');
```

AUTO_SECTION_GROUP を使用したセクション・プリファレンスの作成

索引付け操作を設定し、セクション・グループ AUTO_SECTION_GROUP を使用して XML 文書からセクションを自動的に作成することができます。XML タグに対するゾーン・セクションは、Oracle が作成します。属性を持つタグには属性セクションが作成されます。これらの属性セクションには、「タグ名 @ 属性名」という形式の名前が付けられます。

次のコマンドは、AUTO_SECTION_GROUP グループ・タイプを使用して、autogroup というセクション・グループを作成します。このセクション・グループは、自動的に XML 文書内のタグからセクションを作成します。

```
EXEC ctx_ddl.create_section_group('autogroup', 'AUTO_SECTION_GROUP');
```

注意： XML セクション・グループのみに属性セクションを追加できます。AUTO_SECTION_GROUP を使用すると、属性セクションが自動的に作成されます。自動的に作成された属性セクションには、「タグ名 @ 属性名」という形式の名前が付けられます。

PATH_SECTION_GROUP を使用したセクション・プリファレンスの作成

パス・セクション検索を有効化するには、PATH_SECTION_GROUP を使用して XML 文書を索引付けします。次に例を示します。

```
EXEC ctx_ddl.create_section_group('xmlpathgroup', 'PATH_SECTION_GROUP');
```

手順 2: プリファレンスの属性の設定

XML_SECTION_GROUP 用のプリファレンスの属性を設定するには、次のプロシージャを使用します。

- Add_Zone_Section
- Add_Attr_Section
- Add_Field_Section
- Add_Special_Section

AUTO_SECTION_GROUP および PATH_SECTION_GROUP 用のプリファレンスの属性を設定するには、次のプロシージャを使用します。

- Add_Stop_Section

対応する CTX_DDL.DROP セクションおよび CTX_DDL.REMOVE セクション・コマンドがあります。

XML_SECTION_GROUP: CTX_DDL.Add_Zone_Section の使用

次に、CTX_DDL.Add_Zone_Section の構文を示します。

```
CTX_DDL.Add_Zone_Section (
 group_name      => 'my_section_group' /* whatever you called it in the preceding
section */
 section_name    => 'author' /* what you want to call this section */
 tag             => 'my_tag' /* what represents it in XML */ );
```

この場合の 'my_tag' は、<my_tag> でオープンし、</my_tag> でクローズすることを意味します。

Add_Zone_Section についてのガイドライン

次に、Add_Zone_Section についてのガイドラインを示します。

- 検索する必要がある XML 文書内の各タグごとに CTX_DDL.Add_Zone_Section をコールします。

XML_SECTION_GROUP: CTX_DDL.Add_Attr_Section の使用

次に、CTX_DDL.ADD_ATTR_SECTION の構文を示します。

```
CTX_DDL.Add_Attr_Section ( /* call this as many times as you need to describe
                              the attribute sections */
 group_name      => 'my_section_group' /* whatever you called it in the preceding
section */
 section_name    => 'author' /* what you want to call this section */
 tag             => 'my_tag' /* what represents it in XML */ );
```

この場合の 'my_tag' は、<my_tag> でオープンし、</my_tag> でクローズすることを意味します。

Add_Attr_Section についてのガイドライン

次に、Add_Attr_Section についてのガイドラインを示します。

- 次の meta_data 属性 author について考えてみます。

  <meta_data author = “John Smith” title=”How to get to Mars”>

ADD_ATTR_SECTION は、XML セクション・グループに属性セクションを追加します。このプロシージャは、XML 文書内の属性をセクションとして定義する場合に有効です。これによって、WITHIN 演算子を使用して、XML 属性テキストを検索できるようになります。

次に、section_name についてのガイドラインを示します。

- 属性テキストの WITHIN 問合せに使用される名前です。
- コロン（:）またはドット（.）文字を含むことができません。
- group_name 内で一意である必要があります。
- 大 / 小文字を区別しません。
- 64 バイト以下である必要があります。

このタグは、属性の名前を「タグ名 @ 属性名」形式で指定します。この URL の大 / 小文字は区別されます。

注意： ADD_ATTR_SECTION プロシージャでは、問合せ時に、様々なタグを同じセクション名で表すことができます。したがって、キーワード WITHIN 検索の引数として使用される名前が、実際の XML タグ名と異なる場合があります。これは、問合せ時に、様々なタグを同じ名前にマップできることを意味します。この機能によって、問合せの検索性が向上します。

XML_SECTION_GROUP: CTX_DDL.Add_Field_Section の使用

次に、CTX_DDL.Add_Field_Section の構文を示します。

```
CTX_DDL.Add_Field_Section (
 group_name     => 'my_section_group' /* whatever you called it in the preceding
section */
 section_name   => 'qq' /* what you want to call this section */
 tag            => 'my_tag' /* what represents it in XML */ );
 visible        => TRUE or FALSE );
```

Add_Field_Section についてのガイドライン

次に、Add_Field_Section についてのガイドラインを示します。

- フィールド・セクションを使用した検索は、ゾーン・セクションを使用した検索より高速です。
- VISIBLE 属性 : これは、Add_Field_Section では使用可能ですが、Add_Zone_Section では使用できません。VISIBLE が TRUE に設定されている場合、フィールド・セクション内のテキストは、囲みドキュメントの一部として索引付けされます。次に例を示します。

```
   <state> Virginia </state>
   CTX_DDL.Add_Field_Section (
group_name      => 'my_section_group'
section_name    => 'state'
tag             => 'state'
visible         => TRUE or FALSE );
```

VISIBLE が TRUE に設定されている場合、state フィールド・セクションを指定せずに Virginia を検索すると、一致する値が戻されます。

VISIBLE が FALSE に設定されている場合、state フィールド・セクションを指定せずに Virginia を検索すると、一致する値は戻されません。

属性セクションとフィールド・セクションの違い

属性セクションとフィールド・セクションは、次の点で異なります。

- **属性テキストは、表示されないとみなされます。**

  ```
  WHERE CONTAINS (..., '... jeeves',...)...
  ```

  したがって、この句では、ドキュメントが検索されません。これは、フィールド・セクションで VISIBLE が FALSE に設定されている場合と同様です。ただし、フィールド・セクションとは異なり、検索内の属性セクションは出現箇所を区別できます。次のドキュメントについて考えてみます。

  ```
  <comment author="jeeves">
      I really like Oracle Text
  </comment>
  <comment author="bertram">
      Me too
  </comment>
  ```

  次の問合せを行うと想定します。

  ```
  WHERE CONTAINS (...,'(cryil and bertram) WITHIN author', ...)...
  ```

  「jeeves」および「bertram」が同じ属性テキスト内に現れないため、この問合せは、ドキュメントを検索しません。

- 複数の「タグ名 @ 属性名」を単一のセクション名にマップできますが、**属性セクション名を、ゾーンまたはフィールド・セクション名とオーバーラップさせることはできません。**属性セクションは、デフォルト値をサポートしません。次のドキュメントについて考えてみます。

  ```
  <!DOCTYPE foo [
      <!ELEMENT foo (bar)>
      <!ELEMENT bar (#PCDATA)>
  <!ATTLIST bar
  ```

```
    rev CDATA "8i">
]>
<foo>
    <bar>whatever</bar>
</foo>
```

また、次の属性セクションについて考えてみます。

```
ctx_ddl.add_attr_section('mysg','barrev','bar@rev');
```

次の問合せを行うと想定します。

XML セマンティクスでは、bar 要素には rev 属性に対するデフォルト値がありますが、「8i within barrev」という問合せは、このドキュメントにヒットしません。

AUTO_SECTION_GROUP: CtX_DDL.Add_Stop_Section の使用

```
CtX_DDL.Add_Stop_Section (
group_name     => 'my_section_group' /* whatever you called it in the preceding
section */
section_name   => 'qq' /* what you want to call this section */ );
```

手順 3: 手順 2 で作成したセクション・プリファレンスを使用した索引の作成

プリファレンスの作成に使用したセクション・グループに基づいて、索引を作成します。

XML_SECTION_GROUP を使用した索引の作成

XML_SECTION_GROUP を使用している場合、XML 文書を索引付けするには、次の文を使用します。

```
CREATE INDEX myindex ON docs(htmlfile) INDEXTYPE IS ctxsys.context
     parameters('section group xmlgroup');
```

> **参照：** 7-29 ページの「XML_SECTION_GROUP を使用した索引の作成」を参照してください。

AUTO_SECTION_GROUP を使用した索引の作成

次の文は、AUTO_SECTION_GROUP を使用して、XML ファイルを含む列に myindex という索引を作成します。

```
CREATE INDEX myindex ON xmldocs(xmlfile) INDEXTYPE IS ctxsys.context PARAMETERS
('section group autogroup');
```

PATH_SECTION_GROUP を使用した索引の作成

PATH_SECTION_GROUP を使用している場合、XML 文書を索引付けするには、次の文を使用します。

```
CREATE INDEX myindex ON xmldocs(xmlfile) INDEXTYPE IS ctxsys.context PARAMETERS
('section group xmlpathgroup');
```

> **参照：** CTX_DDL の詳細は、『Oracle Text リファレンス』を参照してください。

例 7-3 XML_SECTION_GROUP を使用した索引の作成

```
EXEC ctx_ddl_create_section_group('myxmlgroup', 'XML_SECTION_GROUP');

/* ADDING A FIELD SECTION */
EXEC ctx_ddl.Add_Field_Section /* THIS IS KEY */
  (  group_name   =>'my_section_group',
     section_name =>'author',/* do this for EVERY tag used after "WITHIN" */
     tag          =>'author'
  );

EXEC ctx_ddl.Add_Field_Section /* THIS IS KEY */
      (  group_name   =>'my_section_group',
         section_name =>'document',/*do this for EVERY tag after "WITHIN" */
         tag          =>'document'
      );

  ...
/
/* ADDING AN ATTRIBUTE SECTION */
EXEC ctx_ddl.add_attr_section('myxmlgroup', 'booktitle', 'book@title');

/* The more sections you add to your index, the longer your search will take.*/
/* Useful for defining attributes in XML documents as sections. This allows*/
/* you to search XML attribute text using the WITHIN operator.*/
/*  The section name:
/* ** Is used for WITHIN queries on the attribute text.
   ** Cannot contain the colon (:) or dot (.) characters.
   ** Must be unique within group_name.
   ** Is case-insensitive.
   ** Can be no more than 64 bytes.
   **  The tag specifies the name of the attribute in tag@attr format. This is
       case-sensitive. */
/* Names used as arguments of the keyword WITHIN can be different from the
   actual XML tag names. Many tags can be mapped to the same name at query
   time.*/
```

```
/* Call CTX_DDL.Add_Zone_Section for each tag in your XML document that you need to
search on. */

EXEC ctx_ddl.add_zone_section('myxmlgroup', 'mybooksec', 'mydocname(book)');

CREATE INDEX my_index ON my_table ( my_column )
  INDEXTYPE IS ctxsys.context
  PARAMETERS ( 'SECTION GROUP my_section_group' );

SELECT my_column FROM my_table
  WHERE CONTAINS(my_column, 'smith WITHIN author') > 0;
```

手順 4: 問合せ構文の作成

問合せ文で CONTAINS 演算子を使用する方法については、7-6 ページの「CONTAINS 演算子を使用した問合せ」を参照してください。

属性セクション内での問合せ

セクション・グループ・タイプとして XML_SECTION_GROUP または AUTO_SECTION_GROUP を使用して索引付けを行っている場合、属性セクション内で問合せを行うことができます。

次の XML 文書があると想定します。

```
<book title="Tale of Two Cities">It was the best of times.</book>
```

title@book セクションを属性セクションのタイトルとして定義できます。これは、CTX_DLL.Add_Attr_Section プロシージャを使用して行うか、または ALTER INDEX を使用して索引付けした後に動的に行います。

注意： AUTO_SECTION_GROUP を使用して XML 文書を索引付けする場合、システムは、自動的に属性セクションを作成し、それに「属性名 @ タグ名」という形式の名前を付けます。

XML_SECTION_GROUP を使用する場合、CTX_DDL.Add_Attr_Section を使用して、属性セクションに任意の名前を付けることができます。

属性セクション・タイトル内で Tale を検索するには、次の問合せを発行します。

```
WHERE CONTAINS (...,'Tale WITHIN title', ...)
```

TITLE 属性セクションをこのように定義し、ドキュメント・セットを索引付けした場合、次のとおり、XML 属性テキストを問い合せることができます。

```
... WHERE CONTAINS (...,'Cities WITHIN booktitle', ....)...
```

AUTHOR 属性セクションをこのように定義し、ドキュメント・セットを索引付けした場合、次のとおり、XML 属性テキストを問い合せることができます。

```
... WHERE 'England WITHIN authors'
```

例 7-4 XML 文書の問合せ

この例では、次の操作を行います。

1. res_xml 表の作成および移入
2. 索引 section_group およびプリファレンスの作成
3. プリファレンスのパラメータ化
4. res_xml に対するテスト問合せの実行

```
drop table res_xml;

CREATE TABLE res_xml (
  pk            NUMBER PRIMARY KEY ,
  text          CLOB
  ) ;

insert into res_xml values(111,
  'ENTITY chap8 "Chapter 8, <q>Keeping it Tidy: the XML Rule Book </q>"> this is the
document section');
commit;

---
--- script to create index on res_xml
---

--- cleanup, in case we have run this before
DROP INDEX res_index ;
EXEC CTX_DDL.DROP_SECTION_GROUP ( 'res_sections' ) ;

--- create a section group
BEGIN
  CTX_DDL.CREATE_SECTION_GROUP ( 'res_sections', 'XML_SECTION_GROUP' ) ;
  CTX_DDL.ADD_FIELD_SECTION ( 'res_sections', 'chap8', '<q>') ;
END ;
/

begin
```

```
  ctx_ddl.create_preference
    (
      preference_name => 'my_basic_lexer',
      object_name     => 'basic_lexer'
    );
  ctx_ddl.set_attribute
    (
      preference_name => 'my_basic_lexer',
      attribute_name  => 'index_text',
      attribute_value => 'true'
    );
  ctx_ddl.set_attribute
    (
      preference_name => 'my_basic_lexer',
      attribute_name  => 'index_themes',
      attribute_value => 'false');
end;
/

CREATE INDEX res_index
  ON res_xml(text)
  INDEXTYPE IS ctxsys.context
  PARAMETERS ( 'lexer my_basic_lexer SECTION GROUP res_sections' ) ;
```

前述の索引を次のようなテスト問合せでテストします。

```
SELECT pk FROM res_xml WHERE CONTAINS( text, 'keeping WITHIN chap8' )>0 ;
```

例 7-5 索引の作成およびテキスト問合せの実行

```
drop table explain_ex;

create table explain_ex
  (
    id        number primary key,
    text      varchar(2000)
  );

insert into explain_ex ( id, text )
  values ( 1, 'thinks thinking thought go going goes gone went' || chr(10) ||
              'oracle orackle oricle dog cat bird'              || chr(10) ||
              'President Clinton' );
insert into explain_ex ( id, text )
  values ( 2, 'Last summer I went to New England'               || chr(10) ||
              'I hiked a lot.'                                  || chr(10) ||
              'I camped a bit.' );
commit;
```

例 7-6 テキスト問合せ式に ABOUT を使用するテキスト問合せ

```
Set Define Off
select text
  from explain_ex
  WHERE CONTAINS ( text,
  '( $( think & go ) , ?oracle ) & ( dog , ( cat & bird ) ) & about(mammal
                                                    during Bill Clinton)' ) > 0;
select text
  from explain_ex
  WHERE CONTAINS ( text, 'about ( camping and hiking in new england )'  ) > 0;
```

例 7-7 AUTO_SECTION_GROUP を使用した索引の作成

```
ctx_ddl_create_section_group('auto', 'AUTO_SECTION_GROUP');

CREATE INDEX myindex ON docs(xmlfile_column)
    INDEXTYPE IS ctxsys.context
    PARAMETERS ('filter ctxsys.null_filter SECTION GROUP auto');

SELECT xmlfile_column FROM docs
    WHERE CONTAINS (xmlfile_column, 'virginia WITHIN title')>0;
```

例 7-8 PATH_SECTION_GROUP を使用した索引の作成

```
EXEC ctx_ddl.create_section_group('xmlpathgroup', 'PATH_SECTION_GROUP');

CREATE INDEX myindex ON xmldocs(xmlfile_column)
    INDEXTYPE IS ctxsys.context
    PARAMETERS ('section group xmlpathgroup');

SELECT xmlfile_column FROM xmldocs
... WHERE CONTAINS (column, 'Tale WITHIN title@book')>0;
```

例 7-9 XML_SECTION_GROUP および add_attr_section を使用した問合せの支援

次のとおり、TITLE 属性を指定して BOOK タグを定義する XML ファイルについて考えてみます。

```
<BOOK TITLE="Tale of Two Cities">
It was the best of times. </BOOK>
<Author="Charles Dickens">
Born in England in the town, Stratford_Upon_Avon </Author>
```

次に、CTX_DDL.ADD_ATTR_SECTION 構文を示します。

```
CTX_DDL.Add_Attr_Section ( group_name, section_name, tag );
```

TITLE 属性を属性セクションとして定義するには、次のとおり、XML_SECTION_GROUP を作成し、属性セクションを定義します。

```
ctx_ddl_create_section_group('myxmlgroup', 'XML_SECTION_GROUP');
ctx_ddl.add_attr_section('myxmlgroup', 'booktitle', 'book@title');
ctx_ddl.add_attr_section('myxmlgroup', 'authors', 'author');
end;
```

注意：

- Oracle は、セクション・グループの作成時に指定する group_type パラメータから、終了タグがどのように見えるかを認識しています。指定する開始タグは、セクション・グループ内で一意にしてください。
- セクション名は、タグ間で一意である必要はありません。検索に対し詳細を透過的にして、同じセクション名を複数のタグに割り当てることができます。

問合せ結果の表示

Oracle Text の問合せアプリケーションを使用すると、問合せによって戻されたドキュメントを表示できます。通常、ヒットリストからドキュメントを選択し、アプリケーションがそのドキュメントを特定の形式で表示します。

Oracle Text を使用すると、ドキュメントを様々な方法でレンダリングすることができます。たとえば、問合せ用語をハイライト表示させることができます。ハイライト表示できる問合せ用語は、英語の場合、ワード問合せのワード、または ABOUT 問合せのテーマです。このレンダリングを行うには、CTX_DOC.HIGHLIGHT または CTX_DOC.MARKUP プロシージャを使用します。

また、CTX_DOC.THEMES という PL/SQL パッケージを使用しても、ドキュメントからテーマ情報を取得できます。この他にも、問合せ結果を表示するためのいくつかの CTX_DOC プロシージャがあります。

INPATH では、テーマの動作がサポートされていません。

参照： CTX_DOC パッケージの詳細は、『Oracle Text リファレンス』を参照してください。

XMLType の索引付け

XML を格納するための Oracle9*i* データベースのデータ型である XMLType は、主要なデータベース機能の 1 つです。

必須のクエリー・リライト権限

> **注意：** これらの権限は、Oracle9*i* データベース リリース 1（9.0.1）でのみ必要です。

この型で Oracle Text 索引を作成する前に、いくつかのデータベース権限が必要です。

1. 索引を作成するユーザーには、QUERY REWRITE 権限が必要です。

   ```
   GRANT QUERY REWRITE TO <user>
   ```

   この権限を持ってない場合、索引の作成は失敗し、次のメッセージが表示されます。

   ```
   ORA-01031: 権限が不足しています。
   ```

   <user> は、索引を作成するユーザーです。そうでない場合、索引を所有するデータベース・スキーマに権限を付与する必要はありません。

2. query_rewrite_enabled は true、query_rewrite_integrity は trusted に指定する必要があります。これらの設定は、init.ora ファイルに追加できます。

   ```
   query_rewrite_enabled=true
   query_rewrite_integrity=trusted
   ```

   または、次のとおりセッションを変更します。

   ```
   ALTER SESSION SET query_rewrite_enabled=true;
   ALTER SESSION SET query_rewrite_integrity=trusted;
   ```

   この設定を行わない場合、問合せは失敗し、次のメッセージが表示されます。

   ```
   DRG-10599: 列に索引が作成されていません。
   ```

XMLType は実際にはオブジェクトであり、このオブジェクトにアクセスするにはファンクションを使用するため、これらの権限が必要です。つまり、XMLType 列の Oracle Text 索引は、実際にはその型の getclobval() メソッドのファンクション索引です。これらは、ファンクション索引を使用するために必要な標準の権限ですが、ファンクション B ツリー索引とは異なり、統計を計算する必要はありません。

注意：『Oracle9*i* SQL リファレンス』の「CREATE INDEX」に、次のとおり記載されています。

自表の自分のスキーマにファンクション索引を作成する場合、従来索引の作成の前提条件の他に、QUERY REWRITE システム権限が必要です。

別のスキーマまたは別のスキーマの表に索引を作成する場合は、GLOBAL QUERY REWRITE 権限が必要です。どちらの場合も、表の所有者は、ファンクション索引で使用されるファンクションについての EXECUTE オブジェクト権限が必要です。

また、問合せでファンクション索引を使用する場合は、QUERY_REWRITE_ENABLED パラメータを TRUE に、QUERY_REWRITE_INTEGRITY パラメータを TRUSTED に設定する必要があります。

デフォルトの CTXSYS.PATH_SECTION_GROUP に設定されたシステム・パラメータ

XMLType 列が検出され、パラメータ文字列にセクション・グループが指定されていない場合、デフォルトのシステムは、新しいシステム・パラメータ DEFAULT_XML_SECTION を調べて、指定されているセクション・グループを使用します。インストール時、このシステム・パラメータは、CTXSYS.PATH_SECTION_GROUP に設定されています。これは、デフォルトのパス・セクショナです。

XMLType のデフォルトのフィルタ・システム・パラメータは DEFAULT_FILTER_TEXT で、これは、デフォルトでは INSO フィルタが使用されないことを意味します。

他の Oracle Text 索引と同様に機能する XMLType 索引

データベース権限および特別なデフォルト・セクション・グループ・システム・パラメータを除き、XMLType 列の索引は、他の Oracle Text 索引と同様に機能します。

例 7-10 XMLType 列での Text 索引の作成

次に単純な例を示します。

```
connect ctxsys/ctxsys
GRANT QUERY REWRITE TO xtest;
connect xtest/xtest

CREATE TABLE xtest(doc sys.xmltype);
INSERT INTO xtest VALUES (sys.xmltype.createxml('<A>simple</A>'));

CREATE INDEX xtestx ON xtest(doc)
   INDEXTYPE IS ctxsys.context;
```

```
ALTER SESSION SET query_rewrite_enabled = true;
ALTER SESSION SET query_rewrite_integrity = trusted;

SELECT a.doc.getclobval() FROM xtest a
   WHERE CONTAINS (doc, 'simple INPATH(A)')>0;
```

Oracle XML DB での Oracle Text の使用

URIType 列での Oracle Text 索引の作成

Oracle Text を使用して、Oracle9*i* データベース内で URIType 列をネイティブに索引付けできます。特別なデータ・ストアは不要です。

例 7-11 URIType 列での Oracle Text 索引の作成

次に例を示します。

```
CREATE TABLE table uri_tab ( url sys.httpuritype);

INSERT INTO uri_tab VALUES
    (sys.httpuritype.createUri('http://www.oracle.com'));

CREATE INDEX urlx ON uri_tab(url) INDEXTYPE IS ctxsys.context;

SELECT url FROM uri_tab WHERE CONTAINS(url, 'Oracle')>0;
```

表 7-5 に、列型が URIType の場合に Oracle Text 索引付け用のデフォルトのプリファレンス名に使用されるシステム・パラメータを示します。

表 7-5 Oracle Text 索引付け用の URIType 列のデフォルトのプリファレンス名

URIType 列	デフォルトのプリファレンス名
DATASTORE	DEFAULT_DATASTORE
FILTER	DEFAULT_FILTER_TEXT
SECTION GROUP	DEFAULT_SECTION_HTML
LEXER	DEFAULT_LEXER
STOPLIST	DEFAULT_STOPLIST
WORDLIST	DEFAULT_WORDLIST
STORAGE	DEFAULT_STORAGE

XML データの問合せ CONTAINS または existsNode() の使用

Oracle9*i* データベース リリース 1（9.0.1）では、Oracle Text の PATH_SECTION_GROUP 演算子、INPATH() 演算子および HASPATH() 問合せ演算子が導入されています。これらの演算子によって、CONTAINS 演算子を使用した、XPath と同様のテキスト問合せ検索を XML 文書で実行できます。ただし、CONTAINS は、XPath 機能のサブセットのみをサポートします。また、CONTAINS 演算子と existsNode() 関数の間に重要なセマンティックの違いがあります。

SQL 関数 existsNode、extract() および extractValue()（および対応する XMLType のメンバー関数）は、XPath を完全にサポートしています。今回のリリースの Oracle9*i* データベースでは、XPath に対する新しい拡張関数が導入され、全文検索がサポートされています。

注意： 今回のリリースでは、Oracle Text の CONTAINS() および existsNode() の検索でのテーマ問合せはサポートされていません。

表 7-6 に、XMLType データを検索するための CONTAINS() および existsNode() の機能を示し、それぞれを比較します。

表 7-6 CONTAINS() および existsNode() を使用した XMLType データの検索

機能	CONTAINS()	existsNode()
XPath への準拠	--	--
述語サポート	--	--
■ 文字列の等価性	可	可
■ 数値の等価性	不可	可
■ 範囲述語	不可	可
■ XPath 機能	不可	可
■ 空白	不可	可
■ 名前空間	不可	可
■ 値の大 / 小文字の区別	不可	可
■ エンティティの処理	不可	可
■ Parent-ancestor 軸および sibling 軸	不可	可
■ ワイルド・カードによる属性検索 (*/@A や ../ など)	可	可
■ XML Schema または DTD 情報の使用	不可	可

表 7-6 CONTAINS() および existsNode() を使用した XMLType データの検索 （続き）

機能	CONTAINS()	existsNode()
■ 空要素による不適切な一致の可能性	可	不可
同期	--	--
■ DML	不可	不可（CTXXPath） 可（他の索引）
■ 問合せ	不可	可
言語検索機能	（INPATH() の場合）	可（ora:contains() の使用）
索引タイプ	ctxsys.context	ctxsys.ctxxpath
クエリー・リライト	不可	可（XML Schema に基づき、オブジェクト・リレーショナル形式で格納されている場合）
ファンクション索引	不可	可（existsNode() 式および extractValue() 式にファンクション索引を作成可能）
ConText 索引が作成済の場合にサポートされる機能	--	--
■ About	可	不可
■ ハイライト表示	可	不可
通常のテキスト検索	全文検索のサポート	ora:contains を使用した制限付きのテキスト検索のサポート
通常の XPath 検索	制限付きの XPath 検索 非同期	XPath の全文検索 同期

ora:contains を使用した XPath での全文検索関数

XPath は、substring() や CONTAINS() などの一連の組込みテキスト関数を指定しますが、Oracle の全文検索関数ほど強力ではありません。Oracle では、より豊富な一連のテキスト検索機能を使用可能にするために、新しい XPath 拡張関数が定義されています。この拡張関数は、Oracle XML DB 名前空間の http://xmlns.oracle.com/xdb で定義されています。

これらの関数は、XMLType インスタンスで動作する existsNode()、extract() および extractValue() 関数のコンテキストに表示される XPath 問合せ内で使用できます。

注意： CTX_DDL パッケージの他のプロシージャと同様に、CTX_DDL.CREATE_POLICY() プロシージャを実行するには、CTXAPP 権限が必要です。

---

ora:contains の機能

ora:contains の機能は次のとおりです。

- テキスト検索拡張関数は、ステミング、ファジー検索、近接検索など、ほぼすべてのテキスト問合せ演算子をサポートしています。
- これらの関数には、評価用の ConText 索引は必要ありません。
- これらの関数によって計算されたスコアの値は、通常の（CONTAINS SQL 演算子を使用した）索引ベースの問合せ処理とは異なる場合があります。ドキュメントの統計が存在しないため、各用語の重みは 10 に固定されます。ワード検索のスコアは、10 の倍数の一致した文字列になります。スコアが 100 を超える場合、100 に切り捨てられます。これは、あいまい一致した用語にも適用されます。

ora:contains の構文

次に、ora:contains 関数の構文を示します。

```
number contains(string, string, string?, string?)
```

パラメータは次のとおりです。

- string: 最初の引数は入力テキスト値です。
- string: 2 番目の引数はテキスト問合せ文字列です。
- string?: オプションの 3 番目の引数はポリシー名です。
- string?: オプションの 4 番目の引数はポリシーの所有者です。

Oracle XML DB 名前空間の contains 拡張関数は、最初の引数として入力テキスト値を、2 番目の引数としてテキスト問合せ文字列を取ります。これは、0 ～ 100 の数字のスコア値を戻します。

オプションの 3 番目および 4 番目の引数を使用して、テキスト問合せの処理に使用される CTX ポリシーの名前および所有者を指定できます。3 番目の引数が指定されていない場合、CTX ポリシー名は、デフォルトで CTXSYS が所有する DEFAULT_POLICY_ORACONTAINS に設定されます。4 番目の引数が指定されていない場合、ポリシーの所有者は現在のユーザーであるとみなされます。

ora:contains の例

xmltab 表に XML 文書が含まれているとします。この文書には chapter が埋め込まれ、各 chapter には title および body が含まれています。

```
<book>
  <chapter>
    <title>...</title>
    <body>...</body>
  </chapter>
  <chapter>
    <title>...</title>
    <body>...</body>
  </chapter>
  ...
</book>
```

例 7-12 ora:contains を使用したテキスト問合せ文字列の検索

指定されたテキスト問合せ文字列が chapter の本体に含まれる book を検索します。

```
select * from xmltab x where
  existsNode(value(x), '/book/chapter[ora:contains(body,"dog OR cat")>0]',
  'xmlns:ora="http://xmlns.oracle.com/xdb"') = 1;
```

例 7-13 ora:contains および extract() を使用したテキスト問合せ文字列の検索

指定されたテキスト問合せ文字列が本体に含まれる chapter を抽出します。

```
select extract(value(x),
          '/book/chapter[ora:contains(body,"dog OR cat")>0]',
          'xmlns:ora="http://xmlns.oracle.com/xdb"')
from xmltab x;
```

参照： 7-42 ページの「Oracle XML DB: ora:contains() のポリシーの作成」を参照してください。

Oracle XML DB: ora:contains() のポリシーの作成

この項では、ora:contains() のポリシーを作成、更新および削除するための構文および例を示します。

次の CTX_DDL プロシージャによって、ora:contains() で使用されるポリシーが作成、更新および削除されます。

ポリシーは、ora:contains() を処理するために使用される一連のプリファレンスです。

> **参照：** Oracle Text プリファレンスの詳細は、次のマニュアルを参照してください。
>
> - 『Oracle Text アプリケーション開発者ガイド』
> - 『Oracle Text リファレンス』

表 7-7 に、XPath 検索で使用するポリシーを作成、更新および削除するための CTX_DDL ファンクションを示します。

表 7-7 ポリシーを作成、更新および削除するための CTX_DDL 構文

CTX_DDL ファンクション	説明
CREATE_POLICY CTX_DDL.create_policy(policy_name in varchar2, filter in varchar2 default NULL, section_group in varchar2 default NULL, lexer n varchar2 default NULL, stoplist in varchar2 default NULL, wordlist in varchar2 default NULL);	ポリシーを定義します。 引数 : policy_name - 新しいポリシーの名前。 filter - 使用するフィルタ・プリファレンス（将来の使用のために予約されています）。 section_group - 使用するセクション・グループ（現在、NULL_SECTION_GROUP のみがサポートされています）。 lexer - 使用するレクサー・プリファレンス。テーマ索引を有効にする必要はありません。 stoplist - 使用するストップリスト・プリファレンス。 wordlist - 使用するワードリスト・プリファレンス。
UPDATE_POLICY CTX_DDL.update_policy(policy_name in varchar2, filter in varchar2 default NULL, section_group in varchar2 default NULL, lexer in varchar2 default NULL, stoplist in varchar2 default NULL, wordlist in varchar2 default NULL);	指定されたプリファレンスを置き換えて、ポリシーを更新します。 引数 : policy_name - ポリシーの名前。 filter - 使用する新しいフィルタ・プリファレンス（将来の使用のために予約されています）。 section_group - 使用する新しいセクション・グループ（現在、NULL_SECTION_GROUP のみがサポートされています）。 lexer - 新しいレクサー・プリファレンス。テーマ索引を有効にする必要はありません。 stoplist - 使用する新しいストップリスト・プリファレンス。 wordlist - 使用する新しいワードリスト・プリファレンス。
DROP_POLICY CTX_DDL.drop_policy(policy_name in varchar2);	ポリシーを削除します。 引数 : policy_name - ポリシーの名前。

例 7-14 ora:contains のポリシーの作成

mylex という名前のレクサー・プリファレンスを作成します。

```
begin
   ctx_ddl.create_preference('mylex', 'BASIC_LEXER');
   ctx_ddl.set_attribute('mylex', 'printjoins', '_-');
   ctx_ddl.set_attribute ( 'mylex', 'index_themes', 'NO');
   ctx_ddl.set_attribute ( 'mylex', 'index_text', 'YES');
end;
```

mystop という名前のストップリスト・プリファレンスを作成します。

```
begin
  ctx_ddl.create_stoplist('mystop', 'BASIC_STOPLIST');
  ctx_ddl.add_stopword('mystop', 'because');
  ctx_ddl.add_stopword('mystop', 'nonetheless');
  ctx_ddl.add_stopword('mystop', 'therefore');
end;
```

mywordlist という名前のワードリスト・プリファレンスを作成します。

```
begin
 ctx_ddl.create_preference('mywordlist', 'BASIC_WORDLIST');
 ctx_ddl.set_attribute('mywordlist','FUZZY_MATCH','ENGLISH');
 ctx_ddl.set_attribute('mywordlist','FUZZY_SCORE','0');
 ctx_ddl.set_attribute('mywordlist','FUZZY_NUMRESULTS','5000');
 ctx_ddl.set_attribute('mywordlist','SUBSTRING_INDEX','TRUE');
 ctx_ddl.set
_attribute('mywordlist','STEMMER','ENGLISH');
end;

exec ctx_ddl.create_policy('my_policy', NULL, NULL, 'mylex', 'mystop',
'mywordlist');
```

または

```
exec ctx_ddl.create_policy(policy_name => 'my_policy',
                           lexer => 'mylex',
                           stoplist => 'mystop',
                           wordlist => 'mywordlist');
```

その後、独自に定義したポリシーを使用して、次の existsNode() 問合せを発行できます。

```
select * from xmltab x where
   existsNode(value(x),
     '/book/chapter[ora:contains(body,"dog OR cat", "my_policy")>0]',
     'xmlns:ora="http://xmlns.oracle.com/xdb"') = 1;
```

また、次のコマンドを使用して、ポリシーを更新することもできます。

```
exec ctx_ddl.update_policy(policy_name => 'my_policy',
                           lexer => 'my_new_lex');
```

次のコマンドを使用して、ポリシーを削除できます。

```
exec ctx_ddl.drop_policy(policy_name => 'my_policy');
```

他のユーザーのポリシーを使用した問合せ

別のユーザーのポリシー（この場合は Scott のポリシー）を使用して、existsNode() 問合せを発行することもできます。

例 7-15 他のユーザーのポリシーの問合せ

```
select * from xmltab x where
  existsNode(value(x),
    '/book/chapter[ora:contains(body,"dog OR cat", "Scotts_policy","Scott")>0]',
    'xmlns:ora="http://xmlns.oracle.com/xdb"') = 1;
```

Oracle XML DB: existsNode() に対する CTXXPATH 索引の使用

CONTAINS 演算子とは異なり、SQL 関数 existsNode() では、ConText 索引を使用して評価を高速化することはできません。existsNode() の XPath 検索のパフォーマンスを向上するために、このリリースでは、新しい索引タイプ CTXXPATH が導入されています。

CTXXPATH 索引は、Oracle Text で提供される新しい索引タイプです。これは、existsNode() の処理に対する 1 次フィルタとして機能するように設計されています。existsNode() 関数によって生成される結果のスーパーセットを生成します。

ConText 索引によって XPath 検索が実行可能な場合に CTXXPATH が必要な理由

既存の ConText 索引タイプには、XPath 検索機能が備わっていますが、ConText 索引タイプには次の制約があります。

- ConText 索引を existsNode() の 1 次フィルタとして使用可能にする場合
  - PATH_SECTION_GROUP を使用して、索引を作成する必要があります。
  - USER_LEXER プリファレンスまたは MULTI_LEXER プリファレンスを使用して、索引を作成することはできません。
  - DIRECT DATASTORE を使用して、索引を作成する必要があります。
  - NULL FILTER を使用して、索引を作成する必要があります。

  これによって、ConText 索引タイプが提供する言語検索機能が制限されます。

- ConText 索引は非同期で、existsNode() と同じトランザクション・セマンティクスには従いません。
- ConText 索引は、名前空間またはユーザー定義のエンティティを処理しません。

これらのすべての制限事項を考慮して、CTXXPATH 索引タイプは、existsNode() の 1 次フィルタ処理の目的を果たすために設計されました。XMLType 列に必要なプリファレンスを使用して、ConText 索引を作成できます。これを使用すると、CONTAINS 演算子が高速化されます。また、CTXXPATH 索引を作成して、existsNode() の処理を高速化できます。

CTXXPATH 索引タイプ

次に、CTXXPATH 索引タイプの特長を示します。

- この索引は、existsNode() の処理を高速化するためにのみ使用できます。これは、existsNode() 関数の 1 次フィルタとして機能します。existsNode() が提供する結果のスーパーセットを提供します。
- この索引は、一部の XPath 式のみを処理できます。索引でサポートされていない XPath 式のリストは、7-48 ページの「適切なプランの選択 : existsNode() の処理における CTXXPATH 索引の使用」を参照してください。
- サポートされているパラメータは、STORAGE パラメータのみです。7-47 ページの「CTX_DDL 文を使用した CTXXPATH 記憶域作業環境の作成」を参照してください。
- DML は非同期です。ユーザーは、特別な DDL コマンドを発行して、ConText 索引の DML と同様に、DML を同期化する必要があります。
- DML 非同期であるにもかかわらず、この索引は、有効な結果のスーパーセットを戻すための要件を保証するために、その結果セットの一部として索引付けされていない行も戻すことによって、existsNode() のトランザクション・セマンティクスに従います。

CTXXPATH 索引の作成

ConText 索引を作成する場合と同じ方法で、CTXXPATH 索引を作成します。構文は、ConText 索引の構文と同じです。

```
CREATE INDEX [schema.]index ON [schema.]table(XMLType column)
   INDEXTYPE IS ctxsys.CTXXPATH [PARAMETERS(paramstring)];
```

この場合の paramstring は次のとおりです。

```
paramstring = '[storage storage_pref] [memory memsize] [populate | nopopulate]'
```

例 7-16 CTXXPATH 索引の作成

次に例を示します。

```
CREATE INDEX xml_idx ON xml_tab(col_xml) indextype is ctxsys.CTXXPATH;
```

または

```
CREATE INDEX xml_idx ON xml_tab(col_xml) indextype is ctxsys.CTXXPATH
         PARAMETERS('storage my_storage memory 40M');
```

索引は、existsNode() を使用した問合せを高速化するためにのみ使用できます。

```
SELECT xml_id FROM xml_tab x WHERE
     x.col_xml.existsnode('/book/chapter[@title="XML"]')>1;
```

> **参照：** existsNode() の使用の詳細は、第 4 章「XMLType の使用」を参照してください。

CTX_DDL 文を使用した CTXXPATH 記憶域作業環境の作成

CTXXPATH 索引タイプに対して使用可能なプリファレンスは、STORAGE のみです。ConText 索引タイプの場合と同じ方法で、STORAGE プリファレンスを作成します。

例 7-17 CTXXPATH 索引の記憶域作業環境の作成

次に例を示します。

```
begin
ctx_ddl.create_preference('mystore', 'BASIC_STORAGE');
ctx_ddl.set_attribute('mystore', 'I_TABLE_CLAUSE',
                      'tablespace foo storage (initial 1K)');
ctx_ddl.set_attribute('mystore', 'K_TABLE_CLAUSE',
                      'tablespace foo storage (initial 1K)');
ctx_ddl.set_attribute('mystore', 'R_TABLE_CLAUSE',
                      'tablespace foo storage (initial 1K)');
ctx_ddl.set_attribute('mystore', 'N_TABLE_CLAUSE',
                      'tablespace foo storage (initial 1K)');
ctx_ddl.set_attribute('mystore', 'I_INDEX_CLAUSE',
                      'tablespace foo storage (initial 1K)');
end;
```

CTXXPATH 索引のパフォーマンス・チューニング 索引の同期化および最適化

DML を同期化するには、CTX_DDL パッケージに含まれる SYNC_INDEX プロシージャを使用します。

例 7-18 CTXXPATH 索引の最適化

次に例を示します。

```
exec ctx_ddl.sync_index('xml_idx');
```

CTXXPATH 索引を最適化するには、CTX_DDL パッケージに含まれる OPTIMIZE_INDEX() プロシージャを使用します。次に例を示します。

```
exec ctx_ddl.optimize_index('xml_idx', 'FAST');
```

または

```
exec ctx_ddl.optmize_index('xml_idx', 'FULL');
```

参照： 次のマニュアルを参照してください。

- 『Oracle Text アプリケーション開発者ガイド』
- 『Oracle Text リファレンス』

適切なプランの選択 : existsNode() の処理における CTXXPATH 索引の使用

existsNode() の処理を高速化するために、CTXXPATH 索引が常に使用されるとはかぎりません。次に、Oracle Text 索引が existsNode() で使用されない場合の理由を示します。

- コストベース・オプティマイザが、CTXXPATH 索引を 1 次フィルタとして使用することは高コストであると判断した。
- XPath 式は、CTXXPATH 索引で処理できない。次に、CTXXPATH 索引で処理できない XPath 構造体のリストを示します。
  - XPATH 関数
  - 数値範囲演算子
  - 数値の等価性
  - 算術演算子
  - Union 演算子「|」
  - 属性の存在
  - 位置指定 / 索引述語（/A/B[5]）
  - Parent 軸および Sibling 軸

- *、//、.. の後に続く属性（'/A/*/@attr'、'/A//@attr'、'/A//../@attr'）
- パス式の終わりの「.」または「*」
- 「.」または「*」の後に続く述語
- 次の制限事項付きで、文字列リテラルの等価性がサポートされています。
  - 左側に、パスを指定する必要があります（.="dog" のように「.」自体は使用できません）。
  - 右側にはリテラルを指定する必要があります。

オプティマイザによる existsNode() 関数のコストおよび選択性の評価を向上させるには、ANALYZE コマンドまたは DBMS_STATS パッケージを使用して、CTXXPATH 索引の統計を収集する必要があります。次のように、ANALYZE コマンドを使用して、索引を分析し、統計を計算できます。

```
ANALYZE INDEX myPathIndex COMPUTE STATISTICS;
```

または、単純に表全体を分析できます。

```
ANALYZE TABLE XMLTAB COMPUTE STATISTICS;
```

Oracle Text の使用 : 高度な使用方法

次の項では、XML データ検索を微調整するための、Oracle Text の高度な使用方法について説明します。

INPATH/HASPATH テキスト演算子に対するハイライト表示のサポート

Oracle Text は、ドキュメントに対するテキスト問合せのオフセットおよび長さをハイライト表示する CTX_DOC.HIGHLIGHT() プロシージャを提供します。これらのオフセットおよび長さは、ワード問合せ、句問合せまたは ABOUT 問合せの条件を満たすドキュメント内の用語に対して生成されます。Oracle9*i* リリース 2（9.2.0.2）では、Oracle Text でのハイライト表示のサポートが拡張され、INPATH および HASPATH 演算子がサポートされます。

INPATH での XML 文書のハイライト表示

INPATH では、CTX_DOC.HIGHTLIGHT() によって、WITHIN 演算子の場合と同様に、INPATH 演算子の直前の子に対してオフセットおよび長さが計算されます。これは、パスの子が要素を指す場合のみに適用されます。たとえば、次のテキスト問合せ式について考えてみます。

```
'txt INPATH(/A/B)'  or
'txt INPATH(/A/B[@attr="atxt" and .="Btxt"])'
```

この場合、CTX_DOC.HIGHTLIGHT() によって、INPATH 問合せの条件を満たすドキュメント内で 'txt' が出現するたびに、オフセットおよび長さが生成されます。

パスの子が属性を指す場合、ハイライト表示は行われません。たとえば、次のテキスト問合せ式について考えてみます。

```
'atxt INPATH(/A/B/@attr)'
```

この場合、ハイライト表示に関する情報は生成されません。

HASPATH での XML 文書のハイライト表示

HASPATH では、パス・オペランドが要素を指す場合、CTX_DOC.HIGHTLIGHT() によって、XPath 式が指す要素の本体に対してオフセットおよび長さが計算されます。たとえば、次のテキスト問合せ式について考えてみます。

```
'HASPATH(/A/B)' or
'HASPATH(/A/B[@att="atxt"])'
```

この場合、オフセットおよび長さは、/A/B が指す要素の本体に対して計算されます。

パス・オペランドが 'HASPATH(/A/B/@Battr)' などの属性を指す場合、ハイライト表示に関する情報は生成されません。

オペランドによって WITHIN-EQUAL/SECTION-EQUAL テストが行われる場合、CTX_DOC.HIGHTLIGHT() によって、「=」の前に示すパスの子が指す要素のオフセットおよび長さが出力されます。「=」の前に示すパスの子が属性を指す場合、ハイライト表示に関する情報は生成されません。たとえば、次のテキスト問合せ式について考えてみます。

```
'HASPATH(/A/B = "ABtxt")' or
'HASPATH(/A/B[@att="atxt"]= "ABtxt")'
```

この場合、/A/B が指す要素の本体に対してオフセットおよび長さが生成されます。一方、次のテキスト問合せ式について考えてみます。

```
'HASPATH(/A/B/@att = "atxt")'
```

この場合、パスの子である '/A/B/@att' が要素ではなく属性を指すため、ハイライト表示に関する情報は生成されません。

注意： CTX_DOC.HIGHTLIGHT() では、オフセットおよび長さが計算および出力されます。一方、CTX_DOC.MARKUP() では、指定されたタグで問合せ用語または要素の本体をマークアップしたバージョンのドキュメントが戻されます。

マークアップを使用すると、ターゲットの XML 文書が無効になる場合があることに注意してください。

DOCTYPE 間のタグの区別

以前のリリースでは、XML_SECTION_GROUP は、異なる DTD のタグを区別できませんでした。たとえば、連絡先情報を格納するために次の DTD を使用するとします。

```
<!DOCTYPE contact>
<contact>
   <address>506 Blue Pool Road</address>
   <email>dudeman@radical.com</email>
</contact>
```

適切なセクションは、次のようになります。

```
ctx_ddl.add_field_section('mysg','email',  'email');
ctx_ddl.add_field_section('mysg','address','address');
```

これは、同じ表内に異なる種類のドキュメントがない場合は問題ありません。

```
<!DOCTYPE mail>
 <mail>
   <address>dudeman@radical.com</address>
</mail>
```

この場合、本来は住所用であったアドレス・セクションが、タグが重複するために、電子メール・アドレスを取得するようになります。

DOCTYPE 制限の指定によるタグの区別

Oracle8*i* リリース 8.1.5 以上では、DOCTYPE 制限を指定して、DOCTYPE 間のタグを区別できます。次のとおり、タグの前のカッコ内にドキュメント・タイプを指定します。

```
ctx_ddl.add_field_section('mysg','email','email');
ctx_ddl.add_field_section('mysg','address','(contact)address');
ctx_ddl.add_field_section('mysg','email','(mail)address');
```

この場合、XML セクション・グループはアドレス・タグを認識し、そのタグを、ドキュメント・タイプが、`contact` である場合は住所セクションとして、ドキュメント・タイプが `mail` である場合は電子メール・セクションとして索引付けします。

セクション・グループ内での DOCTYPE 制限タグおよび未制限タグの混在

次のとおり、1 つのセクション・グループ内に DOCTYPE 制限タグおよび未制限タグが両方あると想定します。

```
ctx_ddl.add_field_section('mysg','sec1','(type1)tag1');
ctx_ddl.add_field_section('mysg','sec2','tag1');
```

この場合、制限タグは該当する DOCTYPE の場合に適用され、未制限タグはそれ以外のすべてのドキュメント・タイプの場合に適用されます。

これは、問合せには影響しません。問合せは、タグではなく、セクション名で行われるため、電子メール・アドレスの問合せは、次のとおり行われます。

```
radical WITHIN email
```

この場合、2 つの異なる種類のタグを同じセクション名にマップしているため、どちらのタグが電子メール・アドレスを表すために使用されているかに関係なく、ドキュメントが検索されます。

XML_SECTION_GROUP 属性セクション

Oracle8*i* リリース 8.1.5 以上では、XML_SECTION_GROUP によって、属性値の範囲内で索引付けおよび検索を行う機能が提供されます。次の行を含むドキュメントについて考えてみます。

```
<comment author="jeeves">
    I really like Oracle Text
</comment>
```

ここで、XML_SECTION_GROUP が、1 つの属性セクションを提供します。これによって、索引に属性値を挿入できるようになります。次に例を示します。

```
ctx_ddl.add_attr_section('mysg','author','comment@author');
```

構文は、他の add_section コールと同様です。最初の引数はセクション・グループの名前、2 番目はセクションの名前、3 番目は「タグ名 @ 属性名」という形式のタグです。この場合、Oracle Text は、comment タグの author 属性の内容を「author」セクションとして索引付けすることを命令されます。

次に、問合せ構文を示します。これは、他のセクションの場合も同様です。

```
WHERE CONTAINS ( ... ,'jeeves WITHIN author...',...)...
```

これによって、ドキュメントが検索されます。

属性値の識別によるセクション検索

属性セクションを使用すると、属性の内容を検索できます。ただし、属性値を使用して検索するセクションを指定することはできません。たとえば、次のドキュメントを想定します。

```
<comment author="jeeves">
    I really like Oracle Text
</comment>
```

このドキュメントは、次の問合せによって検索できます。

```
jeeves within comment@author
```

これは、author 属性の値に「jeeves」という単語を含む comment 要素を持つすべてのドキュメントを検索することに相当します。

ただし、現在は、次のような要求を実行することはできません。

```
interMedia within comment where (@author = "jeeves")
```

この方法では、jeeves が author である comment 要素で *inter*Media が使用されているすべてのドキュメントが検索されます。この機能（属性値の識別によるセクション検索）は、Oracle9*i* データベースの将来のリリースでサポートされる予定です。

動的追加セクション

セクション・グループは索引の作成前に定義されるため、Oracle8*i* リリース 8.1.5 では、構造化ドキュメント・セットの変更に対する機能が制限されます。ドキュメントで新しいタグを初めて使用する場合、または新しい DOCTYPE を初めて取得する場合は、索引を再作成して、それらのタグの使用を開始する必要があります。

Oracle8*i* リリース 8.1.6 以上では、索引を再作成しないで、新しいセクションを既存の索引に追加できます。この場合、次に示すとおり、ALTER INDEX 文に ADD SECTION パラメータ文字列構文を使用します。

```
add zone  section <section_name> tag <tag>
add field section <section_name> tag <tag> [ visible | invisible ]
```

たとえば、タグ title を使用して、tsec という名前の新しいゾーン・セクションを追加するには、次の構文を使用します。

```
alter index <indexname> rebuild
parameters ('add zone section tsec tag title')
```

タグ author を使用して、asec という名前の新しいフィールド・セクションを追加するには、次の構文を使用します。

```
alter index <indexname> rebuild
parameters ('add field section asec tag author')
```

このフィールド・セクションは、ADD_FIELD_SECTION を使用した場合と同様、デフォルトで表示されません。これを参照可能なフィールド・セクションとして追加するには、次の構文を使用します。

```
alter index <indexname> rebuild
parameters ('add field section asec tag author visible')
```

動的追加セクションは、索引メタデータを変更するのみで、索引を再作成しません。これは、動的追加セクションが、操作後に索引付けされたすべてのドキュメントに影響し、既存のドキュメントには影響しないことを意味します。動的追加セクションを含むドキュメントにすでに索引が付いている場合、これらのドキュメントは、(通常、索引列をそれ自身に更新して) 再度索引付けするために、手動でマーク付けする必要があります。

この操作では、特殊セクションの追加はサポートされません。特殊セクションを追加するには、すべてのドキュメントを再度索引付けする必要があります。この操作は、再作成を使用してオンラインで行うことはできませんが、比較的高速な操作です。

属性セクションの問合せに対する制約

次の制約が、属性セクション内での問合せに適用されます。

- 通常の属性テキストの問合せは、WITHIN 句で修飾されていない場合、該当するドキュメントにヒットしません。次の XML 文書があると想定します。

  ```
  <book title="Tale of Two Cities">It was the best of times.</book>
  ```

  Tale の問合せは、WITHIN title@book で修飾されていない場合、それ自体では該当するドキュメントにヒットしません。この動作は、参照可能なフラグ・セットを FALSE に設定した場合のフィールド・セクションの動作と同様です。

- 属性セクションは、ネストした WITHIN 問合せでは使用できません。
- 句は、属性テキストを無視します。たとえば、次のオリジナル・ドキュメントがあると想定します。

  ```
  Now is the time for all good <word type="noun"> men </word> to come to the aid.
  ```

  通常の good men の問合せを行うと、このドキュメントにヒットし、間に挿入されている属性テキストは無視されます。

WITHIN 問合せは、属性セクションの繰返しを区別できます。この動作は、ゾーン・セクションの動作と同様ですが、フィールド・セクションの動作とは異なります。たとえば、次のドキュメントの場合を考えてみます。

```
<book title="Tale of Two Cities">It was the best of times.</book>
<book title="Of Human Bondage">The sky broke dull and gray.</book>
```

この場合、book がゾーン・セクション、book@author が属性セクションであると想定します。次の問合せについて考えてみます。

```
'(Tale and Bondage) WITHIN book@author'
```

Tale および Bondage は、属性セクション book@author の異なる箇所に出現しているため、この問合せは、ドキュメントにヒットしません。

ゾーン・セクションの繰返し

ゾーン・セクションは、繰り返すことができます。各出現箇所は、個別のセクションとして処理されます。たとえば、<H1> がヘッダー・セクションを示す場合、次のとおり、同じドキュメント内で繰り返すことができます。

```
<H1> The Brown Fox </H1>
<H1> The Gray Wolf </H1>
```

これらのゾーン・セクションに Heading という名前が付けられているとします。

次の問合せを行うとします。

```
WHERE CONTAINS (..., 'Brown WITHIN Heading', ...)...
```

この問合せは、このドキュメントを戻します。

次の問合せを行うとします。

```
WHERE CONTAINS (...,' (Brown and Gray) WITHIN Heading',...)...
```

この問合せは、このドキュメントを戻しません。

ゾーン・セクションのオーバーラップ

ゾーン・セクションは相互にオーバーラップできます。たとえば、<B> および <I> が 2 つの異なるゾーン・セクションを示す場合、次のとおり、これらはドキュメント内でオーバーラップできます。

```
plain <B> bold <I> bold and italic </B> only italic </I>  plain
```

セクションのネスト

ゾーン・セクションは、次のとおり、ネストすることができます。

```
<TD>
  <TABLE>
  <TD>nested cell</TD>
  </TABLE>
</TD>
```

WITHIN 演算子を使用すると、セクション内のセクションのテキストを検索する問合せを作成できます。

ネストしたセクション問合せの例

たとえば、BOOK1、BOOK2 および AUTHOR ゾーン・セクションが、次のとおり、ドキュメント doc1 および doc2 内に現れるとします。

doc1 は次のとおりです。

```
<book1><author>Scott Tiger</author> This is a cool book to read.</book1>
```

doc2 は次のとおりです。

```
<book2> <author>Scott Tiger</author> This is a great book to read.</book2>
```

次のネストした問合せについて考えてみます。

```
'Scott WITHIN author WITHIN book1'
```

この問合せは、doc1 のみを戻します。

CTX_OBJECTS 表および CTX_OBJECT_ATTRIBUTES ビューの使用

CTX_OBJECT_ATTRIBUTES ビューは、各オブジェクトのプリファレンスに割り当てることができる属性を表示します。すべてのユーザーは、このビューを問い合せることができます。

次の DESCRIBE コマンドを使用して、CTX_OBJECTS および CTX_OBJECT_ATTRIBUTE ビューの構造を確認してください。この章では、XML 文書の問合せのみを目的としているため、XML_SECTION_GROUP および AUTO_SECTION_GROUP に注目します。

```
Describe ctx_objects
  SELECT obj_class, obj_name FROM ctx_objects
  ORDRR BY obj_class, obj_name;
```

結果は次のとおりです。

```
...
SECTION_GROUP                  AUTO_SECTION_GROUP     <<==
SECTION_GROUP                  BASIC_SECTION_GROUP
SECTION_GROUP                  HTML_SECTION_GROUP
SECTION_GROUP                  NEWS_SECTION_GROUP
SECTION_GROUP                  NULL_SECTION_GROUP
SECTION_GROUP                  XML_SECTION_GROUP      <<==

...

Describe ctx_object_attributes
SELECT oat_attribute FROM ctx_object_attributes
  WHERE oat_object = 'XML_SECTION_GROUP';
```

結果は次のとおりです。

```
OAT_ATTRIBUTE
-------------
ATTR
FIELD
SPECIAL
ZONE

SELECT oat_attribute FROM ctx_object_attributes
  WHERE oat_object = 'AUTO_SECTION_GROUP';
```

結果は次のとおりです。

```
OAT_ATTRIBUTE
-------------
STOP
```

例 : XML ベースの会議議事録の検索

この例では、INPATH、HASPATH および extract() を使用して、XML ベースの会議議事録を検索します。

注意 : このサンプル・アプリケーションは、
http://otn.oracle.com/products/text からダウンロードできます。

XML 文書のコンテンツおよび構造の検索

文書は、情報が構造化されて表現されています。文書は、構造、コンテンツおよび表示を含む資産として定義できます。この例では、コンテンツおよび構造を同時に検索する方法を説明します。この章で説明するすべての機能は、Oracle9*i* データベース リリース 1（9.0.1）以上で使用できます。

オンラインの会議議事録の検索について考えてみます。この議事録では、会議の出席者が議事録の構造で全文検索（タイトル、執筆者、要約など）を実行できます。

Oracle Text を使用した XML ベースの会議議事録の検索

次のタスクに従って、この会議議事録の検索例を構築します。

タスク 1: システム権限の付与 : 初期化パラメータの設定

ファンクション索引を作成するには、QUERY REWRITE システム権限が必要です。また、次のとおり初期化パラメータを設定する必要もあります。

- QUERY_REWRITE_INTEGRITY を TRUSTED に設定します。
- QUERY_REWRITE_ENABLED を TRUE に設定します。
- COMPATIBLE を 8.1.0.0.0 以上の値に設定します。

タスク 2: Proceedings 表の作成

たとえば、2 つの列（議事録の ID である tk とコンテンツである papers）を持つ Proceedings 表を作成します。paper のコンテンツを XMLType として格納します。

```
CREATE TABLE Proceedings (tk number, papers XMLTYPE);
```

タスク 3: 表へのデータの移入

Proceedings 表に会議議事録を移入します。

```
INSERT INTO Proceedings(tk,papers) VALUES (1, XMLType.createXML(
'<?xml version="1.0"?>
  <paper>
   <title>Accelerating Dynamic Web Sites using Edge Side Includes</title>
    <authors>Soo Yun and Scott Davies</authors>
     <company> Oracle Corporation </company>
       <abstract> The main focus of this presentation is on Edge Side Includes
(ESI). ESI is a simple markup language which is used to mark cacheable and
non-cacheable fragments of a web page. An "ESI aware server", such as Oracle Web
Cache and Akamai EdgeSuite, can take in ESI marked content and cache and assemble
pages closer to the users, at the edge of the network, rather than at the
application server level. This session will discuss the challenge many dynamic
websites face today, discuss what ESI is, explain how ESI can be used to alleviate
these issues. The session will also describe how to build pages with ESI, and detail
the ESI and JESI (Edge Side Includes for Java) libraries. </abstract>
     <track> Fast Track to Oracle9i </track>
</paper>'));
```

タスク 4: XMLType 列での Oracle Text 索引の作成

通常の CREATE INDEX 文を使用して、XMLType 列である papers に Oracle Text 索引を作成します。

```
CREATE INDEX proc_idx ON Proceedings(papers)
```

```
INDEXTYPE IS ctxsys.context
parameters('FILTER ctxsys.null_filter SECTION GROUP ctxsys.path_section_group');
```

タスク 5: XPath および Contains() を使用した会議議事録の問合せ

Oracle9*i* データベース リリース 1（9.0.1）では、新しい SQL 関数の `existsNode()` および `extract()` が導入されています。これらの関数は、次のとおり `XMLType` の値を操作します。

- `existsNode()`: 任意の XPath 式で、XML 文書に適用された XPath が有効なノードを戻すかどうかを確認します。
- `extract()`: 任意の XPath 式で、XPath を XML 文書に適用し、フラグメントを `XMLType` として戻します。

たとえば、XML 文書から執筆者のみを選択するとします。

```
SELECT p.papers.extract('/paper/authors/text()').getStringVal()
   FROM Proceedings p;
```

Oracle Text の `CONTAINS()` 演算子を使用して、テキストまたは XML 文書のコンテンツを検索できます。たとえば、タイトルに「Dynamic」を含む議事録を検索するには、次のコマンドを使用できます。

```
SELECT tk FROM Proceedings
WHERE CONTAINS(papers,'Dynamic INPATH(paper/title)')>0
```

`CONTAINS()` 演算子を使用すると、Oracle9*i* データベースによって選択した列が戻されます。XML 文書の場合、文書全体が戻されます。XML フラグメントを抽出するには、`extract()` 関数と組み合せて XML を操作できます。たとえば、タイトルに「Dynamic」を含む議事録の執筆者を選択するとします。

```
SELECT p.papers.extract('/paper/authors/text()').getStringVal()
FROM Proceedings p
WHERE CONTAINS(papers,'Dynamic INPATH(paper/title)')>0
```

Oracle Text 問合せのすべての機能を使用して、コンテンツを検索できます。次の例では、タイトルに「Dynamic」、「Edge」または「Libraries」を含む議事録の執筆者を選択します。

```
SELECT p.papers.extract('/paper/authors/text()').getStringVal()
FROM Proceedings p
WHERE CONTAINS(papers,'Dynamic or Edge or Libraries INPATH(paper/title)')>0
```

従来のデータベースでは、XML のコンテンツまたは構造の検索はできますが、同時に両方の検索はできません。Oracle では、XML のコンテンツと構造の両方の問合せを同時に可能にする独自の機能が提供されています。

図 7-1 に、会議議事録文書の構造で「Libraries」を検索する場合の入力方法を示します。Author(s)、Abstract、Title、Company または Track 内で「Libraries」を検索できます。こ

の例では、Abstract のみで「Libraries」という用語を検索しています。検索対象が XML 文書であるため、表示する XML 文書のフラグメントも選択できます。この例では、議事録のタイトルのみが表示されます。

図 7-2 に、検索結果を示します。

この検索アプリケーションを構築する .jsp コードについては、7-61 ページの「会議議事録の検索の例 : jsp」を参照してください。

参照 : 次のマニュアルまたは Web サイトを参照してください。

- 『Oracle Text リファレンス』
- 『Oracle Text アプリケーション開発者ガイド』
- http://otn.oracle.com/products/text

図 7-1 Oracle Text を使用した、会議議事録の Abstract での「Libraries」の検索

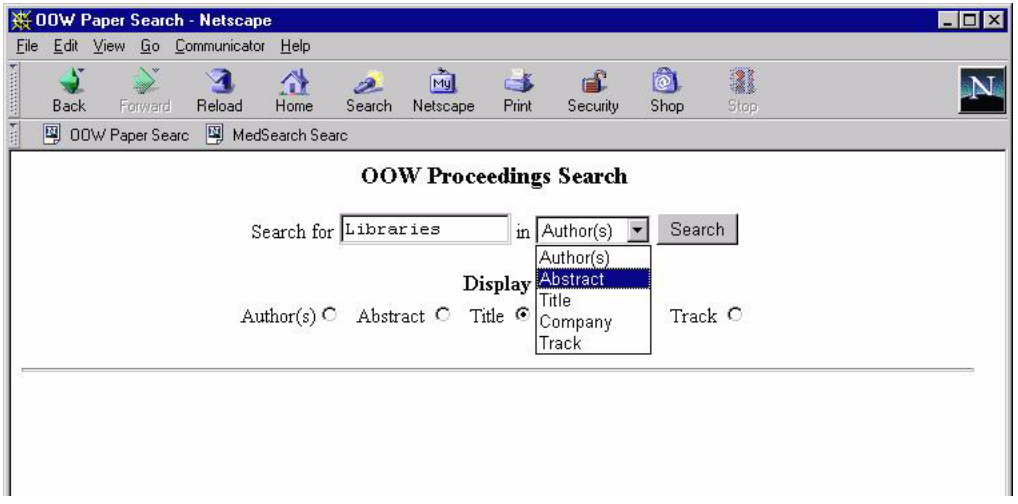
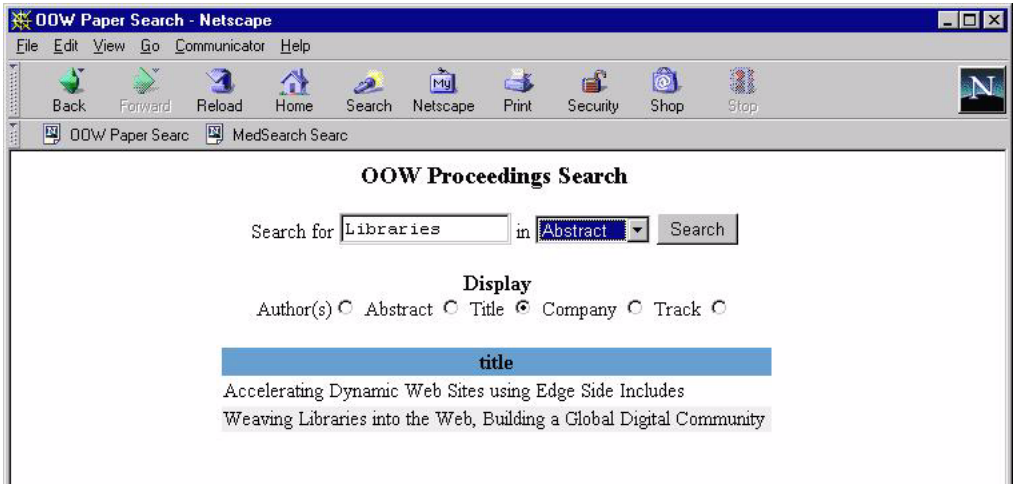


図 7-2 Oracle Text の検索結果



会議議事録の検索の例 : jsp

次に jsp の完全な例を示します。この例では、Oracle Text を使用して XML ベースのオンライン会議議事録検索アプリケーションを作成する方法を示します。

```
<%@ page import="java.sql.* , oracle.jsp.dbutil.*" %>
<jsp:useBean id="name" class="oracle.jsp.jml.JmlString" scope="request" >
<jsp:setProperty name="name" property="value" param="query" />
</jsp:useBean>

<%
  String connStr="jdbc:oracle:thin:@oalonso-sun:1521:betadev";
java.util.Properties info = new java.util.Properties();
Connection conn = null;
  ResultSet  rset = null;
  Statement  stmt = null;

  if (name.isEmpty()) { %>
<html>
       <title>OOW Paper Search</title>
       <body>
       <center>
         <h3>OOW Proceedings Search </h3>
         <form method=post>
         Search for
         <input type=text size=15 maxlength=25 name=query>
```

```
      in
      <select name="tagvalue">
        <option value="authors">Author(s)
        <option value="abstract">Abstract
        <option value="title">Title
        <option value="company">Company
        <option value="track">Track
      </select>
      <input type=submit value="Search">
      <p><b>Display</b><br>
      <table>
       <tr>
        <td>
         Author(s)<input type="radio" name="section" value="authors">
        </td>
        <td>
         Abstract <input type="radio" name="section" value="abstract">
        </td>
        <td>
         Title    <input type="radio" name="section" value="title" checked>
        </td>
        <td>
         Company  <input type="radio" name="section" value="company">
        </td>
        <td>
         Track    <input type="radio" name="section" value="track">
        </td>
       </tr>
      </table>
      </form>
    </center>
    <hr>
  </body>
  </html>

<%
}
else {
%>

 <html>
    <title>OOW Paper Search</title>
    <body>
    <center>
    <h3>OOW Proceedings Search </h3>
      <form method=post action="oowpapersearch.jsp">
      Search for
```

```
          <input type=text size=15 maxlength=25 name="query" value=<%=
name.getValue() %>>
          in
          <select name="tagvalue">
            <option value="authors">Author(s)
            <option value="abstract">Abstract
            <option value="title">Title
            <option value="company">Company
            <option value="track">Track
          </select>
          <input type=submit value="Search">
          <p><b>Display</b><br>
          Author(s)<input type="radio" name="section" value="authors">
          Abstract <input type="radio" name="section" value="abstract">
          Title    <input type="radio" name="section" value="title" checked>
          Company  <input type="radio" name="section" value="company">
          Track    <input type="radio" name="section" value="track">
          </form>
        </center>

    <%
      try {

        DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver() );
        info.put ("user", "ctxdemo");
        info.put ("password","ctxdemo");
        conn = DriverManager.getConnection(connStr,info);

        stmt = conn.createStatement();
        String theQuery =   request.getParameter("query")+"
INPATH(paper/"+request.getParameter("tagvalue")+")";

        String tagValue = request.getParameter("tagvalue");
        String sectionValue = request.getParameter("section");

       // select p.papers.extract('/paper/authors').getStringVal()
       // from oowpapers p
       // where contains(papers,'language inpath(paper/abstract)')>0

         String myQuery = "select
p.papers.extract('/paper/"+request.getParameter("section")+"/text()').getStringVal()
from oowpapers p where contains(papers,'"+theQuery+"')>0";

        rset = stmt.executeQuery(myQuery);
String color = "ffffff";
        String myDesc = null;
```

```
        int items = 0;
        while (rset.next()) {
          myDesc = (String)rset.getString(1);
          items++;
        if (items == 1) {
    %>

            <center>
               <table border="0">
                  <tr bgcolor="#6699CC">
                    <th><%= sectionValue %></th>
                  </tr>
    <%   } %>

         <tr bgcolor="#<%= color %>">
           <td> <%= myDesc %></td>
         </tr>
<%
         if (color.compareTo("ffffff") == 0)
               color = "eeeeee";
             else
               color = "ffffff";

       }
     } catch (SQLException e) {
     %>
        <b>Error: </b> <%= e %><p>
     <%
     } finally {
       if (conn != null) conn.close();
       if (stmt != null) stmt.close();
       if (rset != null) rset.close();
     }
     %>
     </table>
     </center>
     </body></html>
     <%
   }
%>
```

Oracle Text の FAQ

この項の内容は次のとおりです。

FAQ: Oracle Text に関する一般的な質問

XML 関数とともに CONTAINS() 問合せを使用して XML フラグメントを抽出できますか？

回答 : 抽出できます。7-38 ページの「XML データの問合せ CONTAINS または existsNode() の使用」を参照してください。

表データと同様の方法を使用して XML 文書を問合せできますか？

完全な状態の XML 文書は、Oracle の XML ソリューションによって、CLOB に格納されると理解しています。

CLOB または BLOB に格納された XML 文書は、表スキーマのように問い合せることができますか？次に例を示します。

```
[XML document stored in BLOB]...<name id="1111"><first>lee</first>
<sencond>jumee</second></name>...
```

この場合、XML 文書の要素、属性および構造によって、値（lee、jumee）を問い合せることができますか？

回答 : Oracle Text を使用すると、次のような問合せを使用してこの文書を検索できます。

```
lee within first
jumee within second
1111 within name@id
```

これらを次のとおり組み合せることもできます。

```
lee within first and jumee within second, or
(lee within first) within name.
```

詳細は、OTN の Web サイト（http://otn.oracle.com/products/text）からダウンロードできる『Oracle Text Technical Overview』を参照してください。

個別の XML 要素を編集できますか？

挿入、更新または削除できない要素または属性がある場合、文書全体を更新する必要がありますか？また文書は、表スキーマと同様に、挿入、更新または削除できますか？

回答：Oracle Text は、CLOB および BLOB を索引付けしますが、XML を高度に認識しているわけではないので、実際には個々の要素を変更することはできません。文書全体を編集する必要があります。

XML ファイルはどのように CLOB および BLOB にロックされるのですか？

ロックに関する質問ですが、BLOB または CLOB に格納された XML 文書を管理している場合、誰でも同じ XML 文書にアクセスできますか？

回答：他の CLOB と同様、誰かが CLOB に書き込んでいる場合、その CLOB はロックされ、他の人はそれに書き込むことができません。他のユーザーは、その CLOB を読み込むことはできますが、書き込むことはできません。これは、LOB の基本動作です。

別の方法として、XML 文書を分解し、情報をリレーショナル・フィールドに格納する方法があります。これによって、個々の要素の変更、要素レベルでの同時アクセスなどが可能になります。この場合、USER_DATASTORE および PL/SQL を使用すると、再度ドキュメントを XML に構成し、テキストを索引付けすることができます。これによって、テキストを XML として検索できますが、データはリレーショナル・データとして管理されます。詳細は、http://otn.oracle.com/products/text の『Oracle Text Technical Overview』を参照してください。

XML 文書を検索してゾーンを取得する方法は？

大規模な XML ファイルを格納し、それを検索して、特定のタグ付けされた領域を取得する必要があります。Oracle Text を使用すると、次のことが実行できます。

- XML ファイルを CLOB フィールドに格納できます。
- ctxsys.context を使用して、XML ファイルを索引付けすることができます。
- <Zone> および <Field> を作成して、ctx_ddl.add_zone_section(xmlgroup,"dublincore", dc); のように、XML ファイルにタグを表すことができます。
- ゾーンまたはフィールドでテキストを検索できます。たとえば、「Select titile from mytable where CONTAINS(textField,"some words WITHIN doubleness")」という問合せが可能です。

テキスト検索によってゾーンまたはフィールドを取得する方法を教えてください。

回答：Oracle Text は、ヒットした項目のみを戻します。Oracle Text の doc サービスを使用して、セクションをハイライト表示またはマークアップするか、または、CLOB を XMLtype 列に格納して、extract() 関数を使用できます。

XML 文書をデータベースにロードする方法は？

XML 文書をデータベースに挿入する方法を教えてください。XML 文書を、現在の状態のまま、表内の CLOB データ型の列に挿入する必要があります。

回答 : Oracle XSU for Java は、XML データをロードするために使用可能なコマンドライン・ユーティリティを提供します。XML SQL Utility の詳細は、http://otn.oracle.co.jp/ および『Oracle9*i* XML Developer's Kit ガイド -XDK』の第 8 章「XML SQL Utility（XSU）」を参照してください。

XML 文書は、すべてのテキスト・ファイルと同様に挿入できます。CLOB への挿入も、他のテキスト・ファイルの場合と変わりません。

Oracle Text を使用して XML 文書を検索する方法は？

Oracle Text を使用して、CLOB に格納された XML を索引付けおよび検索できますか？これを行うための方法を教えてください。

回答 : Oracle8*i* リリース 8.1.6 より前のリリースの Oracle Text では、タグベースの検索のみが可能でした。今回のリリースでは、XML の構造ベースおよび属性ベースの検索が可能になっています。索引の作成方法および Oracle Text での SQL の使用方法に関するドキュメントを参照してください。

参照 : 『Oracle Text リファレンス』を参照してください。

WITHIN 演算子を使用して XML を検索する方法は？

次の XML コードがあります。

```
<person>
  <name>efrat</name>
 <childrens>
     <child>
         <id>1</id>
         <name>keren</name>
     </child>
  </childrens>
</person>
```

keren という名前の子供を持ち、自分の名前が keren ではない人を検索する方法を教えてください。ネスト可能で、自分自身を含むことができる add_zone_section を使用して、すべてのタグを定義済であると想定します。

回答 :「(keren within name) within child」を使用します。

Oracle Text を使用した XML 検索の例の参照先は？

回答 : 次のマニュアルを参照してください。

- 『Oracle Text アプリケーション開発者ガイド』
- 『Oracle Text リファレンス』

Oracle Text では XML タグが自動的に認識されますか？

Oracle Text は、XML 文書内のタグを認識できますか？または、XML 文書内のタグごとに add_field_section コマンドを使用する必要がありますか？使用する XML 文書には、何百ものタグが含まれています。これを簡単に行う方法はありますか？

回答 : どのリリースのデータベースをご使用ですか？Oracle8*i* リリース 8.1.5 では、タグごとに add_field_section コマンドを使用する必要がありますが、Oracle8*i* リリース 8.1.6 ではその必要はありません。Oracle8*i* リリース 8.1.6 では、AUTO_SECTION_GROUP を使用できます。

XSQL Servlet には、SQL スクリプトの完全（Oracle Text の使用部分は単純）な例が付属しています。このスクリプトは、オブジェクト型から複雑な XML データグラムを作成し、保険請求型に格納された XML 文書のフラグメントに対して Oracle Text 索引を作成します。

XSQL Servlet をダウンロードし、ファイル ./xsql/demo/insclaim.sql を参照すると、ファイルの下にある Oracle Text の要素を参照できます。Oracle8*i* リリース 8.1.6 における Oracle Text の新しい主要機能は、AUTO Sectioner for XML です。

Oracle Text を使用して範囲検索を行うことができますか？

CLOB に格納した XML 文書があります。また、section_group などを使用して、タグに対する索引を作成済です。1 つのタグは、<SALARY> </SALARY> です。給与が 5000 を超えるすべてのレコードを選択する SQL 文を記述する必要があります。これを行うには、どうすればよいですか？WITHIN 演算子は使用できません。このタグ内の値を数値として解釈する必要があります。また、これは給与であるため、浮動小数点の場合もあります。

回答 : これは Oracle Text では行えません。実際には、範囲検索はテキスト操作ではありません。最適なソリューションは、別の Oracle の XML 解析ユーティリティを使用して、給与を NUMBER フィールドに変換することです。これによって、テキスト検索には Oracle Text を使用し、より構造化されたフィールドには通常の SQL 演算子を使用して、同じ結果を取得できます。

Oracle Text を使用してセクションを取り出すことができますか？

XML 形式のすべてのドキュメントを CLOB に格納しています。Oracle には、ドキュメント全体を取得して、その構造を検索するのではなく、1 つのフィールドごとに内容を一度に取得（フィールド名を指定して、タグ間のテキストを取得する）するために使用できるユーティリティはありますか？Oracle Text はそれに該当しますか？

回答 : Oracle Text は、セクションの取出しを行いません。詳細は、第 7 章「XML SQL Utility（XSU）」を参照してください。

3 つの列に対して 1 つの Text 索引を作成できますか？

7 ～ 8 つの表に基づいてビューを作成し、このビューには、`custordnumber`、`product_dsc`、`qty`、`prdid`、`shipdate`、`ship_status` などの列があります。次の 3 つの列に対して 1 つの Oracle Text 索引を作成する必要があります。

- `custordnumber`
- `product_dsc`
- `ship_status`

これらの列に対して 1 つの Text 索引を作成する方法はありますか？

回答 : あります。次の 2 つのオプションがあります。

1. 索引付け実行中に、`USER_DATASTORE` オブジェクトを使用してその場で連結フィールドを作成します。
2. フィールドを連結し、1 つの表の追加の `CLOB` フィールドに格納します。次に、その `CLOB` フィールドに対する索引を作成します。Oracle8*i* リリース 8.1.6 以上を使用している場合、連結前に XML タグを各フィールドの前後に置くオプションもあります。これによって、各フィールド内での検索が可能になります。

Oracle9*i* データベースによるテキストの索引付けの速度は？ブール検索を有効化できますか？

mySQL を使用して、1 日に 900 万の Web ページの部分索引付けを行っています。4 つの CPU を搭載した SPARC 420 で実行していますが、全文索引を作成することができません。Oracle8*i* または Oracle9*i* データベースでこれを行うことができますか？

トランザクションの整合性、テキスト・ページへの特殊なフィルタの適用、単純なブール・ワード検索以外の検索（スコアリング、ステミング、ファジー検索、近接検索など）の実施には関心がありません。

次の質問があります。

- Oracle8*i* または Oracle9*i* データベースでは、mySQL より高速で索引作成を行うことができますか？

- その場合、ブール・ワード検索以外のすべての索引作成機能を無効化する方法はありますか？

回答：あります。Oracle Text は、900 万の Web ページの全文索引を、高速で作成できます。Sun 社製の大型サーバーを使用したベンチマーク・テストでは、100GB の Web ページ（約 1,500 万）を 7 時間で索引付けできました。通常の DML またはパーティション化（Oracle9*i* データベースの場合）を使用して、部分索引付けを行うこともできます。

テーマ索引付けを無効化する「簡易索引付け」を行うこともできます。ドキュメントがすでに ASCII、HTML または XML である場合、そのドキュメントにフィルタを適用する必要はありません。最も一般的な拡張であるファジー、ステミングおよび近接は、問合せ時に行います。

FAQ: Oracle Text を使用した属性値の検索

属性値のテキスト索引を作成できますか？

現在、Oracle Text には、セクション・グループの内容に基づいて索引を作成するためのオプションがあります。ただし、ほとんどの XML 要素は、型の要素です。このため、検索のためのオプションは、属性値のみとなります。属性値の索引を作成できますか？

回答：Oracle8*i* リリース 8.1.6 以上では、属性を索引付けできます。次の Web サイトを参照してください。

```
http://otn.oracle.com/products/intermedia/
```

FAQ: Oracle Text を使用した CLOB 内の XML 文書の検索

CLOB に格納された様々な XML 文書を検索する方法は？

CLOB に XML を格納し、DOM または SAX パーサーを使用して後から必要に応じて XML を再解析します。このドキュメント・リポジトリを検索するには Oracle Text が最適ですか？Oracle8*i* で *inter*Media を使用してこの問題を解決するための例（XML_SECTION_GROUP の定義方法、FIELD に対する ZONE の使用場所など）を参照できれば参考になります。次に例を示します。

次の XML（および DTD）を使用して、CLOB 列内の「aorta」および「damage」を含むレコードを検索するには、*inter*Media パラメータをどのように定義すればよいですか？

```
WellKnownFileName.gif <keyword>echo</keyword>
<keyword>cardiogram aorta</keyword>
```

これは、血管損傷のイメージです。

回答：Oracle8*i* リリース 8.1.6 以上では、属性テキスト内で検索を行うことができます。これは、「state within book@author」のような検索です。Oracle は、次のような属性値の識別による検索を実行できます。

```
state within book[@author = "Eric"]:

begin  ctx_ddl.create_section_group('mygrp','basic_section_group');
  ctx_ddl.add_field_section('mygrp','keyword','keyword');
  ctx_ddl.add_field_section('mygrp','caption','caption');
end;
create index myidx on mytab(mytxtcolumn)indextype is ctxsys.contextparameters
('section group mygrp');
select * from mytab where contains(mytxtcolumn, 'aorta within keyword')>0;
options:
```

- タグに属性が含まれる場合、または大 / 小文字を区別したタグの検出が必要である場合は、基本セクション・グループのかわりに XML セクション・グループを使用します。
- セクションがオーバーラップする場合、またはインスタンスを区別する必要がある場合は、フィールド・セクションのかわりにゾーン・セクションを使用します。たとえば、keywords がフィールド・セクションの場合、「(aorta and echo cardiogram) within keywords」という問合せは、ドキュメントにヒットします。keywords がゾーン・セクションの場合、この問合せは、同じ keywords のインスタンスにないため、ドキュメントにヒットしません。

Oracle Text を使用して CLOB に XML 文書を格納する方法は ?

現在、データベースのファイル・システムに存在する XML ファイルを格納する必要があります。ファイルを文書全体として格納する必要があります。タグによって文書を分割し、情報を別々の表またはフィールドに格納することは望ましくありません。逆に、様々な XML 文書を格納するために使用できる汎用表が必要です。汎用表は内部で CLOB 型のフィールドに格納されると考えています。使用する XML ファイルは、常に ASCII データを含みます。

Oracle Text を使用して、このような処理を行うことができますか ? そのためには、Oracle Text または Oracle Text Annotator のどちらを使用する必要がありますか ? Annotator は入手済ですが、XML 文書をデータベースに格納することができません。

XML 文書を CLOB 列に格納しようとしています。基本的に、次のように定義されている表が 1 つあります。

```
CREATE TABLE xml_store_testing
(
    xml_doc_id  NUMBER,
    xml_doc     CLOB  )
```

XML 文書を xml_doc フィールドに格納する必要があります。

XML 文書のコンテンツを読み込むために、次に示す別の PL/SQL プロシージャを作成しました。XML 文書は、ファイル・システム上で使用可能であり、XML 文書には ASCII データのみが含まれ、バイナリ・データはありません。

```
CREATE OR REPLACE PROCEDURE FileExec
(
```

```
  p_Directory      IN VARCHAR2,
  p_FileName       IN VARCHAR2)
  AS   v_CLOBLocator  CLOB;
       v_FileLocator       BFILE;
BEGIN
    SELECT  xml_doc
    INTO      v_CLOBLocator
    FROM    xml_store_testing
    WHERE  xml_doc_id = 1
    FOR        UPDATE;
    v_FileLocator := BFILENAME(p_Directory, p_FileName);
    DBMS_LOB.FILEOPEN(v_FileLocator, DBMS_LOB.FILE_READONLY);
    dbms_output.put_line(to_char(DBMS_LOB.GETLENGTH(v_FileLocator)));
      DBMS_LOB.LOADFROMFILE(v_CLOBLocator, v_FileLocator,
      DBMS_LOB.GETLENGTH(v_FileLocator));
      DBMS_LOB.FILECLOSE(v_FileLocator);
END FileExec;
```

回答 : XML 文書を CLOB 列に挿入してから、XML_SECTION_GROUP を使用して、Oracle Text 索引をそれに追加してください。詳細は、http://otn.oracle.com/products/intermedia を参照してください。

XML を CLOB に格納するとキャラクタ・セットに影響がありますか ?

XML 文書を CLOB 列に挿入し、XML セクション・グループを使用して Oracle Text 索引を追加すると、正常に実行されました。ただし、表から選択した場合、CLOB フィールド内の表に不明な文字が現れました。これは、オペレーティング・システム（XML ファイルを格納）とデータベース（CLOB データを格納）のキャラクタ・セットの違いが原因ですか ?

回答 : キャラクタ・セットの違いが原因です。キャラクタ・セットが異なる場合は、UTL_RAW.CONVERT を介してデータを渡し、キャラクタ・セットを変換してから CLOB に書き込む必要があります。

表の作成時に構造化データのみを挿入できますか ?

データを XML ファイルからデータベースに挿入する必要があります。作成済の表を使用して構造化データのみを挿入できると聞きました。これは本当ですか ?

現在、法律プロジェクトに取り組んでおり、このプロジェクトでは、構造化データおよび非構造化データを含む法律データを格納し、Oracle Text を使用してそのデータを検索する必要があります。非構造化データも挿入できますか ? これを行うには、カスタム・アプリケーションを作成する必要がありますか ? また、構造化部分および非構造化部分を持つデータを格納した場合、Oracle Text を使用してそのデータを検索できますか ? 非構造化部分を CLOB に格納し、その CLOB にタグが含まれている場合、特定のタグ内にあるデータのみを検索することはできますか ?

回答 : Oracle 9*i*FS の使用を検討してください。Oracle 9*i*FS を使用すると、文書を分割し、それを複数の表および LOB に格納できます。Oracle Text は、タグを使用してデータ検索を行

うことができ、XML の階層構造も理解します。Oracle8*i* リリース 8.1.6 以上では、Oracle Text に、この機能および名前 / 値のペア属性の検索機能が追加されています。

カスタム・アプリケーションを作成せずに XML 文書を分割できますか？

カスタム・アプリケーションを開発しない場合、文書を分割できますか？Oracle Text では、XML の階層構造が認識されませんが、次のような検索はできますか？

```
<report>
   <day>yesterday</day> there was a disaster <cause>hurricane</cause>
</report>
```

Oracle Text を使用して索引付けし、cause が hurricane である LOB を検索できますか？

回答 : Oracle Text の今回のリリースを使用すると、このレベルの検索は可能です。現在、文書を分割するには、Oracle XML Parser を XSLT とともに使用して、XML を DDL に変換するスタイルシートを作成する必要があります。Oracle 9*i*FS を使用すると、さらに高レベルのインタフェースが提供されます。

もう 1 つの方法は、JDBC プログラムを使用して文書または文書のフラグメントのテキストを CLOB 列または LONG 列に挿入し、索引を設定してから、CONTAINS() 演算子を使用して検索を行う方法です。

XML_SECTION_GROUP を含む部分文字列索引を作成する構文は？

CLOB の XML コンテンツ内に存在する特定のタグに、既存の XML_SECTION_GROUP を含む CLOB 列があります。

```
begin
 ctx_ddl.create_section_group('XMLDOC','XML_SECTION_GROUP');
 end;
 /
 begin
 ctx_ddl.add_zone_section ('XMLDOC','title','title');
 ctx_ddl.add_zone_section('XMLDOC','keywords','keywords');
 ctx_ddl.add_zone_section('XMLDOC','author','author');
 end;
 /
 create index xmldoc_idx on xml_documents(xmldoc)
 indextype is ctxsys.context
 parameters ('section group xmldoc') ;
```

最初の文字のみで「author」ゾーン・セクションを検索する必要があります。部分文字列索引を使用する必要があると考えますが、部分文字列索引を作成する構文がわかりません。特に、この列で SECTION_GROUP プリファレンスを宣言済で、WORDLIST プリファレンスも作成する必要がある場合です。

回答 : ここでの主な問題は、author セクションのみにそのように便利な部分文字列処理を適用できないということです。部分文字列処理はすべての部分に適用され、索引のサイズが膨大になります。すべての処理にドキュメントの索引付けが必要になるため、索引全体を再構築する必要があります。この問題を解決するための様々な方法を次に示します。

1. 何もしません。単純に Z% `WITHIN` AUTHOR のような問合せを実行します。

   メリット : 索引を再構築する必要がありません。

   デメリット : 問合せには時間がかかります。ワイルド・カードの文字数制限のために、実行できない問合せもあります。

2. PREFIX_INDEX を TRUE、PREFIX_MIN_LENGTH を 1、PREFIX_MAX_LENGTH を 1 に設定して、ワードリスト・プリファレンスを作成します。問合せは、Z% `WITHIN` AUTHOR のようになります。

   メリット : 問合せが比較的高速に実行されます。

   デメリット : Oracle8*i* リリース 8.1.7 以上を使用しない場合は、他のセクションから不適切な文字列が戻されます。

3. 前述の問合せに、AUTHOR、KEYWORDS、TITLE のフィールド・セクションを追加します。

   メリット : 2 より問合せが高速に実行されます。

   デメリット : フィールド・セクションは、ネストおよび繰返しの出現に関して柔軟性が低下します。

4. user_datastore または procedure_filter を使用して、次のとおりデータを変換します。

   ```
   <AUTHOR>Steven King</AUTHOR>
   ```

   このデータが、次のとおり変換されます。

   ```
   <AUTHORINIT>AIK</AUTHORINIT><AUTHOR>Steven King<AUTHOR>
   ```

   AUTHORINIT のフィールド・セクションを使用すると、問合せは次のようになります。

   ```
   AIK within AUTHORINIT
   ```

   I および A の非ストップワードを作成する必要がないように、K のみではなく AIK を使用します。

   メリット : 問合せが最も高速に実行され、索引が最も小さくなります。

   デメリット : データを操作するたびに多くの作業が必要となるため、索引付けに時間がかかります。

関連性 Y を持つ項目 X を XML 検索すると不適切な結果が戻るのはなぜですか？

Sun SPARC Solaris 5.8 および Oracle8*i* Enterprise Edition リリース 8.1.7.2.0 で Oracle Text を使用しています。XML タグ内に属性を含む XML 文書を索引付けしています。XML のセクションの 1 つは、文書に関連付けられたサブジェクトのリストです。各サブジェクトには、ドキュメントへの関連性が設定されています。関連性 y を持つ項目 x を検索する必要がありますが、不適切な結果が戻されました。たとえば、サブジェクト PA については、ある行のデータは次のようになります。

```
DOC  1 --> Story_seq_num = 561106
<ne-metadata.subjectlist>
  <ne-subject code="PA" source="NEWZ" relevance="50" confidence="100"/>
  <ne-subject code="CONW" source="NEWZ" relevance="100" confidence="100"/>
  <ne-subject code="LENF" source="NEWZ" relevance="100" confidence="100"/>
  <ne-subject code="TRAN" source="NEWZ" relevance="100" confidence="100"/>
</ne-metadata.subjectlist>
DOC  2 --> Story_seq_num =561107
<ne-metadata.subjectlist>
  <ne-subject code="CONW" source="NEWZ" relevance="100" confidence="100"/>
...
```

関連性が 100 であるサブジェクト PA を問い合せた場合、DOC 2 のみが戻されると予測します。テスト結果を次に示します。

これは、適切な結果ですか？

表

```
drop table t_stories1 ;
create table t_stories1 as select * from t_Stories_bck
where story_Seq_num in (561114,562571,562572,561106,561107);
```

索引セクション

```
BEGIN
-- Drop the preference if it already exists
CTX_DDL.DROP_SECTION_GROUP('sg_nitf_story_body2');
END;
/
BEGIN
--Define a section group
ctx_ddl.create_section_group ('sg_nitf_story_body2','xml_section_group');
-- Create field sections for headline and body
ctx_ddl.add_field_section('sg_nitf_story_body2','HL','hedline',true);
ctx_ddl.add_field_section('sg_nitf_story_body2','ST','body.content', true);
--Define attribute sections for the source fields
ctx_ddl.add_attr_section( 'sg_nitf_story_body2', 'P', 'ne-provider@id');
ctx_ddl.add_attr_section( 'sg_nitf_story_body2', 'C', 'ne-publication@id');
```

```
ctx_ddl.add_attr_section( 'sg_nitf_story_body2', 'S', 'ne-publication@section');
ctx_ddl.add_attr_section( 'sg_nitf_story_body2', 'D', 'date.issue@norm');
ctx_ddl.add_attr_section( 'sg_nitf_story_body2', 'SJ', 'ne-subject@code');
ctx_ddl.add_attr_section( 'sg_nitf_story_body2', 'SJR', 'ne-subject@relevance');
ctx_ddl.add_attr_section( 'sg_nitf_story_body2', 'CO', 'ne-company@code');
ctx_ddl.add_attr_section( 'sg_nitf_story_body2', 'TO', 'ne-topic@code');
ctx_ddl.add_attr_section( 'sg_nitf_story_body2', 'TK', 'ne-orgid@value');

 ENd;
 /
```

次のとおり索引を作成します。

```
drop index ix_stories ;
CREATE INDEX ix_stories on T_STORIES1(STORY_BODY)
 INDEXTYPE IS CTXSYS.CONTEXT
PARAMETERS ('SECTION GROUP sg_nitf_story_body2 STORAGE ixst_story_body  ');
```

-- 索引をテストします。

-- 関連性が 100 であるサブジェクト PA を検索しています。

-- 次の問合せでは適切な結果が戻されます。

```
SELECT STORY_SEQ_NUM, STORY_BODY FROM T_STORIES1 WHERE CONTAINS(STORY_BODY, 'PA
WITHIN SJ')>0;
```

-- 次の問合せでは不適切な結果が戻されます。

```
SELECT STORY_SEQ_NUM, STORY_BODY  FROM T_STORIES1 WHERE CONTAINS(STORY_BODY, 'PA
WITHIN SJ AND 100 within SJR')>0;
```

いくつかの行のデータは次のようになります。

```
Story_seq_num = 561106
<ne-metadata.subjectlist>
   <ne-subject code="PA" source="NEWZ" relevance="50" confidence="100"/>
   <ne-subject code="CONW" source="NEWZ" relevance="100" confidence="100"/>
   <ne-subject code="LENF" source="NEWZ" relevance="100" confidence="100"/>
  <ne-subject code="TRAN" source="NEWZ" relevance="100" confidence=”100”/>
</ne-metadata.subjectlist>

Story_seq_num =561107
<ne-metadata.subjectlist>
   <ne-subject code="CONW" source="NEWZ" relevance="100" confidence="100"/>
...
```

関連性が 100 であるサブジェクト PA を検索しています。

Story_seq_num = 561107 のみが戻されると予測します。

関連性が 100 であるサブジェクト PA を問い合せたため、結果は不適切です。関連性が 50 で <ne-subject code="PA" source="NEWZ" relevance="50" confidence="100"/> の story_seq_num=561106 が戻されます。

```
SQL> connect sosa/sosa
Connected.
SQL> select object_name, object_type from user_objects;

OBJECT_NAME
--------------------------------------------------------------------------------
OBJECT_TYPE
------------------
IX_STORIES
INDEX
SYS_LOB0000025364C00005$$
LOB
SYS_LOB0000025364C00009$$
LOB

OBJECT_NAME
--------------------------------------------------------------------------------
OBJECT_TYPE
------------------
SYS_LOB0000025364C00014$$
LOB
SYS_LOB0000025364C00016$$
LOB
T_STORIES1
TABLE

6 rows selected.
SQL> drop index ix_stories force;
Index dropped....
```

回答 : Oracle8*i* リリース 8.1.7 では、このような検索は実行できません。Oracle9*i* データベース リリース 1（9.0.1）の PATH セクション・グループが必要です。これは、このような関連性をより高度に認識します。Oracle8*i* リリース 8.1.7 でこれを実行するには、ドキュメントを再フォーマット（必要に応じて、プロシージャ・フィルタまたはユーザー・データ・ストアを使用）し、ゾーン・セクションを使用して、内部にネストする必要があります。

```
<A B="C" D="E">...
```

これによって、次のようになります。

```
<A><B>C</B><D>E</D>...
```

また、問合せは次のようになります。

Oracle9*i* データベース リリース 1（9.0.1）の A 内の（B 内の C と D 内の E）という場合、次のような問合せを使用して、未修正のデータ上で PATH_SECTION_GROUP を使用できます。

```
haspath(//ne-subject[@code = "PA" and @relevance = "100"])
```

第 III 部

XMLType API を使用した XML データの操作

第 III 部では、PL/SQL および Java 用の Oracle XML DB の `XMLType` API を使用して、XML データにアクセスおよび操作する方法を説明します。第 III 部に含まれる章は、次のとおりです。

- 第 8 章「PL/SQL API for XMLType」
- 第 9 章「Java API for XMLType」

8

PL/SQL API for XMLType

この章では、PL/SQL API for XMLType の使用方法を説明します。この章の内容は次のとおりです。

- PL/SQL API for XMLType の概要
- PL/SQL DOM API for XMLType（DBMS_XMLDOM）
- PL/SQL Parser API for XMLType（DBMS_XMLPARSER）
- PL/SQL XSLT Processor for XMLType（DBMS_XSLPROCESSOR）

PL/SQL API for XMLType の概要

この章では、PL/SQL Application Program Interface（API）for XMLType について説明します。内容は次のとおりです。

- **PL/SQL DOM API for XMLType（パッケージ DBMS_XMLDOM）**: XMLType オブジェクトにアクセスするために使用します。XML Schema に基づく文書および XML Schema に基づかない文書にアクセスできます。データベースを起動する前に、初期化ファイル（.ORA）に読込み元ディレクトリおよび書込み先ディレクトリを指定する必要があります。次に例を示します。

  ```
  UTL_FILE_DIR=/mypath/insidemypath
  ```

  読込み元ファイルおよび書込み先ファイルは、サーバーのファイル・システムに存在する必要があります。

- **PL/SQL XML Parser API for XMLType（パッケージ DBMS_XMLPARSER）**: XML 文書のコンテンツおよび構造にアクセスするために使用します。
- **PL/SQL XSLT Processor for XMLType（パッケージ DBMS_XSLPROCESSOR）**: XSLT を使用して XML 文書を他のフォーマットに変換するために使用します。

Oracle9*i* データベース リリース 1（9.0.1）の XDK for PL/SQL との下位互換性

今回のリリースでは、下位互換性を確保するために、次のコンポーネントに対する XDK for PL/SQL のサポートが保持されています。

- XML Parser for PL/SQL
- XSLT Processor for PL/SQL

そのため、Oracle9*i* データベース リリース 1（9.0.1）の XML Parser for PL/SQL および XSLT Processor for PL/SQL のインスタンス用に作成された多くのアプリケーションを変更する必要はありません。今回のリリースでは、更新された PL/SQL DOM および XMLType API の拡張機能を使用して作成された新しいアプリケーションに、XDK の XML Parser for PL/SQL および XSLT Processor for PL/SQL は必要ありません。

> **参照：** 『Oracle9*i* XML Developer's Kit ガイド - XDK』を参照してください。

キャラクタ・セット変換およびファイル・システムを使用するアプリケーション

キャラクタ・セット変換およびファイル・システムの相互作用が頻繁に発生するアプリケーションには変更が必要です。変更が必要になるのは、UTL_FILE パッケージの制限（データベースの起動時に指定された UTL_FILE_DIR のファイルの読込み / 書込みなど）のためです。

注意： 今回のリリースでは、以前の XDK パッケージ XMLDOM、XMLPARSER および XSLPROCESSOR が PL/SQL パッケージ DBMS_XMLDOM、DBMS_XMLPARSER および DBMS_XSLPROCESSOR に置き換えられています。

PL/SQL API for XMLType と XDK for PL/SQL の違い

この項では、Oracle XML DB にネイティブな PL/SQL API と、XML Developer's Kit（XDK）で使用可能な PL/SQL API の違いについて説明します。

- **PL/SQL API for XMLType:** サーバーで実行するアプリケーションを開発するには、PL/SQL API for XMLType を使用します。Oracle XML DB の PL/SQL API for XMLType は、データベース内でネイティブな XML サポートを提供します。
- **Oracle XML XDK for PL/SQL:** 中間層およびクライアント側の XML サポートには、Oracle XDK for PL/SQL を使用します。

注意： Oracle XML DB API は、Oracle9*i* データベース リリース 2（9.2）にネイティブに統合されており、個別には使用できません。Oracle XML DB API は、Oracle Technology Network Japan（OTN-J）サイトからダウンロードできません。

ただし、Oracle XDK は次の OTN-J サイトから個別にダウンロードできます。

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

参照： 8-5 ページの「PL/SQL DOM API for XMLType（DBMS_XMLDOM）」を参照してください。

PL/SQL API for XMLType の機能

PL/SQL API for XMLType を使用すると、次のタスクを実行できます。

- XMLType 表、列およびビューの作成
- XMLType データへのアクセス
- XMLType データの操作

参照： 次の項またはマニュアルを参照してください。

- Oracle XML DB のアーキテクチャおよび新機能の概要については、8-2 ページの「Oracle XML DB の主要な機能」
- 第 4 章「XMLType の使用」
- 『Oracle9*i* XML API リファレンス - XDK および Oracle XML DB』

XML の遅延ロード（実体化の遅延）

XMLType ではメモリー内または仮想ドキュメント・オブジェクト・モデル（DOM）が提供されるため、XML の遅延ロード（実体化の遅延ともいう）というメモリー節約プロセスを使用できます。このプロセスは、データ行が要求されたときに、データ行のみをロードすることによって、メモリー使用量を最適化します。メモリーの使用量が大きくなりすぎた場合に、文書内の参照済の部分を廃棄します。XML の遅延ロードは、多くの同時ユーザーが大規模な XML 文書にアクセスする必要があるスケーラブルなアプリケーションに有効です。

XMLType データ型での XML Schema のサポート

今回のリリースでは、XMLType データ型が拡張され、XML Schema もサポートされています。XML Schema を作成し、XML を使用してオブジェクト・リレーショナル・マッピングに注釈を付けることができます。PL/SQL DOM API を使用するには、まず XML Schema を作成して登録します。その後、作成する XMLType 表および列が、Oracle XML DB に定義および登録した XML Schema に準拠するように指定できます。

PL/SQL API for XMLType を使用した XML 要素の変更および格納

一般的な XML パーサーでは、XML データに対する標準的な読込みアクセスが提供されますが、個別の XML 要素を変更および格納する方法は提供されません。

要素の概要 XML 文書の基本論理単位です。他の要素（子、データ、属性およびそれらの値など）のコンテナとして機能します。要素は、<name> などの開始タグおよび </name> などの終了タグ、または <name/> などの空要素によって識別されます。

DOM パーサーの概要 埋込み DOM パーサーは、XML 形式の文書を受け入れ、文書の構造に基づいてメモリー内に DOM ツリーを構築します。文書が整形式であるかどうかを確認し、オプションで特定の Document Type Definition（DTD）に準拠しているかどうかを確認します。DOM パーサーは、DOM ツリーを全検索して、DOM ツリーからデータを戻す方法も提供します。

PL/SQL DOM API を使用する場合、NamedNodeMap メソッドを使用して、XML ファイルから要素を取得できます。

サーバー側のサポート PL/SQL API for XMLType は、サーバー側の処理のみをサポートします。今回のリリースでは、クライアント側の処理はサポートされません。

PL/SQL DOM API for XMLType（DBMS_XMLDOM）

W3C のドキュメント・オブジェクト・モデル（DOM）勧告の概要

World Wide Web Consortium（W3C）によって勧告された一般的な DOM 仕様をすでに十分に理解している場合は、この項をスキップしてください。

W3C によって勧告されたドキュメント・オブジェクト・モデル（DOM）は、XML 文書の構造に対する汎用 API です。DOM は、動的 HTML を形式化するために開発されました。動的 HTML を使用すると、Web ページのアニメーション、対話および動的更新が可能になります。通常、DOM は、Web ページおよび XML 文書の構造に、言語およびプラットフォームに依存しないオブジェクト・モデルを提供します。DOM は、XML のコンポーネントおよび要素にアクセスおよび操作するための、言語およびプラットフォームに依存しないインタフェースを記述します。XML 文書の構造を、コンテンツに依存しない汎用的な方法で表現します。XML 文書のコンテンツ、属性およびスタイルを動的に削除、追加および編集するアプリケーションを作成できます。また、DOM によって、すべてのブラウザ、サーバーおよびプラットフォームで適切に動作するアプリケーションを作成できます。

この項では、便宜上、DOM 勧告の状況の概要を示します。

今回のリリースでサポートされない W3C の DOM 拡張

W3C の DOM API の拡張機能のうち、今回のリリースで唯一サポートされないのは、クライアント側ファイル・システムの入出力およびキャラクタ・セット変換に関する機能です。この種類の手続き型処理は、SAX インタフェースを介して実行できます。

W3C の DOM 勧告のサポート

XML にアクセスおよび操作するためのすべての Oracle XML DB API は、W3C で承認された標準の XML 処理要件に準拠します。PL/SQL DOM は、W3C の DOM 仕様のレベル 1.0 およびレベル 2.0 をサポートします。

- Oracle9*i* データベース リリース 1（9.0.1）では、XDK for PL/SQL に DOM レベル 1.0 およびレベル 2.0 の一部が実装していました。
- Oracle9*i* データベース リリース 2（9.2）では、PL/SQL API for `XMLType` に DOM レベル 1.0 およびレベル 2.0 コアが実装され、`XMLType` API の拡張機能を介して Oracle9*i* データベースに完全に統合されています。

次に、それぞれのレベルを簡単に説明します。

- **DOM レベル 1.0**: 1998 年 10 月に勧告された、DOM 仕様の最初の形式レベルです。レベル 1.0 では、XML 1.0 および HTML へのサポートが定義されています。

- **DOM レベル 2.0**: 2000 年 11 月に勧告されました。レベル 2.0 では、名前空間を含む XML 1.0 をサポートすることによって、レベル 1.0 が拡張されています。また、カスケーディング・スタイルシート（CSS）およびイベント（ユーザー・インタフェース・イベントおよびツリー操作イベント）のサポートが追加され、ツリー操作（ツリーの範囲および検索メカニズム）が拡張されています。
- **DOM レベル 3.0**: 現在作成中です。レベル 3.0 では、名前空間を含む XML 1.0 を完全にサポート（XML Information Set との整合および XML Base のサポート）することによって、レベル 2.0 が拡張されます。また、ユーザー・インタフェース・イベント（キーボード）が拡張されます。さらに、抽象スキーマ（DTD および XML Schema）のサポート、および文書や抽象スキーマをロードおよび保存する機能も追加されます。この勧告では、複合マークアップ・ボキャブラリおよび DOM API（埋込み DOM）の組込みを検討中であり、XPath がサポートされる予定です。

DOM と SAX の違い

XML 用の一般的な API は、主に 2 つのカテゴリに分類できます。

- ツリーベース : DOM は、XML 用の主要な汎用ツリーベース API です。
- イベントベース : Simple API for XML（SAX）は、XML Parser と XML アプリケーション間の、主要な汎用イベントベース・プログラム・インタフェースです。

DOM は、オブジェクトを作成することによって機能します。これらのオブジェクトは子オブジェクトおよびプロパティを持ち、子オブジェクトはさらにその子オブジェクトおよびプロパティを持ち、以下同様に続きます。オブジェクトは、オブジェクト階層を下って検索するか、HTML 要素に ID 属性を明示的に指定することによって参照されます。次に例を示します。

```
<img src="employee_jdoe.gif" ID="0123jdoe">
```

次に、構造的な操作の例を示します。

- 要素の再順序付け
- 要素の追加または削除
- 属性の追加または削除
- 要素名の変更

PL/SQL DOM API for XMLType（DBMS_XMLDOM）：機能

PL/SQL DOM API for XMLType（DBMS_XMLDOM）のデフォルトの動作は、次のとおりです。

- DOM API によってアクセス可能な解析ツリーを生成します。
- DTD が検出された場合、XML Parser for PL/SQL は妥当性の検証を行います。検出されない場合は、検証を行わないパーサーになります。
- 解析が正常に実行されない場合、アプリケーション・エラーが発生します。
- このマニュアルに示すタイプおよびメソッドは、PL/SQL パッケージ DBMS_XMLPARSER に付属しています。

DTD の検証は、Oracle9*i* データベース リリース 1（9.0.1）の XDK を介して使用可能な XML Parser に対して公開されている規則と同じ規則に従います。異なるのは、対象のドキュメントが指示された時点で検証が実行されることのみです。たとえば、実体化の遅延を使用する場合、ドキュメントの使用時に検証が行われます。

Oracle XML DB は、XML テキストの SQL サポートおよび XML データの格納と取出し以外にも、Oracle XML 開発プラットフォームを拡張しています。今回のリリースでは、PL/SQL および Java で DOM を使用して XMLType インスタンスを操作できます。そのため、アプリケーションまたはプラグインに最適な言語を使用して、個別の XML 要素およびデータを直接操作できます。

今回のリリースでは、サーバーで C ベース表現の XML を使用し、XML Schema に基づく XML インスタンスを操作するために、PL/SQL DOM API が更新されています。Oracle XML DB の PL/SQL DOM API for XMLType および Java DOM API for XMLType は、W3C の DOM 勧告に準拠し、リレーショナル列またはオブジェクト・リレーショナル列に XMLType のメモリー内インスタンスとして XML の構造化記憶域を定義および実装します。W3C の DOM 勧告の詳細は、8-9 ページの「PL/SQL DOM API for XMLType の使用 : XML データの準備」を参照してください。

XML Schema のサポート

PL/SQL DOM API for XMLType は、XML Schema をサポートしています。Oracle XML DB は、XML Schema 内で注釈をメタデータとして使用し、XML 文書の構造とデータベース・スキーマに対するその文書のマッピングの両方を決定します。

注意： 下位互換性および柔軟性を得るために、PL/SQL DOM は、XML Schema に基づく文書と XML Schema に基づかない文書の両方をサポートします。

Oracle XML DB に XML Schema が登録されると、PL/SQL DOM API for XMLType は、ノード・オブジェクトの階層として XML 文書のメモリー内ツリー表現を構築します。各ツリー表現は、独自の特別なインタフェースを持ちます。多くのノード・オブジェクト型は子

ノード型を持つことができ、これによって、さらに特別な追加のインタフェースを実装できます。様々な型の子ノードを持つことができるノード型がありますが、リーフ・ノード以外のノードを持つことができず、文書構造内に子ノードを持つことができないノード型もあります。

パフォーマンスの向上

前述の機能に加えて、Oracle XML DB は、DOM を使用して、複数のバックエンド・データ・ソースと XML 間でデータを変換する標準的な方法を提供します。これによって、環境内の様々なデータ・ソースに個別の XML 変換方法を使用する必要がなくなります。XML データを交換する必要があるアプリケーションでは、1 つのネイティブな XML データベースを使用して、XML 文書をキャッシュできます。そのため、Oracle XML DB は、Web アプリケーションとバックエンド・データ・ソース間の中間キャッシュとして機能することによって、リレーショナル・データベースまたは異なるファイル・システムかにかかわらず、アプリケーションのパフォーマンスを向上させることができます。

参照： 第 9 章「Java API for XMLType」を参照してください。

XDK および Oracle XML DB を使用したエンド・トゥ・エンド・アプリケーションの設計

Oracle XML DB に基づくアプリケーションを構築する場合、XDK に追加のコンポーネントは必要ありません。ただし、XDK のコンポーネントと Oracle XML DB を組み合せて、エンド・トゥ・エンドで実行する XML 対応アプリケーションの統合スイートを配置することができます。たとえば、XDK の機能を次の処理に使用できます。

- Simple API for XML（SAX）インタフェース処理。SAX は、XML Parser によって提供され、手続き型およびイベント・ベース・アプリケーションによって使用される XML 標準インタフェースです。
- DOM インタフェース処理。再帰的な構造型オブジェクト・ベース処理に使用します。

Oracle XDK には、クライアント、ブラウザまたはプラグインで実行する XML 文書の読込み、操作、変換、表示などを行うアプリケーションを作成するための基本的な構成ブロックが含まれています。様々な配置オプションを提供するために、Oracle XDK は Java、JavaBeans、C、C++ および PL/SQL でも使用できます。多くのシェアウェアおよび試用版の XML コンポーネントとは異なり、Oracle XDK はフル・サポートされており、商用再配布ライセンスを受けています。

Oracle XDK for Java は、次のコンポーネントで構成されています。

- XML Parser: Java、C、C++ および PL/SQL をサポートします。このコンポーネントは、業界標準の DOM インタフェースおよび SAX インタフェースを使用して、XML を作成および解析します。

- XSLT プロセッサ : XML を、HTML などの他のテキストベース形式に変換またはレンダリングします。
- XML Schema プロセッサ : Java、C および C++ をサポートします。XML の単純型および複合型を使用可能にします。
- XML Class Generator: DTD および XML Schema から自動的に Java クラスおよび C++ クラスを生成し、Web フォームまたはアプリケーションから XML データを送信します。
- XML Transviewer JavaBeans: Java コンポーネントを使用して、XML 文書および XML データを表示および変換します。
- XML SQL Utility: Java をサポートします。SQL 問合せから、XML 文書、DTD および XML Schema を生成します。
- TransXUtility: インストールに有効な追加機能を使用して、XML にカプセル化されたデータをデータベースにロードします。
- XSQL Servlet: サーバー内の XML、SQL および XSLT を組み合せて、動的 Web コンテンツを配信します。

> **参照 :** 『Oracle9*i* XML Developer's Kit ガイド - XDK』を参照してください。

PL/SQL DOM API for XMLType の使用 : XML データの準備

Oracle XML DB DOM API を利用するには、Oracle XML DB で XML データからデータ・モデルを開発するための処理を実行する必要があります。この章では PL/SQL を中心に説明していますが、すべての言語に適用されます。実行する処理は、データの状態とアプリケーションの要件によって異なります。

Oracle XML DB で PL/SQL DOM API を使用するためにデータを準備するには、次の手順を実行します。

1. 標準の XML Schema を作成します（使用していない場合）。リレーショナル・データベースまたはオブジェクト・リレーショナル・データベースに定義された SQL オブジェクトの定義を、XML Schema の注釈として付けます。
2. XML Schema を登録して、必要なデータベース・マッピングを生成します。

これによって、次の操作を実行できます。

- `XMLType` ビューを使用して、既存のリレーショナル・データベースまたはオブジェクト・リレーショナル・データベースを XML 形式にラップできます。これによって、アプリケーションからアクセス可能な XML 構造を作成できるようになります。8-11 ページの「XMLType ビューを使用した XML への既存データのラッピング」を参照してください。
- XML 文書（およびフラグメント）を `XMLType` 列に挿入できます。

- Oracle XML DB の PL/SQL DOM API および Java DOM API を使用して、XMLType 列および表に格納された XML データにアクセスし、操作します。

標準の XML Schema を作成して登録すると、それに準拠した XML 文書をデータベースに挿入できます。データベースでは、XML 文書を分解、解析し、アプリケーションからアクセス可能なオブジェクト・リレーショナル列に格納できます。

SQL オブジェクト型への XML Schema のマッピングの生成

XML Schema は、任意のコンテキストで使用または参照する前に、登録する必要があります。XML Schema を登録すると、その中で宣言される要素および属性が、データベース・スキーマ内の対応する SQL オブジェクト型に含まれる個別の属性にマップされます。

登録の完了後、この XML Schema に準拠する XML 文書、および文書内の URL を使用して XML Schema を参照する XML 文書を、Oracle XML DB で処理することができます。この XML Schema に準拠する文書を格納するための表および列を、この XML Schema によって定義されたルート XML 要素に対して作成できます。

> **参照：** 詳細および例については、第 5 章「XMLType の構造化されたマッピング」を参照してください。

XML Schema を登録するには、DBMS_XMLSCHEMA パッケージで、スキーマ・ドキュメントおよびその URL（スキーマの場所ともいう）を指定します。ここで使用される URL は、データベース内に登録されたスキーマを一意に識別する名前であり、スキーマ・ドキュメントが実際に存在する URL である必要はありません。

また、スキーマのターゲットの名前空間は、要素および型が宣言される抽象名前空間を指定する別の URL（スキーマの場所を示す URL とは異なる）です。XML 文書のインスタンスは、ルート要素の名前空間と、この要素を定義するスキーマの場所（URL）の両方を指定する必要があります。

HTTP や FTP など、パスベースのプロトコルを使用して Oracle XML DB に文書のインスタンスが挿入されると、指定された XML Schema の名前および場所が事前に登録されていない場合は、その文書が準拠する XML Schema が暗黙的に登録されます。

> **参照：** 次のマニュアルを参照してください。
> - 『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』
> - 『Oracle9*i* XML API リファレンス - XDK および Oracle XML DB』

XML Schema のマッピングのための DOM 再現性

XML Schema 内で宣言された要素および属性は、対応する SQL オブジェクト型内の個別の属性にマップされますが、XML 文書内のエンコードされた一部の情報は、直接表現されません。DOM 検索を行うために、戻される XML 文書が元の文書と同じであること（DOM 再現性という）を保証するために、生成されるすべての SQL オブジェクト型に SYS_XDBPD$ というバイナリ属性が追加されます。この属性では、他の属性で格納されないすべての情報が格納されるため、Oracle XML DB に格納された XML 文書の DOM 再現性が保証されます。

XML Schema のマッピングでは表現されず、SYS_XDBPD$ によって処理されるデータには、次のものが含まれます。

- コメント
- 名前空間宣言
- 接頭辞情報

注意： このマニュアルの多くの例では、簡略化のため SYS_XDBPD$ 属性を省略しています。ただし、この属性は、XML Schema の登録処理によって生成された SQL オブジェクト型に常に存在します。

XMLType ビューを使用した XML への既存データのラッピング

既存のリレーショナル・データおよびオブジェクト・リレーショナル・データを XML アプリケーションで使用できるようにするには、既存のデータを XML 形式にラップするメカニズムを提供する XMLType ビューを作成します。これによって、要素およびエンティティが公開され、PL/SQL DOM API を使用してアクセスできるようになります。

XML の SQL オブジェクト型の間のマッピングを表す注釈を含む XML Schema を登録します。これによって、Oracle XML DB は、この XML Schema に準拠する XMLType ビューを作成できます。

参照： 第 11 章「XMLType ビュー」を参照してください。

PL/SQL DOM API for XMLType（DBMS_XMLDOM）のメソッド

表 8-1 に、リリース 2（9.2.0.2）でサポートされる PL/SQL DOM API for XMLType（DBMS_XMLDOM）のメソッドを示します。

リリース 2（9.2.0.2）でサポートされない DBMS_XMLDOM のメソッド

次の DBMS_XMLDOM のメソッドは、リリース 2（9.2.0.2）ではサポートされません。

- hasFeature
- getVersion
- setVersion
- getCharset
- setCharset
- getStandalone
- setStandalone
- writeExternalDTDToFile
- writeExternalDTDToBuffer
- writeExternalDTDToClob

表 8-2 に、リリース 2（9.2.0.2）でサポートされるその他のメソッドを示します。

表 8-1 リリース 2（9.2.0.2）でサポートされる DBMS_XMLDOM のメソッドの概要

グループ / メソッド	説明
ノード・メソッド	--
isNull()	ノードが NULL であるかどうかをテストします。
makeAttr()	ノードを属性にキャストします。
makeCDataSection()	ノードを CDATA セクションにキャストします。
makeCharacterData()	ノードを文字データにキャストします。
makeComment()	ノードをコメントにキャストします。
makeDocumentFragment()	ノードをドキュメント・フラグメントにキャストします。
makeDocumentType()	ノードをドキュメント・タイプにキャストします。
makeElement()	ノードを要素にキャストします。
makeEntity()	ノードをエンティティにキャストします。

表 8-1 リリース 2（9.2.0.2）でサポートされる DBMS_XMLDOM のメソッドの概要（続き）

グループ / メソッド	説明
makeEntityReference()	ノードを実体参照にキャストします。
makeNotation()	ノードを表記法にキャストします。
makeProcessingInstruction()	ノードを DOM 処理命令にキャストします。
makeText()	ノードを DOM テキストにキャストします。
makeDocument()	ノードを DOM 文書にキャストします。
writeToFile()	ノードのコンテンツをファイルに書き込みます。
writeToBuffer()	ノードのコンテンツをバッファに書き込みます。
writeToClob()	ノードのコンテンツを CLOB に書き込みます。
getNodeName()	ノードの名前を取得します。
getNodeValue()	ノードの値を取得します。
setNodeValue()	ノードの値を設定します。
getNodeType()	ノードのタイプを取得します。
getParentNode()	ノードの親ノードを取得します。
getChildNodes()	ノードの子ノードを取得します。
getFirstChild()	ノードの最初の子ノードを取得します。
getLastChild()	ノードの最後の子ノードを取得します。
getPreviousSibling()	ノードの以前の兄弟関係を取得します。
getNextSibling()	ノードの次の兄弟関係を取得します。
getAttributes()	ノードの属性を取得します。
getOwnerDocument()	ノードの所有者ドキュメントを取得します。
insertBefore()	参照先の子ノードの前に子を挿入します。
replaceChild()	古い子を新しい子ノードと置き換えます。
removeChild()	ノードから指定した子ノードを削除します。
appendChild()	ノードに新しい子ノードを追加します。
hasChildNodes()	ノードに子ノードが存在するかどうかをテストします。
cloneNode()	ノードを複製します。

表 8-1 リリース 2（9.2.0.2）でサポートされる DBMS_XMLDOM のメソッドの概要（続き）

グループ / メソッド	説明
名前付きノード・マップ・メソッド	--
isNull()	ノード・マップが NULL であるかどうかをテストします。
getNamedItem()	名前で指定された項目を取得します。
setNamedItem()	名前で指定された項目をマップに設定します。
removeNamedItem()	名前で指定された項目を削除します。
item()	マップ内の指定された索引の項目を取得します。
getLength()	マップ内の項目数を取得します。
ノード・リスト・メソッド	--
isNull()	ノード・リストが NULL であるかどうかをテストします。
item()	ノード・リスト内の指定された索引の項目を取得します。
getLength()	リスト内の項目数を取得します。
属性メソッド	--
isNull()	属性リストが NULL であるかどうかをテストします。
makeNode()	属性をノードにキャストします。
getQualifiedName()	属性の修飾名を取得します。
getNamespace()	属性の名前空間 URI を取得します。
getLocalName()	属性のローカル名を取得します。
getExpandedName()	属性の拡張名を取得します。
getName()	属性の名前を取得します。
getSpecifiied()	属性が所有する要素内で指定されているかどうかをテストします。
getValue()	属性値を取得します。
setValue()	属性値を設定します。
CDATA セクション・メソッド	--
isNull()isNull()	CDATA セクションが NULL であるかどうかをテストします。

表 8-1 リリース 2（9.2.0.2）でサポートされる DBMS_XMLDOM のメソッドの概要（続き）

グループ / メソッド	説明
makeNode()makeNode()	CDATA セクションをノードにキャストします。
文字データ・メソッド	--
isNull()	文字データが NULL であるかどうかをテストします。
makeNode()	文字データをノードにキャストします。
getData()	ノードのデータを取得します。
setData()	ノードにデータを設定します。
getLength()	データの長さを取得します。
substringData()	データの部分文字列を取得します。
appendData()	指定されたデータをノード・データに追加します。
insertData()	データをノード内の指定されたオフセットに挿入します。
deleteData()	指定されたオフセットのデータを削除します。
replaceData()	指定されたオフセットのデータを置換します。
コメント・メソッド	--
isNull()	コメントが NULL であるかどうかをテストします。
makeNode()	ノードにコメントをキャストします。
DOM インプリメンテーション・メソッド	--
isNull()	DOMImplementation ノードが NULL であるかどうかをテストします。
hasFeature()	指定された機能を DOM が実装しているかどうかをテストします。（このリリースではサポートされません。）
ドキュメント・フラグメント・メソッド	--
isNull()	ドキュメント・フラグメントが NULL であるかどうかをテストします。
makeNode()	ドキュメント・フラグメントをノードにキャストします。
ドキュメント・タイプ・メソッド	--
isNull()	ドキュメント・タイプが NULL であるかどうかをテストします。

表 8-1 リリース 2（9.2.0.2）でサポートされる DBMS_XMLDOM のメソッドの概要（続き）

グループ / メソッド	説明
makeNode()	ドキュメント・タイプをノードにキャストします。
findEntity()	ドキュメント・タイプ内で、指定されたエンティティを検索します。
findNotation()	ドキュメント・タイプ内で、指定された表記法を検索します。
getPublicId()	ドキュメント・タイプの公開識別子を取得します。
getSystemId()	ドキュメント・タイプのシステム識別子を取得します。
writeExternalDTDToFile()	DTD をファイルに書き込みます。
writeExternalDTDToBuffer()	DTD をバッファに書き込みます。
writeExternalDTDToClob()	DTD を CLOB に書き込みます。
getName()	ドキュメント・タイプの名前を取得します。
getEntities()	ドキュメント・タイプのエンティティのノード・マップを取得します。
getNotations()	ドキュメント・タイプの表記法のノード・マップを取得します。
要素メソッド	--
isNull()	要素が NULL であるかどうかをテストします。
makeNode()	要素をノードにキャストします。
getQualifiedName()	要素の修飾名を取得します。
getNamespace()	要素の名前空間 URI を取得します。
getLocalName()	要素のローカル名を取得します。
getExpandedName()	要素の拡張名を取得します。
getChildrenByTagName()	要素の子をタグ名で取得します。
getElementsByTagName()	サブツリーの要素を要素ごとに取得します。
resolveNamespacePrefix()	接頭辞を名前空間の URI に変換します。
getTagName()	要素のタグ名を取得します。
getAttribute()	名前で指定された属性ノードを取得します。
setAttribute()	名前で指定された属性を設定します。
removeAttribute()	名前で指定された属性を削除します。

表 8-1 リリース 2（9.2.0.2）でサポートされる DBMS_XMLDOM のメソッドの概要（続き）

グループ / メソッド	説明
getAttributeNode()	名前で指定された属性ノードを取得します。
setAttributeNode()	要素に属性ノードを設定します。
removeAttributeNode()	要素の属性ノードを削除します。
normalize()	要素の子テキストを正規化します。（このリリースではサポートされません。）
エンティティ・メソッド	--
isNull()	エンティティが NULL であるかどうかをテストします。
makeNode()	エンティティをノードにキャストします。
getPublicId()	エンティティの公開識別子を取得します。
getSystemId()	エンティティのシステム識別子を取得します。
getNotationName()	エンティティの表記法名を取得します。
実体参照メソッド	--
isNull()	実体参照が NULL であるかどうかをテストします。
makeNode()	実体参照を NULL にキャストします。
表記法メソッド	--
isNull()	表記法が NULL であるかどうかをテストします。
makeNode()	表記法をノードにキャストします。
getPublicId()	表記法の公開識別子を取得します。
getSystemId()	表記法のシステム識別子を取得します。
処理命令メソッド	--
isNull()	処理命令が NULL であるかどうかをテストします。
makeNode()	処理命令をノードにキャストします。
getData()	処理命令のデータを取得します。
getTarget()	処理命令のターゲットを取得します。
setData()	処理命令のデータを設定します。
テキスト・メソッド	--
isNull()	テキストが NULL であるかどうかをテストします。

表 8-1 リリース 2（9.2.0.2）でサポートされる DBMS_XMLDOM のメソッドの概要（続き）

グループ / メソッド	説明
makeNode()	テキストをノードにキャストします。
splitText()	テキスト・ノードのコンテンツを 2 つのテキスト・ノードに分割します。
文書メソッド	--
isNull()	文書が NULL であるかどうかをテストします。
makeNode()	文書をノードにキャストします。
newDOMDocument()	新しい文書を作成します。
freeDocument()	文書を解放します。
getVersion()	文書のバージョンを取得します。（このリリースではサポートされません。）
setVersion()	文書のバージョンを設定します。（このリリースではサポートされません。）
getCharset()	文書のキャラクタ・セットを取得します。（このリリースではサポートされません。）
setCharset()	文書のキャラクタ・セットを設定します。（このリリースではサポートされません。）
getStandalone()	文書が単独で指定されているかどうかを取得します。（このリリースではサポートされません。）
setStandalone()	文書を単独に設定します。（このリリースではサポートされません。）
writeToFile()	文書をファイルに書き込みます。
writeToBuffer()	文書をバッファに書き込みます。
writeToClob()	文書を CLOB に書き込みます。
writeExternalDTDToFile()	文書の DTD をファイルに書き込みます。（このリリースではサポートされません。）
writeExternalDTDToBuffer()	文書の DTD をバッファに書き込みます。（このリリースではサポートされません。）
writeExternalDTDToClob()	文書の DTD を CLOB に書き込みます。（このリリースではサポートされません。）
getDoctype()	文書の DTD を取得します。
getImplementation()	DOM インプリメンテーションを取得します。
getDocumentElement()	文書のルート要素を取得します。

表 8-1 リリース 2（9.2.0.2）でサポートされる DBMS_XMLDOM のメソッドの概要（続き）

グループ / メソッド	説明
createElement()	新しい要素を作成します。
createDocumentFragment()	新しいドキュメント・フラグメントを作成します。
createTextNode()	テキスト・ノードを作成します。
createComment()	コメント・ノードを作成します。
createCDATASection()	CDATA セクション・ノードを作成します。
createProcessingInstruction()	処理命令を作成します。
createAttribute()	属性を作成します。
createEntityReference()	実体参照を作成します。
getElementsByTagName()	要素をタグ名で取得します。

表 8-2 リリース 2（9.2.0.2）で追加された DBMS_XMLDOM のメソッド

メソッド	構文
createDocument	FUNCTION createDocument (namspaceURI IN VARCHAR2, qualifiedName IN VARCHAR2, doctype IN DOMType :=NULL) REURN DocDocument;
getPrefix	FUNCTION getPrefix(n DOMNode) RETURN VARCHAR2;
setPrefix	PROCEDURE setPrefix (n DOMNode) RETURN VARCHAR2;
hasAttributes	FUNCTION hasAttributes (n DOMNode) RETURN BOOLEAN;
getNamedItem	FUNCTION getNamedItem (nnm DOMNamedNodeMap, name IN VARCHAR2, ns IN VARCHAR2) RETURN DOMNode;
setNamedItem	FUNCTION getNamedItem (nnm DOMNamedNodeMap, arg IN DOMNode, ns IN VARCHAR2) RETURN DOMNode;
removeNamedItem	FUNCTION removeNamedItem (nnm DOMNamesNodeMap, name in VARCHAR2, ns IN VARCHAR2) RETURN DOMNode;
getOwnerElement	FUNCTION getOwnerElement (a DOMAttr) RETURN DOMElement;
getAttribute	FUNCTION getAttribute (elem DOMElement, name IN VARCHAR2, ns IN VARCHAR2) RETURN VARCHAR2;
hasAttribute	FUNCTION hasAttribute (elem DOMElement, name IN VARCHAR2) RETURN BOOLEAN;
hasAttribute	FUNCTION hasAttribute (elem DOMElement, name IN VARCHAR2, ns IN VARCHAR2) RETURN BOOLEAN;

表 8-2 リリース 2（9.2.0.2）で追加された DBMS_XMLDOM のメソッド（続き）

メソッド	構文
setAttribute	PROCEDURE setAttribute (elem DOMElement, name IN VARCHAR2, newvalue IN VARCHAR2, ns IN VARCHAR2);
removeAttribute	PROCEDURE removeAttribute (elem DOMElement, name IN VARCHAR2, ns IN VARCHAR2);
getAttributeNode	FUNCTION getAttributeNode(elem DOMElement, name IN VARCHAR2, ns IN VARCHAR2) RETURN DOMAttr;
setAttributeNode	FUNCTION setAttributeNode(elem DOMElement, newAttr IN DOMAttr, ns IN VARCHAR2) RETURN DOMAttr;
createElement	FUNCTION createElement (doc DOMDocument, tagname IN VARCHAR2, ns IN VARCHAR2) RETURN DOMElement;
createAttribute	FUNCTION createAttribute (doc DOMDocument, name IN VARCHAR2, ns IN VARCHAR2) RETURN DOMAttr;

PL/SQL DOM API for XMLType（DBMS_XMLDOM）の例外

次に、PL/SQL DOM API for XMLType（DBMS_XMLDOM）の例外を示します。詳細は、『Oracle9*i* XML Developer's Kit ガイド - XDK』を参照してください。

例外は、以前のリリースから変更されていません。

- INDEX_SIZE_ERR
- DOMSTRING_SIZE_ERR
- HIERARCHY_REQUEST_ERR
- WRONG_DOCUMENT_ERR
- INVALID_CHARACTER_ERR
- NO_DATA_ALLOWED_ERR
- NO_MODIFICATION_ALLOWED_ERR
- NOT_FOUND_ERR
- NOT_SUPPORTED_ERR
- INUSE_ATTRIBUTE_ERR

PL/SQL DOM API for XMLType: ノード型

DOM 仕様では、「ドキュメント」という用語は、DOM の対象となる様々な情報またはデータのコンテナを示すために使用されます。DOM は、XML 文書コンテナ内の要素を使用して、オブジェクト・ベースのツリー構造を作成し、XML 文書に格納されたオブジェクトを管理および使用するためのインタフェースを定義および公開する方法を指定します。また、DOM は様々なシステムの文書の格納をサポートします。

`getNodeType(myNode)` などの要求が指定されると、親ノードによってサポートされるノード型 `myNodeType` が戻されます。次の定数は、ノードが使用可能な様々な型を表します。

- `ELEMENT_NODE`
- `ATTRIBUTE_NODE`
- `TEXT_NODE`
- `CDATA_SECTION_NODE`
- `ENTITY_REFERENCE_NODE`
- `ENTITY_NODE`
- `PROCESSING_INSTRUCTION_NODE`
- `COMMENT_NODE`
- `DOCUMENT_NODE`
- `DOCUMENT_TYPE_NODE`
- `DOCUMENT_FRAGMENT_NODE`
- `NOTATION_NODE`

表 8-3 に、XML および HTML のノード型と、それに対応する使用可能な子ノード型を示します。

表 8-3 XML および HTML の DOM ノード型および対応する子ノード型

ノード型	子ノード型
Document	Element（1 つ以下）、ProcessingInstruction、Comment、DocumentType（1 つ以下）
DocumentFragment	Element、ProcessingInstruction、Comment、Text、CDATASection、EntityReference
DocumentType	子を持ちません。
EntityReference	Element、ProcessingInstruction、Comment、Text、CDATASection、EntityReference

表 8-3 XML および HTML の DOM ノード型および対応する子ノード型 （続き）

ノード型	子ノード型
Element	Element、Text、Comment、ProcessingInstruction、CDATASection、EntityReference
Attr	Text、EntityReference
ProcessingInstruction	子を持ちません。
Comment	子を持ちません。
Text	子を持ちません。
CDATASection	子を持ちません。
Entity	Element、ProcessingInstruction、Comment、Text、CDATASection、EntityReference
Notation	子を持ちません。

Oracle XML DB の DOM API for XMLType は、次のインタフェースも指定します。

- **`NodeList` インタフェース :** 次のような、順序付けられたノードのリストを処理します。
  - ノードの子
  - 要素インタフェースの `getElementsByTagName` メソッドによって戻された要素
- **`NamedNodeMap` インタフェース :** 要素の属性など、名前属性によって参照される、順序付けられていない一連のノードを処理します。

XML Schema に基づく XML インスタンスの操作

今回のリリースでは、キャラクタ・セット変換およびファイル・システムへの入出力用に、いくつかの拡張機能が導入されています。前述のとおり、以前のリリースの PL/SQL Parser API に対して作成されたアプリケーションは動作しますが、次の項で説明する変更を加える必要があります。

PL/SQL API for `XMLType` は、XML Schema に基づく XML インスタンスを操作するために最適化されています。

指定された `XMLType` 値で DOM Document 文書ハンドルを構成する新機能 `newDOMDocument` が提供されています。

PL/SQL アプリケーションの一般的な使用例は次のとおりです。

1. `XMLType` インスタンスのフェッチまたは構成
2. `XMLType` インスタンス上での `DOMDocument` ノードの構成
3. DOM API を使用した XML データへのアクセスおよび操作

注意： DOMDocument では、ノード型は XML フラグメントのデータ自体ではなく、XML フラグメントへのハンドルを表します。

たとえば、ノード値をコピーすると、DOMDocument によって、基礎となる同じデータへのハンドルもコピーされます。一方のハンドルによって変更されたデータは、もう一方のハンドルでアクセスしても参照できます。DOMDocument ハンドルが構成された XMLType 値は実際のデータであり、このハンドルに対して行われたすべての DOM 操作の結果が反映されます。

DOM NodeList オブジェクトおよび NamesNodeMap オブジェクト

DOM の NodeList オブジェクトおよび NamedNodeMap オブジェクトは常に最新です。基礎となる文書構造への変更は、関連するすべての NodeList オブジェクトおよび NamedNodeMap オブジェクトに反映されます。

たとえば、DOM ユーザーが要素の子を含む NodeList オブジェクトを取得し、その要素に子を追加（または削除や変更）した場合、これらの変更は自動的に NodeList に伝播されます。同様に、ツリー内のノードへの変更は、NodeList オブジェクトおよび NamedNodeMap オブジェクトのノードへのすべての参照に伝播されます。

Text、Comment および CDATASection インタフェースは、CharacterData インタフェースを継承します。

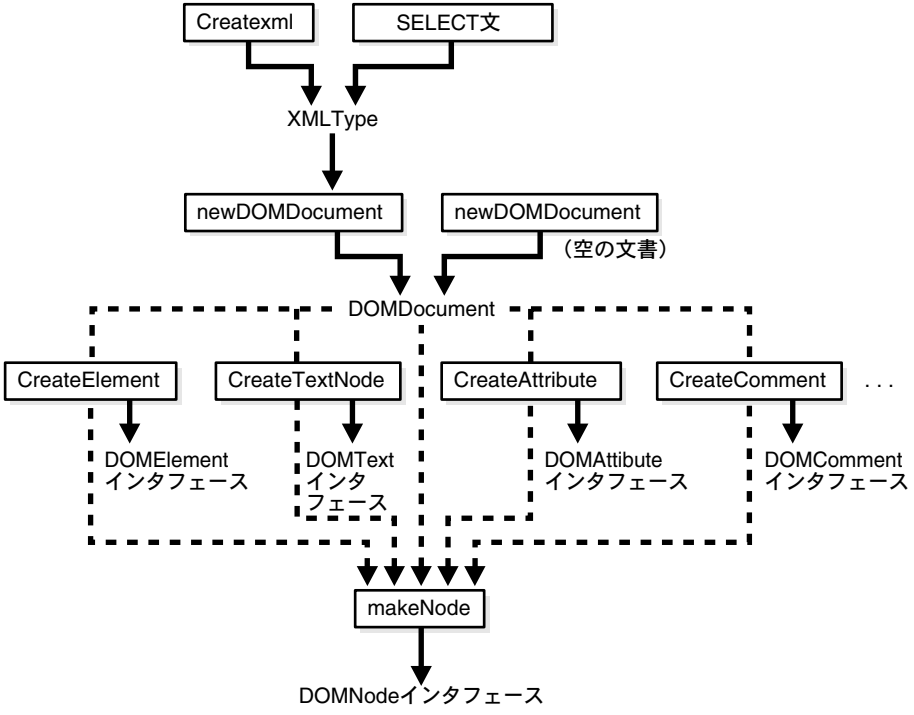
PL/SQL DOM API for XMLType（DBMS_XMLDOM）：コール順序

図 8-1 に、PL/SQL DOM API for XMLType（DBMS_XMLDOM）のコール順序を示します。

DOM 文書（DOMDocument）は、既存の XMLType から作成するか、または空の文書として作成できます。

1. newDOMDocument プロシージャが、XMLType または空の文書を処理します。これによって、DOMDocument が作成されます。
2. DOM API メソッド（createElement、createText、createAttribute、createComment など）を使用して、DOM ツリーを検索および拡張します。使用可能なメソッドの完全なリストは、表 8-1 を参照してください。
3. これらのメソッド（DOMElement、DOMText など）の結果を makeNode に渡して、DOMNode インタフェースを取得することもできます。

図 8-1 PL/SQL DOM API for XMLType: コール順序



PL/SQL DOM API for XMLType の例

例 8-1 DOM 文書の作成および操作

次の例では、要素の例 PERSON に DOMDocument ハンドルを作成します。

```
-- This example illustrates how to create a DOMDocument handle for an example
element PERSON:
declare
  var       XMLType;
  doc       dbms_xmldom.DOMDocument;
  ndoc      dbms_xmldom.DOMNode;
  docelem   dbms_xmldom.DOMElement;
  node      dbms_xmldom.DOMNode;
  childnode dbms_xmldom.DOMNode;
  nodelist  dbms_xmldom.DOMNodelist;
  buf       varchar2(2000);
```

```
begin
   var := xmltype('<PERSON> <NAME> ramesh </NAME> </PERSON>');

   -- Create DOMDocument handle:
   doc      := dbms_xmldom.newDOMDocument(var);
   ndoc     := dbms_xmldom.makeNode(doc);

   dbms_xmldom.writetobuffer(ndoc, buf);
   dbms_output.put_line('Before:'||buf);

   docelem := dbms_xmldom.getDocumentElement( doc );

   -- Access element:
   nodelist := dbms_xmldom.getElementsByTagName(docelem, 'NAME');
   node := dbms_xmldom.item(nodelist, 0);
   childnode := dbms_xmldom.getFirstChild(node);

   -- Manipulate:
   dbms_xmldom.setNodeValue(childnode, 'raj');

   dbms_xmldom.writetobuffer(ndoc, buf);
   dbms_output.put_line('After:'||buf);
end;
/
```

例 8-2 sys.xmltype を使用した DOM 文書の作成

次の例では、XMLType から DOM 文書を作成します。

```
declare
 doc  dbms_xmldom.DOMDocument;

 buf   varchar2(32767);

begin
  -- new document
  doc := dbms_xmldom.newDOMDocument(sys.xmltype('<person> <name>Scott</name>
  </person>'));
  dbms_xmldom.writeToBuffer(doc, buf);
  dbms_output.put_line(buf);
end;
/
```

例 8-3 要素ノードの作成

次の例では、空の DOM 文書から開始する要素ノードを作成します。

```
declare
 doc            dbms_xmldom.DOMDocument;
 elem           dbms_xmldom.DOMElement;
 nelem          dbms_xmldom.DOMNode;
begin
    -- new document
       doc := dbms_xmldom.newDOMDocument;

    -- create a element node
       elem := dbms_xmldom.createElement(doc, 'ELEM');

    -- make node
       nelem := dbms_xmldom.makeNode(elem);
       dbms_output.put_line(dbms_xmldom.getNodeName(nelem));
       dbms_output.put_line(dbms_xmldom.getNodeValue(nelem));
       dbms_output.put_line(dbms_xmldom.getNodeType(nelem));
end;
/
```

PL/SQL Parser API for XMLType（DBMS_XMLPARSER）

XML 文書は、エンティティというデータの格納単位で構成され、各エンティティには解析対象または解析対象外のデータが含まれます。解析対象データは文字で構成され、中には文書内の文字データを形成したり、タグを形成するものもあります。タグ付けは、文書のデータの格納レイアウトおよび論理構造の記述をエンコーディングします。XML は、データ格納レイアウトおよび論理構造を制約するメカニズムを提供します。

XML パーサーまたはプロセッサというソフトウェア・モジュールは、XML 文書を読み込み、その文書のコンテンツおよび構造にアクセスします。XML パーサーは、他のモジュール（通常はアプリケーション）のかわりに作業を行います。

PL/SQL Parser API for XMLType: 機能

一般に、PL/SQL Parser API for XMLType（DBMS_XMLPARSER）は、次のタスクを実行します。

- PL/SQL API によってアクセス可能な結果ツリーの構築
- 解析が正常に実行されない場合のエラーの表示

表 8-4 に、PL/SQL Parser API for XMLType（DBMS_XMLPARSER）のメソッドを示します。

表 8-4 DBMS_XMLPARSER のメソッド

メソッド	引数、戻り値および結果
parse	引数 : (url VARCHAR2) 結果 : 指定した URL またはファイルに格納された XML を解析し、構築された DOM 文書を戻します。
newParser	戻り値 : 新しいパーサーのインスタンス
parse	引数 : (p Parser, url VARCHAR2) 結果 : 指定した URL またはファイルに格納された XML を解析します。
parseBuffer	引数 : (p Parser, doc VARCHAR2) 結果 : 指定したバッファに格納された XML を解析します。
parseClob	引数 : (p Parser, doc CLOB) 結果 : 指定した CLOB に格納された XML を解析します。
parseDTD	引数 : (p Parser, url VARCHAR2, root VARCHAR2) 結果 : 指定した URL またはファイルに格納された XML を解析します。
parseDTDBuffer	引数 : (p Parser, dtd VARCHAR2, root VARCHAR2) 結果 : 指定したバッファに格納された XML を解析します。
parseDTDClob	引数 : (p Parser, dtd CLOB, root VARCHAR2) 結果 : 指定した CLOB に格納された XML を解析します。
setBaseDir	引数 : (p Parser, dir VARCHAR2) 結果 : 関連 URL の解決に使用されるベース・ディレクトリを設定します。
showWarnings	引数 : (p Parser, yes BOOLEAN) 結果 : 警告のオンまたはオフを切り替えます。
setErrorLog	引数 : (p Parser, fileName VARCHAR2) 結果 : エラーが指定ファイルに送信されるように設定します。
setPreserveWhitespace	引数 : (p Parser, yes BOOLEAN) 結果 : 空白保存モードを設定します。
setValidationMode	引数 : (p Parser, yes BOOLEAN) 結果 : 検証モードを設定します。

表 8-4 DBMS_XMLPARSER のメソッド（続き）

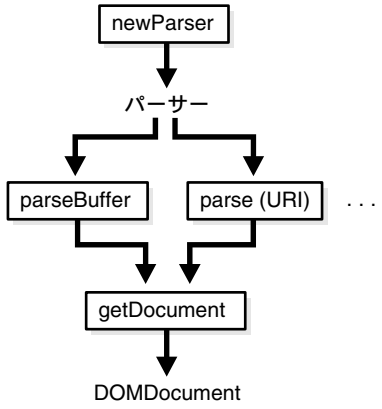
メソッド	引数、戻り値および結果
getValidationMode	引数 : (p Parser) 結果 : 検証モードを取得します。
setDoctype	引数 : (p Parser, dtd DOMDocumentType) 結果 : DTD を設定します。
getDoctype	引数 : (p Parser) 結果 : DTD を取得します。
getDocument	引数 : (p Parser) 結果 : DOM 文書を取得します。
freeParser	引数 : (p Parser) 結果 : パーサー・オブジェクトを解放します。

PL/SQL Parser API for XMLType（DBMS_XMLPARSER）: コール順序

図 8-2 に、PL/SQL Parser for XMLType（DBMS_XMLPARSER）のコール順序を示します。

1. newParser メソッドを使用して、パーサーのインスタンスを構成します。
2. parseBuffer、parseClob、parse(URI) などのメソッドを持つパーサーを使用して、XML 文書を解析します。パーサーのメソッドの完全なリストは、表 8-4 を参照してください。
3. 入力された文書が妥当な XML 文書ではない場合、エラーを戻されます。
4. 解析済の XML 文書インスタンスで PL/SQL DOM API for XMLType を使用するには、パーサーで getDocument をコールして、DOMDocument インタフェースを取得する必要があります。

図 8-2 PL/SQL Parser API for XMLType: コール順序



PL/SQL Parser API for XMLType の例

例 8-4 XML 文書の解析

次の例では、単純な XML 文書を解析し、DOM API を使用可能にします。

```
declare
 indoc       VARCHAR2(2000);
 indomdoc    dbms_xmldom.domdocument;
 innode      dbms_xmldom.domnode;
 myParser    dbms_xmlparser.Parser;
begin
   indoc   := '<emp><name> Scott </name></emp>';
   myParser := dbms_xmlparser.newParser;
   dbms_xmlparser.parseBuffer(myParser, indoc);
   indomdoc    := dbms_xmlparser.getDocument(myParser);
   innode      := dbms_xmldom.makeNode(indomdoc);
-- DOM APIs can be used here
end;
/
```

PL/SQL XSLT Processor for XMLType（DBMS_XSLPROCESSOR）

W3C の XSL 勧告では、ソース・ツリーから結果ツリーへ変換する規則が記載されています。eXtensible Stylesheet Language Transformation（XSLT）で表現される変換を、XSLT スタイルシートといいます。指定された変換は、XSLT スタイルシートに定義したテンプレートとパターンを対応付けることによって実行されます。結果ツリーの一部を作成するために、テンプレートはインスタンス化されます。

XSLT を使用した変換

Oracle XML DB の PL/SQL DOM API for `XMLType` は、XSLT もサポートします。これによって、ある XML 文書を別の XML、または HTML や PDF などのその他の形式に変換できます。XSLT は、XML をブラウザで表示するために、HTML に変換するためにも広く使用されます。

埋込み XSLT プロセッサは、eXtensible Stylesheet Language（XSL）文に従い、DOM ツリー構造内の `XMLType` に存在する XML データを検索します。Oracle XML DB アプリケーションでは、以前のリリースで必要であった XML Parser for PL/SQL のような個別のパーサーは必要ありません。ただし、外部処理を必要とするアプリケーションでは、最初に XML Parser for PL/SQL を使用して文書構造を公開することができます。

注意： Oracle XDK の XML Parser for PL/SQL は、XML 文書（または単独の DTD）を解析して、通常はクライアントで実行しているアプリケーションでこの文書を処理できるようにします。PL/SQL API for `XMLType` は、サーバーで実行され、データベースにネイティブに統合されたアプリケーションに使用されます。これによるメリットには、パフォーマンスの向上、アクセスや操作のオプションの拡張などが含まれます。

参照： 付録 D「XSLT の手引き」を参照してください。

PL/SQL XSLT Processor for XMLType: 機能

PL/SQL XSLT Processor for `XMLType`（`DBMS_XSLPROCESSOR`）は、Oracle XML DB による XSLT プロセッサの実装です。これは、W3C の XSLT 最終勧告（REC-xslt-19991116）に準拠しています。この最終勧告には、XSLT スタイルシートの読込み方法および変換方法に関して、XSLT プロセッサに必要な動作が記載されています。

PL/SQL XSLT Processor の型およびメソッドは、PL/SQL パッケージ `DBMS_XSLPROCESSOR` に付属しています。

PL/SQL XSLT Processor API（DBMS_XSLPROCESSOR）：メソッド

PL/SQL XSLT Processor API（DBMS_XSLPROCESSOR）のメソッドでは、XSLT プロセッサの実装に固有の 2 つの PL/SQL 型（Processor 型および Stylesheet 型）が使用されます。

表 8-5 に、PL/SQL XSLT Processor（DBMS_XSLPROCESSOR）のメソッドを示します。

注意： メソッドの宣言と引数の間には、次のように、空白が入りません。

```
processXSL(p Processor, ss Stylesheet, xmldoc DOMDocument)
```

表 8-5 PL/SQL XSLT Processor（DBMS_XSLPROCESSOR）のメソッド

メソッド	引数、戻り値または結果
newProcessor	戻り値 : 新しいプロセッサ・インスタンス
processXSL	引数 : (p Processor, ss Stylesheet, xmldoc DOMDocument) 結果 : 指定した DOMDocument およびスタイルシートを使用して、入力された XML 文書を変換します。
processXSL	引数 : (p Processor, ss Stylesheet, xmldoc DOMDocumentFragment) 結果 : 指定した DOMDocumentFragment およびスタイルシートを使用して、入力された XML 文書を変換します。
showWarnings	引数 : (p Processor, yes BOOLEAN) 結果 : 警告のオン / オフを切り替えます。
setErrorLog	引数 : (p Processor, Filename VARCHAR2) 結果 : エラーが指定ファイルに送信されるように設定します。
NewStylesheet	引数 : (Input VARCHAR2, Reference VARCHAR2) 結果 : エラーが指定ファイルに送信されるように設定します。
transformNode	引数 : (n DOMNode, ss Stylesheet) 結果 : 指定したスタイルシートを使用して、DOM ツリー内のノードを変換します。
selectNodes	引数 : (n DOMNode, pattern VARCHAR2) 結果 : 指定したパターンに一致する DOM ツリーのノードを選択します。
selectSingleNodes	引数 : (n DOMNode, pattern VARCHAR2) 結果 : 指定したパターンに一致するツリーの最初のノードを選択します。

表 8-5 PL/SQL XSLT Processor（DBMS_XSLPROCESSOR）のメソッド（続き）

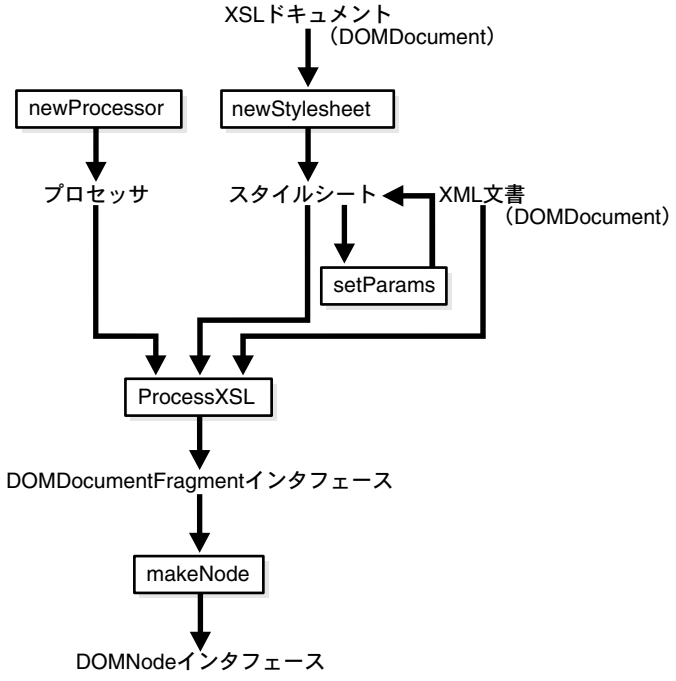
メソッド	引数、戻り値または結果
valueOf	引数 : (n DOMNode, pattern VARCHAR2) 結果 : 指定したパターンに一致するツリーの最初のノード値を取り出します。
setParam	引数 : (ss Stylesheet, name VARCHAR2, value VARCHAR2) 結果 : 最上位のスタイルシートのパラメータを設定します。
removeParam	引数 : (ss Stylesheet, name VARCHAR2) 結果 : 最上位のスタイルシートのパラメータを削除します。
ResetParams	引数 : (ss Stylesheet) 結果 : 最上位のスタイルシートのパラメータをリセットします。
freeStylesheet	引数 : (ss Stylesheet) 結果 : スタイルシート・オブジェクトを解放します。
freeProcessor	引数 : (p Processor) 結果 : プロセッサ・オブジェクトを解放します。

PL/SQL Parser API for XMLType（DBMS_XSLPROCESSOR）：コール順序

図 8-2 に、XSLT Processor for XMLType（DBMS_XSLPROCESSOR）のコール順序を示します。

1. メソッド newProcessor を使用して、XSLT プロセッサを構成します。
2. メソッド newStylesheet を使用して、DOM 文書からスタイルシートを構築します。
3. オプションで、コール setParams を使用して、スタイルシートにパラメータを設定します。
4. 手順 1 ～ 3 で作成されたプロセッサおよびスタイルシートを使用して、コール processXSL で XSLT 処理を実行します。
5. 変換する XML 文書をコール processXSL に渡します。
6. その結果作成された DOMDocumentFragment インタフェースは、PL/SQL DOM API for XMLType を使用して操作できます。

図 8-3 PL/SQL XSLT Processor for XMLType: コール順序



PL/SQL XSLT Processor for XMLType の例

例 8-5 XSLT スタイルシートを使用した XML 文書の変換

次の例では、processXSL コールを使用して XML 文書を変換します。予測される出力（タグ名順の XML）を次に示します。

```
<emp>
   <empno>1</empno>
   <fname>robert</fname>
   <job>engineer</job>
   <lname>smith</lname>
   <sal>1000</sal>
</emp>

declare
 indoc       VARCHAR2(2000);
 xsldoc      VARCHAR2(2000);
```

```
 myParser    dbms_xmlparser.Parser;
 indomdoc    dbms_xmldom.domdocument;
 xsltdomdoc  dbms_xmldom.domdocument;
 xsl         dbms_xslprocessor.stylesheet;
 outdomdocf  dbms_xmldom.domdocumentfragment;
 outnode     dbms_xmldom.domnode;
 proc        dbms_xslprocessor.processor;
 buf         varchar2(2000);
begin
   indoc      := '<emp><empno> 1</empno> <fname> robert </fname> <lname>
smith</lname> <sal>1000</sal> <job> engineer </job> </emp>';
   xsldoc     :=
   '<?xml version="1.0"?>
    <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
      <xsl:output encoding="utf-8"/>
      <!-- alphabetizes an xml tree -->
      <xsl:template match="*">
        <xsl:copy>
         <xsl:apply-templates select="*|text()">
         <xsl:sort select="name(.)" data-type="text" order="ascending"/>
         </xsl:apply-templates>
        </xsl:copy>
      </xsl:template>
      <xsl:template match="text()">
        <xsl:value-of select="normalize-space(.)"/>
      </xsl:template>
    </xsl:stylesheet>';

   myParser := dbms_xmlparser.newParser;
   dbms_xmlparser.parseBuffer(myParser, indoc);
   indomdoc   := dbms_xmlparser.getDocument(myParser);
   dbms_xmlparser.parseBuffer(myParser, xsldoc);
   xsltdomdoc := dbms_xmlparser.getDocument(myParser);
   xsl        := dbms_xslprocessor.newstylesheet(xsltdomdoc, '');
   proc       := dbms_xslprocessor.newProcessor;

   --apply stylesheet to DOM document
   outdomdocf := dbms_xslprocessor.processxsl(proc, xsl, indomdoc);
   outnode    := dbms_xmldom.makenode(outdomdocf);
-- PL/SQL DOM API for XMLType can be used here
   dbms_xmldom.writetobuffer(outnode, buf);
   dbms_output.put_line(buf);
end;
/
```

9

Java API for XMLType

この章では、JDBC を介した XMLType データのフェッチなど、Java で XMLType を使用する方法を説明します。

Java DOM API for XMLType の概要

Oracle XML DB は、Java ドキュメント・オブジェクト・モデル（DOM）API for XMLType をサポートします。XML Schema に基づく文書と XML Schema に基づかない文書に対する、クライアントおよびサーバー用の汎用 API です。この API は、Java パッケージ `oracle.xdb.dom` を使用して実装されます。

JDBC を使用して `XMLType` データにアクセスするには、クラス `oracle.xdb.XMLType` を使用します。

どの XML Schema にも準拠しない XML 文書には、すべての妥当な XML 文書を処理可能な Java DOM API for `XMLType` を使用できます。

> **参照：** 『Oracle9*i* XML API リファレンス - XDK および Oracle XML DB』を参照してください。

Java DOM API for XMLType

Java DOM API for `XMLType` は、Oracle XML DB への格納方法にかかわらず、すべての妥当な XML 文書を処理します。XML 文書は、XML Schema に基づくか、XML Schema に基づかないか（基礎となる記憶域の形式が何であるか）にかかわらず、アプリケーションで同様に表示されます。Java DOM API は、クライアントおよびサーバーで動作します。

第 8 章「PL/SQL API for XMLType」で説明したとおり、Oracle XML DB DOM API は、W3C の DOM レベル 1.0 およびレベル 2.0 のコア勧告に準拠しています。

リポジトリ内の XML 文書へのアクセス

Oracle XML DB Resource API for Java API によって、Java アプリケーションが Oracle XML DB Repository に格納された XML 文書にアクセスできます。ネーミングは、W3C の DOM 勧告で指定されている、DOM への Java バインディングに準拠しています。Oracle XML DB Repository の階層には、XML Schema に基づく文書と XML Schema に基づかない文書の両方の文書を格納できます。

> **参照：** 第 17 章「Oracle XML DB Resource API for Java」を参照してください。

Oracle9*i* データベースに格納された XML 文書へのアクセス（Java）

Oracle XML DB は、データベースに格納された XML データに Java アプリケーションでアクセスするための次の方法（Resource API for Java の一部の機能）を提供します。

JDBC の使用

Oracle XML DB 内の XML 文書を含む、Oracle9*i* データベース内のすべてのデータにアクセスするための、Java アプリケーション用の SQL ベースの方法です。`oracle.xdb.XMLType` クラスの `createXML()` メソッドを使用します。

JDBC を使用して Java アプリケーションで Oracle XML DB 内の XML 文書にアクセスする方法

JDBC ユーザーは、XMLType 表を問い合せて、SQL XMLType データ型でサポートされるすべてのメソッドをサポートする JDBC XMLType インタフェースを取得できます。Java（JDBC）API for XMLType インタフェースは、DOMDocument インタフェースを実装できます。

例 9-1 XMLType Java: JDBC を使用した XMLType 表の問合せ

次に、JDBC を使用して XMLType 表を問い合せる例を示します。

```
import oracle.xdb.XMLType;
            ...
   OraclePreparedStatement stmt = (OraclePreparedStatement)
conn.prepareStatement("select e.poDoc from po_xml_tab e");
       ResultSet rset = stmt.executeQuery();
       OracleResultSet orset = (OracleResultSet) rset;

while(orset.next())
        {
       // get the XMLType
       XMLType poxml = XMLType.createXML(orset.getOPAQUE(1));
       // get the XMLDocument as a string...
Document podoc = (Document)poxml.getDOM();
        }
```

例 9-2 XMLType Java: XMLType データの選択

次のいずれかの方法で、JDBC の XMLType データを選択できます。

- SQL で getClobVal() または getStringVal() を使用して、Java で oracle.sql.CLOB または java.lang.String として結果を取得します。次に、Java のコード例を示します。

```
DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

 Connection conn =
    DriverManager.getConnection("jdbc:oracle:oci8:@", "scott", "tiger");

 OraclePreparedStatement stmt =
    (OraclePreparedStatement) conn.prepareStatement(
          "select e.poDoc.getClobVal() poDoc, "+
                 "e.poDoc.getStringVal() poString "+
          " from po_xml_tab e");

ResultSet rset = stmt.executeQuery();
OracleResultSet orset = (OracleResultSet) rset;
```

```
while(orset.next())
        {
// the first argument is a CLOB
oracle.sql.CLOB clb = orset.getCLOB(1);

// the second argument is a string..
String poString = orset.getString(2);

// now use the CLOB inside the program
        }
```

- PreparedStatement で getOPAQUE() コールを使用して XMLType インスタンス全体を取得し、XMLType コンストラクタを使用して、そのインスタンスから oracle.xdb.XMLType クラスを作成します。これによって、XMLType クラスで Java 関数を使用して、データにアクセスできます。

```
import oracle.xdb.XMLType;
...

 OraclePreparedStatement stmt =
    (OraclePreparedStatement) conn.prepareStatement(
          "select e.poDoc from po_xml_tab e");

ResultSet rset = stmt.executeQuery();
OracleResultSet orset = (OracleResultSet) rset;

// get the XMLType
XMLType poxml = XMLType(orset.getOPAQUE(1));

// get the XML as a string...
String poString = poxml.getStringVal();
```

例 9-3 XMLType Java: XMLType データを直接戻す方法

次の例では、getObject を使用して ResultSet から XMLType を直接取得します。これは、最も簡単に ResultSet から XMLType を取得する方法です。

```
import oracle.xdb.XMLType;
...

 OraclePreparedStatement stmt =
    (OraclePreparedStatement) conn.prepareStatement(
          "select e.poDoc from po_xml_tab e");

ResultSet rset = stmt.executeQuery();
OracleResultSet orset = (OracleResultSet) rset;
```

```
while(orset.next())
        {

// get the XMLType
XMLType poxml = (XMLType)orset.getObject(1);

// get the XML as a string...
String poString = poxml.getStringVal();
        }
```

JDBC を使用したデータベースに格納されている XML 文書の操作

JDBC を使用して、XMLType データを更新、挿入および削除することもできます。

例 9-4 XMLType Java: XMLType データの更新、挿入および削除

次のいずれかの方法で、Java で XMLType データを挿入できます。

- CLOB または文字列を INSERT、UPDATE または DELETE 文にバインドし、SQL 内で XMLType コンストラクタを使用して XML インスタンスを作成します。

```
OraclePreparedStatement stmt =
    (OraclePreparedStatement) conn.prepareStatement(
        "update po_xml_tab set poDoc = XMLType(?) ");

// the second argument is a string..
String poString = "<PO><PONO>200</PONO><PNAME>PO_2</PNAME></PO>";

// now bind the string..
stmt.setString(1,poString);
stmt.execute();
```

- PreparedStatement で setObject()（または setOPAQUE()）を使用して、XMLType インスタンス全体を設定します。

```
import oracle.xdb.XMLType;
...
OraclePreparedStatement stmt =
    (OraclePreparedStatement) conn.prepareStatement(
        "update po_xml_tab set poDoc = ? ");

// the second argument is a string
String poString = "<PO><PONO>200</PONO><PNAME>PO_2</PNAME></PO>";
XMLType poXML = XMLType.createXML(conn, poString);

// now bind the string..
stmt.setObject(1,poXML);
stmt.execute();
```

例 9-5 XMLType Java: XMLType でのメタデータの取得

XMLType 値を選択する場合、JDBC が列を OPAQUE 型として記述します。列型の名前を選択し、それを XMLTYPE と比較すると、XMLType を処理しているかどうかを確認できます。

```
import oracle.sql.*;
import oracle.jdbc.*;
...
OraclePreparedStatement stmt =
    (OraclePreparedStatement) conn.prepareStatement(
        "select poDoc from po_xml_tab");

OracleResultSet rset = (OracleResultSet)stmt.exuecuteQuery();

// Now, we can get the resultset metadata
OracleResultSetMetaData mdata =
        (OracleResultSetMetaData)rset.getMetaData();

// Describe the column = the column type comes out as OPAQUE
// and column type name comes out as XMLTYPE
if (mdata.getColumnType(1) == OracleTypes.OPAQUE &&
    mdata.getColumnTypeName(1).compareTo("SYS.XMLTYPE") == 0)
{
   // we know it is an XMLtype
}
```

例 9-6 XMLType Java: XMLType 列の要素の更新

次の例では、XMLType 列に格納されている PurchaseOrder 内の discount 要素を更新します。この場合、Java（JDBC）および oracle.xdb.XMLType クラスを使用します。この例では、Java（JDBC）を使用して XMLType を挿入、更新および削除する方法も示します。パーサーを使用してメモリー内 DOM ツリーを更新し、更新された XML 値を列に書き込みます。

```
-- create po_xml_hist table to store old PurchaseOrders
create table po_xml_hist (
 xpo xmltype
);

/*
   DESCRIPTION
    Example for oracle.xdb.XMLType

   NOTES
   Have classes12.zip, xmlparserv2.jar, and oraxdb.jar in CLASSPATH

*/
```

```
import java.sql.*;
import java.io.*;

import oracle.xml.parser.v2.*;
import org.xml.sax.*;
import org.w3c.dom.*;

import oracle.jdbc.driver.*;
import oracle.sql.*;

import oracle.xdb.XMLType;

public class tkxmtpje
{

  static String conStr = "jdbc:oracle:oci8:@";
  static String user = "scott";
  static String pass = "tiger";
  static String qryStr =
        "SELECT x.poDoc from po_xml_tab x "+
        "WHERE  x.poDoc.extract('/PO/PONO/text()').getNumberVal()=200";

 static String updateXML(String xmlTypeStr)
  {
     System.out.println("\n==============================");
     System.out.println("xmlType.getStringVal():");
     System.out.println(xmlTypeStr);
     System.out.println("==============================");
     String outXML = null;
     try{
        DOMParser parser  = new DOMParser();
        parser.setValidationMode(false);
        parser.setPreserveWhitespace (true);

        parser.parse(new StringReader(xmlTypeStr));
        System.out.println("xmlType.getStringVal(): xml String is well-formed");

        XMLDocument doc = parser.getDocument();

        NodeList nl = doc.getElementsByTagName("DISCOUNT");

        for(int i=0;i<nl.getLength();i++){
           XMLElement discount = (XMLElement)nl.item(i);
           XMLNode textNode = (XMLNode)discount.getFirstChild();
           textNode.setNodeValue("10");
        }
```

```
      StringWriter sw = new StringWriter();
      doc.print(new PrintWriter(sw));

      outXML = sw.toString();

     //print modified xml
      System.out.println("\n==============================");
      System.out.println("Updated PurchaseOrder:");
      System.out.println(outXML);
      System.out.println("==============================");
     }
   catch ( Exception e )
   {
     e.printStackTrace(System.out);
   }
   return outXML;
 }

public static void main(String args[]) throws Exception
 {
   try{

       System.out.println("qryStr="+ qryStr);

       DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

       Connection conn =
         DriverManager.getConnection("jdbc:oracle:oci8:@", user, pass);

       Statement s = conn.createStatement();
       OraclePreparedStatement stmt;

       ResultSet rset = s.executeQuery(qryStr);
       OracleResultSet orset = (OracleResultSet) rset;

       while(orset.next()){

       //retrieve PurchaseOrder xml document from database
        XMLType xt = XMLType.createXML(orset.getOPAQUE(1));

        //store this PurchaseOrder in po_xml_hist table
        stmt = (OraclePreparedStatement)conn.prepareStatement(
                  "insert into po_xml_hist values(?)");

        stmt.setObject(1,xt);  // bind the XMLType instance
        stmt.execute();
```

```
          //update "DISCOUNT" element
          String newXML = updateXML(xt.getStringVal());

          // create a new instance of an XMLtype from the updated value
          xt = XMLType.createXML(conn,newXML);

         // update PurchaseOrder xml document in database
          stmt = (OraclePreparedStatement)conn.prepareStatement(
            "update po_xml_tab x set x.poDoc =? where "+
              "x.poDoc.extract('/PO/PONO/text()').getNumberVal()=200");

          stmt.setObject(1,xt);  // bind the XMLType instance
          stmt.execute();

          conn.commit();
          System.out.println("PurchaseOrder 200 Updated!");

         }

        //delete PurchaseOrder 1001
         s.execute("delete from po_xml x"+
            "where x.xpo.extract"+
               "('/PurchaseOrder/PONO/text()').getNumberVal()=1001");
         System.out.println("PurchaseOrder 1001 deleted!");
    }
    catch( Exception e )
    {
      e.printStackTrace(System.out);
    }
  }
}

----------------------
-- list PurchaseOrders
----------------------

set long 20000
set pages 100
select x.xpo.getClobVal()
from po_xml x;
```

更新された発注書の XML での結果は次のとおりです。

```
<?xml version = '1.0'?>
<PurchaseOrder>
  <PONO>200</PONO>
  <CUSTOMER>
```

```
    <CUSTNO>2</CUSTNO>
    <CUSTNAME>John Nike</CUSTNAME>
    <ADDRESS>
     <STREET>323 College Drive</STREET>
     <CITY>Edison</CITY>
     <STATE>NJ</STATE>
     <ZIP>08820</ZIP>
    </ADDRESS>
    <PHONELIST>
     <VARCHAR2>609-555-1212</VARCHAR2>
     <VARCHAR2>201-555-1212</VARCHAR2>
    </PHONELIST>
   </CUSTOMER>
   <ORDERDATE>20-APR-97</ORDERDATE>
   <SHIPDATE>20-MAY-97 12.00.00.000000 AM</SHIPDATE>
   <LINEITEMS>
    <LINEITEM_TYP LineItemNo="1">
     <ITEM StockNo="1004">
      <PRICE>6750</PRICE>
      <TAXRATE>2</TAXRATE>
     </ITEM>
     <QUANTITY>1</QUANTITY>
     <DISCOUNT>10</DISCOUNT>
    </LINEITEM_TYP>
    <LINEITEM_TYP LineItemNo="2">
     <ITEM StockNo="1011">
      <PRICE>4500.23</PRICE>
      <TAXRATE>2</TAXRATE>
     </ITEM>
     <QUANTITY>2</QUANTITY>
     <DISCOUNT>10</DISCOUNT>
    </LINEITEM_TYP>
   </LINEITEMS>
   <SHIPTOADDR>
    <STREET>55 Madison Ave</STREET>
    <CITY>Madison</CITY>
    <STATE>WI</STATE>
    <ZIP>53715</ZIP>
   </SHIPTOADDR>
  </PurchaseOrder>
```

例 9-7 XMLType 列の操作

この例では、次の操作を実行します。

- XMLType 表からの XMLType の選択
- XPath 式に基づく XMLType の一部の抽出
- 要素の有無の確認
- XSL に基づく別の XML 形式への XMLType の変換
- XML Schema に対する XMLType 文書の妥当性の確認

```
import java.sql.*;
import java.io.*;
import java.net.*;
import java.util.*;

import oracle.xml.parser.v2.*;
import oracle.xml.parser.schema.*;
import org.xml.sax.*;
import org.w3c.dom.*;

import oracle.xml.sql.dataset.*;
import oracle.xml.sql.query.*;
import oracle.xml.sql.docgen.*;
import oracle.xml.sql.*;

import oracle.jdbc.driver.*;
import oracle.sql.*;

import oracle.xdb.XMLType;

public class tkxmtpk1
{

  static String conStr = "jdbc:oracle:oci8:@";
  static String user = "tpjc";
  static String pass = "tpjc";
  static String qryStr = "select x.resume from t1 x where id<3";
  static String xslStr =
                "<?xml version='1.0' ?> " +
                "<xsl:stylesheet version='1.0' xmlns:xsl='http://www.w3.org/1
999/XSL/Transform'> " +
                "<xsl:template match='ROOT'> " +
                "<xsl:apply-templates/> " +
                "</xsl:template> " +
                "<xsl:template match='NAME'> " +
                "<html> " +
```

```
                "  <body> " +
                "      This is Test " +
                "  </body> " +
                "</html> " +
                "</xsl:template> " +
                "</xsl:stylesheet>";

 static void parseArg(String args[])
 {
   conStr = (args.length >= 1 ? args[0]:conStr);
   user = (args.length >= 2 ? args[1].substring(0, args[1].indexOf("/")):user);
   pass = (args.length >= 2 ? args[1].substring(args[1].indexOf("/")+1):pass);
   qryStr = (args.length >= 3 ? args[2]:qryStr);
 }
 /**
  * Print the byte array contents
  */
 static void showValue(byte[] bytes) throws SQLException
 {
   if (bytes == null)
     System.out.println("null");
   else if (bytes.length == 0)
     System.out.println("empty");
   else
   {
     for(int i=0; i<bytes.length; i++)
       System.out.print((bytes[i]&0xff)+" ");
     System.out.println();
   }
 }

public static void main(String args[]) throws Exception
 {
   tkxmjnd1 util = new tkxmjnd1();

   try{

       if( args != null )
           parseArg(args);

       //      System.out.println("conStr=" + conStr);
       System.out.println("user/pass=" + user + "/" +pass );
       System.out.println("qryStr="+ qryStr);

       DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

       Connection conn = DriverManager.getConnection(conStr, user, pass);
```

```
            Statement s = conn.createStatement();

            ResultSet rset = s.executeQuery(qryStr);
            OracleResultSet orset = (OracleResultSet) rset;
            OPAQUE xml;

            while(orset.next()){
              xml = orset.getOPAQUE(1);
              oracle.xdb.XMLType xt = oracle.xdb.XMLType.createXML(xml);

              System.out.println("Testing getDOM() ...");
              Document doc = xt.getDOM();
              util.printDocument(doc);

              System.out.println("Testing getBytesValue() ...");
              showValue(xt.getBytesValue());

              System.out.println("Testing existsNode() ...");
              try {
                System.out.println("existsNode(/)" + xt.existsNode("/", null));
              }
              catch (SQLException e) {
                System.out.println("Thin driver Expected exception: " + e);
              }

              System.out.println("Testing extract() ...");
              try {
                XMLType xt1 = xt.extract("/RESUME", null);
                System.out.println("extract RESUME: " + xt1.getStringVal());
                System.out.println("should be Fragment: " + xt1.isFragment());
              }
              catch (SQLException e) {
                System.out.println("Thin driver Expected exception: " + e);
              }

              System.out.println("Testing isFragment() ...");
              try {
                System.out.println("isFragment = " + xt.isFragment());
              }
              catch (SQLException e) {
                System.out.println("Thin driver Expected exception: " + e);
              }

              System.out.println("Testing isSchemaValid() ...");
              try {
                System.out.println("isSchemaValid(): " + xt.isSchemaValid(null, "RES
UME"));
```

```
          }
          catch (SQLException e) {
            System.out.println("Thin driver Expected exception: " + e);
          }

          System.out.println("Testing transform() ...");
          System.out.println("XSLDOC: \n" + xslStr + "\n");
          try {
          /*  XMLType xslDoc = XMLType.createXML(conn, xslStr);
            System.out.println("XSLDOC Generated");
            System.out.println("After transformation:\n" + (xt.transform(xslDoc,
 null)).getStringVal()); */
            System.out.println("After transformation:\n" + (xt.transform(null, n
ull)).getStringVal());
          }
          catch (SQLException e) {
            System.out.println("Thin driver Expected exception: " + e);
          }

          System.out.println("Testing createXML(conn, doc) ...");
          try {
            XMLType xt1 = XMLType.createXML(conn, doc);
            System.out.println(xt1.getStringVal());
          }
          catch (SQLException e) {
            System.out.println("Got exception: " + e);
          }

        }
    }
    catch( Exception e )
    {
      e.printStackTrace(System.out);
    }
  }
}
```

Java DOM API for XMLType の機能

Java DOM API を使用して Oracle XML DB から XML データを取得する場合、取得する XML データまたはファイルを表す XDBDocument オブジェクトを取得します。このドキュメント・インタフェースから、ドキュメントの要素を取得して、W3C の DOM 仕様に指定されたすべての操作を実行できます。DOM は、次のものに対して機能します。

- すべての形式の XML 文書
  - XML Schema に基づく文書
  - XML Schema に基づかない文書
- 文書で使用されるすべての形式の基礎となる記憶域
  - CLOB
  - BLOB
  - オブジェクト・リレーショナル

Java DOM API for XMLType は、XML オブジェクトの子およびプロパティ（名前、名前空間など）を取得するために、文書内でのディープ検索またはシャロウ検索をサポートします。Java DOM API for XMLType は DOM 2.0 勧告に準拠し、名前空間を認識します。

プログラムによる XML 文書の作成

Java API for XMLType を使用すると、アプリケーションではプログラムによって XML 文書を作成することができます。この方法では、アプリケーションは、登録済の XML Schema に基づく文書または XML Schema に基づかない文書のいずれかをその場で（動的に）作成できます。

XML Schema に基づく文書の作成

XML Schema に基づく文書を作成するには、Java DOM API for XMLType の拡張機能を使用して、使用する XML Schema URL を指定します。XML Schema に基づく文書では、作成中の DOM が指定された XML Schema に準拠するかどうか（適切な子が適切な文書に挿入されているか）を検証します。

注意： 今回のリリースでは、Java DOM API for XMLType は型および制約のチェックは実行しません。

DOM オブジェクトの作成後、Oracle XML DB Resource API for Java を使用して、DOM オブジェクトを Oracle XML DB Repository に保存できます。XML 文書は次の適切な形式で格納されます。

- CLOB または BLOB（XML Schema に基づかない文書の場合）
- XML Schema で指定された形式（XML Schema に基づく文書の場合）

例 9-8 Java DOM API for XMLType: DOM オブジェクトの作成および XML Schema で指定された形式での格納

次の例では、Java DOM API for XMLType を使用して DOM オブジェクトを作成し、それを XML Schema で指定された形式で格納します。この例では、XML Schema に対する検証は示されていません。

```
import oracle.xdb.XMLType;
...
OraclePreparedStatement stmt =
    (OraclePreparedStatement) conn.prepareStatement(
        "update po_xml_tab set poDoc = ? ");

// the second argument is a string
String poString = "<PO><PONO>200</PONO><PNAME>PO_2</PNAME></PO>";
XMLType poXML = XMLType.createXML(conn, poString);
Document poDOM = (Document)poXML.getDOM();

Element rootElem = poDOM.createElement("PO");
poDOM.insertBefore(poDOM, rootElem, null);

// now bind the string..
stmt.setObject(1,poXML);
stmt.execute();
```

JDBC/SQLJ

XMLType インスタンスは、oracle.xdb.XMLType によって Java で表現されます。XMLType インスタンスが JDBC を使用してフェッチされると、このインスタンスは提供された XMLType クラスのオブジェクトとして自動的に明示されます。同様に、このクラスのオブジェクトは、データ操作言語（DML）文の値としてバウンドできます。通常、この値は XMLType です。SQLJ クライアントでも同じ動作がサポートされます。

Java DOM API for XMLType のクラス

Oracle XML DB は、W3C の DOM レベル 2 勧告をサポートします。W3C 勧告に加えて、Oracle XML DB DOM API は、主にアプリケーションと Oracle XDK for Java の相互作用を円滑にする Oracle 固有の拡張機能も提供します。Oracle の拡張機能のリストは、次の URL を参照してください。

http://otn.oracle.com/docs/tech/xml/xdk_java/content.html

XDBDocument() は、インスタンス化された XML 文書用の DOM を表すクラスです。XMLType は、XDBDocument() クラスでファンクション getXMLType() を使用して XML 文書から取得できます。

表 9-1 に、Java DOM API for XMLType のクラスおよび実装される W3C の DOM インタフェースを示します。

表 9-1 Java DOM API for XMLType: クラス

Java DOM API for XMLType のクラス	W3C の DOM インタフェース勧告のクラス
oracle.xdb.dom.XDBDocument	org.w3c.dom.Document
oracle.xdb.dom.XDBCData	org.w3c.dom.CDataSection
oracle.xdb.dom.XDBComment	org.w3c.dom.Comment
oracle.xdb.dom.XDBPI	org.w3c.dom.ProcessingInstruction
oracle.xdb.dom.XDBText	org.w3c.dom.Text
oracle.xdb.dom.XDBEntity	org.w3c.dom.Entity
oracle.xdb.dom.DTD	org.w3c.dom.DocumentType
oracle.xdb.dom.XDBNotation	org.w3c.dom.Notation
oracle.xdb.dom.XDBNodeList	org.w3c.dom.NodeList
oracle.xdb.dom.XDBAttribute	org.w3c.dom.Attribute
oracle.xdb.dom.XDBDOMImplementation	org.w3c.dom.DOMImplementation
oracle.xdb.dom.XDBElement	org.w3c.dom.Element
oracle.xdb.dom.XDBNamedNodeMap	org.w3c.dom.NamedNodeMap
oracle.xdb.dom.XDBNode	org.w3c.dom.Node

サポートされない Java メソッド

リリース 2（9.2.0.1）のマニュアルには次のメソッドに関する説明が含まれていますが、現在はサポートされていません。

- `XDBDocument.getElementByID`
- `XDBDocument.importNode`
- `XDBNode.normalize`
- `XDBNode.isSupported`
- `XDBDomImplementation.hasFeature`

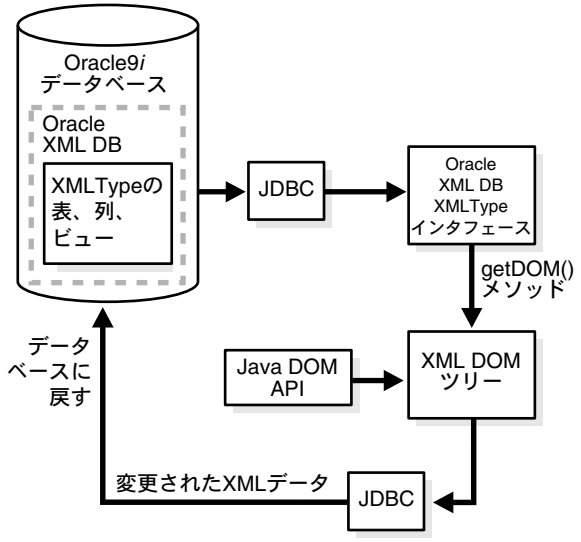
Java DOM API for XMLType: コール順序

次の Java DOM API for XMLType のコール順序の説明では、XML データが XML Schema に登録済であり、XMLType データ型の列に格納されていると想定しています。Java DOM API for XMLType を使用するには、次の手順を実行します。

1. XMLType 表または表の XMLType 列から XML データを取り出します。XML データをフェッチすると、Oracle XML DB は XMLType の DOMDocument インスタンスを作成し、文書を DOM ツリーに解析します。その後、Java DOM API for XMLType を使用して DOM ツリーの要素を操作できます。
2. Java DOM API for XMLType を使用して、DOM ツリーの要素を操作します。
3. Java DOM API for XMLType は、変更された XML データを Oracle XML DB に戻します。

図 9-1 に、Java DOM API for XMLType のコール順序を示します。

図 9-1 Java DOM API for XMLType: コール順序



第 IV 部

既存データの XML 表示

第 IV 部では、既存のデータを XML として表示する方法を説明します。第 IV 部に含まれる章は、次のとおりです。

- 第 10 章「データベースからの XML データの生成」
- 第 11 章「XMLType ビュー」
- 第 12 章「URL を介したデータの作成およびアクセス」

10

データベースからの XML データの生成

この章では、データベースから XML を生成するための Oracle XML DB オプションについて説明します。また、リレーショナル・コンテンツから XML データを生成するための SQLX 標準関数および Oracle が提供する関数とパッケージについて説明します。

この章の内容は次のとおりです。

Oracle9*i* データベースから XML データを生成するための Oracle XML DB オプション

Oracle9*i* データベースは、ネイティブな XML の生成をサポートします。今回のリリースでは、次の場所に XML データを生成または再生成するための新しいオプションがいくつか提供されています。

- Oracle9*i* データベース
- Oracle9*i* データベースの `XMLType` の列および表

図 10-1 に、Oracle9*i* データベースから XML を生成する場合に使用可能な Oracle XML DB オプションを示します。

SQLX 関数を使用した XML の生成

Oracle XML DB がサポートする SQLX 関数は次のとおりです。

- 「XMLElement() 関数」（10-5 ページ）
- 「XMLForest() 関数」（10-9 ページ）
- 「XMLConcat() 関数」（10-15 ページ）
- 「XMLAgg() 関数」（10-17 ページ）

SQLX の Oracle 拡張関数を使用した XML の生成

SQLX の Oracle 拡張関数は次のとおりです。

- 「XMLColAttVal() 関数」（10-19 ページ）

DBMS_XMLGEN を使用した XML の生成

Oracle XML DB は、提供された PL/SQL パッケージ **`DBMS_XMLGEN`** をサポートします。`DBMS_XMLGEN` は、SQL 問合せから XML を生成します。10-21 ページの「DBMS_XMLGEN を使用した Oracle9i データベースからの XML の生成」を参照してください。

SQL 関数を使用した XML の生成

Oracle XML DB は、Oracle が提供する次の SQL 関数もサポートします。これらの関数は、SQL 問合せから XML を生成します。

- 「SYS_XMLGEN() 関数」（10-42 ページ）。この関数は、行を操作し、XML 文書を生成します。

- 「SYS_XMLAGG() 関数」（10-51 ページ）。この関数は、行のグループを操作し、複数の XML 文書を 1 つに集約します。
- 「XMLSEQUENCE() 関数」（10-11 ページ）。この関数のカーソル・バージョンのみが XML を生成します。この関数は、SQLX 関数としても分類されます。

XSQL Pages パブリッシング・フレームワークを使用した XML の生成

Oracle9*i* データベースから XML を生成することもできます。10-51 ページの「XSQL Pages パブリッシング・フレームワークを使用した XML の生成」を参照してください。

XSQL Pages パブリッシング・フレームワークは、XDK for Java の一部で、XSQL Servlet ともいいます。

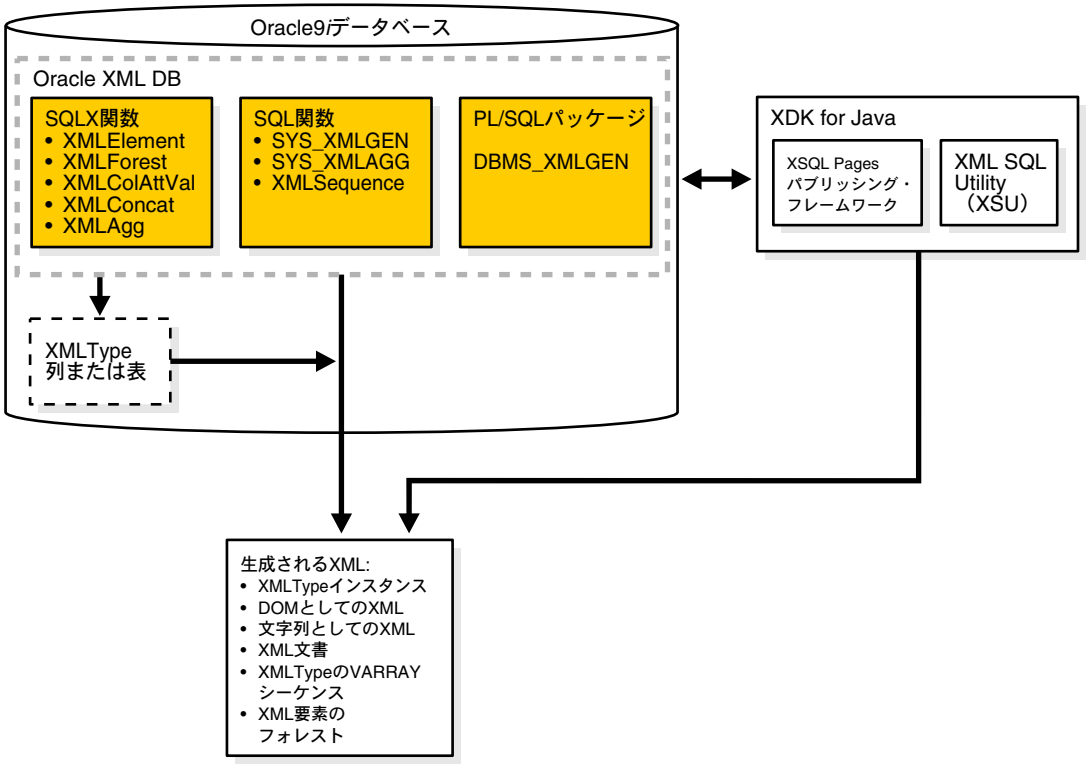
XML SQL Utility（XSU）を使用した XML の生成

XML SQL Utility（XSU）を使用すると、`XMLType` の表および列内のデータに次のタスクを実行できます。

- オブジェクト・リレーショナル・データベースの表またはビューから取り出したデータを XML に変換する。
- XML 文書からデータを抽出し、正規マッピングを使用して、表またはビューの適切な列または属性にデータを挿入する。
- XML 文書からデータを抽出し、このデータを適用して適切な列または属性の値を更新または削除する。

> **参照：** 10-54 ページの「XML SQL Utility（XSU）を使用した XML の生成」を参照してください。

図 10-1 Oracle9*i* データベースから XML を生成するための Oracle XML DB オプション



参照： 次の章またはマニュアルを参照してください。

- 第 6 章「XMLType データの変換および検証」
- 第 8 章「PL/SQL API for XMLType」
- 第 9 章「Java API for XMLType」
- 『Oracle9*i* XML API リファレンス - XDK および Oracle XML DB』

SQLX 関数を使用したデータベースからの XML の生成

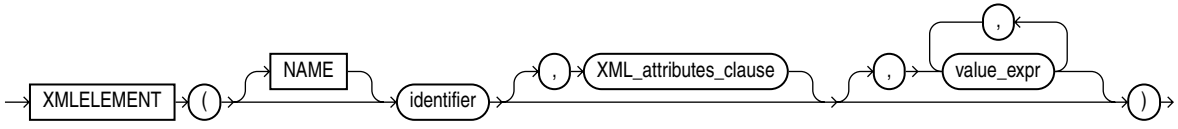
XMLElement()、XMLForest()、XMLConcat() および XMLAgg() は、XML に対する SQL 標準である SQLX 標準に準拠しています。これらの新しい関数の構文やセマンティクスは、標準に合わせて将来変更されることがあります。

すべての生成関数では、ユーザー定義型（UDT）が正規の XML 形式に変換されます。正規マッピングでは、ユーザー定義型の属性は XML 要素にマップされます。

XMLElement() 関数

XMLElement() 関数は、新しい XML の SQL 標準に基づいています。この関数は、要素名、その要素の属性のコレクション（オプション）、およびその要素の内容を構成する 0（ゼロ）以上の引数を取り、XMLType 型のインスタンスを戻します。図 10-2 を参照してください。XML_attributes_clause については、次の項で説明します。

図 10-2 XMLElement() の構文



XMLElement() は SYS_XMLGEN() と類似していますが、プロローグ（XML バージョン情報）を含む XML 文書を作成しないという点で SYS_XMLGEN() とは異なります。この関数には複数の引数を指定でき、戻された XML に属性を含めることができます。

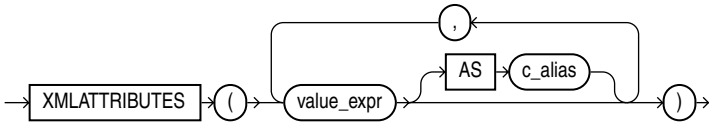
XMLElement() は、主に、リレーショナル・データから XML インスタンスを作成するために使用されます。この関数は、部分的にエスケープされた識別子を取り、作成するルート XML 要素名を指定します。この識別子は、列名または列の参照である必要はありませんが、式にはできません。指定した識別子が NULL である場合、要素は戻されません。

SQL 識別子から妥当な XML 要素名を生成する場合、XML 要素名に無効な文字はエスケープされます。部分的なエスケープでは、XML では表示できないコロン（:）以外の SQL 識別子は、シャープ（#）とその後に続く 16 進数文字の Unicode 表現を使用してエスケープされます。これは、生成される要素に名前空間の接頭辞を指定するために使用できます。完全にエスケープされたマッピングでは、SQL 識別子名の XML 以外のすべての文字（: 記号を含む）がエスケープされます。

XML_Attributes_Clause

XMLElement() は、オプションの XMLAttributes() 句も取ります。この句は、その要素の属性を指定します。この句の後には、新しく作成された要素の子を構成する値をリストすることができます。図 10-3 を参照してください。

図 10-3 XML_attributes_clause の構文



XMLAttributes() 句では、属性に対する値を取得するために値の式が評価されます。指定された値の式では AS 句が省略されている場合、完全にエスケープされた形式の列名が属性の名前として使用されます。AS 句が指定されている場合、部分的にエスケープされた形式の別名が属性の名前として使用されます。式が NULL であると評価された場合、その式に対して属性は作成されません。式の型には、オブジェクト型またはコレクションにすることはできません。

XMLAttributes() 句に続く値のリストは、XML 形式に変換され、最上位の要素の子になります。式が NULL であると評価された場合、その式に対して要素は作成されません。

例 10-1 XMLElement(): 各従業員の要素の生成

次の例では、従業員の名前をコンテンツとして持つ XML の Emp 要素を各従業員に 1 つ作成します。

```
SELECT e.employee_id, XMLELEMENT ( "Emp", e.fname ||' ' || e.lname ) AS "result"
   FROM employees e
   WHERE employee_id > 200;

-- This query produces the following typical result:
-- ID    result
-- --------------------
-- 1001 <Emp>John Smith</Emp>
-- 1206 <Emp>Mary Martin</Emp>
```

XMLElement() をネストして、ネストした構造の XML データを作成することもできます。

例 10-2 XMLElement(): ネストした XML の生成

従業員の名前および雇用日を指定する要素をコンテンツとして持つ Emp 要素を各従業員に作成するには、次の問合せを実行します。

```
SELECT XMLELEMENT("Emp", XMLELEMENT("name", e.fname ||' '|| e.lname),
                          XMLELEMENT ( "hiredate", e.hire)) AS "result"
FROM employees e
WHERE employee_id > 200 ;
```

この問合せによって、次の典型的な XML が結果として戻されます。

```
result
-----------------
<Emp>
  <name>John Smith</name>
  <hiredate>2000-05-24</hiredate>
</Emp>
<Emp>
  <name>Mary Martin</name>
  <hiredate>1996-02-01</hiredate>
</Emp>
```

> **注意：** 属性が指定されている場合は、XMLElement() の 2 つ目の引数に次のように表示されます。
>
> "XMLATTRIBUTES(attribute, ...)"

例 10-3 XMLElement(): ID および名前属性を含む各従業員の要素の生成

次の例では、id および name 属性を含む Emp 要素を各従業員に 1 つ作成します。

```
SELECT XMLELEMENT ( "Emp",
        XMLATTRIBUTES (e.id,e.fname ||' ' || e.lname AS "name")) AS "result"
    FROM employees e
    WHERE employee_id > 200;
```

この問合せによって、次の典型的な XML フラグメントが結果として戻されます。

```
result
--------------
<Emp ID="1001" name="John Smith"/>
<Emp ID="1206" name="Mary Martin"/>
```

要素または属性の名前が AS 句に指定した別名から作成されている場合、部分的にエスケープされたマッピングが使用されます。要素または属性の名前が列の参照から作成されている場合は、完全にエスケープされたマッピングが使用されます。

次の例では、これらのマッピングを示します。

```
SELECT XMLELEMENT ( "Emp:Exempt",
  XMLATTRIBUTES ( e.fname, e.lname AS "name:last", e."name:middle")) AS "result"
   FROM employees e
   WHERE ... ;
```

この問合せによって、次の XML が結果として戻されます。

```
<Emp:Exempt FNAME="John" name:last="Smith" name_x003A_middle="Quincy" /> ...
```

> **注意：** XMLElement() では、これらの名前空間の接頭辞を使用して作成された文書は検証されません。適切な名前空間宣言が指定されているかどうかは、ユーザーが確認してください。部分的なエスケープおよび完全なエスケープの定義は、新しい XML の SQL 標準の一部として指定されています。

例 10-4 XMLElement(): 名前空間を使用した XML Schema に基づく XML 文書の作成

次の例では、名前空間を使用して XML Schema に基づく文書を作成する方法を示します。XML Schema `http://www.oracle.com/Employee.xsd` が存在し、ターゲットの名前空間が存在しない場合、次の問合せによってそのスキーマに準拠した `XMLType` インスタンスが作成されます。

```
SELECT XMLELEMENT ( "Employee",
  XMLATTRIBUTES ( 'http://www.w3.org/2001/XMLSchema' AS "xmlns:xsi",
                  'http://www.oracle.com/Employee.xsd' AS
                         "xsi:nonamespaceSchemaLocation" ),
       XMLForest(empno, ename, sal)) AS "result"
   FROM scott.emp
   WHERE deptno = 100;
```

これによって、XML Schema である `Employee.xsd` に準拠した次の XML 文書が結果として生成されます。

```
--------------
<Employee xmlns:xsi="http://www.w3.org/2001/XMLSchema"
          xsi:nonamespaceSchemaLocation="http://www.oracle.com/Employee.xsd">
   <EMPNO>1769</EMPNO>
   <ENAME>John</ENAME>
   <SAL>200000</SAL>
</Employee>
```

例 10-5 XMLElement(): ユーザー定義型からの要素の生成

DBMS_XMLGEN の項（10-31 ページの「例 10-18DBMS_XMLGEN: 複雑な XML の生成」）に示す例を使用して、次のとおり、従業員および部門を例とした階層 XML を生成できます。

```
SELECT XMLElement("Department",
  dept_t(deptno,dname,
        CAST(MULTISET(
            select empno, ename
            from emp e
            where e.deptno = d.deptno) AS emplist_t)))
     AS deptxml
FROM dept d;
```

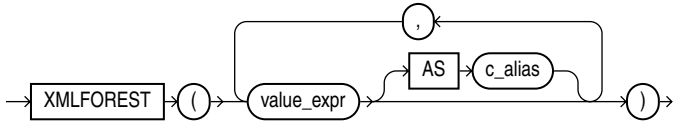
これによって、Department 要素および dept_t 型の正規マッピングを含む XML 文書が生成されます。

```
<Department>
 <DEPT_T DEPTNO="100">
   <DNAME>Sports</DNAME>
  <EMPLIST>
    <EMP_T EMPNO="200">
      <ENAME>John</ENAME>
    <EMP_T>
    <EMP_T>
      <ENAME>Jack</ENAME>
    </EMP_T>
 </EMPLIST>
</DEPT_T>
  </Department>
```

XMLForest() 関数

XMLForest() 関数は、指定された引数のリストから XML 要素のフォレストを生成します。引数には、オプションで別名を指定した値の式を指定することもできます。図 10-4 を参照してください。

図 10-4 XMLForest() の構文



値の式のリストは、XML 形式に変換されます。指定された値の式では AS 句が省略されている場合、完全にエスケープされた形式の列名が要素の囲みタグ名として使用されます。

オブジェクト型またはコレクションの場合、AS 句は必須です。その他の型の場合も、AS 句はオプションで指定できます。AS 句が指定されている場合、部分的にエスケープされた形式の別名が囲みタグ名として使用されます。式が NULL であると評価された場合、その式に対して要素は作成されません。

例 10-6 XMLForest(): 名前属性、雇用日および部門をコンテンツとして持つ、従業員ごとの要素の生成

次の例では、名前属性、雇用日および部門をコンテンツとして持つ Emp 要素を各従業員に 1 つ作成します。

```
SELECT XMLELEMENT("Emp", XMLATTRIBUTES ( e.fname ||' '|| e.lname AS "name" ),
XMLForest ( e.hire, e.dept AS "department")) AS "result"
FROM employees e;
```

この問合せによって、次の XML が結果として戻されます。

```
<Emp name="John Smith">
  <HIRE>2000-05-24</HIRE>
  <department>Accounting</department>
</Emp>
<Emp name="Mary Martin">
  <HIRE>1996-02-01</HIRE>
  <department>Shipping</department>
</Emp>
```

例 10-7 XMLForest(): ユーザー定義型からの要素の生成

XMLForest() を使用してユーザー定義型から XML を生成することもできます。DBMS_XMLGEN の項（10-31 ページの「例 10-18DBMS_XMLGEN: 複雑な XML の生成」）に示す例を使用して、次のとおり、従業員および部門を例とした階層 XML を生成できます。

```
SELECT XMLForest(
  dept_t(deptno,dname,
        CAST(MULTISET(
            select empno, ename
            from emp e
            where e.deptno = d.deptno) AS emplist_t)) AS "Department")
    AS deptxml
FROM dept d;
```

これによって、Department 要素および dept_t 型の正規マッピングを含む XML 文書が生成されます。

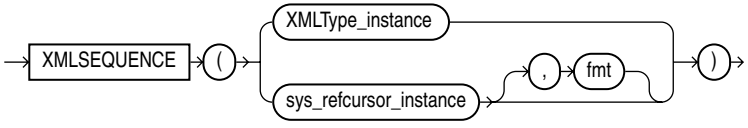
注意： XMLElement() の場合とは異なり、dept_t 要素は省略されます。

```
<Department DEPTNO="100">
  <DNAME>Sports</DNAME>
    <EMPLIST>
      <EMP_T EMPNO="200">
        <ENAME>John</ENAME>
      </EMP_T>
      <EMP_T>
        <ENAME>Jack</ENAME>
      </EMP_T>
   </EMPLIST>
</Department>
```

XMLSEQUENCE() 関数

XMLSequence() 関数は、XMLType の順序を戻します。この関数は、XMLType インスタンスの VARRAY である XMLSequenceType を戻します。この関数はコレクションを戻すため、SQL 問合せの FROM 句で使用できます。図 10-5 を参照してください。

図 10-5 XMLSequence() の構文



XMLSequence() 関数には、次の 2 つの形式があります。

- 1 つ目の形式では、XMLType インスタンスを入力し、最上位のノードの VARRAY を戻します。この形式は、XML フラグメントを複数行に断片化するために使用できます。
- 2 つ目の形式では、入力として REFCURSOR 引数、インスタンスとして XMLFormat オブジェクト（オプション）を取り、カーソルの各行に対応する XMLType の VARRAY を戻します。この形式は、任意の SQL 問合せから XMLType インスタンスを作成するために使用できます。今回のリリースでは、この XMLFormat の使用方法では XML Schema をサポートしていません。

XMLSequence() は、XMLType を含む SQL 問合せの効率的な実行には不可欠です。

例 10-8 XMLSequence(): ある XML 文書からの別の XML 文書の生成

従業員情報を含む次の XML 文書があるとします。

```
<EMPLOYEES>
 <EMP>
    <EMPNO>112</EMPNO>
    <EMPNAME>Joe</EMPNAME>
    <SALARY>50000</SALARY>
  </EMP>
 <EMP>
    <EMPNO>217</EMPNO>
    <EMPNAME>Jane</EMPNAME>
    <SALARY>60000</SALARY>
 </EMP>
 <EMP>
    <EMPNO>412</EMPNO>7
    <EMPNAME>Jack</EMPNAME>
    <SALARY>40000</SALARY>
 </EMP>
</EMPLOYEES>
```

年収が $50,000 以上の従業員のみを含む新しい XML 文書を作成するには、次の構文を使用します。

```
SELECT SYS_XMLAGG(value(e), xmlformat('EMPLOYEES'))
   FROM TABLE(XMLSequence(Extract(doc, '/EMPLOYEES/EMP'))) e
   WHERE EXTRACTVALUE(value(e), '/EMP/SALARY') >= 50000;
```

これによって、次の XML 文書が戻されます。

```
<EMPLOYEES>
  <EMP>
     <EMPNO>112</EMPNO>
     <EMPNAME>Joe</EMPNAME>
     <SALARY>50000</SALARY>
  </EMP>
  <EMP>
     <EMPNO>217</EMPNO>
     <EMPNAME>Jane</EMPNAME>
     <SALARY>60000</SALARY>
  </EMP>
 </EMPLOYEES>
```

すべての従業員を抽出するための `Extract()` の使用方法に注意してください。

1. `Extract()` によって、EMP 要素のフラグメントが戻されます。
2. `XMLSequence()` によって、これらの最上位要素のコレクションが `XMLType` インスタンスに作成され、そのインスタンスが戻されます。

3. TABLE 関数を使用して、そのコレクションが、問合せの FROM 句で使用できる表の値に変換されます。

例 10-9 XMLSequence(): SYS_REFCURSOR 引数を使用した、カーソル式の各行に対する XML 文書の生成

次の例では、XMLSequence() によってカーソル式の各行に対する XML 文書を作成し、値が XMLSequenceType として戻されます。XMLFormat オブジェクトを使用して、結果として戻される XML 文書の構造を変更することもできます。たとえば、次のようなコールを発行するとします。

```
SELECT value(e).getClobVal()
FROM TABLE(XMLSequence(Cursor(SELECT * FROM emp))) e;
```

次の XML が戻されます。

```
XMLType
--------------------------------
  <ROW>
    <EMPNO>300</EMPNO>
    <ENAME>John</ENAME>
  </ROW>

  <ROW>
    <EMPNO>413</EMPNO>
    <ENAME>Jane</ENAME>
  </ROW>

  <ROW>
    <EMPNO>968</EMPNO>
    <ENAME>Jack</ENAME>
  </ROW>
...
```

各行に使用されている行タグは、XMLFormat オブジェクトを使用して変更できます。

例 10-10 XMLSequence(): XML 文書内のコレクションの SQL 行へのネスト解除

TABLE 関数である XMLSequence() を使用して、XML 文書内の要素をネスト解除できます。次の XML 文書があるとします。

```
<Department deptno="100">
   <DeptName>Sports</DeptName>
  <EmployeeList>
    <Employee empno="200">
      <Ename>John</Ename>
      <Salary>33333</Salary>
    </Employee>
    <Employee empno="300">
      <Ename>Jack</Ename>
      <Salary>333444</Salary>
    </Employee>
   </EmployeeList>
</Department>

<Department deptno="200">
  <DeptName>Garment</DeptName>
  <EmployeeList>
    <Employee empno="400">
      <Ename>Marlin</Ename>
      <Salary>20000</Salary>
    </Employee>
  </EmployeeList>
</Department>
```

この XML 文書が XMLType の dept_xml_tab 表に格納されている場合、XMLSequence() 関数を使用して Employee リスト項目を最上位の SQL 行にネスト解除できます。

```
CREATE TABLE dept_xml_tab OF XMLTYPE;

INSERT INTO dept_xml_tab VALUES(
  xmltype('<Department deptno="100">
    <DeptName>Sports</DeptName><EmployeeList>
    <Employee empno="200"><Ename>John</Ename><Salary>33333</Salary></Employee>
    <Employee empno="300"><Ename>Jack</Ename><Salary>333444</Salary></Employee>
    </EmployeeList></Department>'));

INSERT INTO dept_xml_tab VALUES (
  xmltype('<Department deptno="200">
    <DeptName>Sports</DeptName><EmployeeList>
    <Employee empno="400"><Ename>Marlin</Ename><Salary>20000</Salary></Employee>
    </EmployeeList></Department>'));

SELECT extractvalue(value(d),'/Department/@deptno') as deptno,
```

```
       extractvalue(value(e),'/Employee/@empno') as empno,
       extractvalue(value(e),'/Employee/Ename') as ename
   FROM  dept_xml_tab d,
    TABLE(XMLSequence(extract(value(d),'/Department/EmployeeList/Employee'))) e;
```

これによって、次の結果が戻されます。

```
DEPTNO       EMPNO          ENAME
---------------------------------
100          200            John
100          300            Jack
200          400            Marlin

3 rows selected
```

dept_xml_tab 表内の各行に対して、TABLE 関数が評価されます。ここで、extract() 関数によって、すべての従業員要素のフラグメントを含む新しい XMLType インスタンスが作成されます。このインスタンスは、すべての従業員のコレクションを作成する XMLSequence() に送られます。

次に、TABLE 関数によって、そのコレクション要素が、親表 dept_xml_tab に関連付けられている、複数の行に分解されます。そのため、dept_xml_tab の親であるすべての行のリストおよび関連付けられた従業員が戻されます。

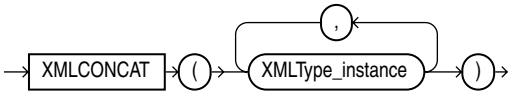
extractValue() 関数によって、部門番号、従業員番号および名前のスカラー値が抽出されます。

XMLConcat() 関数

XMLConcat() 関数は、渡されたすべての引数を連結して XML フラグメントを作成します。図 10-6 を参照してください。XMLConcat() 関数には、次の 2 つの形式があります。

- 1 つ目の形式では、XMLType の VARRAY である XMLSequenceType を取り、VARRAY のすべての要素を連結して単一の XMLType インスタンスを戻します。この形式は、XMLType のリストを単一のインスタンスに縮小する場合に有効です。
- 2 つ目の形式では、任意の数の XMLType 値を取り、それらの値を連結します。値の 1 つが NULL である場合、その値は結果に出力されません。すべての値が NULL である場合、結果は NULL になります。この形式は、任意の数の XMLType インスタンスを同じ行に連結するために使用できます。XMLAgg() を使用すると、複数行にわたって XMLType インスタンスを連結できます。

図 10-6 XMLConcat() の構文



例 10-11 XMLConcat(): 引数順序で使用される XML 要素の連結の取得

次の例では、XMLConcat() によって XMLSequenceType から XMLType の連結が戻されます。

```
SELECT XMLConcat(XMLSequenceType(
          xmltype('<PartNo>1236</PartNo>'),
          xmltype('<PartName>Widget</PartName>'),
          xmltype('<PartPrice>29.99</PartPrice>'))).getClobVal()
    FROM dual;
```

この問合せによって、次の形式の単一のフラグメントが戻されます。

```
<PartNo>1236</PartNo>
<PartName>Widget</PartName>
<PartPrice>29.99</PartPrice>
```

例 10-12 XMLConcat(): 引数要素の連結による XML 要素の取得

次の例では、名前および姓の XML 要素を作成し、結果を連結します。

```
SELECT XMLConcat ( XMLElement ("first", e.fname), XMLElement ("last", e.lname))  AS
"result"
FROM employees e ;
```

この問合せによって、次の XML 文書が戻されます。

```
<first>Mary</first>
<last>Martin</last>

<first>John</first>
<last>Smith</last>
```

XMLAgg() 関数

XMLAgg() 関数は、XML 要素のコレクションから XML 要素のフォレストを生成する集計関数です。図 10-7 に、XMLAgg() 構文を示します。この構文で、order_by_clause は次のとおりです。

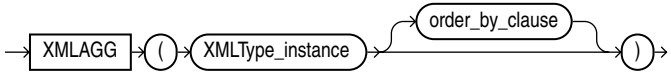
```
ORDER BY [list of: expr [ASC|DESC]  [NULLS {FIRST|LAST} ] ]
```

また、数値リテラルは、列の位置として解釈されません。たとえば、ORDER BY 1 は、最初の列での順序付けを意味しません。数値リテラルは、他のリテラルと同様に解釈されます。

XMLConcat() と同様に、NULL である引数は結果から削除されます。XMLAgg() 関数は SYS_XMLAGG() 関数に類似していますが、ノードのフォレストを戻し、XMLFormat() パラメータを取らない点で異なります。この関数を使用すると、複数行にわたって XMLType インスタンスを連結できます。また、オプションの ORDER BY 句を使用して、集計する XML 値を順序付けできます。

XMLAgg() は集計関数であるため、各グループの結果を集計し、1 つの XML を生成します。問合せで GROUP BY が指定されていない場合、問合せのすべての行に対する結果が集計され、1 つの XML が戻されます。NULL 値は結果から削除されます。

図 10-7 XMLAgg() の構文



例 10-13 XMLAgg(): 従業員要素のリストを含む部門要素の生成

次の例では、従業員のジョブ ID および姓を要素のコンテンツとして持つ Employee 要素を含む Department 要素を作成します。また、その部門に属する従業員の XML 要素を従業員の姓で順序付けします。

```
SELECT XMLELEMENT("Department",
           XMLAGG(
                XMLELEMENT("Employee", e.job_id||' '||e.last_name)
                  ORDER BY last_name))
          as "Dept_list"
FROM employees e
WHERE e.department_id = 30;

Dept_list
------------------------------------------------------------
<Department>
  <Employee>PU_CLERK Baida</Employee>
  <Employee>PU_CLERK Colmenares</Employee>
  <Employee>PU_CLERK Himuro</Employee>
```

```
  <Employee>PU_CLERK Khoo</Employee>
  <Employee>PU_MAN Raphaely</Employee>
  <Employee>PU_CLERK Tobias</Employee>
</Department>
```

XMLAgg() によって行が集計されるため、単一の行が戻されます。GROUP BY 句を使用すると、戻された一連の行を複数のグループにグループ化できます。

```
SELECT XMLELEMENT("Department", XMLAttributes(department_id AS deptno),
      XMLAGG(XMLELEMENT("Employee", e.job_id||' '||e.last_name)))
   AS "Dept_list"
   FROM employees e
   GROUP BY e.department_id;

Dept_list
------------------------------------------------------------
<Department deptno="1001">
  <Employee>AD_ASST Whalen</Employee>
</Department>

<Department deptno="2002">
  <Employee>MK_MAN Hartstein</Employee>
  <Employee>MK_REP Fay</Employee>
</Department>

<Department deptno="3003">
  <Employee>PU_MAN Raphaely</Employee>
  <Employee>PU_CLERK Khoo</Employee>
  <Employee>PU_CLERK Tobias</Employee>
  <Employee>PU_CLERK Baida</Employee>
  <Employee>PU_CLERK Colmenares</Employee>
  <Employee>PU_CLERK Himuro</Employee>
</Department>
```

XMLAgg() 式で ORDER BY 句を使用すると、各部門内で従業員を順序付けできます。

注意： この *order_by_clause* にかぎり、他の使用方法とは異なり、数値リテラルは列の位置として解釈されず、単に数値リテラルとして解釈されます。

例 10-14 XMLAgg(): 部門要素、部門ごとの従業員要素および従業員の扶養家族の生成

XMLAgg() を使用すると、表に存在するいくつかの階層関係を反映できます。次の例では、部門ごとに部門要素を生成します。この部門要素内に、その部門のすべての従業員の要素を作成します。また、各従業員内に、従業員の扶養家族をリストします。

```
SELECT XMLELEMENT( "Department", XMLATTRIBUTES ( d.dname AS "name" ),
        (SELECT XMLAGG(XMLELEMENT ("emp", XMLATTRIBUTES (e.ename AS name),
                  ( SELECT XMLAGG(XMLELEMENT( "dependent",
                                XMLATTRIBUTES(de.name AS "name")))
                     FROM dependents de
                     WHERE de.empno = e.empno )  ))
            FROM emp e
            WHERE e.deptno = d.deptno) ) AS "dept_list"
   FROM dept d ;
```

この問合せによって、XMLType インスタンスを含む行が部門ごとに生成されます。

```
<Department name="Accounting">
    <emp name="Smith">
      <dependent name="Sara Smith"/d>
      <dependent name="Joyce Smith"/>
    </emp>
    <emp name="Yates"/>
</Department>

<Department name="Shipping">
    <emp name="Martin">
      <dependent name="Alan Martin"/>
    </emp>
    <emp name="Oppenheimer">
      <dependent name="Ellen Oppenheimer"/>
    </emp>
</Department>
```

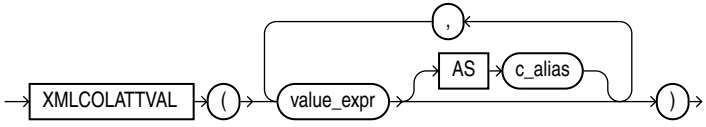
SQLX 関数を使用したデータベースからの XML の生成

XMLColAttVal() は、Oracle が提供する SQLX の拡張関数です。

XMLColAttVal() 関数

XMLColAttVal() 関数は、渡された引数の値を含む XML の column 要素のフォレストを生成します。図 10-8 を参照してください。

図 10-8 XMLColAttVal() の構文



引数の名前は、column 要素の name 属性に挿入されます。XMLForest() 関数とは異なり、要素の名前はエスケープされません。そのため、この関数を使用すると、名前をエスケープすることなく SQL の列および値を転送できます。

例 10-15 XMLColAttVal(): 名前属性および雇用日と部門の要素をコンテンツとして持つ、各従業員の Emp 要素の生成

次の例では、名前属性、雇用日および部門をコンテンツとして持つ Emp 要素を各従業員に 1 つ作成します。

```
SELECT XMLELEMENT("Emp",XMLATTRIBUTES(e.fname ||' '||e.lname AS "name" ),
                 XMLCOLATTVAL ( e.hire, e.dept AS "department")) AS "result"
 FROM employees e;
```

この問合せによって、次の XML が結果として戻されます。

```
<Emp name="John Smith">
   <column name="HIRE">2000-05-24</column>
   <column name="department">Accounting</column>
</Emp>
<Emp name="Mary Martin">
  <column name="HIRE">1996-02-01</column>
   <column name="department">Shipping</column>
</Emp>
<Emp name="Samantha Stevens">
   <column name="HIRE">1992-11-15</column>
   <column name="department">Standards</column>
</Emp>
```

各 XMLColAttVal() 引数に関連付けられた名前を使用して属性の値を移入するため、完全にエスケープされたマッピングと部分的にエスケープされたマッピングのいずれも使用されません。

DBMS_XMLGEN を使用した Oracle9*i* データベースからの XML の生成

DBMS_XMLGEN は、データベース問合せの結果を XML にマップすることによって、SQL 問合せから XML 文書を作成します。DBMS_XMLGEN は、XML 文書を CLOB または XMLType として取得します。DBMS_XMLGEN にはフェッチ・インタフェースがあり、それによって行の最大数とスキップする行を指定できます。これは、Web アプリケーションでのページ区切り要件を満たすために有効です。DBMS_XMLGEN には、ROW、ROWSET などのタグ名を変更するオプションもあります。

DBMS_XMLGEN パッケージのパラメータは、取り出す行の数を制限し、タグ名を囲みます。PL/SQL パッケージ DBMS_XMLGEN の機能の概要は次のとおりです。

- SQL 問合せから XML 文書インスタンスを作成し、その文書を CLOB または XMLType として取得します。
- 行の最大数とスキップする行を指定するフェッチ・インタフェースを使用すると、たとえば、最初のフェッチで最大 10 行を取り出し、最初の 4 行をスキップできます。これは、Web ベース・アプリケーションでのページ区切り要件を満たすために有効です。
- ROW、ROWSET などのタグ名を変更するためのオプションを提供します。

> **参照：** OracleXMLQuery と DBMS_XMLGEN の機能の比較については、『Oracle9*i* XML Developer's Kit ガイド - XDK』の第 8 章「XML SQL Utility（XSU）」の「XSU の OracleXMLQuery を使用した XML の生成」を参照してください。

DBMS_XMLGEN の問合せ結果の例

次に、データベースで「select * from scott.emp」問合せを実行した結果の例を示します。

```
<?xml version="1.0"?>
<ROWSET>
<ROW>
  <EMPNO>30</EMPNO>
  <ENAME>Scott</ENAME>
  <SALARY>20000</SALARY>
</ROW>
<ROW>
  <EMPNO>30</EMPNO>
  <ENAME>Mary</ENAME>
  <AGE>40</AGE>
</ROW>
</ROWSET>
```

DBMS_XMLGEN パッケージを使用した getXML() の結果は、CLOB になります。デフォルト・マッピングは次のとおりです。

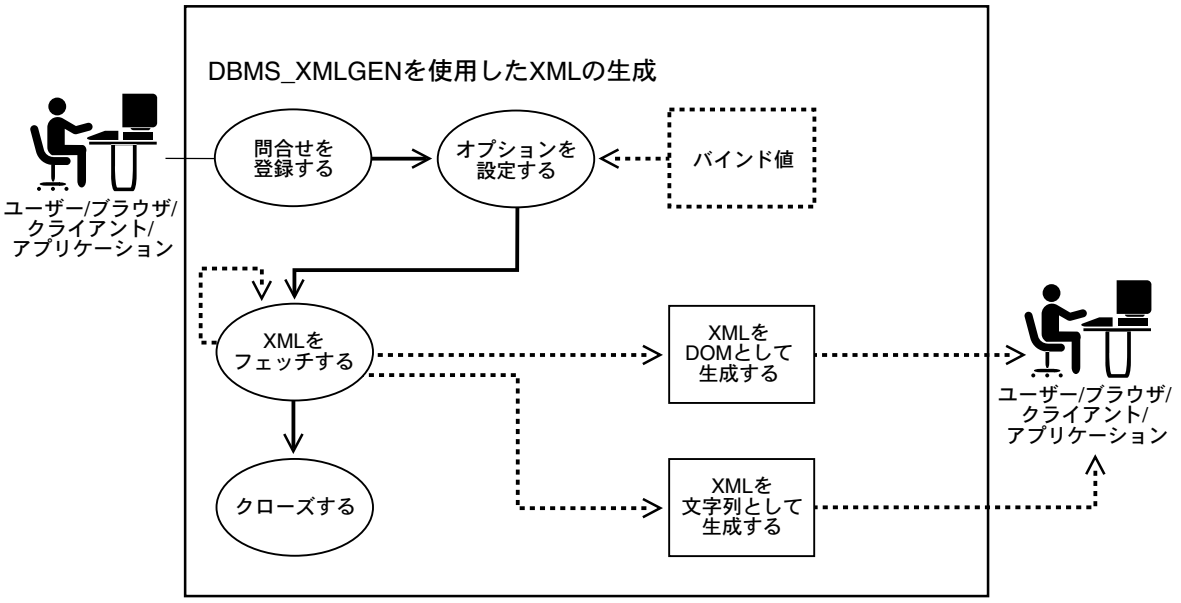
- 問合せ結果のすべての行は、デフォルトのタグ名 ROW を持つ XML 要素にマップされます。
- 結果全体は、ROWSET 要素で囲まれます。これらの名前は両方とも、DBMS_XMLGEN の setRowTagName() プロシージャおよび setRowSetTagName() プロシージャを使用して変更できます。
- SQL 問合せ結果の各列は、ROW 要素のサブ要素としてマップされます。
- バイナリ・データは、16 進数表現に変換されます。

文書が CLOB に存在する場合、データベース・キャラクタ・セットと同じエンコーディングになります。データベース・キャラクタ・セットが SHIFTJIS の場合、XML 文書も SHIFTJIS になります。

DBMS_XMLGEN のコール順序

図 10-9 に、DBMS_XMLGEN のコール順序の概要を示します。

図 10-9 DBMS_XMLGEN のコール順序



DBMS_XMLGEN のコール順序は次のとおりです。

1. SQL 問合せを実行し、newContext() をコールすることによって、パッケージからコンテキストを取得します。
2. そのコンテキストをパッケージ内のすべてのプロシージャおよびファンクションに渡して、様々なオプションを設定します。たとえば、ROW 要素の名前を設定するには、setRowTag(ctx) を使用します。この ctx は、前の newContext() のコールから取得したコンテキストです。
3. getXML() または getXMLType() を使用して XML 結果を取得します。setMaxRows() をコールしてフェッチごとに取り出す行の最大数を設定すると、このファンクションを繰り返しコールして、コールごとに最大数の行を取得できます。問合せに行が残っていない場合、このファンクションは NULL を戻します。

   getXML() および getXMLType() は、取り出す行がない場合でも、常に XML 文書を戻します。取り出す行があるかどうかを確認するには、getNumRowsProcessed() ファンクションを使用します。
4. 問合せをリセットして再度開始し、手順 3 を繰り返すことができます。
5. closeContext() をクローズし、内部に割り当てられているすべてのリソースを解放します。

表 10-1 に、DBMS_XMLGEN のファンクションおよびプロシージャの概要を示します。

表 10-1 DBMS_XMLGEN のファンクションおよびプロシージャ

ファンクションまたはプロシージャ	説明
DBMS_XMLGEN 型定義 SUBTYPE ctxHandle IS NUMBER	すべてのファンクションで使用されるコンテキスト・ハンドルです。 DTD または Schema の指定は次のとおりです。 NONE CONSTANT NUMBER:= 0; 今回のリリースでサポートされています。 DTD CONSTANT NUMBER:= 1; SCHEMA CONSTANT NUMBER:= 2; これらの指定を getXML ファンクションで使用して、DTD を生成するか、XML Schema を生成するか、または何も生成しないか（NONE）を指定できます。今回のリリースの getXML ファンクションでは、NONE 指定のみがサポートされています。
ファンクション・プロトタイプ **newContext()**	任意の問合せ文字列に対して、後続のファンクションで使用される新しいコンテキスト・ハンドルを生成します。
ファンクション newContext(queryString IN VARCHAR2)	新しいコンテキストを戻します。 パラメータ : queryString (IN) - XML に変換する必要がある問合せ文字列の結果。 戻り値 : コンテキスト・ハンドル。最初にこのファンクションをコールして、結果から XML を戻すために getXML() およびその他のファンクションで使用できるハンドルを取得します。
ファンクション newContext(queryString IN SYS_REFCURSOR) RETURN ctxHandle;	渡された PL/SQL REF カーソルから新しいコンテキスト・ハンドルを作成します。コンテキスト・ハンドルは、他のファンクションにも使用できます。次の例を参照してください。
setRowTag()	すべての行を区切る要素の名前を設定します。デフォルト名は ROW です。
プロシージャ setRowTag(ctx IN ctxHandle,rowTag IN VARCHAR2);	パラメータ : ctx (IN) - newContext をコールして取得されるコンテキスト・ハンドル。 rowTag (IN) - ROW 要素の名前。ROW 要素が必要ない場合は、NULL を指定します。このファンクションをコールして、ROW 要素の名前を設定すると、デフォルト名の ROW は表示されません。この名前を NULL に設定して、ROW 要素自体を出力しないようにすることもできます。ROW および ROWSET の両方が NULL で、出力に複数の列または行がある場合は、エラーになります。
setRowSetTag()	文書のルート要素の名前を設定します。デフォルト名は ROWSET です。

表 10-1 DBMS_XMLGEN のファンクションおよびプロシージャ（続き）

ファンクションまたはプロシージャ	説明
プロシージャ setRowSetTag(ctx IN ctxHandle, rowSetTag IN VARCHAR2);	パラメータ : ctx (IN) - newContext をコールして取得されるコンテキスト・ハンドル。 rowsetTag (IN) - 文書要素の名前。ROW 要素が必要ない場合、NULL を指定します。このファンクションをコールして、文書のルート要素の名前を設定すると、出力にデフォルト名の ROWSET が表示されません。この名前を NULL に設定して、この要素自体を出力しないようにすることもできます。ROW および ROWSET の両方が NULL で、出力に複数の列または行がある場合、エラーになります。
getXML()	指定されている最大数の行をフェッチすることによって XML 文書を取得します。また、その XML 文書を渡された CLOB に追加します。
プロシージャ getXML(ctx IN ctxHandle, clobval IN OUT NCOPY clob, dtdOrSchema IN number:= NONE);	パラメータ : ctx (IN) - newContext() をコールして取得されるコンテキスト・ハンドル。 clobval (IN/OUT) - XML 文書を追加する CLOB。 dtdOrSchema (IN) - DTD または Schema のどちらを生成するかどうか。このパラメータはサポートされていません。 このバージョンの getXML ファンクションを使用して、余分な CLOB コピーの生成を回避したり、後続のコールで同じ CLOB を再利用します。この getXML のコールは、LOB ロケータの作成を必要としますが、次のファンクションより効率的です。XML を生成するときは、setSkipRows コールに指定された数の行がスキップされ、setMaxRows コールに指定された最大数の行（または指定がない場合は結果全体）がフェッチされ、XML に変換されます。getNumRowsProcessed ファンクションを使用して、行が取り出されたかどうかを確認します。
getXML()	XML 文書を生成し、それを CLOB として戻します。
ファンクション getXML(ctx IN ctxHandle, dtdOrSchema IN number:= NONE) RETURN clob	パラメータ : ctx (IN) - newContext() をコールして取得されるコンテキスト・ハンドル。 dtdOrSchema (IN) - DTD または Schema のどちらを生成するかどうか。このパラメータはサポートされていません。 戻り値 : 文書を含む一時 CLOB。dbms_lob.freetemporary をコールして、このファンクションから取得した一時 CLOB を解放します。

表 10-1 DBMS_XMLGEN のファンクションおよびプロシージャ（続き）

ファンクションまたはプロシージャ	説明
ファンクション getXMLType(ctx IN ctxHandle, dtdOrSchema IN number:= NONE) RETURN XMLTYPE	パラメータ : ctx (IN) - newContext() をコールして取得されるコンテキスト・ハンドル。 dtdOrSchema (IN) - DTD または Schema のどちらを生成するかどうか。このパラメータはサポートされていません。 戻り値 : 文書を含む XMLType インスタンス。
ファンクション getXML(sqlQuery IN VARCHAR2, dtdOrSchema IN NUMBER := NONE) RETURN CLOB;	渡された SQL 問合せ文字列からの問合せ結果を XML 形式に変換し、XML を CLOB として戻します。
ファンクション getXMLType(sqlQuery IN VARCHAR2, dtdOrSchema IN NUMBER := NONE) RETURN XMLTYPE;	渡された SQL 問合せ文字列からの問合せ結果を XML 形式に変換し、XML を CLOB として戻します。
getNumRowsProcessed()	getXML をコールして XML を生成するときに処理される、SQL の行数を取得します。この行数には、XML の生成前にスキップされる行数は含まれません。
ファンクション getNumRowsProcessed(ctx IN ctxHandle) RETURN number	パラメータ : queryString (IN) - XML に変換する必要がある結果の問合せ文字列。 戻り値 : getXML を最後にコールしたときに処理された SQL の行数。このファンクションをコールして、結果セットの最後に達したかどうかを確認できます。この行数にはスキップされた行数は含まれません。ループで getXML をコールする場合、このファンクションを使用して終了条件を決定します。getXML は、行がない場合でも常に XML 文書を生成することに注意してください。
setMaxRows()	getXML のコールごとに、SQL 問合せ結果からフェッチする行の最大数を設定します。
プロシージャ setMaxRows(ctx IN ctxHandle, maxRows IN NUMBER);	パラメータ : ctx (IN) - 実行されている問合せに対応するコンテキスト・ハンドル。 maxRows (IN) - getXML をコールするたびに取得する最大行数。 maxRows パラメータは、このユーティリティを使用して結果のページ区切りを生成するときに使用できます。たとえば、XML または HTML データのページを生成する場合、XML に変換する行数を制限し、後続のコールにおいて、次の一連の行を次々に取得することができます。これによって応答時間が短縮できます。
setSkipRows()	getXML ルーチンをコールするたびに、XML 出力を生成する前に任意の数の行をスキップします。

表 10-1 DBMS_XMLGEN のファンクションおよびプロシージャ（続き）

ファンクションまたはプロシージャ	説明
プロシージャ setSkipRows(ctx IN ctxHandle, skipRows IN NUMBER);	パラメータ : ctx (IN) - 実行されている問合せに対応するコンテキスト・ハンドル。 skipRows (IN) - getXML をコールするたびにスキップする行数。 skipRows パラメータは、このユーティリティを使用して、ステートレスな Web ページの結果の区切りを生成するときに使用できます。たとえば、XML または HTML データの最初のページを生成するときに、skipRows を 0（ゼロ）に設定できます。次の設定では、skipRows を最初に指定した行数に設定できます。
setConvertSpecialChars()	XML データの特殊文字を、XML の等価のエスケープ文字に変換する必要があるかどうかを設定します。たとえば、< 符号は < に変換されます。デフォルトでは、変換が実行されます。
プロシージャ setConvertSpecialChars(ctx IN ctxHandle, conv IN boolean);	パラメータ : ctx (IN) - 使用するコンテキスト・ハンドル。 conv (IN) - 変換が必要かどうか。 入力データに、エスケープする必要がある特殊文字（<、>、"、' など）が含まれていないことがわかっている場合、このファンクションを使用して XML 処理を高速化できます。実際に文字データをスキャンして特殊文字を置換するには、特に大量のデータを処理する場合、大きいコストがかかります。したがって、データが XML で問題ない場合、このファンクションをコールしてパフォーマンスを向上できます。
setNullHandling	
プロシージャ setNullHandling(ctx IN ctxHandle, flag IN NUMBER);	setNullHandling のフラグの値は次のとおりです。 ■ DROP_NULLS CONSTANT NUMBER := 0; これはデフォルトの設定です。 ■ NULL_ATTR CONSTANT NUMBER := 1; ■ EMPTY_TAG CONSTANT NUMBER := 2; ■ DROP_NULLS は NULL 要素のタグを削除します。 ■ NULL_ATTR は xsi:nil="true" を設定します。 ■ EMPTY_TAG は <foo/> などを設定します。
useNullAttributeIndicator	これは setNullHandling(ctx, NULL_ATTR) のショートカットです。
プロシージャ useNullAttributeIndicator(ctx IN ctxHandle, attrind IN boolean := TRUE);	
useItemTagsForColl()	コレクション要素名を設定します。コレクション要素のデフォルト名は、型名自体です。このファンクションを使用して、列名を、列名に _ITEM タグを追加した名前にオーバーライドできます。

表 10-1 DBMS_XMLGEN のファンクションおよびプロシージャ（続き）

ファンクションまたはプロシージャ	説明
プロシージャ useItemTagsForColl(ctx IN ctxHandle);	パラメータ : ctx (IN) - コンテキスト・ハンドル。 たとえば、NUMBER というコレクションがある場合、コレクション要素のデフォルトのタグ名は NUMBER になります。このプロシージャをコールすると、このデフォルト動作を変更して、デフォルト名に _ITEM タグを追加したコレクション列名を生成できます。
restartQuery()	問合せを再度開始し、最初の行から XML を生成します。
プロシージャ restartQuery(ctx IN ctxHandle);	パラメータ : ctx (IN) - 現行の問合せに対応するコンテキスト・ハンドル。このプロシージャをコールすると、新しいコンテキストを作成することなく、問合せの実行を再度開始できます。
closeContext()	任意のコンテキストをクローズし、SQL カーソル、バインドや定義バッファなど、そのコンテキストに対応付けられているすべてのリソースを解放します。
プロシージャ closeContext(ctx IN ctxHandle);	パラメータ : ctx (IN) - クローズするコンテキスト・ハンドル。このハンドルに対応付けられているすべてのリソースをクローズします。クローズした後は、他のすべての DBMS_XMLGEN ファンクションをコールするときに、このハンドルを使用できなくなります。
変換ファンクション ファンクション convert(xmlData IN varchar2, flag IN NUMBER := ENTITY_ENCODE) return varchar2;	渡された XML データ文字列をエンコードまたはデコードします。 ■ エンコーディングとは、< などの実体参照を < などのエスケープされた同等のものに置き換えることです。 ■ デコーディングとは、逆方向の変換です。
ファンクション convert(xmlData IN CLOB, flag IN NUMBER := ENTITY_ENCODE) return CLOB;	渡された XML の CLOB データをエンコードまたはデコードします。 ■ エンコーディングとは、< などの実体参照を < などのエスケープされた同等のものに置き換えることです。 ■ デコーディングとは、逆方向の変換です。

例 10-16 DBMS_XMLGEN: 単純な XML の生成

この例では、オブジェクト・リレーショナル表から従業員データを選択して XML 文書を作成し、結果の CLOB を表に挿入します。

```
CREATE TABLE temp_clob_tab(result CLOB);

DECLARE
   qryCtx DBMS_XMLGEN.ctxHandle;
   result CLOB;
BEGIN
  qryCtx := dbms_xmlgen.newContext('SELECT * from scott.emp');
```

```
  -- set the row header to be EMPLOYEE
  DBMS_XMLGEN.setRowTag(qryCtx, 'EMPLOYEE');

  -- now get the result
  result := DBMS_XMLGEN.getXML(qryCtx);

  INSERT INTO temp_clob_tab VALUES(result);

  --close context
  DBMS_XMLGEN.closeContext(qryCtx);
END;
/
```

この問合せによって、次の XML が戻されます。

```
SELECT * FROM temp_clob_tab;

RESULT
-----------------------------------
<?xml version=''1.0''?>
<ROWSET>
 <EMPLOYEE>
  <EMPNO>7369</EMPNO>
  <ENAME>SMITH</ENAME>
  <JOB>CLERK</JOB>
  <MGR>7902</MGR>
  <HIREDATE>17-DEC-80</HIREDATE>
  <SAL>800</SAL>
  <DEPTNO>20</DEPTNO>
 </EMPLOYEE>
 <EMPLOYEE>
  <EMPNO>7499</EMPNO>
  <ENAME>ALLEN</ENAME>
  <JOB>SALESMAN</JOB>
  <MGR>7698</MGR>
  <HIREDATE>20-FEB-81</HIREDATE>
  <SAL>1600</SAL>
  <COMM>300</COMM>
  <DEPTNO>30</DEPTNO>
 </EMPLOYEE>
...
</ROWSET>
```

例 10-17 DBMS_XMLGEN: ページ区切りを使用した単純な XML の生成

すべての行に対して XML 全体を生成するかわりに、DBMS_XMLGEN が提供するフェッチ・インタフェースを使用して、1 回に固定数の行を取り出すことができます。これによって、特に行数が多い場合に、応答時間が短縮され、結果の XML に DOM API を使用する必要があるアプリケーションをスケーリングするために有効です。

次の例は、DBMS_XMLGEN を使用して scott.emp 表から結果を取り出す方法を示します。

```
-- create a table to hold the results..!
CREATE TABLE temp_clob_tab ( result clob);

declare
   qryCtx dbms_xmlgen.ctxHandle;
   result CLOB;
begin

  -- get the query context;
  qryCtx := dbms_xmlgen.newContext('select * from scott.emp');

  -- set the maximum number of rows to be 5,
  dbms_xmlgen.setMaxRows(qryCtx, 5);

  loop
    -- now get the result
    result := dbms_xmlgen.getXML(qryCtx);

    -- if there were no rows processed, then quit..!
    exit when dbms_xmlgen.getNumRowsProcessed(qryCtx) = 0;

    -- do some processing with the lob data..!
    -- Here, we are inserting the results
    -- into a table. You can print the lob out, output it to a stream,
    -- put it in a queure
    -- or do any other processing.
    insert into temp_clob_tab values(result);

  end loop;
  --close context
  dbms_xmlgen.closeContext(qryCtx);
end;
/
```

この場合、5 行ごとに XML 文書が生成されます。

例 10-18 DBMS_XMLGEN: 複雑な XML の生成

オブジェクト型を使用すると、ネストした構造を表す複雑な XML を生成できます。

```
CREATE TABLE new_departments (
    department_id   NUMBER PRIMARY KEY,
    department_name VARCHAR2(20)
  );

CREATE TABLE new_employees (
    employee_id    NUMBER PRIMARY KEY,
    last_name    VARCHAR2(20),
    department_id  NUMBER REFERENCES new_departments
  );

CREATE TYPE emp_t AS OBJECT(
    "@employee_id" NUMBER,
     last_name VARCHAR2(20)
  );
/

CREATE TYPE emplist_t AS TABLE OF emp_t;
/

CREATE TYPE dept_t AS OBJECT(
    "@department_id" NUMBER,
     department_name VARCHAR2(20),
     emplist emplist_t
  );
/

qryCtx := dbms_xmlgen.newContext
     ('SELECT dept_t(department_id, department_name,
              CAST(MULTISET
                   (SELECT e.employee_id, e.last_name
                    FROM new_employees e
                    WHERE e.department_id = d.department_id)
                           AS   emplist_t))  AS deptxml
         FROM new_departments d');
DBMS_XMLGEN.setRowTag(qryCtx, NULL);

-- Here is the resulting XML:
-- <ROWSET>
--    <DEPTXML DEPARTMENT_ID="10">
--       <DEPARTMENT_NAME>SALES</DEPARTMENT_NAME>
--           <EMPLIST>
--             <EMP_T EMPLOYEE_ID="30">
--               <LAST_NAME>Scott</LAST_NAME>
```

```
--              </EMP_T>
--              <EMP_T EMPLOYEE_ID="31">
--                <LAST_NAME>Mary</LAST_NAME>
--              </EMP_T>
--            </EMPLIST>
--       </DEPTXML>
--    <DEPTXML DEPARTMENT_ID="20">
--    ...
-- </ROWSET>
```

これで、temp_clob_Tab 表から LOB データを選択し、結果を検証できます。結果は、10-21 ページの「DBMS_XMLGEN の問合せ結果の例」に示す例に類似しています。

リレーショナル・データの場合、結果はフラットでネストしていない XML 文書になります。ネストした XML 構造にするには、オブジェクト・リレーショナル・データを使用します。この場合のマッピングは次のとおりです。

- オブジェクト型は、XML 要素としてマップされます。第 5 章「XMLType の構造化されたマッピング」を参照してください。
- 型の属性は、親要素のサブ要素にマップされます。

> **注意：** 複雑な構造にするには、オブジェクト型を使用し、オブジェクト・ビューまたはオブジェクト表を作成します。正規マッピングを使用して、オブジェクト・インスタンスを XML にマップしてください。
>
> アットマーク（@）を列名または属性名で使用すると、マッピングで XML の囲み要素の属性に変換されます。

例 10-19 DBMS_XMLGEN: 複雑な XML の生成 - ネストした XML 文書に対するユーザー定義型の入力

ユーザー定義型の値を DBMS_XMLGEN ファンクションに入力すると、ユーザー定義型は正規マッピングを使用して XML 文書にマップされます。正規マッピングでは、ユーザー定義型の属性は XML 要素にマップされます。「@」で始まる名前の属性は、その前にある要素の属性にマップされます。

ユーザー定義型を使用して、結果の XML 文書内でネストすることができます。たとえば、EMP および DEPT という 2 つの表を考えてみます。

```
CREATE TABLE DEPT
(
 deptno number primary key,
 dname varchar2(20)
);

CREATE TABLE EMP
(
```

```
  empno number primary key,
  ename varchar2(20),
  deptno number references dept
);
```

この場合、部門とその従業員のデータの階層ビューを生成するには、適切なオブジェクト型を定義してデータベース内に構造を作成できます。次に例を示します。

```
CREATE TYPE EMP_T AS OBJECT
(
  "@empno" number,  -- empno defined as an attribute!
   ename varchar2(20)
);
/
-- You have defined the empno with an @ sign in front, to denote that it must
-- be mapped as an attribute of the enclosing Employee element.
CREATE TYPE EMPLIST_T AS TABLE OF EMP_T;
/
CREATE TYPE DEPT_T AS OBJECT
(
  "@deptno" number,
  dname varchar2(20),
  emplist emplist_t
);
/

-- Department type, DEPT_T, denotes the department as containing a list of
-- employees. You can now query the employee and department tables and get
-- the result as an XML document, as follows:
declare
   qryCtx dbms_xmlgen.ctxHandle;
   result CLOB;
begin

  -- get the query context;
  qryCtx := dbms_xmlgen.newContext(
 'SELECT
  dept_t(deptno,dname,
           CAST(MULTISET(select empno, ename
                from emp e
                where e.deptno = d.deptno) AS emplist_t)) AS deptxml
FROM dept d');

  -- set the maximum number of rows to be 5,
  dbms_xmlgen.setMaxRows(qryCtx, 5);

  -- set no row tag for this result as we have a single ADT column
```

```
  dbms_xmlgen.setRowTag(qryCtx,null);

  loop
    -- now get the result
    result := dbms_xmlgen.getXML(qryCtx);

    -- if there were no rows processed, then quit..!
    exit when dbms_xmlgen.getNumRowsProcessed(qryCtx) = 0;

    -- do whatever with the result..!
  end loop;
end;
/
```

MULTISET 演算子は、その部門の従業員のサブセット結果を、リストとそれを囲む CAST として扱い、それを適切なコレクション型にキャストします。こうすることで、それを囲む部門インスタンスを作成し、DBMS_XMLGEN ルーチンをコールして、オブジェクト・インスタンスの XML を作成します。結果は次のようになります。

```
-- <?xml version="1.0"?>
-- <ROWSET>
--  <DEPTXML deptno="10">
--    <DNAME>Sports</DNAME>
--    <EMPLIST>
--     <EMP_T empno="200">
--      <ENAME>John</ENAME>
--     </EMP_T>
--     <EMP_T empno="300">
--      <ENAME>Jack</ENAME>
--     </EMP_T>
--    </EMPLIST>
--  </DEPTXML>
--   <DEPTXML deptno="20">
--       <!-- .. other columns -->
--   </DEPTXML>
-- </ROWSET>
```

デフォルト名 ROW は、NULL に設定しているため存在しません。deptno および empno は囲み要素の属性になっています。

例 10-20 DBMS_XMLGEN: XML 形式でのデータベースからの発注書の生成

次の例では、DBMS_XMLGEN.getXMLType() を使用して、オブジェクト・ビューによってリレーショナル・データベースから発注書を XML 形式で生成します。この例は 5 ページにわたります。

```
-- Create relational schema and define Object Views
-- Note: DBMS_XMLGEN Package maps UDT attributes names
```

```
--       starting with '@' to xml attributes
-----------------------------------------------------
-- Purchase Order Object View Model

-- PhoneList Varray object type
CREATE TYPE PhoneList_vartyp AS VARRAY(10) OF VARCHAR2(20)
/

-- Address object type
CREATE TYPE Address_typ AS OBJECT (
  Street         VARCHAR2(200),
  City           VARCHAR2(200),
  State          CHAR(2),
  Zip            VARCHAR2(20)
  )
/

-- Customer object type
CREATE TYPE Customer_typ AS OBJECT (
  CustNo           NUMBER,
  CustName         VARCHAR2(200),
  Address          Address_typ,
  PhoneList        PhoneList_vartyp
)
/

-- StockItem object type
CREATE TYPE StockItem_typ AS OBJECT (
  "@StockNo"    NUMBER,
  Price      NUMBER,
  TaxRate    NUMBER
)
/

-- LineItems object type
CREATE TYPE LineItem_typ AS OBJECT (
  "@LineItemNo"   NUMBER,
  Item    StockItem_typ,
  Quantity     NUMBER,
  Discount     NUMBER
  )
/
-- LineItems Nested table
CREATE TYPE LineItems_ntabtyp AS TABLE OF LineItem_typ
/

-- Purchase Order object type
```

```
CREATE TYPE PO_typ AUTHID CURRENT_USER AS OBJECT (
  PONO                 NUMBER,
  Cust_ref             REF Customer_typ,
  OrderDate            DATE,
  ShipDate             TIMESTAMP,
  LineItems_ntab       LineItems_ntabtyp,
  ShipToAddr           Address_typ
 )
/

-- Create Purchase Order Relational Model tables

--Customer table
CREATE TABLE Customer_tab(
  CustNo                NUMBER NOT NULL,
  CustName              VARCHAR2(200) ,
  Street                VARCHAR2(200) ,
  City                  VARCHAR2(200) ,
  State                 CHAR(2) ,
  Zip                   VARCHAR2(20) ,
  Phone1                VARCHAR2(20),
  Phone2                VARCHAR2(20),
  Phone3                VARCHAR2(20),
  constraint cust_pk PRIMARY KEY (CustNo)
)
ORGANIZATION INDEX OVERFLOW;

-- Purchase Order table
CREATE TABLE po_tab (
   PONo        NUMBER, /* purchase order no */
   Custno      NUMBER constraint po_cust_fk references Customer_tab,
                                /*  Foreign KEY referencing customer */
   OrderDate   DATE, /*  date of order */
   ShipDate    TIMESTAMP, /* date to be shipped */
   ToStreet    VARCHAR2(200), /* shipto address */
   ToCity      VARCHAR2(200),
   ToState     CHAR(2),
   ToZip       VARCHAR2(20),
   constraint po_pk PRIMARY KEY(PONo)
);

--Stock Table
CREATE TABLE Stock_tab (
  StockNo      NUMBER constraint stock_uk UNIQUE,
  Price        NUMBER,
  TaxRate      NUMBER
);
```

```
--Line Items Table
CREATE TABLE LineItems_tab(
  LineItemNo           NUMBER,
  PONo                 NUMBER constraint LI_PO_FK REFERENCES po_tab,
  StockNo              NUMBER ,
  Quantity             NUMBER,
  Discount             NUMBER,
  constraint LI_PK PRIMARY KEY (PONo, LineItemNo)
);

-- create Object Views

--Customer Object View
CREATE OR REPLACE VIEW Customer OF Customer_typ
   WITH OBJECT IDENTIFIER(CustNo)
   AS SELECT c.Custno, C.custname,
             Address_typ(C.Street, C.City, C.State, C.Zip),
             PhoneList_vartyp(Phone1, Phone2, Phone3)
        FROM Customer_tab c;

--Purchase order view
CREATE OR REPLACE VIEW PO OF PO_typ
  WITH OBJECT IDENTIFIER (PONO)
   AS SELECT P.PONo,
             MAKE_REF(Customer, P.Custno),
             P.OrderDate,
             P.ShipDate,
             CAST( MULTISET(
                    SELECT LineItem_typ( L.LineItemNo,
                                 StockItem_typ(L.StockNo,S.Price,S.TaxRate),
                                           L.Quantity, L.Discount)
                     FROM LineItems_tab L, Stock_tab S
                     WHERE L.PONo = P.PONo and S.StockNo=L.StockNo )
                AS LineItems_ntabtyp),
         Address_typ(P.ToStreet,P.ToCity, P.ToState, P.ToZip)
        FROM PO_tab P;

-- create table with XMLType column to store po in XML format
create table po_xml_tab(
  poid number,
  poDoc XMLTYPE /* purchase order in XML format */
)
/

--------------------
-- Populate data
```

```
-------------------
-- Establish Inventory

INSERT INTO Stock_tab VALUES(1004, 6750.00, 2) ;
INSERT INTO Stock_tab VALUES(1011, 4500.23, 2) ;
INSERT INTO Stock_tab VALUES(1534, 2234.00, 2) ;
INSERT INTO Stock_tab VALUES(1535, 3456.23, 2) ;

-- Register Customers

INSERT INTO Customer_tab
  VALUES (1, 'Jean Nance', '2 Avocet Drive',
         'Redwood Shores', 'CA', '95054',
         '415-555-1212', NULL, NULL) ;

INSERT INTO Customer_tab
  VALUES (2, 'John Nike', '323 College Drive',
         'Edison', 'NJ', '08820',
         '609-555-1212', '201-555-1212', NULL) ;

-- Place Orders

INSERT INTO PO_tab
  VALUES (1001, 1, '10-APR-1997', '10-MAY-1997',
          NULL, NULL, NULL, NULL) ;

INSERT INTO PO_tab
  VALUES (2001, 2, '20-APR-1997', '20-MAY-1997',
         '55 Madison Ave', 'Madison', 'WI', '53715') ;

-- Detail Line Items

INSERT INTO LineItems_tab VALUES(01, 1001, 1534, 12,  0) ;
INSERT INTO LineItems_tab VALUES(02, 1001, 1535, 10, 10) ;
INSERT INTO LineItems_tab VALUES(01, 2001, 1004,  1,  0) ;
INSERT INTO LineItems_tab VALUES(02, 2001, 1011,  2,  1) ;


-----------------------------------------------------
-- Use DBMS_XMLGEN Package to generate PO in XML format
-- and store XMLTYPE in po_xml table
-----------------------------------------------------

declare
   qryCtx dbms_xmlgen.ctxHandle;
   pxml XMLTYPE;
   cxml clob;
```

```
begin

  -- get the query context;
  qryCtx := dbms_xmlgen.newContext('
                    select pono,deref(cust_ref) customer,p.OrderDate,p.shipdate,
                           lineitems_ntab lineitems,shiptoaddr
                    from po p'
            );

  -- set the maximum number of rows to be 1,
  dbms_xmlgen.setMaxRows(qryCtx, 1);
  -- set rowset tag to null and row tag to PurchaseOrder
  dbms_xmlgen.setRowSetTag(qryCtx,null);
  dbms_xmlgen.setRowTag(qryCtx,'PurchaseOrder');

  loop
    -- now get the po in xml format
    pxml := dbms_xmlgen.getXMLType(qryCtx);

    -- if there were no rows processed, then quit..!
    exit when dbms_xmlgen.getNumRowsProcessed(qryCtx) = 0;

    -- Store XMLTYPE po in po_xml table (get the pono out)
    insert into po_xml_tab (poid, poDoc)
       values(
            pxml.extract('//PONO/text()').getNumberVal(),
            pxml);
  end loop;
end;
/

---------------------------
-- list xml PurchaseOrders
---------------------------

set long 100000
set pages 100
select x.podoc.getClobVal() xpo
from   po_xml_tab x;
```

これによって、次の発注書の XML 文書が生成されます。

PurchaseOrder 1001:

```
<?xml version="1.0"?>
 <PurchaseOrder>
  <PONO>1001</PONO>
  <CUSTOMER>
```

```
   <CUSTNO>1</CUSTNO>
   <CUSTNAME>Jean Nance</CUSTNAME>
   <ADDRESS>
    <STREET>2 Avocet Drive</STREET>
    <CITY>Redwood Shores</CITY>
    <STATE>CA</STATE>
    <ZIP>95054</ZIP>
   </ADDRESS>
   <PHONELIST>
    <VARCHAR2>415-555-1212</VARCHAR2>
   </PHONELIST>
  </CUSTOMER>
  <ORDERDATE>10-APR-97</ORDERDATE>
  <SHIPDATE>10-MAY-97 12.00.00.000000 AM</SHIPDATE>
  <LINEITEMS>
   <LINEITEM_TYP LineItemNo="1">
    <ITEM StockNo="1534">
     <PRICE>2234</PRICE>
     <TAXRATE>2</TAXRATE>
    </ITEM>
    <QUANTITY>12</QUANTITY>
    <DISCOUNT>0</DISCOUNT>
   </LINEITEM_TYP>
   <LINEITEM_TYP LineItemNo="2">
    <ITEM StockNo="1535">
     <PRICE>3456.23</PRICE>
     <TAXRATE>2</TAXRATE>
    </ITEM>
    <QUANTITY>10</QUANTITY>
    <DISCOUNT>10</DISCOUNT>
   </LINEITEM_TYP>
  </LINEITEMS>
  <SHIPTOADDR/>
 </PurchaseOrder>
```

PurchaseOrder 2001:

```
<?xml version="1.0"?>
 <PurchaseOrder>
  <PONO>2001</PONO>
  <CUSTOMER>
   <CUSTNO>2</CUSTNO>
   <CUSTNAME>John Nike</CUSTNAME>
   <ADDRESS>
    <STREET>323 College Drive</STREET>
    <CITY>Edison</CITY>
    <STATE>NJ</STATE>
    <ZIP>08820</ZIP>
```

```
   </ADDRESS>
   <PHONELIST>
    <VARCHAR2>609-555-1212</VARCHAR2>
    <VARCHAR2>201-555-1212</VARCHAR2>
   </PHONELIST>
  </CUSTOMER>
  <ORDERDATE>20-APR-97</ORDERDATE>
  <SHIPDATE>20-MAY-97 12.00.00.000000 AM</SHIPDATE>
  <LINEITEMS>
   <LINEITEM_TYP LineItemNo="1">
    <ITEM StockNo="1004">
     <PRICE>6750</PRICE>
     <TAXRATE>2</TAXRATE>
    </ITEM>
    <QUANTITY>1</QUANTITY>
    <DISCOUNT>0</DISCOUNT>
   </LINEITEM_TYP>
   <LINEITEM_TYP LineItemNo="2">
    <ITEM StockNo="1011">
     <PRICE>4500.23</PRICE>
     <TAXRATE>2</TAXRATE>
    </ITEM>
    <QUANTITY>2</QUANTITY>
    <DISCOUNT>1</DISCOUNT>
   </LINEITEM_TYP>
  </LINEITEMS>
  <SHIPTOADDR>
   <STREET>55 Madison Ave</STREET>
   <CITY>Madison</CITY>
   <STATE>WI</STATE>
   <ZIP>53715</ZIP>
  </SHIPTOADDR>
 </PurchaseOrder>
```

例 10-21 DBMS_XMLGEN: 渡された PL/SQL REF カーソルからの新しいコンテキスト・ハンドルの生成

```
CREATE OR REPLACE FUNCTION joe3 RETURN CLOB
IS
  ctx1 number := 2;
  ctx2 number;
  xmldoc CLOB;
  page NUMBER := 0;
  xmlpage boolean := true;
  refcur SYS_REFCURSOR;
BEGIN
   OPEN refcur FOR 'select * from emp where rownum < :1' USING ctx1;
```

```
   ctx2 := DBMS_XMLGEN.newContext( refcur);

   ctx1 := 4;
   OPEN refcur FOR 'select * from emp where rownum < :1' USING ctx1;
   ctx1 := 5;
   OPEN refcur FOR 'select * from emp where rownum < :1' USING ctx1;
   dbms_lob.createtemporary(xmldoc, TRUE);
    -- xmldoc will have 4 rows
   xmldoc := DBMS_XMLGEN.getXML(ctx2,DBMS_XMLGEN.NONE);
    DBMS_XMLGEN.closeContext(ctx2);
 return xmldoc;
END;
/
```

Oracle が提供する SQL 関数を使用した XML の生成

Oracle9*i* データベースでは、SQL 標準関数に加えて、XML の生成を支援するために SYS_XMLGEN 関数および SYS_XMLAGG 関数が提供されています。

SYS_XMLGEN() 関数

この Oracle 固有の SQL 関数は、XMLElement() に類似していますが、単一の引数を取り、結果を XML に変換するという点で異なります。他の XML 生成関数とは異なり、SYS_XMLGEN() は常に整形式の XML 文書を戻します。また、問合せレベルで機能する DBMS_XMLGEN とは異なり、SYS_XMLGEN() は行レベルで機能し、各行に対して XML 文書を戻します。

例 10-22 SQL_XMLGEN を使用した XML の作成

SYS_XMLGEN() は、SQL 問合せ内で XML インスタンスを作成し、問い合せます。次に例を示します。

```
SELECT SYS_XMLGEN(employee_id)
   FROM employees WHERE last_name LIKE 'Scott%';
```

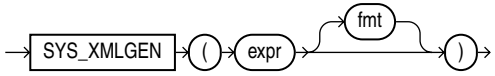
結果の XML 文書は次のようになります。

```
<?xml version=''1.0''?>
<employee_id>60</employee_id>
```

SYS_XMLGEN の構文

SYS_XMLGEN() は、XML 文書に変換するスカラー値、オブジェクト型または XMLType インスタンスを取ります。また、オプションの XMLFormat（以前の XMLGenFormatType）オブジェクトも取ります。このオブジェクトを使用すると、結果として戻される XML 文書の書式設定オプションを指定できます。図 10-10 を参照してください。

図 10-10 SYS_XMLGEN の構文



SYS_XMLGEN() は、データベースの特定の行および列に対して評価する式を使用して、XML 文書を含む XMLType 型のインスタンスを戻します。*expr* は、スカラー値、ユーザー定義型または XMLType インスタンスのいずれかになります。

- *expr* がスカラー値の場合、この関数は、スカラー値を含む XML 要素を戻します。
- *expr* が型の場合、この関数は、ユーザー定義型属性を XML 要素にマップします。
- *expr* が XMLType インスタンスの場合、この関数は、デフォルトのタグ名が ROW である XML 要素で文書を囲みます。

デフォルトでは、XML 文書の要素は *expr* の要素と一致します。たとえば、*expr* が列名に変換される場合、XML の囲み要素は同じ列名になります。XML 文書を別の方法でフォーマットする場合は *fmt* を指定します。これは、XMLFormat オブジェクトのインスタンスです。

今回のリリースでは、SYS_XMLGEN() の書式設定引数は、スキーマおよび要素名を受け入れ、登録されたスキーマに準拠する XML 文書を生成します。

```
SELECT sys_xmlgen(
   dept_t(d.deptno, d.dname, d.loc,
        cast(multiset(
          SELECT emp_t(e.empno, e.ename, e.job, e.mgr, e.hiredate,e.sal, e.comm)
          FROM emp e
          WHERE e.deptno = d.deptno) AS emplist_t),
   xmlformat.createformat('Department', 'http://www.oracle.com/dept.xsd'))
 FROM dept d;
```

例 10-23 SYS_XMLGEN(): 従業員表からの従業員の電子メール ID の取得および EMAIL 要素を持つ XML の生成

次の例では、サンプル表 oe.employees から、従業員（employee_id 値が 205）の電子メール ID を取得し、EMAIL 要素を持つ XML 文書を含む XMLType のインスタンスを生成します。

```
SELECT SYS_XMLGEN(email).getStringVal()
   FROM employees
   WHERE employee_id = 205;

SYS_XMLGEN(EMAIL).GETSTRINGVAL()
--------------------------------------------------------------
<EMAIL>SHIGGENS</EMAIL>
```

SYS_XMLGEN() のメリット

SYS_XMLGEN() のメリットは次のとおりです。

- SQL 問合せ内で XML インスタンスを作成し、問い合せることができます。
- オブジェクト・リレーショナル・インフラストラクチャを使用して、簡単なリレーショナル表から複雑なネストした XML インスタンスを作成できます。

SYS_XMLGEN() は、次のいずれかから XML 文書を作成します。

- ユーザー定義型インスタンス
- 渡されるスカラー値
- XML

また、文書に含まれる XMLType インスタンスを戻します。

また、SYS_XMLGEN() は、オプションで XMLFormat オブジェクト型を入力します。このオブジェクト型を介して、SQL の結果をカスタマイズできます。書式設定オブジェクトが NULL の場合、デフォルトのマッピング動作が使用されます。

XMLFormat オブジェクト型の使用

XMLFormat を使用して、SYS_XMLGEN() および SYS_XMLAGG() 関数に対して書式設定引数を指定できます。

SYS_XMLGEN() は、XML 文書を含む XMLType 型のインスタンスを戻します。Oracle9*i* データベースでは、XMLFormat オブジェクトが提供されます。このオブジェクトを使用すると、SYS_XMLGEN 関数の出力を書式設定できます。

表 10-2 に、XMLFormat オブジェクトの XMLFormat 属性を示します。この型を実装する関数を表の後に示します。

表 10-2 XMLFormat オブジェクトの属性

属性	データ型	用途
enclTag	VARCHAR2(100)	SYS_XMLGEN 関数の結果の囲みタグ名です。関数への入力が列名の場合、デフォルトはその列名です。それ以外の場合、デフォルトは ROW です。schemaType が USE_GIVEN_SCHEMA に設定されている場合、この属性によって XMLSchema 要素の名前も指定されます。
schemaType	VARCHAR2(100)	出力ドキュメントに対するスキーマ生成の型です。有効な値は、NO_SCHEMA および USE_GIVEN_SCHEMA です。デフォルト値は、NO_SCHEMA です。
schemaName	VARCHAR2(4000)	schemaType の値が USE_GIVEN_SCHEMA である場合に、Oracle が使用するターゲット・スキーマの名前です。schemaName を指定すると、Oracle では囲みタグが要素名として使用されます。
targetNameSpace	VARCHAR2(4000)	スキーマ（schemaType が GEN_SCHEMA_* または USE_GIVEN_SCHEMA）を指定した場合のターゲットの名前空間です。
dburl	VARCHAR2(2000)	WITH_SCHEMA を指定した場合に使用するデータベースの URL です。この属性を指定しない場合、Oracle では、型の URL が相対参照 URL として宣言されます。
processingIns	VARCHAR2(4000)	関数出力の先頭の要素の前に追加されるユーザーの処理命令です。

例 10-24 createFormat を使用した書式設定オブジェクトの作成

静的メンバー関数 createformat を使用して、XMLFormat オブジェクトを実装できます。この関数では、ほとんどの値がデフォルトで指定されています。次に例を示します。

```
STATIC FUNCTION createFormat(
     enclTag IN varchar2 := 'ROWSET',
     schemaType IN varchar2 := 'NO_SCHEMA',
     schemaName IN varchar2 := null,
     targetNameSpace IN varchar2 := null,
     dburlPrefix IN varchar2 := null,
     processingIns IN varchar2 := null) RETURN XMLGenFormatType,
  MEMBER PROCEDURE genSchema (spec IN varchar2),
  MEMBER PROCEDURE setSchemaName(schemaName IN varchar2),
  MEMBER PROCEDURE setTargetNameSpace(targetNameSpace IN varchar2),
  MEMBER PROCEDURE setEnclosingElementName(enclTag IN varchar2),
  MEMBER PROCEDURE setDbUrlPrefix(prefix IN varchar2),
  MEMBER PROCEDURE setProcessingIns(pi IN varchar2),
  CONSTRUCTOR FUNCTION XMLGenFormatType (
     enclTag IN varchar2 := 'ROWSET',
```

```
schemaType IN varchar2 := 'NO_SCHEMA',
schemaName IN varchar2 := null,
targetNameSpace IN varchar2 := null,
dbUrlPrefix IN varchar2 := null,
processingIns IN varchar2 := null) RETURN SELF AS RESULT
```

注意： XMLFormat オブジェクトは、XMLGenFormatType の新しい名前です。どちらの名前も使用できます。

例 10-25 SYS_XMLGEN(): スカラー値から XML 文書要素のコンテンツへの変換

SYS_XMLGEN() にスカラー値を入力すると、SYS_XMLGEN() はそのスカラー値を、スカラー値を含む要素に変換します。次に例を示します。

```
select sys_xmlgen(empno) from scott.emp where rownum < 2;
```

次に示すとおり、SYS_XMLGEN() は要素として empno 値を含む XML 文書を戻します。

```
<?xml version="1.0"?>
<EMPNO>30</EMPNO>
```

囲み要素名（この場合は EMPNO）は、演算子に渡される列名から導出されます。SELECT 文の結果は、XMLType を含む行になることに注意してください。

例 10-26 デフォルトの列名 ROW の生成

前述の例では、文書に列名 EMPNO を使用しています。列名を直接導出できない場合、デフォルト名 ROW が使用されます。次の構文を考えてみます。

```
SELECT sys_xmlgen(empno).getclobval()
FROM scott.emp
WHERE rownum < 2;
```

次の XML 出力が生成されます。

```
<?xml version="1.0"?>
<ROW>60</ROW>
```

これは、この関数が式の名前を推論できないためです。デフォルトの ROW タグは、XMLFormat（以前の XMLGenFormatType）オブジェクトを演算子の最初の引数に指定することによって変更できます。

例 10-27 デフォルトの列名のオーバーライド : 演算子の最初の引数への XMLFormat オブジェクトの指定

たとえば、前述の例の場合、タグ名として EMPNO を使用した結果を取得するには、次のとおり、書式設定引数を関数に指定します。

```
SELECT sys_xmlgen(empno *2,
     xmlformat.createformat('EMPNO')).getClobVal()
FROM emp;
```

これによって、次の XML が出力されます。

```
<?xml version="1.0"?>
<EMPNO>60</EMPNO>
```

例 10-28 SYS_XMLGEN(): ユーザー定義型から XML への変換

ユーザー定義型の値を SYS_XMLGEN() に入力すると、ユーザー定義型は、正規マッピングを使用して XML 文書にマップされます。正規マッピングでは、ユーザー定義型の属性は XML 要素にマップされます。

「@」で始まる名前のすべての型属性は、その前にある要素の属性にマップされます。ユーザー定義型を使用して、結果の XML 文書内でネストすることができます。

DBMS_XMLGEN の項（10-31 ページの「例 10-18DBMS_XMLGEN: 複雑な XML の生成」）に示す例を使用して、次のとおり、従業員および部門を例とした階層 XML を生成できます。

```
SELECT SYS_XMLGEN(
  dept_t(deptno,dname,
        CAST(MULTISET(
            select empno, ename
            from emp e
            where e.deptno = d.deptno) AS emplist_t))).getClobVal()
     AS deptxml
FROM dept d;
```

MULTISET 演算子は、その部門の従業員のサブセット結果を、リストとそれを囲む CAST として扱い、それを適切なコレクション型にキャストします。こうすることで、それを囲む部門インスタンスを作成し、SYS_XMLGEN() をコールして、オブジェクト・インスタンスの XML を作成します。

部門の各行の結果は次のようになります。

```
<?xml version="1.0"?>
<ROW DEPTNO="100">
  <DNAME>Sports</DNAME>
  <EMPLIST>
    <EMP_T EMPNO="200">
      <ENAME>John</ENAME>
    <EMP_T>
```

```
    <EMP_T>
      <ENAME>Jack</ENAME>
    </EMP_T>
 </EMPLIST>
</ROW>
```

関数が入力オペランドの名前を直接推論できないため、デフォルト名 ROW が存在します。

注意： 前述の例では、SYS_XMLGEN() 関数と DBMS_XMLGEN パッケージの違いが明確になります。

- SYS_XMLGEN は、SQL 問合せ内で動作し、行内の式および列を操作します。
- DBMS_XMLGEN は、結果セット全体で動作します。

例 10-29 SYS_XMLGEN(): XMLType インスタンスの変換

XML 文書を SYS_XMLGEN() に渡す場合、SYS_XMLGEN() は、タグ名がデフォルトの ROW である要素、または書式設定オブジェクトを介して渡される名前で、その文書（またはフラグメント）を囲みます。この機能を使用して、XML 文書のフラグメントを整形式の文書に変換できます。

たとえば、次の文書の extract() 操作では、フラグメントを戻すことができます。次の文書から EMPNO 要素を抽出するとします。

```
<DOCUMENT>
  <EMPLOYEE>
    <ENAME>John</ENAME>
    <EMPNO>200</EMPNO>
  </EMPLOYEE>
  <EMPLOYEE>
    <ENAME>Jack</ENAME>
    <EMPNO>400</EMPNO>
  </EMPLOYEE>
  <EMPLOYEE>
    <ENAME>Joseph</ENAME>
    <EMPNO>300</EMPNO>
  </EMPLOYEE>
</DOCUMENT>
```

次の文を使用します。

```
SELECT e.podoc.extract('/DOCUMENT/EMPLOYEE/ENAME')
   FROM po_xml_tab e;
```

次の XML 文書のフラグメントが生成されます。

```
<ENAME>John</ENAME>
<ENAME>Jack</ENAME>
<ENAME>Joseph</ENAME>
```

このフラグメントを妥当な XML 文書にするには、次のとおり、SYS_XMLGEN() をコールして、文書の前後に囲み要素を置きます。

```
select SYS_XMLGEN(e.podoc.extract('/DOCUMENT/EMPLOYEE/ENAME')).getclobval()
   from po_xml_tab e;
```

これによって、次のとおり、結果の前後に要素 ROW が置かれます。

```
<?xml version="1.0"?>
<ROW>
  <ENAME>John</ENAME>
  <ENAME>Jack</ENAME>
  <ENAME>Joseph</ENAME>
</ROW>
```

注意： 入力が列の場合、その列名はデフォルトとして使用されます。囲み要素名は、書式設定オブジェクトを使用して変更できます。このオブジェクトは、追加の引数として関数に渡すことができます。10-44 ページの「XMLFormat オブジェクト型の使用」を参照してください。

例 10-30 SYS_XMLGEN(): オブジェクト・ビューでの SYS_XMLGEN() の使用

```
-- create Purchase Order object type
CREATE OR REPLACE TYPE PO_typ AUTHID CURRENT_USER AS OBJECT (
  PONO                  NUMBER,
  Customer              Customer_typ,
  OrderDate             DATE,
  ShipDate              TIMESTAMP,
  LineItems_ntab        LineItems_ntabtyp,
  ShipToAddr            Address_typ
 )
/

--Purchase order view
CREATE OR REPLACE VIEW PO OF PO_typ
  WITH OBJECT IDENTIFIER (PONO)
   AS SELECT P.PONo,
             Customer_typ(P.Custno,C.CustName,C.Address,C.PhoneList),
             P.OrderDate,
             P.ShipDate,
             CAST( MULTISET(
```

```
                    SELECT LineItem_typ( L.LineItemNo,
                                StockItem_typ(L.StockNo,S.Price,S.TaxRate),
                                         L.Quantity, L.Discount)
                     FROM LineItems_tab L, Stock_tab S
                     WHERE L.PONo = P.PONo and S.StockNo=L.StockNo )
                AS LineItems_ntabtyp),
         Address_typ(P.ToStreet,P.ToCity, P.ToState, P.ToZip)
        FROM PO_tab P, Customer C
        WHERE P.CustNo=C.custNo;

-------------------------------------------------------
-- Use SYS_XMLGEN() to generate PO in XML format
-------------------------------------------------------
set long 20000
set pages 100
SELECT SYS_XMLGEN(value(p),
              sys.xmlformat.createFormat('PurchaseOrder')).getClobVal() PO
FROM po p
WHERE p.pono=1001;
```

これによって、XML 形式で発注書が戻されます。

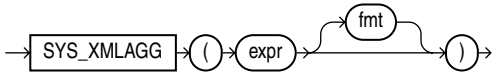
```
<?xml version="1.0"?>
<PurchaseOrder>
 <PONO>1001</PONO>
 <CUSTOMER>
  <CUSTNO>1</CUSTNO>
  <CUSTNAME>Jean Nance</CUSTNAME>
  <ADDRESS>
   <STREET>2 Avocet Drive</STREET>
   <CITY>Redwood Shores</CITY>
   <STATE>CA</STATE>
   <ZIP>95054</ZIP>
  </ADDRESS>
  <PHONELIST>
   <VARCHAR2>415-555-1212</VARCHAR2>
  </PHONELIST>
 </CUSTOMER>
 <ORDERDATE>10-APR-97</ORDERDATE>
 <SHIPDATE>10-MAY-97 12.00.00.000000 AM</SHIPDATE>
 <LINEITEMS_NTAB>
  <LINEITEM_TYP LineItemNo="1">
   <ITEM StockNo="1534">
    <PRICE>2234</PRICE>
    <TAXRATE>2</TAXRATE>
   </ITEM>
   <QUANTITY>12</QUANTITY>
   <DISCOUNT>0</DISCOUNT>
```

```
  </LINEITEM_TYP>
  <LINEITEM_TYP LineItemNo="2">
   <ITEM StockNo="1535">
    <PRICE>3456.23</PRICE>
    <TAXRATE>2</TAXRATE>
   </ITEM>
   <QUANTITY>10</QUANTITY>
   <DISCOUNT>10</DISCOUNT>
  </LINEITEM_TYP>
 </LINEITEMS_NTAB>
 <SHIPTOADDR/>
</PurchaseOrder>
```

SYS_XMLAGG() 関数

SYS_XMLAGG() 関数は、すべての XML 文書またはフラグメント（expr によって表される）を集計し、単一の XML 文書を生成します。この関数は、デフォルト名 ROWSET の新しい囲み要素を追加します。XML 文書を別の方法でフォーマットするには、fmt を指定します。これは、XMLFORMAT オブジェクトのインスタンスです。

図 10-11 SYS_XMLAGG() の構文



参照：『Oracle9*i* SQL リファレンス』を参照してください。

XSQL Pages パブリッシング・フレームワークを使用した XML の生成

Oracle9*i* データベースでは、XML ベースのデータベース・コンテンツの格納および問合せに使用する XMLType が導入されています。これらのデータベースの XML 機能を使用して、<xsql:include-xml> アクション要素を使用することによって XSQL ページに挿入する XML を生成できます。

<xsql:include-xml> 要素内の SELECT 文によって、1 列を含む 1 行が戻されます。この列は、整形式の XML 文書を含む CLOB 値または VARCHAR2 値のいずれかです。XML 文書は解析され、XSQL ページに挿入されます。

例 10-31 XSQL Servlet の <xsql:include-xml> およびネストした XMLAgg() 関数を使用した、単一の XML 文書への結果の集計

次の例では、ネストした xmlagg() 関数を使用して、部門およびネストした従業員を含む、動的に構築された XML 文書の結果を集計し、結果として、<DepartmentList> 要素で囲んだ単一の XML 文書を戻します。

```
<xsql:include-xml connection="orcl92" xmlns:xsql="urn:oracle-xsql">
  select XmlElement("DepartmentList",
           XmlAgg(
             XmlElement("Department",
               XmlAttributes(deptno as "Id"),
               XmlForest(dname as "Name"),
               (select XmlElement("Employees",
                         XmlAgg(
                           XmlElement("Employee",
                             XmlAttributes(empno as "Id"),
                             XmlForest(ename as "Name",
                                       sal   as "Salary",
                                       job   as "Job")
                           )
                         )
                       )
                 from emp e
                where e.deptno = d.deptno
               )
             )
           )
         ).getClobVal()
   from dept d
  order by dname
</xsql:include-xml>
```

例 10-32 XSQL Servlet の <xsql:include-xml>、XMLElement() および XMLAgg() を使用した Oracle9*i* データベースからの XML の生成

データベースでは、XML フラグメントを単一の結果の文書に集計する方が効率的であるため、<xsql:include-xml> 要素では、指定した問合せの最初の行のみを取り出す方法をお薦めします。

たとえば、MOVIES という XmlType 表に多くの XML 文書 <Movie> を格納している場合、各文書の形式は次のようになります。

```
<Movie Title="The Talented Mr.Ripley" RunningTime="139" Rating="R">
    <Director>
      <First>Anthony</First>
      <Last>Minghella</Last>
    </Director>
```

```
    <Cast>
      <Actor Role="Tom Ripley">
        <First>Matt</First>
        <Last>Damon</Last>
      </Actor>
      <Actress Role="Marge Sherwood">
        <First>Gwenyth</First>
        <Last>Paltrow</Last>
      </Actress>
      <Actor Role="Dickie Greenleaf">
        <First>Jude</First>
        <Last>Law</Last>
        <Award From="BAFTA" Category="Best Supporting Actor"/>
      </Actor>
    </Cast>
</Movie>
```

Oracle9*i* データベースの組込み XPath 問合せ機能を使用して、次のような問合せを発行することによって、データベース内の映画からオスカーを受賞したすべての出演者の集計リストを抽出できます。

```
SELECT xmlelement("AwardedActors",
       xmlagg(extract(value(m),
              '/Movie/Cast/*[Award[@From="Oscar"]]')))
    FROM movies m;

-- To include this query result of XMLType into your XSQL page,
-- simply paste the query inside an <xsql:include-xml> element, and add
-- a getClobVal() method call to the query expression so that the result will
-- be returned as a CLOB instead of as an XMLType to the client:
<xsql:include-xml connection="orcl92" xmlns:xsql="urn:oracle-xsql">
  select xmlelement("AwardedActors",
           xmlagg(extract(value(m),
                  '/Movie/Cast/*[Award[@From="Oscar"]]'))).getClobVal()
    from movies m
</xsql:include-xml>
```

> **注意：** この場合も、XMLElement() と XMLAgg() を組み合せて使用し、データベースで、問合せによって識別されたすべての XML フラグメントを単一の整形式の XML 文書に集計します。

そうしなかった場合、XSQL Page Processor によって、次のような CLOB が解析されます。

```
<Actor>...</Actor>
<Actress>...</Actress>
```

XML 1.0 仕様で必要な単一のドキュメント要素が存在しないため、この XML 文書は整形式ではありません。xmlelement() と xmlagg() を組み合せて使用すると、次のような整形式の結果が生成されます。

```
<AwardedActors>
  <Actor>...</Actor>
  <Actress>...</Actress>
</AwardedActors>
```

この整形式の XML 文書は解析され、XSQL ページに挿入されます。

> **参照：** XSQL Pages パブリック・フレームワークの詳細は、『Oracle9*i* XML Developer's Kit ガイド - XDK』の「XDK for Java」の章を参照してください。

XML SQL Utility（XSU）を使用した XML の生成

Oracle9*i* データベースでは、Oracle XML SQL Utility（XSU）を使用して、XML を生成できます。これは、中間層またはクライアントで XML を生成する場合に有効です。また、今回のリリースの XSU は、XMLType 列を含む表での XML の生成をサポートしています。

例 10-33 XSU for Java の getXML を使用した XML の生成

たとえば、次のような表 parts があるとします。

```
CREATE TABLE parts ( PartNo number, PartName varchar2(20), PartDesc xmltype );
```

この表に XML を生成するには、Java を使用して次のようにコールします。

```
java OracleXML getXML -user "scott/tiger" -rowTag "Part" "select * from parts"
```

これによって、次の結果が戻されます。

```
<Parts>
  <Part>
    <PartNo>1735</PartNo>
    <PartName>Gizmo</PartName>
    <PartDesc>
      <Description>
        <Title>Description of the Gizmo</Title>
        <Author>John Smith</Author>
        <Body>
          The <b>Gizmo</b> is <i>grand</i>.
        </Body>
      </Description>
    </PartDesc>
  </Part>
```

```
  ...
</Parts>
```

参照： XSU の詳細は、『Oracle9*i* XML Developer's Kit ガイド - XDK』を参照してください。

11

XMLType ビュー

この章では、XMLType ビューを作成および使用する方法を説明します。この章の内容は次のとおりです。

- XMLType ビューの概要
- XML Schema に基づかない XMLType ビューの作成
- XML Schema に基づく XMLType ビューの作成
- XMLType 表からの XMLType ビューの作成
- REF() を使用した XMLType ビュー・オブジェクトの参照
- XMLType ビューでのデータ操作言語（DML）
- XMLType ビューでのクエリー・リライト
- XML Schema に基づく XML の非定型生成
- ユーザーが指定する情報の検証

XMLType ビューの概要

XMLType ビューは、既存のリレーショナル・データおよびオブジェクト・リレーショナル・データを XML 形式でラップします。XMLType ビューを使用する主なメリットは次のとおりです。

- 基本のレガシー・データを移行せずに、XMLSchema 機能を使用する Oracle XML DB の新しい XML 機能を使用できます。
- XMLType 表で使用可能なオブジェクト・リレーショナル記憶域または CLOB 記憶域の他に、様々な形式の記憶域を使用できます。

XMLType ビューは、オブジェクト・ビューに類似しています。XMLType ビューの各行は、XMLType インスタンスに対応しています。ビュー内の各行を一意に識別するためのオブジェクト識別子は、XMLType 値に対して extract() などの式を使用して作成できます。

XMLType 表と同様に、XMLType ビューは XML Schema に準拠させることができます。これによって、より厳密な型指定ができ、これらのビューの問合せを最適化することができます。

XML Schema で XMLType ビューを使用するには、まず、XML と SQL オブジェクト型の間のマッピングを表す注釈が付いた XML Schema を登録する必要があります。その後、適切な SQL オブジェクト型のインスタンスを構築する基礎となる問合せを指定することによって、登録された XML Schema に準拠する XMLType ビューを作成できます。

> **参照：** 次の章、項または付録を参照してください。
>
> - 第 5 章「XMLType の構造化されたマッピング」
> - 26-71 ページの「7.0 XML DB Demo: ビューを使用した、関連ツールから XML へのアクセス」
> - 付録 B「XML Schema の手引き」

この章では、XMLType ビューを作成する、次の 2 つの主な方法について説明します。

- XML 生成関数に基づく方法
- オブジェクト型に基づく方法

XML Schema に基づかない XMLType ビューの作成

XMLType のビュー、または 1 つ以上の XMLType 列を含むビューを作成するには、SQL の XML 生成関数（特に新しい SQLX 標準に準拠したもの）を使用します。

> **参照：** SQLX 生成関数の詳細は、第 10 章「データベースからの XML データの生成」を参照してください。

例 11-1 XMLType ビュー : XMLElement() 関数を使用した XMLType ビューの作成

次の例では、XMLElement() 生成関数を使用して XMLType ビューを作成します。

```
DROP TABLE employees;
CREATE TABLE employees
(empno number(4), fname varchar2(20), lname varchar2(20), hire date, salary
number(6));

INSERT INTO employees VALUES
(2100, 'John', 'Smith', Date'2000-05-24', 30000);

INSERT INTO employees VALUES
(2200, 'Mary', 'Martin', Date'1996-02-01', 30000);

CREATE OR REPLACE VIEW Emp_view OF XMLTYPE  WITH OBJECT ID
     (EXTRACT(sys_nc_rowinfo$,'/Emp/@empno').getnumberval())
       AS SELECT XMLELEMENT("Emp", XMLAttributes(empno),
            XMLForest(e.fname ||' '|| e.lname AS "name",
                        e.hire AS "hiredate")) AS "result"
          FROM employees e
          WHERE salary > 20000;
```

この XMLType ビューに対して問合せを行うと、次の従業員データが XML 形式で戻されます。

```
SELECT * FROM Emp_view;
 <Emp empno="2100">
   <name>John Smith</name>
   <hiredate>2000-05-24</hiredate>
 </Emp>

 <Emp empno="2200">
  <name>Mary Martin</name>
  <hiredate>1996-02-01</hiredate>
 </Emp>
```

文書内の empno 属性は、各行に一意の識別子になります。SYS_NC_ROWINFO$ は、行 XMLType インスタンスを参照する仮想列です。

注意： 以前のリリースでは、オブジェクト識別子句は、指定されたビューのオブジェクト型の属性のみをサポートしていました。今回のリリースでは、スカラー値を戻すすべての式をサポートするように拡張されています。

これらの XMLType ビューに対して DML 操作を行うことができますが、通常、DML 操作を処理するために INSTEAD OF トリガーを作成する必要があります。

XMLType ビューは、SYS_XMLGEN を使用して作成することもできます。SYS_XMLGEN を使用して同じ問合せ結果を生成する同等の問合せを次に示します。

```
CREATE TYPE Emp_t AS OBJECT  ("@empno" number(4), fname varchar2(2000),
lname     varchar2(2000), hiredate date);

CREATE VIEW employee_view OF XMLTYPE WITH OBJECT ID
     (EXTRACT(sys_nc_rowinfo$,'/Emp/@empno').getnumberval())
        AS SELECT SYS_XMLGEN(emp_t(empno, fname, lname, hire),
XMLFORMAT('EMP'))
          FROM employees e
          WHERE salary > 20000;
```

リレーショナル表、リレーショナル・ビュー、オブジェクト・リレーショナル表およびオブジェクト・リレーショナル・ビュー内の既存のデータは、このメカニズムを使用して XML として公開できます。また、単純な XPath 検索を含む SYS_XMLGEN ビューの問合せは、オブジェクト属性に直接アクセスするためのクエリー・リライトの対象になります。

XML Schema に基づく XMLType ビューの作成

XML Schema に基づく XMLType ビューでは、結果として戻される XML 値が、登録された XML Schema 内の特定の要素に制限されます。XML Schema に基づく XMLType を作成するには、主に 2 つの方法があります。

- **XMLAgg、XMLElement、XMLForest などの SQL/XML 関数を使用する方法：** この方法では、オブジェクト型を作成せずに、単純な XML 生成関数を使用して XMLType ビューを作成します。SQL/XML 関数を使用して XMLType ビューを作成するメカニズムは、オブジェクト型またはオブジェクト・ビューを作成する必要がないため、より簡単です。ただし、XML Schema とともにオブジェクト型を使用すると、クエリー・リライト機能が使用可能になるというメリットもあります。

  参照： 11-5 ページの「SQL/XML 生成関数を使用した XML Schema に基づく XMLType ビューの作成」を参照してください。

- **オブジェクト型またはオブジェクト・ビュー（あるいはその両方）を使用する方法 :** この方法では、オブジェクト型またはオブジェクト・ビューを使用して、XMLType ビューを作成します。XMLType ビューを作成するメカニズムは、すでにオブジェクト・リレーショナル・スキーマが存在し、そのスキーマを直接 XML にマップする場合に便利です。また、ビューが XML Schema に基づくため、パフォーマンス（メモリーおよびアクセス）の最適化を行うことができます。

  参照 : 11-10 ページの「オブジェクト型およびビューを使用した XMLType ビューの作成」を参照してください。

SQL/XML 生成関数を使用した XML Schema に基づく XMLType ビューの作成

SQL/XML 生成関数を使用すると、前述の XML Schema に基づかない場合と同様に、XML Schema に基づく XMLType ビューを作成できます。XML Schema に基づく XMLType ビューを作成するには、次の手順を実行します。

1. 必要な XML 構造を含む XML Schema 文書を作成および登録します。
2. SQL/XML 関数を使用して、その XML Schema に準拠する XMLType ビューを作成します。

注意 : 今回のリリースでは、これらの SQL/XML ビューに対する XPath の述語はリライトされません。そのため、これらのビューに対する問合せで extract や existsNode などの述語が使用されている場合、ビューのすべての行に対して問合せが機能的に評価されます。ビューの問合せ実行性が重要である場合、かわりにオブジェクト・リレーショナル形式の使用を検討してください。

例 11-2 SQL/XML 関数を使用した XML Schema に基づく XMLType ビューの作成

手順 1: XML Schema emp_simple.xsd の登録

従業員を定義する XML 構造を含む emp_simple.xsd という XML Schema があると想定します。まず XML Schema を登録し、URL を使用して識別します。

```
BEGIN
 dbms_xmlschema.deleteSchema('http://www.oracle.com/emp_simple.xsd', 4);
END;
/
BEGIN
 dbms_xmlschema.registerSchema('http://www.oracle.com/emp_simple.xsd',
   '<schema xmlns="http://www.w3.org/2001/XMLSchema"
       targetNamespace="http://www.oracle.com/emp_simple.xsd" version="1.0"
       xmlns:xdb="http://xmlns.oracle.com/xdb"
       elementFormDefault="qualified">
 <element name = "Employee">
```

```
  <complexType>
   <sequence>
    <element name = "EmployeeId" type = "positiveInteger"/>
    <element name = "Name" type = "string"/>
    <element name = "Job" type = "string"/>
    <element name = "Manager" type = "positiveInteger"/>
    <element name = "HireDate" type = "date"/>
    <element name = "Salary" type = "positiveInteger"/>
    <element name = "Commission" type = "positiveInteger"/>
    <element name = "Dept">
       <complexType>
          <sequence>
            <element name = "DeptNo" type = "positiveInteger" />
            <element name = "DeptName" type = "string"/>
            <element name = "Location" type = "string"/>
          </sequence>
       </complexType>
    </element>
   </sequence>
  </complexType>
 </element>
</schema>', TRUE, TRUE, FALSE);
END;
/
```

この文によって、XML Schema が次の目的の位置に登録されます。

```
"http://www.oracle.com/emp_simple.xsd"
```

手順 2: SQL/XML 関数を使用した XMLType ビューの作成

今回のリリースでは、SQL/XML 関数を使用して、XML Schema に基づく XMLType ビューを作成できます。また、XMLTransform()、または XML を生成するその他の SQL 関数を使用することもできます。結果として戻される XML は、このビューに指定した XML Schema に準拠している必要があります。

SQL/XML 関数を使用して XML Schema に基づくコンテンツを生成する場合、すべての要素に適切な名前空間情報を指定し、xsi:schemaLocation 属性を使用してスキーマの場所を指定する必要があります。これらは、すべて XMLAttributes 句を使用して指定できます。

```
CREATE OR REPLACE VIEW emp_simple_xml OF XMLTYPE
     XMLSCHEMA "http://www.oracle.com/emp_simple.xsd" ELEMENT "Employee"
     WITH OBJECT ID (extract(sys_nc_rowinfo$,
                             '/Employee/EmployeeId/text()').getnumberval()) AS
     SELECT XMLElement("Employee",
       XMLAttributes( 'http://www.oracle.com/emp_simple.xsd' AS "xmlns" ,
              'http://www.w3.org/2001/XMLSchema-instance' AS "xmlns:xsi",
```

```
            'http://www.oracle.com/emp_simple.xsd
             http://www.oracle.com/emp_simple.xsd' AS "xsi:schemaLocation"),
       XMLForest(e.empno AS "EmployeeId",
                 e.ename AS "Name",
                 e.job   AS "Job",
                 e.mgr   AS "Manager",
                 to_char(e.hiredate,'SYYYY-MM-DD') AS "HireDate",
                 e.sal  AS "Salary",
                 e.comm AS "Commission",
                 XMLForest(d.deptno AS "DeptNo",
                           d.dname  AS "DeptName",
                           d.loc    AS "Location") AS "Dept"))
    FROM emp e, dept d
    WHERE e.deptno = d.deptno;
```

前述の例では、XMLElement() 関数によって XML の Employee 要素が作成され、内側の XMLForest() 関数によって Employee 要素の子が作成されます。XMLElement() 内の XMLAttributes 句によって、生成された XML がビューの XML Schema に準拠するように、必須の XML の namespace およびスキーマの location 属性が作成されます。最も内側にある XMLForest() 関数によって、Employee 要素内にネストされた XML の department 要素が作成されます。

XML 生成関数は、XML Schema に基づかない XML インスタンスを生成するのみです。ただし、XMLType ビューの場合、要素と属性の名前が XML Schema 内の要素と属性の名前に一致するかぎり、Oracle によってこの XML が暗黙的に整形式で妥当な XML Schema に基づく文書に変換されます。XML インスタンス上で検証や抽出などの操作が実行された場合、生成された XML でエラーが発生します。

次の式について考えてみます。

```
to_char(e.hiredate,'SYYYY-MM-DD') AS "HireDate"
```

日付を文字列に変換するためのデフォルトのグローバリゼーション・サポート日付書式が異なる場合があるため、SQL の日付書式の日付を XML Schema の書式に変換するには、この式が必要です。

これで、ビューに対して問合せを行い、employee および department リレーショナル表から結果の XML を取得できます。

```
SQL> select value(p) as result from emp_simple_xml p where rownum < 2;

RESULT
--------------------------------------------------------------------------------
<Employee xmlns="http://www.oracle.com/emp_simple.xsd"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://www.oracle.com/emp_simple.xsd
                         http://www.oracle.com/emp_simple.xsd">
  <EmployeeId>7782</EmployeeId>
```

```
   <Name>CLARK</Name>
   <Job>MANAGER</Job>
   <Manager>7839</Manager>
   <HireDate>1981-06-09</HireDate>
   <Salary>2450</Salary>
  <Dept>
    <DeptNo>10</DeptNo>
    <DeptName>ACCOUNTING</DeptName>
    <Location>NEW YORK</Location>
  </Dept>
</Employee>
```

SQL/XML 関数での名前空間の使用

複数の名前空間を含む複雑な XML Schema が存在する場合、SQL 関数で提供される部分的にエスケープされたマッピングを使用して、適切な名前空間と接頭辞を持つ要素を作成する必要があります。

例 11-3 XMLType ビューでの名前空間接頭辞の使用

たとえば、次の SQL 問合せを実行するとします。

```
SELECT  XMLElement("ipo:Employee",
          XMLAttributes('http://www.oracle.com/emp_simple.xsd' AS "xmlns:ipo",
                        'http://www.oracle.com/emp_simple.xsd
                         http://www.oracle.com/emp_simple.xsd' AS "xmlns:xsi"),
            XMLForest(e.empno AS "ipo:EmployeeId",
                      e.ename AS "ipo:Name",
                      e.job   AS "ipo:Job"       ,
                      e.mgr   AS "ipo:Manager",
                      to_char(e.hiredate,'SYYYY-MM-DD') AS "ipo:HireDate",
                      e.sal  AS "ipo:Salary",
                      e.comm AS "ipo:Commission",
                      XMLForest(d.deptno AS "ipo:DeptNo",
                                d.dname  AS "ipo:DeptName",
                                d.loc    AS "ipo:Location") AS "ipo:Dept"))
  FROM emp e, dept d
  WHERE e.deptno = d.deptno;
```

これによって、正しい名前空間、接頭辞およびターゲット・スキーマの場所を持つ XML インスタンスが作成され、emp_simple_xml ビューの定義で問合せとして使用できます。この問合せによって次のようなインスタンスが作成されます。

```
<ipo:Employee xmlns:ipo="http://www.oracle.com/emp_simple.xsd"
   xmlns:xsi="http://www.oracle.com/emp_simple.xsd
              http://www.oracle.com/emp_simple.xsd">
  <ipo:EmployeeId>7782</ipo:EmployeeId>
  <ipo:Name>CLARK</ipo:Name>
  <ipo:Job>MANAGER</ipo:Job>
  <ipo:Manager>7839</ipo:Manager>
  <ipo:HireDate>1981-06-09</ipo:HireDate>
  <ipo:Salary>2450</ipo:Salary>
  <ipo:Dept>
    <ipo:DeptNo>10</ipo:DeptNo>
    <ipo:DeptName>ACCOUNTING</ipo:DeptName>
    <ipo:Location>NEW YORK</ipo:Location>
  </ipo:Dept>
</ipo:Employee>
```

XML Schema にターゲットの名前空間が存在しない場合、`xsi:noNamespaceSchemaLocation` 属性を使用してターゲットの名前空間を示すことができます。たとえば、`emp-noname.xsd` という場所に登録された次の XML Schema について考えてみます。

```
BEGIN
 dbms_xmlschema.deleteSchema('emp-noname.xsd', 4);
END;
/

BEGIN
 dbms_xmlschema.registerSchema('emp-noname.xsd',
  '<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns:xdb="http://xmlns.oracle.com/xdb">
  <xs:element name = "Employee" xdb:defaultTable="EMP37_TAB">
    <xs:complexType>
     <xs:sequence>
      <xs:element name = "EmployeeId" type = "xs:positiveInteger"/>
      <xs:element name = "FirstName" type = "xs:string"/>
      <xs:element name = "LastName" type = "xs:string"/>
      <xs:element name = "Salary" type = "xs:positiveInteger"/>
     </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>');
END;
/
```

次の CREATE OR REPLACE VIEW 文によって、この XML Schema に準拠するビューが作成されます。

```
CREATE OR REPLACE VIEW emp_xml OF XMLTYPE
     XMLSCHEMA "emp-noname.xsd" ELEMENT "Employee"
     WITH OBJECT ID (extract(sys_nc_rowinfo$,
                          '/Employee/EmployeeId/text()').getnumberval()) AS
     SELECT XMLElement("Employee",
       XMLAttributes('http://www.w3.org/2001/XMLSchema-instance' AS "xmlns:xsi",
                     'emp-noname.xsd' AS "xsi:noNamespaceSchemaLocation"),
        XMLForest(e.empno AS "EmployeeId",
                  e.ename AS "Name",
                  e.job   AS "Job" ,
                  e.mgr   AS "Manager",
                  to_char(e.hiredate,'SYYYY-MM-DD') AS "HireDate",
                  e.sal  AS "Salary",
                  e.comm AS "Commission",
                  XMLForest(d.deptno AS "DeptNo",
                            d.dname  AS "DeptName",
                            d.loc    AS "Location") AS "Dept"))
     FROM emp e, dept d
     WHERE e.deptno = d.deptno;
```

XMLAttributes 句によって、noNamespace スキーマの場所属性を含む XML 要素が作成されます。

オブジェクト型およびビューを使用した XMLType ビューの作成

厳密な型指定を持つ XML を含むリレーショナル・データをラップするには、オブジェクト・ビューを使用して、次の手順を実行します。

1. オブジェクト型を作成します。
2. XML 構造を含む XML Schema 文書を作成または生成し、SQL オブジェクト型および属性へのマッピングとともに登録します。第 5 章「XMLType の構造化されたマッピング」を参照してください。
3. XMLType ビューを作成し、XML Schema の URL およびルート要素名を指定します。基礎となるビュー問合せによって、まずオブジェクト・インスタンスが作成され、これらのインスタンスが XML に変換されます。この手順は、次の 2 つの手順で実行することもできます。
   1. オブジェクト・ビューを作成します。
   2. そのオブジェクト・ビューの XMLType ビューを作成します。

従業員と部門の正規のリレーショナル表、およびこのデータの XML ビューに基づく次の例について考えてみます。

- オブジェクト・ビューからの、XML Schema に基づく XMLType ビューの作成
- XMLType ビュー : ネストした従業員データを含む部門のリレーショナル・データの XML としてのラッピング

例 11-4 オブジェクト・ビューからの、XML Schema に基づく XMLType ビューの作成

1 つ目の例では、次の手順を実行して、ネストした部門情報を含む従業員のリレーショナル・データを XML としてラップします。

手順 1: オブジェクト型の作成

```
CREATE OR REPLACE TYPE dept_t AS OBJECT
      (
       DEPTNO        NUMBER(2),
       DNAME         VARCHAR2(14),
       LOC           VARCHAR2(13)
      );
/
CREATE OR REPLACE TYPE emp_t AS OBJECT
      (
       EMPNO         NUMBER(4),
       ENAME         VARCHAR2(10),
       JOB           VARCHAR2(9),
       MGR           NUMBER(4),
       HIREDATE      DATE,
       SAL           NUMBER(7,2),
       COMM          NUMBER(7,2),
       DEPT          DEPT_T
      );
/
```

手順 2: XML Schema emp.xsd の作成または生成

XML Schema は、手動で作成するか、または DBMS_XMLSchema パッケージを使用して既存のオブジェクト型から自動的に作成します。次に例を示します。

```
SELECT DBMS_XMLSchema.generateSchema('SCOTT','EMP_T') AS result FROM DUAL;
```

この問合せによって、従業員型の XML Schema が生成されます。このファンクションには、名前空間を追加するためなどの、様々な引数を指定できます。この XML Schema に追加の編集を行い、生成された様々なデフォルトのマッピングを変更できます。パッケージ内の generateSchemas() ファンクションによって、オブジェクト型とその属性によって参照される様々な SQL データベース・スキーマごとに、1 つの XML Schema のリストが生成されます。

手順 3: XML Schema emp.xsd の登録

XML Schema emp.xsd は、XML 要素および属性を、オブジェクト型の対応する属性にマップする方法も指定します。

```
BEGIN
 dbms_xmlschema.deleteSchema('http://www.oracle.com/emp.xsd', 4);
END;
/
BEGIN
 dbms_xmlschema.registerSchema('http://www.oracle.com/emp.xsd',
'<schema xmlns="http://www.w3.org/2001/XMLSchema"
 targetNamespace="http://www.oracle.com/emp.xsd" version="1.0"
 xmlns:xdb="http://xmlns.oracle.com/xdb"
 elementFormDefault="qualified">
 <element name = "Employee" xdb:SQLType="EMP_T" xdb:SQLSchema="SCOTT">
  <complexType>
   <sequence>
    <element name = "EmployeeId" type = "positiveInteger" xdb:SQLName="EMPNO"
                                                          xdb:SQLType="NUMBER"/>
    <element name = "Name" type = "string" xdb:SQLName="ENAME"
                                                          xdb:SQLType="VARCHAR2"/>
    <element name = "Job" type = "string" xdb:SQLName="JOB"
xdb:SQLType="VARCHAR2"/>
    <element name = "Manager" type = "positiveInteger" xdb:SQLName="MGR"
                                                          xdb:SQLType="NUMBER"/>
    <element name = "HireDate" type = "date" xdb:SQLName="HIREDATE"
                                                          xdb:SQLType="DATE"/>
    <element name = "Salary" type = "positiveInteger" xdb:SQLName="SAL"
                                                          xdb:SQLType="NUMBER"/>
    <element name = "Commission" type = "positiveInteger" xdb:SQLName="COMM"
                                                          xdb:SQLType="NUMBER"/>
    <element name = "Dept" xdb:SQLName="DEPT" xdb:SQLType="DEPT_T"
xdb:SQLSchema="SCOTT">
       <complexType>
          <sequence>
            <element name = "DeptNo" type = "positiveInteger" xdb:SQLName="DEPTNO"
                                                          xdb:SQLType="NUMBER"/>
            <element name = "DeptName" type = "string" xdb:SQLName="DNAME"
                                                          xdb:SQLType="VARCHAR2"/>
            <element name = "Location" type = "string" xdb:SQLName="LOC"
                                                          xdb:SQLType="VARCHAR2"/>
          </sequence>
       </complexType>
    </element>
   </sequence>
  </complexType>
 </element>
```

```
</schema>', TRUE, FALSE, FALSE);
END;
/
```

この文によって、XML Schema が次の目的の位置に登録されます。

```
"http://www.oracle.com/emp.xsd"
```

手順 4a: 1 ステップ・プロセスを使用した XMLType ビューの作成

1 ステップ・プロセスでは、次のとおり、リレーショナル表に XMLType ビューを作成します。

```
CREATE OR REPLACE VIEW emp_xml OF XMLTYPE
     XMLSCHEMA "http://www.oracle.com/emp.xsd" ELEMENT "Employee"
     WITH OBJECT ID (ExtractValue(sys_nc_rowinfo$, '/Employee/EmployeeId')) AS
     SELECT emp_t(e.empno, e.ename, e.job, e.mgr, e.hiredate, e.sal, e.comm,
                  dept_t(d.deptno, d.dname, d.loc))
     FROM emp e, dept d
     WHERE e.deptno = d.deptno;
```

この例では、ここで OBJECT ID 句に SQL 関数 extractValue() を使用します。これは、extractValue() が、XML Schema 情報を使用して適切な SQL データ型マッピング（この場合 SQL の NUMBER 型）を自動的に判断できるためです。

手順 4b: 最初にオブジェクト・ビューを作成する 2 ステップ・プロセスを使用した XMLType ビューの作成

2 ステップ・プロセスでは、次のとおり、まずオブジェクト・リレーショナル・ビューを作成し、次にそのオブジェクト・リレーショナル・ビューに XMLType ビューを作成します。

```
CREATE OR REPLACE VIEW emp_v OF emp_t WITH OBJECT ID (empno) AS
     SELECT emp_t(e.empno, e.ename, e.job, e.mgr, e.hiredate, e.sal, e.comm,
                  dept_t(d.deptno, d.dname, d.loc))
     FROM emp e, dept d
     WHERE e.deptno = d.deptno;

-- Create the employee XMLType view over the emp_v object view
CREATE OR REPLACE VIEW emp_xml OF XMLTYPE
     XMLSCHEMA "http://www.oracle.com/emp.xsd" ELEMENT "Employee"
     WITH OBJECT ID DEFAULT
 AS SELECT VALUE(p) FROM emp_v p;
```

例 11-5 XMLType ビュー : ネストした従業員データを含む部門のリレーショナル・データの XML としてのラッピング

2 つ目の例では、次の手順を実行して、ネストした従業員情報を含む部門のリレーショナル・データを XML としてラップします。

手順 1: オブジェクト型の作成

```
DROP TYPE emp_t FORCE;
DROP TYPE dept_t FORCE;
CREATE OR REPLACE TYPE emp_t AS OBJECT
      (
        EMPNO        NUMBER(4),
        ENAME        VARCHAR2(10),
        JOB          VARCHAR2(9),
        MGR          NUMBER(4),
        HIREDATE     DATE,
        SAL          NUMBER(7,2),
        COMM         NUMBER(7,2)
       );
/
CREATE OR REPLACE TYPE emplist_t AS TABLE OF emp_t;
/
CREATE OR REPLACE TYPE dept_t AS OBJECT
       (
        DEPTNO       NUMBER(2),
        DNAME        VARCHAR2(14),
        LOC          VARCHAR2(13),
        EMPS         EMPLIST_T
       );
/
```

手順 2: XML Schema dept.xsd の登録

既存の XML Schema を使用するか、または DBMS_XMLSchema.generateSchema(s) ファンクションを使用してオブジェクト型から XML Schema を生成します。

```
BEGIN
dbms_xmlschema.deleteSchema('http://www.oracle.com/dept.xsd', 4);
END;
/
BEGIN
dbms_xmlschema.registerSchema('http://www.oracle.com/dept.xsd',
'<schema xmlns="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.oracle.com/dept.xsd" version="1.0"
  xmlns:xdb="http://xmlns.oracle.com/xdb"
  elementFormDefault="qualified">
  <element name = "Department" xdb:SQLType="DEPT_T" xdb:SQLSchema="SCOTT">
```

```
    <complexType>
      <sequence>
        <element name = "DeptNo" type = "positiveInteger" xdb:SQLName="DEPTNO"
                                          xdb:SQLType="NUMBER"/>
        <element name = "DeptName" type = "string" xdb:SQLName="DNAME"
                                          xdb:SQLType="VARCHAR2"/>
        <element name = "Location" type = "string" xdb:SQLName="LOC"
                                          xdb:SQLType="VARCHAR2"/>
        <element name = "Employee" maxOccurs = "unbounded" xdb:SQLName = "EMPS"
                                          xdb:SQLType="EMPLIST_T"
xdb:SQLSchema="SCOTT">
         <complexType>
           <sequence>
             <element name = "EmployeeId" type = "positiveInteger"
                                          xdb:SQLName="EMPNO" xdb:SQLType="NUMBER"/>
             <element name = "Name" type = "string" xdb:SQLName="ENAME"
                                          xdb:SQLType="VARCHAR2"/>
             <element name = "Job" type = "string" xdb:SQLName="JOB"
                                          xdb:SQLType="VARCHAR2"/>
             <element name = "Manager" type = "positiveInteger" xdb:SQLName="MGR"
                                          xdb:SQLType="NUMBER"/>
             <element name = "HireDate" type = "date" xdb:SQLName="HIREDATE"
                                          xdb:SQLType="DATE"/>
             <element name = "Salary" type = "positiveInteger" xdb:SQLName="SAL"
                                          xdb:SQLType="NUMBER"/>
             <element name = "Commission" type = "positiveInteger"
                                          xdb:SQLName="COMM" xdb:SQLType="NUMBER"/>
           </sequence>
         </complexType>
        </element>
      </sequence>
    </complexType>
  </element>
</schema>', TRUE, FALSE, FALSE);
END;
/
```

手順 3a: リレーショナル表の XMLType ビューの作成

次のとおり、部門オブジェクト型から dept_xml XMLType ビューを作成します。

```
CREATE OR REPLACE VIEW dept_xml OF XMLTYPE
    XMLSCHEMA "http://www.oracle.com/dept.xsd" ELEMENT "Department"
    WITH OBJECT ID (EXTRACTVALUE(sys_nc_rowinfo$, '/Department/DeptNo')) AS
       SELECT dept_t(d.deptno, d.dname, d.loc,
              cast(multiset(
                 SELECT emp_t(e.empno, e.ename, e.job, e.mgr, e.hiredate,
                        e.sal,e.comm) FROM emp e
                        WHERE e.deptno = d.deptno)
                        AS emplist_t))
                 FROM dept d;
```

手順 3b: SQL 関数を使用したリレーショナル表の XMLType ビューの作成

オブジェクト型の定義を使用せずに、リレーショナル表から dept_xml XMLType ビューを作成することもできます。

```
CREATE OR REPLACE VIEW dept_xml OF XMLTYPE
    XMLSCHEMA "http://www.oracle.com/dept.xsd" ELEMENT "Department"
    WITH OBJECT ID (EXTRACT(sys_nc_rowinfo$, '/Department/DeptNo').getNumberVal())
AS
       SELECT  XMLElement("Department",
            XMLAttributes( 'http://www.oracle.com/emp.xsd' AS "xmlns" ,
                     'http://www.w3.org/2001/XMLSchema-instance' AS "xmlns:xsi",
                     'http://www.oracle.com/dept.xsd
                      http://www.oracle.com/dept.xsd' AS "xsi:schemaLocation"),
             XMLForest(d.deptno "DeptNo",
                       d.dname "DeptName",
                       d.loc "Location"),
             (SELECT XMLAGG(XMLElement("Employee",
                              XMLForest(e.empno "EmployeeId",
                                   e.ename "Name",
                                   e.job "Job",
                                   e.mgr "Manager",
                                   to_char(e.hiredate,'SYYYY-MM-DD')
                                        "Hiredate"),
                                   e.sal "Salary",
                                   e.comm "Commission"))
                 FROM emp e
                 WHERE e.deptno = d.deptno))
       FROM dept d;
```

注意： XML Schema および要素情報は、ビュー・レベルで指定する必要があります。これは、SELECT リストでは、基礎となる表から別の XML Schema の XML が任意で作成される可能性があるためです。

XMLType 表からの XMLType ビューの作成

いくつかの述語を使用して、XML を変換したり、戻される行を制限することによって、XMLType 表の XMLType ビューを作成できます。

例 11-6 XMLType 表の行の制限による XMLType ビューの作成

次の例では、基礎となる XMLType 表から戻される行を制限して、XMLType ビューを作成します。前述の項に示す XML Schema dept.xsd を使用して、基礎となる表を作成します。

```
DROP TABLE dept_xml_tab;

CREATE TABLE dept_xml_tab OF XMLTYPE
    XMLSCHEMA "http://www.oracle.com/dept.xsd" ELEMENT "Department"
    nested table xmldata."EMPS" store as dept_xml_tab_tab1;

CREATE OR REPLACE VIEW dallas_dept_view OF XMLTYPE
    XMLSCHEMA "http://www.oracle.com/dept.xsd" ELEMENT "Department"
        AS SELECT VALUE(p) FROM dept_xml_tab p
           WHERE Extractvalue(value(p), '/Department/Location') = 'DALLAS';
```

dallas_dept_view によって、XMLType 表の行が、所在地がダラスである部門に制限されます。

例 11-7 XMLType 表の変換による XMLType ビューの作成

スタイルシートを使用して XML データを変換することによって、XMLType ビューを作成できます。たとえば、XMLType 表 po_tab を作成する場合を考えてみます。xmltransform() の例は、6-6 ページの「例 6-1XSLT スタイルシートを取り出すための、XMLTransform() および DBUriType を使用した XMLType インスタンスの変換」を参照してください。

```
DROP TABLE po_tab;

CREATE TABLE po_tab OF xmltype xmlschema "ipo.xsd" element
    "PurchaseOrder";
```

これによって、次のとおり表のビューを作成できます。

```
CREATE OR REPLACE VIEW HR_PO_tab OF xmltype xmlschema "hrpo.xsd" element
   "PurchaseOrder"
    WITH OBJECT ID DEFAULT
    AS SELECT
        xmltransform(value(p),xdburitype('/home/SCOTT/xsl/po2.xsl').getxml())
    FROM po_tab p;
```

REF() を使用した XMLType ビュー・オブジェクトの参照

XMLType ビュー・オブジェクトは、REF() 構文を使用して参照できます。

```
SELECT REF(p) FROM dept_xml p;
```

XMLType ビュー参照の REF() は、次のいずれかのオブジェクト ID に基づきます。

- システム生成 OID: XMLType 表のビューまたはオブジェクト・ビュー
- 主キー・ベースの OID: OBJECT ID 式を持つビュー

これらの REF は、OCI オブジェクト・キャッシュ内の OCIXMLType インスタンスをフェッチするために使用できます。また、SQL 問合せ内でも使用できます。これらの REF は、オブジェクト・ビューへの REF と同様に動作します。

XMLType ビューでのデータ操作言語（DML）

本来、XMLType ビューは更新可能でありません。そのため、すべてのデータ操作（DML）を処理するには、INSTEAD OF トリガーを作成する必要があります。基礎となるビュー問合せを分析することによって、ビューが暗黙的に更新可能な場合を識別できます。

例 11-8 暗黙的に更新可能なビューの識別

たとえば、XMLType ビュー問合せが、本来更新可能であるオブジェクト・ビューまたはオブジェクト・コンストラクタに基づいているとします。

```
DROP TYPE dept_t force;
CREATE OR REPLACE TYPE dept_t AS OBJECT
      (
       DEPTNO        NUMBER(2),
       DNAME         VARCHAR2(14),
       LOC           VARCHAR2(13)
      );
/

BEGIN
dbms_xmlschema.deleteSchema('http://www.oracle.com/dept.xsd', 4);
END;
```

```
/
BEGIN
dbms_xmlschema.registerSchema('http://www.oracle.com/dept.xsd',
'<schema xmlns="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.oracle.com/dept.xsd" version="1.0"
  xmlns:xdb="http://xmlns.oracle.com/xdb"
  elementFormDefault="qualified">
  <element name = "Department" xdb:SQLType="DEPT_T" xdb:SQLSchema="SCOTT">
    <complexType>
      <sequence>
        <element name = "DeptNo" type = "positiveInteger" xdb:SQLName="DEPTNO"
                                          xdb:SQLType="NUMBER"/>
        <element name = "DeptName" type = "string" xdb:SQLName="DNAME"
                                          xdb:SQLType="VARCHAR2"/>
        <element name = "Location" type = "string" xdb:SQLName="LOC"
                                          xdb:SQLType="VARCHAR2"/>
      </sequence>
    </complexType>
  </element>
</schema>', TRUE, FALSE, FALSE);
END;
/

CREATE OR REPLACE VIEW dept_xml of xmltype
xmlschema "http://www.oracle.com/dept.xsd" element "Department"
with object id (sys_nc_rowinfo$.extract('/Department/DeptNo').getnumberval()) as
select dept_t(d.deptno, d.dname, d.loc) from dept d;

INSERT INTO dept_xml VALUES (XMLType.createXML(
'<Department xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://www.oracle.com/dept.xsd">
 <DeptNo>50</DeptNo>
 <DeptName>EDP</DeptName>
 <Location>NEW YORK</Location>
</Department>'));

UPDATE dept_xml d
     SET d.sys_nc_rowinfo$ = updateXML(d.sys_nc_rowinfo$,
'/Department/DeptNo/text()', 60)
     WHERE existsNode(d.sys_nc_rowinfo$, '/Department[DeptNo=50]') = 1;
```

XMLType ビューでのクエリー・リライト

クエリー・リライトに関しては、XMLType ビューは通常の XMLType 表列と同じです。したがって、ビュー列に対する extract() または existsNode() 操作は、パフォーマンスを向上させるために、基礎となるリレーショナル・アクセスにリライトされます。

今回のリリースでは、これらの SQL/XML ビューに対する XPath の述語はリライトされません。そのため、これらのビューに対する問合せで extract や existsNode などの述語が使用されている場合、ビューのすべての行に対して問合せが機能的に評価されます。ビューの問合せ実行性が重要である場合、SQL/XML 関数のかわりにオブジェクト・リレーショナル形式の使用を検討してください。

XML Schema に基づくビューでのクエリー・リライト

たとえば、次の例について考えてみます。

例 11-9 XMLType ビューでのクエリー・リライト : XML Schema に基づくビューでのクエリー・リライト

```
XCREATE OR REPLACE VIEW dept_ov OF dept_t
     WITH OBJECT ID (deptno) as
     SELECT d.deptno, d.dname, d.loc, cast(multiset(
        SELECT emp_t(e.empno, e.ename, e.job, e.mgr, e.hiredate, e.sal, e.comm)
               FROM emp e
               WHERE e.deptno = d.deptno)
               AS emplist_t)
        FROM dept d;

CREATE OR REPLACE VIEW dept_xml OF XMLTYPE
     WITH OBJECT ID (EXTRACT(sys_nc_rowinfo$, '/ROW/DEPTNO').getNumberVal()) AS
     SELECT sys_xmlgen(value(p)) FROM dept_ov p;
```

給与が $200,000 を超える従業員が 1 人以上いる部門の番号を選択する問合せを発行します。

```
SELECT EXTRACTVALUE(value(x),'/ROW/DEPTNO')
  FROM dept_xml x
  WHERE EXISTSNODE(value(x), '/ROW/EMPS/EMP_T[SAL > 200]') = 1;
```

この問合せは、次のようにリライトされます。

```
ELECT d.deptno
  FROM dept d
  WHERE EXISTS (SELECT NULL FROM emp e WHERE e.deptno = d.deptno
                AND e.sal > 200);
```

XML Schema に基づかない XMLType ビューでのクエリー・リライト

次の例について考えてみます。

例 11-10 XML Schema に基づかないビューでのクエリー・リライト

XML Schema に基づかない XMLType ビューは、既存のリレーショナル表、リレーショナル・ビュー、オブジェクト・リレーショナル表およびオブジェクト・リレーショナル・ビューに対して作成できます。これによって、ユーザーは、基礎となるデータの XML ビューを使用できます。

既存のリレーショナル・データは、適切な型を作成し、最上位で SYS_XMLGEN を実行することによって XMLType ビューに変換できます。たとえば、emp 表内のデータを次のように公開できます。

```
CREATE TYPE Emp_t AS OBJECT  (
  EMPNO NUMBER(4),
  ENAME VARCHAR2(10),
  JOB   VARCHAR2(9),
  MGR   NUMBER(4),
  HIREDATE DATE,
  SAL   NUMBER(7,2),
  COMM  NUMBER(7,2));

CREATE VIEW employee_xml OF XMLTYPE
  WITH OBJECT  ID
        (SYS_NC_ROWINFO$.EXTRACT('/ROW/EMPNO/text()').getnumberval()) AS
    SELECT SYS_XMLGEN(
        emp_t(e.empno, e.ename, e.job, e.mgr, e.hiredate, e.sal, e.comm))
    FROM emp e;
```

XML Schema に基づかないビューの主なメリットは、DDL を追加せずに、既存のオブジェクト・ビューを簡単に XMLType ビューに変換できることです。たとえば、次の定義を持つオブジェクト・ビュー employee_ov を含むデータベースについて考えてみます。

```
CREATE VIEW employee_ov OF EMP_T
WITH OBJECT ID (empno) AS
SELECT  emp_t(e.empno, e.ename, e.job, e.mgr, e.hiredate, e.sal, e.comm)
    FROM emp e;

-- Creating a non-schema-based XMLType views can be achieved by simply
-- calling SYS_XMLGEN over the top-level object column. No additional
-- types need to be created.

CREATE OR REPLACE VIEW employee_ov_xml OF XMLTYPE
 WITH OBJECT  ID
  (SYS_NC_ROWINFO$.EXTRACT('/ROW/EMPNO/text()').getnumberval()) AS
       SELECT SYS_XMLGEN(value(x)) from employee_ov x;
```

```
-- Certain kinds of queries on SYS_XMLGEN views are rewritten to
```

この問合せは、オブジェクト属性に直接アクセスするためにリライトされます。existsNode()、extractValue() および extract() による単純な XPath 検索は、リライトの対象になります。クエリー・リライトの詳細は、5-47 ページの「XML Schema に基づく構造化記憶域でのクエリー・リライト」を参照してください。

> **注意：** クエリー・リライトは、SYS_XMLGEN でのみ実行されます。その他の関数に基づくビューに対する問合せはリライトされません。

たとえば、次のような問合せを発行するとします。

```
SELECT EXTRACT(VALUE(x), '/ROW/EMPNO') FROM employee_ov_xml x
WHERE  EXTRACTVALUE(value(x), '/ROW/ENAME') = 'SMITH';
```

この問合せは、次のようにリライトされます。

```
SELECT SYS_XMLGEN(empno)
FROM emp e
WHERE e.ename = 'SMITH';
```

XML Schema に基づかない XMLType ビューのサポート

SYS_XMLGEN を使用した XML Schema に基づかない XMLType ビューに対する問合せは、SYS_XMLGEN への引数を操作するためにリライトされる可能性があります。

- SYS_XMLGEN にはどのようなオブジェクト型の引数でも指定でき、それらの引数はオブジェクト・ビューの列になることが可能です。SYS_XMLGEN の引数のオブジェクト型に埋込みコレクションが含まれる場合は、問合せで単一の XML 演算子（existsnode()、extract()、extractvalue() など）が使用されている場合のみ、リライトが行われます。複数の演算子が使用されている場合、リライトは行われません。
- SYS_XMLGEN は、追加のパラメータ（XMLFormat など）を取ることはできません。XML Schema に基づかない XMLType ビューの主なメリットは、データを新しい表に移行する必要がないため、データを XML として直接表示したり、XML として問い合せることができることです。デメリットは、XML への豊富なマッピング・セットが XML Schema を使用して最も適切に取得され、XML へのオブジェクト型の正規マッピングのみが、XML Schema に基づかない XMLType ビューでサポートされることです。

XPath のリライト方法（XML Schema に基づかない XMLType ビュー用）

次の XMLType ビューの定義について考えてみます。この項の実行例では、XMLType ビューの定義を使用します。この項では、XML Schema に基づかない XMLType ビュー用に XPath をリライトする方法を説明します。

```
create type emp_t as object (
  EMPNO NUMBER(4), ename VARCHAR2(10), job VARCHAR2(9), mgr NUMBER(4),
  HIREDATE DATE);

create type emp_list is varray(100) of emp_t;

create or replace type dept_t as object (
  "@DEPTNO"   NUMBER(2), DeptNAME VARCHAR2(14),  LOC VARCHAR2(13),
  employees emp_list);

create view dept_ov of dept_t with object id (deptname) as
       select deptno, dname, loc, CAST( MULTISET(
              select emp_t(empno, ename, job, mgr, hiredate)
              from emp e where e.deptno = d.deptno) AS emp_list)
       from dept d;

create view dept_xv of xmltype
     with object id(SYS_NC_ROWINFO$.extract('/ROW/@DEPTNO').getnumberval()) as
        select  SYS_XMLGEN(VALUE(p)) FROM dept_ov p ;
```

XML Schema に基づかない XMLType ビュー : 単純な XPath のマッピング

単純な XPath のリライトは、XPath 式に対応する属性へのアクセスを伴います。つまり、'/ROW/DEPTNAME'、'/ROW/@DEPTNO' および '/ROW/EMPLOYEES' という式のリライトは、それぞれ dname、deptno および CAST(MULTISET(select emp_t(empno, ...) from emp e where ...) AS emplist) にマップされます。

たとえば、部門ビュー dept_xv から部門番号 dno を抽出する問合せについて考えてみます。

```
SELECT extractvalue(value(p),'/ROW/@DEPTNO') dno from dept_xv p ;
```

この問合せは、次の問合せにリライトされます。

```
SELECT SYS_ALIAS_1.DNO "DEPTNO" FROM DEPT SYS_ALIAS_1;
```

XML Schema に基づかない XMLType ビュー : スカラー・ノードのマッピング

text() 演算子を含む Xpath 式は、基礎となるリレーショナル列にリライトされる場合があります。たとえば、部門ビュー dept_xv から部門名 dname を抽出する問合せについて考えてみます。

```
SELECT extract(value(p),'/ROW/DEPTNAME/text()') dname from dept_xv p ;
```

この問合せは、次の問合せにリライトされます。

```
SELECT SYS_ALIAS_1.DNAME "DNAME" FROM DEPT SYS_ALIAS_1;
```

XML Schema に基づかない XMLType ビュー : 述語のマッピング

述語は、SQL 述語式にマップされます。たとえば、ACCOUNTING という部門の数を取得する問合せについて考えてみます。

```
select count(*)  from dept_xv e where existsnode(value(e),
'/ROW[DEPTNAME="ACCOUNTING"]')  = 1;
```

この問合せは、次の問合せにリライトされます。

```
SELECT COUNT(*) "COUNT(*)" FROM DEPT "SYS_ALIAS_1"
WHERE "SYS_ALIAS_1"."DNAME"='ACCOUNTING' AND DEPT_T(?.) IS NOT NULL;
```

XML Schema に基づかない XMLType ビュー : 単純なコレクション全検索

単純なコレクション全検索は、XMLAGG() を使用してリライトされ、コレクションの要素を集計します。部門ビューからすべての従業員番号を抽出する次の問合せについて考えてみます。各部門の従業員のコレクション要素を反復処理する必要があるため、XMLAGG() が追加され、最終結果が集計されます。

```
SELECT extract(value(e), '/ROW/EMPLOYEES/EMP_T/EMPNO') empno from  dept_xv e;
```

この問合せは、次のように XMLAGG() を使用してリライトされます。

```
SELECT (SELECT XMLAGG(SYS_XMLGEN(TO_CHAR(EMPNO))) "VALUE(P)"
    FROM EMP E WHERE EMPNO IS NOT NULL AND EMP_T(?) IS NOT NULL
    AND E.DEPTNO = "SYS_ALIAS1".DEPTNO) "EMPNO"
  FROM DEPT  "SYS_ALIAS1";
```

XML Schema に基づかない XMLType ビュー : コレクション索引

XPath 式でコレクションの特定の索引にアクセスすることもできます。たとえば、'/ROW/EMPLOYEES/EMP_T[1]/EMPNO' が、部門ビューの従業員のの最初のコレクション要素を取得し、この最初の要素から従業員番号を取得するようにリライトされます。

XML Schema に基づかない XMLType ビュー : コレクションの述語

コレクションを全検索する XPath 式には、述語が含まれる場合があります。たとえば、extract(value(p), '/ROW[EMPLOYEES/EMP_T/EMPNO > 7900]') という式には、比較述語（> 7900）が含まれます。XPath 1.0 では、コレクションの任意の項目が述語を満たす場合、このような XPath は条件を満たすと定義されています。つまり、この式は、コレク

ション内に 7900 より大きい従業員番号を持つ従業員が 1 人以上いる従業員コレクションを持つすべての行を取得するようにリライトされます。EXISTS() を使用した確認によって、該当する行のリストが取得されます。

XML Schema に基づく XML の非定型生成

前述の例では、CREATE VIEW 文で XML Schema の URL および要素名を指定し、基礎となるビュー問合せによって XML Schema に基づかない XMLType を作成しました。ただし、ビューを作成せずに、XML Schema に基づく XML を作成する必要がある場合もあります。

これを行うには、次の XML 生成関数を使用してオプションで XML Schema の URL および要素名を受け入れます。

- createXML()
- SYS_XMLGEN()
- SYS_XMLAGG()

> **参照：** 第 10 章「データベースからの XML データの生成」を参照してください。

XML Schema 情報を指定すると、結果として戻され XML は XML Schema に基づきます。

```
SELECT XMLTYPE.createXML(dept_t(d.deptno, d.dname, d.loc,
    CAST(MULTISET(SELECT emp_t(e.empno, e.ename, e.job, e.mgr,
                  e.hiredate, e.sal, e.comm)
    FROM emp e WHERE e.deptno = d.deptno) AS emplist_t),
              'http://www.oracle.com/dept.xsd', 'Department')
               FROM dept d;
```

ユーザーが指定する情報の検証

XML Schema を登録する前に、オプションの Oracle XML DB 属性を指定できます。この場合、Oracle によって追加情報が検証され、Oracle XML DB 属性に指定した値が他の XML Schema 宣言と互換性があるかどうかが確認されます。この形式の XML Schema 登録は、通常、XMLType ビューを使用して既存のデータをラップするときに行われます。

> **参照：** このプロセスの詳細は、第 5 章「XMLType の構造化されたマッピング」を参照してください。

DBMS_XMLSchema の generateSchema() ファンクションおよび generateSchemas() ファンクションを使用して、指定したオブジェクト型に対するデフォルトの XML マッピングを生成できます。生成された XML Schema 文書には、SQLType、SQLSchema などの属性が指定されています。その後、これらの XML Schema 文書を登録すると、次の検証フォームが生成されます。

- **simpleType に基づく属性または要素に対する SQLType:** これは、対応する XMLType と互換性があります。たとえば、XML 文字列のデータ型は、VARCHAR2 またはラージ・オブジェクト（LOB）にのみマップされます。
- **complexType に基づく要素に対して指定された SQLType:** これは、LOB または complexType の宣言と互換性がある構造を持つオブジェクト型のいずれかです。オブジェクト型には、正しいデータ型を持つ正しい数の属性が含まれます。

12

URL を介したデータの作成およびアクセス

この章では、データベース内に URL を生成および格納して、その URL が指すデータを取り出す方法を説明します。データベース内に格納されたリレーショナル・データへの URL である DBUri の概要も説明します。また、Oracle XML DB Repository の階層に格納されたデータへの参照を作成および格納する方法も説明します。

この章の内容は次のとおりです。

URL および URI での Oracle9*i* データベースの動作

インターネット・アプリケーション、特にインターネット・ベースの XML アプリケーションを開発する場合、URL または URI を使用して、ネットワーク上のデータの参照が必要な場合があります。

- Uniform Resource Locator（URL）は、ドキュメント全体、またはドキュメント内の特定の部分を参照します。
- Uniform Resource Identifier（URI）は、URL の一般的な形式です。URI は、URL と同一にできます。または、アンカーのかわりに追加の表記法を使用して、単一の場所ではなくドキュメントの特定の部分を識別できます。

> **注意：** URI の方がより一般的な用語であるため、この章では、URI を使用して説明します。ただし、URI の詳細は URL にも当てはまります。URI ではなく Uri を使用している型名もあります。この情報の大部分は SQL および PL/SQL に基づくため、通常、これらの名前の大 / 小文字は区別されません。Web サイト上の実際のファイル名や Java API 名を表すときのみ大 / 小文字を正確に指定する必要があります。

Oracle9*i* データベースでは、データベース内で様々なパスを表すことができます。各パスは、`URIType` という一般的な型から導出されている異なるオブジェクト型に対応しています。

- **`HTTPUriType`:** http:// で始まる URL を表します。この型を使用すると、Web ページへのリンクを表すオブジェクトを作成し、オブジェクト・メソッドをコールすることによって、これらの Web ページを取得できます。
- **`DBUriType`:** データベース内の行セット、単一の行または単一の列を指す URI を表します。この型を使用すると、表データへのリンクを表すオブジェクトを作成し、オブジェクト・メソッドをコールすることによって、このデータを取得できます。
- **`XDBUriType`:** データベース内の Oracle XML DB Repository に格納された XML 文書を指す URI を表します。これらの文書やその他のデータを、リソースといいます。この型を使用すると、リソースへのリンクを表すオブジェクトを作成し、オブジェクト・メソッドをコールすることによって、リソース全体または一部を取得できます。

HTTP を介したデータのアクセスおよび処理

Oracle XML DB Repository に格納されたリソースは、Oracle XML DB の HTTP Server を使用して取り出すこともできます。また、Oracle9*i* データベースには、HTTP の URL を介して表データを使用可能にするサーブレットも含まれています。データは、プレーン・テキスト、HTML または XML として戻されます。

Web 対応のすべてのクライアントまたはアプリケーションは、SQL プログラミングや特別なデータベース API がなくてもこのデータを使用できます。データを取り出すには、Web ページ内でそのデータにリンクさせるか、または Java、PL/SQL または Perl の、HTTP を認識する API を介してそのデータを要求します。このデータは、通常の Web ブラウザや、スプレッドシートなどの XML を認識するアプリケーションを含むすべての種類のアプリケーションで表示または処理できます。サーブレットは、XML コンテンツおよび XML 以外のコンテンツの生成をサポートしており、XSLT スタイルシートを使用して結果を変換します。

URIType を使用した列の作成およびデータの格納

`URIType` またはその子である型を使用してデータベース列を作成するか、あるいは各 URI または URL のテキストのみを格納し、必要に応じてオブジェクト型を作成できます。データベースにサブタイプの組合せを格納する場合、同じ列内に様々なサブタイプを格納できる `URIType` 列を定義できます。

これらの機能では、オブジェクト型やメソッドなどのオブジェクト指向プログラミング機能が使用されるため、Oracle が提供する型から継承したユーザー独自の型も導出できます。新しい型を導出すると、特別な方法を使用してデータを取り出したり、プログラムに戻す前にそのデータを変換またはフィルタ処理できます。

> **参照：** 26-76 ページの「8.0 XML DB Demo: DBUri サーブレットを使用したコンテンツへのアクセスおよび XSL を使用したコンテンツの変換」を参照してください。

URIFactory パッケージ

データベースに URI テキストのみを格納している場合、`URIFactory` パッケージを使用して各 URI を適切なサブタイプのオブジェクトに変換できます。`URIFactory` パッケージは、指定された文字列によって表される URI の種類を確認することによって、適切な型のインスタンスを作成します。たとえば、`http://` で始まるすべての URI は、HTTP の URL とみなされます。`URIFactory` パッケージにそのような URI 文字列が渡されると、このパッケージは `HTTPUriType` オブジェクトのインスタンスを戻します。

> **参照：** 12-26 ページの「URIFactory パッケージを使用した URIType の新しいサブタイプの登録」を参照してください。

URI および URL に関するその他のソース

この章で説明する機能を使用する前に、様々な種類の URI の表記法を理解しておく必要があります。

参照：

- HTTP の URL の表記法については、`http://www.w3.org/2002/ws/Activity.html` を参照してください。
- XML の XPath の表記法については、`http://www.w3.org/TR/xpath` を参照してください。
- XML の XPointer の表記法については、`http://www.w3.org/TR/xptr/` を参照してください。
- MIME タイプの詳細は、`http://xml.coverpages.org/xmlMediaMIME.html` を参照してください。

URI の概念

この項では、URI の概念を説明します。

URI の概要

Uniform Resource Identifier（URI）は、URL の一般的な形式です。URL と同様に、URI はすべてのドキュメントを参照でき、ドキュメントの特定の部分も参照できます。URI にはドキュメントの関連する部分を指定する強力なメカニズムが備わっているため、URL より一般的です。URI は、次の 2 つの部分で構成されます。

- 通常の URL と同じ表記法を使用してドキュメントを識別する **URL 部分**。
- ドキュメント内のフラグメントを識別する**フラグメント部分**。フラグメントの表記法は、ドキュメント・タイプによって異なります。HTML ドキュメントの場合は *#anchor_name* という形式です。XML 文書の場合は、XPath 表記法が使用されます。

フラグメントは、次の項に示す例の # 以降の部分です。

注意： 今回のリリースでは、`XDBUriType` および `HTTPUriType` のみが URI フラグメントをサポートします。`DBUriType` は URI フラグメントをサポートしません。

XML 文書のビューから URL パスを作成する方法

図 12-1 に、データベース内のリレーショナル表 EMP に格納されている XML データのビュー、および XML 文書の要素にマップされているデータ列を示します。このマッピングは、**XML のビジュアル化**ともいいます。結果として作成される URL パスは、XML 文書のビューから導出できます。

URI の典型的な形式は次のとおりです。

- **HTML の場合 :** `http://www.url.com/document1#Anchor`

  この `Anchor` は、ドキュメント内に指定されたアンカーです。

- **XML の場合 :** `http://www.xml.com/xml_doc#//po/cust/custname`

  この各部分は、次の内容を表します。

  – `#` の前の部分は、文書の場所を示します。

  – `#` の後の部分は、文書のフラグメントを示します。この部分は W3C の XPointer 勧告で定義されています。

異なるプロトコルを使用した URIType オブジェクトによるデータの取出し

Oracle9*i* データベースでは、URI を表すオブジェクトを格納および取り出すための、新しいデータ型が導入されています。詳細は、12-6 ページの「URI 参照を格納するための URIType」を参照してください。各データ型は、HTTP などの異なるプロトコルを使用してデータを取り出します。

Oracle9*i* データベースでは、データベース表の行と列への参照を表す新しい形式の URI が導入されています。

DBUri および XDBUri を使用するメリット

`DBUri` および `XDBUri` を使用する主なメリットは次のとおりです。

- データベースから生成された Web ページ内でスタイルシートを参照できます。Oracle が提供する `DBMS_METADATA` パッケージは、`DBUri` を使用して XSLT スタイルシートを参照します。`XDBUri` を使用すると、Oracle XML DB Repository に格納された XSLT スタイルシートも参照できます。
- データベースに格納された HTML、イメージおよびその他のデータを参照できます。URL を使用して、表またはリポジトリの階層フォルダに格納されたデータを指すことができます。
- Web サーバーをバイパスすることによって、パフォーマンスが向上します。XML 文書内に URL がある場合、次のいずれかの方法でそれをデータベースへの参照に置換できます。

  – サーブレットの使用

  - DBUri または XDBUri を使用した結果の取得

    DBUri または XDBUri を使用すると、Web サーバーを介さずにデータベースと直接対話できるため、パフォーマンスが向上します。

- データベース内の XML 文書へ SQL を使用せずにアクセスできます。データベース内に格納されている XML 文書にアクセスするための SQL の知識が必要ありません。DBUri を使用すると、SQL を使用せずにデータベースから XML 文書にアクセスできます。

  Oracle XML DB Repository 内のファイルやリソースは表に格納されているため、XDBUri を使用するか、または DBUri を介して表の隠喩を使用することによってそれらのファイルやリソースにアクセスできます。

URI 参照を格納するための URIType

Universal Resource Identifier（URI）は、Web ページなどの Web 上のリソースを識別します。Oracle9*i* データベースでは、外部 URI 参照および内部 URI 参照を格納し、それらにアクセスするための次の URIType が提供されています。

- **DBUriType:** データベース内のリレーショナル・データへの参照を格納します。
- **HTTPUriType:** リモート・ページにアクセスするための HTTP プロトコルを実装します。外部 Web ページまたはファイルへの URL を格納します。また、Hyper Text Transfer Protocol（HTTP）プロトコルを使用してこれらのファイルにアクセスします。
- **XDBUriType:** Oracle XML DB Repository 内のリソースへの参照を格納します。

これらのデータ型は、オブジェクトによって指されるオブジェクトまたはページにアクセスするために使用できるメンバー関数を持つオブジェクト型です。URIType を使用して次の操作を実行できます。

- データベース内外のデータを指す表の列の作成
- URIType の提供する関数を使用したデータベース列の問合せ

これらは、継承階層によって関連付けられます。URIType は抽象型で、DBUriType、HTTPUriType および XDBUriType は URIType のサブタイプです。CLOB またはその他の列に格納されたデータを参照し、URL として外部に公開できます。Oracle9*i* データベースは、これらの URL を解析する Oracle Servlet Engine でインストールおよび実行できる標準サーブレットを提供します。

URIType を使用するメリット

Oracle は、URL 参照のフェッチ用に、PL/SQL パッケージの UTL_HTTP および Java クラスの java.net.URL を提供しています。SQL で新しい URIType データ型を定義するメリットは次のとおりです。

- **列に対する XML 文書のマッピングが改善されます。**XML 文書をオブジェクト・リレーショナル列に分解するには、URIRef のサポートが必要です。そのため、文書で指定された URIRef は、データベース内の URL 列にマップできます。
- **サーバーの内外に格納されたデータに統一アクセスできます。**URIRef を使用して HTTP/DB URL へのポインタを格納できるため、格納されている場所に関係なくデータに統一アクセスできます。これによって、データの場所を考慮せずに問合せおよび索引を作成できます。

参照： 12-23 ページの「URIType オブジェクトの使用」を参照してください。

URIType 関数

URIType 抽象型は、すべてのサブタイプで使用可能な様々な関数をサポートします。表 12-1 に、URIType メンバー関数を示します。

表 12-1 URIType メンバー関数

URIType メンバー関数	説明
getClob()	URL が指す値を CLOB 値として戻します。文字コードは、データベース・キャラクタ・セットの文字コードになります。
getUrl()	URIType に格納されている URL を戻します。属性「url」を直接使用せず、かわりにこの関数を使用してください。この関数をサブタイプでオーバーライドし、正しい URL を取得できます。たとえば、HTTPUriType は、URL のみを格納して接頭辞「http://」を格納しません。そのため、getUrl() は接頭辞を付加して値を戻します。
getExternalUrl()	URL 仕様に準拠するために、URL の文字をエスケープするエスケープ・メカニズムをコールすることを除き、getUrl と同じです。たとえば、空白はエスケープ値 %20 に変換されます。 12-2 ページの「URL および URI での Oracle9i データベースの動作」を参照してください。
getContentType()	URL の MIME 情報を戻します。URIType の場合、これは抽象関数です。

表 12-1 URIType メンバー関数（続き）

URIType メンバー関数	説明
getXML()	指定された URI に対応する XMLType オブジェクトを戻します。これは、getClob/getBlob 以外の操作を実行する必要があるアプリケーションが、これらの操作の実行に XMLType メソッドを使用できるように提供されています。 URI が妥当な XML 文書を指していない場合、例外が発生します。
getBlob()	URL が指す BLOB 値を戻します。文字変換は実行されず、文字コードは URL が指す文字コードになります。これは、バイナリ・データのフェッチにも使用できます。
createUri(uri IN VARCHAR2)	URIType を作成します。これは実際は URIType ではなく、URI のサブタイプを作成するために使用されます。

HTTPUriType 関数

HTTPUriType を使用すると、HTTP プロトコルを介してアクセス可能なデータへの参照を格納できます。HTTPUriType は、UTL_HTTP パッケージを使用してデータをフェッチするため、パッケージのセッション設定を使用して、このメカニズムを使用する HTTP フェッチに影響を与えることができます。表 12-2 に、HTTPUriType メンバー関数を示します。

表 12-2 HTTPUriType メンバー関数

HTTPUriType メソッド	説明
getClob	URL が指す値を CLOB 値として戻します。文字コードは、データベース・キャラクタ・セットの文字コードになります。
getUrl	格納された URL を戻します。
getExternalUrl	URL 仕様に準拠するために、URL の文字をエスケープするエスケープ・メカニズムをコールすることを除き、getUrl と同じです。たとえば、空白はエスケープ値 %20 に変換されます。
getBlob	バイナリ・コンテンツを BLOB として取得します。ターゲット・データがバイナリ以外の場合、BLOB は、データベース・キャラクタ・セットにデータの XML またはテキスト表現を含みます。
getXML	この URI に対応する XMLType オブジェクトを戻します。ターゲット・データが XML 以外の場合はエラーが発生します。12-9 ページの「getXML() 関数」を参照してください。
getContentType()	URL の MIME 情報を戻します。12-9 ページの「getContentType() 関数」を参照してください。
createUri()	HTTPUriType コンストラクタ。HTTPUriType で作成されます。

表 12-2 HTTPUriType メンバー関数（続き）

HTTPUriType メソッド	説明
httpUriType()	HTTPUriType コンストラクタ。HTTPUriType で作成されます。

getContentType() 関数

getContentType() 関数は、URL の MIME 情報を戻します。HTTPUriType は、URL を参照し、MIME ヘッダー情報を取得します。この情報を使用して、MIME 型に基づいて、URL を BLOB と CLOB のどちらとして取得するかを判断できます。MIME タイプ x/jpeg の Web ページを BLOB として処理し、MIME タイプ text/plain または text/html の Web ページを CLOB として処理します。

例 12-1 getContentType() および HTTPUriType を使用した HTTP ヘッダーの取得

コンテンツ・タイプを取得しても、すべてのデータがフェッチされるわけではありません。転送されるデータは、HTTP ヘッダー（HTTPUriType の場合）または列のメタデータ（DBUriType の場合）のみです。次に例を示します。

```
declare
  httpuri HttpUriType;
  x clob;
  y blob;
begin
  httpuri := HttpUriType('http://www.oracle.com/object1');
  if httpuri.getContentType() = 'application-x/bin' then
     y := httpuri.getblob();
  else
     x := httpuri.getclob();
  end if;
end;
```

getXML() 関数

getXML() 関数は、結果の XMLType 情報を戻します。文書が妥当な XML（または XHTML）でない場合は、エラーが発生します。

DBUri およびデータベース内参照

DBUri（データベースにおける URI）は、URIRef メカニズムの特殊なケースです。この場合、DBUri は、データベースおよびセッションのコンテキスト内で動作することが保証されます。DBUri は、HTTP の URL のようなグローバルな参照ではなく、データベース内のローカル参照（ローカル URL）です。

また、DBUri を処理できるサーブレットを識別する HTTP の URL パスに、DBUri を追加することによって、この URL が指すオブジェクトへグローバルにアクセスすることもできます。詳細は、12-34 ページの「DBUri サーブレットを使用した URL のデータベース問合せへの変換」を参照してください。

DBUri の表現

URL の構文は、データベースの仮想的な XML のビジュアル化で XPath に類似した構文を指定することによって取得されます。図 12-1「DBUri: ビジュアル・ビュー、SQL ビューまたは XML ビューと対応付けられた XPath」を参照してください。

- ビジュアル・モデルは、SQL スキーマ、表、行および列に関して現在接続中のユーザーが参照する階層的なビューです。
- XML ビューには、データベースにマップするルート要素が含まれます。ルート XML 要素には、子要素が含まれます。これらの子要素は、ユーザーが任意のオブジェクトに対する権限を取得しているスキーマです。スキーマ要素には、ユーザーが参照できる表およびビューが含まれます。

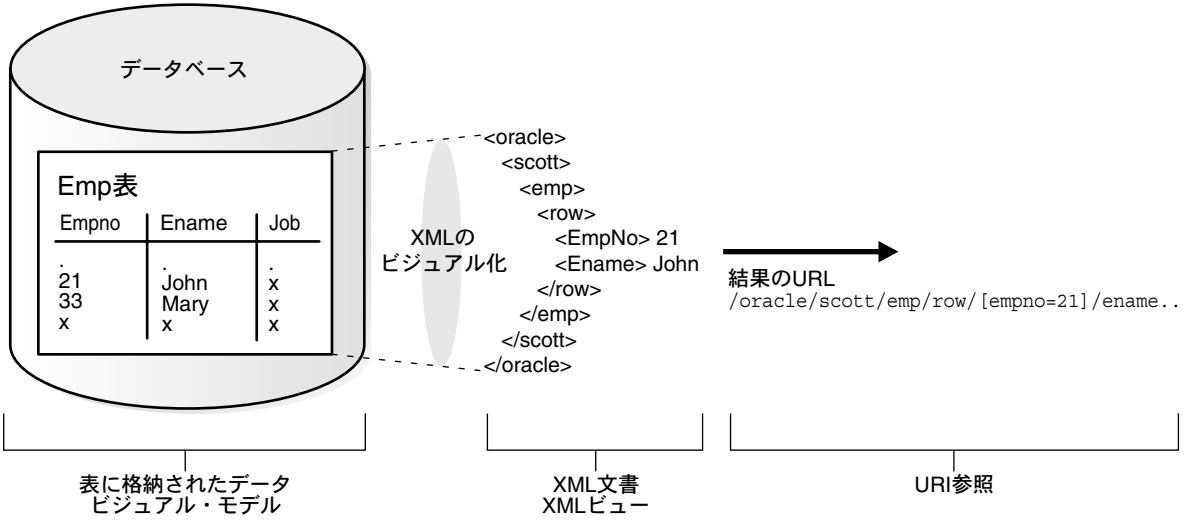
例 12-2 scott が参照できる仮想 XML 文書

たとえば、ユーザー scott は次の仮想 XML 文書を参照できます。

```
<?xml version='1.0'?>
<oradb SID="ORCL">
 <PUBLIC>
    <ALL_TABLES>
       ..
     </ALL_TABLES>
    <EMP>
      <!-- EMp table -->
    </EMP>
 </PUBLIC>
 <SCOTT>
  <ALL_TABLES>
    ....
  </ALL_TABLES>
  <EMP>
   <ROW>
     <EMPNO>1001</EMPNO>
     <ENAME>John</ENAME>
```

```
      <EMP_SALARY>20000</EMP_SALARY>
    </ROW>
    <ROW>
      <EMPNO>2001</EMPNO>
        </ROW>
  </EMP>
  <DEPT>
   <ROW>
     <DEPTNO>200</DEPTNO>
     <DNAME>Sports</DNAME>
   </ROW>
  </DEPT>
</SCOTT>
<JONES>
 <CUSTOMER_OBJ_TAB>
   <ROW>
     <NAME>xxx</NAME>
     <ADDRESS>
        <STATE>CA</STATE>
        <ZIP>94065</ZIP>
     </ADDRESS>
   </ROW>
  </CUSTOMER_OBJ_TAB>
 </JONES>
</oradb>
```

図 12-1 DBUri: ビジュアル・ビュー、SQL ビューまたは XML ビューと対応付けられた XPath



この XML 文書は、問合せの実行時に、その時点でユーザーが所有している権限に基づいて作成されます。

前述の例から、次のことがわかります。

- ユーザー scott は、scott データベース・スキーマおよび jones データベース・スキーマを参照できます。これらは、ユーザーが読み込むことができる表またはビューが存在するスキーマです。
- emp 表は、EMP として表示され、行の要素タグが含まれます。これはすべての表に対するデフォルトのマッピングです。dept 表および jones スキーマの customer_obj_tab 表でも同様です。
- 今回のリリースでは、NULL 要素は空として表示されます。
- また、スキーマ修飾がなくてもアクセスできる表およびビューが存在する PUBLIC 要素もあります。たとえば、次のような SELECT 問合せ文を発行するとします。

  ```
  SELECT * FROM emp;
  ```

  この場合、ユーザー scott による問合せでは、scott スキーマ内の emp 表と一致します。このような表が検出されない場合、emp という名前のパブリック・シノニムとの一致を試みます。同様に、PUBLIC 要素には次のものが含まれます。

  - ユーザーが自分のデータベース・スキーマを介して参照できるすべての表やビュー
  - PUBLIC シノニムを介して参照できるすべての表

DBUriType フラグメントの表記法

Oracle9*i* データベースが XML ツリーとしてビジュアル化される場合、仮想文書の任意の部分に対して XPath 検索を実行できます。これによって、データベース表およびビューにある行と列のすべての共通部分を取得できます。ビジュアル・モデル上で XPath を指定することによって、データベースのすべてのデータに対する参照を作成できます。

DBUri は、簡素化された XPath 形式で指定します。現在、DBUriType に対する XPath 勧告または XPointer 勧告が完全にサポートされていません。次の項では、これらの DBUri の構造について説明します。

前述のとおり、すべてのデータに対して DBUri を作成できます。次のインスタンスを列内で参照として使用できます。

- スカラー
- オブジェクト
- コレクション
- 列内のオブジェクト型の属性

次に例を示します。

```
.../ROW[empno=7263]/COL_OBJ/OBJ_ATTR
```

これらはアドレス指定可能な最小単位です。たとえば、次のように指定できます。

```
/oradb/SCOTT/EMP
```

または

```
/oradb/SCOTT/EMP/ROW[empno=7263]
```

> **注意：** 現在、スカラー、XMLType または LOB のデータ列内の参照はサポートされていません。

DBUri の構文のガイドライン

参照を指定するために使用できる XPath 問合せの種類には制限があります。通常、フラグメント部分は、次の構文ガイドラインに従う必要があります。

- ユーザー・データベース・スキーマ名を含めるか、または特定のスキーマを持たない表名を変換するために PUBLIC を指定する必要があります。
- 表名またはビュー名を含める必要があります。
- ROW 要素を識別するために ROW タグを含める必要があります。
- 抽出する列またはオブジェクト属性を識別する必要があります。

- パス内のスキーマ要素および表要素以外のレベルに述語を含める必要があります。
- 選択パスではなく、ROW 要素内に述語を指定する必要があります。

例 12-3 ROW ノードと述語 pono=100 の指定

たとえば、述語 pono = 100 を指定する場合に、選択パスが次のとおりであるとします。

```
/oradb/scott/purchase_obj_tab/ROW/line_item_list
```

この場合、次のとおり、ROW ノードとともに pono 述語を含める必要があります。

```
/oradb/scott/purchase_obj_tab/ROW[pono=100]/line_item_list
```

- DBUri は、単一のデータ値を正確に識別する必要があります。これらの値は、オブジェクト型かコレクションです。データ値が行全体になる場合、ROW ノードを使用してそれを明示する必要があります。また、DBUri は表全体を指すこともできます。

DBUri での述語（XPath）式の使用

述語式では、次の XPath 式を使用できます。

- ブール演算子 : AND、OR および NOT
- 関係演算子 : <、>、<=、!=、>=、=、mod、div、*（かけ算）

注意 :

- child 軸以外の XPath 軸は、サポートされていません。ワイルド・カード（*）、子孫（//）および他の操作は無効です。
- text() 以外の XPath 関数はサポートされていません。また、text() は、スカラー・ノード上でのみ有効です。行レベルや表レベルでは適用できません。

述語は、スキーマ要素および表要素以外のすべての要素で定義できます。オブジェクト列がある場合、属性値も検索できます。

例 12-4 DBUri を使用したオブジェクト列内の属性値の検索

たとえば、次の DBUri が、州、都市、通り、郵便番号などの属性を含む ADDRESS 列を参照するとします。

```
/oradb/SCOTT/EMP/ROW[ADDRESS/STATE='CA' OR
ADDRESS/STATE='OR']/ADDRESS[CITY='Portland' OR /ZIPCODE=94404]/CITY
```

この DBUri は、州が California か Oregon、または都市名が Portland か郵便番号が 94404 である都市属性を識別します。

参照： XML の XPath の表記法については、
http://www.w3.org/TR/xpath を参照してください。

DBUri の一般的な使用例

DBUri は、表、特定の行、行内の特定の列、オブジェクト列の特定の属性など、様々なオブジェクトを識別できます。次の項に、様々な種類のオブジェクトを識別する方法を示します。

表全体の指定

この例では、表全体を取り出す XML 文書を戻します。囲みタグは表名です。行の値は ROW 要素に囲まれます。次の構文を使用します。

/oradb/*schemaname*/*tablename*

例 12-5 DBUri を使用した表全体の XML 文書としての識別

次に例を示します。

```
/oradb/SCOTT/EMP
```

次の形式の XML 文書が戻されます。

```
<?xml version="1.0"?>
<EMP>
 <ROW>
   <EMPNO>7369</EMPNO>
   <ENAME>Smith</ENAME>
   ... <!-- other columns -->
 </ROW>
 <!-- other rows -->
</EMP>
```

表の特定の行の識別

この例では、表内の特定の ROW 要素を識別します。結果は、ROW 要素およびその子要素としての列を含む XML 文書です。次の構文を使用します。

/oradb/*schemaname*/*tablename*/ROW[*predicate_expression*]

例 12-6 DBUri を使用した表の特定の行の識別

次に例を示します。

```
/oradb/SCOTT/EMP/ROW[EMPNO=7369]
```

次の形式の XML 文書が戻されます。

```
<?xml version="1.0"?>
<ROW>
  <EMPNO>7369</EMPNO>
  <ENAME>SMITH</ENAME>
  <JOB>CLERK</JOB>
  <!-- other columns -->
</ROW>
```

注意： 前述の例では、述語式は一意の行を識別する必要があります。

ターゲット列の指定

この場合、ターゲット列または列の属性が識別され、XML として取り出されます。

注意： ネストした表または VARRY 列に対する全検索はできません。

次の構文を使用します。

/oradb/*schemaname*/*tablename*/ROW[*predicate_expression*]/*columnname*
/oradb/*schemaname*/*tablename*/ROW[*predicate_*
expression]/*columnname/attribute1/../attributen*

例 12-7 DBUri を使用した特定の列の識別

```
/oradb/SCOTT/EMP/ROW[EMPNO=7369 and DEPTNO=20]/ENAME
```

この例では、次のとおり、従業員番号が 7369 で部門番号が 20 の、emp 表内の ename 列を取り出します。

```
<?xml version="1.0"?>
<ENAME>SMITH</ENAME>
```

例 12-8 DBUri を使用した列内の属性の識別

```
/oradb/SCOTT/EMP/ROW[EMPNO=7369]/ADDRESS/STATE
```

この例では、次のとおり、従業員番号が 7369 である従業員の住所オブジェクト列内の州属性を取り出します。

```
<?xml version="1.0"?>
<STATE>CA</STATE>
```

列のテキスト値の取出し

多くの場合、囲みタグを取り出さずに、列のテキスト値のみを取り出す方が有効です。たとえば、XSLT スタイルシートが CLOB 列に格納されている場合、文書を列名のタグで囲まなくても、その文書を取り出すことができます。この場合、text() 関数を使用して、ノードのテキスト値のみを取り出すように指定できます。次の構文を使用します。

```
/oradb/schemaname/tablename/ROW[predicate_expression]/columnname/text()
```

例 12-9 DBUri を使用したノードのテキスト値のみの取出し

次に例を示します。

```
/oradb/SCOTT/EMP/ROW[EMPNO=7369]/ENAME/text()
```

この例では、従業員番号が 7369 の従業員名のテキスト値のみが、XML タグなしで取り出されます。戻される値は XML 文書ではなく、SMITH という値を持つテキスト・ドキュメントです。

注意： XPath のみでは、有効な URI を構成できません。Oracle では、これを DBUri と呼びますが（データベース内で URI のように動作するため）、DBUri をグローバルに有効な URIRef に変換することもできます。

注意： パスの大 / 小文字は区別されます。scott.emp を指定するには、SCOTT/EMP を使用します。これは、Oracle データ・ディクショナリでは、実際の表名が大文字で格納されているためです。小文字のパス値を使用する必要がある場合、小文字の表名または列名を二重引用符で囲みます。

DBUri とオブジェクト参照の相違点

DBUri は、列および属性にアクセスでき、型指定も厳密ではありません。オブジェクト参照は、行オブジェクトにのみアクセスできます。DBUri は、この参照メカニズムのスーパーセットです。

データベースおよびセッションに適用される DBUri

DBUri の有効範囲は、データベースおよびセッションです。ユーザーは、特定のセッション・コンテキストでデータベースに接続している必要があります。データにアクセスするために必要なスキーマおよび権限は、そのコンテキストで解決されます。

注意： 特に PUBLIC パスが使用される場合、使用するセッション・コンテキストによっては、同じ URI 文字列でも結果が異なる場合があります。

たとえば、/PUBLIC/FOO_TAB は、scott として接続している場合は SCOTT.FOO_TAB に変換され、JONES として接続している場合は JONES.FOO_TAB に変換されます。

DBUri が使用可能な場合

URIRef は、次の項に示すような多くの場合で使用できます。

関連ドキュメントに対する URL の格納

旅行体験記を表に格納している旅行体験記についての Web サイトでは、関連の体験記へのリンクを作成する場合があります。これらのリンクを DBUriType 列内で表すことによって、データベース内リンクを作成し、問合せを介して関連の体験記を取り出すことができます。

データベースへのスタイルシートの格納

アプリケーションは、XSLT スタイルシートを使用して、XML を他の形式に変換できます。スタイルシートは XML 文書として表され、CLOB として格納されます。アプリケーションは、次の目的で DBUriType オブジェクトを使用できます。

- 解析中に、データベースに格納されている XSLT スタイルシートにアクセスするため。
- 関連 XSLT スタイルシートへの参照（import や include など）を取得するため。これらの参照は、XSLT スタイルシート自体の中にエンコードできます。

注意：

- DBUri は、XML データに対する汎用 XPointer メカニズムではありません。
- DBUri は、データベース・オブジェクト参照の代替機能ではありません。参照の構文およびセマンティクスは、URIRef の構文およびセマンティクスとは異なります。
- DBUri は、新しいセキュリティ・モデルまたは制限を規定または作成しません。権限の規定に対しては、基礎となるセキュリティ・アーキテクチャに依存します。

DBUriType 関数

表 12-3 に、DBUriType メソッドおよび関数のリストを示します。

表 12-3 DBUriType メソッドおよび関数

メソッド / 関数	説明
getClob()	URL が指す値を CLOB 値として戻します。文字コードは、データベース・キャラクタ・セットの文字コードと同じです。
getUrl()	DBUriType に格納されている URL を戻します。
getExternalUrl()	URL 仕様に準拠するために、URL の文字をエスケープするエスケープ・メカニズムをコールすることを除き、getUrl と同じです。たとえば、空白はエスケープ値 %20 に変換されます。
getBlob()	バイナリ・コンテンツを BLOB として取得します。ターゲット・データがバイナリ以外の場合、BLOB は、データベース・キャラクタ・セットにデータの XML またはテキスト表現を含みます。
getXML()	この URI に対応する XMLType オブジェクトを戻します。
getContentType()	URL の MIME 情報を戻します。
createUri()	DBUriType インスタンスを作成します。
dbUriType()	DBUriType インスタンスを作成します。

次の項では、DBUriType 関数のうち、異なる動作や特別な動作をする関数について説明します。

getContentType() 関数

この関数は、URL の MIME 情報を戻します。DBUriType オブジェクトのコンテンツ・タイプは、次のようになります。

- DBUri がスカラー値を指す場合、MIME タイプは text/plain です。
- その他の場合、MIME タイプは text/xml です。

たとえば、SCOTT 下の dbtab 表について考えてみます。

```
CREATE TABLE DBTAB( a varchar2(20), b blob);
```

DBUriType の「/SCOTT/DBTAB/ROW/A」は列全体を指し、結果が XML であるため、コンテンツ・タイプは text/xml です。

DBUriType の「/SCOTT/DBTAB/ROW/B」のコンテンツ・タイプも text/xml です。

DBUriType の「/SCOTT/DBTAB/ROW/A/text()」のコンテンツ・タイプは text/plain です。

DBUriType の「/SCOTT/DBTAB/ROW/B/text()」のコンテンツ・タイプも text/plain です。

getClob() 関数および getBlob() 関数

DBUri の場合、スカラー・バイナリ・データは特別に処理されます。DBUri の「/SCOTT/DBTAB/ROW/B/text()」（B は BLOB 列）に対する getClob() コールの場合、データは 16 進形式に変換され、送信されます。

getBlob() コールの場合、データはバイナリ形式で戻されます。ただし、「/SCOTT/DBTAB/ROW/B」のように XML 文書を要求した場合、XML 文書には 16 進形式のバイナリが含まれます。

XDBUriType

XDBUriType は、URIType の新しいサブタイプです。XDBUriType を使用すると、表内のすべての URIType 列に埋込み可能な URI として、Oracle XML DB Repository 内のドキュメントを公開できます。

URI の URL 部分は、その URI が参照する XML 文書の階層名です。オプションのフラグメント部分には、XPath 構文を使用します。この部分は、「#」によって URL 部分と区切られます。

次に、Oracle XML DB の URI の例を示します。

```
/home/scott/doc1.xml
/home/scott/doc1.xml#/purchaseOrder/lineItem
```

ここで、各部分は次の内容を表します。

- 「/home/scott」は、Oracle XML DB Repository 内のフォルダです。
- doc1.xml は、このフォルダ内の XML 文書です。
- XPath 式 /purchaseOrder/lineItem は、この発注書内の明細項目を参照します。

表 12-4 に、XDBUriType メソッドのリストを示します。これらのメソッドは、引数を取りません。

表 12-4 XDBUriType メソッド

メソッド	説明
getClob()	URL が指す値をキャラクタ・ラージ・オブジェクト（CLOB）値として戻します。文字コードは、データベース・キャラクタ・セットの文字コードと同じです。
get Blob()	URL が指す値をバイナリ・ラージ・オブジェクト（BLOB）値として戻します。
getUrl()	XDBUriType に格納されている URL を戻します。
getExternalUrl()	URL 仕様に準拠するために、URL の文字をエスケープするエスケープ・メカニズムをコールすることを除き、getUrl と同じです。たとえば、空白はエスケープ値 %20 に変換されます。
getXML()	この URI が指すリソースのコンテンツに対応する XMLType オブジェクトを戻します。これは、getClob/getBlob 以外の操作を実行する必要があるアプリケーションが、これらの操作を実行するために XMLType メソッドを使用できるように提供されています。
getContentType()	Oracle XML DB Repository に格納されたリソースの MIME 情報を戻します。
XDBUriType()	コンストラクタです。指定された URI に対応する XDBUriType を戻します。

XDBUriType インスタンスの作成方法

XDBUriType は、URIFactory に自動的に登録されるため、URI を getURI メソッドに指定することによって、XDBUriType インスタンスを生成できます。

現在、XDBUriType は、「http://」、「/DBURI」、「/ORADB」などの認識されている接頭辞が URI に含まれていない場合、UriFactory.getUri メソッドによって生成されるデフォルトの URIType です。

DBUriType のすべての URI には、/DBURI か /ORADB（大 / 小文字は区別されない）のいずれかの接頭辞が含まれる必要があります。

例 12-10 XDBUriType インスタンスの取得

たとえば、次の文は、/home/scott/doc1.xml を参照する XDBUriType インスタンスを戻します。

```
SELECT sys.UriFactory.getUri('/home/scott/doc1.xml') FROM dual;
```

例 12-11 XDBUriType の作成、発注書表への値の挿入およびすべての発注書の選択

次に、XDBUriType の使用方法の例を示します。

```
CREATE TABLE uri_tab
(
  poUrl SYS.UriType,  -- Note that we have created an abstract type column
                       --so that any type of URI can be used
   poName VARCHAR2(1000)
);

 -- insert an absolute url into poUrl
 -- the factory will create an XDBUriType since there's no prefix
INSERT INTO uri_tab VALUES
  (UriFactory.getUri('/public/orders/po1.xml'), 'SomePurchaseOrder');

-- Now get all the purchase orders
SELECT e.poUrl.getClob(), poName FROM uri_tab e;

-- Using PL/SQL, you can access table uri_tab as follows:
declare
   a UriType;
begin
  -- Get the absolute URL for purchase order named like 'Some%'
   SELECT poUrl into a from uri_tab WHERE poName like 'Some%';
   printDataOut(a.getClob());
end;
/
```

例 12-12 URIType、getXML() および extractValue() を使用した、ある URL に存在する発注書の取出し

getXML() は XMLType を戻すため、演算子の EXTRACT グループで使用できます。次に例を示します。

```
SELECT e.poUrl.getClob() FROM uri_tab e
   WHERE extractValue(e.poUrl.getXML(),'/User') = 'SCOTT';
```

この文によって、ユーザー SCOTT のすべての発注書が戻されます。

URIType 列での Oracle Text 索引の作成

Oracle Text を使用して、Oracle9*i* データベース内で URIType 列をネイティブに索引付けできます。特別なデータストアは不要です。

参照： 7-34 ページの「XMLType の索引付け」を参照してください。

URIType オブジェクトの使用

この項では、ドキュメントへのポインタを格納する方法、およびデータベースまたは Web サイトのいずれかからネットワークを介してこれらのドキュメントを取り出す方法について説明します。

URIType を使用したドキュメントへのポインタの格納

前述のとおり、URIType は、URI を指定する VARCHAR2 属性を含む抽象型です。オブジェクト型には、参照の全検索およびデータ抽出用の関数もあります。

URIType を使用して列を作成し、これらのポインタをデータベースに格納することができます。通常、URIType を使用して列を宣言し、格納するオブジェクトに HTTPUriType などの導出された型を 1 つ以上使用します。

表 12-4 に、有効な URIType メソッドを示します。

注意： 継承メカニズムを使用して、任意の新しいプロトコルをプラグインできます。Oracle は、HTTP プロトコルの処理および DBUri 参照の解読のために、HTTPUriType 型および DBUriType 型を提供します。たとえば、URIType のサブタイプを実装して、gopher などのプロトコルを処理できます。

例 12-13 発注書のリストに対する URL 参照の作成

発注書のリストを、発注書に対する URL 参照付きで作成できます。次に例を示します。

```
CREATE TABLE uri_tab
(
   poUrl SYS.UriType,  -- Note that we have created abstract type columns
-- if you know what kind of uri's you are going to store, you can
-- create the appropriate types.
   poName VARCHAR2(200)
);

 -- insert an absolute url into SYS.UriType..!
 -- the Urifactory creates the correct instance (in this case a HttpUriType
```

```
INSERT INTO uri_tab VALUES
  (sys.UriFactory.getUri('http://www.oracle.com/cust/po'),'AbsPo');

-- insert a URL by directly calling the SYS.HttpUriType constructor.
-- Note this is strongly discouraged. Note the absence of the
-- http:// prefix when creating SYS.HttpUriType instance through the default
-- constructor.
INSERT INTO uri_tab VALUES (sys.HttpUriType('proxy.us.oracle.com'),'RelPo');

-- Now extract all the purchase orders
SELECT e.poUrl.getClob(), poName FROM uri_tab e;

-- In PL/SQL
declare
   a SYS.UriType;
begin

 -- absolute URL
 SELECT poUrl into a from uri_Tab WHERE poName like 'AbsPo%';

 SELECT poUrl into a from uri_Tab WHERE poName like 'RelPo%';
 -- here u need to supply a prefix before u can get at the data..!
 printDataOut(a.getClob());
end;
/
```

> **参照：** URIFactory の使用方法の詳細は、12-25 ページの「URIFactory パッケージを使用した URIType オブジェクトのインスタンスの作成」を参照してください。

置換メカニズムの使用

URIType の列を直接作成し、その列に HTTPUriType、XDBUriType および DBUriType を挿入できます。また、参照されるドキュメントの格納場所がわからなくても、列を問い合せることができます。たとえば、前述の例では、次のように、uri_tab 表に DBUri 参照を挿入することもできます。

```
INSERT INTO uri_tab VALUES
  (UriFactory.getUri(
     '/SCOTT/PURCHASE_ORDER_TAB/ROW[PONO=1000]'),'ScottPo');
```

この INSERT 文では、SCOTT スキーマ内に発注書の表があると想定しています。この場合、表の URL 列には、HTTP を介してグローバルなドキュメントを指し、データベース内の仮想ドキュメントも指す値が含まれます。

getClob() メソッドを使用してその列に SELECT 文を実行すると、ドキュメントの格納場所にかかわらず、結果が CLOB として取り出されます。この SELECT 文では、最初の行に

格納されているグローバル HTTP アドレス、およびローカルな DBUri 参照から値が取り出されます。

```
SELECT e.poURL.getclob() FROM uri_tab e;
```

HTTPUriType および DBUriType の使用

HTTPUriType および DBUriType は、URIType のサブタイプです。これらは、それぞれ HTTP 参照用および DBUri 参照用の関数を実装します。

> **注意：** 今回のリリースでは、HTTPUriType は相対 HTTP 参照を格納できません。

例 12-14 DBUriType:DBUri 参照の作成

次の例では、DBUriType 型の列を含む表を作成し、その表に値を割り当てます。

```
CREATE TABLE DBURiTab(DBUri DBUriType, dbDocName VARCHAR2(2000));

-- insert values into it..!
INSERT INTO DBUriTab VALUES
   (sys.DBUriType.createUri('/ORADB/SCOTT/EMP/ROW[EMPNO=7369]'),'emp1');

INSERT INTO DBUriTab VALUES
   (sys.DBUriType('/SCOTT/EMP/ROW[EMPNO=7369]/',null);

-- access the references
SELECT e.DBUri.getCLOB() from DBUriTab e;
```

URIFactory パッケージを使用した URIType オブジェクトのインスタンスの作成

URIFactory パッケージ内のファンクションは、URIType の適切なサブタイプ（HTTPUriType、DBUriType および XDBUriType）のインスタンスを生成します。これによって、プログラムで実装をハードコード化する必要がなくなり、どのような URI 文字列が入力されても処理できます。表 12-5 を参照してください。

getUri メソッドは、サポートされている様々な URI を表す文字列を取り、適切なサブタイプのインスタンスを戻します。次に例を示します。

- 接頭辞が http:// で始まる場合、getUri は SYS.HttpUriType オブジェクトのインスタンスを作成して戻します。
- 文字列が /oradb/ または /dburi/ で始まる場合、getUri は SYS.DBUriType オブジェクトのインスタンスを作成して戻します。

- 文字列が前述のいずれかの接頭辞で始まらない場合、getUri は SYS.XDBUriType オブジェクトのインスタンスを作成して戻します。

注意： URIFactory が DBUriType インスタンスを生成する方法は、Oracle9*i* データベース リリース 1（9.0.1）から変更されています。

Oracle9*i* データベース リリース 1（9.0.1）では、登録された接頭辞または http://... などの標準の接頭辞で始まらない URL は、URIFactory によって DBUriType にマップされていました。

今回のリリースでは、URIFactory で DBUriType を生成するには、接頭辞に /oradb または /dburi を使用する必要があります。それ以外の場合、XDBUriType が生成されます。

URIFactory パッケージを使用した URIType の新しいサブタイプの登録

URIFactory パッケージを使用すると、次のとおり、URIType の新しいサブタイプを登録できます。

- SQL の CREATE TYPE 文を使用して、これらの型を導出します。
- デフォルトのメソッドをオーバーライドして、データを取り出すときに特別な処理を実行するか、または XML データを変換してから表示します。
- この特別な処理に使用する URI を識別する新しい接頭辞を決定します。
- UriFactory.registerURLHandler を使用して、接頭辞を登録します。これによって、定義した新しい接頭辞で始まる URI を受け取った URIFactory パッケージが、新しいサブタイプのインスタンスを作成できるようになります。

たとえば、新しいプロトコル ecom:// を作成し、そのプロトコルを処理するための URIType のサブタイプを定義できます。そのサブタイプは、getCLOB のいくつかの特別な論理を実装したり、getXML 内で XML タグやデータを変更します。ecom:// 接頭辞を URIFactory に登録すると、UriFactory.getUri へコールすることによって、ecom:// 接頭辞で始まる URI に対して新しいサブタイプのインスタンスが生成されます。

表 12-5 URIFactory: ファンクションおよびプロシージャ

URIFactory ファンクション	説明
escapeUri() MEMBER FUNCTION escapeUri() RETURN varchar2	URIRef 仕様に指定された URL 以外の文字列を同等のエスケープ・シーケンスで置き換えることによって、URL 文字列をエスケープします。
unescapeUri() FUNCTION unescapeUri() RETURN varchar2	指定された URL をエスケープしません。
registerUrlHandler() PROCEDURE registerUrlHandler(prefix IN varchar2, schemaName in varchar2, typename in varchar2, ignoreCase in boolean:= true, strippprefix in boolean := true)	特定の URL を処理するための特定の型名を登録します。 この型は、次の静的メンバー関数も実装します。 `STATIC FUNCTION createUri(url IN varchar2) RETURN <typename>;` このファンクションは、その型のインスタンスを生成するために `getUrl()` によってコールされます。`stripprefix` は、その型の適切なコンストラクタをコールする前に、接頭辞を削除する必要があることを示します。
unRegisterUrlHandler() PROCEDURE unregisterUrlHandler(prefix in varchar2)	URL ハンドラを登録解除します。

例 12-15 URIFactory: ecom プロトコルの登録

単一の表に様々な URI を格納するとします。

```
CREATE TABLE url_tab (urlcol varchar2(80));

-- Insert an HTTP URL
INSERT INTO url_tab VALUES ('http://www.oracle.com/');

-- Insert a database URI
INSERT INTO url_tab VALUES ('/oradb/SCOTT/EMPLOYEE/ROW[ENAME="Jack"]');

-- Create a new type to handle a new protocol called ecom://
CREATE TYPE EComUriType UNDER SYS.UriType
(
  overriding member function getClob return clob,
  overriding member function getBlob RETURN blob,
  overriding member function getExternalUrl return varchar2,
  overriding member function getUrl return varchar2,

-- Must have this for registering with the URL handler
static function createUri(url in varchar2) return EcomUriType
```

```
);
/

-- Register a new handler for the ecom:// prefix.
begin
  -- register a new handler for ecom:// prefixes. The handler
  -- type name is ECOMUriTYPE, schema is SCOTT
  -- Ignore the prefix case, so that UriFactory creates the same subtype
  -- for URIs beginning with ECOM://, ecom://, eCom://, and so on.
  -- Strip the prefix before calling the createUri function
  -- so that the string 'ecom://' is not stored inside the
  -- ECOMUriTYPE object. (It is added back automatically when
  -- you call ECOMUriTYPE.getURL.)
  urifactory.registerURLHandler
  (
    prefix => 'ecom://',
    schemaname => 'SCOTT',
    typename => 'ECOMURITYPE',
    ignoreprefixcase => true,
    stripprefix => true
  );
end;
/

-- Now the example inserts this new type of URI into the table.
insert into url_tab values ('ECOM://company1/company2=22/comp');

-- Use the factory to generate an instance of the appropriate
-- subtype for each URI in the table.
select urifactory.getUri(urlcol) from url_tab;

-- would now generate
HttpUriType('www.oracle.com'); -- a Http uri type instance

DBUriType('/oradb/SCOTT/EMPLOYEE/ROW[ENAME="Jack"],null); -- a DBUriType

EComUriType('company1/company2=22/comp'); -- an EComUriType instance
```

URIType の新しいサブタイプを定義するメリット

プロトコルごとに新しいクラスを導出するメリットは、次の 2 つです。

- 列を表すためにサブタイプを選択する場合、その列に対して暗黙的な制約が規定され、特定のプロトコル型のインスタンスのみが含まれます。これは、その列に特定のプロトコルのスペシャライズ・インデックスを実装する場合に有効です。たとえば、DBUri の場合、SQL 問合せを実行するかわりに、直接ディスク・ブロックに移動してデータをフェッチできるスペシャライズ・インデックスを実装できます。
- 関連の型に基づいて、列に異なる制約を規定できます。たとえば HTTP の場合、列に対してプロキシおよびファイアウォールの制約を潜在的に定義して、HTTP を介したすべてのアクセスでプロキシ・サーバーを使用するようにできます。

SQL 関数 SYS_DBURIGEN()

DBUriType 型のインスタンスは、コンストラクタ・メソッドまたは URIFactory メソッドへのパス式を指定することによって作成できます。ただし、表の列に格納された文字列に基づいて、これらのオブジェクトを動的に生成するためのメソッドも必要です。これを行うには、SQL 関数 SYS_DBURIGEN() を使用します。

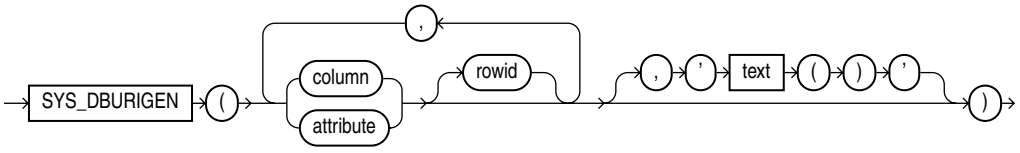
例 12-16 SYS_DBURIGEN(): 列を指す DBUriType 型の URI の生成

次の例では、SYS_DBURIGEN() を使用して、サンプル表 hr.employees にある employee_id = 206 の行の email 列に対して、DBUriType データ型の URI を生成します。

```
SELECT SYS_DBURIGEN(employee_id, email)
   FROM employees
   WHERE employee_id = 206;

SYS_DBURIGEN(EMPLOYEE_ID,EMAIL)(URL, SPARE)
--------------------------------------------------------------------
DBURITYPE('/PUBLIC/EMPLOYEES/ROW[EMPLOYEE_ID = "206"]/EMAIL', NULL)
```

SYS_DBURIGEN() は、引数として 1 つ以上の列または属性、およびオプションで ROWID を取り、特定の列オブジェクトまたは行オブジェクトに対して DBUriType データ型の URI を生成します。その URI を使用して、データベースから XML 文書を取り出せます。また、この関数は、追加のパラメータを取り、ノードのテキスト値が必要かどうかを示します。図 12-2 を参照してください。

図 12-2 SYS_DBURIGEN の構文



参照するすべての列または属性は、同じ表内に存在する必要があります。それらの列または属性は、一意の値を参照する必要があります。複数の列を指定した場合、最初の列はデータベース内の行を識別し、最後の列はその行内の列を識別します。

デフォルトでは、URI は、フォーマットされた XML 文書を指します。URI が文書のテキストのみを指すようにするには、オプションの `text()` キーワードを指定します。

> **参照：** `SYS_DBURIGEN` の構文については、『Oracle9*i* SQL リファレンス』を参照してください。

XML Schema を指定しない場合、Oracle は、表名またはビュー名をパブリック・シノニムに変換します。

列またはオブジェクト属性を SYS_DBURIGEN() に渡す規則

`SYS_DBURIGEN()` 関数に渡される列または属性は、次の規則に従う必要があります。

- **一意のマッピング :** 列またオブジェクト属性は、元の表またはビューに一意にマップ可能である必要があります。使用可能な仮想列は、`VALUE` 演算子および `REF` 演算子のみです。列は、`TABLE()` 副問合せまたはインライン・ビュー（そのインライン・ビューが列の名前を変更しない場合）から取得される場合があります。
- **キー列 :** ROWID または一連のキー列のいずれかを指定する必要があります。キー列のリストは、列が結果内の特定の行を一意に識別できるかぎり、一意キーまたは主キーとして宣言する必要はありません。
- **同一の表 :** `SYS_DBURIGEN()` 関数で参照されるすべての列は、同じ表またはビューから取得する必要があります。
- **PUBLIC 要素 :** ROWID またはキー列が指す表またはビューに指定されたデータベース・スキーマがない場合、スキーマのかわりに `PUBLIC` キーワードが使用されます。`DBUri` がアクセスされたときに、表名が、その `DBUri` が作成されたときにその名前で参照できた表、シノニムまたはビューに変換されます。
- **TEXT 関数 :** デフォルトでは、`DBUri` は、結果を含む XML 文書を取り出します。テキスト値のみを取り出すには、関数の最後の引数として `text()` キーワードを使用します。

次に例を示します。

```
select SYS_DBURIGEN(empno,ename,'text()') from scott.emp,
   WHERE empno=7369;
```

または、次の形式の URL のみを生成できます。

```
/SCOTT/EMP/ROW[EMPNO=7369]/ENAME/text()
```

- **単一列の引数 :** 単一列の引数がある場合、列は、行を識別するキー列と参照列の両方として使用されます。

例 12-17 SYS_DBURIGEN() への単一の引数を持つ列の送信

次に例を示します。

```
select SYS_DBURIGEN(empno) from emp
   WHERE empno=7369;
```

この構文は、empno をキー列と参照列の両方として使用し、次の形式の URL を生成します。

```
/SCOTT/EMP/ROW[EMPNO=7369]/EMPNO,
```

ここで、行は empno=7369 です。

SYS_DBURIGEN の例

例 12-18 SYS_DBURIGEN() を使用したデータベース参照の挿入

```
CREATE TABLE doc_list_tab(docno number primary key, doc_ref SYS.DBUriType);

-- inserts /SCOTT/EMP/ROW[rowid='xxx']/EMPNO
INSERT INTO doc_list_tab values(1001,
     (select SYS_DBURIGEN(rowid,empno) from emp where empno = 100));

-- insert a Uri-ref to point to the ename column of emp!
INSERT INTO doc_list_tab values(1002,
   (select SYS_DBURIGEN(empno, ename) from emp where empno = 7369));

-- result of the DBURIGEN looks like, /SCOTT/EMP/ROW[EMPNO=7369]/ENAME
```

部分的な結果の取得

大きい列の結果を選択する場合、結果の一部のみを取り出し、かわりに列に対する URL を作成することがあります。たとえば、旅行体験記についての Web サイトを考えてみます。表にすべての旅行体験記が格納されている場合、ユーザーは、関連する一連の体験記を検索します。この場合、問合せ結果のページには、体験記の全文ではなく、最初の 100 文字または話の要点のみを表示し、実際の話の URL を戻す必要があります。これを行うには、次の手順を実行します。

例 12-19 ビューの作成および SYS_DBURIGEN() を使用した部分的な結果の取得

旅行体験記の表が、次のように定義されているとします。

```
CREATE TABLE travel_story
(
  story_name varchar2(100),
  story clob
);

-- insert some value..!
INSERT INTO travel_story values ('Egypt','This is my story of how I spent my time in
Egypt, with the pyramids in full view from my hotel room');
```

旅行体験記の最初の 20 文字のみを戻す関数を作成します。

```
create function charfunc(clobval IN clob ) return varchar2 is
 res varchar2(20);
 amount number := 20;
begin
  dbms_lob.read(clobval,amount,1,res);
  return res;
end;
/
```

旅行体験記の最初の 100 文字のみを選択し、story 列に対する DBUri 参照を戻すビューを作成します。

```
CREATE VIEW travel_view as select story_name, charfunc(story) short_story,
   SYS_DBURIGEN(story_name,story,'text()') story_link
FROM travel_story;
```

ビューに対する SELECT 文の結果は、次のようになります。

```
SELECT * FROM travel_view;

STORY_NAME      SHORT_STORY             STORY_LINK
--------------------------------------------------------------------------------
Egypt           This is my story of h

SYS.DBUriType('/PUBLIC/TRAVEL_STORY/ROW[STORY_NAME='Egypt']/STORY/text()')
```

URIRef の取得

DML 文の RETURNING 句で SYS_DBURIGEN() を使用すると、オブジェクトの挿入時にそのオブジェクトの URL を取得できます。

例 12-20 RETURNING 句で SYS_DBURIGEN を使用したオブジェクトの URL の取得

たとえば、表 CLOB_TAB について考えてみます。

```
CREATE TABLE clob_tab ( docid number, doc clob);
```

ドキュメントを挿入する場合、そのドキュメントの URL を別の表 URI_TAB に格納する必要がある場合があります。

```
CREATE TABLE uri_tab (docs sys.DBUriType);
```

そのドキュメントの URL の格納は、CLOB_TAB に対する挿入の一部として指定できます。これを行うには、RETURNING 句および EXECUTE IMMEDIATE 構文を使用して、PL/SQL 内で SYS_DBURIGEN 関数を実行します。次に例を示します。

```
declare
  ret sys.dburitype;
begin
  -- exucute the insert and get the url
 EXECUTE IMMEDIATE
'insert into clob_tab values (1,''TEMP CLOB TEST'')
 RETURNING SYS_DBURIGEN(docid, doc, ''text()'') INTO :1 '
 RETURNING INTO ret;
 -- insert the url into uri_tab
insert into uri_tab values (ret);
end;
/
```

作成される URL は、次の形式になります。

```
/SCOTT/CLOB_TAB/ROW[DOCID="xxx"]/DOC/text()
```

注意： text() キーワードを最後に追加すると、CLOB 値のみが戻り、CLOB テキストを囲む XML 文書は戻されません。

DBUri サーブレットを使用した URL のデータベース問合せへの変換

表データは、ブラウザまたは Web クライアントからアクセス可能にすることができます。これを行うには、次の方法を使用して、URL 内で URI 表記法を使用して取得するデータを指定します。

- データベース・サーバーにリンクされた DBUri サーブレットを使用します。
- サーブレット・エンジン上で動作する独自のサーブレットを作成します。サーブレットは、コール側の URL のパスから URI 文字列を読み込み、その URI を使用して DBUriType オブジェクトを作成し、データを取り出す URIType メソッドをコールして、Web ページまたは XML 文書の形式で値を戻します。

> **注意：** Oracle Servlet Engine（OSE）はサポートされていません。したがって、Oracle9*i* データベース リリース 1（9.0.1）でサポートされている oracle.xml.dburi.OraDBUriServlet も、サポートされていません。かわりに、Oracle XML DB Servlet システムを使用する DBUri C サーブレットを使用してください。第 20 章「Java での Oracle XML DB アプリケーションの作成」を参照してください。

DBUri サーブレットのメカニズム

前述の方法の場合、サーブレットは HTTP を介してこの情報にアクセスするように動作します。このサーブレットは、パス式で、サーブレット名に続いて DBUri 参照を取り、DBUri が指す文書を出力ストリームに出力します。

生成されるドキュメントは、Web ページ、XML 文書、プレーン・テキストなどです。ブラウザまたはその他のアプリケーションが内容を判断できるように、MIME タイプを指定できます。

- デフォルトでは、サーブレットは text/xml および text/plain の MIME タイプを作成できます。URI の最後が text() 関数の場合は、text/plain の MIME タイプが使用されます。それ以外の場合は、XML 文書は text/xml の MIME タイプを使用して生成されます。
- サーブレットに対するコンテンツ型の引数を使用して、MIME タイプをオーバーライドし、binary/x-jpeg またはその他の値に設定できます。

例 12-21 コンテンツ型の引数を生成し、MIME タイプをオーバーライドするための URL（Employee 表の empno 列を取り出す場合）

たとえば、employee 表の empno 列を取り出すには、次のような URL のいずれかを記述できます。

```
-- Generates a contenttype of text/plain
http://machine.oracle.com:8080/oradb/SCOTT/EMP/ROW[EMPNO=7369]/ENAME/text()
-- Generates a contenttype of text/xml
http://machine.oracle.com:8080/oradb/SCOTT/EMP/ROW[EMPNO=7369/ENAME
```

ここでのマシン machine.oracle.com は Oracle9*i* データベースを実行しており、Web サービスがポート 8080 でリクエストをリスニングしています。oradb は、サーブレットにマップする仮想パスです。

DBUri サーブレット : オプションの引数

表 12-6 に、出力をカスタマイズするために DBUri サーブレットに渡すことができる 3 つのオプションの引数を示します。

表 12-6 DBUri サーブレット : オプションの引数

引数	説明
rowsettag	XML 文書のデフォルトのルート・タグ名を変更します。次に例を示します。 http://machine.oracle.com:8080/oradb/SCOTT/EMP?rowsettag=Employee
contenttype	戻されるドキュメントの MIME タイプを指定します。次に例を示します。 http://machine.oracle.com:8080/oradb/SCOTT/EMP?contenttype=text/plain
transform	URL を URIFactory に渡します。これによって、その場所にある XSLT スタイルシートが取得されます。このスタイルシートは、サーブレットによって戻された XML 文書に適用されます。次に例を示します。 http://machine.oracle.com:8080/oradb/SCOTT/EMP?transform=/oradb/SCOTT/XSLS/DOC/text()&contenttype=text/html

注意 : このサーブレットの URL に XPath 表記法を使用する場合、大カッコ（[、]）などの特定の文字をエスケープする必要がある場合があります。エスケープ済の URL を取得するには、URIType 型の getExternalUrl() 関数を使用します。

注意： HTTP アクセスでは、特殊文字（]、[、&、| など）は、%*xx* フォーマットを使用してエスケープする必要があります。この *xx* は、その特殊文字の ASCII コードを指す 16 進数の数字です。エスケープ済の URL を取得するには、`URIType` グループの `getExternalUrl()` 関数を使用します。

DBUri サーブレットのインストール

`DBUri` サーブレットはデータベースに組み込まれており、インストールは Oracle XML DB の構成ファイルによって処理されます。サーブレットのインストールをカスタマイズするには、Oracle XML DB の構成ファイルを編集する必要があります。構成ファイル（`/xdbconfig.xml`）は、WebDAV または FTP を介して、Oracle Enterprise Manager から、またはデータベース内で編集できます。FTP または WebDAV を使用してファイルを更新するには、ドキュメントをダウンロードし、必要に応じて編集してからデータベース内に再度保存します。構成ファイルを編集して、他にもいくつかの項目をカスタマイズできます。

参照： 次の章または付録を参照してください。

- 第 20 章「Java での Oracle XML DB アプリケーションの作成」
- 第 21 章「Oracle Enterprise Manager を使用した Oracle XML DB の管理」
- 付録 A「Oracle XML DB のインストールおよび構成」

サーブレットは、servlet-pattern タグで指定された `/oradb/*` にインストールされることに注意してください。「*」は、`oradb` 以降のすべてのパスが同じサーブレットにマップされることを示すために必要です。oradb は仮想パスとして公開されます。ここで、サーブレットにアクセスするために使用するパスを変更できます。

例 12-22 /dburi/* への DBUri サーブレットのインストール

たとえば、サーブレットを `/dburi/*` にインストールするには、次の PL/SQL を実行します。

```
declare
  doc XMLType;
  doc2 XMLType;
begin
  doc := dbms_xdb.cfg_get();
  select updateXML(doc,
'/xdbconfig/sysconfig/protocolconfig/httpconfig/webappconfig/servletconfig/servlet-m
appings/servlet-mapping[servlet-name="DBUriServlet"]/servlet-pattern/text()',
'/dburi/*') into doc2 from dual;
  dbms_xdb.cfg_update(doc2);
```

```
  commit;
end;
/
```

セキュリティ・パラメータ、サーブレットの表示名および説明も、xdbconfig.xml 構成ファイル内でカスタマイズできます。付録 A「Oracle XML DB のインストールおよび構成」および第 20 章「Java での Oracle XML DB アプリケーションの作成」を参照してください。サーブレットは、servlet-pattern を削除することによって削除できます。また、updateXML() を使用して servlet-mapping 要素を NULL に更新することによって削除することもできます。

DBUri のセキュリティ

サーブレットのセキュリティは、Oracle9*i* データベースでロールを使用することによって実行されます。サーブレットにログインするユーザーは、ユーザー自身のデータベース・ユーザー名およびパスワードを使用します。サーブレットは、ログインするユーザーが、構成ファイルに指定されているロールに属していることを確認します。サーブレットへのアクセスを許可するロールは、security-role-ref タグ内に指定されています。デフォルトでは、サーブレットは、**authenticatedUser** という特別なロールで使用できます。有効なデータベース・ユーザー名およびパスワードを使用してサーブレットにログインするすべてのユーザーは、このロールに属しています。

このパラメータを変更して、データベース内の任意のロールにアクセスを制限することができます。デフォルトの authenticatedUser ロールから、ユーザーが作成した servlet-users などのロールに変更するには、次のように宣言します。

```
declare
  doc XMLType;
  doc2 XMLType;
  doc3 XMLType;
begin
  doc := dbms_xdb.cfg_get();
  select updateXML(doc,
'/xdbconfig/sysconfig/protocolconfig/httpconfig/webappconfig/servletconfig/servlet-l
ist/servlet[servlet-name="DBUriServlet"]/security-role-ref/role-name/text()',
'servlet-users') into doc2 from dual;
  select updateXML(doc2,
'/xdbconfig/sysconfig/protocolconfig/httpconfig/webappconfig/servletconfig/servlet-l
ist/servlet[servlet-name="DBUriServlet"]/security-role-ref/role-link/text()',
'servlet-users') into doc3 from dual;
  dbms_xdb.cfg_update(doc3);
  commit;
end;
/
```

DBUri を処理する URIFactory パッケージの構成

URIFactory（12-25 ページの「URIFactory パッケージを使用した URIType オブジェクトのインスタンスの作成」を参照）は、URL を取り、URIType の適切なサブタイプを生成して、対応するプロトコルを処理します。HTTP の URL の場合、URIFactory は HTTPUriType のインスタンスを作成します。ただし、HTTP の URL が URI のパスを表している場合、その URL をデータベースの DBUriType インスタンスとして格納および処理する方がより効率的です。DBUriType の処理に必要な通信レイヤーは少なく、文字変換も少ない場合があります。

http://machine-name/servlets/oradb/ のようなすべての URL をサーブレットで処理するために OraDBUriServlet をインストールした場合、URIFactory がその接頭辞を使用して、HTTPUriType のかわりに DBUriType のインスタンスを作成するように構成できます。

```
begin
 -- register a new handler for the dburi prefix..
urifactory.registerHandler('http://machine-name/servlets/oradb'
        ,'SYS','DBUriTYPE', true,true);
end;
/
```

セッションでこのブロックを実行すると、そのセッションで UriFactory.getUri() をコールするたびに、その接頭辞を含む HTTP の URL に対して自動的に DBUriType のインスタンスが作成されます。

> **参照：** DBUriFactory パッケージのすべてのファンクションの詳細は、『Oracle9*i* XML API リファレンス - XDK および Oracle XML DB』を参照してください。

第 V 部

Oracle XML DB Repository: フォルダリング、セキュリティおよびプロトコル

第 V 部では、Oracle XML DB Repository について説明します。データのバージョンを管理する方法、セキュリティを実装および管理する方法、および関連付けられた Oracle XML DB API を使用してリポジトリのデータにアクセスおよび操作する方法を説明します。

第 V 部に含まれる章は、次のとおりです。

13

Oracle XML DB Foldering

この章では、FTP、HTTP、WebDAV などの標準プロトコルおよびその他の Oracle XML DB Resource API を使用して、Oracle XML DB Repository のデータにアクセスする方法を説明します。また、リポジトリ・データにアクセスおよび操作するための SQL メカニズムとして RESOURCE_VIEW および PATH_VIEW を使用する方法も説明します。この章では、様々な Resource API を使用したリポジトリ操作を比較する表を示します。

この章の内容は次のとおりです。

Oracle XML DB Foldering の概要

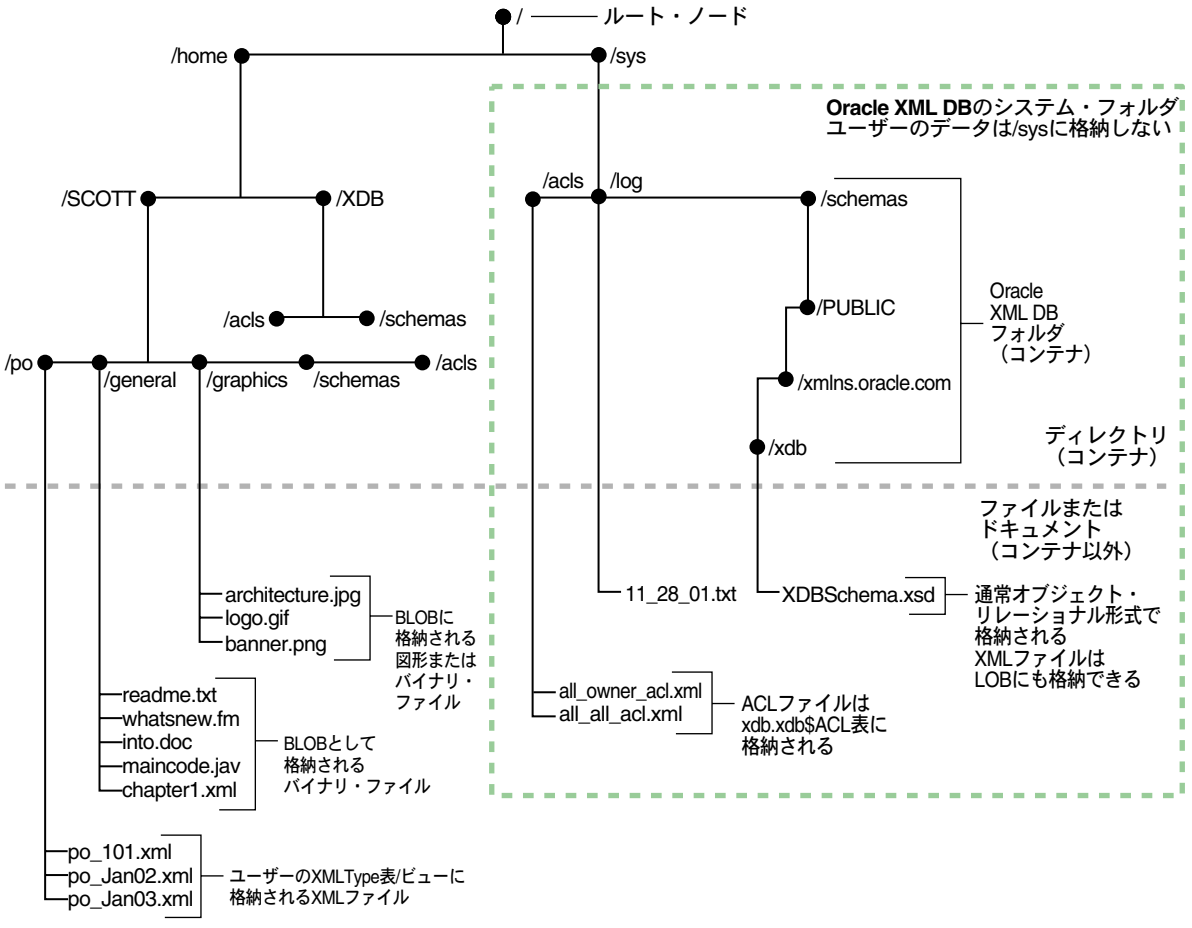
Oracle XML DB Foldering 機能を使用すると、従来のリレーショナル・データベース構造ではなく、階層構造のデータベースにコンテンツを格納できます。

図 13-1 に、Oracle XML DB Repository のフォルダおよびファイルの典型的なツリーを示す階層構造の例を示します。ツリーの上部は、ルート・フォルダ（/）です。

フォルダリングによって、アプリケーションでは、データベース・コンテンツがファイル・システムに格納されている場合と同様に、FTP、HTTP および WebDAV の標準プロトコルを使用して、データベース内の階層的に索引付けられたコンテンツにアクセスできます。

この章では、標準のプロトコルを使用して、Oracle XML DB Repository フォルダのデータにアクセスする方法の概要を説明します。今回のリリースでは、Java、SQL および PL/SQL を使用してリポジトリ・オブジェクト階層にアクセスできる API が他にもあります。

図 13-1 Oracle XML リポジトリの階層構造を示す典型的なフォルダ・ツリー



注意： ディレクトリ /sys は、Oracle XML DB によって、システム定義の XML Schema や ACL などをメンテナンスするために使用されます。一般的な制限事項は次のとおりです。

- /sys ディレクトリの下にデータを格納しないでください。
- /sys ディレクトリ内のコンテンツを変更しないでください。

参照： 次の章を参照してください。

- 第 15 章「RESOURCE_VIEW および PATH_VIEW」
- 第 16 章「Oracle XML DB Resource API for PL/SQL（DBMS_XDB）」
- 第 17 章「Oracle XML DB Resource API for Java」
- 第 19 章「FTP、HTTP および WebDAV プロトコルの 使用」

Oracle XML DB Repository

Oracle XML DB Repository（リポジトリ）は、すべての XML Schema およびデータベース・スキーマに対する一連のデータベース・オブジェクトです。これらのデータベース・オブジェクトはパス名にマップされます。リポジトリは、単一のルート・ノード（/）を持ち、リソースを連結、分岐した非環式で図式化したものです。図式内の各リソースは、関連付けられた 1 つ以上のパス名を持ちます。

リポジトリは、ファイルではなくオブジェクトのファイル・システムと考えられます。

注意： リポジトリは、指定されたリソースへの複数のリンクをサポートします。

リポジトリの用語

次に、Oracle XML DB Repository で使用される用語について説明します。

- リソース : 階層内のすべてのオブジェクトまたはノードです。リソースは URL で識別されます。13-6 ページの「Oracle XML DB リソース」を参照してください。
- フォルダ : リソースのコレクションを格納できる階層内のノード（またはディレクトリ）です。フォルダは、リソースでもあります。
- パス名 : ルート要素（最初の /）、要素セパレータ（/）および様々なサブ要素（またはパス要素）で構成された階層形式の名前です。パス要素は、Oracle XML DB で特別な意味を持つ \ および / を除いて、データベース・キャラクタ・セットのすべての文字で構成できます。スラッシュは、パス名内のデフォルトの名前セパレータであり、バックスラッシュは、エスケープ文字として使用されます。Oracle XML DB 構成ファイル `xdbconfig.xml` には、パス名（`<invalid-pathname-chars>`）内に表示されない場合があるユーザー定義の文字のリストも含まれます。
- リソース名またはリンク名 : 親フォルダ内のリソースの名前です。リソース名は、フォルダ内で一意（今回のリリースでは大 / 小文字が区別される）である必要があります。リソース名のキャラクタ・セットは、常に UTF-8（NVARCHAR）です。
- コンテンツ : リソースの本体です。これは、ファイルのようなリソースを処理する場合にそのコンテンツを要求すると戻されます。コンテンツは、常に `XMLType` です。

- `XDBBinary`: バイナリ・データを含む Oracle XML DB スキーマで定義された XML 要素です。`XDBBinary` 要素は、非構造化バイナリ・データが Oracle XML DB にアップロードされたときに、リポジトリに格納されます。
- アクセス制御リスト（ACL）：リソースへのアクセスを制限します。Oracle XML DB は、ACL を使用して任意の Oracle XML DB リソース（Oracle XML DB ファイル・システムの階層にマップされたすべての `XMLType` オブジェクト）へのアクセスを制限します。

> **参照：** 第 18 章「Oracle XML DB リソースのセキュリティ」を参照してください。

表 13-1 に示すとおり、Oracle XML DB で使用される多くの用語には、他のコンテキストで使用される共通の同義語が存在します。

表 13-1 Oracle XML DB Foldering の用語の同義語

同義語	Oracle XML DB Foldering の用語	用途
コレクション	フォルダ	WebDAV
ディレクトリ	フォルダ	オペレーティング・システム
権限	権限	権限
権利	権限	様々
WebDAV フォルダ	フォルダ	Web フォルダ
ロール	グループ	アクセス制御
リリース	バージョン	RCS、CVS
ファイル・システム	リポジトリ	オペレーティング・システム
階層	リポジトリ	様々
ファイル	リソース	オペレーティング・システム
バインド	リンク	WebDAV

Oracle XML DB リソース

Oracle XML DB リソースは、Oracle XML DB によって定義されている xdbresource.xsd スキーマに準拠します。リソースの要素には、作成日、変更日、WebDAV ロック、所有者、ACL、言語、キャラクタ・セットなどの WebDAV 定義のプロパティを永続的に格納するために必要な要素が含まれます。

リソース索引の Contents 要素

リソース索引には、リソースのコンテンツを含む、Contents という特別な要素が含まれます。

any 要素

あるリソースの XML Schema では、maxOccurs が無制限の any 要素も定義します。この場合、Oracle XML DB の XML 名前空間外のすべての要素を含むことができます。これによって、任意のインスタンス定義のプロパティをリソースと関連付けることができます。

リポジトリ・データが格納される場所

Oracle XML DB は、リポジトリ・データを Oracle XML DB データベース・スキーマの一連の表および索引に格納します。これらの表はアクセス可能です。XML Schema を登録し、Oracle XML DB によって表が生成されるように要求した場合、表はユーザーのデータベース・スキーマに作成されます。これは、その表が参照または変更可能であることを意味します。ただし、他のユーザーは、表を参照する権限を明示的に付与されないかぎり、その表を参照できません。

生成された表の名前

生成された表の名前は、Oracle XML DB によって割り当てられ、ユーザーの XML Schema 文書（またはデフォルトの XML Schema 文書）の xdb:defaultTable=XXX 属性を検索することによって取得できます。XML Schema を登録する場合、ユーザー独自の表名を指定して、Oracle XML DB によって作成されたデフォルト名をオーバーライドすることもできます。

> **参照：** 5-13 ページの「登録された XML Schema を使用する場合のガイドライン」を参照してください。

リソース用の構造化記憶域の定義

リソース用の構造化記憶域を定義する必要があるアプリケーションでは、次のいずれかの操作で行います。

- Oracle XML DB リソース・タイプのサブクラス化。Oracle XML DB リソースのサブクラス化には、XDB$RESOURCE 表に対する権限が必要です。
- 参照可能な登録済 XML Schema に準拠するデータの格納。

参照： 第 5 章「XMLType の構造化されたマッピング」を参照してください。

パス名の解決

フォルダをその子に関連付けるデータは、Oracle XML DB の階層索引によって管理されます。これによって、オペレーティング・システムのファイル・システムで使用されるディレクトリ・メカニズムと同様に、パス名を評価するための高速なメカニズムが提供されます。

フォルダであるリソースでは、`Container` 属性が TRUE に設定されます。

フォルダ内のリソース名を解決するには、現行のユーザーに次の権限が必要です。

- フォルダに対する `resolve` 権限
- フォルダのリソースに対する `read-properties` 権限

ユーザーがこれらの権限を所有していない場合、`access denied` エラーが発生します。そのリソースに対する `read-properties` 権限が拒否されている場合、フォルダのリストおよびその他の問合せによって行は戻されません。

注意： パス名の解決におけるエラー処理では、ファイル・システムとの互換性に対して、無効なリソース名とフォルダでないリソースが区別されます。Oracle XML DB リソースは、SQL を使用してリポジトリの外側からアクセスできるため、リソースを含むフォルダの読込みアクセスを制限しても、そのリソースへの読込みアクセスは制限されません。

リソースの削除

リンクを削除すると、そのリンクがリソースへの最後のリンクで、そのリソースがバージョニングされていない場合のみ、リンクが指しているリソースが削除されます。Oracle XML DB Repository のリンクは、UNIX の「ハード・リンク」に類似しています。

参照： 15-13 ページの「リポジトリ・リソースの削除例」を参照してください。

Oracle XML DB Repository のリソースへのアクセス

Oracle XML DB は、リソースにアクセスするための 2 つの方法を提供しています。

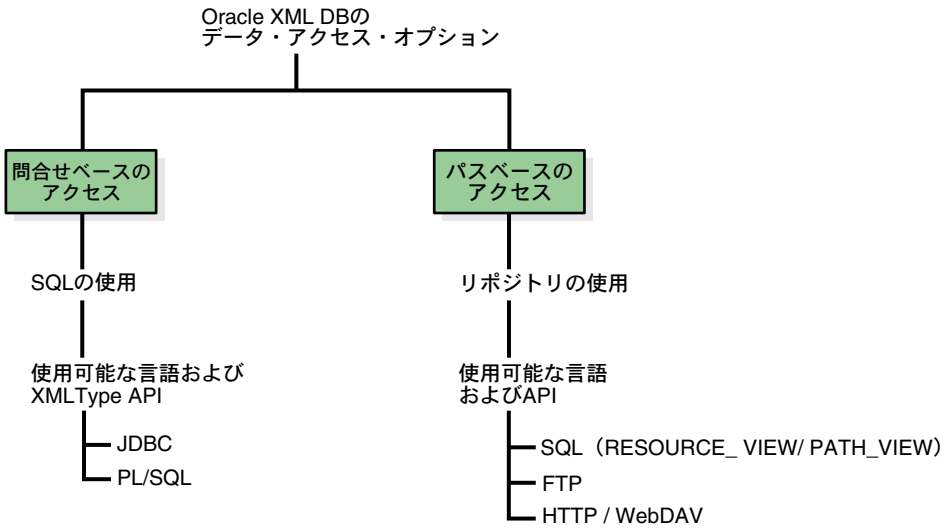
- 「ナビゲーショナル・アクセスまたはパス・アクセス」（13-9 ページ）
  オブジェクトまたはリソースの階層索引を使用して、Oracle XML DB のコンテンツにナビゲーショナル・アクセスまたはパス・アクセスできます。各リソースは、階層内の場所を反映する、1 つ以上の一意のパス名を持ちます。ナビゲーショナル・アクセスを使用して、表領域内のオブジェクトの場所に関係なく、データベースのすべてのオブジェクトを参照できます。

- 「問合せベースのアクセス」（13-12 ページ）
  リソースのプロパティおよびパス名を公開し、階層アクセス演算子を Oracle XML DB スキーマへマップする一連のビューを使用して、リポジトリに SQL アクセスできます。

図 13-2 に、Oracle XML DB のデータ・アクセス・オプションを示します。データ・アクセス・オプションの選択の詳細は、2-7 ページの「Oracle XML DB アプリケーションの設計 : b. アクセス・モデル」を参照してください。

> **参照：** 表 13-3「Oracle XML DB Repository へのアクセス :API オプション」を参照してください。

図 13-2 Oracle XML DB Repository のデータ・アクセス・オプション



Uniform Resource Locator（URL）を使用して、Oracle XML DB リソースにアクセスできます。URL には、オブジェクトのホスト名、プロトコル情報、パス名およびリソース名が含まれます。

ナビゲーショナル・アクセスまたはパス・アクセス

Oracle XML DB のフォルダは、多くのオペレーティング・システムによって使用されている、標準的なプロトコルをサポートしています。これによって、Oracle XML DB フォルダは、サポートされているオペレーティング・システム環境で、ネイティブなフォルダまたはディレクトリと同様に機能できます。たとえば、次の操作を実行できます。

- Windows エクスプローラを使用して、Windows NT ファイル・システムの他のディレクトリまたはリソースにアクセスする場合と同じ方法で、Oracle XML DB のフォルダおよびリソースを開いたり、アクセスできます（図 13-3 を参照）。
- Web フォルダを参照する場合などは、Internet Explorer ブラウザから HTTP または WebDAV を使用して、リポジトリ・データにアクセスできます（図 13-4 を参照）。

図 13-3 Windows エクスプローラでの Oracle XML DB のフォルダ

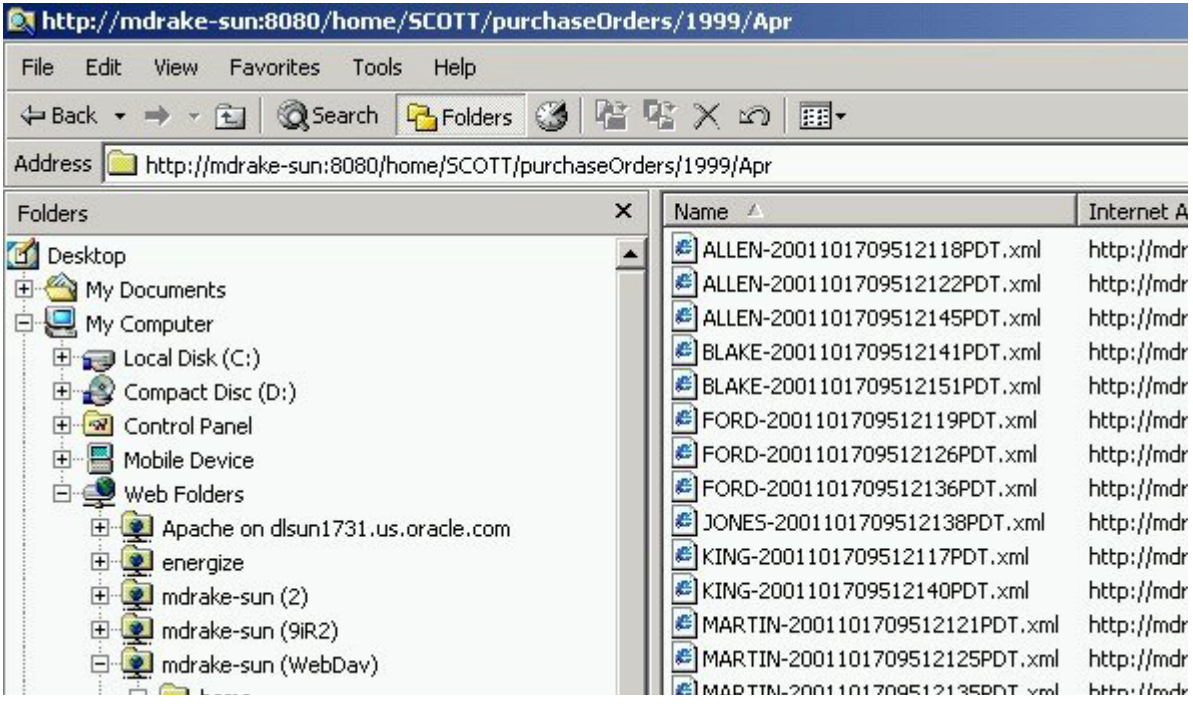
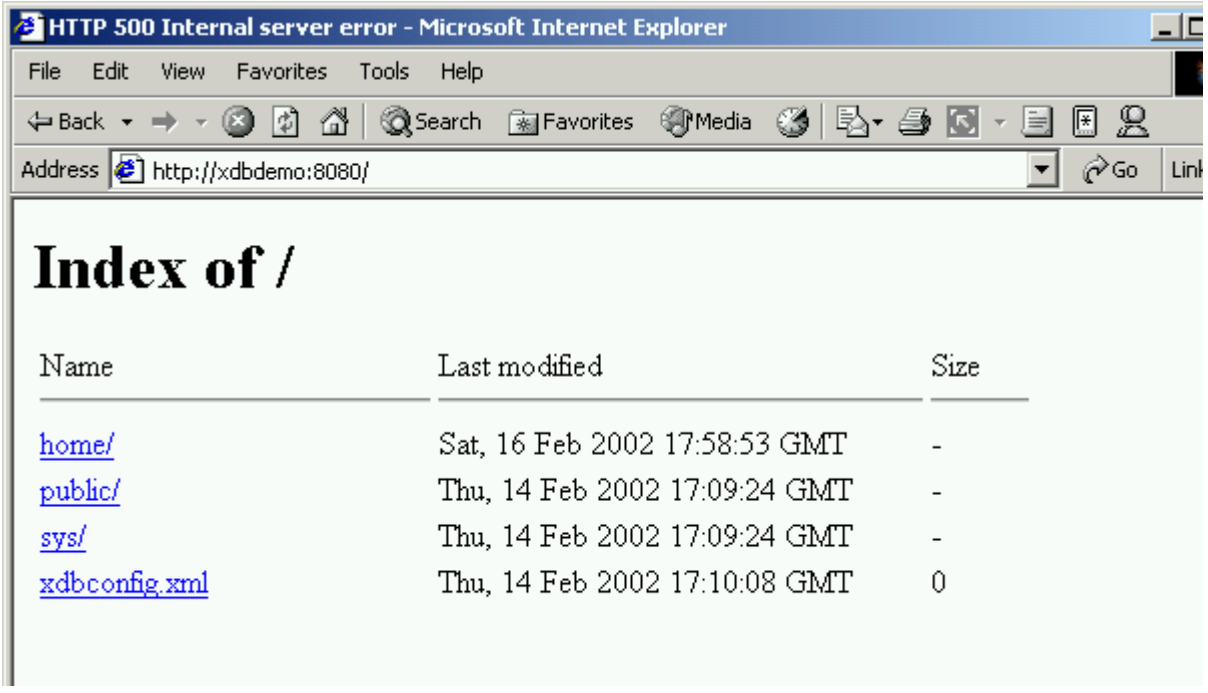


図 13-4 HTTP や WebDAV およびナビゲーショナル・アクセスを使用した IE ブラウザからリポジトリ・データへのアクセス : Web フォルダの参照



インターネット・プロトコルを使用した Oracle XML DB リソースへのアクセス

Oracle Net Services では、データベース・リソースへのアクセス方法が提供されます。インターネット・プロトコルをサポートしている Oracle XML DB では、データベース・リソースにアクセスする別の方法が提供されます。

Oracle XML DB のプロトコル・アクセスを使用できる場所

Oracle Net Services は、レコード指向のデータに対して最適化されています。インターネット・プロトコルは、バイナリ・ファイルや XML テキスト・ドキュメントなどのストリーム指向のデータ用に設計されています。

次のような場合は、Oracle Net Services より Oracle XML DB のプロトコル・アクセスを使用することをお薦めします。

- ファイル・システムのようなデータベースを使用した、ファイル指向アプリケーションからデータベースへの直接アクセス

- 均一なデータ・アクセス方法（MS SQL Server、Exchange、Notes、多くの XML データベース、株価情報サービス、ニュース配信を含む多くのデータ・サーバーによってサポートされている HTTP を介した XML など）を必要とする異機種間アプリケーション・サーバー環境
- XML テキストとしてのデータを必要とするアプリケーション・サーバー環境
- 多くのアプリケーション処理を必要とせずに、クライアント側の XSL を使用して、データグラムをフォーマットする Web アプリケーション
- データベース内で実行する Java サーブレットを使用する Web アプリケーション
- XML 指向のストアド・プロシージャに対する Web アクセス

プロトコル・アクセスのコール順序

Oracle XML DB では、次の手順でプロトコル・アクセスが実行されます。

1. 接続オブジェクトが確立されます。また、プロトコルによってリクエストの一部の読込みが決定される場合があります。
2. プロトコルによって、認証済のユーザーが既存のセッションを再利用できるか、または接続を再認証する必要があるか（通常はこちら）が決定されます。
3. 既存のセッションがセッション・プールから取り出されるか、または新しいセッションが作成されます。
4. 認証が行われず、リクエストが HTTP の `Get` または `Head` の場合、セッションは `ANONYMOUS` ユーザーとして実行されます。セッションがすでに `ANONYMOUS` ユーザーとして認証されている場合は、そのまま既存のセッションを再利用できます。認証が行われている場合は、データベースの再認証ルーチンを使用して、接続が認証されます。
5. リクエストが解析されます。
6. 要求されたパス名がサーブレットにマップされる場合（HTTP の場合のみ）、サーブレットは JVM を使用して起動されます。サーブレット・コードがレスポンス・ストリームに対するレスポンスを書き出すか、または `XMLType` インスタンスにレスポンスを書き出すように要求します。

Oracle XML DB リソースの取出し

プロトコルがリソースの取出しを示した場合、リソースへのパス名が解決されます。フェッチされているリソースは、常に XML として配信されます。ただし、XML バイナリ・データ型として定義済の要素である `XDBBinary` 要素を含むリソースで、そのコンテンツが RAW フォームで配信されるものは除きます。

Oracle XML DB リソースの格納

プロトコルがリソースを格納する必要があることを示した場合、Oracle XML DB は、文書のファイル名の拡張子が .xml、.xsl、.xsd などであることを確認します。文書が XML である場合、事前解析が実行されます。その場合、文書のルート要素の XML `schemaLocation` および `namespace` を判断するために十分なリソースが読み込まれます。この場所を使用して、その `schemaLocation` の URL で登録されたスキーマを検索します。登録されたスキーマが、現在の文書のルート要素に対する定義とともに配置されている場合、その要素に対して指定されたデフォルト表を使用して、リソースのコンテンツが格納されます。

インターネット・プロトコルおよび XMLType の使用 : ストリームへの XMLType の直接書込み

Oracle XML DB は、`XMLType` に対して Java メソッド `writeToStream()` を使用することによって、`XMLType` レベルでインターネット・プロトコルをサポートします。このメソッドはネイティブに実装され、`XMLType` データをプロトコルのリクエスト・ストリームに直接書き込みます。これによって、Java データ型を介したデータベース・データの変換のオーバーヘッド、Java オブジェクトの作成および JVM の実行コストが回避され、パフォーマンスが大幅に向上します。パフォーマンスが大幅に向上するのは、特に、Java コードが多くのリーフ要素を検索せずにルートに近い XML 要素ツリーのみを処理し、作成される Java オブジェクトの数が最小化される場合です。

> **参照：** 第 19 章「FTP、HTTP および WebDAV プロトコルの 使用」を参照してください。

問合せベースのアクセス

Oracle XML DB は、リポジトリ・データへの SQL アクセス可能な 2 つのビューを提供します。

- `PATH_VIEW`
- `RESOURCE_VIEW`

表 13-2 に、`PATH_VIEW` と `RESOURCE_VIEW` の違いを示します。

表 13-2 PATH_VIEW と RESOURCE_VIEW の違い

PATH_VIEW	RESOURCE_VIEW
リンク・プロパティを含みます。	リンク・プロパティを含みません。
リソース・プロパティおよびパス名を含みます。	リソース・プロパティおよびパス名を含みます。
リポジトリの一意の各パスに 1 つの行を持ちます。	リポジトリの各リソースに 1 つの行を持ちます。

注意： 各リソースは、複数のパスを持つことができます。

RESOURCE_VIEW の単一のパスは、リソースを参照する多くの選択可能なパスから任意に選択されます。Oracle XML DB は、アプリケーションで、特定のフォルダに含まれるリソースを再帰的に検索したり、リソースの深さを取得することができる UNDER_PATH などの演算子を提供します。ビューの各行は、XMLType です。

ビュー上で DML を使用すると、リソースのプロパティおよびコンテンツの挿入、名前の変更、削除および更新ができます。その他の操作（既存のリソースへのリンクの作成など）に対しては、プログラム API を使用する必要があります。

参照：

- リポジトリへの SQL アクセスの詳細は、第 15 章「RESOURCE_VIEW および PATH_VIEW」を参照してください。
- 第 18 章「Oracle XML DB リソースのセキュリティ」も参照してください。
- 『Oracle9*i* XML API リファレンス - XDK および Oracle XML DB』も参照してください。

サーブレットを使用したリポジトリ・データへのアクセス

Oracle XML DB は、Java サーブレット API バージョン 2.2 を実装します。ただし、次の例外があります。

- すべてのサーブレットが配布可能である必要があります。これらのサーブレットは、異なる VM で実行可能である必要があります。
- WAR ファイルおよび web.xml ファイルはサポートされません。Oracle XML DB は、このファイルの XML 構成のサブセットをサポートしています。XSLT スタイルシートを web.xml に適用して、サーブレットの定義を生成できます。外部ツールを使用して、web.xml ファイルに定義されたサーブレットのデータベース・ロールを作成する必要があります。
- JavaServer Pages（JSP）サポートをサーブレットとしてインストールし、手動で構成できます。
- HTTPSession および関連するクラスはサポートされません。
- 1 つのサーブレット・コンテキスト（1 つの Web アプリケーション）のみがサポートされます。

参照： 第 20 章「Java での Oracle XML DB アプリケーションの作成」を参照してください。

Oracle XML DB Repository のリソースに格納されたデータへのアクセス

Oracle XML DB Repository のリソースに格納されたデータへは、次の 3 つの主要な方法でアクセスできます。

- Oracle XML DB Resource API for Java
- Oracle XML DB Resource View API と Oracle XML DB Resource API for PL/SQL の組合せ
- インターネット・プロトコル（HTTP、WebDAV および FTP）および Oracle XML DB Protocol Server

表 13-3 に、一般的な Oracle XML DB Repository の操作を示し、その 3 つの各方法を使用してこれらの操作を実行する方法を説明します。この表では、3 つの方法に共通の機能を示します。すべての方法が、特定の一連のタスクに対して同等に適しているわけではないことに注意してください。

参照： 次の章またはマニュアルを参照してください。

- 第 15 章「RESOURCE_VIEW および PATH_VIEW」
- 第 16 章「Oracle XML DB Resource API for PL/SQL（DBMS_XDB）」
- 第 17 章「Oracle XML DB Resource API for Java」
- 第 19 章「FTP、HTTP および WebDAV プロトコルの 使用」
- 『Oracle9*i* XML API リファレンス - XDK および Oracle XML DB』

表 13-3 Oracle XML DB Repository へのアクセス :API オプション

データ・アクセス操作	問合せベースのアクセス : RESOURCE_VIEW API パスベースのアクセス : Resource API for PL/SQL	パスベースのアクセス : プロトコル
リソースの作成	INSERT INTO PATH_VIEW VALUES (path, res, linkprop) 「DBMS_XDB.CreateResource」 も参照。	HTTP PUT; FTP PUT
パス名を使用したリソースのコンテンツの更新	UPDATE RESOURCE_VIEW SET resource = updateXML(res, '/Resource/Contents', lob) WHERE EQUALS_PATH(res, :path) > 0	HTTP PUT; FTP PUT

表 13-3 Oracle XML DB Repository へのアクセス :API オプション（続き）

データ・アクセス操作	問合せベースのアクセス : RESOURCE_VIEW API パスベースのアクセス : Resource API for PL/SQL	パスベースのアクセス : プロトコル
パス名を使用したリソースのプロパティの更新	UPDATE RESOURCE_VIEW SET resource = updateXML(res, '/Resource/propname', newval, '/Resource/propname2', newval2, …) WHERE EQUALS_PATH(res, :path) > 0	WebDAV PROPPATCH; FTP なし
リソースの ACL の更新	UPDATE RESOURCE_VIEW SET resource = updateXML(res, '/ Resource/ACL', XMLType) WHERE EQUALS_PATH(res, :path) > 0	なし
リソースのリンクの解除または削除（最後のリンクの場合）	DELETE FROM RESOURCE_VIEW WHERE EQUALS_PATH(res, :path) > 0	HTTP DELETE; FTP DELETE
リソースへのすべてのリンクの強制削除	DELETE FROM PATH_VIEW WHERE extractValue(res,'display_name') = 'My resource';	なし
リソースまたはフォルダの移動	UPDATE PATH_VIEW SET path = newpath WHERE EQUALS_PATH(res, :path) > 0	WebDAV MOVE; FTP RENAME
リソースまたはフォルダのコピー	INSERT INTO PATH_VIEW SELECT:newpath, resource, link FROM PATH_VIEW WHERE EQUALS_PATH(res,:oldpath)> 0	WebDAV COPY; FTP なし
既存のリソースへのリンクの作成	Call dbms_xdb.Link(srcpath IN VARCHAR2, linkfolder IN VARCHAR2, linkname IN VARCHAR2);	なし
パス名ごとのリソース・コンテンツのバイナリまたはテキスト表現の取得	SELECT p.res.extract ('/Resource/Contents') FROM RESOURCE_VIEW p WHERE EQUALS_PATH(res, :path) > 0 SELECT XDBUriType(:path).getBLob() FROM dual;	HTTP GET; FTP GET
パス名ごとのリソース・コンテンツの XMLType 表現の取得	SELECT extract(res, '/Resource/Contents/*') FROM RESOURCE_VIEW p WHERE EQUALS_PATH(Res, :path) > 0	なし
パス名ごとのリソース・プロパティの取得	SELECT extractValue(res, '/Resource/XXX') FROM RESOURCE_VIEW WHERE EQUALS_PATH(res,:path) > 0	WebDAV PROPFIND (depth = 0); FTP なし

表 13-3 Oracle XML DB Repository へのアクセス :API オプション（続き）

データ・アクセス操作	問合せベースのアクセス : RESOURCE_VIEW API パスベースのアクセス : Resource API for PL/SQL	パスベースのアクセス : プロトコル
ディレクトリのリスト	SELECT * FROM PATH_VIEW WHERE UNDER_PATH(res, :path, 1) > 0	WebDAV PROPFIND (depth = 1); FTP LS
フォルダの作成	Call dbms_xdb.createFolder(VARCHAR2)	WebDAV MKCOL; FTP MKDIR
フォルダのリンクの解除	DELETE FROM PATH_VIEW WHERE EQUALS_PATH(res, :path) > 0	HTTP DELETE; FTP RMDIR
フォルダおよびそのフォルダに対するすべてのリンクの強制削除	Call dbms_xdb.deleteFolder(VARCHAR2)	なし
行ロックされたリソースの取得	SELECT … FROM RESOURCE_VIEW FOR UPDATE …;	なし
リソースへの WebDAV ロック	DBMS_XDB.LockResource(path, true, true);	WebDAV LOCK; FTP: QUOTE LOCK
WebDAV ロックの解除	DBMS_XDB.GetLockToken(:path, deltoken); DBMS_XDB.UnlockResource(path, deltoken);	WebDAV UNLOCK; QUOTE UNLOCK
変更のコミット	COMMIT;	要求終了時に自動的にコミット
変更のロールバック	ROLLBACK;	なし

リソースへのアクセスの管理および制御

Oracle XML DB のフォルダおよびリソースに対するアクセス制御権限を設定できます。

参照：

- Oracle XML DB のフォルダに対するアクセス制御の使用の詳細は、第 18 章「Oracle XML DB リソースのセキュリティ」を参照してください。
- 第 21 章「Oracle Enterprise Manager を使用した Oracle XML DB の管理」も参照してください。
- 『Oracle9*i* XML API リファレンス - XDK および Oracle XML DB』の DBMS_XDB に関する章も参照してください。

リソースのメタデータ・プロパティの拡張

Oracle XML DB リソースは、XML Schema の XDBResource.xsd によって記述されます。この XML Schema の詳細は、付録 G「設定スクリプトの例および Oracle XML DB によって提供される XML Schema」を参照してください。XDBResource.xsd は、Owner、CreationDate などのメタデータ・プロパティの固定セットを宣言します。リソースの作成または更新中に、これらのメタデータ属性の値を指定できます。

リソースを持つ追加メタデータとして独自の（ユーザー定義の）タグを格納するための 2 つのオプションがあります。このメタデータは、リソースの XML Schema によって定義されないメタデータ・プロパティです。

- **オプション 1: CLOB（ResExtra 要素）としての追加メタデータの格納。**デフォルトのスキーマには、最上位の any 要素（maxOccurs= "unbounded" で宣言される）が含まれるため、すべての妥当な XML データがリソース・ドキュメントの一部として許可され、RESEXTRA CLOB 列に格納されます。

注意： ユーザー・レベルのメタデータを表す XML は、XDBResource 名前空間以外の名前空間内に存在する必要があります。

```
<Resource xmlns="http://xmlns.oracle.com/xdb/XDBResource.xsd"
    <Owner>SCOTT</Owner>
    ... <!-- other system defined metadata -->
   <!-- User Metadata (appearing within different namespace) -->
   <ResExtra>
      <myCustomAttrs xmlns="http://www.example.com/customattr">
           <attr1>value1</attr1>
           <attr2>value2</attr2>
      </myCustomAttrs>
     </ResExtra>
```

```
    <!-- contents of the resource -->
    <Contents>
        ...
     </Contents>
</Resource>
```

この方法は、リソース・メタデータに対する非定型の拡張の場合に機能しますが、ユーザー・メタデータが CLOB に格納されるため、ユーザー・メタデータの問合せの実行性および更新の実行性に影響します。

- **オプション 2: リソースの XML Schema の拡張**。XML Schema 仕様で指定された方法を使用して、リソースの XML Schema を拡張できます。ResourceType を拡張した新しい XML Schema を登録できます。これによって、XDB$RESOURCE_T オブジェクト型の下にオブジェクト・サブタイプが作成されるため、XDB$RESOURCE 表に新しい列が追加されます。

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:xdbres="http://xmlns.oracle.com/xdb/XDBResource.xsd"
    <complexType name="myCustomResourceType">
      <complexContent>
        <extension base="xdbres:ResourceType>
          <element name="custom-attr1" type="string">
          ...
        </extension>
      </complexContent>
   </complexType>
</schema>
```

FAQ: XML DB リポジトリ

XML リポジトリの階層索引が機能しないのはなぜですか？

XML DB リリース 2（9.2）では、階層索引を使用して次の問合せが機能しました。

```
SELECT XDBURITYPE(ANY_PATH).GETXML()
   FROM RESOURCE_VIEW
   WHERE CONTAINS(RES,'MYXML') >0;
```

この問合せは、最新のリリースでは機能せず、列に索引が作成されていないというエラーが発生します。階層索引を使用して XML リポジトリ内のドキュメントのコンテンツを検索するには、どうすればよいでしょうか？マニュアルのある箇所でパス名を索引付けする階層索引について説明していますが、これがどのタイプの索引かについては言及していません。また、DBMS_XDBT パッケージの説明では、前述の説明に類似した、Oracle XML DB Repository 階層に対する ConText 索引について説明しています。

回答 : リポジトリに ConText 索引を作成してください。Contains() は、評価に ConText 索引が必要であることに注意してください。xdb$resource 上に ConText 索引を作成するには、次のいずれかの操作を実行します。

a. DBMS_XDBT（rdbms/admin の dbmsxdbt.sql）パッケージを使用します。

b. 次のコマンドを実行して、ConText 索引を明示的に作成します。

```
CREATE INDEX xdb.ctxi ON xdb.xdb$resource x (value(x))
   indextype is ctxsys.context;
```

(a) ではリポジトリ内のバイナリ・コンテンツが考慮され、適切にフィルタされるため、(a) の方法を使用することをお薦めします。(b) の方法では、リポジトリ内の XML およびテキスト・コンテンツのみが索引付けされます。

xdb$resource 表には、次の 2 つの異なる索引があります。

- **階層索引**。これは常に存在します。階層索引の再作成プロシージャは、この索引の再作成に使用します。
- **ConText 索引。**これはオプションの索引であり、必要に応じて DBA が作成できます。Contains() 問合せを実行するには、この索引が必要です。

14

Oracle XML DB Versioning

この章では、Oracle XML DB リソースのバージョンを作成および管理する方法を説明します。この章の内容は次のとおりです。

- Oracle XML DB Versioning の概要
- バージョン管理されたリソース（VCR）の作成
- VCR のアクセス制御およびセキュリティ
- FAQ: Oracle XML DB Versioning

Oracle XML DB Versioning の概要

Oracle XML DB Versioning によって、Oracle XML DB のリソースの異なるバージョンを作成および管理する方法が提供されます。Oracle9*i* データベースの以前のリリースでは、リソース（表や列など）の更新後、以前のコンテンツおよびプロパティが失われます。Oracle XML DB Versioning では、データベースにリソースのバージョンを格納することによって、更新の発行時に、古いリソースのコンテンツおよびプロパティが保持されます。

Oracle XML DB は、リソースをバージョン管理し、リソースの異なるバージョンを取り出すための PL/SQL パッケージ `DBMS_XDB_VERSION` を提供しています。

Oracle XML DB Versioning 機能

Oracle XML DB Versioning によって、バージョン管理されている Oracle XML DB リソース（VCR）のすべての変更を追跡できます。次の項では、これらの機能の詳細を説明します。Oracle XML DB Versioning 機能には、次の機能が含まれます。

- **リソースのバージョン管理 :** Oracle XML DB リソースのバージョン管理を有効または無効にするオプションがあります。14-4 ページの「バージョン管理されたリソース（VCR）の作成」を参照してください。
- **バージョン管理されたリソースのプロセスの更新 :** Oracle XML DB は、バージョン管理されたリソースを更新する場合に、リソースの新しいバージョンも作成します。このバージョンは、バージョン管理されたリソースを削除しても、データベースから削除されません。14-7 ページの「バージョン管理されたリソース（VCR）の更新」を参照してください。
- **バージョン管理されたリソースのロード :** これは、パス名を使用して Oracle XML DB の通常の他のリソースをロードする場合に類似しています。14-4 ページの「バージョン管理されたリソース（VCR）の作成」を参照してください。
- **リソースのバージョンのロード :** リソースのバージョンをロードするには、最初にそのバージョンのリソース・オブジェクト ID を検索し、その ID を使用してバージョンをロードする必要があります。リソース・オブジェクト ID は、リソース・バージョンの履歴またはバージョン管理されたリソース自体から検索できます。14-4 ページの「Oracle XML DB リソースの ID およびパス名」を参照してください。

注意 : 今回のリリースでは、Oracle XML DB のバージョニングは、Oracle XML DB リソースのバージョン制御をサポートします。Oracle9*i* データベースに含まれるユーザー定義の表またはデータのバージョン管理はサポートされません。

参照：『Oracle9*i* XML API リファレンス - XDK および Oracle XML DB』を参照してください。

この章で使用される Oracle XML DB Versioning の用語

表 14-1 に、この章で使用される Oracle XML DB Versioning の用語を示します。

表 14-1 Oracle XML DB Versioning の用語

Oracle XML DB Versioning の用語	説明
バージョン管理	Oracle XML DB リソースに対するすべての変更の記録または履歴が格納または管理される場合、そのリソースはバージョン管理されているといいます。
バージョン対応リソース	バージョン対応リソースは、バージョン管理可能な Oracle XML DB リソースです。
バージョン管理されたリソース（VCR）	バージョン管理されたリソースは、バージョン管理された Oracle XML DB リソースです。この場合、VCR は、あるバージョンの Oracle XML DB リソースへの参照です。
バージョン・リソース	バージョン・リソースは、バージョン管理された Oracle XML DB リソースのバージョンです。バージョン・リソースは、読込み専用の Oracle XML DB リソースです。これは、更新または削除できません。ただし、バージョン履歴がシステムから削除されると、バージョン・リソースもシステムから削除されます。
チェックアウト・リソース	チェックアウト・リソースは、バージョン管理されたリソースがチェックアウトされた場合に作成される Oracle XML DB リソースです。
チェックアウト、チェックインおよびアンチェックアウト	チェックアウト、チェックインおよびアンチェックアウトは、Oracle XML DB リソースを更新するための操作です。バージョン管理されたリソースは、変更前にチェックアウトする必要があります。チェックイン操作を使用して、変更を永続的にします。アンチェックアウトを使用して、変更を無効にします。

Oracle XML DB リソースの ID およびパス名

Oracle XML DB リソース ID は、Oracle XML DB リソースに対する一意のシステム生成 ID です。リソース ID は、パス名を持たないリソースの識別に有効です。たとえば、バージョン・リソースは、パス名を持たないシステム生成のリソースです。
`GetResourceByResId()` 関数を使用して、任意のリソース・オブジェクト ID のリソースを取り出すことができます。

例 14-1 DBMS_XDB_VERSION.GetResourceByResId(): リソース 'home/index.html' で MakeVersioned を実行した場合の最初のバージョン ID の取得

```
declare
resid DBMS_XDB_VERSION.RESID_TYPE;
res XMLType;
begin
resid := DBMS_XDB_VERSION.MakeVersioned('/home/SCOTT/versample.html');
-- Obtain the resource
res := DBMS_XDB_VERSION.GetResoureceByResId(resid);
```

バージョン管理されたリソース（VCR）の作成

Oracle XML DB は、更新の履歴を自動的には保持しません。これは、履歴を保持する必要がない Oracle XML DB リソースが存在するためです。Oracle XML DB に要求を送信して、Oracle XML DB リソースをバージョン管理にする必要があります。今回のリリースでは、次のものを除いて、すべての Oracle XML DB リソースがバージョン対応リソースです。

- フォルダ（ディレクトリまたはコレクション）
- ACL

バージョン管理されたリソース（VCR）が作成されると、VCR の最初のバージョン・リソースが作成され、VCR は新しく作成されたバージョンへの参照になります。

14-5 ページの「バージョン・リソースまたは VCR バージョン」を参照してください。

例 14-2 DBMS_XDB_VERSION.MakeVersioned(): バージョン管理されたリソース（VCR）の作成

リソース '/home/SCOTT/versample.html' は、バージョン管理されたリソースになります。

```
declare
resid DBMS_XDB_VERSION.RESID_TYPE;
begin
resid := DBMS_XDB_VERSION.MakeVersioned('/home/SCOTT/versample.html');
end;
/
```

MakeVersioned() は、バージョン管理されたリソースの最初のバージョンのリソース ID を戻します。このバージョンは、リソース ID によって表されます（14-5 ページの「新しいバージョンのリソース ID」を参照）。

MakeVersioned() は、自動コミットの SQL 操作ではありません。ユーザーが操作をコミットする必要があります。

バージョン・リソースまたは VCR バージョン

Oracle XML DB は、バージョン・リソースのパス名は提供しません。ただし、バージョン・リソース ID は提供します。バージョン・リソースは読込み専用のリソースです。

バージョン ID は、次の項で説明する DBMS_XDB_VERSION パッケージのメソッドによって戻されます。

新しいバージョンのリソース ID

VCR がチェックインされると、新しいバージョン・リソースが作成され、そのリソース ID が戻されます。VCR はこの新しいバージョン・リソースへの参照であるため、バージョンのリソース ID は、VCR パス名を入力して DBMS_XDB.getResourceID() メソッドをコールすることによっても検索できます。

例 14-3 チェックイン後の新しいバージョンのリソース ID の取出し

次の例は、'/home/index.html' のチェックイン後、新しいバージョンのリソース ID を取得する方法を示します。

```
-- Declare a variable for resource id
declare
resid DBMS_XDB_VERSION.RESID_TYPE;
res XMLType;
begin
-- Get the id as user checks in.
resid := DBMS_XDB_VERSION.checkin('/home/SCOTT/versample.html');

-- Obtain the resource
res := DBMS_XDB_VERSION.GetResourceByResId(resid);
end;
/
```

例 14-4 Oracle XML DB: バージョン管理されたリソース（VCR）の作成および更新

```
-- Variable definitions.
declare
resid1 DBMS_XDB_VERSION.RESID_TYPE;
resid2 DBMS_XDB_VERSION.RESID_TYPE;
begin
-- Put a resource under version control.
resid1 := DBMS_XDB_VERSION.MakeVersioned('/home/SCOTT/versample.html');

-- Checkout to update contents of the VCR
DBMS_XDB_VERSION.Checkout('/home/SCOTT/versample.html');

-- Use resource_view to update versample.html
update resource_view
set res = sys.xmltype.createxml(
'<Resource xmlns="http://xmlns.oracle.com/xdb/XDBResource.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://xmlns.oracle.com/xdb/XDBResource.xsd
http://xmlns.oracle.com/xdb/XDBResource.xsd">
<Author>Jane Doe</Author>
<DisplayName>versample</DisplayName>
<Comment>Has this got updated or not ?? </Comment>
<Language>en</Language>
<CharacterSet>ASCII</CharacterSet>
<ContentType>text/plain</ContentType>
</Resource>')
where any_path = '/home/SCOTT/versample.html';

-- Checkin the change
resid2 := DBMS_XDB_VERSION.Checkin('/home/SCOTT/versample.html');
end;
/

-- At this point, you can download the first version with the resource object ID,
-- resid1 and download the second version with resid2.

-- Checkout to delete the VCR
DBMS_XDB_VERSION.Checkout('/home/SCOTT/versample.html');

-- Delete the VCR
delete from resource_view where any_path = '/home/SCOTT/versample.html';

-- Once the delete above is done, any reference
-- to the resource (that is, checkin, checkout,and so on, results in
-- ORA-31001: Invalid resource handle or path name "/home/SCOTT/versample.html"
```

バージョン管理されたリソース（VCR）へのアクセス

VCR も、通常のリソースと同様にパス名を持ちます。VCR へのアクセスは、Oracle XML DB の他のリソースへのアクセスと同じです。

バージョン管理されたリソース（VCR）の更新

通常の Oracle XML DB リソースに対する操作では、VCR をチェックアウトする必要はありません。VCR の更新には、通常の Oracle XML DB リソースの場合より多くの手順が必要です。

VCR のコンテンツおよびプロパティを更新する前に、リソースをチェックアウトします。更新を永続的にするには、リソースにチェックインする必要があります。これらのすべての操作は、自動コミットの SQL 操作ではありません。SQL トランザクションを明示的にコミットする必要があります。VCR を更新する手順は次のとおりです。

1. **リソースをチェックアウトします。**リソースをチェックアウトするには、そのリソースのパス名を Oracle XML DB に渡す必要があります。
2. **リソースを更新します。**リソースのコンテンツまたはプロパティのいずれかを更新できます。これらの機能は、Oracle XML DB でサポートされています。リソースの新しいバージョンは、そのリソースがチェックインされるまで作成されません。そのため、更新または削除は、リソースのチェックイン要求が実行されるまで永続的ではありません。
3. **リソースにチェックインまたはアンチェックアウトします**。VCR がチェックインされると、新しいバージョンが作成され、VCR は新しいバージョンと同じコンテンツおよびプロパティを持ちます。VCR がアンチェックアウトされると、VCR は変更されません。VCR は、古いバージョンと同じコンテンツおよびプロパティを持ちます。リソースにチェックインまたはアンチェックアウトするには、そのリソースのパス名を Oracle XML DB に渡す必要があります。パス名がチェックアウト後に更新された場合、新しいパス名を使用する必要があります。古いパス名を使用することはできません。

チェックアウト

Oracle9*i* データベース リリース 2（9.2）では、`DBMS_XDB_VERSION.CheckOut()` をコールすることによって、VCR のチェックアウト操作が実行されます。リソースの更新をコミットする必要がある場合、チェックアウト後にコミットすることをお薦めします。チェックアウト直後にコミットしないと、後でトランザクションをロールバックする必要がある場合に、更新内容が失われます。

例 14-5 VCR のチェックアウト

次に例を示します。

```
-- Resource '/home/SCOTT/versample.html' is checked out.
DBMS_XDB_VERSION.CheckOut('/home/SCOTT/versample.html');
```

チェックイン

Oracle9*i* データベース リリース 2（9.2）では、DBMS_XDB_VERSION.CheckIn() をコールすることによって、VCR のチェックイン操作が実行されます。チェックインには、リソースのパス名を指定します。このパス名は、チェックアウトに渡したパス名と同じである必要はありませんが、チェックインのパス名とチェックアウトのパス名は、同じリソースのものである必要があります。

例 14-6 VCR のチェックイン

次に例を示します。

```
-- Resource '/home/SCOTT/versample.html' is checked in.
declare
   resid DBMS_XDB_VERSION.RESID_TYPE;
begin
   resid := DBMS_XDB_VERSION.CheckIn('/home/SCOTT/versample.html');
end;
/
```

アンチェックアウト

Oracle9*i* データベース リリース 2（9.2）では、DBMS_XDB_VERSION.UncheckOut() をコールすることによって、VCR のアンチェックアウト操作が実行されます。このパス名は、チェックアウトに渡したパス名と同じである必要はありませんが、チェックインのパス名とチェックアウトのパス名は、同じリソースのものである必要があります。

例 14-7 VCR のアンチェックアウト

次に例を示します。

```
-- Resource '/home/SCOTT/versample.html' is unchecked out.
declare
   resid DBMS_XDB_VERSION.RESID_TYPE;
begin
   resid := DBMS_XDB_VERSION.UncheckOut('/home/SCOTT/versample.html');
end;
/
```

コンテンツおよびプロパティの更新

VCR のチェックアウト後、通常のリソースのコンテンツおよびプロパティを更新するためのすべての方法を、VCR に適用できます。

> **参照：** Oracle XML DB リソースの更新の詳細は、第 15 章「RESOURCE_VIEW および PATH_VIEW」を参照してください。

VCR のアクセス制御およびセキュリティ

VCR およびバージョン・リソースのアクセス制御は、通常のリソースと同じです。これらのリソースへのアクセスを要求すると、ACL が確認されます。

> **参照：** 第 18 章「Oracle XML DB リソースのセキュリティ」を参照してください。

バージョン・リソース

通常のリソースに MakeVersioned を実行すると、最初のバージョン・リソースが作成されます。この最初のバージョンの ACL は、元のリソースの ACL と同じです。チェックアウトされたリソースにチェックインすると、新しいバージョンが作成されます。この新しいバージョンの ACL は、チェックアウトされたリソースの ACL と同じです。バージョン・リソースの作成後、その ACL は変更できず、通常のリソースの ACL と同じ方法で使用されます。

最初のバージョンと同じ VCR の ACL

MakeVersioned を実行して VCR を作成すると、VCR の ACL は、リソースの最初のバージョンの ACL と同じになります。リソースにチェックインすると、新しいバージョンが作成され、VCR は、この新しいバージョンと同じコンテンツおよびプロパティ（ACL プロパティを含む）を持ちます。

表 14-2 に、DBMS_XDB_VERSION のサブプログラムを示します。

表 14-2 DBMS_XDB_VERSION のファンクションおよびプロシージャ

DBMS_XDB_VERSION のファンクションおよびプロシージャ	説明
ファンクション **MakeVersioned** MakeVersioned(pathname VARCHAR2) RETURN DBMS_XDB_VERSION.resid_type;	パス名が指定された通常のリソースをバージョン管理されたリソースに変換します。複数のパス名が同じリソースにバインドされている場合、そのリソースのコピーが作成され、指定されたパス名が新しく作成されたコピーにバインドされます。この新しいリソースは、バージョン管理されます。他のすべてのパス名は、引続き元のリソースを参照します。 pathname - バージョン管理するリソースのパス名。 戻り値 - VCR の最初のバージョン（ルート）のリソース ID。これは、自動コミットの SQL 操作ではありません。VCR に MakeVersioned をコールすることは正当な操作で、例外または警告のいずれも発生しません。フォルダ、バージョン・リソースおよび ACL に MakeVersioned を実行することは不正な操作で、リソースが存在しない場合、例外が発生します。

表 14-2 DBMS_XDB_VERSION のファンクションおよびプロシージャ （続き）

DBMS_XDB_VERSION のファンクションおよびプロシージャ	説明
プロシージャ **Checkout** Checkout(pathname VARCHAR2);	更新または削除前に VCR をチェックアウトします。 pathname - チェックアウトする VCR のパス名。これは、自動コミットの SQL 操作ではありません。同じ作業領域の 2 人のユーザーが、同時に同じ VCR をチェックアウトすることはできません。これを行った場合、1 人のユーザーがロールバックする必要があります。そのため、リソースの更新前に、チェックアウト操作をコミットすることをお薦めします。これによって、トランザクションをロールバックする場合に更新内容が失われなくなります。次の場合に例外が発生します。 ■ 指定されたリソースが VCR ではない場合 ■ VCR がすでにチェックアウトされている場合 ■ リソースが存在しない場合
ファンクション **Checkin** Checkin(pathname VARCHAR2) RETURN DBMS_XDB_VERSION.resid_type;	チェックアウトし VCR をチェックインします。 pathname - チェックアウトしたリソースのパス名。 戻り値 - 新しく作成されたバージョンのリソース ID。 これは、自動コミットの SQL 操作ではありません。チェックインでは、チェックアウト操作に渡したパス名と同じパス名を指定する必要はありません。ただし、チェックインのパス名とチェックアウトのパス名は、操作を適切に実行するために、同じリソースのものである必要があります。 リソースの名前が変更された場合、古い名前が無効であるか、または異なるリソースにバインドされているため、新しい名前を使用してチェックインする必要があります。パス名が存在しない場合、例外が発生します。パス名が変更された場合、新しいパス名を使用して、リソースにチェックインする必要があります。
ファンクション **Uncheckout** Uncheckout(pathname VARCHAR2) RETURN DBMS_XDB_VERSION.resid_type;	チェックアウトしリソースをチェックインします。 pathname - チェックアウトしたリソースのパス名。 戻り値 - リソースがチェックアウトされる前のバージョンのリソース ID。これは、自動コミットの SQL 操作ではありません。アンチェックアウトでは、チェックアウト操作に渡したパス名と同じパス名を指定する必要はありません。ただし、アンチェックアウトのパス名とチェックアウトのパス名は、操作を適切に実行するために、同じリソースのものである必要があります。リソースの名前が変更された場合、古い名前が無効であるか、または異なるリソースにバインドされているため、新しい名前を使用してアンチェックアウトする必要があります。パス名が存在しない場合、例外が発生します。パス名が変更された場合、新しいパス名を使用して、リソースにチェックインする必要があります。

表 14-2 DBMS_XDB_VERSION のファンクションおよびプロシージャ （続き）

DBMS_XDB_VERSION のファンクションおよびプロシージャ	説明
ファンクション **GetRoot** GetRoot(vh_id dbms_xdb.resid_type) RETURN DBMS_XDB_VERSION.resid_type;	指定されたバージョン履歴で、すべてのバージョンのルートを取得します。 vh_id- バージョン履歴のリソース ID。 戻り値 - 最初のバージョンのリソース ID。vh_id が無効な場合、例外が発生します。
ファンクション **GetPredecessors** GetPredecessors(pathname VARCHAR2) RETURN resid_list_type; GetPredsByResId(resid DBMS_XDB_VERSION.resid_type) RETURN DBMS_XDB_VERSION.resid_list_type;	指定されたバージョン・リソースまたは VCR で、パス名（リソースのパス名）によって先行リソースを取得します。 戻り値 - 先行リソースのリスト。 リソース ID による先行リソースの取得は、パス名による取得より効率的です。リソース ID またはパス名が無効な場合、例外が発生します。 指定されたバージョン・リソースまたは VCR で、resid（リソース ID）によって先行リソースを取得します。 **注意 :** Oracle の今回のリリースではブランチがサポートされていないため、先行リソースのリストは 1 つの要素（直接の親）のみを含みます。次の GetSuccessors ファンクションも、1 つの要素のみを戻します。
ファンクション **GetSuccessors** GetSuccessors(pathname VARCHAR2) RETURN DBMS_XDB_VERSION.resid_list_type; GetSuccsByResId(resid DBMS_XDB_VERSION.resid_type) RETURN DBMS_XDB_VERSION.resid_list_type;	指定されたバージョン・リソースまたは VCR で、pathname（リソースのパス名）によって後続リソースを取得します。 戻り値 - 後続リソースのリスト。リソース ID による後続リソースの取得は、パス名による取得より効率的です。リソース ID またはパス名が無効な場合、例外が発生します。 指定されたバージョン・リソースまたは VCR で、resid（リソース ID）によって後続リソースを取得します。
ファンクション **GetResourceByResId** GetResourceByResId(resid DBMS_XDB_VERSION.resid_type) RETURN XMLType;	指定されたリソース・オブジェクト ID で、リソースを XMLType として取得します。 resid - リソース・オブジェクト ID。 戻り値 - XMLType としてのリソース。

FAQ: Oracle XML DB Versioning

VCR を非 VCR に切り替えることはできますか？

回答 : できません。

更新後に、VCR の古いコピーにアクセスする方法は？

回答 : 古いコピーは、最後にチェックインしたバージョン・リソースです。

- バージョン ID またはパス名がわかっている場合、その ID を使用して古いコピーをロードできます。
- ID がわからない場合、getPredecessors() をコールして ID を取得できます。

Oracle XML DB のデータ以外のデータにバージョン管理を使用できますか？

回答 : 今回のリリースでは、Oracle XML DB リソースのみバージョン管理できます。

15

RESOURCE_VIEW および PATH_VIEW

この章では、Oracle XML DB Repository のデータへのアクセスに使用する SQL ベースのメカニズム、RESOURCE_VIEW および PATH_VIEW について説明します。リソースのパス名に基づくリソースの問合せに使用する SQL 演算子 UNDER_PATH と EQUALS_PATH、およびリソースのパス名と深さを戻す SQL 演算子 PATH と DEPTH についても説明します。

この章の内容は次のとおりです。

Oracle XML DB の RESOURCE_VIEW および PATH_VIEW

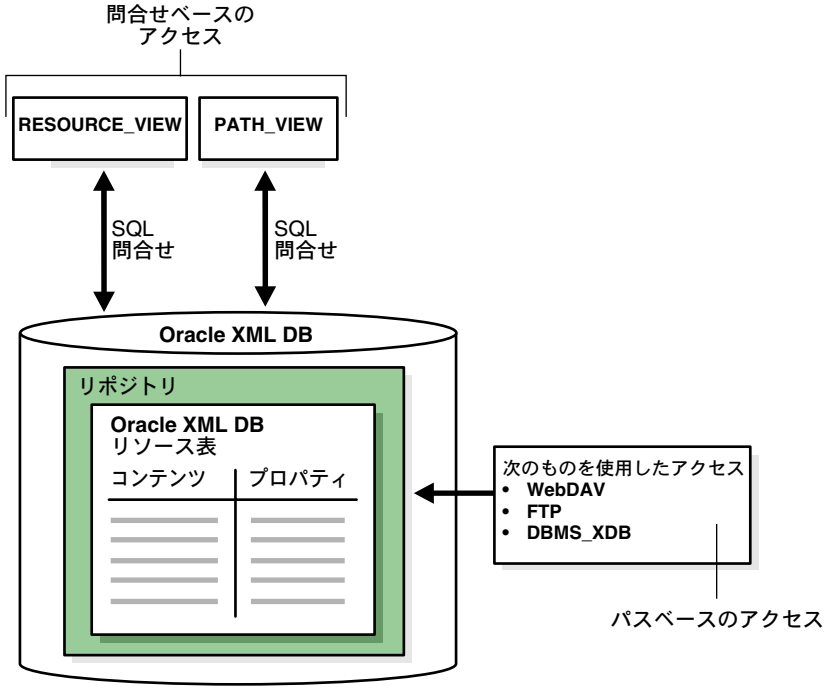
図 15-1 に、Oracle XML DB の RESOURCE_VIEW および PATH_VIEW が、SQL を使用して Oracle XML DB Repository に格納されているデータにアクセスするメカニズムを提供する方法を示します。FTP や WebDAV などのプロトコルまたはプログラミング API を使用して Oracle XML DB Repository に格納されたデータには、RESOURCE_VIEW および PATH_VIEW を使用して SQL でアクセスできます。また、RESOURCE_VIEW および PATH_VIEW を使用して SQL で格納されたデータには、FTP や WebDAV などのプロトコルまたはプログラミング API を使用してアクセスできます。

RESOURCE_VIEW および PATH_VIEW を PL/SQL パッケージ DBMS_XDB とともに使用すると、Oracle XML DB への問合せベースのアクセス、およびプログラミング API を使用して実行可能なすべての DML 機能が提供されます。

RESOURCE_VIEW の実表は XDB.XDB$RESOURCE であり、RESOURCE_VIEW または DBMS_XDB API を介してのみアクセスできます。

> **参照：** 26-61 ページの「6.0 XML DB Demo: SQL を使用した RESOURCE_VIEW の問合せ」を参照してください。

図 15-1 RESOURCE_VIEW および PATH_VIEW を使用したリポジトリ・リソースへのアクセス



RESOURCE_VIEW の定義および構造

RESOURCE_VIEW には、リポジトリのリソースごとに 1 行が含まれます。次に RESOURCE_VIEW の構造を示します。

```
Column      Datatype     Description
------      --------     -----------------------------------------------------
RES         XMLTYPE      A resource in Oracle XML Repository
ANY_PATH    VARCHAR2     A path that can be used to access the resource in the
                         Repository
```

参照： 付録 G「設定スクリプトの例および Oracle XML DB によって提供される XML Schema」を参照してください。

PATH_VIEW の定義および構造

PATH_VIEW には、リポジトリのリソースにアクセスするための一意のパスごとに 1 行が含まれます。次に PATH_VIEW の構造を示します。

```
Column      Datatype     Description
------      --------     ----------------------------
PATH        VARCHAR2     Path name of a resource
RES         XMLTYPE      The resource referred by PATH
LINK        XMLTYPE      Link property
```

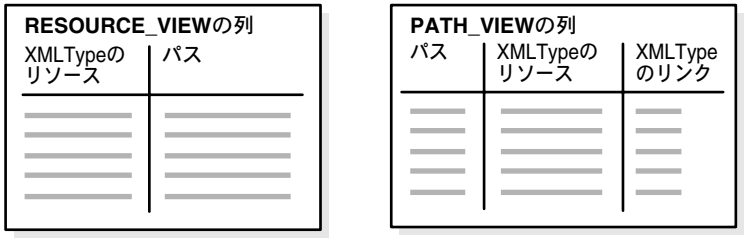
参照： 付録 G「設定スクリプトの例および Oracle XML DB によって提供される XML Schema」を参照してください。

図 15-2 に、RESOURCE_VIEW および PATH_VIEW の構造を示します。

注意： 各リソースは、リンクという複数のパスを持つ場合があります。

RESOURCE_VIEW 内のパスは任意のパスであり、そのリソースへのアクセスに使用できるパスのいずれかです。Oracle XML DB は、演算子 UNDER_PATH を提供します。この演算子によって、アプリケーションで、特定のフォルダ内のリソースを再帰的に検索したり、リソースの深さを取得することができます。PATH_VIEW および RESOURCE_VIEW の列内の各行は、XMLType です。Oracle XML DB Repository ビュー上で DML を使用して、リソースのプロパティおよびコンテンツの挿入、名前の変更、削除および更新を実行できます。既存のリソースへのリンクの作成などの一部の操作には、プログラム API を使用する必要があります。

図 15-2 RESOURCE_VIEW および PATH_VIEW の構造



RESOURCE_VIEW と PATH_VIEW の違いの理解

RESOURCE_VIEW と PATH_VIEW の主な違いは次のとおりです。

- PATH_VIEW は、特定のリソースに対するすべてのパス名を表示しますが、RESOURCE_VIEW は、リソースに対してアクセス可能なパス名の 1 つを表示します。
- PATH_VIEW は、リンクのプロパティも表示します。

図 15-3 に、RESOURCE_VIEW と PATH_VIEW の違いを示します。

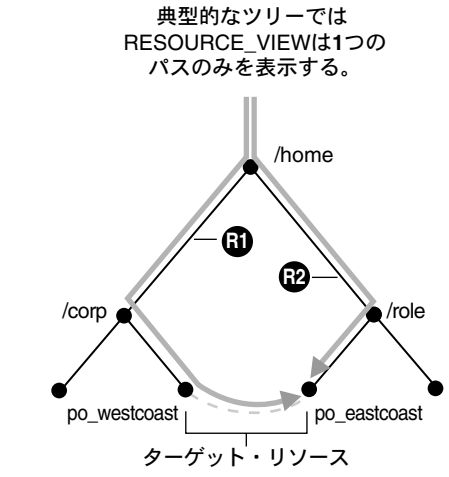
多くのインターネット・アプリケーションでは、リソースにアクセスするために 1 つの URL のみが必要であるため、RESOURCE_VIEW は広範囲に適用できます。

PATH_VIEW には、リンク・プロパティおよびリソース・プロパティが含まれますが、RESOURCE_VIEW には、リソース・プロパティのみが含まれます。

通常、RESOURCE_VIEW のメリットは最適化です。データベースで 1 つのパスのみが必要であるとされている場合、索引で多くの作業を実行してすべてのアクセス可能なパスを判断する必要がありません。

注意： UNDER_PATH 演算子または EQUALS_PATH 演算子を使用してパスを指定する場合、RESOURCE_VIEW を使用すると、そのパスが、RESOURCE_VIEW を使用して、通常そのリソースを表示するために選択する任意のパスであるかどうかにかかわらず、リソースが検索されます。

図 15-3 RESOURCE_VIEW および PATH_VIEW の説明



PATH_VIEWでは、ターゲット・リソース・ノードにアクセスするためにリンクを作成できる。これによって、ターゲット・ノードに対する2つのアクセス・パス R1 または R2 が提供され、高速アクセスが可能になる。

RESOURCE_VIEWの例:
select path(1) from resource_view where under_path(res, '/sys',1);
この場合、次のとおり、リソースに対する1つのパスが表示される。
/home/corp/po_westcoast

PATH_VIEWの例:
select path from path_view;
この場合、次のとおり、リソースに対するすべてのパス名が表示される。
/home/corp/po_westcoast
/home/role/po_eastcoast

UNDER_PATH および EQUALS_PATH を使用して実行できる操作

UNDER_PATH および EQUALS_PATH を使用して、次の操作を実行できます。

- パス名を指定した場合
  - リソースの取得
  - パス名で指定されたディレクトリのリスト
  - リソースの作成
  - リソースの削除
  - リソースの更新
- UNDER_PATH 演算子または他の SQL 演算子を含む条件を指定した場合
  - リソースの更新
  - リソースの削除
  - リソースの取得

15-9 ページの「EQUALS_PATH」および 15-11 ページの「Resource View および Path View の API の使用」を参照してください。

RESOURCE_VIEW API および PATH_VIEW API

この項では、RESOURCE_VIEW および PATH_VIEW で使用する演算子について説明します。

UNDER_PATH

UNDER_PATH 演算子は、Oracle XML DB Repository の階層索引を使用して、特定のパスの下に存在するパスを戻します。階層索引は、パス名を上から下へ高速にアクセスできるように設計されています（通常の使用）。

ただし、問合せ述語の他の部分の選択性が高い場合、リポジトリを下から上に検索する UNDER_PATH の機能実装を選択できます。この場合、検索が必要になるリンクの数が非常に少ないため、より効率的です。図 15-4 に、UNDER_PATH の構文を示します。

図 15-4 UNDER_PATH の構文

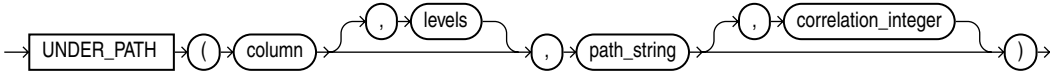


表 15-1 に、UNDER_PATH の構文の説明を示します。

表 15-1 RESOURCE_VIEW API および PATH_VIEW API の構文 :UNDER_PATH

構文	説明
INTEGER UNDER_PATH(resource_column, pathname);	リソースが指定されたパスの下に存在するかどうかを判断します。 パラメータ : ■ resource_column - path_view または resource_view のリソース列の列名または列の別名。 ■ pathname - 解決するパス名。

表 15-1 RESOURCE_VIEW API および PATH_VIEW API の構文 :UNDER_PATH（続き）

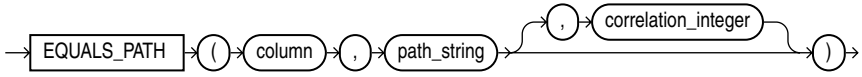
構文	説明
INTEGER UNDER_PATH(resource_column, depth, pathname);	depth 引数で検索するレベルの数を制限して、リソースが指定されたパスの下に存在するかどうかを判断します。 パラメータ : ■ resource_column - path_view または resource_view のリソース列の列名または列の別名。 ■ depth - 検索する深さの最大値。0（ゼロ）未満の深さは 0 として処理されます。 ■ pathname - 解決するパス名。
INTEGER UNDER_PATH(resource_column, pathname, correlation)	補助演算子に対して correlation 引数を指定して、リソースが指定されたパスの下に存在するかどうかを判断します。 パラメータ : ■ resource_column - path_view または resource_view のリソース列の列名または列の別名。 ■ pathname - 解決するパス名。 ■ correlation - UNDER_PATH 演算子（主演算子）と補助演算子（PATH および DEPTH）を相互に関連付けるために使用できる整数。
INTEGER UNDER_PATH(resource_column, depth, pathname, correlation)	depth 引数で検索するレベルの数を制限し、補助演算子に対して correlation 引数を指定して、リソースが指定されたパスの下に存在するかどうかを判断します。 パラメータ : ■ resource_column - path_view または resource_view のリソース列の列名または列の別名。 ■ depth - 検索する深さの最大値。0（ゼロ）未満の深さは 0 として処理されます。 ■ pathname - 解決するパス名。 ■ correlation - UNDER_PATH 演算子（主演算子）と補助演算子（PATH および DEPTH）を相互に関連付けるために使用できる整数。 戻されるリソースの pathname 引数の下には、そのリソースにアクセス可能なパスが 1 つしか必要ないことに注意してください。

EQUALS_PATH

EQUALS_PATH 演算子は、特定のパス名を指定したリソースの検索に使用されます。これは、UNDER_PATH の深さの制限を 0（ゼロ）に指定した場合と機能的に同じです。EQUALS_PATH の構文は次のとおりです。

```
EQUALS_PATH INTEGER EQUALS_PATH( resource_column,pathname);
```

図 15-5 EQUALS_PATH の構文



パラメータは次のとおりです。

- resource_column は、path_view または resource_view のリソース列の列名または列の別名です。
- pathname は、解決するパス名です。

PATH

PATH は、指定された pathname 引数の下に存在するリソースの相対パス名を戻す補助演算子です。RESOURCE_VIEW のパス列には、常にリソースの絶対パスが含まれることに注意してください。PATH の構文は次のとおりです。

```
PATH VARCHAR2 PATH( correlation);
```

パラメータは次のとおりです。

- correlation は、UNDER_PATH 演算子（主演算子）と補助演算子（PATH および DEPTH）を相互に関連付けるために使用できる整数です。

> **注意：** 指定された pathname 引数の下にパスがない場合、NULL 値が現行のパスの出力として戻されます。

次のパスで指定されたリソースを含む RESOURCE_VIEW の例を次に示します。

```
'/a/b/c'
'/a/b/c/d'
'/a/e/c'
'/a/e/c/d'
```

例 15-1 指定された pathname 引数の下のパスの判別

```
SELECT path(1) FROM resource_view
   WHERE UNDER_PATH(res, '/a/b', 1) = 1;
```

次の結果が戻されます。

```
PATH(1)
-------
c
c/d
2 rows returned
```

例 15-2 指定された pathname 引数の下にないパスの判別

```
SELECT path(1) FROM resource_view
   WHERE UNDER_PATH(res, '/a/b', 1)!=1
```

次の結果が戻されます。

```
PATH(1)
-------


2 rows returned
```

注意： 絶対パスには、次のとおり ANY_PATH を使用してください。

```
SELECT ANY_PATH
FROM resource_view
WHERE UNDER_PATH(res, '/a/b')=1;
```

次の結果が戻されます。

```
ANY_PATH
--------
/a/e/c
/a/e/c/d
2 rows returned
```

例 15-3 複数の correlation を使用したパスの判別

```
SELECT ANY_PATH, path(1), path(2)
FROM resource_view
WHERE UNDER_PATH(res, '/a/b', 1) = 1 or UNDER_PATH(res, '/a/e', 2) = 1;
```

次の結果が戻されます。

```
ANY_PATH           PATH(1)                 PATH(2)
--------------------------------------------------
/a/b/c             c
/a/b/c/d           c/d
/a/e/c                                     c
/a/e/c/d                                   c/d
4 rows returned
```

DEPTH

DEPTH は、指定された開始パスの下に存在するリソースのフォルダの深さを戻す補助演算子です。DEPTH の構文は次のとおりです。

```
DEPTH  INTEGER DEPTH( correlation);
```

パラメータは次のとおりです。

correlation は、UNDER_PATH 演算子（主演算子）と補助演算子（PATH および DEPTH）を相互に関連付けるために使用できる整数です。

Resource View および Path View の API の使用

次の RESOURCE_VIEW および PATH_VIEW の例では、UNDER_PATH 演算子、EQUALS_PATH 演算子、PATH 演算子および DEPTH 演算子を使用します。

パスおよびリポジトリ・リソースへのアクセス : 例

次の例は、リポジトリ・パス、リソースおよびリンク・プロパティにアクセスする方法を示します。

例 15-4 UNDER_PATH の使用 : パス名を指定して、RESOURCE_VIEW からパス名ごとディレクトリをリストする

```
select any_path from resource_view where any_path like '/sys%';
```

例 15-5 UNDER_PATH の使用 : パス名を指定して、RESOURCE_VIEW からリソースを取得する

```
select any_path, extract(res, '/Resource') from resource_view
   where under_path(res, '/sys') = 1;
```

例 15-6 RESOURCE_VIEW の使用 : パスを指定して、リソースに対する最大 3 レベルのすべての相対パス名を取得する

```
select path(1) from resource_view
   where under_path (res, 3, '/sys',1)=1;
```

例 15-7 UNDER_PATH の使用 : パス名を指定して、PATH_VIEW から指定したパスの下に存在するパスおよび深さを取得する

```
select path(1) PATH,depth(1) depth
  from path_view
  where under_path(RES, 3,'/sys',1)=1;
```

例 15-8 パス名を指定して、PATH_VIEW からパスおよびリンク・プロパティを取得する

```
select path, extract(link, '/LINK/Name/text()').getstringval(),
               extract(link, '/LINK/ParentName/text()').getstringval(),
               extract(link, '/LINK/ChildName/text()').getstringval(),
               extract(res, '/Resource/DisplayName/text()').getstringval()
   from path_view
   where path LIKE '/sys%';
```

例 15-9 UNDER_PATH の使用 : パス名を指定して、PATH_VIEW から、指定したパスの下に存在するリンクを含む、特定のレベル以下のすべてのパスを検索する

```
select path(1) from path_view
   where under_path(res, 3,'/sys', 1) > 0 ;
```

例 15-10 EQUALS_PATH を使用して、パスの位置を設定する

```
select any_path from resource_view
   where equals_path(res, '/sys') > 0;
```

リポジトリ・リソースへのデータの挿入例

次の例は、リソースへデータを挿入する方法を示します。

例 15-11 リソースの作成 : リソースへのデータの挿入

```
insert into resource_view values(sys.xmltype.createxml('
  <Resource xmlns="http://xmlns.oracle.com/xdb/XDBResource.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://xmlns.oracle.com/xdb/XDBResource.xsd
http://xmlns.oracle.com/xdb/XDBResource.xsd">
  <Author>John Doe</Author>
  <DisplayName>example</DisplayName>
  <Comment>This resource was contrived for resource view demo</Comment>
  <Language>en</Language>
```

```
<CharacterSet>ASCII</CharacterSet>
<ContentType>text/plain</ContentType>
</Resource>'), '/home/SCOTT');
```

リポジトリ・リソースの削除例

次の例は、リソースまたはパスを削除する方法を示します。

例 15-12 リソースの削除

```
delete from resource_view where any_path = '/home/SCOTT/example';
```

リーフ・リソースのみを削除するには、delete from resource_view where... を使用します。

空でないコンテナの再帰的な削除

リーフ・リソースのみを削除するには、delete from resource_view where... を使用します。次に、リーフ・ノード /public/test/doc.xml を削除する方法の例を示します。

```
delete from resource_view where under_path(res, '/public/test/doc.xml') = 1;
```

ただし、空でないコンテナを再帰的に削除する場合、次の規則が適用されます。

- 空でないコンテナ上では、削除を実行できません。
- where 句の述語から戻されたパスの順序は保証されません。

そのため、コンテナの子が削除された後にのみ、そのコンテナが削除されることを保証する必要があります。

例 15-13 パスの再帰的な削除

たとえば、/public の下のパスを再帰的に削除するには、次のコマンドを実行します。

```
delete from
(select 1 from resource_view
 where UNDER_PATH(res, '/public', 1) = 1
 order by depth(1) desc);
```

リポジトリ・リソースの更新例

次の例は、リソースおよびパスを更新する方法を示します。

例 15-14 リソースの更新

```
update resource_view set res = sys.xmltype.createxml('
  <Resource xmlns="http://xmlns.oracle.com/xdb/XDBResource.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://xmlns.oracle.com/xdb/XDBResource.xsd
http://xmlns.oracle.com/xdb/XDBResource.xsd">
  <Author>John Doe</Author>
  <DisplayName>example</DisplayName>
  <Comment>Has this got updated or not ? </Comment>
  <Language>en</Language>
  <CharacterSet>ASCII</CharacterSet>
  <ContentType>text/plain</ContentType>
  </Resource>')
    where any_path = '/home/SCOTT/example';
```

例 15-15 PATH_VIEW 内のパスの更新

```
update path_view set path = '/home/XDB'
  where path = '/home/SCOTT/example';
```

注意： あるディレクトリの下に存在するすべてのリソースを取得する必要がある場合は、LIKE 演算子を使用できます（15-11 ページの例 15-4 を参照）。

特定のレベル以下のリソースを取得する必要がある場合、または相対パスを取得する必要がある場合は、UNDER_PATH 演算子を使用します（15-11 ページの例 15-5 を参照）。

例 15-4 は、例 15-5 より適切な問合せ計画です。

参照： RESOURCE_VIEW 演算子および PATH_VIEW 演算子を使用したその他の例は、13-14 ページ の表 13-3「Oracle XML DB Repository へのアクセス :API オプション」を参照してください。

複数の Oracle XML DB リソースの同時操作

13-14 ページの表 13-3 に示す操作は、通常、一度に 1 つのリソースのみに適用されます。複数の Oracle XML DB リソースに同じ操作を実行するか、または一連の特定基準を満たす 1 つ以上の Oracle XML DB リソースを検索するには、SQL で RESOURCE_VIEW および PATH_VIEW を使用します。

たとえば、SQL 句 RESOURCE_VIEW および PATH_VIEW を使用して、次の操作を実行できます。

- 属性に基づいた更新

```
UPDATE RESOURCE_VIEW SET … WHERE extractValue(resource, '/Resource/DisplayName')
= 'My stuff';
```

- フォルダ内の再帰的な検索

```
SELECT FROM RESOURCE_VIEW WHERE UNDER_PATH(resource, '/sys') …
```

- 一連の Oracle XML DB リソースのコピー

```
INSERT INTO PATH_VIEW SELECT …. FROM PATH_VIEW WHERE ...
```

高速問合せのための XML DB のチューニング

XML DB は、システムおよびプロトコル環境の構成に xdbconfig.xml ファイルを使用します。リリース 2（9.2.0.2）には、RESOURCE_VIEW キャッシュのメモリー内サイズを定義する resource-view-cache-size パラメータ要素が含まれています。デフォルト値は、1048576 です。

RESOURCE_VIEW および PATH_VIEW の問合せは、resource-view-cache-size をチューニングすることによって高速化できる場合があります。一般に、キャッシュ・サイズが大きいほど問合せは高速化します。多くの場合、デフォルトの resource-view-cache-size は適切です。ただし、大規模な RESOURCE_VIEW の問合せ時には、resource-view-cache-size 要素を大きくすると有効な場合があります。

> **参照：** 付録 A「Oracle XML DB のインストールおよび構成」を参照してください。

Oracle Text を使用したリソースの検索

Oracle XML DB のユーザー・スキーマの XDB$RESOURCE 表は、ファイルやフォルダなどのリソースに対応するメタデータおよびデータを Oracle XML DB に格納します。RESOURCE_VIEW または PATH_VIEW で CONTAINS 演算子を使用して、特定のキーワードを含むリソースを検索できます。

例 15-16 キーワード「Oracle」および「Unix」を含むすべてのリソースの検索

```
select path
   from path_view
   where contains(res, 'Oracle AND Unix') > 0;
```

例 15-17 キーワード「Oracle」を含み、指定したパスの下に存在するすべてのリソースの検索

```
select any_path
   from resource_view
   where contains(res, 'Oracle') > 0
   and under_path(res, '/myDocuments') > 0;
```

このような問合せを評価するには、XDB$RESOURCE 表に ConText 索引を作成する必要があります。Oracle XML DB に格納されたドキュメント・タイプに応じて次のいずれかのオプションを選択して、ConText 索引を作成します。

- **Oracle XML DB が XML 文書のみを含む場合**、つまり、バイナリ・データを含まない場合、通常の ConText 索引を XDB$RESOURCE 表に作成することができます。

  参照： 7-35 ページの「XMLType の索引付け」を参照してください。

  ```
  create index xdb$resource_ctx_i
     on xdb.xdb$resource x (value(x))
     indextype is ctxsys.context;
  ```

- **Oracle XML DB がバイナリ・データ（Word ドキュメントなど）を含む場合**、索引付けの前にドキュメントをフィルタリングするユーザー・フィルタが必要です。ConText 索引の作成と構成には、DBMS_XDBT パッケージ（dbmsxdbt.sql）を使用することをお薦めします。

  参照：

  - DBMS_XDBT のインストールおよび使用の詳細は、『Oracle9*i* XML API リファレンス - XDK および Oracle XML DB』の DBMS_XDBT に関する章を参照してください。
  - F-28 ページの「DBMS_XDBT」も参照してください。

```
Rem Install the package - connected as SYS
SQL>@dbmsxdbt
Rem Create the preferences
SQL>exec dbms_xdbt.createPreferences;
Rem Create the index
SQL>exec dbms_xdbt.createIndex;
```

DBMS_XDBT パッケージには、索引を同期化および最適化するプロシージャも含まれています。configureAutoSync() プロシージャを使用して、ジョブ・キューを使用した索引の自動同期化を構成することができます。

16

Oracle XML DB Resource API for PL/SQL（DBMS_XDB）

この章では、PL/SQL を使用した Oracle XML DB Repository のリソースおよびデータへのアクセスおよび管理に使用する Oracle XML DB Resource API for PL/SQL（DBMS_XDB）について説明します。また、リソースのセキュリティおよび Oracle XML DB の構成の管理方法も説明します。

この章の内容は次のとおりです。

Oracle XML DB Resource API for PL/SQL の概要

この章では、Oracle XML DB Resource API for PL/SQL（PL/SQL パッケージ DBMS_XDB）について説明します。これは、PL/SQL フォルダリング API ともいいます。

Oracle XML DB Repository は XML でモデル化されており、すべてのデータに対するデータベース・ファイル・システムを提供します。Oracle XML DB Repository は、XMLType のデータベース・オブジェクトにパス名（または URL）をマップし、これらのオブジェクトの管理機能を提供します。

DBMS_XDB パッケージは、PL/SQL を使用して、Oracle XML DB Repository にアクセスおよび管理するためのファンクションおよびプロシージャを提供します。

> **参照：** 『Oracle9*i* XML API リファレンス - XDK および Oracle XML DB』を参照してください。

DBMS_XDB の概要

DBMS_XDB は、PL/SQL アプリケーションの開発者に次のものを管理するための API を提供します。

- Oracle XML DB のリソース
- Oracle XML DB の ACL ベースのセキュリティ
- Oracle XML DB の構成
- Oracle XML DB の階層索引の再作成

DBMS_XDB: Oracle XML DB のリソース管理

表 16-1 に、DBMS_XDB の Oracle XML DB リソース管理メソッドを示します。

表 16-1 DBMS_XDB のリソース管理メソッド

DBMS_XDB のメソッド	引数、戻り値
Link	引数 : (srcpath VARCHAR2, linkfolder VARCHAR2, linkname VARCHAR2) 戻り値 : なし
LockResource	引数 : (path IN VARCHAR2, depthzero IN BOOLEAN, shared IN boolean) 戻り値 : TRUE（正常終了した場合）
GetLockToken	引数 : (path IN VARCHAR2, locktoken OUT VARCHAR2) 戻り値 : なし

表 16-1 DBMS_XDB のリソース管理メソッド（続き）

DBMS_XDB のメソッド	引数、戻り値
UnlockResource	引数 : (path IN VARCHAR2, deltoken IN VARCHAR2) 戻り値 : TRUE（正常終了した場合）
CreateResource	FUNCTION CreateResource (path IN VARCHAR2, data IN VARCHAR2) RETURN BOOLEAN; 指定された文字列をそのコンテンツとして持つ新しいリソースを作成します。 FUNCTION CreateResource (path IN VARCHAR2, data IN SYS.XMLTYPE) RETURN BOOLEAN; 指定された XMLType データをそのコンテンツとして持つ新しいリソースを作成します。 FUNCTION CreateResource (path IN VARCHAR2, datarow IN REF SYS.XMLTYPE) RETURN BOOLEAN; 既存の XMLType 行に対する REF を指定して、その行を指すコンテンツを持つリソースを作成します。その行は、他のリソースの内部には存在していない必要があります。 FUNCTION CreateResource (path IN VARCHAR2, data IN CLOB) RETURN BOOLEAN; 指定された CLOB をそのコンテンツとして持つリソースを作成します。 FUNCTION CreateResource (path IN VARCHAR2, data IN BFILE) RETURN BOOLEAN; 指定された BFILE をそのコンテンツとして持つリソースを作成します。
CreateFolder	引数 : (path IN VARCHAR2) 戻り値 : TRUE（正常終了した場合）
DeleteResource	引数 : (path IN VARCHAR2) 戻り値 : なし

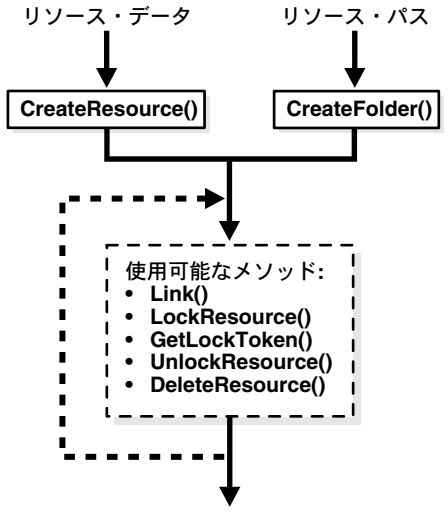
DBMS_XDB を使用したリソースの管理 : コール順序

図 16-1 に、DBMS_XDB を使用してリポジトリのリソースを管理するときのコール順序を示します。

1. コール順序の図では、リポジトリのリソースを管理するときに、リソースおよびフォルダがすでに存在していると想定しています。リソースおよびフォルダが存在していない場合、CreateResource() を使用してリソースを作成するか、または CreateFolder() を使用してフォルダを作成する必要があります。
   - CreateResource() は、パラメータとしてリソース・データおよびリソース・パスを取ります。
   - CreateFolder() は、パラメータとしてリソース・パスを取ります。
2. リソースまたはフォルダに追加の処理または管理が必要ない場合、リソースまたはフォルダがそのまま出力されます。
3. リソースまたはフォルダに追加の処理または管理が必要な場合、表 16-1 に示すとおり、次のメソッドのいずれかまたはすべてを適用できます。
   - Link()
   - LockResource()
   - GetLockToken()
   - UnlockResource()
   - DeleteResource()

DBMS_XDB を使用してリポジトリのリソースを管理する方法は、例 16-1 を参照してください。

図 16-1 DBMS_XDB を使用したリソースの管理 : コール順序



例 16-1 DBMS_XDB を使用したリソースの管理

```
DECLARE
   retb boolean;
BEGIN
   retb := dbms_xdb.createfolder('/public/mydocs');
   commit;
END;
/

declare
  bret boolean;
begin
  bret :=
dbms_xdb.createresource('/public/mydocs/emp_scott.xml','<emp_name>scott</emp_name>')
;
  commit;
end;
/

declare
  bret boolean;
begin
  bret :=
dbms_xdb.createresource('/public/mydocs/emp_david.xml','<emp_name>david</emp_name>')
```

```
;
  commit;
end;
/

call dbms_xdb.link('/public/mydocs/emp_scott.xml','/public/mydocs',
'person_scott.xml');
call dbms_xdb.link('/public/mydocs/emp_david.xml','/public/mydocs',
'person_david.xml');
commit;

call dbms_xdb.deleteresource('/public/mydocs/emp_scott.xml');
call dbms_xdb.deleteresource('/public/mydocs/person_scott.xml');
call dbms_xdb.deleteresource('/public/mydocs/emp_david.xml');
call dbms_xdb.deleteresource('/public/mydocs/person_david.xml');
call dbms_xdb.deleteresource('/public/mydocs');
commit;
```

DBMS_XDB: Oracle XML DB の ACL ベースのセキュリティ管理

表 16-2 に、DBMS_XDB の Oracle XML DB の ACL ベースのセキュリティ管理メソッドを示します。メソッドの引数および戻り値の詳細については省略します。

表 16-2 DBMS_XDB: セキュリティ管理メソッド

DBMS_XDB のメソッド	引数、戻り値
GetAclDocument	引数 : (abspath VARCHAR2) 戻り値 : XMLType（ACL 文書の場合）
ACLCheckPrivileges	引数 : (acl_path IN VARCHAR2, owner IN VARCHAR2, privs IN XMLType) 戻り値 : 正の整数（権限が付与されている場合）
CheckPrivileges	引数 : (res_path IN VARCHAR2, privs IN XMLType) 戻り値 : 正の整数（権限が付与されている場合）
GetPrivileges	引数 : (res_path IN VARCHAR2) 戻り値 : <privilege> 要素の XMLType インスタンス
ChangePrivileges	引数 : (res_path IN VARCHAR2, ace IN XMLType) 戻り値 : 正の整数（ACL が正常に変更された場合）
SetAcl	引数 : (res_path IN VARCHAR2, acl_path IN VARCHAR2) res_path に存在するリソースの ACL を acl_path に存在する ACL に設定します。 戻り値 : なし

DBMS_XDB を使用したセキュリティの管理 : コール順序

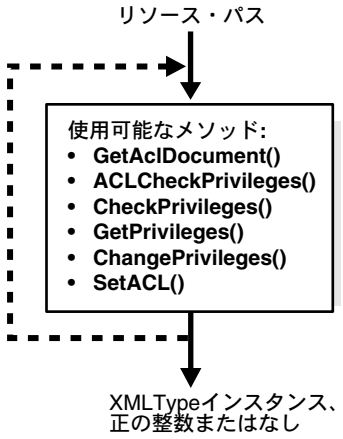
図 16-2 に、DBMS_XDB を使用してセキュリティを管理するときのコール順序を示します。

1. 各 DBMS_XDB のセキュリティ管理メソッドが、パス（resource_path、abspath または acl_path）を取ります。
2. 表 16-2 に示す DBMS_XDB のメソッドのいずれかまたはすべてを使用して、セキュリティ管理タスクを実行できます。
   - `GetAclDocument()`
   - `ACLCheckPrivileges()`
   - `CheckPrivileges()`
   - `GetPrivileges()`

- ChangePrivileges()
- SetACL()

DBMS_XDB を使用してリポジトリのリソースのセキュリティを管理する方法は、例 16-2 を参照してください。

図 16-2 DBMS_XDB を使用したセキュリティの管理 : コール順序



例 16-2 DBMS_XDB を使用した ACL ベースのセキュリティの管理

```
DECLARE
   retb boolean;
BEGIN
   retb := dbms_xdb.createfolder('/public/mydocs');
   commit;
END;
/

declare
  bret boolean;
begin
  bret :=
dbms_xdb.createresource('/public/mydocs/emp_scott.xml','<emp_name>scott</emp_name>')
;
  commit;
end;
/
```

```
call dbms_xdb.setacl('/public/mydocs/emp_scott.xml',
'/sys/acls/all_owner_acl.xml');
commit;

select dbms_xdb.getacldocument('/public/mydocs/emp_scott.xml') from
dual;

declare
  r          pls_integer;
  ace        xmltype;
  ace_data   varchar2(2000);
begin
   ace_data :=
'<ace
      xmlns="http://xmlns.oracle.com/xdb/acl.xsd"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://xmlns.oracle.com/xdb/acl.xsd
                          http://xmlns.oracle.com/xdb/acl.xsd
                          DAV:http://xmlns.oracle.com/xdb/dav.xsd">
    <principal>SCOTT</principal>
    <grant>true</grant>
    <privilege>
      <all/>
    </privilege>
 </ace>';
  ace := xmltype.createxml(ace_data);
  r := dbms_xdb.changeprivileges('/public/mydocs/emp_scott.xml', ace);
  dbms_output.put_line('retval = ' || r);
  commit;
end;
/

select dbms_xdb.getacldocument('/public/mydocs/emp_scott.xml') from
dual;

select dbms_xdb.getprivileges('/public/mydocs/emp_scott.xml') from dual;

call dbms_xdb.deleteresource('/public/mydocs/emp_scott.xml');
call dbms_xdb.deleteresource('/public/mydocs');
commit;
```

DBMS_XDB: Oracle XML DB の構成管理

表 16-3 に、DBMS_XDB の Oracle XML DB の構成管理メソッドを示します。メソッドの引数および戻り値の詳細については省略します。

表 16-3 DBMS_XDB: 構成管理メソッド

DBMS_XDB のメソッド	引数、戻り値
CFG_Get	引数 : なし 戻り値 : XMLType（セッション構成情報）
CFG_Refresh	引数 : なし 戻り値 : なし
CFG_Update	引数 : (xdbconfig IN XMLType) 戻り値 : なし

DBMS_XDB を使用した構成の管理 : コール順序

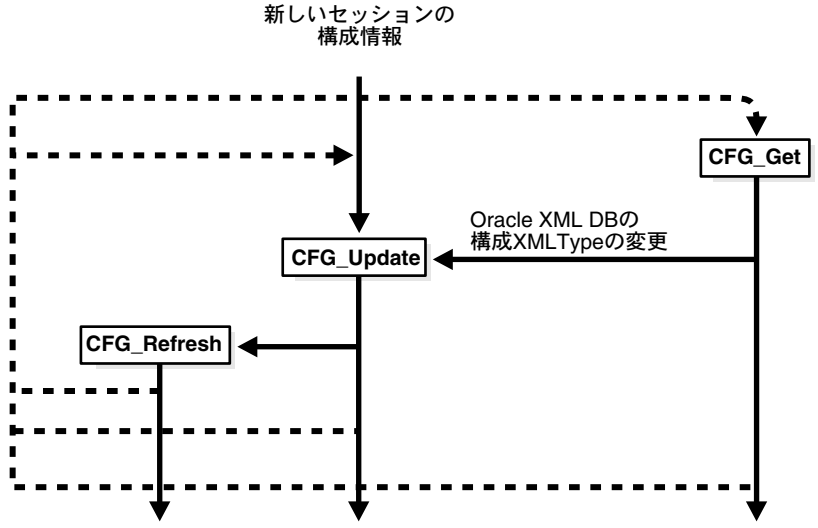
図 16-3 に、DBMS_XDB を使用して構成を管理するときのコール順序を示します。

この図は、次の順序を示しています。

1. Oracle XML DB の構成を管理するには、まず、CFG_Get を使用して、構成のインスタンスを取り出す必要があります。
2. 次に、オプションで、Oracle XML DB の構成 XMLType インスタンスを更新するために変更したり、Oracle XML DB の構成をそのまま出力することもできます。
3. Oracle XML DB の構成リソースを更新するには、CFG_Update を使用します。新しい Oracle XML DB の構成 XMLType インスタンスを入力するか、または現在の構成の変更済バージョンを使用する必要があります。
4. Oracle XML DB の構成リソースをリフレッシュするには、CFG_Refresh を使用します。構成 XMLType インスタンスを入力する必要はありません。

DBMS_XDB を使用してリポジトリのリソースの構成を管理する方法は、例 16-3 を参照してください。

図 16-3 DBMS_XDB を使用した構成の管理 : コール順序



例 16-3 DBMS_XDB を使用した Oracle XML DB の構成の管理

```
connect system/manager

select dbms_xdb.cfg_get() from dual;

declare
  config   xmltype;
begin
  config := dbms_xdb.cfg_get();
  -- Modify the xdb configuration using updatexml, etc ...
  dbms_xdb.cfg_update(config);
end;
/

-- To pick up the latest XDB Configuration
--   In this example it is not needed as cfg_update(),
--   automatically does a cfg_refresh().
call dbms_xdb.cfg_refresh();
```

DBMS_XDB: Oracle XML DB の階層索引の再構築

表 16-4 に、DBMS_XDB の Oracle XML DB の階層索引の再構築メソッドを示します。メソッドの引数および戻り値の詳細については省略します。

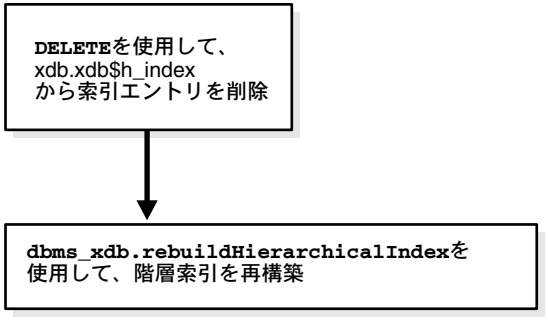
表 16-4 DBMS_XDB: 階層索引の再構築メソッド

DBMS_XDB のメソッド	引数、戻り値
RebuildHierarchicalIndex	引数 : なし 戻り値 : なし

DBMS_XDB を使用した階層索引の再構築 : コール順序

図 16-4 に、DBMS_XDB を使用して階層索引を再構築するときのコール順序を示します。階層索引を再構築するには、まず xdb.xdb$h_index からエントリを削除し、DBMS_XDB.RebuildHierarchicalIndex を実行して階層索引を再構築します。例 16-4 に、DBMS_XDB を使用してリポジトリの階層索引を再構築する方法を示します。

図 16-4 DBMS_XDB を使用した階層索引の再構築 : コール順序



例 16-4 DBMS_XDB を使用した階層索引の再構築

```
connect system/manager
delete from xdb.xdb$h_index;
commit;
execute dbms_xdb.RebuildHierarchicalIndex;
```

17

Oracle XML DB Resource API for Java

この章では、Oracle XML DB Resource API for Java について説明します。この章の内容は次のとおりです。

- Oracle XML DB Resource API for Java の概要
- Oracle XML DB Resource API for Java の使用
- Oracle XML DB Resource API for Java のパラメータ
- Oracle XML DB Resource API for Java: 例

Oracle XML DB Resource API for Java の機能は、現在サポートされていません。

Oracle XML DB Resource API for Java の概要

参照：

- 『Oracle9*i* XML API リファレンス - XDK および Oracle XML DB』を参照してください。
- 第 9 章「Java API for XMLType」も参照してください。

Oracle XML DB Resource API for Java の使用

Oracle XML DB Resource API for Java の動作は次のとおりです。

- JDBC は、Oracle XML DB リソース・ビューにアクセスして、リソースの取出しおよび変更を行うことができます。

また、次の操作を実行できます。

- Java API for `XMLType` を使用した、`XMLType` オブジェクトの一部へのアクセスおよび変更

Oracle XML DB Resource API for Java には、リソース・タイプのバージョニング情報の WebDAV を示す一連のサブインタフェースが含まれています。

API には、WebDAV のバージョニング仕様で定義された作業領域、ブランチ、ベースラインなどのオブジェクト用のインタフェースが含まれています。

Oracle XML DB Resource API for Java のパラメータ

表 17-1 に、Oracle XML DB Resource API for Java でサポートされているパラメータを示します。

表 17-1 Oracle XML DB Resource API for Java: パラメータ

パラメータ名	説明
PROVIDER_URL	オブジェクトを戻す元の開始パスを指定します。
INITIAL_CONTEXT_FACTORY	コンテキストの生成に使用するコンテキスト・ファクトリを指定します（常に、oracle.xdb.spi.XDBContextFactory）。

表 17-1 Oracle XML DB Resource API for Java: パラメータ（続き）

パラメータ名	説明
XDB_RESOURCE_TYPE	パス名が解決されたときに、デフォルトでアプリケーションに戻すデータを決定します。 ■ Resource：リソース・クラスを戻します。 ■ XMLType：リソースのコンテンツを戻します。 今回のリリースでは、Oracle XML DB は javax.naming パッケージのみを実装しています。Oracle XML DB には、このパッケージへの拡張機能（lookup() メソッドへの拡張機能）があります。また、拡張機能（オーバーロード）は、行ロックの取得を指示するブールを取って「lookup FOR UPDATE」を示し、XPath とともに文字列を取って（実体化の遅延機能を使用せず）すぐにロードする文書のフラグメントを定義します。

Oracle XML DB Resource API for Java: 例

例 17-1 リソース JDBC: SQL を使用した発注書のプロパティの判断

次に、XMLType オブジェクトの検索にパス名以外の条件を指定して SQL の SELECT を使用する例を示します。

```
PreparedStatement pst = con.prepareStatement(
"SELECT r.RESOLVE_PATH('/companies/oracle') FROM XDB$RESOURCE r");

pst.executeQuery();
XMLType po = (XMLType)pst.getObject(1);
Document podoc = (Document) po.getDOM();
```

18

Oracle XML DB リソースのセキュリティ

この章では、Oracle XML DB リソースに対するアクセス制御リスト（ACL）・ベースのセキュリティ・メカニズムについて説明します。ACL を作成する方法、リソースに対する ACL を設定および変更する方法、および ACL セキュリティが他のデータベース・セキュリティ・メカニズムと相互作用する方法を説明します。

この章の内容は次のとおりです。

Oracle XML DB リソースのセキュリティおよび ACL の概要

Oracle XML DB は、Oracle XML DB Repository の階層のすべてのリソースに対して、オブジェクトレベルのセキュリティを保持します。

注意： Oracle XML DB Repository に格納されていない XML オブジェクトでは、オブジェクトレベルのアクセス制御は行われません。

Oracle XML DB は、アクセス制御リスト（ACL）のメカニズムを使用して、すべての Oracle XML DB リソースまたは Oracle XML DB Repository にマップされたすべてのデータベース・オブジェクトへのアクセスを制限します。

Oracle XML DB の ACL セキュリティ・メカニズムは、WebDAV の ACL 仕様をサポートします。ACL は、Java、Windows NT およびその他のシステムで使用される標準のセキュリティ・メカニズムです。

Oracle XML DB の ACL セキュリティ・メカニズムは、Oracle9*i* データベースに格納された大量の XML データを処理するように設計されています。ドキュメントの所有者を表すプリンシパル dav:owner には、その所有者が誰であるかにかかわらず、権限を付与または拒否できます。

参照： 次の章またはマニュアルを参照してください。

- 第 21 章「Oracle Enterprise Manager を使用した Oracle XML DB の管理」
- 『Oracle9*i* XML API リファレンス - XDK および Oracle XML DB』

ACL ベースのセキュリティ・メカニズムの動作

ユーザーがリソースに対して操作またはメソッドを実行する前に、そのリソースに対してユーザーが所有している権限が確認されます。確認される権限は、実行される操作またはメソッドによって異なります。たとえば、従業員 Scott の給与を 10% 上げるには、`scott/salary.xml` リソースに対して、`READ` 権限および `WRITE` 権限が必要です。

アクセス制御リストの用語

この項では、いくつかのアクセス制御リスト（ACL）の用語について説明します。

- **プリンシパル :** Oracle XML DB リソースへのアクセス制御権限を付与されたエンティティです。Oracle XML DB は、次のプリンシパルをサポートします。
  - データベース・ユーザー。
  - データベース・ロール。データベース・ロールは、グループとして認識されます。たとえば、DBA ロールは、DBA ロールを付与されたすべてのユーザーの DBA グループを表します。

  注意 : LDAP サーバーからインポートされたユーザーおよびロールも、データベースの一般的な認証モデルの一部としてサポートされます。

  保護されているオブジェクトの個別のプロパティに対応する、dav:owner という特別なプリンシパルが存在します。通常、ドキュメントの所有者は特別な権限を所有しているため、dav:owner プリンシパルを使用すると、より多くのユーザー間で ACL を共有できます。18-6 ページの「アクセス制御 : ユーザーおよびグループ・アクセス」を参照してください。

- **権限 :** プリンシパルに付与できる特定の権限です。Oracle XML DB には、すべての ACL で参照可能なシステム定義の権限のセット（READ、INSERT、UPDATE など）が存在します。権限は、次のいずれかになります。
  - 集約権限（他の権限を含みます）
  - 基本権限（分割できません）

  集約権限は、権限の数が多くなった場合に権限を簡単に使用できるようにし、ACL クライアント間の相互運用性を高めるために有効なネーミングです。権限のセットによって、指定された操作またはメソッドを Oracle XML DB リソースに実行する権限が制御されます。たとえば、プリンシパル `Scott` が指定されたリソースの `read` 操作を実行する場合、読込み操作を実行する前に、`Scott` に `read` 権限が付与されている必要があります。したがって、権限によって指定されたリソースへのユーザーの操作が制御されます。

- **アクセス制御エントリ（ACE）:** ACL の一部で、特定のプリンシパルへのアクセスを許可または拒否します。ACL は、ACE のリストで構成されます（順不同）。単一の ACL の特定のプリンシパルには、1 つの `grant ACE` および 1 つの `deny ACE` のみが許可されます。

  注意 : 多数のロールが付与された特定のユーザーには、多数の `grant ACE`（または `deny ACE`）を適用できます。

Oracle XML DB の ACE 要素は、次の属性を持ちます。

- 操作 : `grant` または `deny` のいずれか
- プリンシパル : ユーザーまたはグループ（コレクション）のいずれか
- 権限のセット : 特定のプリンシパルに対して付与または拒否される、特定の権限のセット

- **アクセス制御リスト（ACL）:** リソースへのアクセス制御を定義する、要素名 `ace` を持つアクセス制御エントリの要素のリストです。ACE は、プリンシパルの権限を付与または拒否します。

  注意： ACL は、Oracle XML DB リソースとして格納されるため、Oracle XML DB の ACL によって保護する必要があります。**ブートストラップ ACL** という自己保護型の ACL が 1 つ存在します。これは、それ自体のコンテンツによって保護されます。

- **名前付き ACL:** 独自のパス名を持つ、それ自体がリソースである ACL です。名前付き ACL は、複数のリソースによって共有できます。これによって、管理性が向上し、簡単に使用できるようになり、パフォーマンスが向上します。名前付き ACL は、一意の名前を持ち、また次のようなオプションの型リストリクタも持ちます。

  `http://xmlns.oracle.com/xdb/XDBDemo.xsd#PurchaseOrder`

  これは、ACL が、その XML 要素のインスタンスおよびその要素を含む代替グループの要素にのみ適用できることを指定します。

- **デフォルトの ACL:** Oracle XML DB Repository にリソースを挿入する場合、次の 2 つの方法でこのリソースに ACL を指定できます。
  - デフォルトの ACL（親フォルダの ACL）を使用する方法
  - 特定の ACL を指定する方法
- **ブートストラップ ACL:** ブートストラップ ACL 以外のすべての ACL は、別の ACL のコンテンツによって保護されます。

  `/sys/acls/bootstrap_acl.xml` に格納されたブートストラップ ACL は、それ自体のコンテンツによって保護される唯一の ACL です。すべてのデフォルト ACL は、ブートストラップ ACL によって保護されます。ブートストラップ ACL は、すべてのユーザーに `xdb:readContents` 権限を付与します。ブートストラップ ACL は、Oracle XML DB の `ADMIN` グループおよび `DBA` グループにフル・アクセス権を付与します。`XDBADMIN` ロールは、グローバル XML Schema の登録を必要とするユーザーには特に有効です。

- **Oracle XML DB によって提供されるその他の ACL:**
  - `all_all_acl.xml`: すべての権限をすべてのユーザーに付与します。
  - `all_owner_acl.xml`: すべての権限を所有者ユーザーに付与します。
  - `ro_all_acl.xml`: read 権限をすべてのユーザーに付与します。
- **ACL ファイル・ネーミング規則:** 提供される ACL は、ファイル・ネーミング規則 `privilege_users_acl.xml` を使用します。

  ここでの `privilege` は付与される権限を示し、`user` はリソースへのアクセスを許可されるユーザーを示します。

Oracle XML DB の ACL 機能

Oracle XML DB は、次の ACL 機能をサポートします。

ACL と Oracle XML DB の表またはビューのセキュリティの相互作用

ユーザーは、XML オブジェクトが格納されている基礎となる表またはビューに対する適切な権限、および個別のインスタンスに対する ACL を介した権限を所有している必要があります。

注意: 特定の表の（すべてではなく）一部のオブジェクトを Oracle XML DB リソースにマップすることができます。その場合、すべてのオブジェクトで表レベルのセキュリティが確保されますが、Oracle XML DB Repository の階層にマップされたオブジェクトのみで ACL の確認が行われます。

LDAP の統合およびユーザー ID

LDAP は Oracle XML DB に統合されているため、外部ユーザーに Oracle XML DB へのアクセスを許可できます。外部ユーザーは、ローカル・データベース・ユーザーと同じ操作を実行できます。

ACL 用の Oracle XML DB Resource API（PL/SQL）

ACL セキュリティ用の PL/SQL API では、PL/SQL 開発者がセキュリティ・メカニズムを使用して、特定の ACL が指定された権限を確認し、特定の ACL およびオブジェクトに対して現行ユーザーが所有している権限のセットをリストすることができます。

参照: 『Oracle9*i* XML API リファレンス - XDK および Oracle XML DB』を参照してください。

Oracle XML DB の ACL による並行性問題の解消

Oracle XML DB の ACL は、高速評価のためにキャッシュされます。ACL を変更するトランザクションがコミットされると、Oracle XML DB の構成ファイルに指定されたタイムアウトの経過後、変更済の ACL が取り出されます。この構成パラメータへの XPath は、`/xdbconfig/sysconfig/acl-max-age` です。

アクセス制御 : ユーザーおよびグループ・アクセス

プリンシパルは、個別のユーザーまたはグループのいずれかです。グループは、**コレクション**ともいいます。ユーザーにデータベース・ロールが付与されている場合、ユーザーにはグループ・プリンシパルとしてのアクセス権が付与されます。

各プリンシパルのアクセス権は、ACL のアクセス制御エントリ（ACE）に格納されます。

例 18-1 ユーザーおよびグループ・アクセスを制御する ACL の ACE エントリ

次に、ACL のエントリの例を示します。

```
<acl description="myacl"
      xmlns="http://xmlns.oracle.com/xdb/acl.xsd"
      xmlns:dav="DAV:"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://xmlns.oracle.com/xdb/acl.xsd
                          http://xmlns.oracle.com/xdb/acl.xsd">
  <ace>
    <principal>OWNER</principal>
    <grant>true</grant>
    <privilege>
      <all/>
    </privilege>
  </ace>
</acl>
```

ACE 要素によるプリンシパルに対するアクセス権の指定

前述の ACL によって、ドキュメントの所有者にすべての権限が付与されます。Oracle XML DB リソースへのアクセス権は、プリンシパルごとに付与されます。表 18-1 に、アクセス制御エントリ（ACE）の要素を示します。各 ACE 要素は、次の要素に設定された値を使用して、指定されたプリンシパルへのアクセス権を指定します。

表 18-1 アクセス制御エントリ（ACE）の要素

要素	説明
<principal>	プリンシパル（ユーザーまたはグループ）を指定します。
<grant>	プリンシパルにリソースへのアクセス権が付与されているかどうかを指定するブール値です。値が TRUE の場合、アクセス権が付与されています。FALSE の場合、アクセス権は付与されていません。
<privilege>	プリンシパルに付与する権限を指定します。

Oracle XML DB でサポートされる権限

Oracle XML DB は、Oracle XML DB リソースへのアクセスを制御する権限のセットを提供します。ACE のアクセス権は、privilege 要素に格納されます。権限は、次のとおりです。

- 集約権限（他の権限で構成されています）
- 基本権限（分割できません）

ACL が Oracle XML DB に格納される場合、集約権限はそのまま保持され、対応する基本権限に分割されません。WebDAV では、これらは非抽象的な集約権限であるため、ACE 内で使用できます。

- 基本権限
  - read-properties
  - read-contents
  - update
  - link（コンテナにのみ適用）
  - unlink（コンテナにのみ適用）
  - read-acl
  - write-acl-ref
  - update-acl
  - link-to

  - unlink-from
  - resolve
  - dav:lock
  - dav:unlock
- 集約権限
  - dav:read（read-properties、read-contents、resolve）
  - dav:write（update、link、unlink、unlink-from）
  - dav:read-acl（read-acl）
  - dav:write-acl（write-acl-ref、update-acl）
  - dav:all（dav:read、dav:write、dav:read-acl、dav:write-acl、dav:lock、dav:unlock）

基本権限

表 18-2 に、Oracle XML DB でサポートされている基本権限を示します。

表 18-2 基本権限

権限名	説明	データベースでの操作
read-properties	リソースのプロパティを読み込みます。	SELECT
read-contents	リソースのコンテンツを読み込みます。	SELECT
update	リソースのプロパティおよびコンテンツを更新します。	UPDATE
link	コンテナのみを対象とします。コンテナへリソースをバインドできます。	INSERT
unlink	コンテナのみを対象とします。コンテナからリソースのバインドを解除できます。	DELETE
link-to	リソースをリンクできます。	なし
unlink-from	リソースのリンクを解除できます。	なし
read-acl	リソースの ACL を読み込みます。	SELECT
write-acl-ref	リソースの ID を変更します。	UPDATE
update-acl	リソースの ACL のコンテンツを変更します。	UPDATE

表 18-2 基本権限 （続き）

権限名	説明	データベースでの操作
resolve	コンテナのみを対象とします。コンテナを全検索できます。	SELECT
dav:lock	WebDAV ロックを使用してリソースをロックします。	UPDATE
dav:unlock	WebDAV ロックを使用してロックされたリソースのロックを解除します。	UPDATE

> **注意：** 権限名は、XML 要素名です。dav: の接頭辞を持つ権限は、WebDAV 名前空間の一部であり、その他の権限は、Oracle XML DB の ACL 名前空間（`http://xmlns.oracle.com/xdb/acl.xsd`）の一部です。

ACL の `XMLType` 記憶域には直接アクセスできるため、XML 構造は、クライアント・インタフェースの一部になります。そのため、ACL は `XMLType` API を使用して操作できます。

集約権限

表 18-3 に、Oracle XML DB によって定義される集約権限およびそれらを構成する基本権限を示します。

表 18-3 集約権限

集約権限名	基本権限
all	すべての基本権限（dav:read、dav:write、dav:read-acl、dav:write-acl、dav:lock、dav:unlock）
dav:all	link-to を除くすべての基本権限
dav:read	read-properties、read-contents、resolve
dav:write	update、link、unlink、unlink-from
dav:read-acl	read-acl
dav:write-acl	write-acl-ref、update-acl

表 18-4 に、Oracle XML DB Repository のリソースで実行されるいくつかの一般的な操作に必要な権限を示します。必要な権限については、コンテナ C および階層のルート以下のすべての親コンテナに対して、ユーザーがすでに `resolve` 権限を所有していると想定しています。

表 18-4 Oracle XML DB のリソースへの操作に必要な権限

操作	説明	必要な権限
CREATE	コンテナ C に新しいリソースを作成します。	C に対する update および link。
DELETE	コンテナ C からリソース R を削除します。	R に対する update および unlink-from、C に対する update および unlink。
UPDATE	リソース R のコンテンツまたはプロパティを更新します。	R に対する update。
GET	リソース R の FTP GET または HTTP GET を実行します。	R に対する read-properties、read-contents。
SET_ACL	リソース R の ACL を設定します。	R に対する dav:write-acl。
LIST	コンテナ C のリソースをリストします。	C および C のリソースに対する read-properties。ユーザーが read-properties 権限を所有しているリソースのみがリストされます。

ACL の評価規則

ACL を評価するために、データベースは、現行データベース・セッションにログインしているユーザーに適用されている ACE のリストを収集します。指定されたユーザーに対して現在アクティブなロールのリストは、セッションの一部として保持され、ACE を現行のユーザーと一致させるために使用されます。ACE 間の競合を解消するには、「権限が任意の ACE によって拒否された場合、その権限は ACL 全体に対して拒否される」という規則が適用されます。

ACL のエントリは、次の規則に従う必要があります。

- 各プリンシパルは、最大 2 つの ACE を持ちます。1 つは権限を付与するための ACE、もう 1 つは権限を拒否するための ACE です。
- どのプリンシパルも、複数の grant ACE を持つことはできません。
- どのプリンシパルも、複数の deny ACE を持つことはできません。

Oracle XML DB ACL の使用

Oracle XML DB Repository の階層に存在するすべてのリソースには、ACL が関連付けられています。ACL のメカニズムによって、プリンシパルに権限ベースのリソースのアクセス制御が指定されます。リソースがアクセスされるたびに、セキュリティ・チェックが実行されます。ACL では、リソースにアクセスするためにどの権限のセットをどのプリンシパルが持つかを判断します。Oracle XML DB は、次のいずれかのプリンシパルを持つことができます。

- データベース・ユーザーなどの個別のプリンシパル
- 共通のロールを付与されたデータベース・ユーザーのグループなどのグループ・プリンシパル

各 ACL は、ACE のリストを含みます。ACE は、次の要素を含みます。

- この ACE が権限を付与するか拒否するかを示すブール値
- ACL が適用されるユーザーを示すプリンシパル（ユーザーまたはロールのいずれか）
- 付与または拒否されている権限のリスト

名前付き ACL は、名前属性およびオプションの型リストリクタ（`http://xmlns.oracle.com/xdb/XDBDemo.xsd#PurchaseOrder` など）も含みます。型リストリクタは、ACL がその XML 要素（およびその要素を含む代替グループの要素）のインスタンスのみに適用されることを指定します。ユーザーに対して付与または拒否のいずれもされていない権限は、拒否されているとみなされることに注意してください。

ACL を評価するために、データベースは、現行データベース・セッションにログインしているユーザーに適用されている ACE のリストを収集します。指定されたユーザーに対して現在アクティブなロールのリストは、セッションの一部として保持され、ACE を現行のユーザーと一致させるために使用されます。

ユーザーが特定の権限を所有しているかどうかを確認するには、ACL の ID および保護されているオブジェクトの所有者についての情報が必要です。Oracle XML DB の階層は、ACL の ID および所有者を、そのファイル・システムにマップされたオブジェクトに自動的に関連付けます（これらは、Oracle XML DB スキーマの表に格納されます）。

フォルダのデフォルト ACL の更新

例 18-2 フォルダのデフォルト ACL およびフォルダの所有者の更新

この例では、Oracle XML DB 管理者である `xdbadmn` および Oracle XML DB ユーザーである `xdbuser` という 2 人のユーザーを作成します。管理者は、「/」の下にユーザーのフォルダを作成します。このフォルダのデフォルト ACL は、親コンテナから継承されたもので、次の権限を付与します。

- 所有者へのすべての権限
- パブリックへの読込み権限のみ

resource_view を更新すると、フォルダの所有者がユーザーに変更されます。ACL を all_owner_acl.xml などの別のシステム ACL に変更することによって、ユーザーのフォルダを完全にプライベートにすることもできます。

```
connect system/manager

Rem Create an Oracle XML DB administrator user (has XDBADMIN role)
grant connect, resource, xdbadmin to xdbadmn identified by xdbadmn;

Rem Create Oracle XML DB user
grant connect, resource to xdbuser identified by xdbuser;

conn xdbadmn/xdbadmn

Rem create the user's folder
declare
   retval boolean;
begin
   retval := dbms_xdb.createfolder('/xdbuser');
end;
 /

Rem update the OWNER of the user folder
update resource_view
set res = updatexml(res, '/Resource/Owner/text()', 'XDBUSER')
where any_path = '/xdbuser';

commit;

connect xdbuser/xdbuser

Rem XDBUSER has full permissions to operate on her folder
declare
retval boolean;
begin
retval := dbms_xdb.createfolder('/xdbuser/workdir');
end;
 /

Rem All users can read /xdbuser folder at this time.
Rem change ACL to make folder completely private
call dbms_xdb.setacl('/xdbuser', '/sys/acls/all_owner_acl.xml');
```

ACL およびリソース管理

次の項では、Oracle XML DB Repository での ACL およびリソース管理について説明します。

> **参照：** 第 21 章「Oracle Enterprise Manager を使用した Oracle XML DB の管理」を参照してください。

リソース・プロパティの ACL を設定する方法

すべての Oracle XML DB リソースは、リソース・プロパティとして 1 つの ACL を持ちます。したがって、次のいずれかの方法を使用して、ACL のリソース・プロパティを設定できます。

- RESOURCE_VIEW の更新
- **PL/SQL API:** `DBMS_XDB.setacl(res_path VARCHAR2,acl_path VARCHAR2)` を使用して、`res_path` で表現される ACL のリソース・プロパティを、`acl_path` で表現される ACL に設定します。
- **FTP の quote コマンド:** 「`quote sacl <res_path> <acl_path>`」を使用して、`res_path` の ACL のリソース・プロパティを acl_path の `ACLOID` に設定します。

ACL のデフォルト割当て

Oracle XML DB の階層にリソースが挿入され、リソースが ACL を指定しない場合、リソースはその親コンテナの ACL を共有します。

リソース用の ACL の取得

次の `DBMS_XDB` API を使用して、指定されたリソースの ACL を取得できます。

```
DBMS_XDB.getAclDocument(res_path IN VARCHAR2)
```

これは、res_path に存在するリソースの ACL を表す `<acl>` 要素の `XMLType` インスタンスを戻します。

指定されたリソースに対する権限の変更

次の `DBMS_XDB` API を使用して、リソースの ACL に ACE を追加できます。

```
DBMS_XDB.changePrivileges(res_path IN VARCHAR2, ace IN XMLType)
```

ACL への操作の制限事項

すべての名前付き ACL は、Oracle XML DB Repository の階層内の XML Schema に基づくリソースです。Oracle XML DB Repository の階層内の他のリソースに使用されるすべてのメソッドも、ACL に使用できます。たとえば、FTP コマンド、PL/SQL DOM および XMLType メソッドは、ACL で実行できます。ただし、ACL はアクセス制御セキュリティ構造および Oracle XML DB Repository の階層の一部であるため、次の制限が適用されます。

- ACL の挿入 : ACL の特定のプリンシパルには、最大 1 つの grant ACE および 1 つの deny ACE を挿入できます。
- ACL の削除 : リソースが現在 ACL を使用している場合、ACL は削除できません。
- ACL の更新（変更）: ACL のリソースが非 ACL のコンテンツで更新される場合、ACL の削除と同じ規則が適用されます。

DBMS_XDB を使用した権限の確認

Oracle XML DB のアクセス制御は、次の DBMS_XDB ファンクションを使用して施行できます。

- CheckPrivileges、getAclDocument および getPrivileges（Oracle XML DB リソースの場合）。
- AclCheckPrivileges（データベース・オブジェクトの場合）。このファンクションは ACL をキャッシュからロードして、アクセス要求の評価を実行します（次の項を参照）。

アクセス制御セキュリティにおける行レベルのセキュリティ

Oracle XML DB の ACL セキュリティは、XML オブジェクトに対するデータベース・セキュリティとともに動作します。ユーザーは、XML オブジェクトが格納されている基礎となる表またはビューに対する適切な権限、および個別のインスタンスに対する ACL の権限を所有している必要があります。特定の表のオブジェクトが初めて Oracle XML DB の階層に格納およびリソースにマップされると、その表に行レベルのセキュリティ（RLS）・ポリシーが追加されます。このポリシーは、リソースにマップされた表の行についてのみ、ACL ベースの権限を確認します。RLS は、Oracle XML DB の階層の一部である XMLType の表またはビューに対して施行されます。

19

FTP、HTTP および WebDAV プロトコルの使用

この章では、FTP、HTTP および WebDAV プロトコルを使用して Oracle XML DB Repository にアクセスする方法を説明します。この章の内容は次のとおりです。

Oracle XML DB Protocol Server の概要

第 2 章「Oracle XML DB を使用する前に」および第 13 章「Oracle XML DB Foldering」で説明したとおり、Oracle XML DB Repository は、XML でモデル化されたデータベースに階層データ・リポジトリを提供します。Oracle XML DB Repository は、XMLType のデータベース・オブジェクトにパス名（または URL）をマップし、これらのオブジェクトの管理機能を提供します。

Oracle XML DB は、Oracle XML DB Protocol Server も提供します。Oracle XML DB は、その階層リポジトリまたはファイル・システムにアクセスするための標準インターネット・プロトコルである FTP、WebDAV および HTTP をサポートします。XML 文書は URL（通常は HTTP URL）を使用して相互参照するため、Oracle XML DB Repository およびそのプロトコル・サポートは、Oracle XML DB の重要なコンポーネントです。多くのユーザーは、これらのプロトコルによって、追加のソフトウェアをインストールせずに、Oracle XML DB に直接アクセスできます。

> **参照：** 13-10 ページの「インターネット・プロトコルを使用した Oracle XML DB リソースへのアクセス」を参照してください。

セッション・プーリング

Oracle XML DB Protocol Server は、セッションの共有プールを保持します。各プロトコル接続は、このプールの 1 つのセッションに関連付けられます。接続のクローズ後、セッションは共有プールに戻され、その後の接続で使用できます。

HTTP パフォーマンスの向上

セッション・プーリングでは、特に、リクエストごとに新しい接続が作成される HTTP 1.0 を使用する場合、セッション状態を再作成するコストが削減され、HTTP のパフォーマンスが向上します。たとえば、データベース・セッションを作成する必要がある場合に、既存の HTTP/1.1 接続でいくつかの小さいファイルを取り出すことができます。Oracle XML DB の `xdbconfig.xml` ファイルでセッション・プール・サイズを設定することによってセッションの数を調整したり、このプール・サイズを 0（ゼロ）に設定することによってセッションを使用禁止にすることができます。

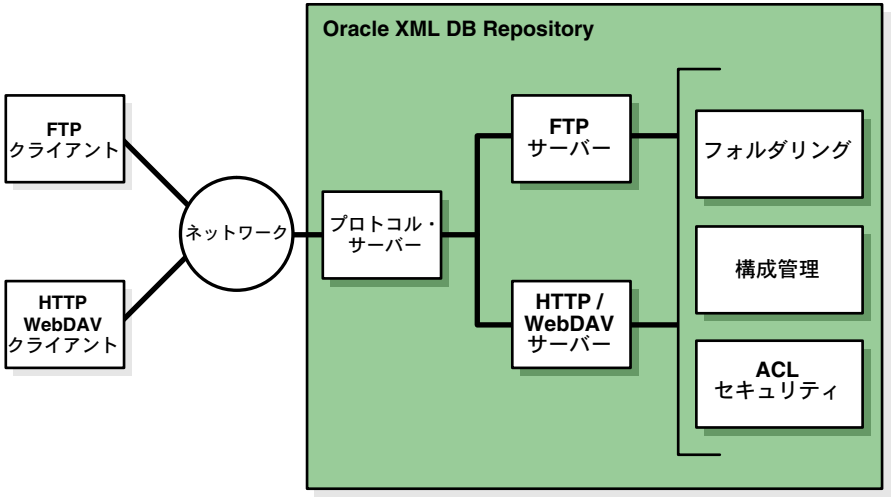
Java サーブレット

セッション・プーリングは、Java サーブレットを作成するユーザーに影響する可能性があります。これは、他のユーザーがアクセスして、別のユーザーの別の要求によって初期化されたセッションの状態を参照できるためです。そのため、サーブレットの作成者は、Java 静的変数などのセッション・メモリーのみを使用して、特定のユーザーではなくアプリケーション全体に対するデータを保持する必要があります。セッションが 1 人のユーザーに対してのみ存在すると想定するのではなく、ユーザーごとの状態をデータベースまたは参照表に格納する必要があります。

参照： 第 20 章「Java での Oracle XML DB アプリケーションの作成」を参照してください。

図 19-1 に、Oracle XML DB Protocol Server のコンポーネントおよびそれらを使用して Oracle XML DB XML リポジトリのファイルおよび他のデータにアクセスする方法を示します。この図では、リポジトリの関連するコンポーネントのみを示します。

図 19-1 Oracle XML DB のアーキテクチャ：プロトコル・サーバー



Oracle XML DB Protocol Server の構成管理

Oracle XML DB Protocol Server は、`/xdbconfig.xml` に格納された構成パラメータを使用して、起動状態の初期化およびセッション・レベルの構成を管理します。次の項では、Oracle XML DB 構成ファイルで構成可能な、プロトコル固有の構成パラメータについて説明します。

参照： 付録 A「Oracle XML DB のインストールおよび構成」を参照してください。

プロトコル・サーバーのパラメータの構成

表 19-1 に、すべてのプロトコルに共通のパラメータを示します。/xdbconfig で始まるパラメータを除き、この表に含まれるすべてのパラメータ名は、Oracle XML DB 構成スキーマの次の XPath に関連しています。

```
/xdbconfig/sysconfig/protocolconfig/common
```

- **FTP 固有のパラメータ :** 表 19-2 に FTP 固有のパラメータを示します。これらは、Oracle XML DB 構成スキーマの次の XPath に関連しています。

  ```
  /xdbconfig/sysconfig/protocolconfig/ftpconfig
  ```

- **サーブレット関連のパラメータを除く HTTP または WebDAV 固有のパラメータ :** 表 19-3 に、HTTP または WebDAV 固有のパラメータを示します。これらのパラメータは、Oracle XML DB 構成スキーマの次の XPath に関連しています。

  ```
  /xdbconfig/sysconfig/protocolconfig/httpconfig
  ```

これらのパラメータの使用例については、付録 A「Oracle XML DB のインストールおよび構成」に記載されている Oracle XML DB Repository の構成ファイル /xdbconfig.xml および G-18 ページの「xdbconfig.xsd: Oracle XML DB を構成するための XML Schema」を参照してください。

表 19-1 共通プロトコルの構成パラメータ

パラメータ	説明
extension-mappings/mime-mappings	ファイル拡張子のマッピングを MIME タイプに指定します。リソースが Oracle XML DB Repository に格納され、その MIME タイプが指定されていない場合、このマッピング・リストを使用してその MIME タイプが設定されます。
extension-mappings/lang-mappings	ファイル拡張子のマッピングを言語に指定します。リソースが Oracle XML DB Repository に格納され、その言語が指定されていない場合、このマッピング・リストを使用してその言語が設定されます。
extension-mappings/ encoding-mappings	ファイル拡張子のマッピングをエンコーディングに指定します。リソースが Oracle XML DB Repository に格納され、そのエンコーディングが指定されていない場合、このマッピング・リストを使用してそのエンコーディングが設定されます。
extension-mappings/charset-mappings	ファイル拡張子のマッピングをキャラクタ・セットに指定します。リソースが Oracle XML DB Repository に格納され、そのキャラクタ・セットが指定されていない場合、このマッピング・リストを使用してそのキャラクタ・セットが設定されます。

表 19-1 共通プロトコルの構成パラメータ（続き）

パラメータ	説明
session-pool-size	プロトコル・サーバーのセッション・プールに保持されるセッションの最大数を指定します。
/xdbconfig/sysconfig/call-timeout	接続がこの時間（100 分の 1 秒単位）の間アイドル状態であった場合、接続を行っている共有サーバーは、他の接続のために解放されます。
session-timeout	プロトコル・サーバーがセッション（およびそれに対応する接続）を終了するまでの、接続のアイドル状態の経過時間（100 分の 1 秒単位）を指定します。このパラメータは、特定のプロトコルのセッション・タイムアウトが構成に存在しない場合にのみ使用されます。
/xdbconfig/sysconfig/ default-lock-timeout	リソースの WebDAV ロックが無効になるまでの時間を指定します。これは、リソースをロックするクライアントが指定したタイムアウトでオーバーライドできます。

表 19-2 FTP 固有の構成パラメータ

パラメータ	説明
ftp-port	FTP サーバーがリスニングするポートを指定します。デフォルトは 2100 です。
ftp-protocol	FTP サーバーを実行するプロトコルを指定します。デフォルトは tcp です。
session-timeout	プロトコル・サーバーが FTP セッション（およびそれに対応する接続）を終了するまでの、接続のアイドル状態の経過時間（100 分の 1 秒単位）を指定します。

表 19-3 HTTP または WebDAV 固有の構成パラメータ（サーブレット・パラメータを除く）

パラメータ	説明
http-port	HTTP または WebDAV サーバーがリスニングするポートを指定します。
http-protocol	HTTP または WebDAV サーバーを実行するプロトコルを指定します。デフォルトは tcp です。
session-timeout	プロトコル・サーバーが HTTP セッション（およびそれに対応する接続）を終了するまでの、接続のアイドル状態の経過時間（100 分の 1 秒単位）を指定します。
server-name	HTTP レスポンスの Server ヘッダーの値を指定します。
max-header-size	HTTP ヘッダーの最大サイズ（バイト単位）を指定します。
max-request-body	HTTP Request Body の最大サイズ（バイト単位）を指定します。
webappconfig/welcome-file-list	「ウェルカム・ファイル」とみなされるファイル名のリストを指定します。サーバーは、コンテナに対する HTTP GET リクエストを受信すると、これらのいずれかの名前を持つコンテナのリソースが存在するかどうかを最初に確認します。そのようなリソースが存在する場合、コンテナ内のリソースのリストではなく、そのファイルのコンテンツが送信されます。
default-url-charset	受信 URL が UTF-8 またはリクエストの Content-Type フィールドの Charset パラメータに指定されたキャラクタ・セットでエンコードされていない場合に、HTTP プロトコル・サーバーは、受信 URL がこのキャラクタ・セットでエンコードされていると想定します。

Oracle XML DB ファイル・システムのリソースとの相互作用

プロトコル仕様の RFC 959（FTP）、RFC 2616（HTTP）および RFC 2518（WebDAV）では、暗黙的にサーバー側の抽象的な階層ファイル・システムを想定しています。これは、Oracle XML DB の階層リポジトリにマップされます。Oracle XML DB Repository は、次のような機能を提供します。

- 名前解決。
- ACL ベースのセキュリティ。
- すべてのコンテンツの格納および取出し。Oracle XML DB Repository には、FTP を介したバイナリ・データの入力と XML Schema に基づく文書の両方を格納できます。

参照： 次の Web サイトを参照してください。

- http://rfc.sunsite.dk/rfc/rfc959.html
- http://256.com/gray/docs/rfc2616/
- http://www.faqs.org/rfcs/rfc2518.html

プロトコル・サーバーによる XML Schema に基づく XML 文書または基づかない XML 文書の処理

Oracle XML DB Protocol Server は、挿入中の XML 文書がリポジトリに登録された XML Schema に基づいているかどうかを常に確認することによって、プロトコルを拡張します。

- 受信する XML 文書が XML Schema を指定する場合、使用する Oracle XML DB 記憶域はその XML Schema によって決定されます。この機能は、SQL 文を実行するのではなく、FTP や WebDAV などの単純なプロトコルを使用して、XML 文書をオブジェクトに関連した方法でデータベースに格納する必要がある場合に有効です。
- 受信する XML 文書が XML Schema に基づいていない場合、バイナリ・ドキュメントとして格納されます。

イベントベース・ロギング

プロトコル・サーバーが受信したリクエストおよびプロトコル・サーバーから送信されたレスポンスを記録すると有効な場合があります。これを行うには、イベント番号 31098 をレベル 2 に設定します。このイベントを設定するには、次の行を `init.ora` に追加し、データベースを再起動します。

```
event="31098 trace name context forever, level 2"
```

FTP および Oracle XML DB Protocol Server の使用

次の項では、Oracle XML DB でサポートされている FTP 機能について説明します。

Oracle XML DB Protocol Server: FTP 機能

ファイル転送プロトコル（FTP）は、インターネット上で最も古く、最も一般的なプロトコルの 1 つです。FTP は RFC959 で規定され、異機種間のファイル・システムへの均一な方法でのアクセスを提供します。FTP は、クライアントとサーバー間で通信するための定義済コマンドを提供することによって動作します。1 回の接続でコマンドの転送とステータスの取得が実行されます。ただし、データ転送用に、クライアントとサーバー間で新しい接続がオープンされます。HTTP では、1 回の接続でコマンドおよびデータの転送が実行されます。

FTP は、オペレーティング・システムの専用クライアント、ファイル・システムのエクスプローラ・クライアントおよびブラウザで実装されています。FTP は、一般的にセッション指向であり、明示的なログインによってユーザー・セッションが作成され、ファイルまたはディレクトリがダウンロードおよび参照された後、接続がクローズされます。

参照： RFC 959（FTP プロトコル仕様）を参照してください。

サポートされない FTP 機能

Oracle XML DB は、RFC 959 で定義された FTP を実装しますが、次のオプション機能はサポートされません。

- レコード指向のファイル。たとえば、STRU コマンドは FILE 構造のみサポートされています。これは、ファイルの転送に最も広範囲に使用されている構造です。これは、仕様で指定されているデフォルトでもあります。構造マウントはサポートされません。
- 追加。
- 割当て。ファイル転送の前に領域を事前に割り当てます。
- アカウント。安全性の低い Telnet プロトコルを使用します。
- 強制終了。

標準ポートまたは非標準ポートでの FTP の使用

Oracle XML DB の構成ファイル `/xdbconfig.xml` を使用して、任意のポートでリスニングするように構成できます。FTP は、非標準の保護されないポートでのリスニングを提供します。標準ポート（21）で FTP を使用するには、DBA が setuid ORACLE ではなく setuid ROOT への TNS リスナーに対して `chown` を実行する必要があります。

FTP サーバーのセッション管理

プロトコル・サーバーは、FTP プロトコルのセッション管理も提供します。FTP は、新しいコマンドの入力をショート・ウェイトした後、プロトコル・レイヤーに戻ります。共有サーバーは他の接続のために解放されます。このショート・ウェイト時間は、Oracle XML DB 構成ファイルの `call-timeout` パラメータを変更することによって構成できます。通信量が多いサイトでは、`call-timeout` を短くし、より多くの接続を確立できるようにする必要があります。接続で新しいデータが受信されると、FTP サーバーは最新のデータを使用して再起動されます。そのため、FTP が長時間実行する性質を持っていても、プロトコル・サーバーに確立可能な接続の数には影響しません。

HTTP および Oracle XML DB Protocol Server の使用

Oracle XML DB は、RFC2616 仕様に定義されている HyperText Transfer Protocol（HTTP）の HTTP 1.1 を実装します。今回のリリースでは、Oracle XML DB Protocol Server は、HTTP プロトコルの拡張機能である RFC 2109「HTTP State Management」もサポートします。これを「Cookie」といいます。

Oracle XML DB Protocol Server: HTTP 機能

Oracle XML DB Protocol Server の Oracle XML DB HTTP コンポーネントは、次のオプション機能を除いて、RFC2616 仕様を実装します。

- gzip および圧縮転送エンコーディング
- バイト範囲ヘッダー
- TRACE メソッド（プロキシ・エラーのデバッグに使用します。）
- キャッシュ制御ディレクティブ（コンテンツの期限日を指定する必要があります。一般には使用されません。）
- TE、Trailer、Vary および Warning ヘッダー
- 弱いエンティティ・タグ
- Web 共通ログ・フォーマット
- マルチホーム Web サーバー

参照： RFC 2616（HTTP 1.1 プロトコル仕様）を参照してください。

サポートされない HTTP 機能

RFC 2965 に指定された新しい Set-Cookie2 ヘッダーは、インターネットの世界ではまだほとんど使用されていないため、Oracle XML DB はこれを実装しません。Digest 認証（RFC 2617）はサポートされません。今回のリリースでは、Oracle XML DB は Basic 認証をサポートします。この認証では、クライアントが、ユーザー名およびパスワードを Authorization ヘッダーにクリアテキストで送信します。

標準ポートまたは非標準ポートでの HTTP の使用

HTTP は、非標準の保護されないポート（8080）でのリスニングを提供します。標準ポート（80）で HTTP を使用するには、DBA が setuid ORACLE ではなく setuid ROOT への TNS リスナーに対して chown を実行する必要があります。また、Oracle XML DB 構成ファイル /xdbconfig.xml にポート番号を構成する必要があります。

HTTP サーバーおよび Java サーブレット

Oracle XML DB は、Java サーブレットをサポートします。サーブレットを使用するには、その動作をカスタマイズするパラメータとともに、Oracle XML DB の構成ファイルに一意の名前で登録する必要があります。それをコンパイルして、データベースにロードする必要があります。最後に、そのサーブレット名をパターンに関連付ける必要があります。このパターンは、Java サーブレット API バージョン 2.2 に記述されているとおり、「*.jsp」などの拡張子または「/a/b/c」や「/sys/*」などのパス名になります。

HTTP リクエストの処理中、そのリクエストに対するパス名は登録済のパターンと一致します。一致する場合、プロトコル・サーバーは適切な初期化パラメータを使用して対応するサーブレットを起動します。Java サーブレットの場合、既存の Java Virtual Machine（JVM）インフラストラクチャが使用されます。これによって、必要に応じて JVM が起動され、Java メソッドが実行されて、サーブレットの初期化、レスポンスの作成、オブジェクトのリクエストが実行され、それらがサーブレットに渡されて実行されます。

> **参照：** 第 20 章「Java での Oracle XML DB アプリケーションの作成」を参照してください。

URL 内の ASCII 以外の文字

クライアントが URL にマルチバイト・データを含めて送信する場合、RFC 2718 では、クライアントが %HH フォーマット（HH は UTF-8 エンコーディングでのそのバイト値の 16 進文字列）を使用して URL を送信する必要があることが規定されています。次に、HTTP または WebDAV コンテキストのいずれかで XML DB に送信可能な URL の例を示します。

```
http://urltest/xyz%E3%81%82%E3%82%A2
http://%E3%81%82%E3%82%A2
http://%E3%81%82%E3%82%A2/abc%81%86%E3%83%8F.xml
```

XML DB は、リクエストされた URL（マルチバイト・データを含む IF ヘッダー内のすべての URL、DESTINATION ヘッダー内のすべての URL、および REFERRED ヘッダー内のすべての URL）を処理します。

クライアントが 16 進数値を UTF-8 にエンコードしていない場合、またはクライアントが %HH 表記規則に従わずにマルチバイト・データをそのまま送信した場合、XML DB は、特定のアルゴリズムに基づいて、表記規則に従っていないこれらのクライアントを処理します。URL は、次の順序で処理されます。

1. XML DB は UTF-8 でマルチバイト・データのデコードを試行します。
2. 手順 1 が有効でない場合、XML DB は `Content-Type` キャラクタ・セットのヘッダー値を使用します。
3. データがそのエンコーディングでエンコードされていない場合、`default-url-charset` 構成値が指定されているときは、XML DB はそのキャラクタ・セットを使用して値のデコードを試行します。
4. XML DB は、データベース・キャラクタ・セットでエンコードされていると想定します。

次に、表記方法に従っていない URL の例を示します。

```
http://urltest/ã?,ã??ã??ã?^
http://urltest/xyz%<shift-jis HH encoding>%<shift-jis HH encoding>
```

`default-url-charset` は、IANA 名である必要があります。

WebDAV および Oracle XML DB の使用

Web Distributed Authoring and Versioning（WebDAV）は、ユーザーに、インターネット上で Oracle XML リポジトリへのファイル・システム・インタフェースを提供するために使用される標準プロトコルです。WebDAV サーバー・フォルダにアクセスする最も一般的な方法は、Microsoft Windows 2000 または Windows NT の「Web フォルダ」を使用する方法です。

WebDAV は、HTTP 1.1 プロトコルへの拡張機能です。WebDAV を使用すると、クライアントは、メソッド、ヘッダー、Request Body フォーマットおよび Response Body フォーマットの一貫したセットを介して、リモート Web コンテンツのオーサリングを実行できます。WebDAV は、リソースの格納および取出し、リソース・コレクションのコンテンツの作成およびリスト、同時アクセスに対するリソースの調整式ロックおよびリソース・プロパティの設定および取出しを行う操作を提供します。

Oracle XML DB WebDAV の機能

Oracle XML DB は、次の WebDAV 機能をサポートします。

- フォルダリング（RFC2518 で指定済）
- アクセス制御

WebDAV は、HTTP プロトコルへの拡張機能のセットで、リモート Web サーバーのファイルを編集または管理できます。たとえば、WebDAV を使用して、次の操作を実行できます。

- インターネット上でのドキュメントの共有
- インターネット上でのコンテンツの編集

参照： RFC 2518（WebDAV プロトコル仕様）を参照してください。

Oracle XML DB でサポートされない WebDAV 機能

Oracle XML DB は、RFC2518 の内容をサポートしますが、次の例外があります。

- ロック NULL リソースは、長さが 0（ゼロ）の実際のリソースをファイル・システムに作成します。ロック NULL リソースは、フォルダには変換できません。
- WebDAV の ACL プロトコルおよびバインド・プロトコルの実装はサポートされません。
- 無限の深さのロックはサポートされません。
- Basic 認証のみサポートされます。

Oracle XML DB および WebDAV の使用 : Windows 2000 での Web フォルダの作成

Windows 2000 で Web フォルダを作成するには、次の手順を実行します。

1. デスクトップから、「マイ ネットワーク」を選択します。
2. 「ネットワーク プレースの追加」をダブルクリックします。
3. フォルダの場所を入力します。次に例を示します。

   ```
   http://[Oracle server name]:<HTTP port number>
   ```

   図 19-2 を参照してください。
4. 「次へ」をクリックします。
5. この Web フォルダを識別する任意の名前を入力します。
6. 「完了」をクリックします。

これで、Windows フォルダへのアクセスと同様に、Oracle XML DB Repository にアクセスできます。

図 19-2 Windows 2000 での Web フォルダの作成

20

Java での Oracle XML DB アプリケーションの作成

この章では、Oracle XML DB アプリケーションを Java で作成する方法を説明します。サーブレットを含む Java アプリケーションを作成する場合の設計のガイドライン、および Oracle XML DB Servlet を構成する方法を説明します。

この章の内容は次のとおりです。

Oracle XML DB Servlet の機能は、現在サポートされていません。

Oracle XML DB の Java アプリケーションの概要

Oracle XML DB は、Java プログラマに次の 2 つの主要なアーキテクチャを提供します。

- データベースでの Java Virtual Machine（JVM）の使用
- クライアント・サーバーまたはアプリケーション・サーバーでの JDBC Thick ドライバの使用

データベース内の Java は、データベース・サーバー・プロセスのコンテキストで実行されるため、Java コードを配置する方法は、次のいずれかに制限されます。

- SQL または PL/SQL から起動するストアド・プロシージャとして Java コードを実行できます。
- Java サーブレットを実行できます。

ストアド・プロシージャは、SQL コードおよび PL/SQL コードと簡単に統合でき、Oracle9*i* データベースにアクセスするプロトコルとして Oracle Net Services を使用する必要があります。

サーブレットは、Oracle9*i* データベースへの最上位のエントリ・ポイントとして、より適切に機能し、Oracle9*i* データベースにアクセスするプロトコルとして HTTP を使用する必要があります。

データベース内外で使用可能な Oracle XML DB API

今回のリリースでは、サーバーで実行するアプリケーションのみが使用できる Oracle XML DB API が存在します。表 20-1 に、今回のリリースの各アーキテクチャで使用可能な Oracle XML DB API を示します。

Oracle9*i* データベース内外で使用可能な Oracle XML DB API は、次のとおりです。

表 20-1 Oracle9*i* データベース内外で使用可能な Oracle XML DB API

Java Oracle XML DB API の説明	データベース内：Java サーブレットまたはストアド・プロシージャ	データベース外：JDBC（OCI）Thick ドライバの使用
XMLType に対する JDBC のサポート	使用可能	使用可能
XMLType クラス	使用可能	使用可能
Java DOM インプリメンテーション	使用可能	使用可能

設計のガイドライン : データベース内外での Java

Java で Oracle XML DB アプリケーションを作成するアーキテクチャを選択する場合、次のガイドラインを考慮します。

HTTP: Java サーブレットへのアクセスまたは XMLType リソースへの直接アクセス

ダウンストリーム・クライアントが XML をテキスト表現で処理する場合、HTTP を使用して Java サーブレットにアクセスするか、または XMLType リソースに直接アクセスする方法が最適です。この方法は、特に Java プログラムによる XML ノード・ツリーの操作が少ない場合に有効です。

サーバーに Java を実装すると、UCS-2 Unicode（Java 文字列に必要）で文字データを変換せずに、データをデータベースからネットワークへネイティブに移動できます。多くの場合、データはデータベースのバッファ・キャッシュから HTTP 接続に直接コピーされます。データをバッファ・キャッシュから、Oracle Net Services で使用される SQL のシリアル化フォーマットに変換し、それを JDBC クライアントに移して、XML に変換する必要はありません。データベース・サーバー内部では、XMLType のロード・オンデマンドおよび LRU キャッシュが最も効率的です。

多くの XMLType オブジェクト要素へのアクセス : JDBC XMLType サポートの使用

ダウンストリーム・クライアントが、Java プログラムによって XMLType オブジェクトの多くまたはほとんどの要素にアクセスするアプリケーションである場合、通常、JDBC XMLType サポートを使用する方法が最適です。また、Java プログラムのデバッグは、通常、データベース・サーバーの外部で行う方が簡単です。

サーブレットを使用したデータの操作および XML としての迅速な書込み

Oracle XML DB Servlet は、HTTP を使用してアクセス可能な「HTTP ストアド・プロシージャ」を Java で作成するためのものです。インターネット・アプリケーション全体を開発することを目的としたプラットフォームではありません。その場合、Oracle9*i*AS アプリケーション・サーバーにアプリケーション・サーブレットを配置し、JDBC を使用するか、または java.net.* や同様の API を使用してデータベース内のデータにアクセスし、HTTP を使用して XML データを取得する必要があります。

サーブレットは、エンドユーザーのために HTML ページをフォーマットするのではなく、データベースにアクセスし、データを操作して、そのデータを XML として迅速に書き込むアプリケーションに最適です。

Java での Oracle XML DB HTTP Servlet の作成

Oracle XML DB は、FTP、HTTP 1.1、WebDAV および Java サーブレットをサポートするプロトコル・サーバーを提供します。今回のリリースでは、Java サーブレットのサポートは不完全ですが、今後のリリースで、完全に準拠するためのサブセットが提供されます。現在、Oracle XML DB は、Java サーブレットのバージョン 2.2 をサポートします。ただし、次の例外があります。

- Servlet WAR ファイル（`web.xml`）はサポートされません。手動で操作する必要がある `web.xml` 構成パラメータが存在します。たとえば、ロールの作成は、SQL の `CREATE ROLE` コマンドを使用して行う必要があります。
- `RequestDispatcher` および関連付けられたメソッドはサポートされません。
- `HTTPServletRequest.getCookies()` メソッドはサポートされません。
- 現在、1 つの `ServletContext`（および 1 つの Web アプリケーション）のみがサポートされています。
- ステートフル・サーブレット（および `HttpSession` クラスのメソッド）は、サポートされません。サーブレットは、データベース自体の状態を保持する必要があります。

Oracle XML DB Servlet の構成

Oracle XML DB Servlet は、リポジトリの `/xdbconfig.xml` ファイルを使用して構成されます。このファイル内の多くの XML 要素は、Java 2 Enterprise Edition（J2EE）の Java サーブレット 2.2 仕様部分に定義されているものと同じであり、同じセマンティクスを持ちます。表 20-2 に、Java サーブレットの仕様でサーブレット・デプロイメント・ディスクリプタに対して定義された XML 要素、および Oracle XML DB がサポートする拡張要素を示します。

表 20-2 サーブレット・デプロイメント・ディスクリプタに対して定義された XML 要素

XML 要素名	定義元	サポートの有無	説明	コメント
auth-method	Java	非サポート	アクセスに必要な HTTP 認証方式を指定します。	--
charset	Oracle	サポート	IANA キャラクタ・セット名を指定します。	「ISO8859」、「UTF8」などです。
charset-mapping	Oracle	サポート	ファイル名の拡張子とキャラクタ・セットの間のマッピングを指定します。	--
context-param	Java	非サポート	Web アプリケーション用のパラメータを指定します。	現在はサポートされていません。
description	Java	サポート	サーブレットまたは Web アプリケーションを説明する文字列です。	サーブレット用にサポートされています。

表 20-2 サーブレット・デプロイメント・ディスクリプタに対して定義された XML 要素 （続き）

XML 要素名	定義元	サポートの有無	説明	コメント
display-name	Java	サポート	サーブレットまたは Web アプリケーションで表示する文字列です。	サーブレット用にサポートされています。
distributable	Java	非サポート	すべてのインスタンスが同じ Java Virtual Machine で実行していない場合に、このサーブレットが機能できるかどうかを示します。	Oracle9*i* データベースで実行するすべてのサーブレットは、distributable である必要があります。
errnum	Oracle	サポート	Oracle エラー番号です。	『Oracle9*i* データベース・エラー・メッセージ』を参照してください。
error-code	Java	サポート	HTTP エラー・コードです。	RFC 2616 で定義されています。
error-page	Java	サポート	エラーが発生した場合のリダイレクト先の URL を定義します。	HTTP エラー、不明な Java 例外または不明な Oracle エラー・メッセージで指定できます。
exception-type	Java	サポート	エラー・ページにマップされた Java 例外のクラス名です。	--
extension	Java	サポート	MIME タイプやキャラクタ・セットなどに関連付けるために使用するファイル名の拡張子です。	--
facility	Oracle	サポート	エラー・ページをマッピングするための Oracle 機能コードです。	「ORA」、「PLS」などです。
form-error-page	Java	非サポート	フォーム・ログインの試行に対するエラー・ページです。	現在はサポートされていません。
form-login-config	Java	非サポート	フォームベースのログインの構成仕様です。	現在はサポートされていません。
form-login-page	Java	非サポート	フォームベースのログイン・ページの URL です。	現在はサポートされていません。
icon	Java	サポート	サーブレットに関連付けられたアイコンの URL です。	サーブレット用にサポートされています。
init-param	Java	サポート	サーブレットの初期化パラメータです。	--
jsp-file	Java	非サポート	サーブレットに使用する JavaServer Pages ファイルです。	サポートされていません。

表 20-2 サーブレット・デプロイメント・ディスクリプタに対して定義された XML 要素 （続き）

XML 要素名	定義元	サポートの有無	説明	コメント
lang	Oracle	サポート	IANA 言語名です。	en-US などです。
lang-mapping	Oracle	サポート	ファイル名の拡張子と言語コンテンツの間のマッピングを指定します。	--
large-icon	Java	サポート	アイコン表示用の大きいサイズのアイコンです。	--
load-on-startup	Java	サポート	サーブレットの起動時にロードするかどうかを指定します。	--
location	Java	サポート	エラー・ページの URL を指定します。	ローカル・パス名または HTTP URL を指定できます。
login-config	Java	非サポート	認証方式を指定します。	現在はサポートされていません。
mime-mapping	Java	サポート	コンテンツのファイル名の拡張子と MIME タイプの間のマッピングを指定します。	--
mime-type	Java	サポート	リソース・コンテンツの MIME タイプ名です。	text/xml、application/octet-stream などです。
OracleError	Oracle	サポート	エラー・ページに関連付ける Oracle エラーを指定します。	--
param-name	Java	サポート	サーブレットまたは `ServletContext` のパラメータ名です。	サーブレット用にサポートされています。
param-value	Java	サポート	パラメータの値です。	--
realm-name	Java	非サポート	認証に使用する HTTP レルムです。	現在はサポートされていません。
role-link	Java	サポート	特定のユーザーがサーブレットにアクセスするために必要なロールを指定します。	データベース・ロール名を参照します。デフォルトでは、大文字にする必要があります。
role-name	Java	サポート	ロールのサーブレット名です。	データベース・ロールを呼び出すための別の名前です。サーブレットの API で使用します。
security-role	Java	非サポート	サーブレットが使用するロールを定義します。	サポートされていません。ロールは、SQL の `CREATE ROLE` を使用して手動で作成する必要があります。

表 20-2 サーブレット・デプロイメント・ディスクリプタに対して定義された XML 要素 （続き）

XML 要素名	定義元	サポートの有無	説明	コメント
security-role-ref	Java	サポート	サーブレットとロール間の参照です。	--
servlet	Java	サポート	サーブレットの構成情報です。	--
servlet-class	Java	サポート	Java サーブレットのクラス名を指定します。	--
servlet-language	Oracle	サポート	サーブレットを作成するプログラミング言語を指定します。	Java、C、PL/SQL のいずれかです。現在、顧客定義のサーブレットには、Java のみがサポートされています。
servlet-mapping	Java	サポート	サーブレットを関連付けるファイル名のパターンを指定します。	Java で定義されたすべてのマッピングがサポートされています。
servlet-name	Java	サポート	サーブレットの文字列名です。	サーブレットの API で使用します。
servlet-schema	Oracle	サポート	Java クラスがロードされる Oracle スキーマです。指定しない場合、デフォルトのリゾルバ仕様を使用してスキーマが検索されます。	これが指定されない場合、サーブレットを SYS スキーマにロードして、すべてのユーザーがアクセスできるようにするか、またはデフォルトの Java クラス・リゾルバを変更する必要があります。servlet-schema の値は、二重引用符で囲まないかぎり大文字になることに注意してください。
session-config	Java	非サポート	`HTTPSession` の構成情報です。	`HTTPSession` はサポートされません。
session-timeout	Java	非サポート	HTTPSession のタイムアウトです。	`HTTPSession` はサポートされません。
small-icon	Java	サポート	サーブレットに関連付けられた小さいアイコンです。	--
taglib	Java	非サポート	JSP タグ・ライブラリです。	JSP は現在サポートされていません。
taglib-uri	Java	非サポート	web.xml ファイルに相対的な JSP タグ・ライブラリの記述ファイルの URI です。	JSP は現在サポートされていません。
taglib-location	Java	非サポート	タグ・ライブラリが格納されている Web アプリケーションのルートに相対的なパス名です。	JSP は現在サポートされていません。

表 20-2 サーブレット・デプロイメント・ディスクリプタに対して定義された XML 要素 （続き）

XML 要素名	定義元	サポートの有無	説明	コメント
url-pattern	Java	サポート	サーブレットに関連付けられた URL パターンです。	Java サーブレット 2.2 仕様の第 10 項を参照してください。
web-app	Java	非サポート	Web アプリケーションの構成です。	現在、1 つの Web アプリケーションのみがサポートされています。
welcome-file	Java	サポート	ウェルカム・ファイルの名前を指定します。	--
welcome-file-list	Java	サポート	HTTP GET を使用してフォルダが参照された場合に表示するファイルのリストを定義します。	`index.html` などです。

注意：

- **1.** Java によって `web.xml` ファイルに定義されたパラメータ（env-entry、env-entry-name、env-entry-value、env-entry-type、ejb-ref、ejb-ref-type、home、remote、ejb-link、resource-ref、res-ref-name、res-type、res-auth）は、J2EE 準拠の EJB コンテナでのみ使用でき、完全な J2EE 環境をサポートしない Java Servlet コンテナには必要ありません。
- **2.** security-constraint、web-resource-collection、web-resource-name、http-method、user-data-constraint、transport-guarantee および auth-constrain 要素は、リソースにアクセス制御を定義するために使用されます。Oracle XML DB は、アクセス制御リスト（ACL）を使用してこの機能を提供します。`web.xml` ファイルを使用した ACL の生成は、今後のリリースでサポートされます。

参照： `/xdbconfig.xml` ファイルの構成の詳細は、付録 A「Oracle XML DB のインストールおよび構成」を参照してください。

Oracle XML DB Servlet に対する HTTP リクエストの処理

Oracle XML DB は、次の手順を実行して HTTP リクエストを処理します。

1. 接続が確立されていない場合、Oracle リスナーは接続を共有サーバー・ディスパッチャに渡します。
2. 新しい HTTP リクエストを受信すると、ディスパッチャは共有サーバーを起動します。
3. HTTP ヘッダーは、適切な構造に解析されます。
4. 共有サーバーは、XML DB セッション・プールが使用可能な場合、そこからデータベース・セッションの割当てを試行します。XML DB セッション・プールが使用できない場合、新しいセッションを作成します。
5. 新しいデータベース・コールおよび新しいデータベース・トランザクションが開始されます。
6. HTTP に認証ヘッダーが含まれている場合、セッションはそのデータベース・ユーザーとして SQL*Plus にログインしている場合と同様に認証されます。認証情報が含まれず、リクエストが GET または HEAD である場合、Oracle XML DB はセッションを ANONYMOUS ユーザーとして認証しようとします。このデータベース・ユーザーのアカウントがロックされている場合、認証されないアクセスは許可されません。
7. Java サーブレット 2.2 の仕様で指定されているとおり、HTTP リクエストの URL が `xdbconfig.xml` ファイルのサーブレットと一致しているかどうかが確認されます。
8. XML DB Servlet コンテナは、Oracle 内部の JVM で起動されます。指定したサーブレットが初期化されていない場合、初期化されます。
9. サーブレットは、`ServletInputStream` から入力を読み込み、`ServletOutputStream` に出力を書き込み、`service()` メソッドから戻ります。
10. 不明な Oracle エラーが発生した場合、セッションはセッション・プールに戻されます。

> **参照：** 第 19 章「FTP、HTTP および WebDAV プロトコルの 使用」を参照してください。

セッション・プールおよび XML DB Servlet

Oracle データベースは、データベース・セッションごとに 1 つの JVM を保持します。これは、セッション・プールから再利用されるセッションが、セッションが最後に使用されてからは、JVM（Java 静的変数）内でどのような状態にもなることを意味します。

これは、メタデータなど、ユーザー固有でない Java 状態をキャッシュする場合に有効ですが、**保護ユーザーのデータは Java 静的メモリーに格納しないでください**。保護ユーザーのデータを Java 静的メモリーに格納した場合、アプリケーションによってセキュリティ・ホールが出現する可能性があります。

ネイティブな XML ストリームのサポート

DOM ノード・クラスは、write() という Oracle 固有のメソッドを持ちます。このメソッドは次の引数を取り、void を戻します。

- java.io.OutputStream ストリーム : XML テキストの書込み先の Java ストリーム。
- String charEncoding: XML テキストを書き込むキャラクタ・コード。NULL の場合、データベース・キャラクタ・セットが使用されます。
- Short indent: ネストした XML 要素のインデントの文字数。

提供されたストリームがデータベース内部で提供される ServletOutputStream である場合、このメソッドには、ショートカットが実装されます。ノードのコンテンツは、出力ソケットにネイティブなコードの XML で直接書き込まれます。これによって、Java オブジェクトまたは Unicode（Java 文字列に必要）との間の変換が回避され、パフォーマンスが向上します。

Oracle XML DB Servlet の API

Oracle XML DB Servlet でサポートされる API は、Java サーブレット 2.2 仕様で定義されます。現在、Java サーブレット 2.2 仕様の Javadoc は、次のサイトで参照できます。
http://java.sun.com/products/servlet/2.2/javadoc/index.html

表 20-3 に、実装されない Java サーブレット 2.2 のメソッドを示します。今回のリリースでこれらのメソッドを使用すると、実行時に例外が発生します。

表 20-3 実装されない Java 2.2 のメソッド

インタフェース	メソッド
HttpServletRequest	getSession()、isRequestedSessionIdValid()
HttpSession	すべて
HttpSessionBindingListener	すべて

Oracle XML DB Servlet の例

次に、URL に指定されたパラメータをパス名として読み込み、その XML 文書のコンテンツを出力ストリームとして書き込む単純なサーブレットの例を示します。

例 20-1 Oracle XML DB Servlet の作成

サーブレット・コードは、次のようになります。

```
/* test.java */
import javax.servlet.http.*;
import javax.servlet.*;
import java.util.*;
import java.io.*;
import javax.naming.*;
import oracle.xdb.dom.*;

public class test extends HttpServlet
{
  protected void doGet(HttpServletRequest req, HttpServletResponse resp)
    throws ServletException, IOException
  {
    OutputStream os = resp.getOutputStream();
    Hashtable    env = new Hashtable();
    XDBDocument  xt;

    try
    {
      env.put(Context.INITIAL_CONTEXT_FACTORY,
              "oracle.xdb.spi.XDBContextFactory");
      Context ctx = new InitialContext(env);
      String [] docarr  = req.getParameterValues("doc");
      String doc;

      if (docarr == null || docarr.length == 0)
        doc = "/foo.txt";
      else
        doc = docarr[0];
      xt = (XDBDocument)ctx.lookup(doc);
      resp.setContentType("text/xml");
      xt.write(os, "ISO8859", (short)2);
    }
    catch (javax.naming.NamingException e)
    {
      resp.sendError(404, "Got exception: " + e);
    }
    finally
    {
      os.close();
    }
  }
}
```

Oracle XML DB のサンプル・サーブレットのインストール

このサーブレットをインストールしてコンパイルし、Oracle9*i* データベースにロードするには、次のコマンドを使用します。

```
% loadjava -grant public -u scott/tiger -r test.class
```

Oracle XML DB のサンプル・サーブレットの構成

Oracle XML DB Servlet を構成し、/xdbconfig.xml ファイルを更新するには、次の XML 要素ツリーを <servlet-list> 要素に挿入します。

```
<servlet>
  <servlet-name>TestServlet</servlet-name>
  <servlet-language>Java</servlet-language>
  <display-name>XML DB Test Servlet</display-name>
  <servlet-class>test</servlet-class>
  <servlet-schema>scott</servlet-schema>
</servlet>
```

/xdbconfig.xml ファイルを更新するには、次の XML 要素ツリーを <servlet-mappings> 要素に挿入します。

```
<servlet-mapping>
  <servlet-pattern>/testserv</servlet-pattern>
  <servlet-name>TestServlet</servlet-name>
</servlet-mapping>
```

/xdbconfig.xml ファイルは、任意の WebDAV 対応テキスト・エディタまたは SQL 演算子 updateXML() を使用して編集できます。

注意： ユーザーが SYS である場合でも、/xdbconfig.xml ファイルを実際に削除することはできません。

サンプル・サーブレットのテスト

サンプル・サーブレットをテストするには、任意の XML ファイルを /foo.xml にロードして、ブラウザに次の URL を入力します。この場合、hostname およびポート番号に適切な値を代入します。

```
http://hostname:8080/testserv?doc=/foo.xml
```

第 VI 部

Oracle XML DB をサポートする Oracle のツール製品

第 VI 部では、XML データをロードするための Oracle SQL*Loader およびインポート / エクスポート・ユーティリティの概要を説明します。また、Oracle Enterprise Manager を使用して XML データベース・アプリケーションを管理する方法も説明します。

第 VI 部に含まれる章は、次のとおりです。

21

Oracle Enterprise Manager を使用した Oracle XML DB の管理

この章では、Oracle Enterprise Manager を使用して Oracle XML DB を管理する方法を説明します。Oracle Enterprise Manager を使用すると、リポジトリのリソース、および XML Schema や XMLType 表などのデータベース・オブジェクトを構成、作成および管理できます。

この章の内容は次のとおりです。

Oracle XML DB および Oracle Enterprise Manager の概要

この章では、Oracle Enterprise Manager（Enterprise Manager）を使用して Oracle XML DB を管理する方法を説明します。

Oracle Enterprise Manager および Oracle XML DB を使用する前に

Oracle Enterprise Manager は、Oracle9*i* データベース・ソフトウェアの Enterprise Edition および Standard Edition の両方に付属しています。Oracle XML DB の機能をサポートする Enterprise Manager バージョンを実行するには、Oracle9*i* データベース リリース 2（9.2）以上を使用します。

Enterprise Manager: Oracle XML DB のインストール

Oracle XML DB は、Database Configuration Assistant（DBCA）を使用して新しいデータベースを作成するときに、デフォルトでインストールされます。Oracle XML DB のインストール時には、次の操作が行われます。

- Oracle によって、次の構成 XML Schema が登録されます。

  `http://xmlns.oracle.com/xdb/xdbconfig.xsd`

- Oracle によって、デフォルトの構成ドキュメントが挿入されます。構成 XML Schema に準拠するリソース `/xdbconfig.xml` が作成されます。このリソースには、すべての Oracle XML DB パラメータに対するデフォルト値が含まれます。

  参照： 次の章または付録を参照してください。

  - 第 2 章「Oracle XML DB を使用する前に」
  - 付録 A「Oracle XML DB のインストールおよび構成」

Oracle XML DB への XML Schema の登録（必須）

通常、Oracle XML DB は、高速な取出しおよび検索、アクセス制御、および XML 文書のバージョニングに使用されます。データベースに保存された XML インスタンス・ドキュメントを XML Schema に準拠させることができます。XML Schema は、それ自体も XML で作成されているスキーマ定義言語で、XML Schema に準拠する XML インスタンス・ドキュメントの構造およびその他の様々なセマンティクスを記述するために使用できます。

Oracle XML DB は、データベースに XML Schema を登録するメカニズムを提供します。

XML Schema 文書が存在する場合は、はじめに XML Schema を登録します。XML Schema を登録する前に、次のことを確認する必要があります。

1. **XML インスタンス・ドキュメントのデータがリレーショナル表に存在するかどうか。**
   レガシー・アプリケーションの場合、データが存在する場合があります。この場合、XML 用のオブジェクト・ビューを作成する必要があります。

2. **記憶域モデルの種類。**LOB 記憶域を使用するか、またはオブジェクト・リレーショナル記憶域を使用するか、あるいはその両方を使用するか。使用する記憶域モデルは、最も頻繁に問合せが行われるために、高速な取出しが必要なドキュメントの部分によって異なります。
3. **XML Schema 文書に、オブジェクト・データ型およびオブジェクト表を生成するためのコメントが注釈として追加されているかどうか。**このようなコメントが注釈として追加されていない場合、これらのオブジェクトの作成およびデータベース・スキーマへのマッピングを手動で行う必要があります。

参照： 次の章を参照してください。

- 第 3 章「Oracle XML DB の使用」
- 第 5 章「XMLType の構造化されたマッピング」

多くの場合、オブジェクト型およびオブジェクト表を自動生成するための情報が、XML Schema に注釈として付いていると想定されます。したがって、Oracle9*i* データベースでは、XML Schema の登録の一部として、これらのオブジェクトが自動生成されます。

参照： 21-28 ページの「XML Schema および関連するデータベース・オブジェクトの管理」を参照してください。

これで、XML Schema に準拠する XML 文書を Oracle XML DB によって挿入できます。

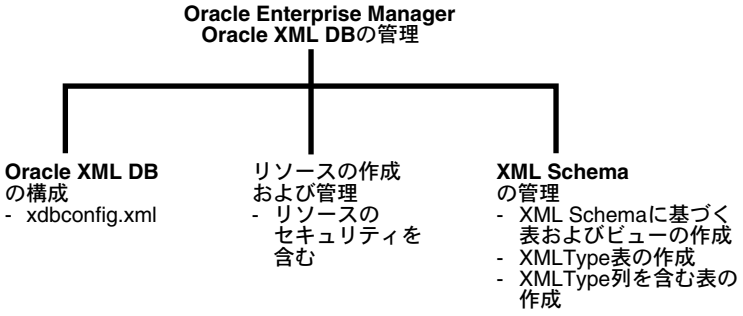
Oracle Enterprise Manager の Oracle XML DB 機能

Enterprise Manager を使用すると、次の主な Oracle XML DB 管理タスクを実行できます。

- Oracle XML DB の構成
- リソースの作成および管理
- XML Schema および関連するデータベース・オブジェクトの管理

図 21-1 および図 21-2 を参照してください。

図 21-1 Enterprise Manager を使用した Oracle XML DB の管理 : 主なタスク



Oracle XML DB の構成

Oracle XML DB は、構成ファイル `xdbconfig.xml` を介して管理されます。Enterprise Manager を介して、このファイルのパラメータを表示または構成できます。Oracle XML DB の構成オプションにアクセスするには、Enterprise Manager の右側のウィンドウで、「Configure XML Database」を選択します。21-7 ページの「Enterprise Manager を使用した Oracle XML DB の構成」を参照してください。

リソースの作成および管理

XML リソースの管理オプションにアクセスするには、ナビゲータで XML データベース・オブジェクトを選択し、詳細ビューで「Create a resource」をクリックします。21-12 ページの「Enterprise Manager を使用した Oracle XML DB のリソースの作成および管理」を参照してください。

XML Schema および関連するデータベース・オブジェクトの管理

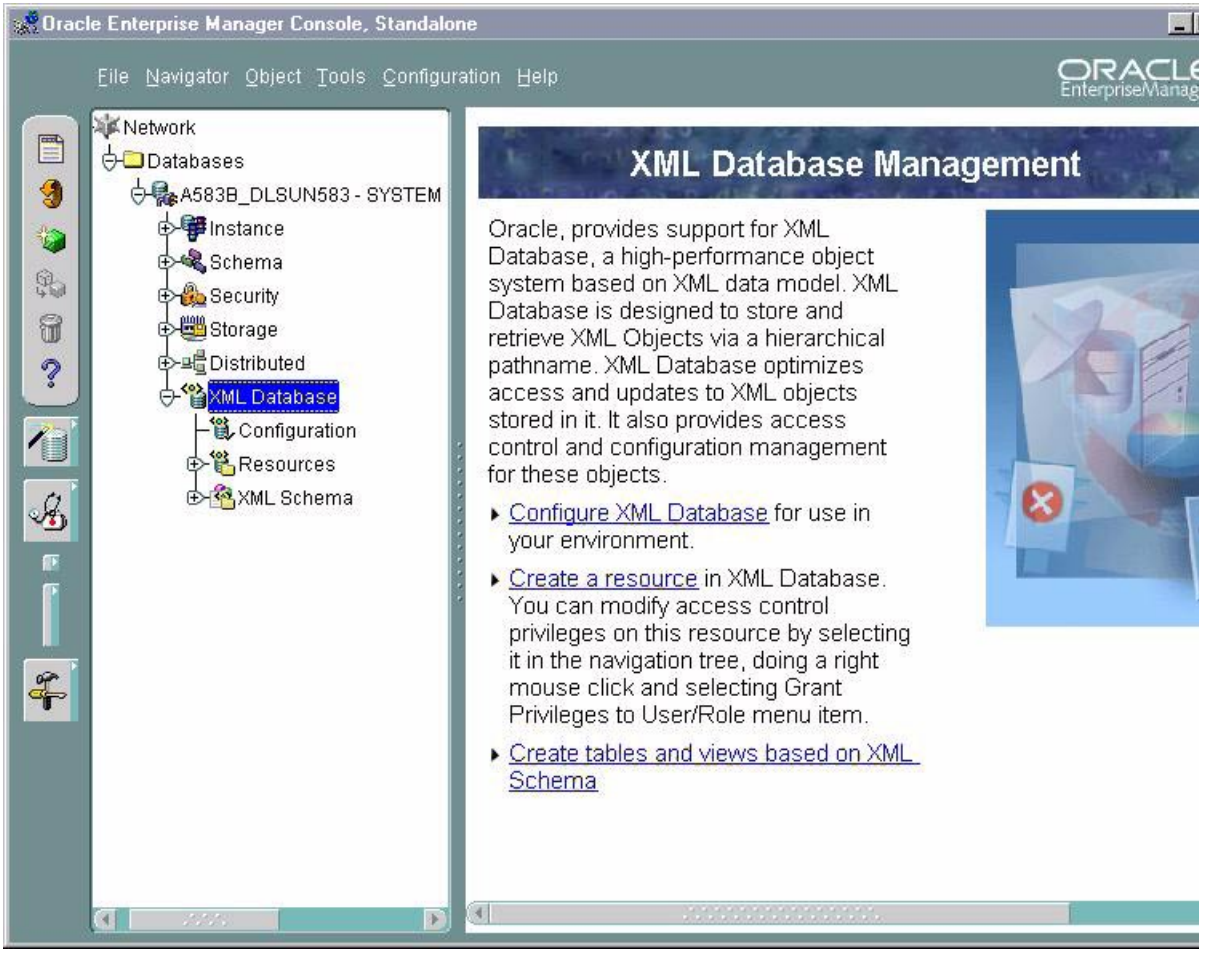
XML Schema の管理オプションにアクセスするには、ナビゲータで XML データベース・オブジェクトを選択し、詳細ビューで「Create tables and views based on XML Schema」をクリックします。21-28 ページの「XML Schema および関連するデータベース・オブジェクトの管理」を参照してください。

XML Schema を登録、削除、表示、生成（リバース・エンジニアリング）し、依存性を確認できます。XML Schema のコンテンツを表示し、構成要素に対して操作を実行することもできます。

- **ビュー :** `XMLType` のオブジェクト・ビューを作成または変更し、対応する索引を参照します。
- **表 :** XML Schema に基づく XMLType 表および列と基づかない `XMLType` 表および列を作成し、対応する索引を参照します。これらの列に LOB 記憶域属性を指定することもできます。

  CLOB またはオブジェクト・リレーショナル形式で `XMLType` 表を作成し、非表示の `XMLType` 列に対して制約および LOB 記憶域属性を指定できます。
- **索引 :** `XMLType` の非表示列に対する索引を作成します。
- **DML 操作 :** XML インスタンス・ドキュメントを使用して、行の挿入や更新など、その他の DML 操作を実行することもできます。
- **XML Schema 要素 :** XML 形式の要素、およびその要素に対応するデータベースに格納された XML インスタンス・データを表示できます。表、ビュー、索引、オブジェクト型、配列型、表型など、XML Schema の依存オブジェクトを表示することもできます。

図 21-2 Enterprise Manager コンソール：「XML Database Management」ウィンドウ



Oracle XML DB 用の Enterprise Manager コンソール

図 21-2 を参照してください。Enterprise Manager コンソールから、Oracle XML DB オブジェクトを迅速に参照および管理できます。

「XML Database Management」ウィンドウ : 右側のダイアログ・ウィンドウ

「XML Database Management」ウィンドウの詳細ビューから、XML DB の管理機能にアクセスできます。この詳細ビューから、実行する必要があるタスクを選択できます。

階層ナビゲーション・ツリー : ナビゲータ

左側のナビゲータを使用して、表示または変更する必要がある Oracle XML DB のリソースおよびデータベース・オブジェクトを選択します。

Enterprise Manager を使用した Oracle XML DB の構成

Oracle XML DB の構成は、Oracle XML DB に統合されています。これは、HTTP、WebDAV、FTP などのプロトコル、および ACL ベースのセキュリティなどのカスタマイズが可能な Oracle XML DB の他のコンポーネントによって使用されます。

Oracle XML DB の構成は、XML Schema に基づく XML リソース `xdbconfig.xml` として Oracle XML DB Repository に格納されます。これは、次の場所に格納された Oracle XML DB の構成 XML Schema に準拠します。

```
http://xmlns.oracle.com/xdb/xdbconfig.xsd
```

この構成 XML Schema は、Oracle XML DB のインストール時に登録されます。構成プロパティ・シートには、次の 2 つのタブがあります。

- **システム構成の場合** :「System Configurations」タブに、一般的なパラメータ、および FTP プロトコルと HTTP プロトコルに固有のパラメータが表示されます。
- **ユーザー構成の場合** :「User Configurations」タブに、カスタム・パラメータが表示されます。

Oracle XML DB で項目を構成するには、ナビゲータの「XML Database」の下の「**Configuration**」ノードを選択します。図 21-3 および図 21-4 を参照してください。

「**XML Database Parameters**」ページには、現在の XML データベースに対する構成パラメータのリストが表示されます。「XML Database Management」メイン・ウィンドウから「Configure XML Database」を選択して、このページを表示することもできます。

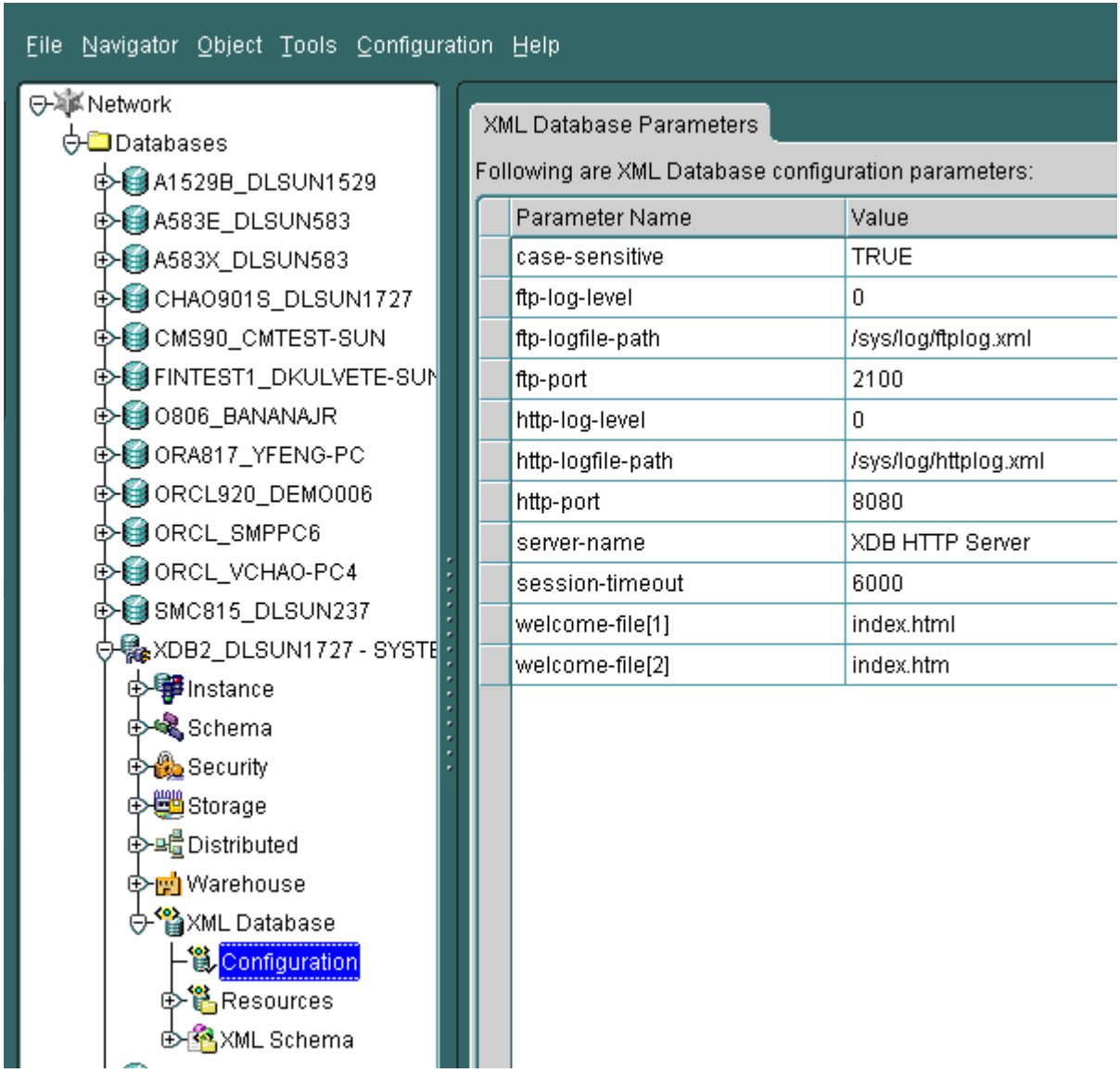
Enterprise Manager ナビゲータで「XML Database」の「**Configuration**」ノードをクリックすると、右側のメイン・パネルに「**XML Database Parameters**」ページが表示されます。「XML Database Parameters」ウィンドウには、次の情報が表示されます。

- Parameter Name: パラメータの名前が表示されます。

- Value: パラメータの現在の値が表示されます。これは編集可能なフィールドです。
- Default: 値がデフォルト値であるかどうかが表示されます。チェック・マークが付いている場合、デフォルト値であることを示します。
- Dynamic: 動的な値であるかどうかが表示されます。チェック・マークが付いている場合、動的な値であることを示します。
- Category: パラメータのカテゴリが表示されます。カテゴリは、HTTP、FTP または Generic です。

パラメータの値を変更するには、「**Value**」フィールドをクリックして変更を加えます。変更を適用するには、「**Apply**」ボタンをクリックします。リストのパラメータをクリックし、「**Description**」ボタンをクリックすると、パラメータの説明を表示できます。「**Description**」テキスト・ボックスには、選択したパラメータのより詳細な説明が表示されます。「**Description**」ボタンを再度クリックすると、「Description」ボックスを閉じることができます。

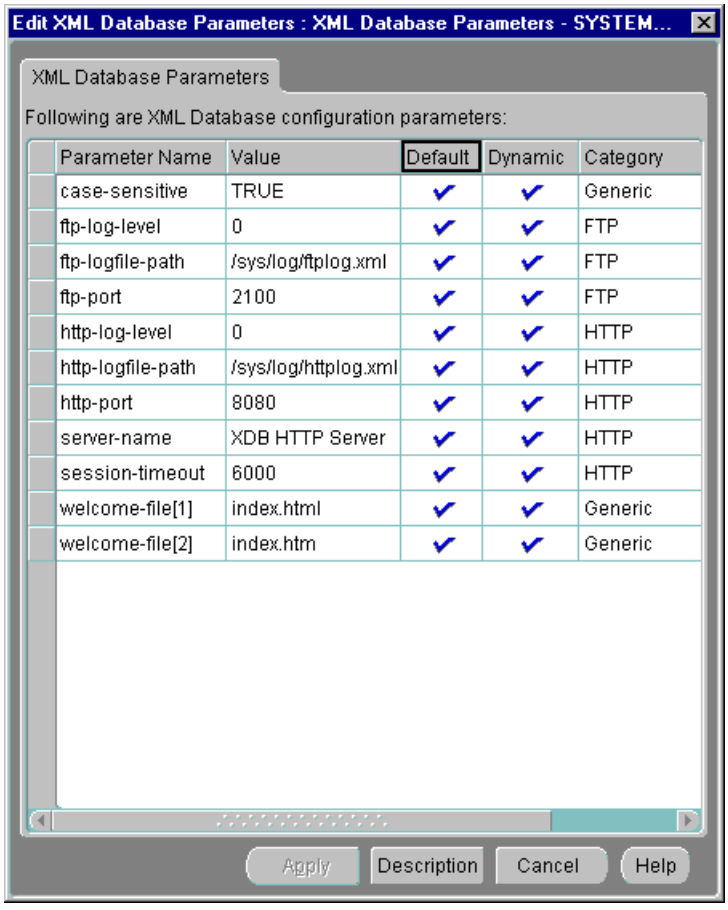
図 21-3 Enterprise Manager コンソール : Oracle XML DB の構成



Oracle XML DB ではリポジトリにアクセスする方法として標準インターネット・プロトコル（FTP、WebDAV、HTTP）をサポートしているため、Enterprise Manager によって関連する情報が提供されます。

- **Oracle XML DB FTP Port:** FTP プロトコルによってリスニングされるポート番号が表示されます。デフォルトでは、FTP は、保護されていない非標準のポートをリスニングします。
- **Oracle XML DB HTTP Port:** HTTP プロトコルによってリスニングされるポート番号が表示されます。HTTP は、共有サーバー表現で管理され、TNS リスナーを介して任意のポートをリスニングするように構成できます。HTTP は、保護されていない非標準のポートをリスニングします。

図 21-4 Enterprise Manager コンソール：「Edit XML Database Parameters」ダイアログ・ボックス



Oracle XML DB 構成パラメータの表示または編集

Enterprise Manager を使用すると、次のカテゴリにある Oracle XML DB の一部の構成パラメータを表示および編集できます。

カテゴリ : Generic

- case-sensitive

> **注意 :** Oracle XML DB では、常に大 / 小文字が区別されます。

カテゴリ : FTP

- ftp-port: Enterprise Manager によって、FTP が共有サーバー表現で管理されます。TNS リスナーを使用して、任意のポートをリスニングするように構成できます。
- ftp-logfile-path: FTP サーバーのログ・ファイルへのファイル・パス。
- ftp-log-level: FTP のエラーおよび警告条件に対するロギングのレベル。

カテゴリ : HTTP

- http-port: Enterprise Manager によって、HTTP が共有サーバー表現で管理されます。TNS リスナーを使用して、任意のポートをリスニングするように構成できます。
- session-timeout: サーバーが接続を切断するまでにクライアントの応答を待機する最大時間。
- server-name: HTTP リダイレクトおよびサーブレット API にデフォルトで使用されるホスト名。
- http-logfile-path: HTTP サーバーのログ・ファイルへのファイル・パス。
- http-log-level: HTTP のエラーおよび警告条件に対するロギングのレベル。
- welcome-file-list: サーバーによって使用されるウェルカム・ファイルのリスト。

Enterprise Manager を使用した Oracle XML DB のリソースの作成および管理

Enterprise Manager のナビゲーション・ツリーでは、「**Resources**」フォルダは、「XML Database」フォルダの下に存在します。このフォルダには、所有者に関係なくデータベースに存在するすべてのリソースが含まれます。図 21-5 に、一般的なリソース・ツリーを示します。

ナビゲータで「Resources」フォルダを選択すると、画面の右側にルートの下にあるすべての最上位の Oracle XML DB のリソース、これらのリソースの名前、作成日および変更日が表示されます。図 21-6 を参照してください。

図 21-5 Enterprise Manager: Oracle XML DB のリソース・ツリーで選択された「Resources」フォルダ

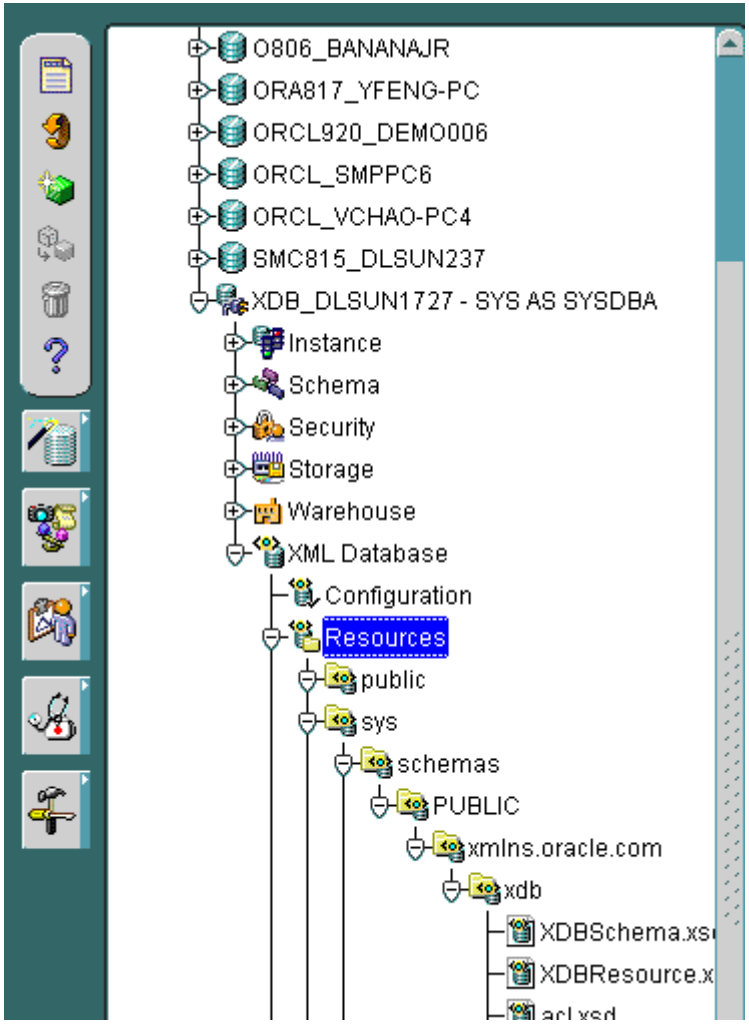
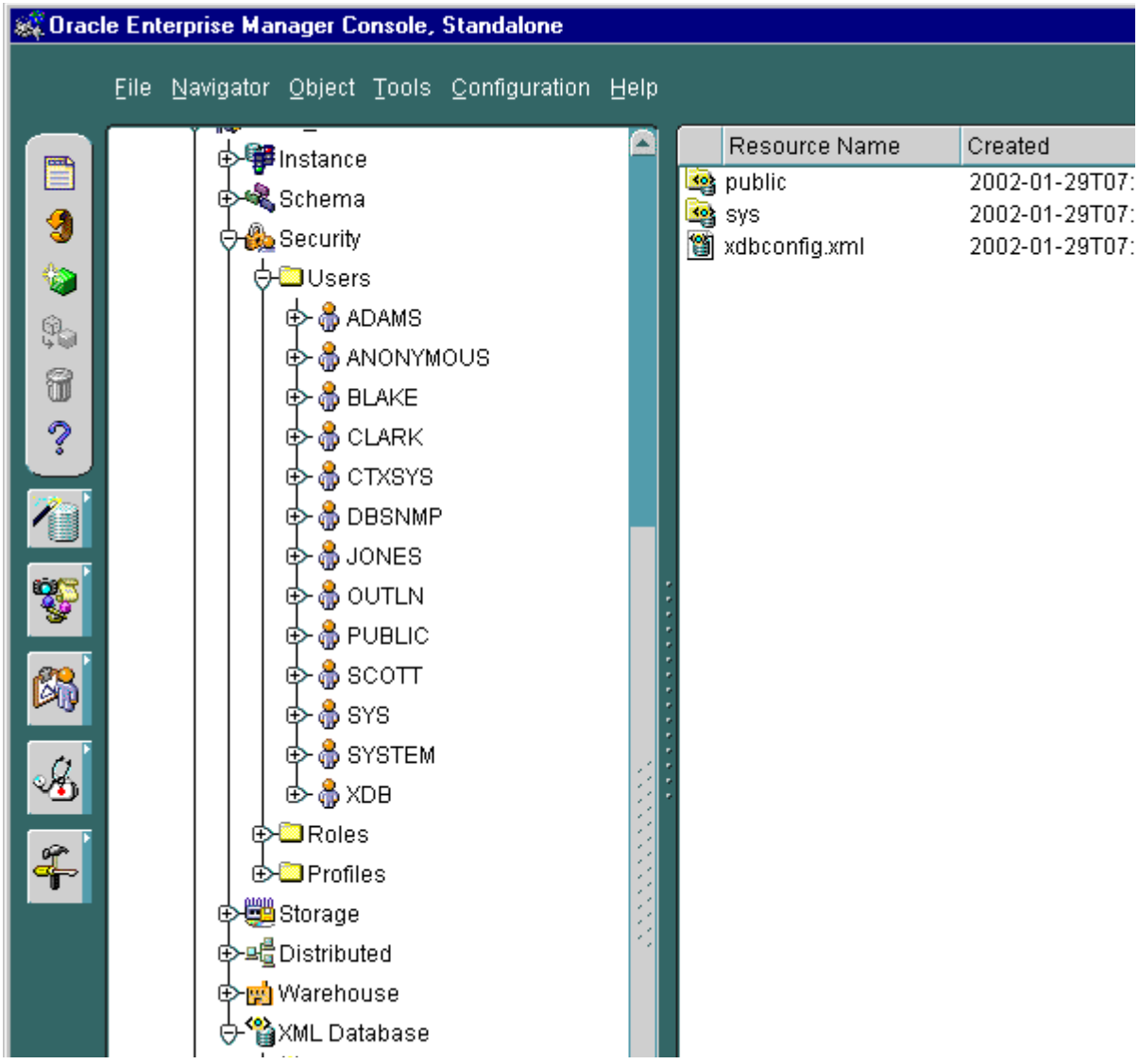


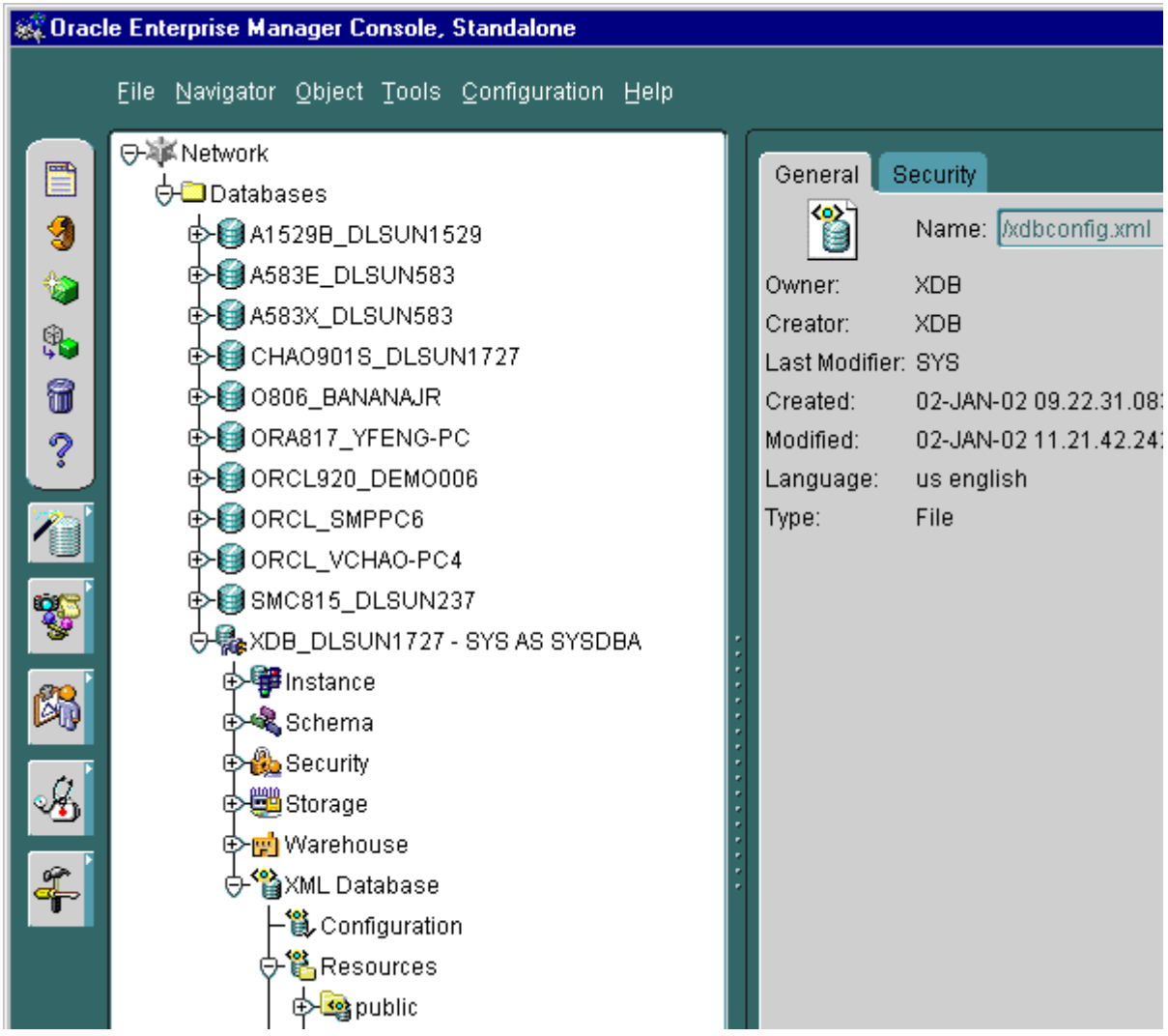
図 21-6 Enterprise Manager: ルートの下にある最上位のリソース



個々のリソースの管理

ナビゲータで「Resources」フォルダの下にあるリソース・サブフォルダを選択すると、図 21-7 に示すとおり、詳細ビューにリソースの「General」ページが表示されます。

図 21-7 Enterprise Manager: 個々の Oracle XML DB リソース - 「General」ページ



「General」ページ

Oracle XML DB の「Resources」フォルダの「General」ページ（XML リソース・ページともいう）には、リソース・コンテナまたはリソース・ファイルについての情報の概要が表示されます。ナビゲータで Oracle XML DB のリソース・コンテナまたはリソース・ファイルの 1 つを選択すると、Enterprise Manager によって Oracle XML DB の「Resources」フォルダの「General」ページが表示されます。これは、読込み専用ページです。このページには、次の情報が表示されます。

- Name: リソース・ファイルまたはリソース・コンテナの名前
- Owner: リソースを所有するユーザー
- Creator: リソースを作成したユーザー
- Last Modifier: リソースを最後に変更したユーザーの名前
- Created: リソースが作成された日時
- Modified: リソースが最後に変更された日時
- Language: リソースの言語（US English など）
- Type: File または Container

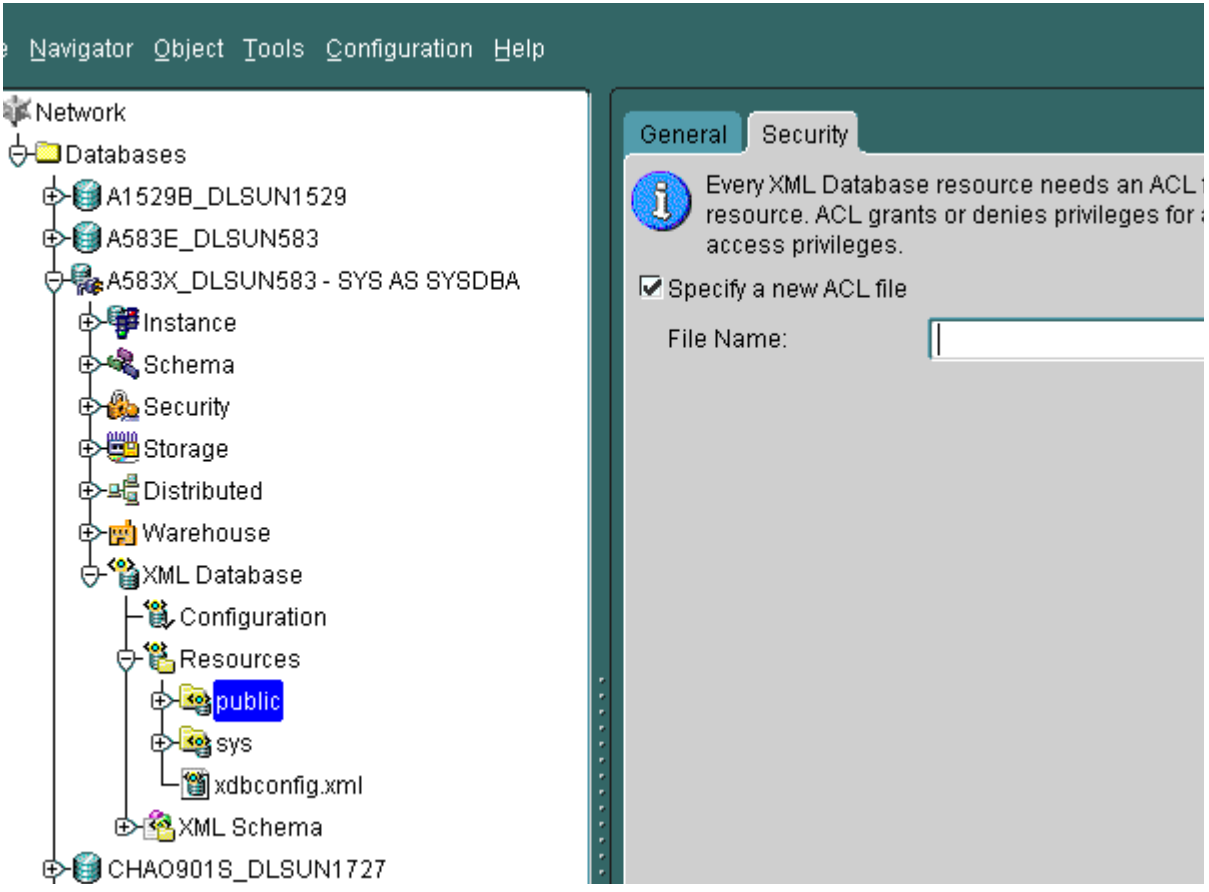
「Security」ページ

Oracle XML DB の「Resources」フォルダの「Security」ページでは、Oracle XML DB のリソース・コンテナまたはリソース・ファイルに関連付けられた ACL を変更できます。ACL ファイルを使用すると、すべての Oracle XML DB リソースへのアクセスを制限できます。リソースに対する ACL ファイルを変更する場合は、リソース・ファイルに対するアクセス権限を変更します。図 21-8 を参照してください。

新しい ACL ファイルを指定するには、次の手順を実行します。

1. 「**Specify a new ACL file**」オプション・ボックスをクリックし、「File Name」フィールドでドロップダウン・リストから新しい ACL を選択します。
2. 変更を適用するには、「**Apply**」ボタンをクリックします。
3. 「File Name」フィールドに対する変更を中止するには、「**Revert**」ボタンをクリックします。

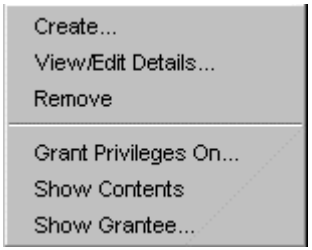
図 21-8 Enterprise Manager: 個々の Oracle XML DB リソース - 「Security」ページ



個々のリソースのコンテンツ・メニュー

図 21-9 に、ナビゲータで個々の Oracle XML DB のリソース・オブジェクトを選択して右クリックすると表示される、コンテキスト・センシティブ・メニューを示します。

図 21-9 Enterprise Manager: 個々のリソースを右クリックした場合



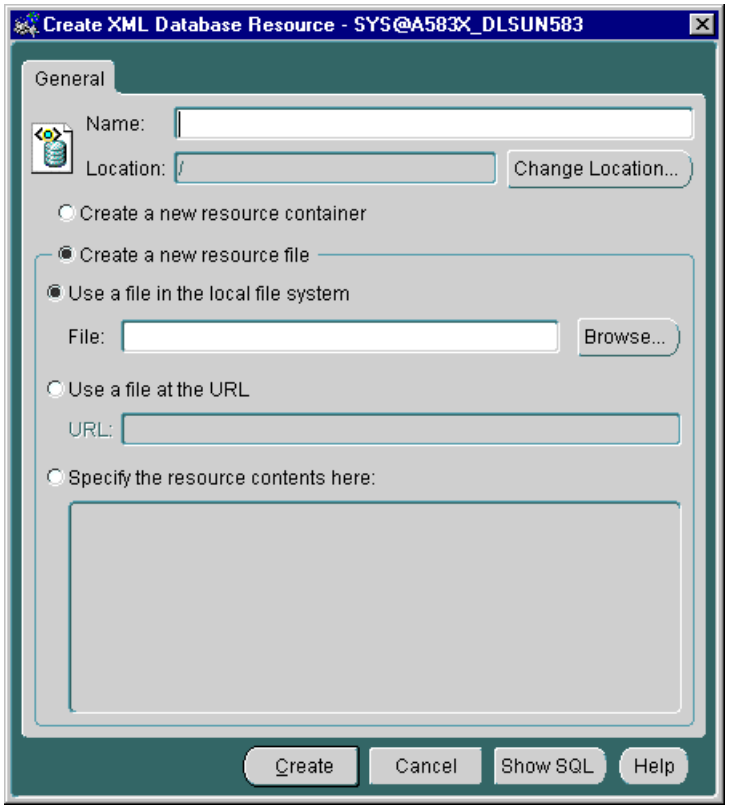
Enterprise Manager のコンテンツ・メニューから、次の Oracle XML DB タスクを実行できます。

リソースの作成

図 21-10 に、「**Create XML Database Resource**」ダイアログ・ボックスを使用して、XML DB のリソース・コンテナまたはリソース・ファイルを作成する方法を示します。「Create XML Database Resource」ダイアログ・ボックスから、リソースに名前を付け、次にファイルの場所をローカル・ファイルまたは指定された URL 上のファイルのいずれかに指定して新しいリソース・コンテナを作成するか、またはファイルのコンテンツを指定して新しいリソース・ファイルを作成できます。

1. 「Resources」フォルダまたは個々のリソース・ノードを右クリックし、コンテキスト・メニューから「**Create**」を選択して、「Create XML Database Resource」ダイアログ・ボックスにアクセスします。「Name」フィールドでリソースを指定する場合は、「Location」フィールドの右側にある「**Change Location**」ボタンをクリックすると場所を変更できます。
2. 作成するリソースが、コンテナであるか、またはファイルであるかを指定します。「**Create a new resource file**」を選択してファイルを作成する場合は、次のいずれかのオプションを選択できます。
   - ローカル・ファイル：「**Use a file in the local file system**」を選択して、ネットワーク上のファイルの場所を参照します。
   - URL 上のファイル：「**Use a file at the URL**」を選択して、インターネットまたはイントラネット上のファイルの場所を入力します。
   - ファイルのコンテンツ：「**Specify the resource contents here**」を選択して、「Create XML Database Resource」ダイアログ・ボックスの下の方にある編集ボックスにファイルのコンテンツを入力します。

図 21-10 Enterprise Manager: 「Create XML Database Resource」ダイアログ・ボックス



権限の付与

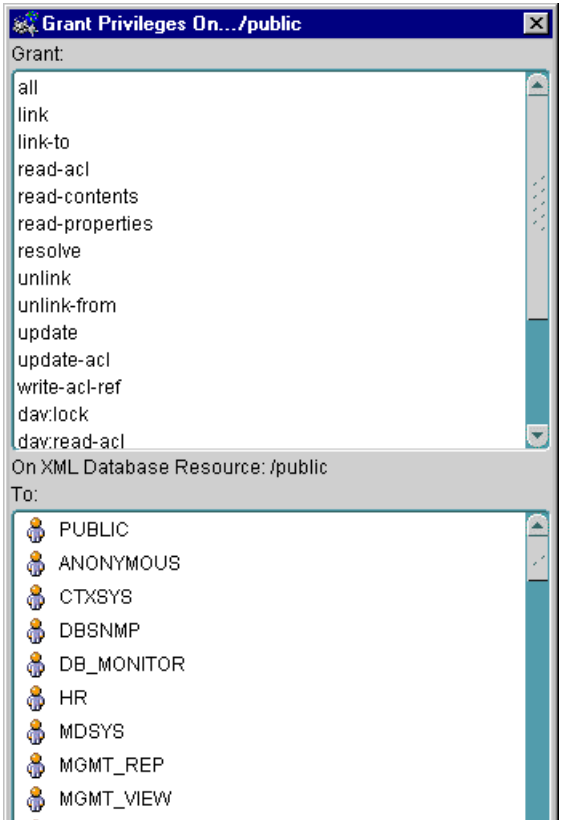
図 21-11 に、ユーザーまたはロールに Oracle XML DB のリソースに対する権限を割り当てる「**Grant Privileges On**」ダイアログ・ボックスを示します。複数のユーザーまたはロールに複数の権限を付与できます。このダイアログ・ボックスの上の方にある「Grant:」セクションには、割当て可能な Oracle XML DB のリソース権限が表示されます。

1. 権限を付与するには、付与する権限をクリックして選択します。[Ctrl] キーを押したまま複数の権限をクリックすると、複数の権限を選択できます。選択する最初の権限をクリックし、[Shift] キーを押したまま選択する最後の権限をクリックすると、その間のすべての権限を選択できます。

2. ダイアログ・ページの下にある「To:」ボックスでユーザーまたはグループを選択します。権限を付与するユーザーまたはロールを複数選択するには、前述と同じ操作を実行します。

   参照： 21-22 ページの「Enterprise Manager および Oracle XML DB: ACL セキュリティ」を参照してください。

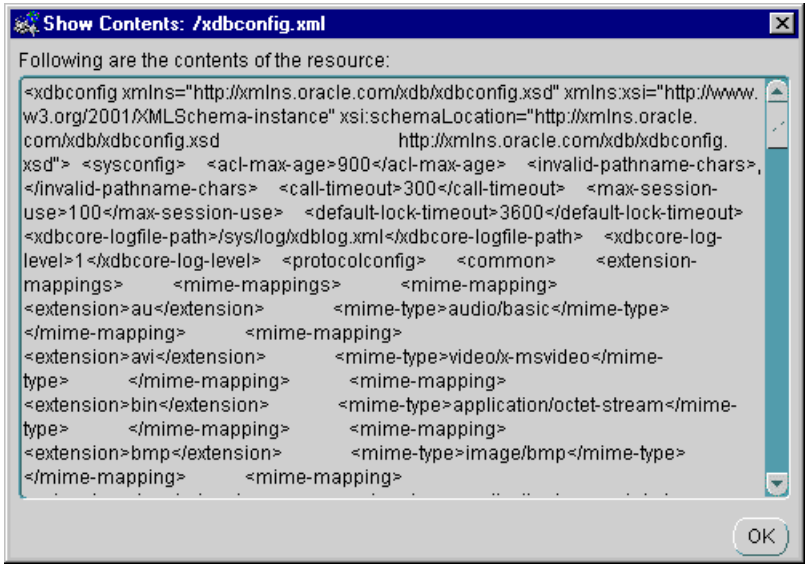
図 21-11 Enterprise Manager: Oracle XML DB のリソースに対する権限の付与



コンテンツの表示

図 21-12 に、「**Show Contents**」ダイアログ・ボックスの例を示します。このダイアログ・ボックスには、選択されたリソース・ファイルのコンテンツが表示されます。

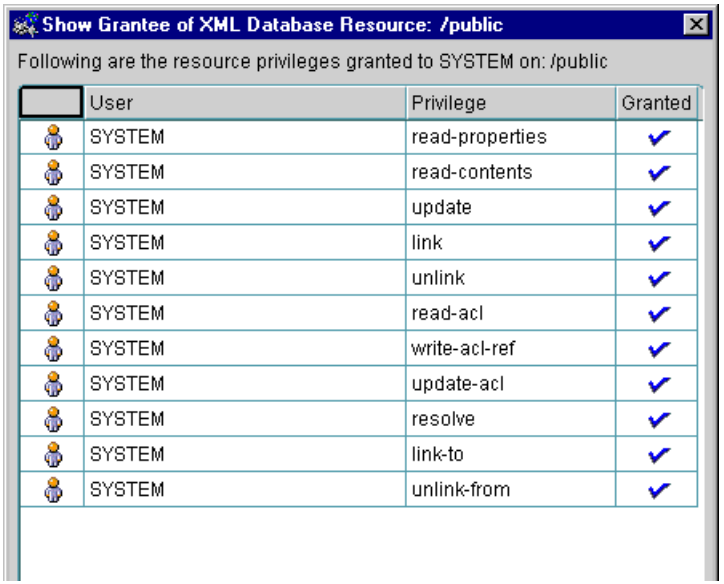
図 21-12 Enterprise Manager: 個々の Oracle XML DB リソースの「Show Contents」メニュー



権限受領者の表示

図 21-13 に、「**Show Grantee of XML Database Resource**」ダイアログ・ボックスを示します。このダイアログ・ボックスには、指定された XML DB のリソースに接続している Enterprise Manager ユーザーに付与されているすべての権限が、Enterprise Manager ユーザー、権限および権限の付与状態が表形式で表示されます。

図 21-13　Enterprise Manager: Oracle XML DB リソースに対する権限受領者の表示



Enterprise Manager および Oracle XML DB: ACL セキュリティ

Enterprise Manager から ACL を使用して、すべての XML DB のリソースへのアクセスを制限できます。既存の「Security」>「Users」> ユーザーおよび「Security」>「Roles」> ロール・インタフェースを使用して、データベース・ユーザーおよびデータベース・ロールに対して個別に XML DB のリソース権限を付与できます。

Enterprise Manager のセキュリティ・オプションにアクセスするには、主に次の 2 つの方法があります。

- **ユーザー（またはロール）のセキュリティを表示または変更する方法 :** ナビゲータで、対象の Oracle XML DB データベース >「Security」>「Users」> ユーザー（または対象の Oracle XML DB データベース >「Security」>「Roles」> ロール）を選択します。詳細ビューで、「XML」タグを選択します。図 21-14 を参照してください。このユーザー・セキュリティ・オプションの詳細は、21-23 ページの「ユーザー >「XML」タブを使用したユーザー権限の付与および取消し」を参照してください。
- **リソースのセキュリティを表示または変更する方法 :** 左側のナビゲーション・パネルで「Resources」フォルダの下にある個々のリソース・ノードを選択します。詳細ビューで「Security」タグを選択します。図 21-15 を参照してください。

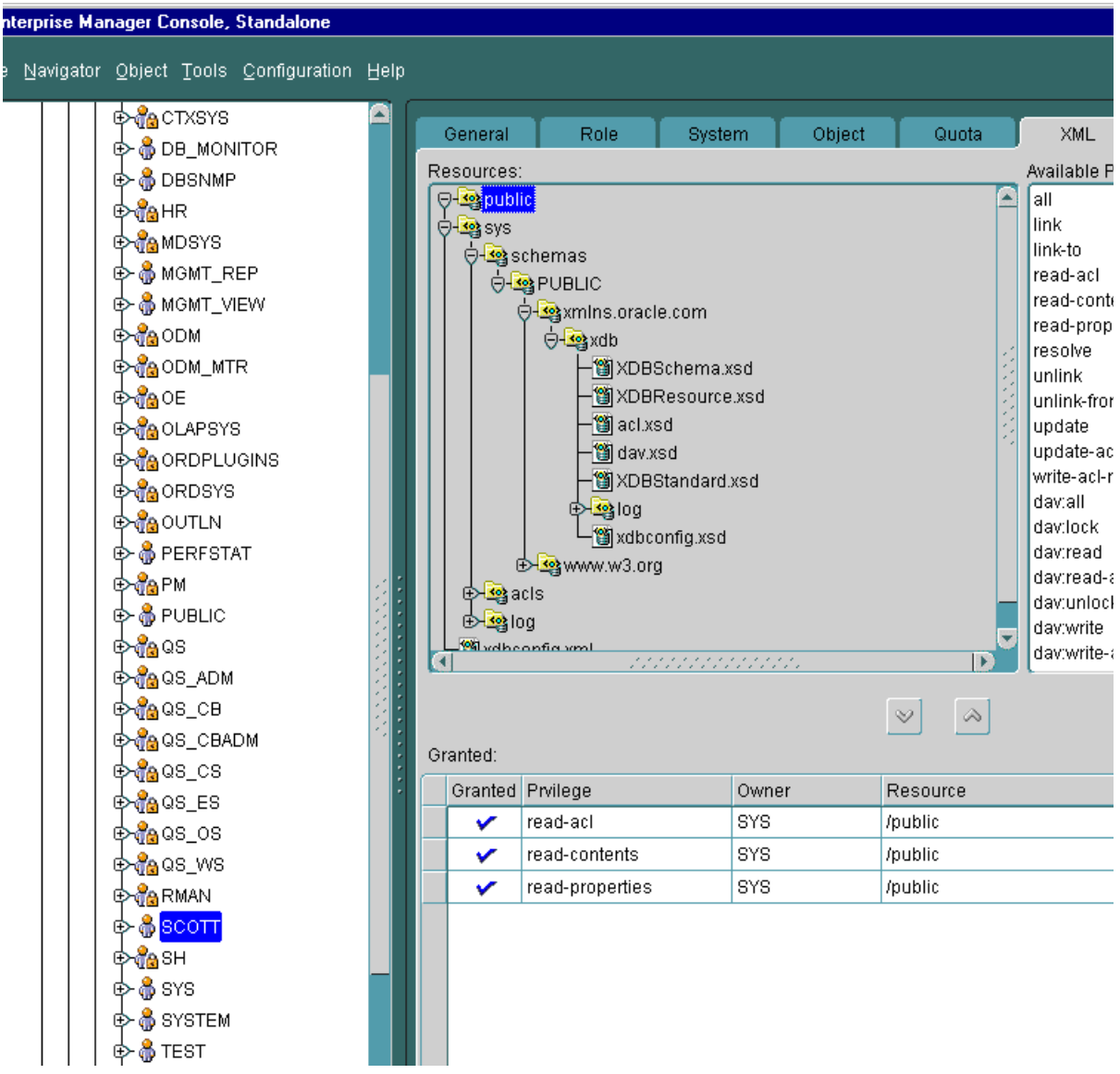
ユーザー >「XML」タブを使用したユーザー権限の付与および取消し

この項では、ユーザーに対する権限を付与および取り消す方法を説明します。ロールに対する権限の付与および取消しにも、同じ手順が適用されます。ユーザーに権限を付与するには、次の手順を実行します。

1. Enterprise Manager のナビゲータから特定のユーザーを選択します。詳細ビューで、既存のプロパティ・シートに「**XML**」タブが追加されます。
2. ユーザーまたはロールに権限を付与するリソースを表示および選択するには、「**XML**」タブを選択します。リソースを選択すると、「Resources」リストの右側にある「**Available Privileges**」リストに、そのリソースについて割当て可能なすべての権限が表示されます。
3. 「**Available Privileges**」リストから必要な権限を選択し、下矢印をクリックして、ウィンドウの下の方に表示される「**Granted**」リストに選択した権限を移動します。

   「**Granted**」リストで権限を選択し、上矢印をクリックすると、その権限を取り消すことができます。
4. 適切な権限を設定した後、「**Apply**」ボタンをクリックして変更を保存します。変更を保存する前に「**Revert**」ボタンをクリックすると、変更を中止できます。

図 21-14「Users」> ユーザー >「XML」を使用した権限の付与または取消し



「Resources」リスト

「**Resources**」リストには、Oracle XML DB Repository に存在するリソースがツリー表示されます。フォルダ階層をナビゲートし、ナビゲータで選択したユーザーまたはロールに権限を設定するリソースを検索して選択できます。ツリー・リストでリソースを選択すると、そのリソースの権限が右側の「**Available Privileges**」に表示されます。

「Available Privileges」リスト

「**Available Privileges**」リストには、リソースについて割当て可能なすべての権限が表示されます。権限をクリックし、下矢印ボタンをクリックして「**Granted**」リストに権限を追加します。リスト内で、最初の権限をクリックし、次に [Shift] キーを押したまま最後の権限をクリックすると、その間のすべての権限を選択できます。また、[Ctrl] キーを押したままクリックすると、クリックしたすべての権限のみを選択できます。

権限は、次のいずれかです。

- 集約権限 : 他の権限が含まれます。
- 基本権限 : 分割できません。

「Granted」リスト

「**Granted**」リストには、「**Resources**」リストで選択されたリソースの、ユーザーまたはロールに付与されたすべての権限が表示されます。権限を選択し、上矢印をクリックして削除すると、その権限を取り消すことができます。

XML データベース・リソース権限

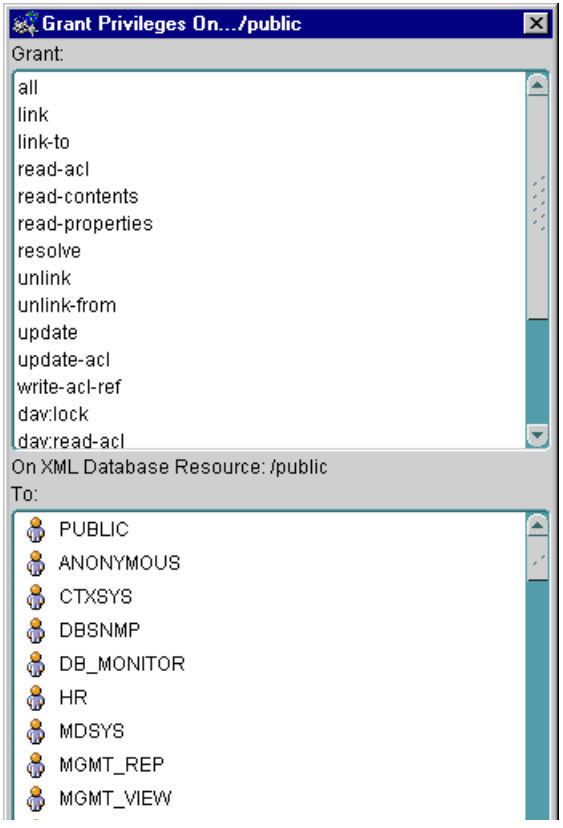
権限は、集約（他の権限が含まれる）または基本（分割できない）のいずれかです。次のシステム権限がサポートされます。

- **基本権限**
  - all
  - read-properties
  - read-contents
  - update
  - link（コンテナのみに適用）
  - unlink（コンテナのみに適用）
  - read-acl
  - write-acl-ref
  - update-acl

  - link-to
  - unlink-from
  - resolve
  - dav:lock
  - dav:unlock

- **集約権限**
  - dav:read（read-properties、read-contents、resolve）
  - dav:write（update、link、unlink、unlink-from）
  - dav:read-acl（read-acl）
  - dav:write-acl（write-acl-ref、update-acl）
  - dav:all（dav:read、dav:write、dav:read-acl、dav:write-acl、dav:look、dav:unlock）

図 21-15 リソースの権限付与



参照： サポートされるシステム権限のリストは、第 18 章「Oracle XML DB リソースのセキュリティ」を参照してください。

XML Schema および関連するデータベース・オブジェクトの管理

「XML Database Management」詳細ビューから「Create tables and views based on XML Schema」を選択すると、「XML Schema Based Objects」ページが表示されます。このページでは、次の作業を実行できます。

- XML Schema の登録
- XML Schema に基づく構造化記憶域の作成
- XMLType 表の作成
- XMLType 列を含む表の作成
- XML Schema に基づくビューの作成
- XPath 式に基づくファンクション索引の作成

Enterprise Manager での XML Schema のナビゲート

「**XML Schema**」フォルダの下には、すべての XML Schema の所有者がツリー表示されます。ここでは、所有者「XDB」の例を示します。図 21-16 を参照してください。

- **スキーマの所有者 :** 個々の XML Schema 所有者の下には、その所有者が所有する XML Schema がツリー表示されます。「XDB」の下には、次の XML Schema が表示されています。

  ```
  http://xmlns.oracle.com/xdb/XDBResource.xsd
  http://xmlns.oracle.com/xdb/XDBSchema.xsd
  http://xmlns.oracle.com/xdb/XDBStandard.xsd
  ```

- **最上位の要素 :** 各 XML Schema の下には、最上位の要素がツリー表示されます。これらの要素を使用して、XMLType 表、XMLType 列を含む表およびビューを作成します。たとえば、図 21-16 には、最上位の要素である「**servlet**」および「**LINK**」が表示されています。これらの要素の数および名前は、関連する XML Schema 定義によって示されます。この場合の XML Schema 定義は `http://xmlns.oracle.com/XDBStandard.xsd` です。
- **依存オブジェクト :** 各要素の下には、作成された依存オブジェクトである「Tables」、「Views」および「User Types」がツリー表示されます。この例では、最上位の要素である「**servlet**」に、依存オブジェクトである XMLType の「**Tables**」、「**Views**」および「**User types**」が存在することを確認できます。

- **依存オブジェクトの所有者** : 各依存オブジェクトの下には、所有者がツリー表示されます。
  - **Tables**: たとえば、「**Tables**」の下には「XDB」という所有者が表示されており、「XDB」は「**SERVLET**」という表を所有しています。
    - 表の特性 : 各表の名前の下には、作成されたすべての「**Indexes**」、「**Materialized View Logs (Snapshots)**」、「**Partitions**」および「**Triggers**」がツリー表示されます。
  - **Views**: この例では表示されていませんが、「Views」の下には、すべてのビューの所有者およびこれらの所有者によって所有されるビューの名前が表示されます。
    - ビューの特性 : この例では表示されていません。
  - **User Types:** 最上位の要素である「servlet」に関連付けられたユーザー・タイプがツリー表示されます。これらは次の型ごとに表示されます。
    - オブジェクト型 : 「Object Types」の下には、オブジェクト型の所有者がツリー表示されます。
    - 配列型 : 「Array Types」の下には、配列型の所有者がツリー表示されます。
    - 表型 : 「Table Types」の下には、表型の所有者がツリー表示されます。

図 21-16 Enterprise Manager コンソール : XML Schema のナビゲート

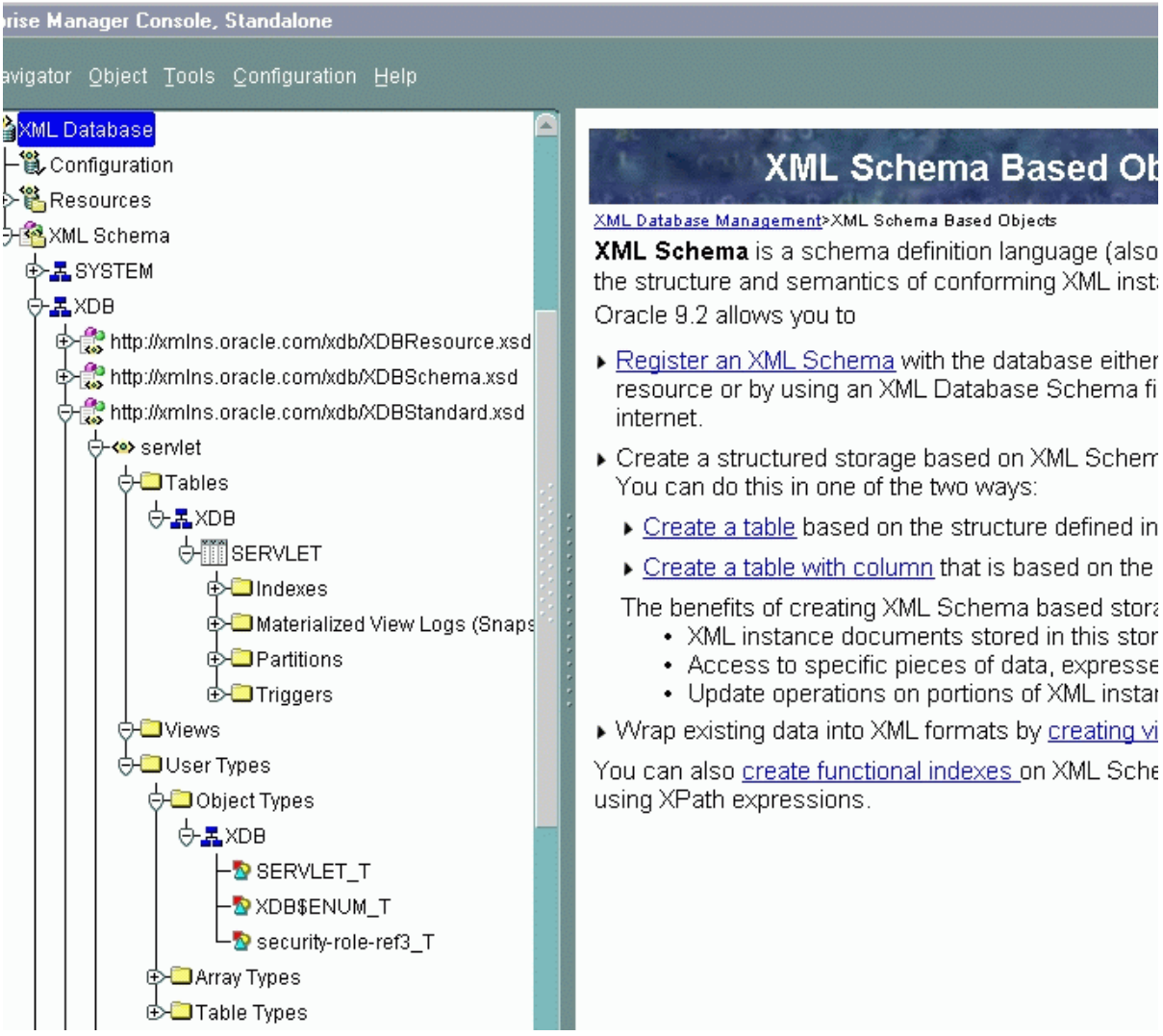
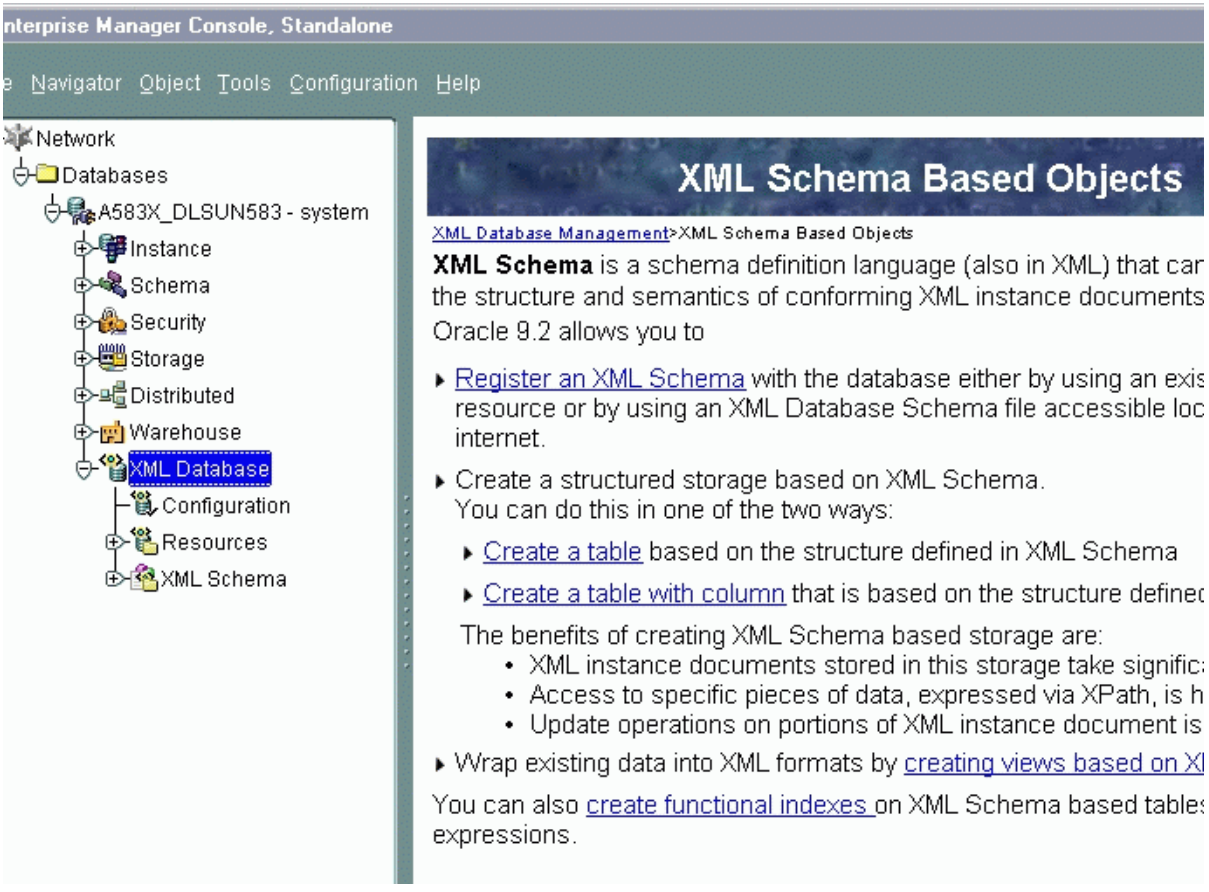


図 21-17 Enterprise Manager: XML Schema に基づくオブジェクトの作成



XML Schema の登録

XML Schema の登録は、Oracle XML DB を使用するための主要なタスクで、多くの場合、最初に実行します。XML Schema は、`DBMS_XMLSCHEMA.registerSchema()` を使用して登録します。

> **参照：** 第 5 章「XMLType の構造化されたマッピング」を参照してください。

図 21-18 に、XML Schema を登録する方法を示します。

「General」ページ

「General」ページでは、XML Schema URL を入力し、ドロップダウン・リストから XML Schema の所有者を選択します。

次のいずれかを選択します。

- **グローバル**：「XML Schema is visible to public」
- **ローカル**：「XML Schema is visible only to the user creating it」

次のいずれか方法を使用して、XML Schema を取得できます。

- ローカル・ファイル・システム内でのファイルの場所の指定
- XML Schema が存在する XML DB（リポジトリ）リソースの指定
- URL を使用した場所の指定
- 他の画面からのテキストの切り取りおよび貼り付け

「Options」ページ

「Options」ページでは、次のオプションを選択できます。

- 「Generate object types based on this XML Schema」
- 「Generate tables based on this XML Schema」
- 「Generate Java beans based on this XML Schema」
- 「Register this XML Schema regardless of any errors」

図 21-19 を参照してください。この XML Schema を登録するには、「General」タブまたは「Options」タブから「Create」をクリックします。

図 21-18 Enterprise Manager: XML Schema の登録 - 「General」ページ

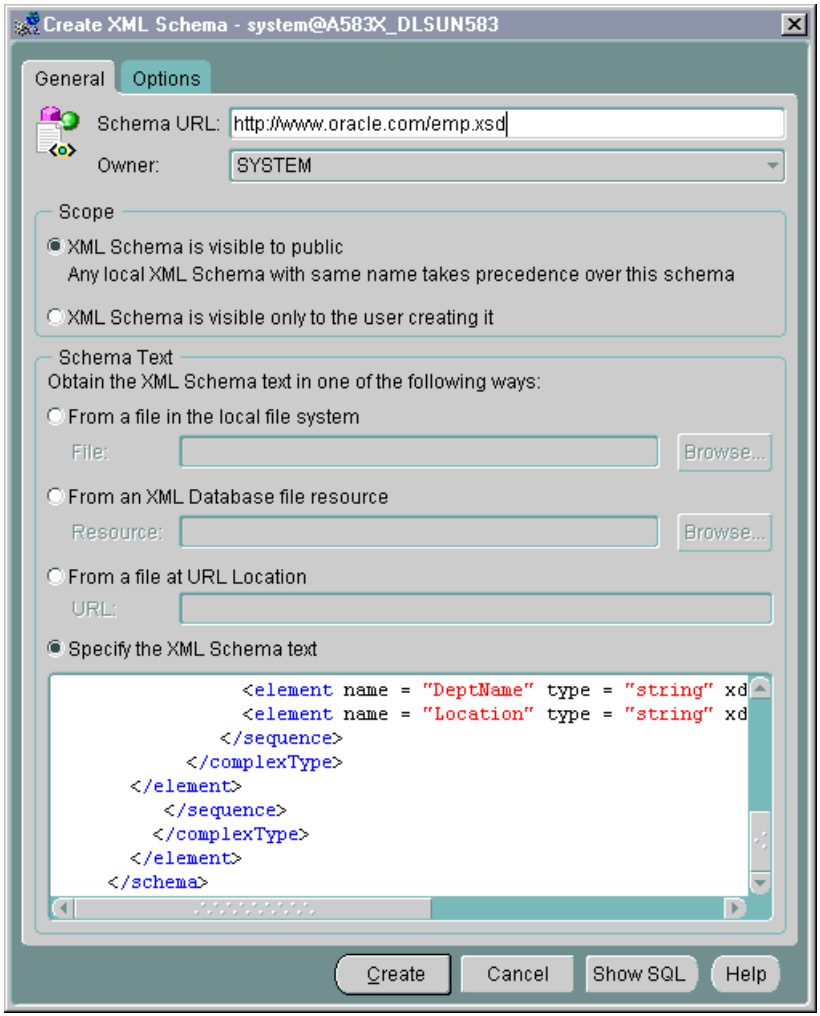
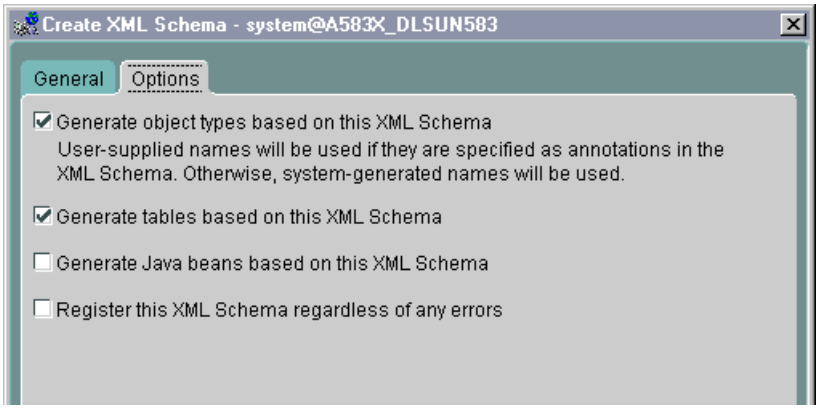


図 21-19 Enterprise Manager: XML Schema の作成 - 「Options」ページ



XML Schema に基づく構造化記憶域インフラストラクチャの作成

この項では、Oracle Enterprise Manager を使用して XMLType 表、ビューおよび索引を作成する方法について説明します。

表の作成

表を作成する場合、主に次の 2 つのオプションがあります。

- 「XMLType 表の作成」(21-35 ページ)
- 「XMLType 列を含む表の作成」(21-37 ページ)

ビューの作成

XMLType ビューを作成するには、21-39 ページの「XML Schema に基づくビューの作成」を参照してください。

ファンクション索引の作成

ファンクション索引を作成するには、21-43 ページの「XPath 式に基づくファンクション索引の作成」を参照してください。

XMLType 表の作成

「Create Table」プロパティ・シートの「General」ページで、作成する表の任意の名前を入力します。「Schema」ドロップダウン・リストから表の所有者を選択します。「Tablespace」では、デフォルトの設定を使用します。「XMLType」表を選択します。

下側の「Schema」オプションで、ドロップダウン・リストから XML Schema の所有者を選択します。

「XML Schema」で、ドロップダウン・リストから実際の XML Schema を選択します。

「Element」で、ドロップダウン・リストから XMLType 表を構成するために必要な要素を選択します。

次のいずれかの格納方法を指定します。

- **Store as defined by XML Schema**: このオプションを選択すると、XML Schema 定義の注釈に基づいて非表示列が作成されます。これらの列に対する選択操作または更新操作は最適化されます。
- **Store as CLOB**: CLOB を選択すると、「LOB Storage」タブ・ダイアログ・ボックスが表示されます。このダイアログ・ボックスでは、CLOB 記憶域をカスタマイズできます。図 21-21 を参照してください。

図 21-20 Enterprise Manager: XMLType 表の作成 - 「General」ページ

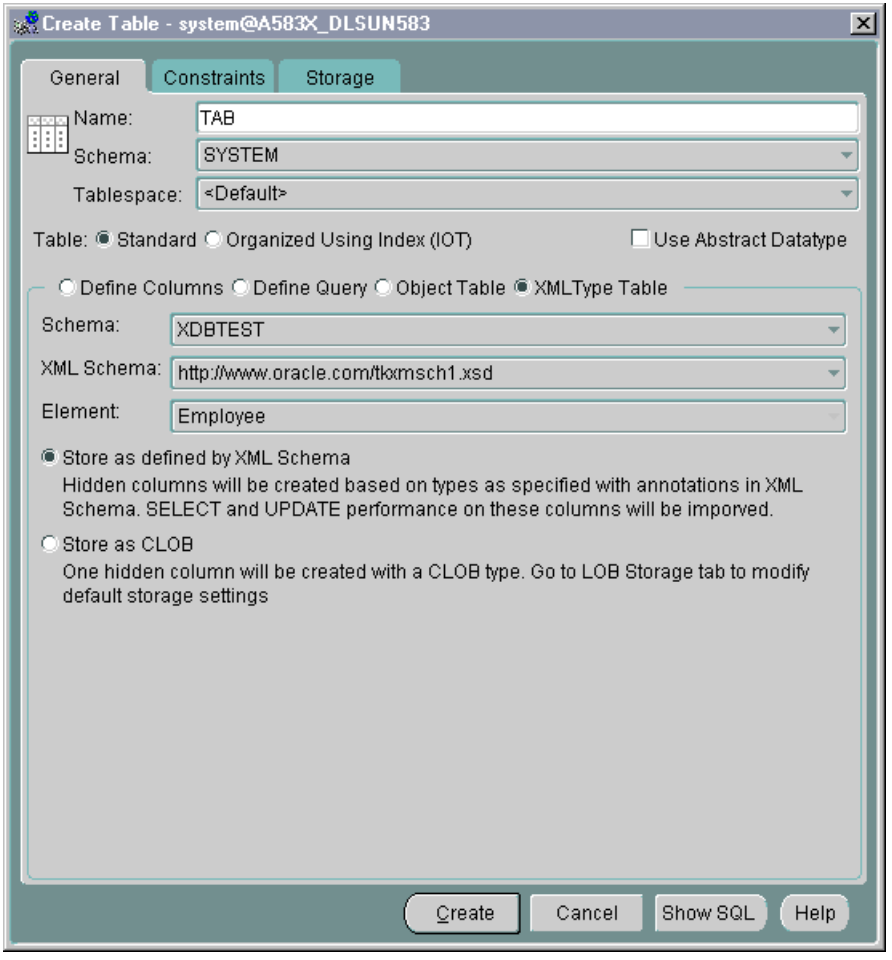
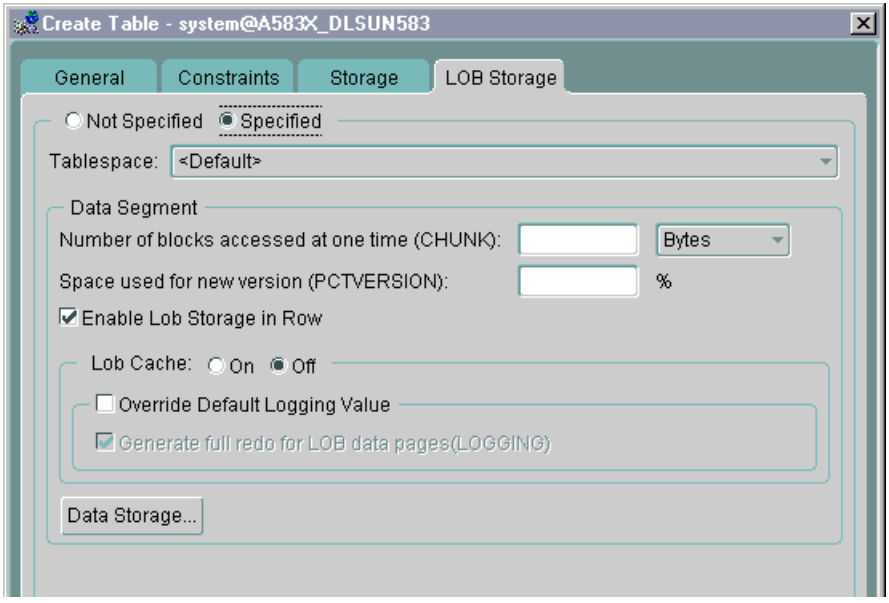


図 21-21 Enterprise Manager: XMLType 表の作成 - LOB 記憶域の指定

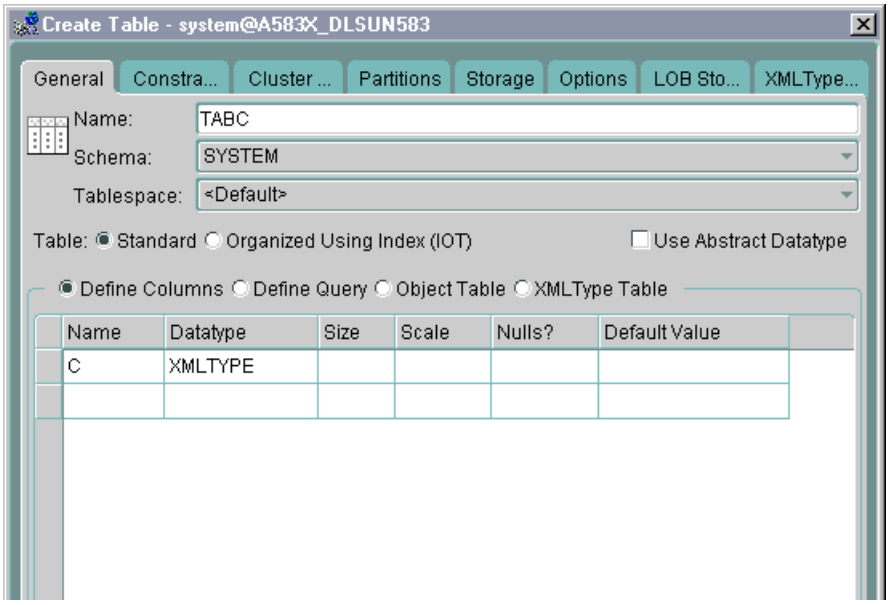


XMLType 列を含む表の作成

図 21-22 に、「Create Table」の「General」ページを示します。XMLType 列を含む表を作成するには、次の手順を実行します。

1. 「Create Table」プロパティ・シートの「General」ページで、作成する表の任意の名前を入力します。
2. 「Schema」ドロップダウン・リストから表の所有者を選択します。「Tablespace」では、デフォルトの設定を使用します。
3. 「Define Columns」を選択します。
4. 名前を入力します。データ型は、ドロップダウン・リストから「XMLTYPE」を選択します。「XMLType Options」ダイアログ・ボックスが表示されます。図 21-23 を参照してください。

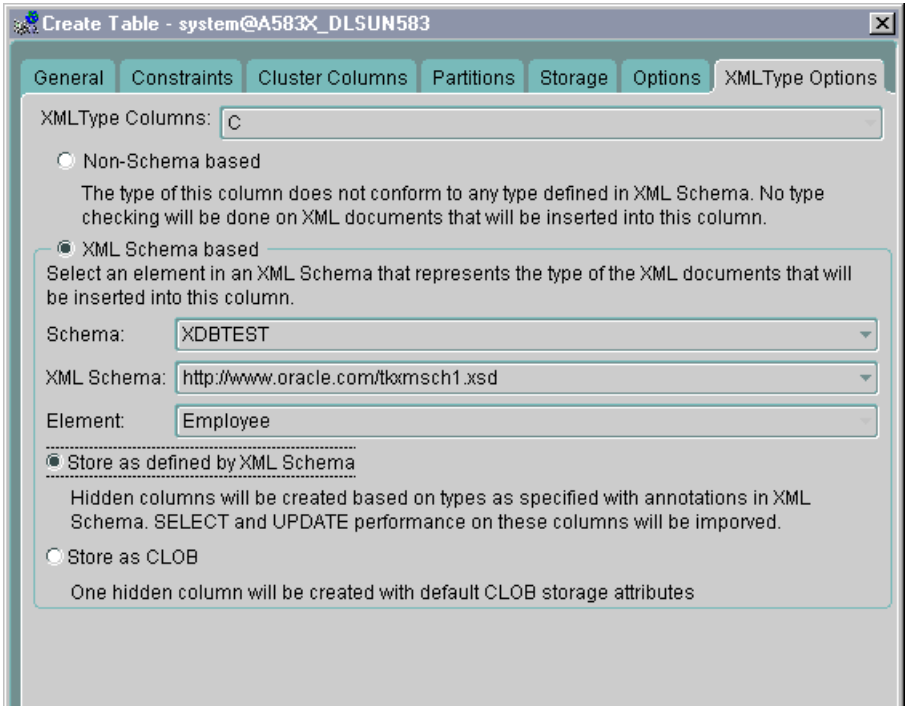
図 21-22 Enterprise Manager: XMLType 列を含む表の作成 - 「General」ページ



5. この画面で、特定の XMLType 列に対して、XML Schema に基づくか、基づかないかを指定できます。

   - XML Schema に基づく場合、次の設定を指定します。
     * 「Schema」オプションで、ドロップダウン・リストから XML Schema の所有者を選択します。
     * 「XML Schema」で、ドロップダウン・リストから実際の XML Schema を選択します。
     * 「Element」で、ドロップダウン・リストから XMLType 列を構成するために必要な要素を選択します。
     * 次のいずれかの格納方法を指定します。
     * Store as defined by XML Schema
     * Store as CLOB: CLOB を選択すると、「LOB Storage」タブ・ダイアログ・ボックスが表示されます。このダイアログ・ボックスでは、CLOB 記憶域をカスタマイズできます。
   - XML Schema に基づかない場合、デフォルトの設定を変更する必要はありません。

図 21-23 Enterprise Manager: XMLType 列を含む表の作成 - 「XMLType Options」



XML Schema に基づくビューの作成

図 21-24 に、「Create View」の「General」ページを示します。XML Schema に基づいてビューを作成するには、次の手順を実行します。

1. 任意のビューの名前を入力します。「Schema」で、ドロップダウン・リストからビューの所有者を選択します。
2. 「Query Text」ウィンドウで SQL 文を入力して、ビューを作成します。「Advanced」タブを選択します。図 21-25 を参照してください。

   このタブから、「Force」モードを選択できます。
3. 「As Object」オプションを選択します。ビューは、「Read Only」または「With Check Option」に設定できます。
4. これは XMLType ビューであるため、「As Object」オプションを選択します。「Object Type」ではなく、「XMLType」を選択します。

5. 「Schema」オプションで、ドロップダウン・リストから XML Schema の所有者を選択します。
6. 「XML Schema」で、ドロップダウン・リストから実際の XML Schema を選択します。
7. 「Element」で、ドロップダウン・リストから `XMLType` 列を構成するために必要な要素を選択します。
8. オブジェクト識別子（OID）を指定します。
   - ビューを作成する SQL 文がオブジェクト型の表に基づいている場合、「Use default if view query is based on object table」を選択します。
   - そうでない場合、「Specify based on XPath expression on the structure of the element」を選択します。XPath 式を入力します。第 11 章「XMLType ビュー」を参照してください。

図 21-24 Enterprise Manager: XMLType ビューの作成 - 「General」ページ

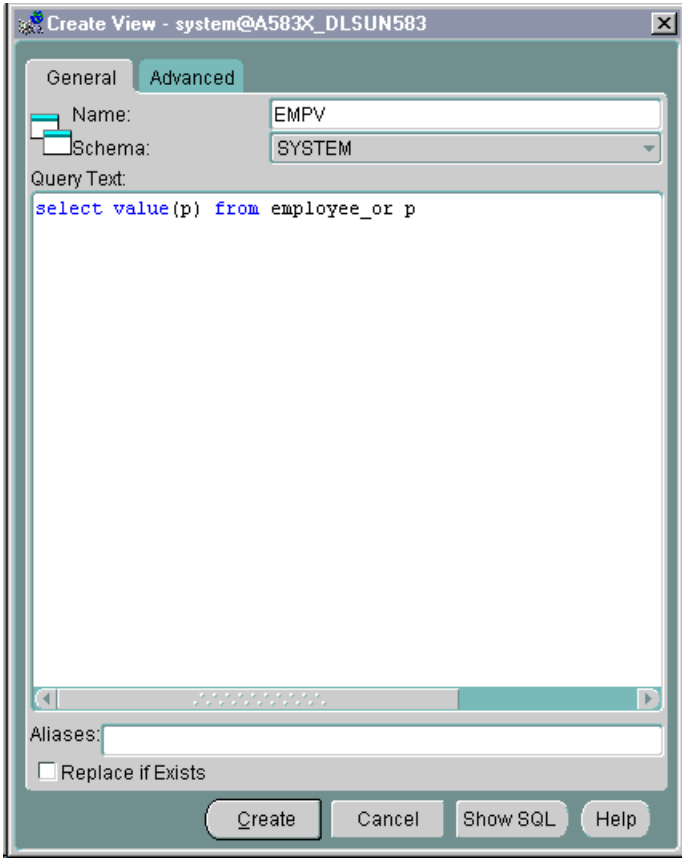
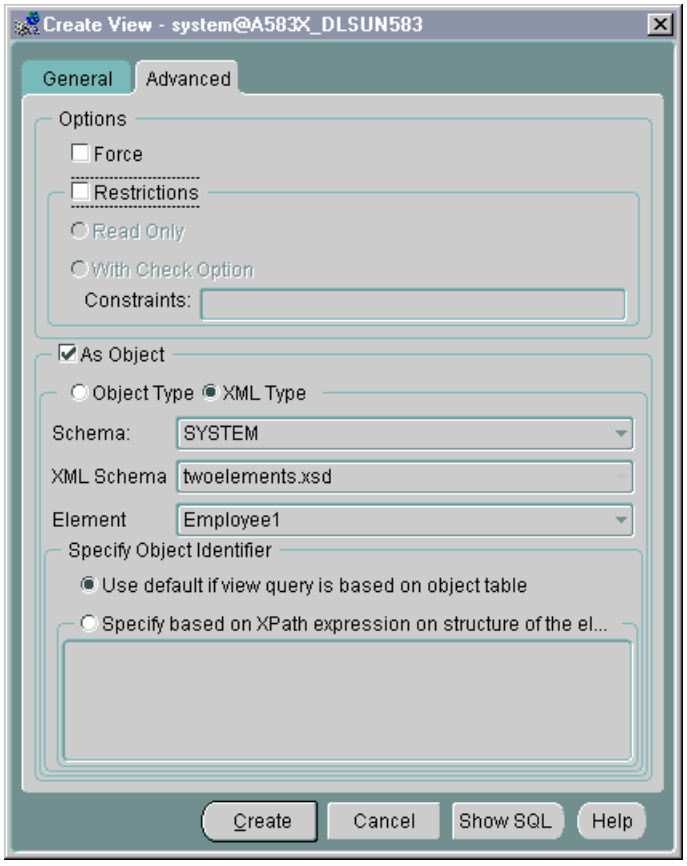


図 21-25 Enterprise Manager: XMLType ビューの作成 - 「Advanced」ページ



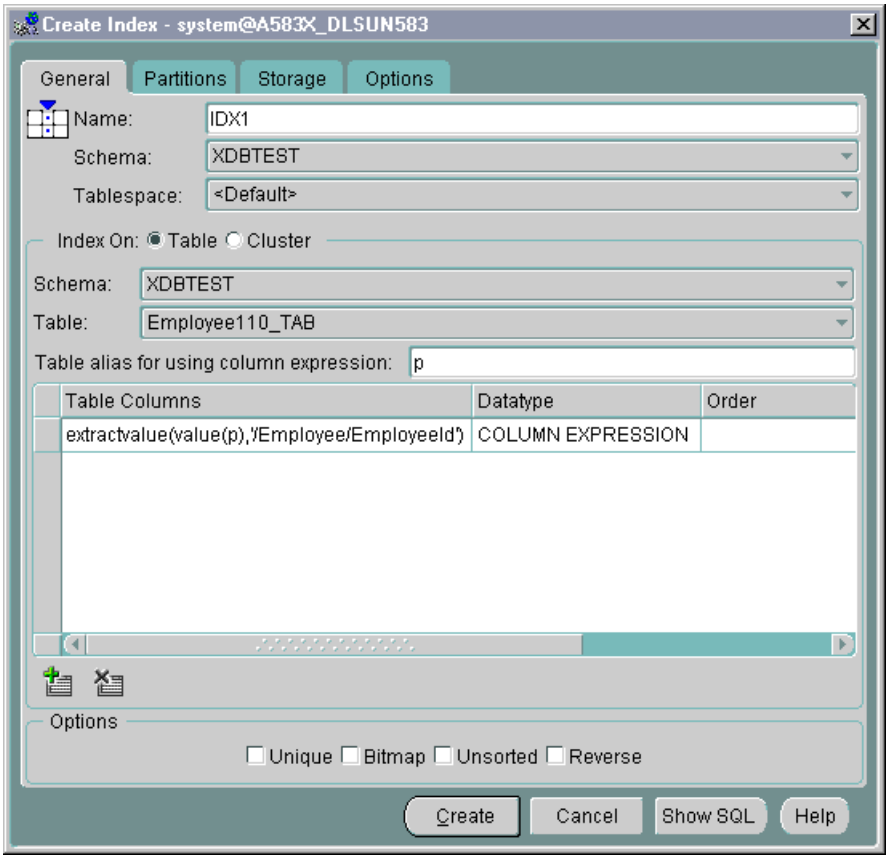
XPath 式に基づくファンクション索引の作成

図 21-26 に、「Create Index」の「General」ページを示します。XPath 式に基づいてファンクション索引を作成するには、次の手順を実行します。

1. 索引の所有者および名前を指定します。「Name」に、必要な索引の名前を入力します。「Schema」で、ドロップダウン・リストから索引の所有者を選択します。
2. 「Index On」で、「Table」を選択します。
3. 下側の「Schema」で、ドロップダウン・リストから表の所有者を選択します。
4. 「Table」で、ドロップダウン・リストから XMLType 表または XMLType 列を含む表のいずれかを選択します。

   列の式に別名を入力することもできます。この別名は、ファンクション索引文内で指定できます。
5. **XMLType 列を含む表の場合**、まず左下の「+」アイコンをクリックします。これによって、「Table Columns」に抽出 XPath 式を入力するための新しい行が作成されます。
6. **XMLType 表の場合**、「Table Columns」に抽出 XPath 式を入力するための新しい空白行が自動的に作成されます。
7. 「Datatype」は、デフォルトで、「COLUMN EXPRESSION」に設定されます。この設定を変更する必要はありません。

図 21-26 Enterprise Manager: XPath 式に基づくファンクション索引の作成



22

Oracle XML DB への XML データのロード

この章では、SQL*Loader を使用して XML データを Oracle XML DB にロードする方法を説明します。

この章の内容は次のとおりです。

Oracle9*i* データベースへの XMLType データのロード

Oracle9*i* データベース リリース 1（9.0.1）以上のエクスポート / インポート・ユーティリティおよび SQL*Loader では、列の型として XMLType がサポートされています。そのため、表に XMLType 型の列が存在する場合、Oracle9*i* データベースとの間でその列を適切にエクスポートおよびインポートできます。また、SQL*Loader を使用して XML データをその列にロードできます。

参照：『Oracle9*i* データベース・ユーティリティ』を参照してください。

リストア

今回のリリースでは、ユーザー・データのエクスポート時、Oracle XML DB Repository の情報はエクスポートされません。これは、リソースおよびすべての情報がエクスポートされないことを意味します。

SQL*Loader を使用した XMLType 列のロード

SQL*Loader は、XML 列を CLOB と同様に処理します。次の項で説明するプライマリ・データ・ファイルまたは LOBFILE からの LOB データのロード方法は、XML 列のロードに適用できます。

参照：『Oracle9*i* データベース・ユーティリティ』を参照してください。

注意： LOB フィールドには SQL 文字列を指定できません。これは、LOBFILE_spec を指定する場合も同じです。

LOB は非常に大きいデータであるため、SQL*Loader では、LOB データをプライマリ・データ・ファイル（残りのデータを持つインライン）または LOBFILE のいずれかからロードできます。この項では、次の項目について説明します。

- プライマリ・データ・ファイルからの LOB データのロード
- 外部 LOBFILE（BFILE）からの LOB データのロード
- LOBFILE からの LOB データのロード

プライマリ・データ・ファイルから内部 LOB（BLOB、CLOB および NCLOB）または XML 列をロードするには、次の標準 SQL*Loader 形式を使用できます。

- 事前にサイズが決まっているフィールド
- デリミタ付きフィールド
- Length-Value Pair フィールド

これらの各形式については、次の項および『Oracle9*i* データベース・ユーティリティ』を参照してください。

事前にサイズが決まっているフィールドの LOB データ

これは LOB をロードする場合、最も高速で、概念的に単純な形式です。

注意 : ロードする LOB データは、サイズが均等ではないため、サイズが小さいデータ・フィールドに空白を埋め込み、全 LOB が同じサイズになるようにできます。

デリミタ付きフィールドの LOB データ

この形式で、同じ列（データ・ファイルのフィールド）で異なるサイズの LOB を、問題なく処理します。ただし、このような柔軟性によって、SQL*Loader で区切り文字列を探してデータをスキャンする必要があるため、パフォーマンスに影響します。

単一キャラクタのデリミタで、文字列のデリミタを指定する場合は、データ・ファイルのキャラクタ・セットに注意してください。データ・ファイルのキャラクタ・セットが制御ファイルのキャラクタ・セットと異なる場合は、デリミタを 16 進文字列の表記法で指定できます（X'hexadecimal string'）。デリミタを実際に 16 進数文字列で指定する場合は、入力データ・ファイルのキャラクタ・セット中の有効な文字で指定する必要があります。一方、16 進数文字列で指定しない場合、デリミタは、クライアント（制御ファイル）のキャラクタ・セットで指定してください。この場合、デリミタは、SQL*Loader によってデータ・ファイル内で検索される前に、データ・ファイルのキャラクタ・セットに変換されます。

LOBFILE からの LOB データのロード

LOB データは、非常に長いデータであるため、プライマリ・データ・ファイルではなく、LOBFILE からロードすることが有効です。LOBFILE では、LOB データのインスタンスは、フィールド（事前に決められたサイズ、デリミタ付き、Length-Value Pair）内にあるとみなされますが、これらのフィールドは、レコードに編成されていません（LOBFILE にはレコードの概念がありません）。そのため、レコードを扱うことによって発生するオーバーヘッドを回避できます。このようなデータの編成方法は、LOB のロードにとって理想的です。

LOBFILE の LOB はメモリーに収まる必要はありません。SQL*Loader は、LOBFILE を 64KB のチャンクで読み込みます。

LOBFILE のデータ・フィールドは、次のいずれかの型です。

- ファイルの内容全体を読み込む単一の LOB フィールド

- サイズが決められたフィールド（固定長フィールド）
- デリミタ付きフィールド（TERMINATED BY または ENCLOSED BY）
  - PRESERVE BLANKS 句は、LOBFILE から読み込むフィールドには使用できません。
  - Length-Value Pair フィールド（可変長フィールド）: SQL*Loader の VARRAY、VARCHAR、VARCHAR2 などのデータ型は、この型のフィールドに使用されます。

前述のすべてのフィールド型は、XML 列のロードに使用できます。

動的および静的 LOBFILE 指定

LOBFILE を静的に指定（ファイルにファイル名を指定）するか、または動的に指定（FILLER フィールドをファイル名のソースとして使用）できます。いずれの場合も、LOBFILE の終わりに到達するとファイルがクローズされ、そのファイルからさらにデータを読み込む場合は、空のフィールドからデータを読み込むことになります。

同じ LOBFILE を、2 つの異なるフィールドのソースとして指定しないでください。指定すると、通常、2 つのフィールドで、データが別々に読み込まれます。

23

XMLType 表のインポートおよびエクスポート

この章では、Oracle XML DB で使用するために XMLType 表をインポートおよびエクスポートする方法を説明します。

この章の内容は次のとおりです。

Oracle XML DB におけるインポート / エクスポート・ユーティリティのサポートの概要

Oracle XML DB は、XML データを格納し、登録された XML Schema に基づくことができる XMLType 表および列をサポートします。XML Schema に基づくデータまたは基づかないデータを格納する表は、インポートおよびエクスポートできます。

リソースおよびフォルダリングで完全にサポートされないインポート / エクスポート・ユーティリティ

Oracle XML DB は、ファイル・システムのようなパラダイムをデータベース・データに提供する、データベースのフォルダリング・メカニズムもサポートします。このモデルは、表名や列名などではなく、パス名および URI を使用してデータ（リソース）を参照します。ただし、今回のリリースでは、インポート / エクスポート・ユーティリティを使用するこのパラダイムはサポートされていません。

ただし、登録された XML Schema に基づくリソースの場合、データを格納する実際の XMLType 表はエクスポートおよびインポートできます。これは、XML データのみがエクスポートされ、Oracle XML DB Foldering 階層内の関係は失われることを意味します。

XML Schema に基づかない XMLType 表および列

XMLType 表および列は、XML Schema 仕様を使用せずに作成できます。この場合、XML データは CLOB として格納されます。

これらの表のデータは、LOB 列と同様の方法でエクスポートおよびインポートできます。実際の XML テキストは、エクスポート・ダンプ・ファイルに格納されます。

XML Schema に基づく XMLType 表

Oracle は、XML Schema に基づく XMLType 表のエクスポートおよびインポートをサポートします。XMLType 表は、それを定義するために使用された XML Schema に依存します。同様に、XML Schema はそれ用に作成または指定された SQL オブジェクト型に依存します。そのため、XMLType 表または XMLType 表を含むデータベースをエクスポートするには、次の手順を実行します。

1. **XML Schema の登録時に SQL 型をエクスポートします。**XML Schema の登録処理の一部として、SQL 型を作成できます。これらの SQL 型は、その OID とともに CREATE TYPE 文の一部としてエクスポートされます。
2. **XML Schema をエクスポートします。**すべての SQL 型のエクスポート後、XML Schema が DBMS_XMLSCHEMA.REGISTERSCHEMA 文の一部の XML テキストとしてエクスポートされます。この文には次の注意事項があります。

- FORCE フラグを TRUE に設定します。これは、インポート時の順不同での XML Schema 登録に必要です。エクスポート処理では XML Schema が特定の順序でダンプされないため、他の XML Schema をインポートする XML Schema が、インポートした XML Schema の後に出現しない場合もあります。また、XML Schema 間での循環参照では、XML Schema を正常に登録するために、FORCE フラグを TRUE に設定する必要があります。
- GENTYPES フラグを FALSE に設定します。型は、すでに手順 1 で生成されています。
- GENTABLES フラグを FALSE に設定します。表は、その OID とともに後（手順 3）で作成されます。OID は registerSchema 文で指定できません。
- GENBEANS フラグを FALSE に設定します。
- 所有者を参照するすべての属性（SchemaOwner や SQLSchema など）は抑制されます。これは、別のユーザーからデータをインポートできるようにするために必要です。

3. **XML 表をエクスポートします。**次に、表をエクスポートします。各表のエクスポート操作は、次の 2 つの手順で構成されます。
   1. 表定義が、その表の OID とともに CREATE TABLE 文の一部としてエクスポートされます。
   2. 表内のデータが、XML テキストとしてエクスポートされます。表外のデータは明示的にエクスポートされないことに注意してください。表外のデータは、親表のデータの一部としてエクスポートされます。

注意： OCT およびネストした表は、個別にエクスポートされません。これらは、親表の一部としてエクスポートされます。

階層対応表をエクスポートする場合のガイドライン

次に、階層対応表をエクスポートする場合のガイドラインを示します。

- 階層対応表の RLS ポリシーおよびパス索引トリガーはエクスポートされません。これは、これらの表がインポート時には階層対応でないことを意味します。
- 階層対応表の非表示列 ACLOID および OWNERID はエクスポートされません。これは、インポートされたデータベースではこれらの列の値が異なる場合があるため、それらの値を再初期化する必要があるためです。

インポート / エクスポート・ユーティリティの構文および例

インポート / エクスポート・ユーティリティの構文および詳細は、『Oracle9*i* データベース・ユーティリティ』を参照してください。この項では、XMLType データにインポート / エクスポート・ユーティリティを使用する場合のその他のガイドラインおよび例を示します。

インポート / エクスポート・ユーティリティの例に関する前提事項

この項に示す例では、次のようなデータベースが使用されていることを前提としています。

- 2 人のユーザー（U1 および U2）が存在する。
- U1 には、SL1 という登録されたローカル XML Schema がある。このユーザーは、デフォルト表 TL1 も作成済である。
- U1 には、SG1 という登録されたグローバル XML Schema がある。このユーザーは、デフォルト表 TG1 も作成済である。
- U2 は、XML Schema SG1 に基づく表 TG2 を作成済である。

ユーザー・レベルのインポートおよびエクスポート

例 23-1 XMLType データのエクスポート

```
export sytem/manager file=file1 owner=U1
```

これによって、次のものがエクスポートされます。

- XML Schema SL1 および SG1 の登録時に生成されたすべての型
- XML Schema SL1 および SG1
- 表 TL1 と TG1、および XML Schema SL1 と SG1 の登録時に生成された他のすべての表
- 前述のすべての表に含まれるすべてのデータ

例 23-2 XMLType 表のエクスポート

```
export sytem/manager file=file2 owner=U2
```

これによって、次のものがエクスポートされます。

- 表 TG2、および TG2 の作成時に生成された他のすべての表
- 前述のすべての表に含まれるすべてのデータ

注意： この例では、XML Schema SG1、または XML Schema SG1 の登録時に作成された型はエクスポートされません。

例 23-3 ファイルからのデータのインポート

```
import system/manager file=file1 fromuser=U1 touser=newuser
```

これによって、file1.dmp 内のすべてのデータがユーザー newuser にインポートされます。

表モード・エクスポート

XMLType 表は、それを定義するために使用された XML Schema に依存します。同様に、XML Schema はそれ用に作成または指定された SQL オブジェクト型に依存します。XMLType 表をインポートするには、その XML Schema および SQL オブジェクト型が必要です。表モード・エクスポートを使用すると、表関連のメタデータおよびデータのみがエクスポートされます。これらのデータを正常にインポートできるようにするには、XML Schema とオブジェクト型の両方が作成済であることを確認する必要があります。

例 23-4 表モードでの XML データのエクスポート

```
exp SYSTEM/MANAGER file=expdat.dmp owner=U1 tables=TG1
```

これによって、次のものがエクスポートされます。

- 表 TG1、および TG1 の作成時に生成された他のすべての表
- 前述のすべての表に含まれるすべてのデータ

注意： この例では、XML Schema SG1、または XML Schema SG1 の登録時に作成された型はエクスポートされません。

例 23-5 表モードでの XML データのインポート

```
imp SYSTEM/MANAGER file=expdat.dmp fromuser=U1 touser=U2 tables=TG1
```

ユーザー U2 は、グローバル XML Schema SG1 およびそれが依存する型へのアクセス権をすでに持っているため、このコマンドによって U2 用の表 TG1 が作成されます。

データベースの全体エクスポート時にエクスポートされないリポジトリのメタデータ

Oracle XML DB は、リポジトリ用のメタデータ（および XML Schema に基づかないデータ）を XML DB データベース・ユーザー・スキーマに格納します。Oracle はリポジトリ構造の

エクスポートをサポートしないため、データベースの全体エクスポートの実行時に、これらのメタデータ表および構造はエクスポートされません。

実際に、データベースの全体エクスポート時に XML DB ユーザー・スキーマ全体がスキップされ、XML DB（Oracle XML DB）が所有するすべてのデータベース・オブジェクトはエクスポートされません。

異なるキャラクタ・セットでのインポートおよびエクスポート

XML データは、他のデータベース・オブジェクトの場合と同様に、エクスポート側サーバーのキャラクタ・セットでエクスポートされます。インポート中、このデータはインポート側サーバーのキャラクタ・セットに変換されます。

第 VII 部

Advanced Queueing を使用した XML データの交換

第 VII 部では、Oracle Advanced Queueing（AQ）を使用した XML データの交換、および XMLType に対する AQ のサポートについて説明します。第 VII 部に含まれる章は、次のとおりです。

- 第 24 章「Advanced Queuing（AQ）および Oracle Streams を使用した XML データの変換」

24

Advanced Queuing（AQ）および Oracle Streams を使用した XML データの変換

この章では、Oracle Advanced Queuing を使用して XML データを変換する方法を説明します。この章の内容は次のとおりです。

AQ の概要

Oracle Advanced Queuing（AQ）には、次のデータベース統合化メッセージ・キューイング機能があります。

- メッセージを使用する 2 つ以上のアプリケーションの非同期通信を可能にし、その管理を行います。
- Point-to-Point 通信モデルおよびパブリッシュ・サブスクライブ通信モデルをサポートします。

Oracle9*i* データベースとメッセージ・キューイングの統合によって、Oracle9*i* データベースの整合性、信頼性、リカバリ能力、スケーラビリティ、パフォーマンスおよびセキュリティ機能がメッセージ・キューイングに提供されます。また、Oracle9*i* データベースとの統合によって、メッセージ・フローからのインテリジェント機能の抽出が容易になります。

AQ と XML の相互補完

XML は、企業間通信の標準フォーマットとして登場しました。XML は、ビジネス・アプリケーション間で通信されるデータを表現するだけでなく、XML でカプセル化されたビジネス・ロジックの表現にも使用されています。

Oracle9*i* データベースでは、AQ がネイティブ XML メッセージをサポートしており、AQ 操作を XML ベースの Internet-Data-Access-Presentation（iDAP）形式で定義できます。拡張可能なメッセージの起動プロトコルである iDAP は、インターネット標準に基づいて構築され、転送メカニズムとして HTTP プロトコルおよび電子メール・プロトコルを使用し、データ表示用の言語として XML を使用します。クライアントは、iDAP を使用して AQ にアクセスできます。

AQ および XML メッセージ・ペイロード

図 24-1 に、AQ を使用して 3 つのアプリケーションと通信する Oracle9*i* データベースを示します（メッセージ・ペイロードとして XML を使用します）。この使用例において AQ が実行する一般的なタスクは、次のとおりです。

- サブスクリプション・ルールを使用したメッセージ・フロー
- メッセージ管理
- メッセージからのビジネス・インテリジェンスの抽出
- メッセージ変換

これは、XML メッセージが、AQ を使用してアプリケーション間で非同期に渡される、企業内業務および企業間業務での使用例です。

- 企業内業務の典型的な使用例には、販売注文の遂行およびサプライチェーンの管理などがあります。

- 企業間業務処理では、複数の統合ハブが、インターネットのバックプレーンを介して通信できます。企業間業務での使用例は、旅行の予約、メーカーとサプライヤ間の調整、銀行間での資金の移動、保険請求の清算などです。

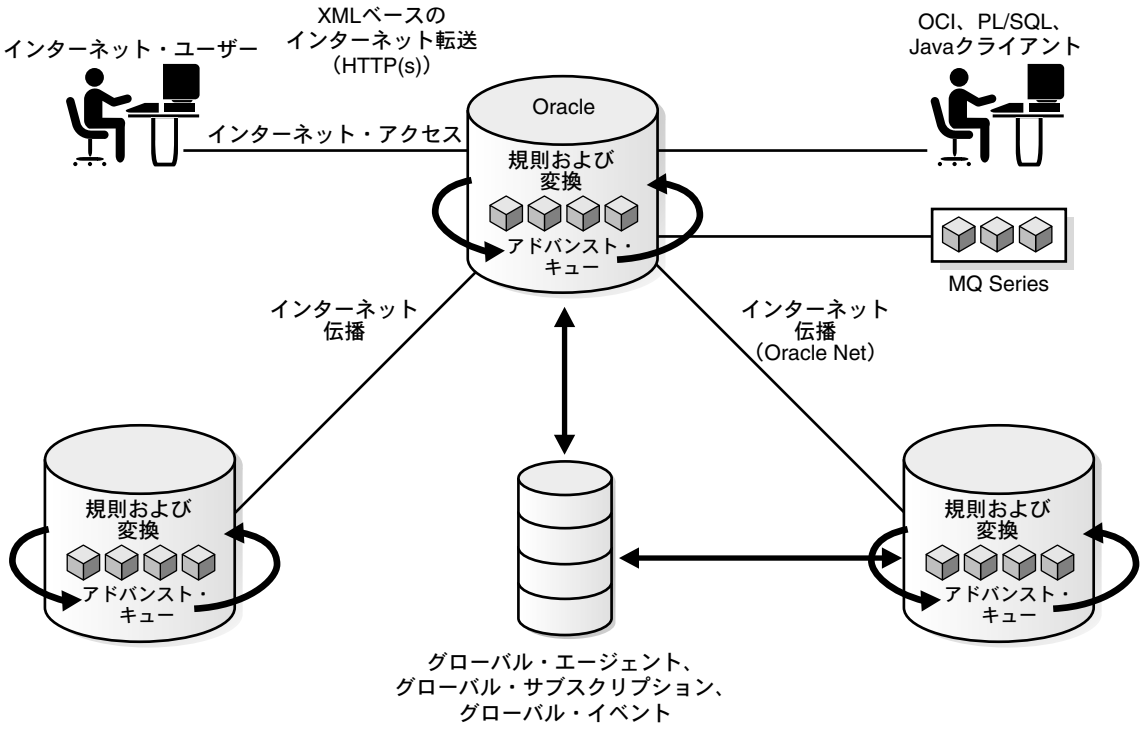
  Oracle は、これを企業アプリケーション統合製品で使用します。XML メッセージは、アプリケーションから AQ ハブへ送信されます。このサーバーは、メッセージを必要とするすべてのアプリケーションに対してメッセージ・サーバーとして機能します。このハブ・アンド・スポーク・アーキテクチャを介して、XML メッセージを、疎結合された複数の受信側アプリケーションへ非同期に送信できます。

図 24-1 に、AQ を使用して、次のアプリケーションを介して配送した XML ペイロード・メッセージを示します。

- iDAP を使用し、HTTP 接続を介した AQ 操作を行う Web ベースのアプリケーション
- Net* 接続を介して XML メッセージを伝播するために、AQ を使用するアプリケーション
- AQ を使用して、HTTP を介してインターネット /XML メッセージを直接データベースへ伝播するアプリケーション

この図は、AQ クライアントが、OCI、Java または PL/SQL を使用してデータにアクセスできることも示しています。

図 24-1 Advanced Queuing および XML メッセージ・ペイロード



アプリケーション統合のためのハブ・アンド・スポーク・アーキテクチャを可能にする AQ

今日の企業が直面している重大な問題の 1 つは、アプリケーションの統合です。アプリケーションの統合には、複雑な業務トランザクションを実行するための、複数の部門アプリケーションの協調、調整および同期化が伴います。

AQ は、アプリケーション統合のためのハブ・アンド・スポーク・アーキテクチャを可能にします。このアーキテクチャを使用すると、統合化ソリューションの管理、構成およびビジネス・ニーズの変化に伴う変更が容易になります。

監査、追跡およびマイニング用に保存できるメッセージ

AQ が提供するメッセージ管理では、異なるアプリケーション間でのメッセージ・フローを管理するだけでなく、将来のビジネス・インテリジェンスの監査、追跡および抽出のためにメッセージを保存することもできます。

SQL ビューを使用したメッセージ内容の表示

AQ は、メッセージを参照するための SQL ビューも提供します。これらの SQL ビューは、システムにおける過去、現在および将来の動向を分析するために使用できます。

AQ を使用するメリット

AQ を使用すると、異なるアプリケーション間での通信を柔軟に構成できます。

Oracle Streams および AQ

Oracle Streams（Streams）を使用すると、ストリーム内のデータおよびイベントを共有できます。ストリームは、この情報をデータベース内またはデータベース間で伝播できます。ストリームは、指定された情報を指定された宛先にルーティングします。これによって、イベントを取得および管理し、他のデータベースおよびアプリケーションとイベントを共有するための従来のソリューションより、機能性と柔軟性が向上します。

Streams を使用すると、あるソリューションを使用したために別のソリューションが使用できなくなるということがなくなります。Streams を使用すると、分散した企業とアプリケーション、データ・ウェアハウスおよび高可用性ソリューションを構築および操作できます。Oracle Streams のすべての機能は同時に使用できます。

Streams を使用して、次の操作を実行できます。

- **データベースでの変更の取得 :** バックグラウンドの取得プロセスが表、データベース・スキーマまたはデータベース全体に加えられた変更を取得するするように構成できます。取得プロセスでは、REDO ログから変更が取得され、取得された各変更が論理変更レコード（LCR）にフォーマットされます。REDO ログに変更が生成されたデータベースをソース・データベースといいます。
- **キューへのイベントのエンキュー :** Streams キューには、LCR およびユーザー・メッセージという 2 つのタイプのイベントをステージングできます。取得プロセスでは、LCR イベントが、指定したキューにエンキューされます。そのキューは、LCR イベントを同じデータベース内で共有するか、または他のデータベースと共有できます。また、ユーザー・アプリケーションを使用してユーザー・イベントを明示的にエンキューすることもできます。明示的にエンキューされたこれらのイベントは、LCR またはユーザー・メッセージになります。
- **あるキューから別のキューへのイベントの伝播 :** これらのキューは、同じデータベース内または異なるデータベース内に存在できます。
- **イベントのデキュー :** バックグラウンドの適用プロセスによって、イベントをデキューできます。また、ユーザー・アプリケーションを使用してイベントを明示的にデキューすることもできます。

- **データベースでのイベントの適用 :** 適用プロセスがキュー内のすべてのイベントまたは指定したイベントのみを適用するように構成できます。また、適用プロセスが、ユーザー独自の PL/SQL サブプログラムをコールしてイベントを処理するように構成することもできます。

  LCR イベントが適用され、他のタイプのイベントが処理されるデータベースを宛先データベースといいます。構成によっては、ソース・データベースと宛先データベースが同じ場合もあります。

Streams メッセージ・キューイング

Streams を使用すると、ユーザー・アプリケーションは次の操作を実行できます。

- 様々な型のメッセージのエンキュー
- 使用可能なメッセージの伝播
- 宛先データベースでのメッセージのデキュー

Streams では、SYS.AnyData 型のメッセージをステージングする新しいタイプのキューが導入されています。ほぼすべての型のメッセージを SYS.AnyData ラッパーでラップし、SYS.AnyData キューにステージングできます。Streams は、Advanced Queuing（AQ）と相互に作用します。これによって、マルチ・コンシューマ・キュー、パブリッシュとサブスクライブ、コンテンツ・ベースのルーティング、インターネット伝播、変換、他のメッセージ・サブシステムへのゲートウェイなど、メッセージ・キューイング・システムのすべての標準機能がサポートされます。

> **参照 :** 『Oracle9*i* Streams』の付録 A「XML Schema for LCRs」を参照してください。

オブジェクト型の XMLType 属性

今回のリリースでは、XMLType 属性を持つ Oracle オブジェクト型を使用するキューを作成できます。これらのキューは、XML 文書であるメッセージを転送および格納するために使用できます。XMLType を使用すると、次の操作を実行できます。

- 任意の型のメッセージをキューに格納する。
- 文書を CLOB として内部的に格納する。
- 複数の型のペイロードをキューに格納する。
- existsNode() 演算子を使用して、XMLType 列を問い合せる。
- サブスクライバ・ルールまたはデキュー条件にその演算子を指定する。

Internet Data Access Presentation（iDAP）

Simple Object Access Protocol（SOAP）を使用して、インターネット上で AQ にアクセスできます。Internet Data Access Presentation（iDAP）は、AQ 操作に対する SOAP 仕様です。iDAP によって、SOAP リクエストの本体に XML メッセージ構造が定義されます。iDAP によって構造化されたメッセージは、HTTP などの転送プロトコルを使用してインターネット上で転送されます。

iDAP は、Content-Type の `text/xml` を使用して、SOAP リクエストの本体を指定します。XML は、iDAP リクエストおよび応答メッセージの表示方法を次のとおり定義します。

- すべてのリクエスト・タグおよびレスポンス・タグは、SOAP 名前空間で有効です。
- AQ 操作は、iDAP 名前空間で有効です。
- 送信者は、SOAP 本体の iDAP 要素および属性に名前空間を含めます。
- 受信者は、適切な名前空間を含む iDAP メッセージを処理します。不適切な名前空間を持つリクエストの場合、リクエストが無効であるというエラーを戻します。
- SOAP 名前空間の値は、`http://schemas.xmlsoap.org/soap/envelope/` です。
- iDAP 名前空間の値は、`http://ns.oracle.com/AQ/schemas/access` です。

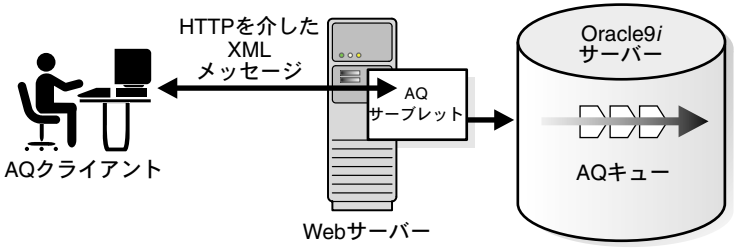
> **参照：**『Oracle9*i* アプリケーション開発者ガイド - アドバンスト・キューイング』を参照してください。

iDAP アーキテクチャ

図 24-2 に、HTTP メッセージを送信するために必要な次のコンポーネントを示します。

- iDAP 形式に準拠した XML メッセージを AQ サーブレットに送信するクライアント・プログラム。これには、Web ブラウザなど任意の HTTP クライアントが含まれます。
- Apache JServ、Tomcat など、受信した XML メッセージを解析するための AQ サーブレットをホストする Web サーバーまたはサーブレット・コンテナ。
- Oracle9*i* データベース・サーバーまたはデータベース。AQ サーブレットは、Oracle9*i* データベースに接続し、キューに対して操作を実行します。

図 24-2 HTTP を使用して AQ 操作を実行するための iDAP アーキテクチャ



XMLType キュー・ペイロード

XMLType 属性を含むペイロードを持つキューを作成できます。これらのペイロードは、XML 文書を含むメッセージの転送および格納に使用できます。XMLType 属性を持つ Oracle オブジェクトを定義すると、次の操作を実行できます。

- 複数の型の XML 文書を同じキュー内に格納します。この文書は、CLOB として内部的に格納されます。
- existsNode() や extract() などの演算子を使用して、XMLType 属性を持つメッセージを選択的にデキューします。
- 変換を定義して、Oracle オブジェクトを XMLType に変換します。
- existsNode() や extract() などの XMLType 演算子を使用して、メッセージ内容を問い合せるルールベースのサブスクライバを定義します。

例 24-1 AQ および XMLType キュー・ペイロードの使用 : 海外向け出荷キュー表およびキューの作成および表示の変換

BooksOnline のアプリケーションの海外向け出荷サイトでは、注文が SYS.XMLType として表現されると想定します。注文入力サイトでは、注文が Oracle オブジェクトの ORDER_TYP として表現されます。

海外向けキュー表およびキューは、次のとおり作成されます。

```
BEGIN
dbms_aqadm.create_queue_table(
   queue_table        => 'OS_orders_pr_mqtab',
   comment            => 'Overseas Shipping MultiConsumer Orders queue table',
   multiple_consumers => TRUE,
   queue_payload_type => 'SYS.XMLTtype',
   compatible         => '8.1');
END;
```

```
BEGIN
dbms_aqadm.create_queue (
   queue_name    => 'OS_bookedorders_que',
   queue_table   => 'OS_orders_pr_mqtab');
END;
```

海外向け出荷サイトの注文の表現は、注文入力サイトでの注文の表現と異なるため、メッセージが注文入力サイトから海外向け出荷サイトに伝播される前に、変換が適用されます。

```
/*  Add a rule-based subscriber (for Overseas Shipping) to the Booked orders queues
with Transformation. Overseas Shipping handles all non-US orders: */
DECLARE
  subscriber     aq$_agent;
BEGIN
  subscriber := aq$_agent('Overseas_Shipping','OS.OS_bookedorders_que',null);

  dbms_aqadm.add_subscriber(
   queue_name     => 'OE.OE_bookedorders_que',
   subscriber     => subscriber,
   rule           => 'tab.user_data.orderregion = ''INTERNATIONAL''',
   transformation => 'OS.OE2XML');
END;
```

注文入力アプリケーションによって使用される型から、海外向け出荷によって使用される型への変換の定義の詳細は、『Oracle9*i* アプリケーション開発者ガイド - アドバンスト・キューイング』の第 8 章の「変換の作成」を参照してください。

例 24-2 AQ および XMLType キュー・ペイロードの使用 : メッセージのデキュー

アプリケーションが、カナダの顧客の注文を処理すると想定します。このアプリケーションは、次のプロシージャを使用してメッセージをデキューできます。

```
/*  Create procedures to enqueue into single-consumer queues: */
create or replace procedure get_canada_orders() as
deq_msgid               RAW(16);
dopt                    dbms_aq.dequeue_options_t;
mprop                   dbms_aq.message_properties_t;
deq_order_data          SYS.XMLTtype;
no_messages             exception;
pragma exception_init   (no_messages, -25228);
new_orders              BOOLEAN := TRUE;

begin
        dopt.wait := 1;

/* Specify dequeue condition to select Orders for Canada */
        dopt.deq_condition := 'tab.user_data.extract(
```

```
''/ORDER_TYP/CUSTOMER/COUNTRY/text()'').getStringVal()=''CANADA''';

           dopt.consumer_name : = 'Overseas_Shipping';

        WHILE (new_orders) LOOP
          BEGIN
            dbms_aq.dequeue(
                queue_name         => 'OS.OS_bookedorders_que',
                dequeue_options    => dopt,
                message_properties => mprop,
                payload            => deq_order_data,
                msgid              => deq_msgid);
            commit;

            dbms_output.put_line(' Order for Canada - Order: ' ||
                                 deq_order_data.getStringVal());

          EXCEPTION
            WHEN no_messages THEN
                 dbms_output.put_line (' ---- NO MORE ORDERS  ---- ');
                 new_orders := FALSE;
          END;
        END LOOP;
end;
```

AQ XML サーブレットを使用したエンキュー

iDAP を使用して、インターネット上でエンキュー・リクエストを実行できます。

> **参照：** iDAP を使用した AQ リクエストの送信の詳細は、『Oracle9*i* アプリケーション開発者ガイド - アドバンスト・キューイング』を参照してください。

シナリオ

BooksOnLine のアプリケーションの注文入力システムでは、入力済注文を FIFO 優先順位のキューに格納します。入力済注文は、地区の入力済注文キューに伝播されます。各地区では、その地区の入力済注文キューにある注文が、出荷の優先順位に従って処理されます。

例 24-3 PL/SQL（DBMS_AQADM パッケージ）

次のコールで、注文入力アプリケーションに優先順位キューを作成します。

```
/* Create a priority queue table for OE: */
EXECUTE dbms_aqadm.create_queue_table( \
   queue_table         => 'OE_orders_pr_mqtab', \
   sort_list           =>'priority,enq_time', \
   comment             => 'Order Entry Priority  \
                          MultiConsumer Orders queue table',\
   multiple_consumers  => TRUE, \
   queue_payload_type  => 'BOLADM.order_typ', \
   compatible          => '8.1', \
   primary_instance    => 2, \
   secondary_instance  => 1);

EXECUTE dbms_aqadm.create_queue ( \
   queue_name          => 'OE_bookedorders_que', \
   queue_table         => 'OE_orders_pr_mqtab');
```

顧客 John が、SOAP を使用してエンキュー・リクエストを送信すると想定します。XML メッセージのフォーマットは、次のとおりです。

```
<?xml version="1.0"?>
   <Envelope xmlns= "http://schemas.xmlsoap.org/soap/envelope/">
      <Body>
        <AQXmlSend xmlns = "http://ns.oracle.com/AQ/schemas/access">
          <producer_options>
            <destination>OE.OE_bookedorders_que</destination>
          </producer_options>

          <message_set>
            <message_count>1</message_count>

            <message>
              <message_number>1</message_number>
              <message_header>
                <correlation>ORDER1</correlation>
                <priority>1</priority>
                <sender_id>
                   <agent_name>john</agent_name>
                </sender_id>
              </message_header>

              <message_payload>
               <ORDER_TYP>
                     <ORDERNO>100</ORDERNO>
                     <STATUS>NEW</STATUS>
```

```
                        <ORDERTYPE>URGENT</ORDERTYPE>
                        <ORDERREGION>EAST</ORDERREGION>
                        <CUSTOMER>
                           <CUSTNO>1001233</CUSTNO>
                           <CUSTID>JOHN</CUSTID>
                           <NAME>JOHN DASH</NAME>
                           <STREET>100 EXPRESS STREET</STREET>
                           <CITY>REDWOOD CITY</CITY>
                           <STATE>CA</STATE>
                           <ZIP>94065</ZIP>
                           <COUNTRY>USA</COUNTRY>
                        </CUSTOMER>
                        <PAYMENTMETHOD>CREDIT</PAYMENTMETHOD>
                        <ITEMS>
                           <ITEMS_ITEM>
                              <QUANTITY>10</QUANTITY>
                              <ITEM>
                                 <TITLE>Perl handbook</TITLE>
                                 <AUTHORS>Randal</AUTHORS>
                                 <ISBN>345620200</ISBN>
                                 <PRICE>19</PRICE>
                              </ITEM>
                              <SUBTOTAL>190</SUBTOTAL>
                           </ITEMS_ITEM>
                           <ITEMS_ITEM>
                              <QUANTITY>10</QUANTITY>
                              <ITEM>
                                 <TITLE>JDBC guide</TITLE>
                                 <AUTHORS>Taylor</AUTHORS>
                                 <ISBN>123420212</ISBN>
                                 <PRICE>59</PRICE>
                              </ITEM>
                              <SUBTOTAL>590</SUBTOTAL>
                           </ITEMS_ITEM>
                        </ITEMS>
                        <CCNUMBER>NUMBER01</CCNUMBER>
                        <ORDER_DATE>08/23/2000 12:45:00</ORDER_DATE>
                     </ORDER_TYP>
                  </message_payload>
               </message>
            </message_set>

            <AQXmlCommit/>
         </AQXmlSend>
      </Body>
   </Envelope>
```

AQ XML サーブレットを使用したデキュー

SOAP を使用して、インターネット上でデキュー・リクエストを実行できます。

> **参照：** SOAP を使用した AQ メッセージの受信の詳細は、『Oracle9*i* アプリケーション開発者ガイド - アドバンスト・キューイング』を参照してください。

BooksOnline のシナリオで、東部向けの出荷アプリケーションが、相関識別子 RUSH の AQ メッセージを、インターネット経由で受信すると想定します。

例 24-4 AQ XML メッセージの受信およびデキュー

デキュー・リクエストのフォーマットは、次のとおりです。

```
<?xml version="1.0"?>
<Envelope xmlns= "http://schemas.xmlsoap.org/soap/envelope/">
      <Body>
        <AQXmlReceive xmlns = "http://ns.oracle.com/AQ/schemas/access">
          <consumer_options>
            <destination>ES_ES_bookedorders_que</destination>
            <consumer_name>East_Shipping</consumer_name>
            <wait_time>0</wait_time>
            <selector>
                 <correlation>RUSH</correlation>
            </selector>
          </consumer_options>

          <AQXmlCommit/>

        </AQXmlReceive>
      </Body>
</Envelope>
```

iDAP スキーマおよび AQ XML Schema

iDAP は、SOAP および AQ XML Schema をクライアントに公開します。パーサーによって送信されるすべてのドキュメントは、これらのスキーマに対して妥当性が検証されます。

- SOAP スキーマ : `http://schemas/xmlsoap.org/soap/envelope/`
- AQ XML Schema: `http://ns.oracle.com/AQ/schemas/access`

参照 :

- `DBMS_AQADM` または Java（JDBC）のいずれかを使用した構造化メッセージ・ペイロード・アプリケーションの実装方法の詳細は、『Oracle9*i* アプリケーション開発者ガイド - アドバンスト・キューイング』の第 8 章を参照してください。
- `DBMS_TRANSFORM` の詳細は、『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

FAQ: XML および AQ

多くの PDF ファイルを持つ AQ XML メッセージを 1 つのレコードとして格納する方法

Oracle Advanced Queuing を使用して、XML 文書をあるビジネス分野から別のビジネス分野へ変更する予定です。受信または送信される各メッセージには、XML ヘッダー、XML アタッチメント（XML データ・ストリーム）、DTD および PDF ファイルが含まれます。これらのすべての情報を、画像ファイルも含めてデータベース表（この場合はキュー表）に格納する必要があります。

このメッセージを、1 つのレコードまたは 1 つのピースとして Oracle キュー表にエンキューできますか ? または、このメッセージを複数のレコード（XML データ・ストリームに対する CLOB 型としての 1 つのレコード、PDF ファイルに対する RAW としての 1 つのレコードなど）としてエンキューしてから、これらのレコードの相関を指定する必要がありますか ? また、メッセージをデキューしたことを確認する必要もあります。

回答 : 次の方法で可能です。

- CLOB、RAW などの属性を持つオブジェクト型を定義し、これを単一メッセージとして格納します。
- AQ メッセージ・グループ化機能を使用して、該当するメッセージを複数のメッセージに格納します。ただし、メッセージのプロパティは 1 つのグループに対応付けられます。メッセージ・グループ化機能を使用するには、すべてのメッセージのペイロード型が同じである必要があります。

最初にペイロード型を CLOB として指定し、次にエンキューして格納する方法

最初にペイロード型を CLOB として指定し、すべてのピース、XML メッセージ・データ・ストリーム、DTD、PDF などを単一のメッセージ・ペイロードとしてエンキューしてキュー表に格納するのですか？その場合、このメッセージをデキューするときに、どのようにしてこの単一メッセージを個々に分割するのですか？

回答：そうではありません。まず、次のとおりオブジェクト型を作成します。

```
CREATE TYPE mypayload_type as OBJECT (xmlDataStream CLOB, dtd CLOB, pdf BLOB);
```

次に、このオブジェクト型を単一のメッセージとして格納します。

メッセージをエンキューした後の新しい受信者の追加

メッセージ割当てをサポートするためにキュー表を使用します。たとえば、他のビジネス分野は、メッセージを Oracle に送信するときに、これらのメッセージを処理するために誰が割り当てられるかは認識していません。ただし、そのメッセージが人事部宛てであることは認識しています。そのため、すべてのメッセージは人事部管理者へ送信されます。

この時点で、メッセージはキュー表にエンキューされています。このメッセージの受信者は人事部管理者のみであり、その他のすべての人事部社員はこのキューのサブスクライバとして事前定義されています。人事部管理者は新しい受信者であるその他の社員を、キュー表内の既存のメッセージに関する message_properties.recipient_list に追加することができますか？

メッセージがエンキューされるときに複数のコンシューマ（受信者）は存在しませんが、メッセージがキュー表にエンキューされた後に、古い受信者を置き換えるか、または新しい受信者を追加する必要があります。この新しい受信者が、新しいメッセージをデキューします。このようなことは可能ですか？または、メッセージを古い受信者から削除してから、同じメッセージ内容を新しい受信者にエンキューする必要がありますか？

回答：メッセージがエンキューされた後に受信者のリストを変更することはできません。受信者のリストを指定しない場合、サブスクライバはそのキューをサブスクライブし、メッセージをデキューできます。

この場合、新しい受信者がそのキューのサブスクライバである必要があります。それ以外の場合は、メッセージをデキューし、そのメッセージを新しい受信者が再度エンキューする必要があります。

Oracle での XML メッセージのエンキュー、デキューおよび処理方法

OTN ドキュメントには、Oracle データベースは、XML メッセージをエンキュー、デキューおよび処理できると記載されています。これは、どのように実行されますか？

XML SQL Utility（XSU）を使用して、XML ファイルを処理する前に表に挿入する必要がありますか？または、XML ファイルを直接エンキューし、解析してから AQ プロセスを介してそのメッセージをディスパッチできますか？XML データを Oracle9*i* データベースに挿入または更新するたびに、XML SQL Utility を使用する必要がありますか？

回答：AQ は、オブジェクトのエンキューおよびデキューをサポートしています。これらのオブジェクトは、XML 文書を含む `XMLType` 型の属性、およびメッセージとともに送信する必要がある、抽出されたメタデータ属性を持つことができます。詳細および例については、『Oracle9*i* アプリケーション開発者ガイド - アドバンスト・キューイング』を参照してください。

XML コンテンツを持つメッセージを AQ キューから解析する方法

XML コンテンツを持つメッセージを AQ キューから解析し、ODS（Operational Data Store）にある表およびフィールドを更新するツールが必要です。XML 文書を取り出して解析し、特定のフィールドをデータベース表および列にマップする必要があります。Oracle Text で可能ですか？

カスタム・ソリューションを使用すると XML SQL Utility（XSU）を使用できます。主な目的はサプライチェーンです。ある特定の XML タグ値の問合せに基づいて、AQ エンキュー / デキューの回数や JMS ヘッダー情報などのメタデータ情報を取得する必要があります。単純に CLOB に XML を格納して、Oracle Text を使用して問合せを発行できますか？

回答：最も簡単な方法は、Oracle9*i* データベース内で AQ と同時に、Oracle XML Parser for Java および Java ストアド・プロシージャを使用することです。

XSU の使用については、次のことに注意してください。

- XML を CLOB で格納する場合、Oracle Text（以前の *inter*Media Text）を使用して検索できますが、この場合は、ある基準に一致する特定のメッセージのみを検索できます。
- メタデータ上で集計操作を行う必要がある場合は、既存の関連ツールからメタデータを参照するか、または通常の SQL 述語をメタデータに使用します。この場合、CLOB に XML で格納するのみでは十分ではありません。

Oracle Text の XML 検索と、抽出された列などの重複したメタデータ記憶域を組み合せ、通常の SQL 述語と Oracle Text の CONTAINS() 句を組み合せる SQL 文を使用すると、両方の最適な機能を利用できます。

参照： 第 7 章「Oracle Text を使用した XML データの検索」を参照してください。

XML 文書が処理されるまでのリスナー停止の回避

クライアントからの XML メッセージの受信後、すぐに処理する必要があります。各 XML 文書の処理には、約 15 秒かかります。PL/SQL を使用しています。1 つの PL/SQL プロシージャがリスナーを起動し、メッセージをデキューし、別のプロシージャをコールして XML 文書を処理します。問題は、XML 文書が処理されるまで、リスナーが停止することです。その間、メッセージがキューに蓄積します。

これを処理する最適な方法を教えてください。リスナー・プログラムが、XML 処理プロシージャを非同期にコールし、リスニングへ戻る方法はありますか？現時点では、Java は使用しません。

回答：メッセージを受信後、`DBMS_JOB` パッケージを使用してジョブを送信します。ジョブは、異なるデータベース・セッションで非同期に起動されます。

Oracle9*i* データベースでは、AQ 通知フレームワークに PL/SQL コールバックが追加されています。これによって、メッセージがキューに入れられると非同期に起動される PL/SQL コールバックを登録できます。

AQ とともに HTTPS を使用する方法

HTTPS を使用して、XML メッセージをサプライヤに送信し、レスポンスを受信する必要があります。

HTTP を使用したメッセージの送信は、`java.net.URLConnection` などを使用して問題なく行うことができますが、HTTPS 接続を使用する方法が不明です。Portal や OC4J などの製品はソース・コードに HTTP クライアントが含まれているようですが、Oracle またはその他の製品には HTTPS 接続に再利用できる HTTP クライアントが含まれていますか？

回答：Oracle9*i* AQ のインターネット・アクセス機能を使用できます。この機能を使用すると、XML を使用して確実なトランザクション処理で HTTP を介してメッセージをエンキューまたはデキューできます。この機能の詳細は、次の Web サイトを参照してください。

`http://otn.oracle.com/products/aq`

AQ メッセージ・ペイロードで XML を格納する場合のオプション

XML を AQ メッセージ・ペイロードに格納する場合、ユーザー定義型の一部として sys.xmltype を使用し、ユーザー定義型をペイロードに含める以外に、ネイティブな方法はありますか？たとえば、オブジェクト（`xml_data sys.XMLType`）として、`xml_data_typ` を作成し、置換します。メッセージ・ペイロードには、RAW 型またはユーザー定義型のいずれか以外は含めることができないのですか？

回答：Oracle9*i* データベース リリース 1（9.0.1）では、ユーザー定義型の一部として sys.xmltype を使用し、そのユーザー定義型をペイロードに含めることが、`XMLType` をキューに格納する唯一の方法です。Oracle9*i* データベース リリース 2（9.2）では、`XMLType` のペイロードおよび属性を持つキューを作成できます。

iDAP と SOAP を比較する方法

回答：iDAP は、AQ 操作に対する SOAP 仕様です。iDAP は、AQ 操作の XML 仕様でもあります。SOAP は、サービスを起動するための共通のメカニズムを定義します。iDAP は、AQ 操作を実行するためのメカニズムを定義します。

さらに iDAP には、SOAP が定義していない、次の主要なプロパティが含まれます。

- トランザクション動作 : AQ 操作をトランザクションとして実行できます。トランザクションは、複数の iDAP 要求にまたがることができます。
- セキュリティ : すべての iDAP 操作は、許可済ユーザーおよび認証済ユーザーのみが実行できます。

第VIII部

Oracle XML DB の事例

第 VIII 部では、Oracle XML DB の事例について説明します。第 VIII 部に含まれる章は、次のとおりです。

- 第 25 章「Oracle XML DB の事例 : Web サービスによる XML 文書の取出しおよび表示」
- 第 26 章「Oracle XML DB Basic Demo」

25

Oracle XML DB の事例 : Web サービスによる XML 文書の取出しおよび表示

この章では、Oracle XML DB Repository に格納された XML 文書に Web サービスを介してアクセスする方法を説明します。この章の内容は次のとおりです。

XML DB Web サービスの事例 : 概要

この事例では、Web サービスを使用して、XML DB に格納された XML 発注書（PO）をフェッチする方法を示します。アプリケーションは、ブラウザで入力された PO 番号に基づいて、Java サーブレットを使用して Web サービスを起動します。次に、Web サービスが XML DB にアクセスし、Java API for XMLType を起動します。SQL 問合せが処理され、アプリケーションが、ブラウザで表示する XML 文書を XML DB から Java 文字列として取り出します。

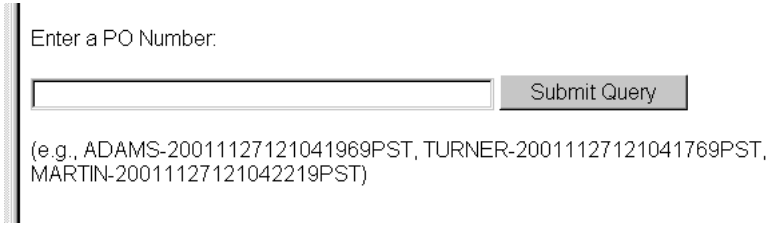
注意 :

- この事例は、Internet Explorer 6.0 以上で実行してください。
- Oracle XML DB での PO XML 文書の格納、アクセスおよび処理方法のその他の例については、第 26 章「Oracle XML DB Basic Demo」を参照してください。

PO 番号の入力時に発生する状況

このアプリケーションは、ブラウザで PO 番号を入力すると起動します。図 25-1 を参照してください。

図 25-1 発注書（PO）番号の入力

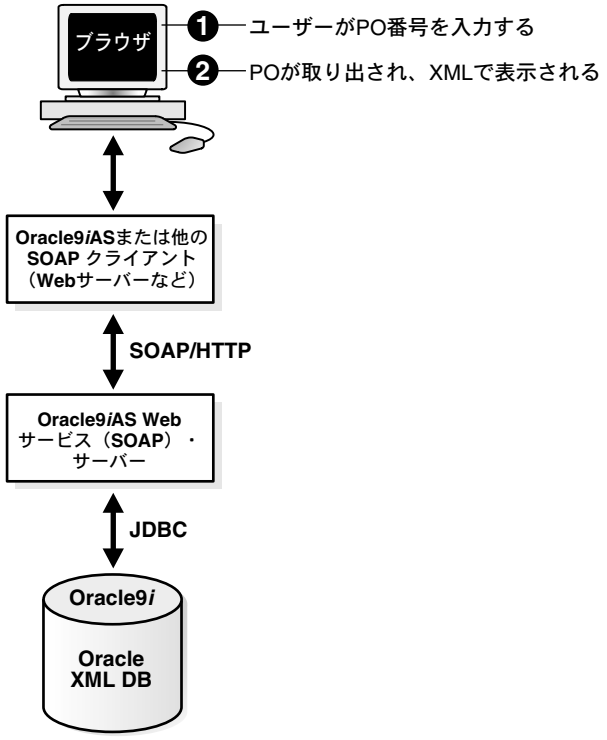


Oracle XML DB Web サービス : 主なコンポーネント

図 25-2 に、XML DB Web サービス・アプリケーションで使用される主なコンポーネントを示します。

- ブラウザ
- SOAP クライアント
- Web サービス・サーバー
- Oracle9*i* 上の Oracle XML DB

図 25-2 XML DB Web サービスの事例 : 主なコンポーネント



XML DB Web サービスの事例の実行 : 実装手順

この項では、XML DB Web サービスの事例を実行する手順を説明します。

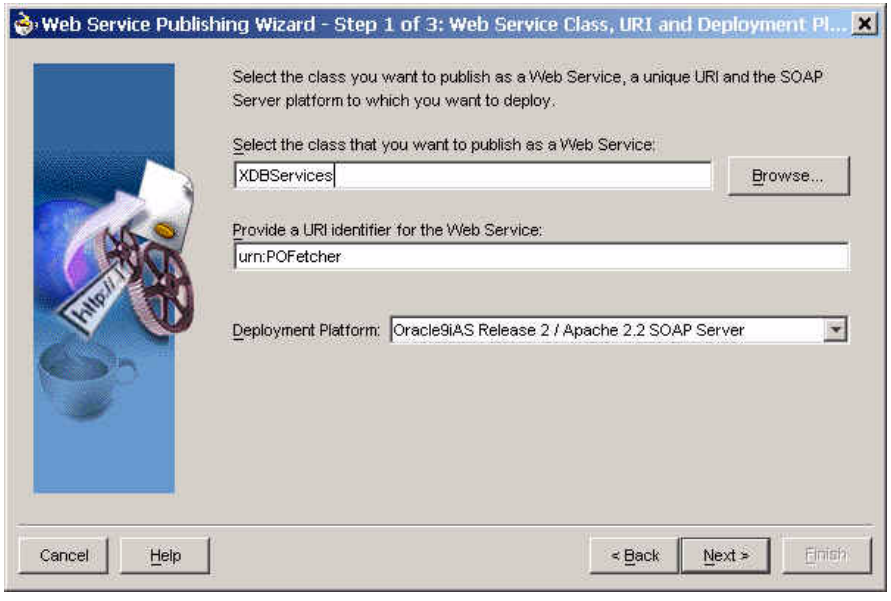
事例デモの実行準備

この XML DB Web サービスの事例を実行する前に、次の操作を実行します。

- Internet Explorer 6.0 以上がインストール済であることを確認します。
- Oracle9*i* JDeveloper を使用して、必要な Web サービス・ファイルを生成します。図 25-3 に、Oracle9*i* JDeveloper JavaBean ウィザードを示します。このウィザードを使用して、Web サービスにアクセスするために使用する次の 3 ファイルを生成します。
  - WSDL 文書（`XDBServicesService.wsdl`）。25-5 ページの「XDBServicesService.wsdl」を参照してください。
  - デプロイメント・ディスクリプタ（`XDBServicesDeploymentDescriptor.dd`）。25-6 ページの「XDBServicesDeploymentDescriptor.dd」を参照してください。
  - Web サービス・クライアント・スタブ（`XDBServicesStub.java`）。25-7 ページの「XDBServicesStub.java」を参照してください。

これら 3 つのファイルについては、次の項を参照してください。

図 25-3 Oracle9*i* JDeveloper JavaBean ウィザードの使用



XDBServicesService.wsdl

```
<?xml version="1.0" ?>
- <!-- Generated by the Oracle9i JDeveloper Web Services WSDL Generator -->
- <!-- Date Created: Mon Jul 15 16:34:24 PDT 2002-->
- <definitions name="XDBServices"
   targetNamespace="http://www.oracle.com/jdeveloper/generated/XDBServices"
   xmlns="http://schemas.xmlsoap.org/wsdl/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns:tns="http://www.oracle.com/jdeveloper/generated/XDBServices"
   xmlns:ns1="http://www.oracle.com/jdeveloper/generated/XDBServices/schema">
- <types>
 <schema
 targetNamespace="http://www.oracle.com/jdeveloper/generated/XDBServices/schema"
 xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/" />
</types>
- <message name="getPOFromNumber0Request">
    <part name="PONumber" type="xsd:string" />
  </message>
- <message name="getPOFromNumber0Response">
    <part name="return" type="xsd:string" />
  </message>
- <message name="getPOXML1Request">
    <part name="PONumber" type="xsd:string" />
  </message>
- <message name="getPOXML1Response">
    <part name="return" type="xsd:string" />
  </message>
- <portType name="XDBServicesPortType">
- <operation name="getPOFromNumber">
<input name="getPOFromNumber0Request" message="tns:getPOFromNumber0Request" />
<output name="getPOFromNumber0Response" message="tns:getPOFromNumber0Response" />
</operation>
- <operation name="getPOXML">
   <input name="getPOXML1Request" message="tns:getPOXML1Request" />
   <output name="getPOXML1Response" message="tns:getPOXML1Response" />
  </operation>
</portType>
- <binding name="XDBServicesBinding" type="tns:XDBServicesPortType">
  <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
- <operation name="getPOFromNumber">
  <soap:operation soapAction="" style="rpc" />
- <input name="getPOFromNumber0Request">
  <soap:body use="encoded" namespace="urn:POFetcher"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
```

```
</input>
- <output name="getPOFromNumber0Response">
<soap:body use="encoded" namespace="urn:POFetcher"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
</output>
</operation>
- <operation name="getPOXML">
   <soap:operation soapAction="" style="rpc" />
- <input name="getPOXML1Request">
   <soap:body use="encoded" namespace="urn:POFetcher"
    encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
  </input>
- <output name="getPOXML1Response">
  <soap:body use="encoded" namespace="urn:POFetcher"
    encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
</output>
</operation>
</binding>
- <service name="XDBServices">
- <port name="XDBServicesPort" binding="tns:XDBServicesBinding">
   <soap:address
    location="http://dlsun653.us.oracle.com:7070/soap/servlet/rpcrouter" />
  </port>
 </service>
</definitions>
```

XDBServicesDeploymentDescriptor.dd

```
<?xml version="1.0" ?>
- <!-- Generated by the Oracle9i JDeveloper Web Services Deployment Descriptor
Generator -->
- <!-- This Deployment Descriptor file is for use with the Oracle9iAS Release 2 /
Apache 2.2 SOAP Server SOAP Server -->
- <!-- Date Created: Mon Jul 15 16:34:26 PDT 2002-->
   - <isd:service id="urn:POFetcher"
         xmlns:isd="http://xml.apache.org/xml-soap/deployment">
     - <isd:provider type="java" methods="getPOFromNumber getPOXML"
         scope="Request">
          <isd:java class="XDBServices" static="false" />
       </isd:provider>
<isd:faultListener>org.apache.soap.server.DOMFaultListener</isd:faultListener>
</isd:service>
```

XDBServicesStub.java

```
package mypackage;
import oracle.soap.transport.http.OracleSOAPHTTPConnection;
import java.net.URL;
import org.apache.soap.Constants;
import org.apache.soap.Fault;
import org.apache.soap.SOAPException;
import org.apache.soap.rpc.Call;
import org.apache.soap.rpc.Parameter;
import org.apache.soap.rpc.Response;
import java.util.Vector;
import java.util.Properties;
/**
 * Generated by the Oracle9i JDeveloper Web Services Stub/Skeleton Generator.
 * Date Created: Mon Jul 15 16:36:21 PDT 2002
 */

public class XDBServicesStub
{
  public String endpoint = "http://localhost:7070/soap/servlet/rpcrouter";
  private OracleSOAPHTTPConnection m_httpConnection = null;

  public XDBServicesStub()
  {
    m_httpConnection = new OracleSOAPHTTPConnection();
  }

  public String getPOFromNumber(String PONumber) throws Exception
  {
    String returnVal = null;

    URL endpointURL = new URL(endpoint);
    Call call = new Call();
    call.setSOAPTransport(m_httpConnection);
    call.setTargetObjectURI("urn:POFetcher");
    call.setMethodName("getPOFromNumber");
    call.setEncodingStyleURI(Constants.NS_URI_SOAP_ENC);
    Vector params = new Vector();
    params.addElement(new Parameter("PONumber", String.class, PONumber, null));
    call.setParams(params);

    Response response = call.invoke(endpointURL, "");

    if (!response.generatedFault())
    {
      Parameter result = response.getReturnValue();
```

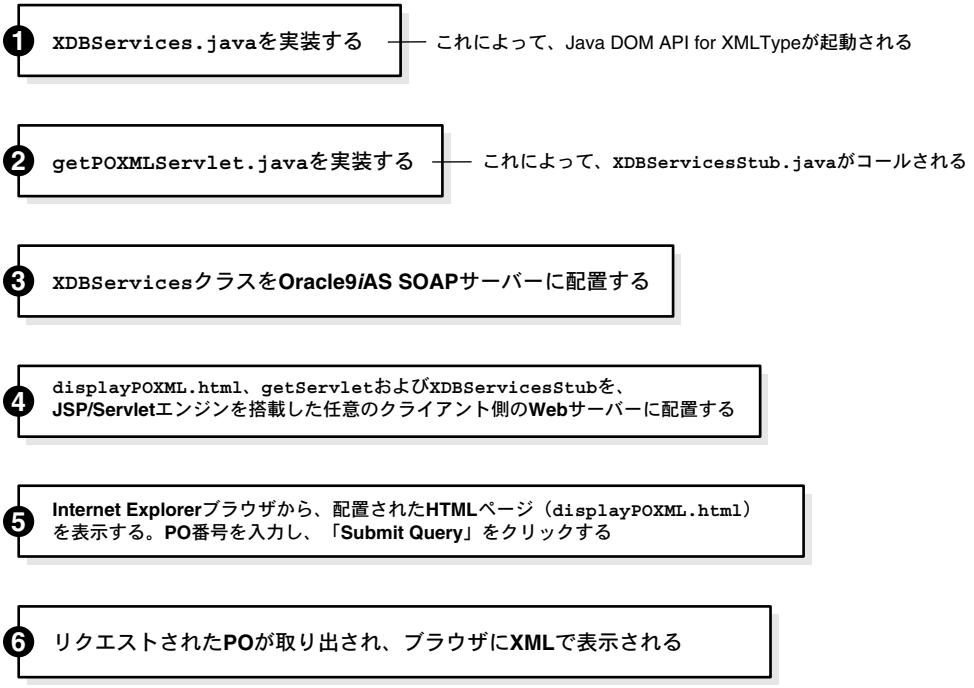
```
    returnVal = (String)result.getValue();
  }
  else
  {
    Fault fault = response.getFault();
    throw new SOAPException(fault.getFaultCode(), fault.getFaultString());
  }

  return returnVal;
}

public String getPOXML(String PONumber) throws Exception
{
  String returnVal = null;

  URL endpointURL = new URL(endpoint);
  Call call = new Call();
  call.setSOAPTransport(m_httpConnection);
  call.setTargetObjectURI("urn:POFetcher");
  call.setMethodName("getPOXML");
  call.setEncodingStyleURI(Constants.NS_URI_SOAP_ENC);
  Vector params = new Vector();
  params.addElement(new Parameter("PONumber", String.class, PONumber, null));
  call.setParams(params);

  Response response = call.invoke(endpointURL, "");

  if (!response.generatedFault())
  {
    Parameter result = response.getReturnValue();
    returnVal = (String)result.getValue();
  }
  else
  {
    Fault fault = response.getFault();
    throw new SOAPException(fault.getFaultCode(), fault.getFaultString());
  }

  return returnVal;
}

public void setMaintainSession(boolean maintainSession)
{
  m_httpConnection.setMaintainSession(maintainSession);
}

public boolean getMaintainSession()
```

```
  {
    return m_httpConnection.getMaintainSession();
  }

  public void setTransportProperties(Properties props)
  {
    m_httpConnection.setProperties(props);
  }

  public Properties getTransportProperties()
  {
    return m_httpConnection.getProperties();
  }
}
```

XML DB Web サービスの事例の実装手順

図 25-4 に、このアプリケーションの実装手順を示します。

図 25-4 Oracle XML DB Web サービスの事例 : 実装手順



1. XDBServices.java の実行

XDBServices.java は、Java DOM API for XMLType を使用して XML DB にアクセスします。この JavaBean は、次のタスクも実行します。

- XMLType 問合せ文字列のフォーマット
- JDBC Thick ドライバを使用した XML DB への接続
- XMLType インスタンスの取出し
- リクエストされた XML PO 文書の Java 文字列としての取出し

XDBServices.java は、次の操作を実装します。

1. **Java DOM API for XMLType のインポート**

```
import oracle.xdb.XMLType;
```

Java DOM API for XMLType は、すべての妥当な XML 文書を処理します。XML 文書は、XML Schema に基づくか基づかないか（Oracle XML DB の基礎となる記憶域の形式が何であるか）にかかわらず、アプリケーションで同様に表示されます。Java DOM API は、クライアントおよびサーバーで動作します。

2. **結果の問合せ文字列のフォーマット**

```
static String qryXMLStr = "select value(x) from purchaseorder x where
existsNode(value(x),'/PurchaseOrder[Reference=\"PO_NUMBER\"]') = 1 ";

 public String getPOXML(String PONumber) throws Exception
  {
    String res = null;
    XMLType xt = null;
    XMLType xt1 = null;
```

3. **JDBC Thick ドライバを使用した XML DB への接続**

```
System.out.println("Driver registering...");
    DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
    System.out.println("Driver registered");

    System.out.println("Connecting...");
    Connection conn = DriverManager.getConnection(conStr, user, pass);
    Hashtable map = (Hashtable) ((OracleConnection)conn).getTypeMap ();
    map.put ("SYS.XMLTYPE", Class.forName ("oracle.xdb.XMLTypeFactory"));
    System.out.println("Connection obtained");
```

4. **XMLType インスタンスの取出し**

```
//retrieve PurchaseOrder xml documnet from database
      System.out.println("Generating XMLType object ...");
      xt = (XMLType)orset.getObject(1);
      res = xt.getStringVal();
      System.out.println("Print out xt as String ...");
      System.out.println(res);
      System.out.println("##Results printed");
    }
```

5. **XML PO 文書の Java 文字列としての取出し**

```
    return res;
```

参照： 完全な XDBServices.java については、25-17 ページの「XDBServices.java」を参照してください。

2. GetPOXMLServlet.java の実装

GetPOXMLServlet.java は、次のタスクを実行します。

- ブラウザで入力したユーザーの PO 番号の受入れ
- XDBServiceStub.java のコール
- XDBServices.java を介した Web サービスの起動による PO のフェッチ
- ブラウザで表示する PO の送信

参照： 完全な getPOXMLServlet.java については、25-20 ページの「getPOXMLServlet.java」を参照してください。

3. Oracle9*i*AS および Web サービス（SOAP）・サーバーへの XDBServices クラスの配置

このアプリケーションを実行するには、XDBServicesDeploymentDescriptor.dd を実際に配置する必要があります。

XDBServicesDeploymentDescriptor.dd

```
<?xml version="1.0" ?>
- <!-- Generated by the Oracle9i JDeveloper Web Services Deployment Descriptor
Generator -->
- <!-- This Deployment Descriptor file is for use with the Oracle9iAS Release 2 /
Apache 2.2 SOAP Server SOAP Server -->
- <!-- Date Created: Mon Jul 15 16:34:26 PDT 2002-->
   - <isd:service id="urn:POFetcher"
         xmlns:isd="http://xml.apache.org/xml-soap/deployment">
     - <isd:provider type="java" methods="getPOFromNumber getPOXML"
         scope="Request">
          <isd:java class="XDBServices" static="false" />
       </isd:provider>
<isd:faultListener>org.apache.soap.server.DOMFaultListener</isd:faultListener>
</isd:service>
```

参照： Web サービスの配置方法の詳細は、『Oracle9*i* Java Developer's Guide』を参照してください。

Web サービス : POFetcher サービスの配置済サービス情報

表 25-1 に、Web サービスの POFetcher プロパティを示します。アプリケーションは、POFetcher のメソッドの 1 つ getPOXML を使用します。このメソッドは、XDBServices クラスを使用して Oracle XML DB にアクセスします。

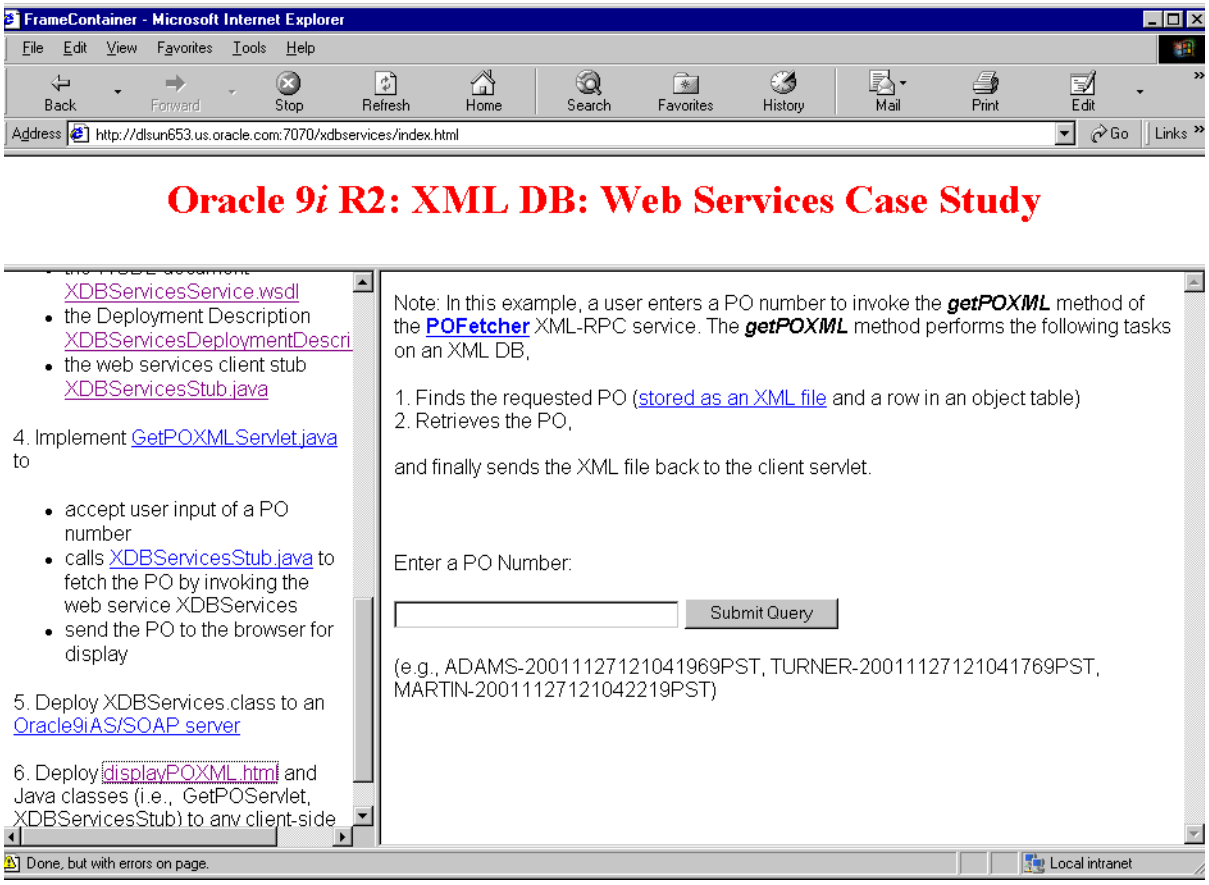
表 25-1 urn:POFetcher サービス・デプロイメント・ディスクリプタ

プロパティ	詳細
ID	urn:POFetcher
Scope	Application
Provider Type	java
Provider Class	XDBServices
Use Static Class	FALSE
Methods	getPOFromNumber、getPOXML

4. displayPOXML.html の配置によるクライアント側 Web サーバーでの結果の表示

`displayPOXML.html` および Java クラス `GetPOXMLServlet` と `XDBServicesStub` を、JSP/Servlet エンジン搭載の任意のクライアント側 Web サーバーに配置します。図 25-5 を参照してください。

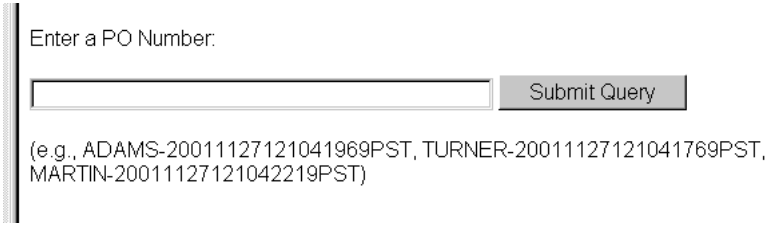
図 25-5 displayPOXML.html の配置結果



5. PO 番号の入力および取り出された PO の表示確認

Internet Explorer ブラウザ・ウィンドウで PO 番号を入力し、「Submit Query」をクリックします。図 25-6 に、PO 番号のエントリ・フィールドを示します。

図 25-6 発注書（PO）番号の入力

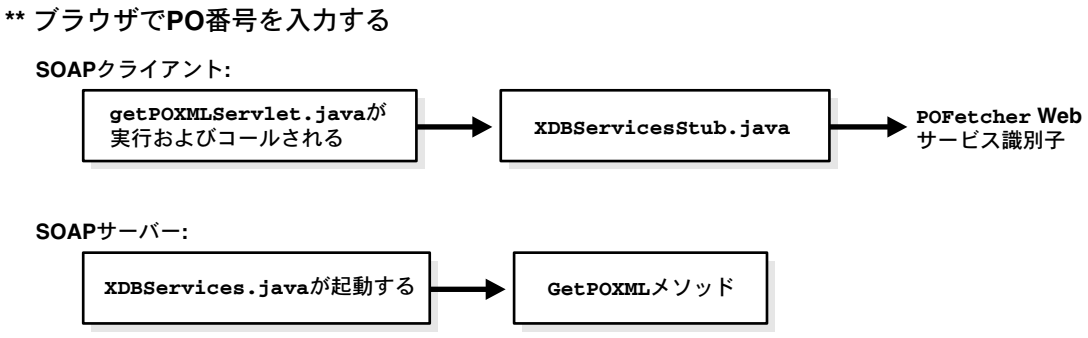


リクエストした PO が `POFetcher` Web サービスを介して取り出され、Internet Explorer ブラウザに XML で表示されます。

XML DB Web サービス : コール順序

図 25-7 に、XML DB Web サービス・アプリケーションのコール順序を示します。

図 25-7 XML DB Web サービス : コール順序



XDBServices.java

この項では、XDBServices.java の完全なコードを示します。コール順序については、前述の項を参照してください。

```
/*
   XDBServices.java:

   DESCRIPTION:
    The server Java bean that provides the XML DB web services to fetch
    a user-specified PO.

   MODIFIED     (MM/DD/YYYY)
    Geoff Lee    07/15/2002 - Cleaned up for viewlet recording

 */

import java.sql.*;
import java.io.*;
import java.util.Hashtable;

import oracle.jdbc.driver.*;
import oracle.sql.*;

import oracle.xdb.XMLType;

public class XDBServices
{
  static String conStr = "jdbc:oracle:oci8:@";
  static String user = "scott";
  static String pass = "tiger";

  static String qryXMLStr = "select value(x) from purchaseorder x where
existsNode(value(x),'/PurchaseOrder[Reference=\"PO_NUMBER\"]') = 1 ";

  static String qryStr = "select
x.transform(xdburitype('/public/SCOTT/xsl/po.xsl').getXML()) from purchaseorder x
where existsNode(value(x),'/PurchaseOrder[Reference=\"PO_NUMBER\"]') = 1 ";

 public String getPOFromNumber (String PONumber) throws Exception
  {
    String res = null;
    XMLType xt = null;
    XMLType xt1 = null;

    try{
```

```
        // Replace the PO_NUMBER placeholder with the input
        int po_start = qryStr.indexOf ("PO_NUMBER");
        String poQryStr = new StringBuffer(qryStr).replace(po_start, po_start + 9,
PONumber).toString();
        System.out.println("poQryStr="+ poQryStr);

        System.out.println("Driver registering...");
        DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
        System.out.println("Driver registered");

        System.out.println("Connecting...");
        Connection conn = DriverManager.getConnection(conStr, user, pass);
        Hashtable map = (Hashtable) ((OracleConnection)conn).getTypeMap ();
        map.put ("SYS.XMLTYPE", Class.forName ("oracle.xdb.XMLTypeFactory"));
        System.out.println("Connection obtained");

        System.out.println("Statement preparing...");
        OraclePreparedStatement stmt  =
(OraclePreparedStatement)conn.prepareStatement (poQryStr);
        System.out.println("Statement prepared");

        System.out.println("Query executing...");
        ResultSet rset = stmt.executeQuery(poQryStr);
        System.out.println("Query executed");
        OracleResultSet orset = (OracleResultSet) rset;
        System.out.println("ResultSet casted");

        while (orset.next())
        {
          //retrieve PurchaseOrder xml documnet from database
          System.out.println("Generating XMLType object ...");
          xt = (XMLType)orset.getObject(1);
          res = xt.getStringVal();

          System.out.println("Print out xt as String ...");
          System.out.println(res);
          System.out.println("##Results printed");
        }

      //close the result set, statement, and the connection
      rset.close();
      stmt.close();
      conn.close();


    }
    catch( Exception e )
```

```
    {
      e.printStackTrace(System.out);
    }

          return res;

  }
 //
 // This method returns the PO in XML format
 //
 public String getPOXML (String PONumber) throws Exception
  {
    String res = null;
    XMLType xt = null;
    XMLType xt1 = null;

    try{
        // Replace the PO_NUMBER placeholder with the input
        int po_start = qryXMLStr.indexOf ("PO_NUMBER");
        String poQryStr = new StringBuffer(qryXMLStr).replace(po_start, po_start +
9, PONumber).toString();

        System.out.println("poQryStr="+ poQryStr);

        System.out.println("Driver registering...");
        DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
        System.out.println("Driver registered");

        System.out.println("Connecting...");
        Connection conn = DriverManager.getConnection(conStr, user, pass);
        Hashtable map = (Hashtable) ((OracleConnection)conn).getTypeMap ();
        map.put ("SYS.XMLTYPE", Class.forName ("oracle.xdb.XMLTypeFactory"));
        System.out.println("Connection obtained");

        System.out.println("Statement preparing...");
        OraclePreparedStatement stmt  =
(OraclePreparedStatement)conn.prepareStatement (poQryStr);
        System.out.println("Statement prepared");

        System.out.println("Query executing...");
        ResultSet rset = stmt.executeQuery(poQryStr);
        System.out.println("Query executed");
        OracleResultSet orset = (OracleResultSet) rset;
        System.out.println("ResultSet casted");

        while (orset.next())
        {
```

```
            //retrieve PurchaseOrder xml documnet from database
            System.out.println("Generating XMLType object ...");
            xt = (XMLType)orset.getObject(1);
            res = xt.getStringVal();

            System.out.println("Print out xt as String ...");
            System.out.println(res);
            System.out.println("##Results printed");
          }
        //close the result set, statement, and the connection
        rset.close();
        stmt.close();
        conn.close();

      }
      catch( Exception e )
      {
        e.printStackTrace(System.out);
      }

            return res;
    }
}
```

getPOXMLServlet.java

この項では、完全な getPOXMLServlet.java を示します。

```
/*
   GetPOServlet.java:

   DESCRIPTION:
    The Java servlet that
    1. Accepts a user-specified PO number
    2. Calls the client Java bean of the web service
    3. Display the PO fetched from the XML DB by the 'POFetched' web service

 */

import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;
import XDBServicesStub;

public class GetPOServlet extends HttpServlet
{
```

```
  private static final String CONTENT_TYPE = "text/html; charset=windows-1252";
  public void init(ServletConfig config) throws ServletException
  {
    super.init(config);
  }

  public void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException
  {
    String PONumber = "";
    String poResult = "";
    try
    {
      PONumber = request.getParameter("PONumberSelect");
      XDBServicesStub poBean = new XDBServicesStub();
      poResult = poBean.GetPO(PONumber);
    }
    catch(Exception e)
    {
      e.printStackTrace();
    }

    response.setContentType(CONTENT_TYPE);
    PrintWriter out = response.getWriter();
    out.println(poResult);
    out.close();
  }

  public void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException
  {
    String PONumber = "";
    String poResult = "";
    try
    {
      PONumber = request.getParameter("PONumberSelect");
      XDBServicesStub poBean = new XDBServicesStub();
      poResult = poBean.GetPO(PONumber);
    }
    catch(Exception e)
    {
      e.printStackTrace();
    }

    response.setContentType(CONTENT_TYPE);
    PrintWriter out = response.getWriter();
    out.println(poResult);
```

```
      out.close();
    }
  }
```

26

Oracle XML DB Basic Demo

この章では、Oracle XML DB Basic Demo のインストール方法および使用方法を説明します。この章の内容は次のとおりです。

XML DB Basic Demo を実行するための前提条件

この XML DB のデモは、http://otn.oracle.com/tech/xml/content.html から表示および実行することもできます。

XML DB Basic Demo を実行する前に、次のソフトウェアがインストールされていることを確認します。

Oracle 以外のソフトウェア

XML DB Basic Demo をインストールおよび実行する前に、Oracle 以外の次のソフトウェアをインストールすることをお薦めします。

- XMLSpy: XMLSpy は、Altova 社の XML Schema エディタです。この製品のライセンスを持っていない場合は、http://www.altova.com から評価コピーをダウンロードできます。
- WS_FTP: WS_FTP は、Ipswitch Software 社のグラフィカル FTP クライアントです。デモは、この製品の LE バージョンに基づいています。これは、http://www.ftpplanet.com/download.htm からダウンロードできます。
- Microsoft cscript インタプリタ バージョン 5.6 以上 : DOS コマンド・プロンプトで cscript コマンドを入力すると、マシンにインストールされている cscript のバージョンを確認できます。cscript プロセッサは、デモ中に使用するショート・カットを作成し、ファイルをインストール・ディレクトリ構造からデモ・ディレクトリ構造にコピーするために使用されます。このソフトウェアは、http://msdn.microsoft.com/downloads/default.asp?URL=/downloads/sample.asp?url=/msdn-files/027/001/733/msdncompositedoc.xml からダウンロードできます。
- Microsoft DOM コントロールおよび VBScript 用の XMLParser: このソフトウェアは、インストール・スクリプトによって、構成ファイルを処理するために使用されます。現在、このソフトウェアの最新バージョンは、http://msdn.microsoft.com/downloads/default.asp?url=/downloads/sample.asp?url=/msdn-files/027/001/766/msdncompositedoc.xml からダウンロードできます。
- 最新のサービス・パックを含む Microsoft Internet Explorer 6.0（推奨）: このデモには、Netscape のいずれのバージョンも使用できません。Internet Explorer 5.5 には多くの重大なページ・キャッシュ問題があるため、このデモの一部のセクションが予測どおりに機能しません。
- Microsoft Office 2000 または Microsoft Office XP: このデモでは、Microsoft Word が使用されます。メモ帳およびワードパッドでは WebDAV を使用できないため、それらを Microsoft Word のかわりに使用することはできません。

Oracle ソフトウェア

XML DB Basic Demo をインストールおよび実行する前に、次の Oracle ソフトウェアをインストールすることをお薦めします。

- Oracle クライアント（Oracle9*i* リリース 2（9.2.0.1）以上の SQL*Plus および Oracle Net Services）: このデモを Oracle9*i* データベース リリース 2（9.2.0.2）で実行するには、Oracle9*i* リリース 2（9.2.0.2）の SQL*Plus クライアントがインストールされている必要があります。このデモはリモート・データベースに対して実行できますが、SQL*Plus および Oracle Net Services がそのクライアント・マシンにインストールされている必要があります。

Oracle Net Services および XML DB の構成

インストールを開始する前に、次の手順を実行して、Oracle Net Services、FTP および HTTP が正しく構成されていることを確認します。

1. Windows の**コマンド・プロンプト**・セッションを開き、`basicDemo` ディレクトリに移動し、SQL*Plus を使用してターゲット・データベースに「SYS」として接続します。

   注意： `ORCL92` のかわりに、適切な `TNSAlias` を指定する必要があります。

   ```
   c:¥...¥BasicDemo>sqlplus "sys@ORCL92 as sysdba"
   ```

2. SCOTT スキーマが作成済であり、EMP 表および DEPT 表が存在することを確認します。SCOTT スキーマが現在ロードされていない場合は、次のコマンドを実行して SCOTT スキーマを作成できます。

   ```
   SQL> @?¥rdbms¥admin¥utlsampl.sql
   ```

3. 次のコマンドを実行して、Oracle XML DB がインストールされていることを確認します。

   ```
   SQL> set long 100000

   SQL> set pagesize 0
   SQL> select XDBUritype('/xdbconfig.xml').getXML()
     2  from dual
     3  /
   ```

   XML DB が正常にインストールされている場合は、XML DB 構成ドキュメントが表示されます。

4. 次のスクリプトを実行して、XDB_UTILITY パッケージをインストールします。

```
C:¥...¥basicDemo>sqlplus "sys@ORCL92 as sysdba"

SQL*Plus: Release 9.2.0.1.0 - Production on Fri Aug 16 12:09:42 2002

Copyright (c) 1982, 2002, Oracle Corporation.  All rights reserved.

Enter password:
Connected to:
Oracle9i Enterprise Edition Release 9.2.0.1.0 - Production With the
Partitioning, OLAP and Oracle Data Mining options
JServer Release 9.2.0.1.0 - Production

SQL> @SQL/xdbUtility

View created.

PL/SQL procedure successfully completed.

Package created.
No errors.

Package body created.
No errors.

Synonym created.
Grant succeeded.

SQL>
```

5. 次のコマンドを実行して、XDB_PORTS パッケージの実行権限をターゲット・ユーザーに付与します。

```
SQL> grant execute on XDB_UTILITY to SCOTT
    2  /
```

6. XDB_DATABASE_SUMMARY ビューに次の SELECT 文を発行して、システムの現在の FTP および HTTP ポート設定を確認します。

```
SQL> set long 10000
SQL> select value(x) from XDB_DATABASE_SUMMARY (x)
  2  /
```

これによって、次の出力が生成されます。

```
VALUE(X)
 ----------------------------------------------------------------
<Database Name="ORCL92" HTTP="8080" FTP="2100">
   <Services>
     <ServiceName>ORCL92.xp.mark.drake.oracle.com</ServiceName>
   </Services>
   <Hosts>
     <HostName>MDRAKE-LAP</HostName>
   </Hosts>
</Database>
```

注意： 前述の例は、XML DB のインストール時に設定されたデフォルトの XML DB ポート番号を示しています。これらのポート番号は、ご使用の環境での値と一致しない場合があります。

---

7. 前述の例に示すポート番号が必要なポートでない場合、次のプロシージャを使用してポートを再構成します。

   - FTP の場合

     ```
     SQL> call XDB_UTILITY.SET_FTP_PORT(nnnn);
     ```

   - HTTP の場合

     ```
     SQL> call XDB_UTILITY.SET_HTTP_PORT(nnnn);
     ```

   前述の例に示す nnnn は、ターゲット・ポート番号を表します。FTP および HTTP ポート番号の選択には、次の制限事項があります。

   - 同じ値は選択できません。
   - システム上の他のサービスが使用するポート番号は選択できません。

   通常、専用ポート番号（0 ～ 1023）以外のポート番号を選択する必要があります。ポート番号のリセット後、手順 4 を繰り返して、新しい番号が受け入れられたことを確認します。

8. 同じホスト上で実行している他のデータベース・インスタンスとの間に HTTP または FTP ポート競合がないことを確認します。これを行うには、次のコマンドを実行してデータベース・リスナーの状態を確認します。

   ```
   C:¥TEMP>lsnrctl status
   ```

デモをリモート・データベースに対して実行している場合は、そのリモート・マシンに接続している DOS **コマンド・プロンプト**・セッションまたは Telnet セッションから状態を確認する必要があります。

9. status コマンドの出力を調べます。リスナーが手順 4 で識別されたポート番号で HTTP および FTP リクエストを監視していることを確認します。status コマンドを実行すると、次のような出力が生成されます。

```
LSNRCTL for 32-bit Windows: Version 9.2.0.1.0 - Production on 05-AUG-2002
16:01:37
Copyright (c) 1991, 2002, Oracle Corporation.  All rights reserved.
Connecting to (DESCRIPTION=(ADDRESS=(PROTOCOL=IPC)(KEY=EXTPROC0)))
STATUS of the LISTENER
------------------------
Alias                     LISTENER
Version                   TNSLSNR for 32-bit Windows: Version 9.2.0.1.0 -
Production
Start Date                03-AUG-2002 21:45:08
Uptime                    1 days 18 hr. 16 min. 28 sec
Trace Level               off
Security                  OFF
SNMP                      OFF
Listener Parameter File   C:¥oracle¥ora92¥network¥admin¥listener.ora
Listener Log File         C:¥oracle¥ora92¥network¥log¥listener.log
Listening Endpoints Summary...
        (DESCRIPTION=(ADDRESS=(PROTOCOL=ipc)(PIPENAME=¥¥.¥pipe¥EXTPROC0ipc)))
        (DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=mdrake-lap)(PORT=1521)))
        (DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=mdrake-lap)(PORT=8080))
                  (Presentation=HTTP)(Session=RAW))
        (DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=mdrake-lap)(PORT=2100))
                  (Presentation=FTP)(Session=RAW))
Services Summary...
Service "ORCL92.xp.mark.drake.oracle.com" has 2 instance(s).
Instance "ORCL92", status UNKNOWN, has 1 handler(s) for this service...
Instance "ORCL92", status READY, has 2 handler(s) for this service...
Service "ORCL92XDB.xp.mark.drake.oracle.com" has 1 instance(s).
Instance "ORCL92", status READY, has 1 handler(s) for this service...
Service "PLSExtProc" has 1 instance(s).
Instance "PLSExtProc", status UNKNOWN, has 1 handler(s) for this service...
The command completed successfully
```

status コマンドの出力に HTTP および FTP 表示の複数のエントリが示される場合、複数回出現するポート番号がないことを確認します。特定のポート番号が複数回出現する場合は、2 つ以上のデータベース・インスタンスがそのポートにサービスの提供を試行していることを意味します。これは許可されません。任意のホスト上で実行しているすべてのデータベース・インスタンスには、一意の FTP および HTTP ポート番号を割り当てる必要があります。

10. 複数のデータベース・インスタンスが同じポートにサービスを提供するように構成されている場合、各インスタンスに一意のポート番号が割り当てられていることを確認します。

    各インスタンスに順に接続し、手順 4 ～ 7 を繰り返して、FTP および HTTP ポートに対する適切な値を指定します。ポート番号を 0（ゼロ）に設定すると、データベースによる FTP および HTTP リクエストの処理を停止できます。すべてのデータベース・インスタンスを再構成した後、リスナーの status コマンドを使用して、各データベース・インスタンスが一意の FTP および HTTP ポート番号にサービスを提供するように再構成されていることを確認します。

Oracle Net Services および XML DB の構成の確認

次のプロシージャを使用して、Oracle Net Services（以前の Net8）、FTP および HTTP プロトコルが予測どおりに構成されていることを確認します。この例では、TNSALIAS ORCL92 を使用してターゲット・データベースへの接続を確立できると想定しています。

1. 次のコマンドを実行して FTP を介してターゲット・データベースに接続し、FTP 構成を確認します。

```
C:¥temp >ftp -n
ftp> open localhost 2100
Connected to mdrake-lap.
220 mdrake-lap FTP Server (Oracle XML DB/Oracle9i Enterprise Edition Release
9.2.0.1.0 - Production) ready.
ftp> user scott tiger
331 pass required for SCOTT
230 SCOTT logged in
```

   FTP サーバーに接続する場合は、次のとおり置き換えます。

   - localhost を、データベース・インスタンスをホストするサーバーの名前に置き換えます。
   - 2100 を、ターゲット FTP ポートの値に置き換えます。

2. 次のコマンドを実行して、databaseSummary.xml ファイルのコンテンツを取り出します。

```
ftp> get /sys/databaseSummary.xml

200 PORT Command successful
150 ASCII Data Connection
<Database Name="ORCL92" HTTP="8080"
FTP="2100"><Services><ServiceName>ORCL92.xp.mark.drake.oracle.com</ServiceName><
/Services><Hosts><HostName>MDRAKE-LAP</HostName></Hosts></Database>226
ASCII Transfer Complete
ftp: 183 bytes received in 0.01Seconds 18.30Kbytes/sec.
ftp>
```

3. Internet Explorer を起動し、HTTP タグに含まれている URL をアドレス・バーに入力して、HTTP 構成を確認します。次に例を示します。

```
http://MDRAKE-LAP:8080/sys/databaseSummary.xml
```

ブラウザで、ユーザー名およびパスワードを入力するためのプロンプトが表示されます。XML DB Basic Demo の実行時に使用するデータベース・ユーザーの名前およびパスワードを入力します。HTTP 構成が正しい場合、次のとおり、ブラウザにファイルのコンテンツが表示されます。

```
- <Database Name="ORCL92" HTTP="8080" FTP="2100">
- <Services>
    <ServiceName>ORCL92.xp.mark.drake.oracle.com</ServiceName>
  </Services>
- <Hosts>
    <HostName>MDRAKE-LAP</HostName>
  </Hosts>
  </Database>
```

XML DB Basic Demo のインストール

XML DB Basic Demo をインストールするには、XDBBasicDemo.zip ファイルを任意のフォルダに解凍します。このインストール・ファイルを解凍すると、basicDemo/ フォルダが作成されます。このフォルダにはサブフォルダ install/ が含まれており、このサブフォルダには install.vbs ファイルが含まれています。この install.vbs によって、XML DB Basic Demo がインストールされます。

basicDemo/ ディレクトリで DOS **コマンド・プロンプト**・セッションを開きます。XML DB デモをインストールするには、次の情報が必要です。

- クライアント側の ORACLE_HOME の正しいパス
- ターゲット・データベースへの Oracle Net Services 接続を確立するために使用可能な TNSALIAS の名前
- ターゲット・データベースへの HTTP および FTP 接続を確立するために必要なホスト名およびポート番号
- ターゲット・データベースに対する CONNECT 権限および RESOURCE 権限を付与されたユーザーのユーザー名およびパスワード
- インストールされた WS_FTP の正しいパス
- インストールされた Microsoft Word の正しいパス

installParameters.xml の編集

install/ フォルダには、インストール・プロセスを制御する installParameters.xml ファイルも含まれています。このファイルには、インストールを調整するための引数が含まれています。

1. テキスト・エディタまたは XML エディタを使用して、installParameters.xml を編集します。このファイルは単純であるため、メモ帳を使用して簡単に編集できます。このファイルの形式は次のとおりです。

```
<demoConfig>
      <oracleHome>c:¥oracle¥ora92</oracleHome>
      <oracleUser>SCOTT</oracleUser>
      <oraclePassword>TIGER</oraclePassword>
      <oracleSID>ORCL92</oracleSID>
      <sqlPort>1521</sqlPort>
      <listenerName>LISTENER</listenerName>
      <hostName>localhost</hostName>
      <httpPort>8080</httpPort>
      <ftpPort>2100</ftpPort>
      <msWordPath>
            c:¥Program Files¥Microsoft Office¥Office¥WINWORD.EXE
     </msWordPath>
      <ftpPath>c:¥Program Files¥WS_FTP¥WS_FTP95.exe</ftpPath>
      <shortCutFolderName>XML DB Basic Demo</shortCutFolderName>
</demoConfig>
```

2. 必要な変更を行います。特に、次の値に注意してください。
   - <oracleHome>
   - <oracleSID>
   - <httpPort>
   - <ftpPort>
3. ファイルを保存します。installationParameters.xml ファイルの編集後、Internet Explorer で開いて、ファイルに整形式の XML が含まれていることを確認します。

インストール・スクリプトの実行

XML DB Basic Demo のインストール・スクリプトを実行するには、次の手順を実行します。

1. `install.vbs` スクリプトをダブルクリックして、インストール・スクリプトを実行します。このスクリプトを実行すると、構成プロセスの開始前に、確認のためのプロンプトが表示されます（図 26-1 を参照）。

図 26-1 確認のためのプロンプト

2. 「OK」をクリックして、デモを構成します。

   インストール・プロセスの終わりに、インストールが完了したことが通知されます。

図 26-2 インストールの完了

3. 「OK」をクリックして、デモのインストールを完了します。

注意： installParameters.xml を使用して指定した値が正しくないことが判明した場合、installParameters.xml ファイルを編集し、install.vbs スクリプトを再実行すると、それらの値を修正できます。

Oracle XML DB の概要

Oracle XML DB は、XML の高パフォーマンスな格納および取出しを実現する、Oracle9*i* データベース リリース 2（9.2）の一連の機能を示す用語です。これらの機能によって、W3C の XML データ・モデルをデータベースに取り込むことが可能になります。このテクノロジが導入された Oracle9*i* データベース リリース 2（9.2）は、最も完成されたリレーショナル・データベースであるだけでなく、ネイティブな XML データベースでもあります。

XML をナビゲートおよび問い合せるための新しい方法

Oracle XML DB は、組織に、XML データを格納および管理するための、記憶域、コンテンツおよびプログラミング言語に依存しないインフラストラクチャを提供します。また、Oracle XML DB は、データベース内に格納された XML コンテンツをナビゲートおよび問い合せる新しい方法も提供します。Oracle XML DB では、リレーショナル・データベース・テクノロジおよび XML テクノロジのすべてのメリットが同時に得られます。

XML をデータベースに格納するためのオプション

Oracle XML DB は、データベースへの XML 文書の格納方法を管理するための様々なオプションを提供します。オプションには次のものが含まれます。

- 非構造化記憶域 : 文書は単に CLOB として格納されます。
- 構造化記憶域 : XML 文書は一連のオブジェクトに断片化されます。

XPath アクセス方法の完全なサポート

XML で作業を行うと、XML が階層隠喩と密接に関係していることがすぐにわかります。XML 文書に含まれているコンテンツの問合せまたはアクセスに使用される標準メカニズムは、XPath です。XPath は、XML 文書の一部を指定するための言語を定義する W3C 標準です。XPath は、パスベースの表記法を使用して、XML 文書の階層構造をナビゲートします。XML 文書で別の XML 文書を参照する必要がある場合、ターゲット・ドキュメントを参照する標準の方法は URL を使用することです。XPath と同様に、URL はパスベースの表記法を使用して対象のドキュメントを識別します。Oracle XML DB は、これらのアクセス方法を完全にサポートしています。

XPath 式を使用した XML の問合せおよび更新

Oracle XML DB が提供する様々な機能によって、XPath 式を使用して XML 文書のコンテンツの問合せおよび更新を行うことが可能になります。また、Oracle XML DB には、単純で軽量なリポジトリも含まれています。このリポジトリによって、URL を使用して XML 文書間の関係を表すことが可能になります。また、URL を使用して XML コンテンツにアクセスすることも可能になります。これは、リレーショナル・メカニズムと階層メカニズムの両方を使用して XML オブジェクトにアクセスできることを意味します。

Oracle XML DB のコンポーネント

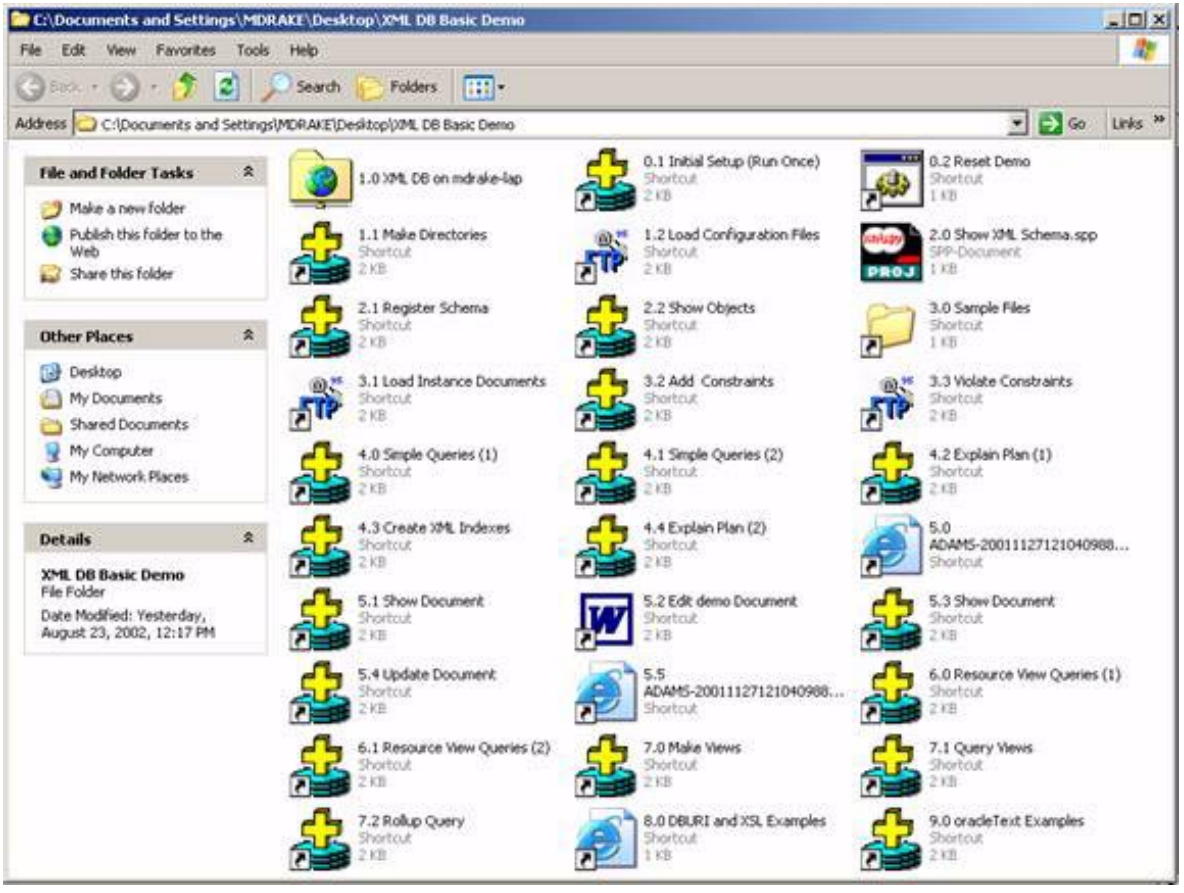
Oracle XML DB の主なコンポーネントは次のとおりです。

- **XMLType:** このネイティブなサーバーのデータ型を使用すると、データベースは、DATE データ型によって列に日付が含まれていることを認識できるように、列または表に XML が含まれていることを認識できます。また、`XMLType` は、XML Schema 検証や XSLT 変換などの一般的な操作を XML コンテンツに対して実行できる方法も提供します。
- **XMLSchema:** Oracle XML DB は、W3C の XML Schema 勧告を完全にサポートしています。一度 XML Schema を Oracle XML DB に登録すると、その XML Schema に対してあらゆるインスタンス・ドキュメントを検証できます。また、XML Schema は、それに準拠するインスタンス・ドキュメントをデータベースに格納する方法を定義するためにも使用されます。
- **XML DB リポジトリ :** Oracle XML DB Repository を使用すると、URL を使用して XML 文書間の関係を定義し、パスベースの隠喩を使用して文書のコンテンツにアクセスできます。これは、文書中心で XML コンテンツを表示するアプリケーションでは重要です。また、Oracle XML DB は、HTTP、FTP および WebDAV プロトコルのネイティブなサポートを追加して、Windows エクスプローラや Microsoft Office などの標準的なクライアントが Oracle XML DB に格納された XML コンテンツに直接アクセスできるようにします。リポジトリは、Internet Engineering Task Force（IETF）の WebDAV 標準に基づく基本バージョニングおよびアクセス制御もサポートしています。
- **SQL/XML:** Oracle XML DB では、将来の SQL/XML 標準に取り入れられる予定の多くの演算子も実装されます。これらの演算子は次の 2 つのカテゴリに分類されます。
  - 通常の SQL 操作の一部として XML コンテンツの問合せおよびアクセスを可能にする一連の演算子
  - SQL SELECT 文の結果から XML を生成するための業界標準の隠喩を提供する一連の演算子

XML DB Basic Demo の起動

XML DB Basic Demo を起動するには、XML DB Basic Demo フォルダを開きます。XML DB Basic Demo が正常にインストールされている場合、このフォルダはデスクトップ上に置かれます。図 26-3 に、BasicDemo/ フォルダを開いたときに表示されるアイコンを示します。

図 26-3 XML DB Basic Demo フォルダのアイコン



図に示すとおり、このフォルダ内のアイコンには番号が付いています。このフォルダ内の各アイコンを順にクリックすると、デモが実行されます。

0.1 XML DB Demo: 初期設定（1 回のみ実行）

初めてデモを実行する前に、このスクリプトを実行する必要があります。

このスクリプトは、データベースを削除して再作成しないかぎり、再度実行する必要はありません。このスクリプトを実行すると、次の操作が行われます。

- デモの他の部分で使用されるグローバル・データベース・オブジェクトが作成されます。
- 「/home」フォルダが作成されます。
- フォルダに適切な ACL が設定されます。
- ターゲット・ユーザーに必要なすべての権限が付与されていることが確認されます。

デモをユーザー SCOTT として実行すると、これらの操作の一部でエラーが発生する場合があります。これらのエラーは無視してもかまいません。

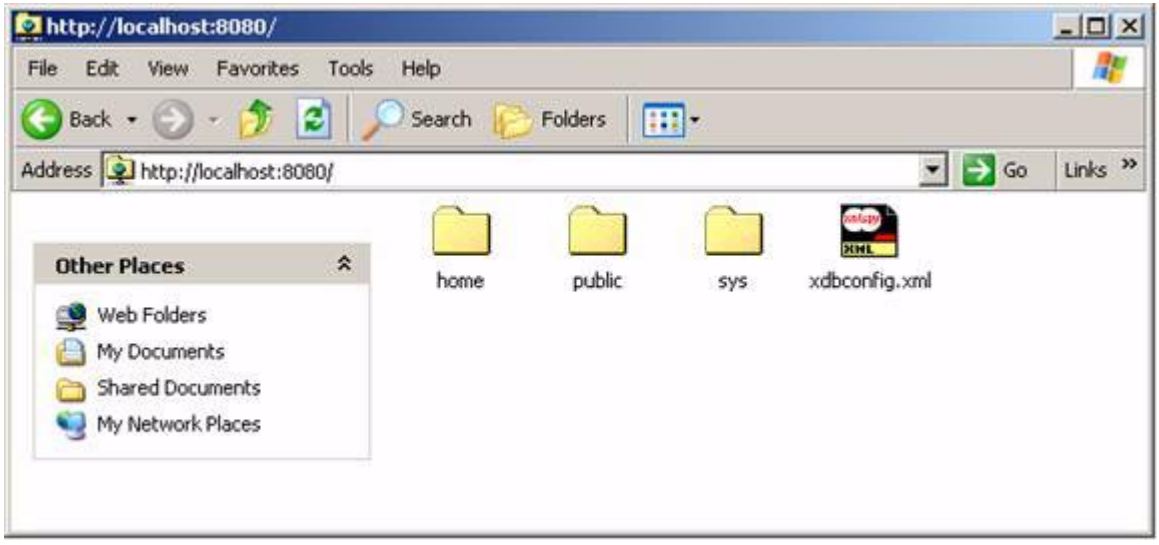
0.2 XML DB Demo: デモのリセット

デモを実行する前に、このスクリプトを実行する必要があります。このスクリプトを実行すると、選択したユーザーの home フォルダに含まれているすべてのファイルが削除され、環境がクリーンな状態であることが確認されます。`deleteSchema()` などの操作でエラーが発生する場合があります。これらのエラーは、無視しても問題ありません。

1.0 XML DB Demo: ローカル・ホスト上の XML DB - WebDAV および FTP のサポート

この手順では、Oracle XML DB の一部として含まれているネイティブな WebDAV のサポートが示されます。「**1.0 localhost**」アイコンをクリックして、XML DB リポジトリへの Web フォルダ（WebDAV）・セッションを開きます。ユーザー名およびパスワードを入力するためのプロンプトが表示されます。適切なデータベース・ユーザー名およびパスワードを入力し、「OK」をクリックします。図 26-4 に示すウィンドウが表示されます。

図 26-4 ローカル・ホスト上の XML DB



要点は次のとおりです。

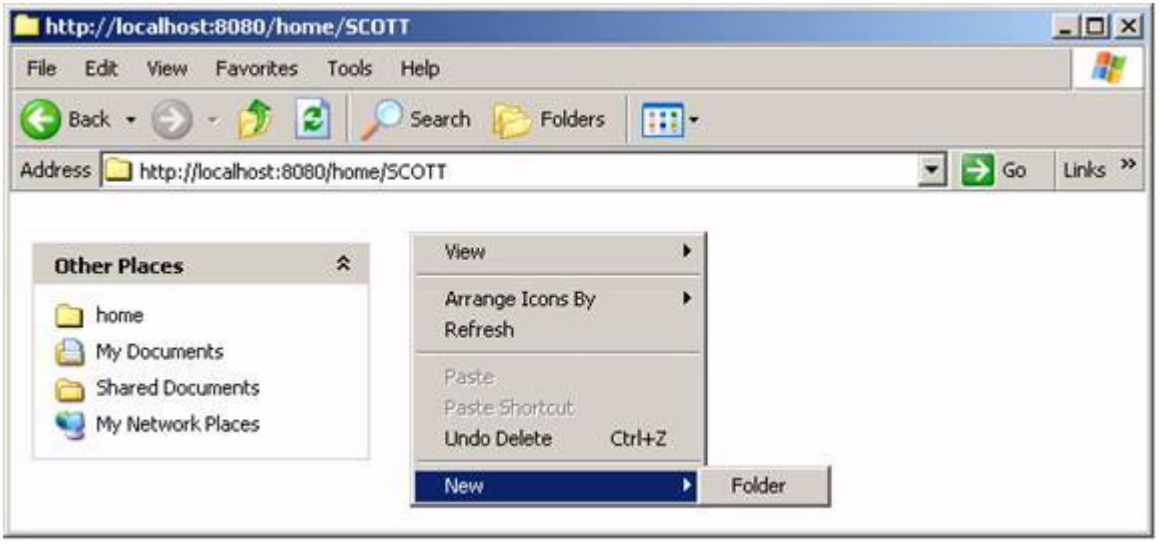
- WebDAV は、HTTP プロトコルの一連の拡張機能を定義する IETF 標準です。これらの機能によって、HTTP サーバーは DAV が使用可能なクライアントのファイル・サーバーとして機能できるようになります。
- Windows エクスプローラは、WebDAV プロトコルを使用して XML DB リポジトリに直接接続できます。これを実現するための Oracle または Microsoft の特別なソフトウェアをインストールする必要はありません。アドレス・パスに表示される場所は、HTTP ベースの URL です。Windows エクスプローラは、WebDAV サーバーとの接続を認識しているため、フォルダのコンテンツをファイル・システムとして表示します。
- Oracle XML DB の一部として WebDAV サポートを提供するメリットは、WebDAV プロトコルを認識する標準的なクライアントが、特別なアダプタまたはプラグイン・テクノロジを使用せずに Oracle XML DB 内の XML コンテンツにアクセスしたり、XML コンテンツを Oracle XML DB に格納できるようになることです。
- データベース側からの変更は必要ありません。データベースおよび TNS リスナーは、Oracle Net Services（以前の Net8）のサポートと同様に、FTP、HTTP および WebDAV をサポートします。リスナーは、Oracle Net Services（以前の Net8）リクエストの場合と同じ方法で FTP または HTTP リクエストを受信し、それらをそのリクエストを処理する共有サーバー・プロセスに渡します。

これを表示するには、サーバー・マシン上でコマンド・ウィンドウを開き、次のコマンドを発行します。

```
c:¥temp> lsnrclt status
```

home フォルダを開きます。デモ・ユーザーの名前が SCOTT である場合、このフォルダには SCOTT/ というフォルダが含まれます。SCOTT/ フォルダを開きます。SCOTT/ フォルダは空です。右クリックし、「New」->「Folder」と選択して、新しいフォルダを作成します。

図 26-5 新しいフォルダの作成



新しいフォルダに、識別しやすい名前を指定します。

要点は次のとおりです。

- エンド・ユーザーは、操作に慣れているツールおよびインタフェースを使用して、Oracle XML DB Repository で作業を行うことができます。

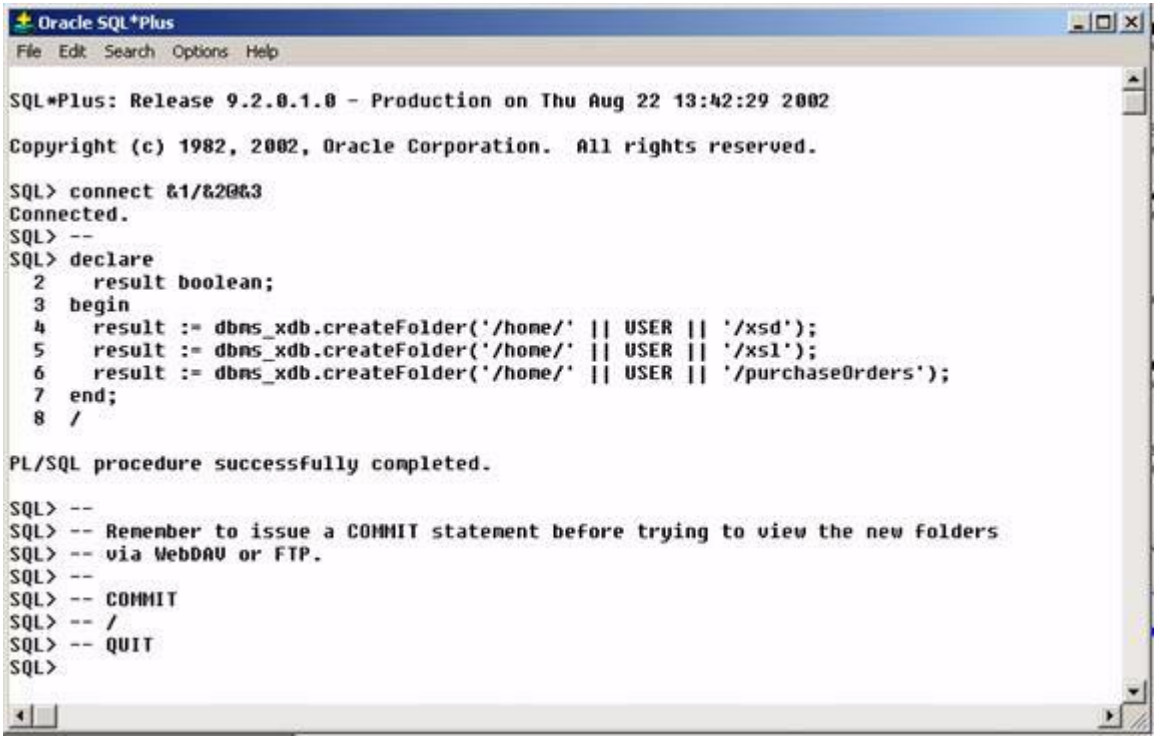
この時点では、このウィンドウを閉じないでください。

1.1 SQL を使用したディレクトリの作成

この手順では、プロトコル経由および SQL で Oracle XML DB Repository にアクセスおよび更新できることが示されます。また、SQL を使用してアクセスした場合はリポジトリ操作がトランザクション型であることも示されます。

1. 「**1.1 Make Directories**」アイコンをクリックして、SQL スクリプトを実行します。

図 26-6 ディレクトリの作成



このスクリプトを実行すると、/home/SCOTT フォルダ内に一連のフォルダが作成されます。

2. この時点では、この SQL*Plus セッションを閉じないでください。

   要点は次のとおりです。

   - Oracle XML DB Repository の SQL で、WebDAV や FTP などの標準プロトコルと同様に、リポジトリへのアクセスおよび操作を行えます。
   - PL/SQL パッケージ `DBMS_XDB` を使用すると、SQL でリポジトリを操作できます。これは、PL/SQL プロシージャをコールできるすべてのプログラムが、Oracle XML DB Repository で動作することを意味します。

3. `/home/SCOTT` の WebDAV 表示を含むウィンドウをクリックします。「Refresh」オプションをクリックします。PL/SQL スクリプトを実行して作成したフォルダは、表示を更新しても参照できないことに注意してください。

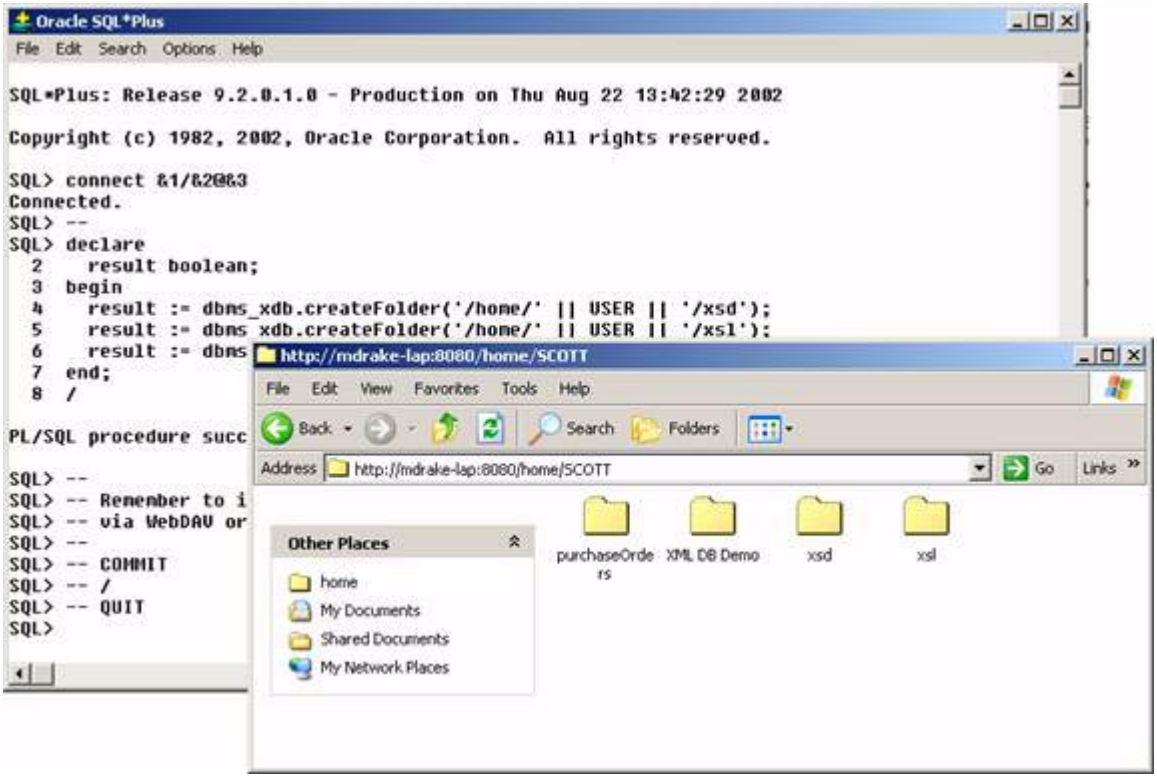
   要点は次のとおりです。

   - これは予測される動作です。PL/SQL 操作はトランザクション型であり、そのトランザクションはコミットされていません。したがって、他のユーザーは PL/SQL セッションで行われた変更を参照できません。

4. SQL*Plus セッションを含むウィンドウをクリックし、そのトランザクションをコミットします。SQL*Plus ウィンドウを閉じます。

5. `/home/SCOTT` の WebDAV 表示を含むウィンドウをクリックします。「Refresh」オプションをクリックします。これによって、PL/SQL スクリプトを実行して作成したフォルダを参照できるようになります。

図 26-7 PL/SQL スクリプトの実行 : 参照できるフォルダ



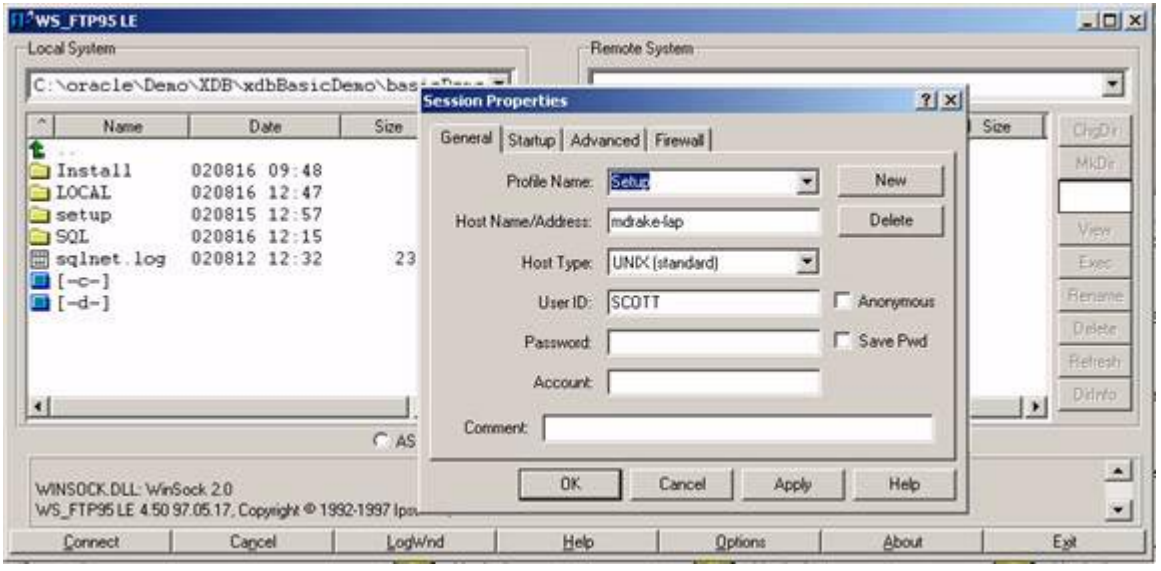
6. SQL> プロンプトに「QUIT」と入力して、SQL*Plus ウィンドウを閉じます。WebDAV ウィンドウを閉じます。

1.2 FTP を使用した構成ファイルのロード

この手順では、標準 FTP クライアントで文書を Oracle XML DB Repository にロードする方法が示されます。この手順では、Ipswitch Software 社の WS_FTP95 を使用していると想定しています。このクライアントが使用されている理由は、デモを簡単に実行できるように構成できることです。実際には、任意の FTP クライアントを使用できます。WS_FTP を使用する場合は、有効なライセンスを所有していることを確認してください。

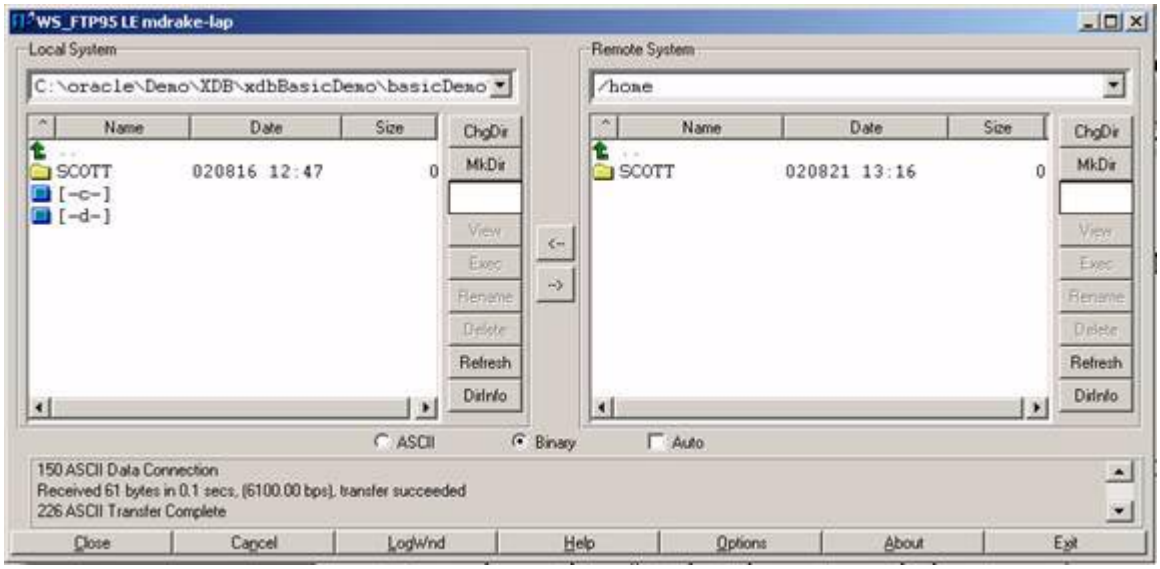
1. 「**1.2 Load Configuration Files**」アイコンをクリックして FTP クライアントを開き、XML DB リポジトリへの FTP 接続を確立します。

図 26-8 構成ファイルのロード



2. データベース・ユーザーのパスワードを入力し、「OK」をクリックします。FTP クライアントがデータベースに接続されます。図 26-9 に示すダイアログ・ボックスが表示されます。

図 26-9 FTP クライアントの表示



3. 「Local System」ペイン内の SCOTT フォルダをクリックし、下の方にある矢印をクリックして、SCOTT フォルダをローカル・ハード・ドライブから Oracle XML DB Repository に転送します。転送操作を確認するように求められた場合、確認してください。操作の完了後、「Exit」をクリックします。

   別の FTP ツールを使用する場合は、ローカルの SCOTT フォルダに含まれているすべてのファイルおよびフォルダが Oracle XML DB Repository 内の /home/SCOTT フォルダにコピーされていることを確認する必要があります。SCOTT フォルダのローカル・バージョンは、`basicDemo¥LOCAL¥Configuration Files` にあります。

   要点は次のとおりです。

   - Oracle や Oracle XML DB を認識しない標準 FTP クライアントが、一連のドキュメントを Oracle XML DB Repository にアップロードするために使用されています。
   - この手順では、XML Schema 文書、HTML ページおよびいくつかの XSLT スタイルシートを含むディレクトリ・ツリーがアップロードされます。
   - Oracle XML DB Repository を使用すると、XML Schema に基づく XML コンテンツと XML Schema に基づかない XML コンテンツと同様に HTML ファイル、JPEG イメージ、Word ドキュメントなどの非 XML コンテンツを格納できます。

4. 「Exit」ボタンをクリックして、FTP クライアントを閉じます。

2.0 XML DB Demo: XML Schema - XML DB が XML を断片化および格納する方法

この手順では、XML Schema の概要、および SQL99 オブジェクト型に基づいた構造化記憶域を使用して XML 文書を断片化および格納する Oracle XML DB の機能が示されます。この手順では、Altova 社の XMLSpy が使用されます。

要点は次のとおりです。

- Oracle XML DB は、W3C の XML Schema 勧告をサポートします。
- W3C の XML Schema 勧告では、一連の XML 文書の構造を定義するために使用可能な XML 言語の仕様が指定されています。XML Schema 定義は、それ自体が、W3C によって定義された一般的な XML Schema（スキーマのためのスキーマ）に準拠する XML 文書です。
- XML Schema によって、文書内の要素および属性の厳密な型指定が可能になります。XML Schema は、47 のスカラー・データ型を定義します。XML Schema で定義される型のベース・セットは、継承や拡張といったオブジェクト指向の技法を使用して、より複合的な型を定義するように拡張できます。
- XML Schema の最も一般的な用途は、一連の XML インスタンス・ドキュメントが XML Schema に準拠していることを確認するメカニズムとして使用することです。Oracle XML DB は、この方法で XML Schema を使用できます。
- Oracle XML DB は、XML を `XMLType` データ型のインスタンスとして表します。`XMLType` によって、データベースが XML を認識するようになり、XML を格納、取出し、問合せおよび操作するための便利な抽象的概念が提供されます。

  Oracle XML DB は、XML をデータベースに格納するための次の 2 つのオプションを提供します。

  * 非構造化記憶域という 1 つ目のオプションでは、CLOB データ型を使用して、XML がバイト文字列としてデータベースに永続的に保持されます。
  * 構造化記憶域という 2 つ目のオプションでは、XML が断片化され、そのコンテンツが一連の SQL オブジェクトとして永続的に保持されます。これらのオブジェクトは、SQL99 オブジェクト標準に基づいています。

  – 構造化記憶域は、XML が XML Schema に準拠する場合にのみ使用できます。Oracle XML DB は、XML Schema を使用して、インスタンス・ドキュメントのコンテンツを永続的に保持するために必要な一連の SQL オブジェクトを生成します。

  – 構造化記憶域には、XML を管理するうえで、多くのメリットがあります。これらのメリットには、メモリー管理の最適化、必要な記憶域の削減、コレクションに対する B ツリー索引の作成、部分的なインプレース更新が含まれます。これらのメリットと引き替えに、収集および取出し中のオーバーヘッドが増加します。

- データベース管理者およびアプリケーション開発者は、XML Schema に注釈を付けてコレクションの管理方法を制御することによって、パフォーマンスを調整できます。

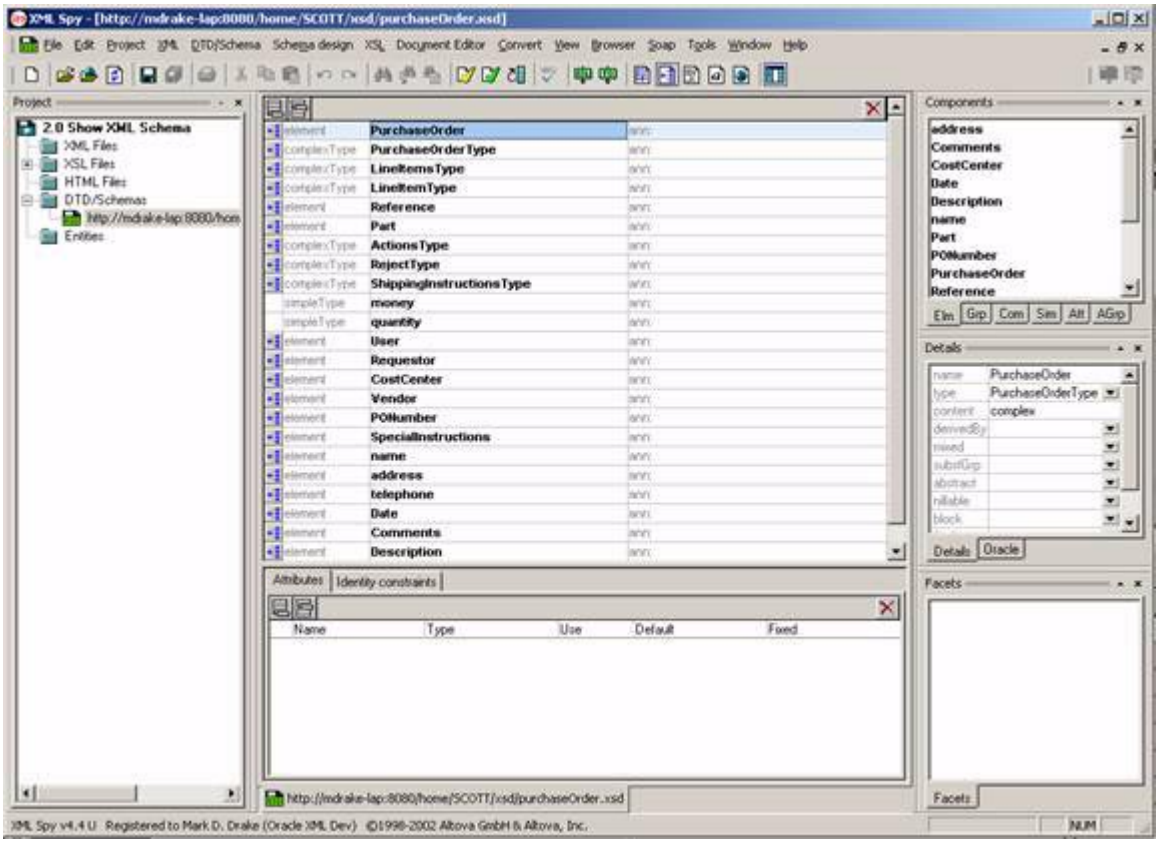
このデモでは、データベースにロードされている文書の 1 つが XML Schema です。この手順では、XMLSpy を使用して、Oracle XML DB による W3C の XML Schema 勧告サポートの主要な機能が示されます。

1. アイコンをクリックして、XMLSpy を起動します。「Project」ウィンドウ内の DTD/Schemas エントリの横にある「+」記号をクリックします。このブランチには、「`http://mdrake-lap:8080/home/SCOTT/xsd/purchaseOrder.xsd`」という項目が含まれています。この項目をダブルクリックして開きます。
2. データベース・パスワードを入力するためのプロンプトが表示されます。

   パスワードを入力し、「OK」をクリックします。

   XMLSpy に、XML Schema PurchaseOrder で定義された要素および型がグラフィカルに表示されます。

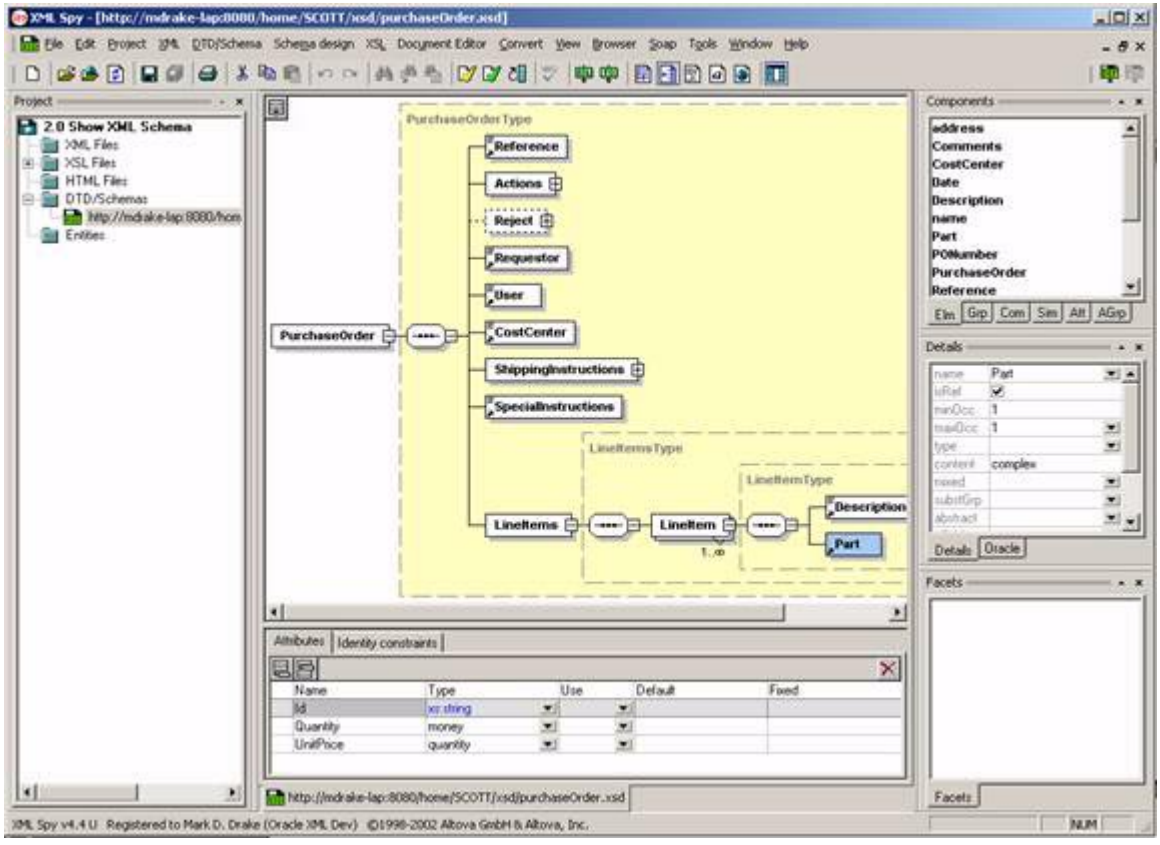
図 26-10 XML Schema で定義された要素および型の XMLSpy のグラフィカル表現



3. `PurchaseOrder` 要素の横にある制御ボタンをクリックします。次に、`lineItems` 要素の横にある「+」記号をクリックし、続いて `lineItem` 要素の横にある「+」記号をクリックします。最後に、`part` 要素をクリックします。

   この時点で、XMLSpy に XML Schema PurchaseOrder がグラフィカルに表示されます。

図 26-11 XMLSpy での XML Schema PurchaseOrder の表示



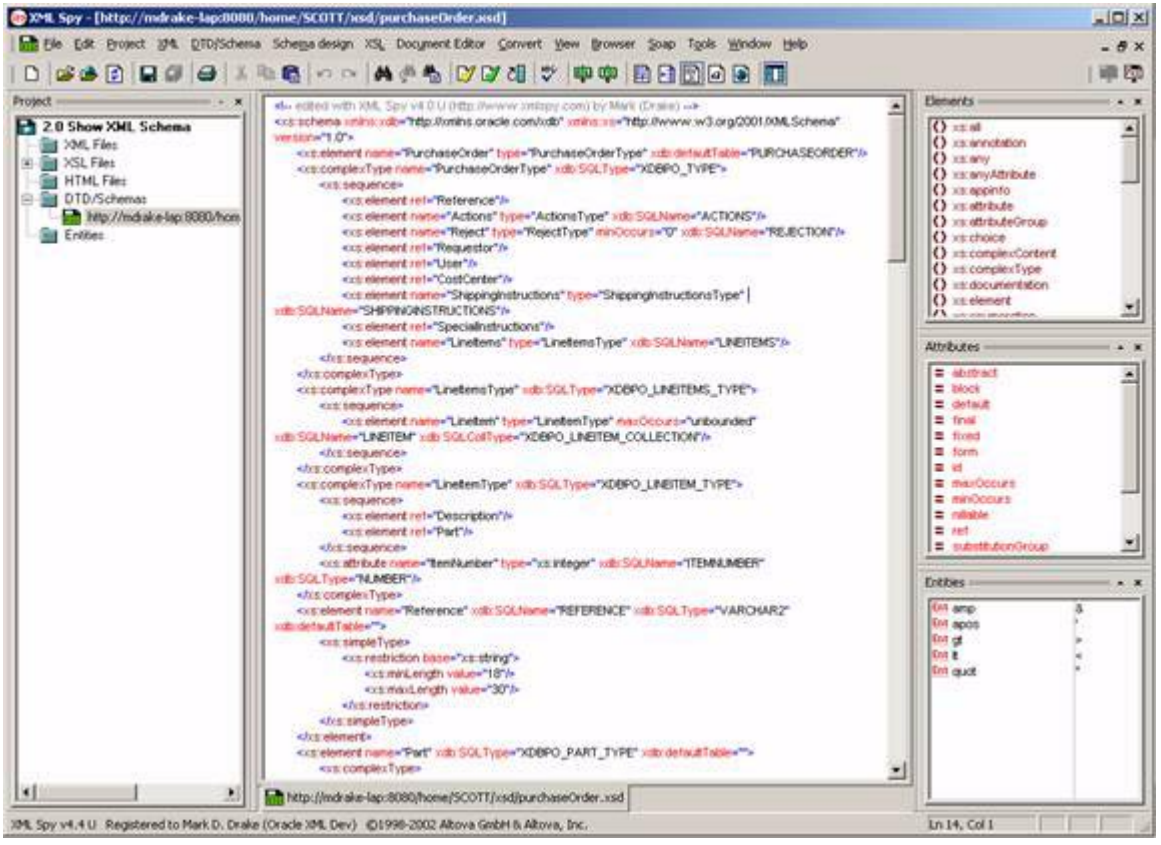
要点は次のとおりです。

- XMLSpy は、WebDAV プロトコルと FTP プロトコルの両方をサポートします。そのため、XMLSpy は Oracle XML DB に格納されたコンテンツに直接アクセスできます。
- グローバル要素 `PurchaseOrder` は、complexType `PurchseOrderType` のインスタンスです。`PurchaseOrderType` では、PurchaseOrder 文書を構成する一連の要素が定義されます。これらの 1 つは、`LineItem` 要素のコレクションを含む `LineItems` です。
- 各 `LineItem` 要素は、2 つの要素（`Description` および `Part`）で構成されます。
- `Part` 要素は、`Id`、`Quantity` および `UnitPrice` 属性を持ちます。

- PurchaseOrder スキーマは、通常の XML 文書の主要な機能を示す比較的単純な XML Schema です。ここでは、この XML Schema がグラフィカルに表示されています。

4. ツールバーの制御ボタンをクリックして、XML Schema のテキスト表示に切り替えます。これによって、XML Schema が XML 形式で表示されます。これは、W3C の XML Schema 委員会で定義されたスキーマのためのスキーマに準拠する XML 文書です。

図 26-12 XML 形式（XML 文書）での XML Schema の表示



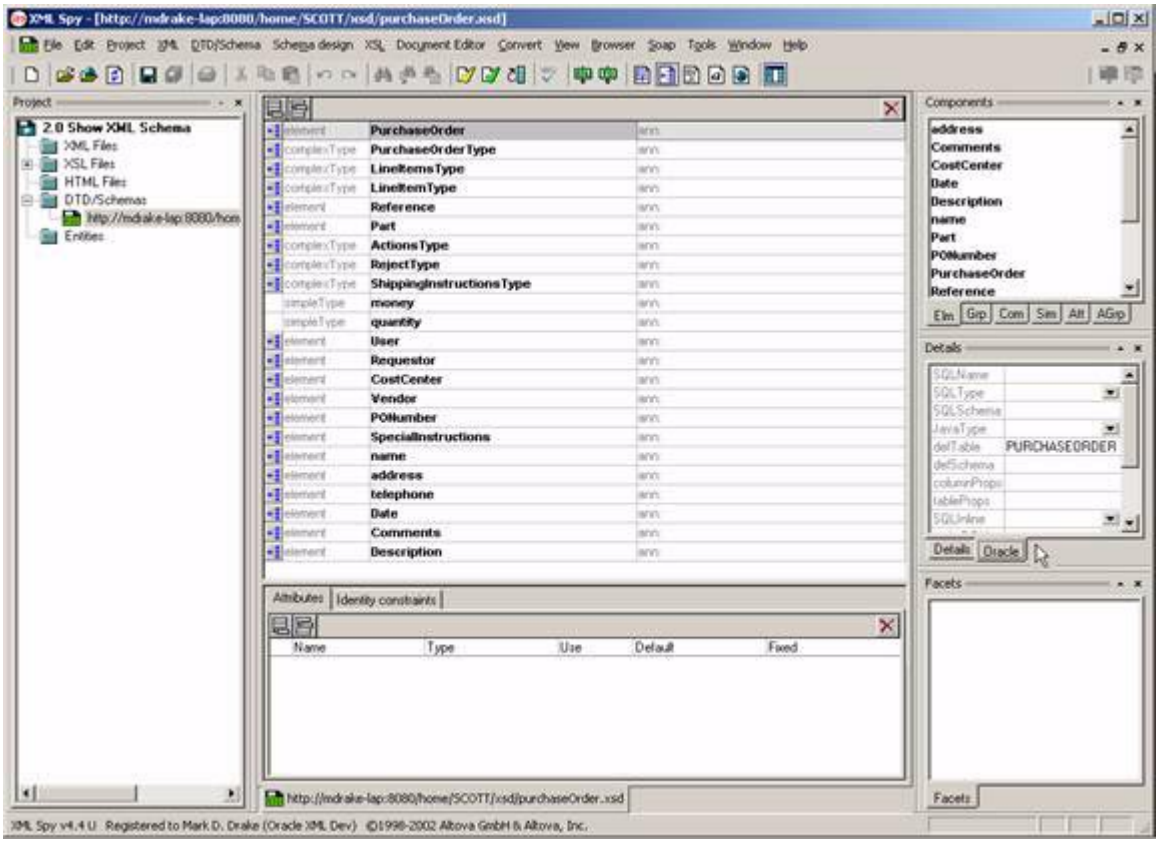
要点は次のとおりです。

- この XML Schema は、次の 2 つの名前空間を定義します。
  - `http://www.w3c.org/2001/XMLSchema`。W3C によってスキーマのためのスキーマ用に予約されている名前空間です。この名前空間は、XML 文書の構造を定義するために使用されます。
  - `http://xmlns.oracle.com/xdb`。オラクル社が Oracle XML DB 注釈のスキーマ注釈用に予約している名前空間です。この名前空間は、インスタンス・ドキュメントをデータベースに格納する方法を制御するスキーマに注釈を追加するために使用されます。

  注釈メカニズムは、W3C で承認された、W3C の XML Schema にベンダー固有の情報を追加するためのメカニズムです。
- Oracle XML DB は、注釈を含まないスキーマを登録できます。Oracle XML DB は、一連のデフォルトの前提を使用してスキーマを登録します。注釈を使用すると、アプリケーション開発者やデータベース管理者はこれらの前提をオーバーライドできます。
- 注釈を使用すると、次のものをオーバーライドできます。
  - 表、SQL オブジェクトおよび SQL 属性のネーミング
  - コレクションの管理方法
  - XML Schema のデータ型と SQL のデータ型間のマッピング
- このスキーマでは、次の注釈が使用されています。
  - `defaultTable` 注釈は、`PurchaseOrder` 要素内で使用され、このスキーマに準拠する XML 文書を `PURCHASEORDER` という表に格納することを定義します。
  - `SQLType` 注釈は、complexType `PurchaseOrderType` から生成される SQL 型の明示的な名前を指定するために使用されます。

5. ツールバーの制御ボタンをクリックして、XML Schema のグラフィカル表示に戻ります。

図 26-13 XML Schema のグラフィカル表示



要点は次のとおりです。

- XMLSpy には、「**Oracle**」タブが表示されます。これを使用すると、グラフィカル編集モードでの作業中に Oracle XML DB のスキーマ注釈を入力できます。

6. complexType `PurchaseOrderType` の横にあるアイコンをクリックします。
7. このウィンドウはまだ閉じないでください。

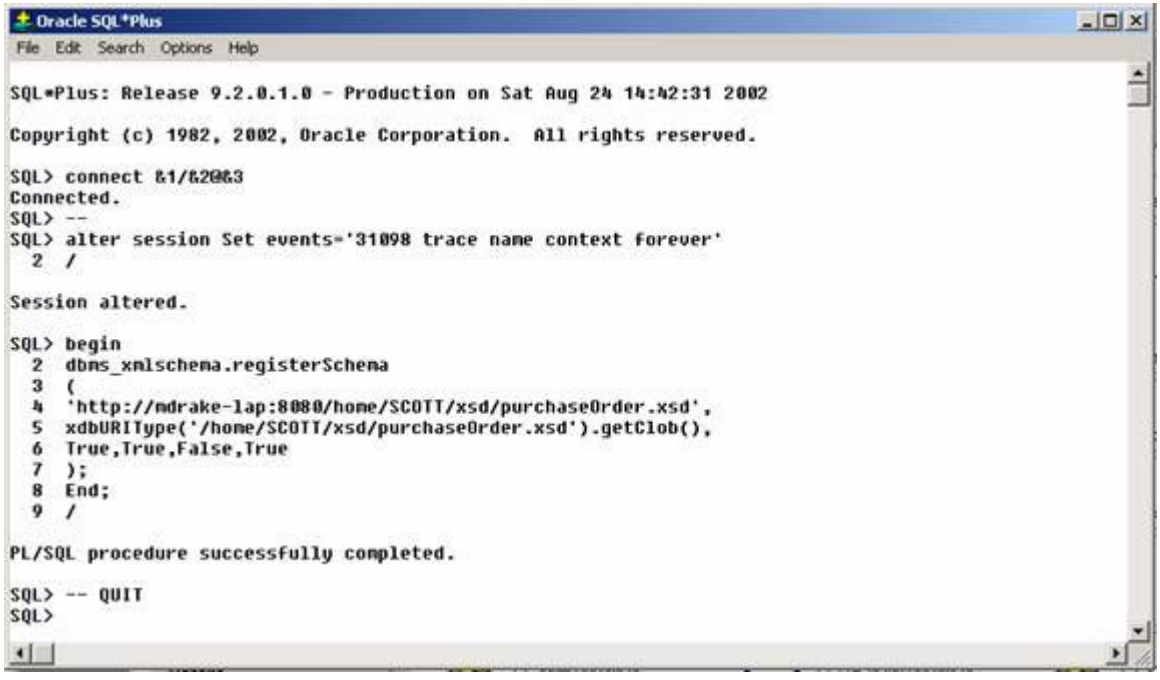
2.1 XML Schema の登録

この手順では、Oracle XML DB に XML Schema を認識させる方法が示されます。デモのこの時点では、XML Schema は Oracle XML DB Repository に格納されているだけで、データベースは XML Schema の存在を認識していません。

> **参照：** 第 5 章「XMLType の構造化されたマッピング」を参照してください。

1. 「**2.1 Register XML `Schema`**」アイコンをクリックして、SQL スクリプトを実行します。

図 26-14 dbms_xmlschema.registerSchema スクリプトの実行



要点は次のとおりです。

- XML Schema は URL に登録されます。この URL は、XML インスタンス・ドキュメントがそれ自身を XML Schema で定義されたクラスのメンバーとして識別するために使用する URL です。

- XML 文書を XML Schema で定義されたドキュメント・クラスのメンバーとして識別する方法は、W3C の XML Schema ワーキング・グループで定義されています。
- この URL は、登録された XML Schema にインスタンス・ドキュメントを関連付けるために使用されるキーです。Oracle XML DB は、この URL にアクセス可能である必要があります。
- `registerSchema()` プロシージャは、XML Schema で定義されたすべてのオブジェクトおよび型を作成します。

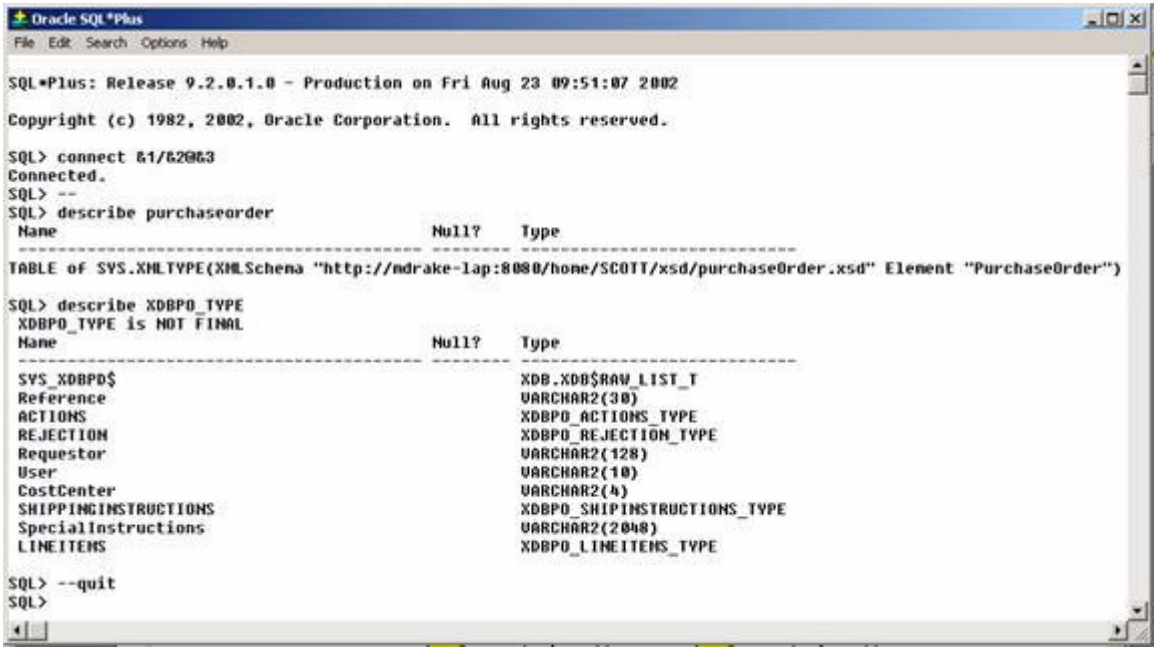
2. SQL> プロンプトに「QUIT」と入力して、SQL*Plus ウィンドウを閉じます。

2.2 XML Schema の登録によるオブジェクトの作成

この手順では、XML Schema を登録した結果として作成されるいくつかのオブジェクトが示されます。

1. 「**2.1 Show Objects**」アイコンをクリックして、SQL スクリプトを実行します。

図 26-15 DESCRIBE を使用した、XML Schema の登録時に作成されたオブジェクトの表示



要点は次のとおりです。

- PurchaseOrder 表はオブジェクト表です。表の各行は、オブジェクトで表されます。対象のオブジェクトは XMLType です。
- 表では、PurchaseOrder のルート・ノードを持つ XML 文書が管理されます。PurchaseOrder 要素の定義は、http://mdrake-lap:8080/home/SCOTT/xdb/purchaseOrder.xsd という URL に登録された XML Schema で定義されます。
- 右方向にスクロールすると、各 PurchaseOrder 文書が XDBPO_TYPE オブジェクトのインスタンスとしてオブジェクト・リレーショナル形式で格納されていることが示されます。
- XDBPO_TYPE オブジェクトの SQL 属性は、complexType PurchaseOrderType で定義された要素および属性から導出されます。

2. XMLSpy ウィンドウをクリックします。XDBPO_TYPE の SQL*Plus 定義を XMLSpy による complexType のグラフィカル表現と比較します。
3. SQL> プロンプトに「QUIT」と入力して、SQL*Plus ウィンドウを閉じます。XMLSpy を閉じます。

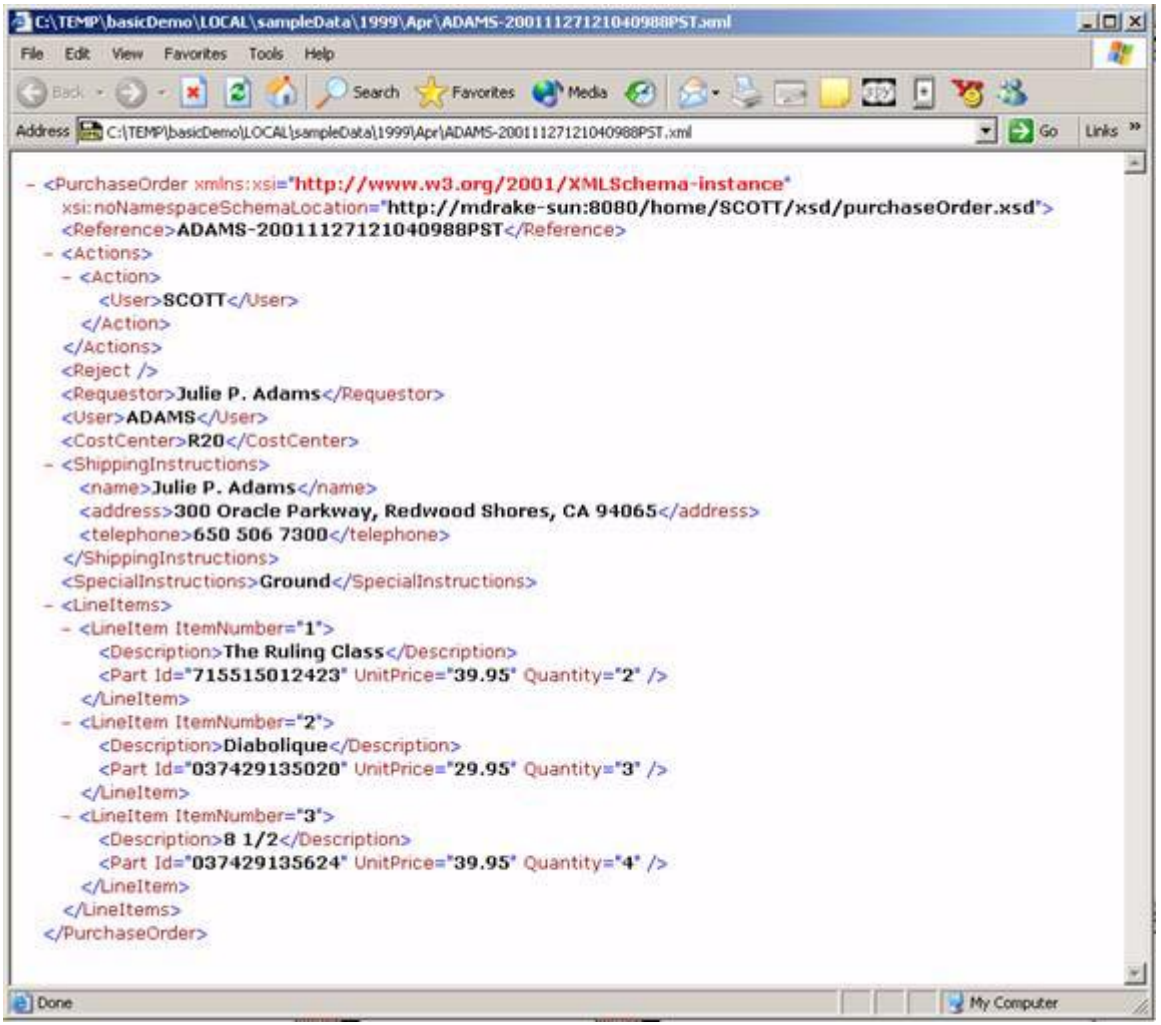
3.0 XML DB Demo: XML Schema への XML ファイルの準拠

この手順では、登録された XML Schema に準拠するインスタンス・ドキュメントがローカル・ハード・ドライブ上の sampleData フォルダに含まれていることが示されます。

1. 「**3.0 Sample Files**」アイコンをクリックして、sampleData フォルダを開きます。1999 フォルダを開きます。Apr フォルダを開きます。ADAMS-20011127121040988PST.xml 文書を右クリックし、「Open」を選択します。

   これによって、Internet Explorer が起動され、文書が表示されます。文書が別のアプリケーションで開かれた場合、Windows エクスプローラのフォルダ オプション機能を使用して、ファイルの関連付けを設定します。

図 26-16 ADAMS-20111127121040988PST.xml 文書の表示



要点は次のとおりです。

- このファイルは XML Schema に準拠しています。
- `noNamespaceSchemaLocation` 属性が、この文書を XML Schema で定義されたドキュメント・クラスのインスタンスとして識別するために使用されています。

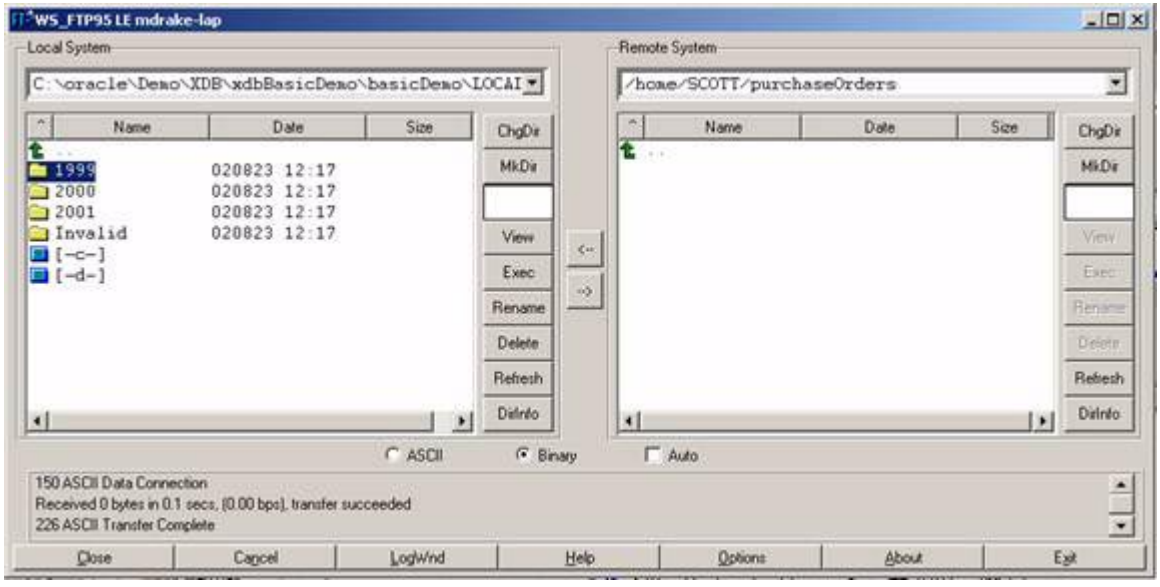
2. Internet Explorer および sampleData/1999/Apr ウィンドウを閉じます。Windows デスクトップ上にあるデモ・フォルダを再度開きます。

3.1 FTP を使用したインスタンス・ドキュメントのロード

この手順では、FTP を使用して 1999 フォルダ・ツリーが Oracle XML DB Repository にコピーされます。この手順では、Oracle XML DB で登録された XML Schema のインスタンスとしてドキュメントが認識され、それに応じてそれらのドキュメントが処理される方法が示されます。

1. このアイコンをクリックして、FTP クライアントを開き、XML DB リポジトリへの FTP 接続を確立します。データベース・ユーザーのパスワードを入力するためのプロンプトが表示された場合、入力して、「OK」をクリックします。パスワードを入力すると、次のウィンドウが表示されます。

図 26-17 FTP を使用した Oracle XML DB Repository への 1999 構造のコピー



2. 「Local System」ペイン内の 1999 フォルダをクリックし、下の方にある矢印をクリックして、1999 フォルダおよびそのすべてのサブフォルダをローカル・ハード・ドライブから Oracle XML DB Repository 内の home/SCOTT/purchaseOrders フォルダにコピーします。操作の完了後、「Exit」をクリックします。

要点は次のとおりです。

- 標準 FTP クライアントを使用して、コンテンツが Oracle データベースに直接ロードされています。その他に項目またはサーバーを変更する必要はありません。これは、Oracle XML DB で FTP および HTTP/WebDAV（データベースで認識される一連のプロトコル）がサポートされているためです。
- 各ドキュメントのルート・ノードには、登録された XML Schema のインスタンスとして識別する `noNameSpaceSchemaLocation` 属性が含まれているため、ドキュメントは断片化され、一連のオブジェクトとしてデータベースに格納されています。

3. 「EXIT」ボタンをクリックして、FTP クライアントを閉じます。

3.2 SQL を使用した XML データへの制約の追加

この手順では、SQL 機能を使用して XML 文書を Oracle XML DB に格納する方法が示されます。表に制約を追加すると、XML データが制約されます。また、この手順では、XML 文書の完全な XML Schema 検証が有効になります。

> **参照：** 第 4 章「XMLType の使用」を参照してください。

要点は次のとおりです。

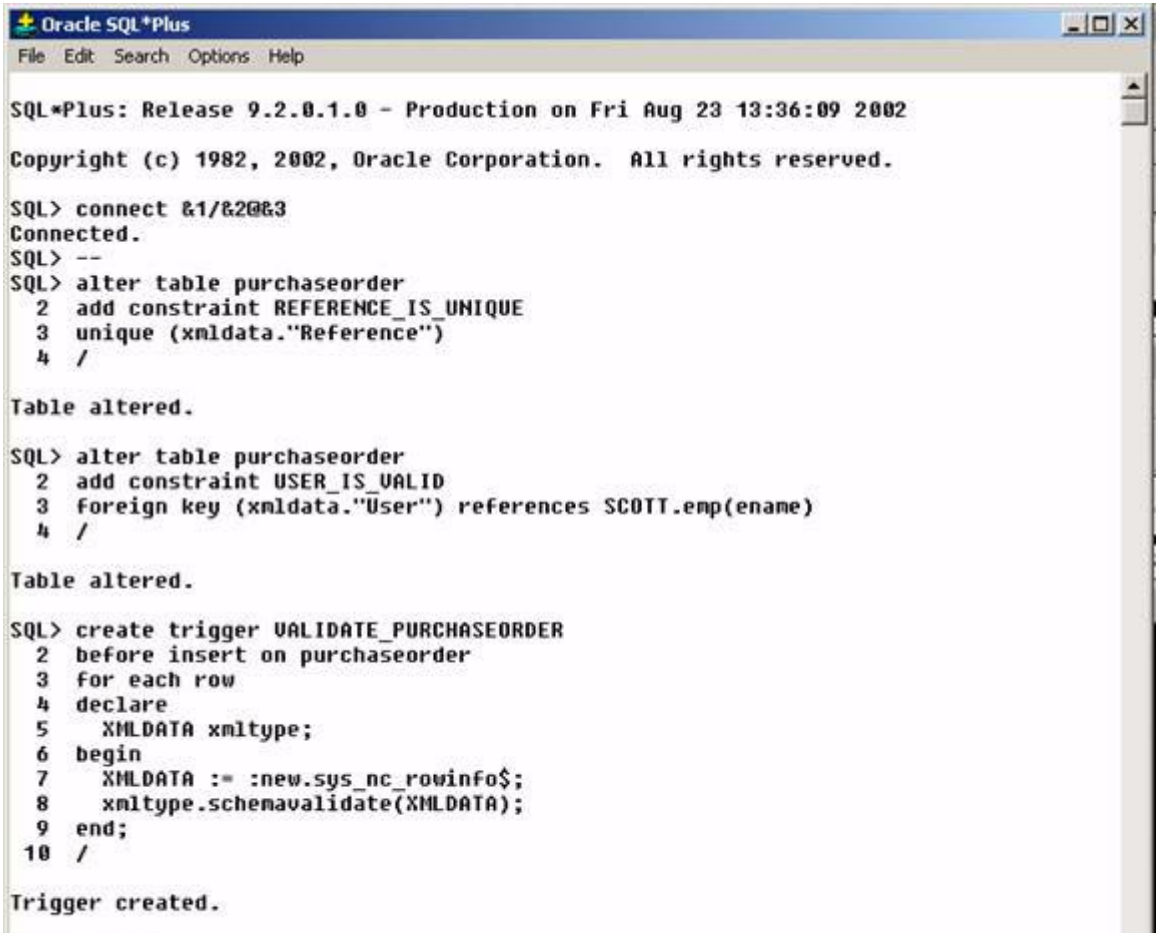
- XML Schema は強力です。Oracle XML DB の今回のリリースでは、XML Schema 勧告のバージョン 1.0 がサポートされています。ただし、XML Schema を使用して表現できない比較的単純ないくつかの概念があります。これらの概念には次のものが含まれます。
  - 要素または属性の値がドキュメントのコレクション全体で一意である必要があります。SQL では、単純な一意制約といいます。
  - 一部の要素または属性の値が、別のドキュメント内または XML Schema に基づかないデータ・ストア（リレーショナル表や LDAP ディレクトリなど）内の値と一致する必要があります。
- Oracle XML DB を使用して XML を管理すると、SQL の機能を使用してこれらのデメリットを解決できます。SQL では、単純な外部キー制約といいます。
- Oracle XML DB は、ドキュメントがデータベースに挿入されたときに、それらのドキュメントの完全な XML Schema 検証を自動的に実行しません。完全な XML Schema 検証はオプションです。
  - デフォルトでは、Oracle XML DB は各ドキュメントの簡易検証を実行します。この検証では、必須の要素および属性が存在していること、コレクション内の要素数が XML Schema で定義された値に準拠していること、および列挙が XML Schema に準拠していることが確認されます。
  - 簡易検証では、パターンへの一致、最小長などは確認されません。これは、パフォーマンスの最適化です。XML Schema 検証は、CPU 集中型の操作です。また、

多くのアプリケーションではドキュメントをデータベースに挿入する前にそのドキュメントの完全な検証が実行されるため、データベースですぐにそのドキュメントを再度検証することは効率的ではありません。

- CHECK 制約またはトリガーを使用すると、完全な XML Schema 検証を XML Schema ごとに有効にすることができます。

1. 「**3.2 Add Constraints**」アイコンをクリックして、SQL スクリプトを実行します。

図 26-18 制約の追加およびトリガーの作成



要点は次のとおりです。

- 現在、制約を定義するには SQL99 オブジェクト構文を使用する必要があります。Oracle XML DB の将来のリリースでは、より直観的に使用できる XPath 式を使用して制約を定義できる予定です。
- XMLType 表の行のコンテンツは、トリガー内から SYS_NC_ROWINFO$ として参照される必要があります。
- XML Schema 検証は、XMLType に schemaValidate() メソッドをコールして実行されます。トリガーによって、検証によって意味のあるエラー・メッセージが戻され、これらのエラー・メッセージを捕捉し、適切な処置を行うようにすることができます。
- 最初の制約によって、XPath 式 /PurchaseOrder/Reference で識別された要素の値が、PURCHASEORDER 表に格納されたすべての PurchaseOrder 文書間で一意であることが保証されます。
- 2 番目の制約によって、XPath 式 /PurchaseOrder/User で識別された要素の値が、SCOTT.EMP 表の ENAME 列で検出されることが保証されます。
- トリガーによって、PURCHASEORDER 表にロードされたすべてのドキュメントに対して完全な XML Schema 検証が行われることが保証されます。

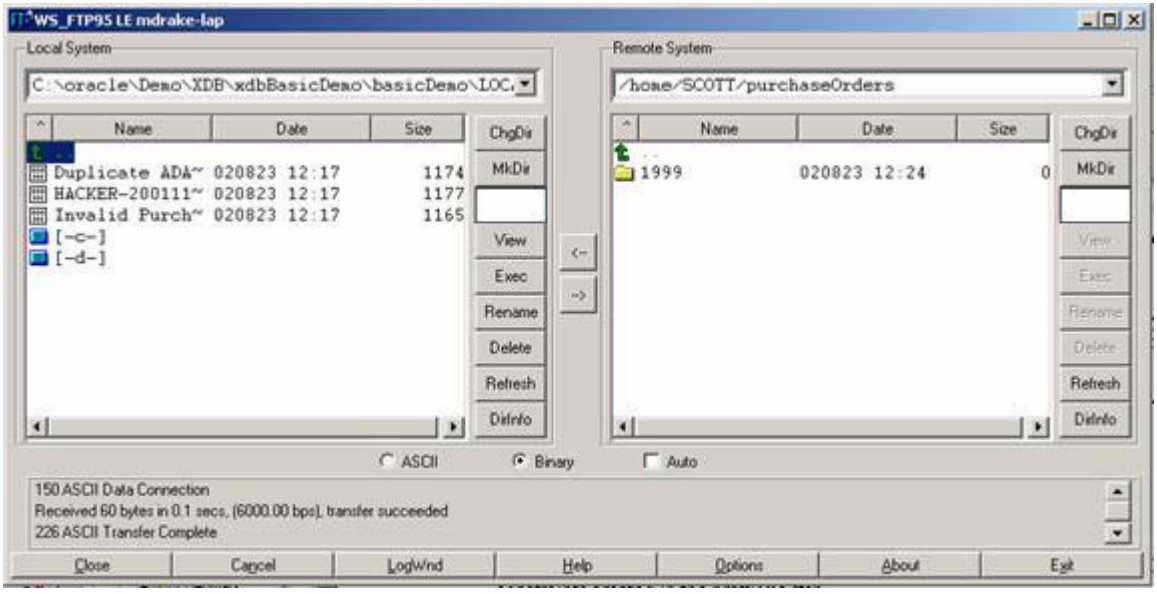
2. SQL> プロンプトに「QUIT」と入力して、SQL*Plus ウィンドウを閉じます。

3.3 FTP を使用した、制約に違反する XML 文書のアップロード

この手順では、FTP を使用して、前述の手順で作成された制約に違反する一連の文書のアップロード（の試行）が実行されます。

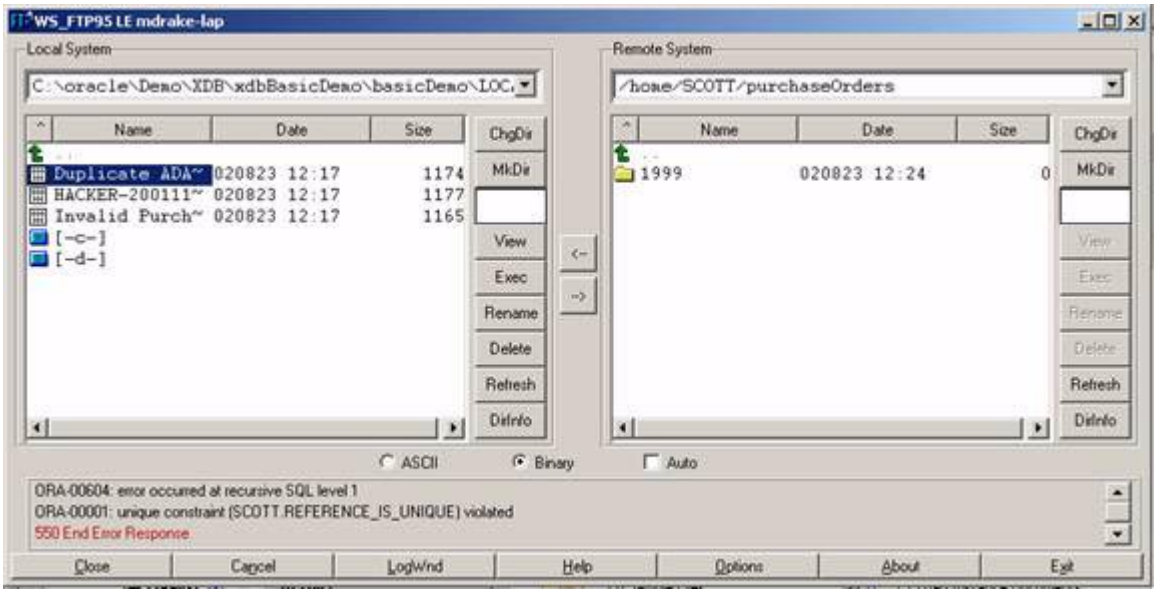
1. 「**3.3 Violate Constraints**」アイコンをクリックして、FTP クライアントを開き、XML DB リポジトリへの FTP 接続を確立します。
2. データベース・ユーザーのパスワードを入力するためのプロンプトが表示された場合、入力して、「OK」をクリックします。パスワードを入力すると、次のウィンドウが表示されます。

図 26-19 FTP クライアントのオープンによる FTP 接続の確立



3. 「Local System」ペイン内の Duplicate ADA~ ファイルをクリックし、下の方にある矢印をクリックして、この文書を Oracle XML DB Repository の `purchaseOrders` フォルダにコピーします。

図 26-20 一意制約への違反によって発生する FTP エラー



文書をアップロードすると、次のエラーが表示されます。

```
ORA-00604: 再帰 SQL レベル 1 でエラーが発生しました。
ORA-00001: 一意制約（SCOTT.REFERENCE_IS_UNIQUE）に反しています
```

この文書をアップロードした結果、手順 3.2 で作成された REFERENCE_IS_UNIQUE 制約への違反が発生します。これは、この文書内の PurchaseOrder/Reference/text() ノードの値が、手順 3.1 でロードされたいずれかの文書内のノード値と同じであるためです。したがって、この操作は失敗します。

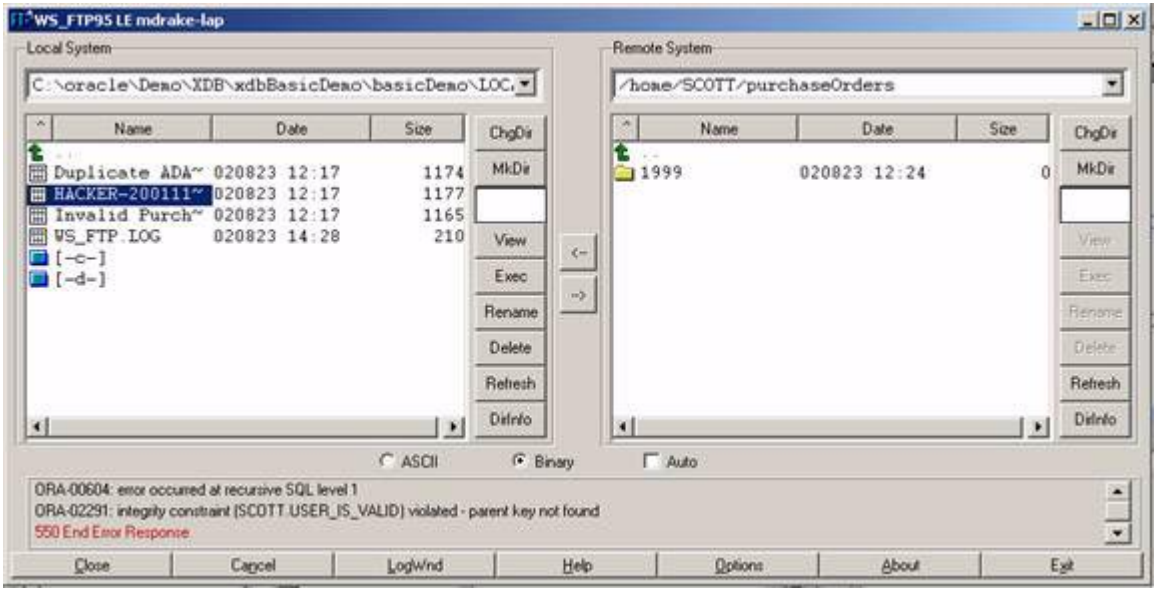
4. 次に、「Local System」ペイン内の HACKER-200111~ ファイルをクリックし、「View」ボタンをクリックします。

注意： /PurchaseOrder/User/text() ノードの値は「HACKER」です。

---

5. 下の方にある矢印をクリックして、この文書を Oracle XML DB Repository の purchaseOrders フォルダにコピーします。

図 26-21 USER_IS_VALID 制約への違反によって発生する FTP エラー



文書をアップロードすると、次のエラーが表示されます。

```
ORA-00604: 再帰 SQL レベル 1 でエラーが発生しました。
ORA-02291: 整合性制約 (SCOTT.USER_IS_VALID) に違反しました - 親キーがありません
```

この文書をアップロードした結果、手順 3.2 で作成された USER_IS_VALID 制約への違反が発生します。これは、この文書内の PurchaseOrder/User/text() ノードの値が「HACKER」であり、この値が SCOTT.EMP の ENAME 列で検出されなかったためです。したがって、この操作は失敗します。

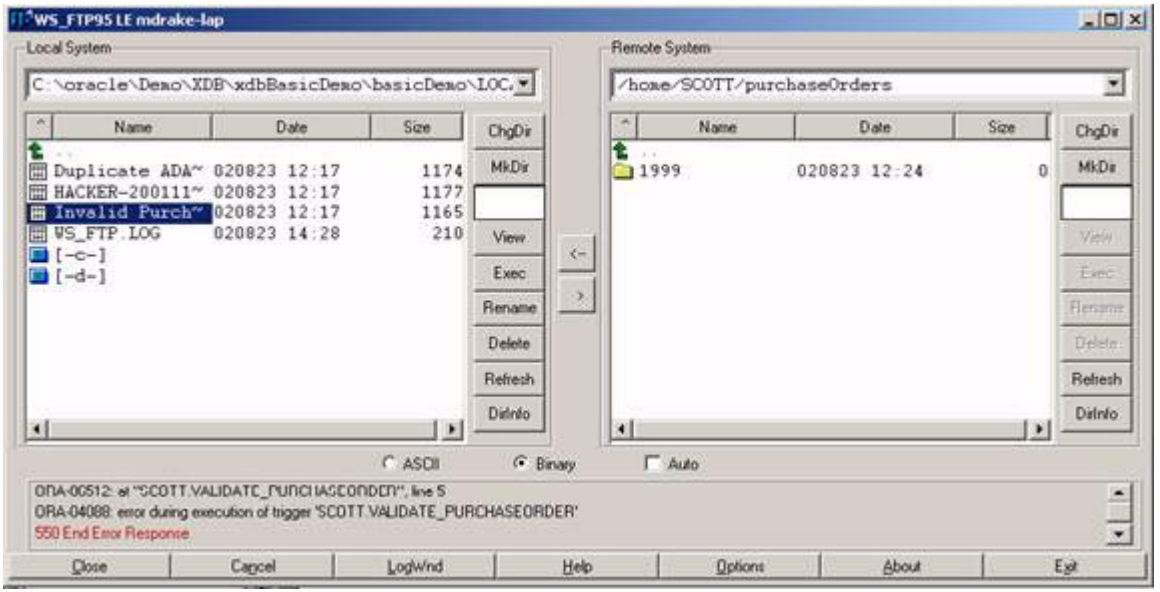
6. 次に、「Local System」ペイン内の Invalid Purch~ ファイルをクリックし、「View」ボタンをクリックします。

注意： /PurchaseOrder/Reference/text() ノードの値は「ADAMS-20011127PST」です。

---

7. 下の方にある矢印をクリックして、この文書を Oracle XML DB Repository の purchaseOrders フォルダにコピーします。

図 26-22 VALIDATE_PURCHASEORDER トリガーの起動によって発生する FTP エラー



文書をアップロードすると、次のエラーが表示されます。

```
ORA-00604: 再帰 SQL レベル 1 でエラーが発生しました。
ORA-31154: 無効な XML 文書
ORA-19202: XML 処理中にエラーが発生しました
LSX-00221: "ADAMS-20011127PST" は短すぎます（最小長 18）
ORA-06512: "SYS.XMLTYPE" 行 0
ORA-06512: "SCOTT.VALIDATE_PURCHASEORDER" 行 5
ORA-04088: トリガー 'SCOTT.VALIDATE_PURCHASEORDER' の実行中にエラーが発生しました
```

この文書をアップロードした結果、VALIDATE_PURCHASEORDER トリガーが起動されます。トリガーで実行された XML Schema 検証処理で、/PurchaseOrder/Reference/text() ノードの値が XML Schema で設定されたルールに準拠していないことが検出されています。XML Schema では、このノードの最小長を 18 文字とすることが定義されています。この文書は XML Schema で定義されたドキュメント・クラスの妥当なインスタンスではないため、この操作は失敗します。

要点は次のとおりです。

- 制約およびトリガーを使用して、XML 文書の整合性を規定できます。データベースを使用して XML を管理すると、SQL の機能を XML の柔軟性と組み合せることができます。
- 制約およびトリガーは、プロトコルを使用してコンテンツをデータベースにアップロードする場合でも規定されます。
- XML 文書を Oracle XML DB に格納すると、組織は、Oracle データベースの信頼性、可用性、拡張性およびセキュリティを XML コンテンツに持たせることができます。

8. 「Exit」ボタンをクリックして、FTP クライアントを閉じます。

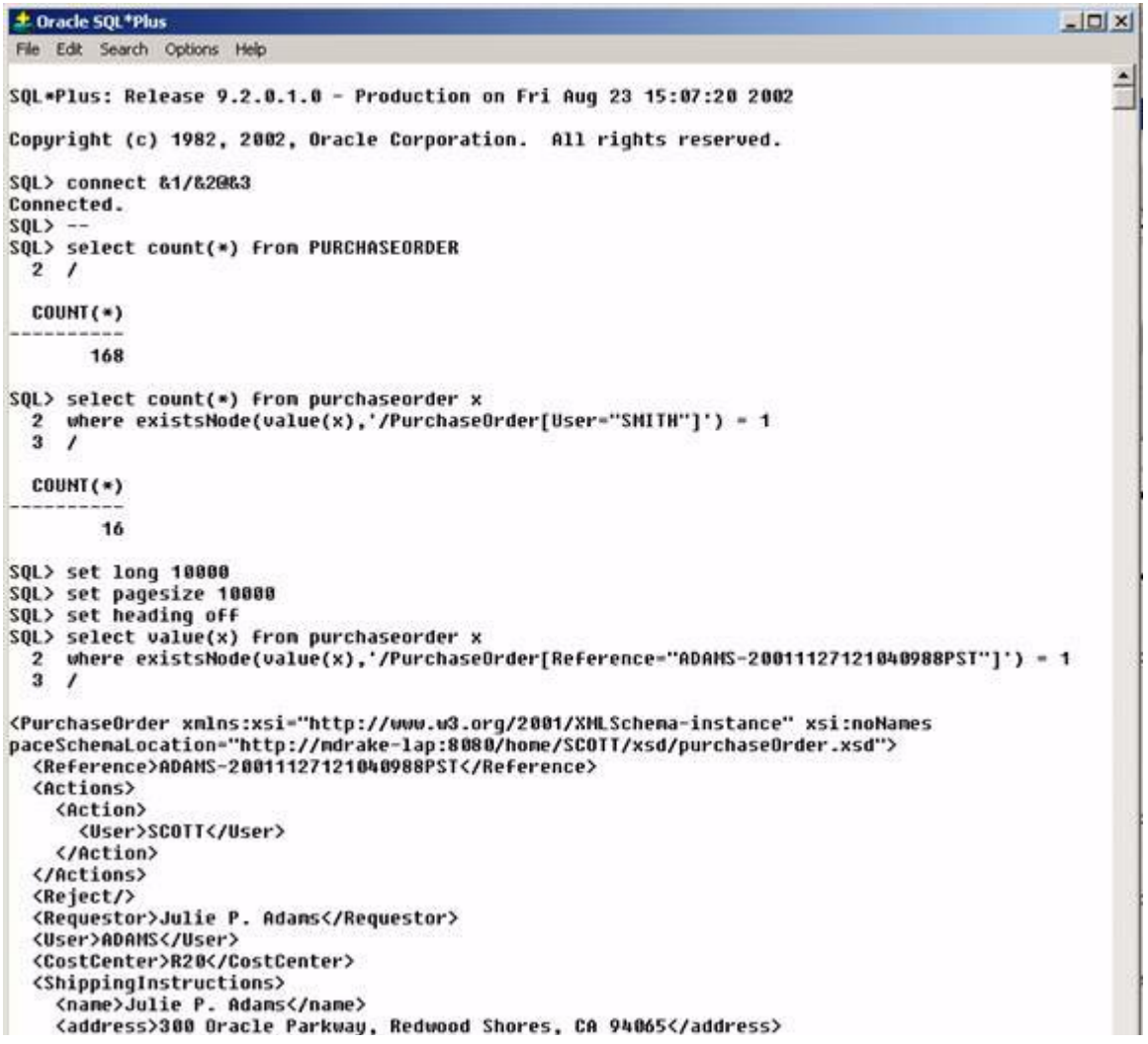
4.0 XML DB Demo: XML 文書に対する単純な XPath 問合せ

この手順では、XML 文書に対して単純な XPath 問合せを実行する方法が示されます。

参照： 第 4 章「XMLType の使用」を参照してください。

1. 「**4.0 Simple Queries**」アイコンをクリックして、SQL スクリプトを実行します。

図 26-23 WHERE 句で existsNode() を使用した、戻される文書の限定



要点は次のとおりです。

- 単純な問合せは、一般的な SQL 構文を使用して記述できます。
- SQL/XML 演算子 `existsNode()` を WHERE 句で使用すると、問合せによって戻される一連の文書を限定できます。`existsNode()` は、XPath 式を XML 文書に適用し、その XPath 式と一致するノードがその文書に含まれている場合は TRUE（1）、そうでない場合は FALSE（0）を戻します。
- XPath は、XML 文書のコンテンツを問合せおよびアクセスするための W3C 標準であり、XML プログラマおよび作成者にとって一般的な構文です。
- 最初の例は、`PurchaseOrder` 表内の行数をカウントして PurchaseOrder 文書の数を調べる方法を示しています。この表内には、文書ごとに 1 つの行が含まれます。
- 2 番目の例は、`existsNode()` 関数および単純な XPath 式を使用して、`PurchaseOrder/User/text()` ノードの値に「SMITH」という値が含まれる PurchaseOrder 文書の数を調べる方法を示しています。
- 3 番目の例は、`value()` 演算子を使用して、`XMLType`（オブジェクト）表に行として格納された文書のコンテンツ全体を表示する方法を示しています。また、`existsNode()` 演算子を使用して、`/PurchaseOrder/Reference/text()` ノードに「ADAMS-20011127121040988PST」という値が含まれる行に結果を限定する方法も示しています。

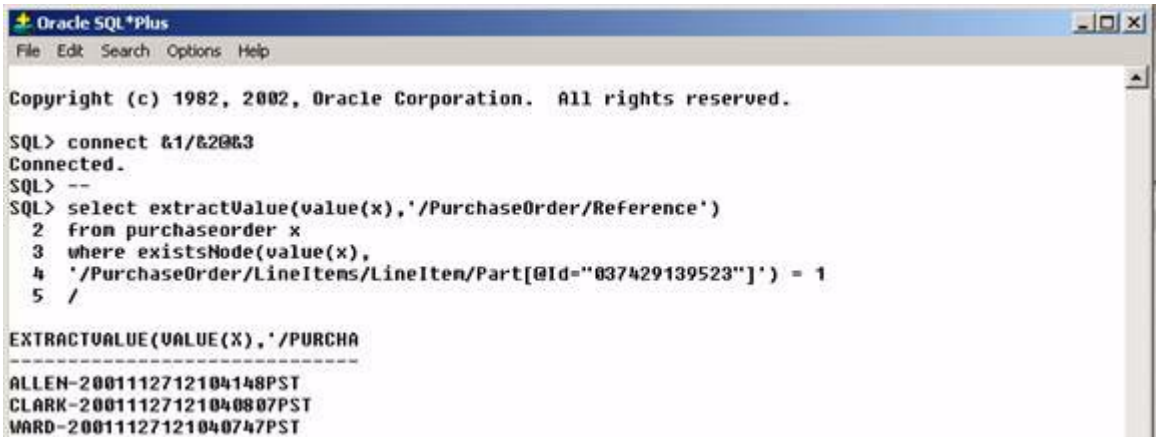
2. SQL> プロンプトに「QUIT」と入力して、SQL*Plus ウィンドウを閉じます。

4.1 XML 文書に対するより複雑な XPath 問合せ

この手順では、`extractValue()` 句を使用して、XPath 式に基づく文書内のノードの値を取得する方法が示されます。また、Oracle XML DB が XML 文書の階層深くナビゲートする複雑な XPath 式を評価できることも示されます。

1. 「**4.1 Simple Queries (2)**」アイコンをクリックして、SQL スクリプトを実行します。

図 26-24 extractValue() を使用した、XPath 式に基づくノード値の取得



要点は次のとおりです。

- `existsNode()` 関数を使用して、問合せによって戻される一連の文書が、「037429139523」という値を含む `Id` 属性を持つ `part` 要素が含まれる `lineItem` 要素を含む文書に限定されています。

  `lineItem` 要素は、各ドキュメントに複数回出現します。

- XPath `/PurchaseOrder/LineItems/LineItem/Part[@Id="037429139523"]` では検索する出現が明示的に識別されていないため、`lineItem` 要素のすべてのインスタンスが、指定した条件を満たしているかどうかを確認するために検索されます。

- `extractValue()` 関数を使用して、XPath 式 `/PurchaseOrder/Reference/text()` で識別されたノードの値のみが戻されます。

- Oracle XML DB は、複雑な XPath 式を効率的に評価できます。

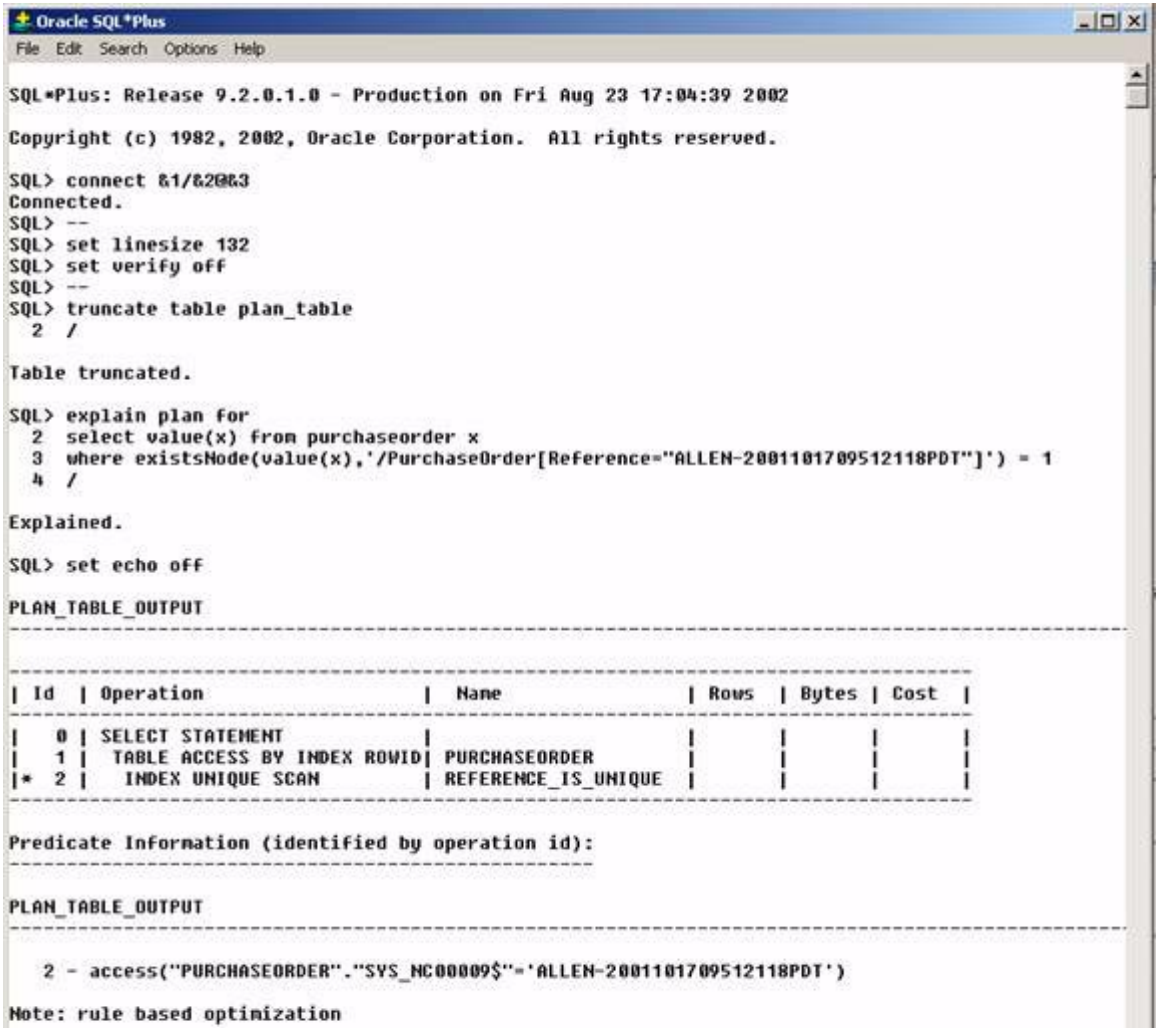
2. SQL> プロンプトに「QUIT」と入力して、SQL*Plus ウィンドウを閉じます。

4.2 XML 表に対する問合せの実行計画

この手順では、XML 文書の表に対して問合せを実行することによって生成される実行計画が示されます。

1. 「**4.2 Explain Plan (1)**」アイコンをクリックして、SQL スクリプトを実行します。

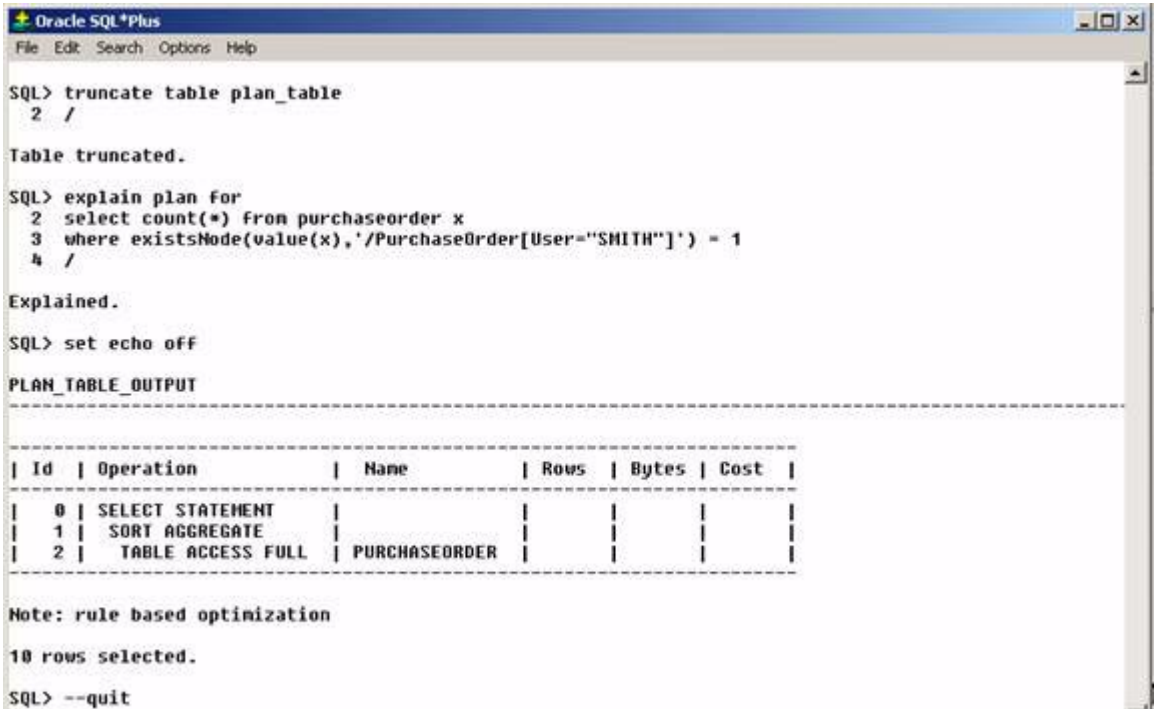
図 26-25 XPath ノード文書を取り出すための問合せの実行計画



要点は次のとおりです。

- この実行計画は、XPath 式 `/PurchaseOrder[Reference="ALLEN-20011017095121l8PDT"]` で識別されたノードを含む文書を取り出すための問合せを示しています。
- この問合せは、一意索引を使用して解決されます。この一意索引は、一意制約が表に追加されたときに作成されています。
- クエリー・リライトによって、XPath 式を索引付けされたアクセスに変換することが可能になります。Oracle XML DB は、文書の XML Schema およびそれから導出された記憶域モデルを認識しているため、XPath 式を基礎となるオブジェクト・モデルに対するオブジェクト・リレーショナルな SQL 問合せにリライトできます。
- Oracle オプティマイザは、その問合せを最適化し、結果セットを戻すための最適な計画を評価します。

図 26-26 ノードで文書の数をカウントする問合せの実行計画



要点は次のとおりです。

- この実行計画は、XPath 式 `/PurchaseOrder[User="SMITH"]` で識別されたノードを含む文書の数をカウントするための問合せを示しています。
- この問合せは、表スキャンを使用して解決されます。これは、このデモで使用されるサンプル・データを構成する 168 文書では問題ありませんが、表に何百万もの文書が含まれているような現実のシステムでは使用できません。

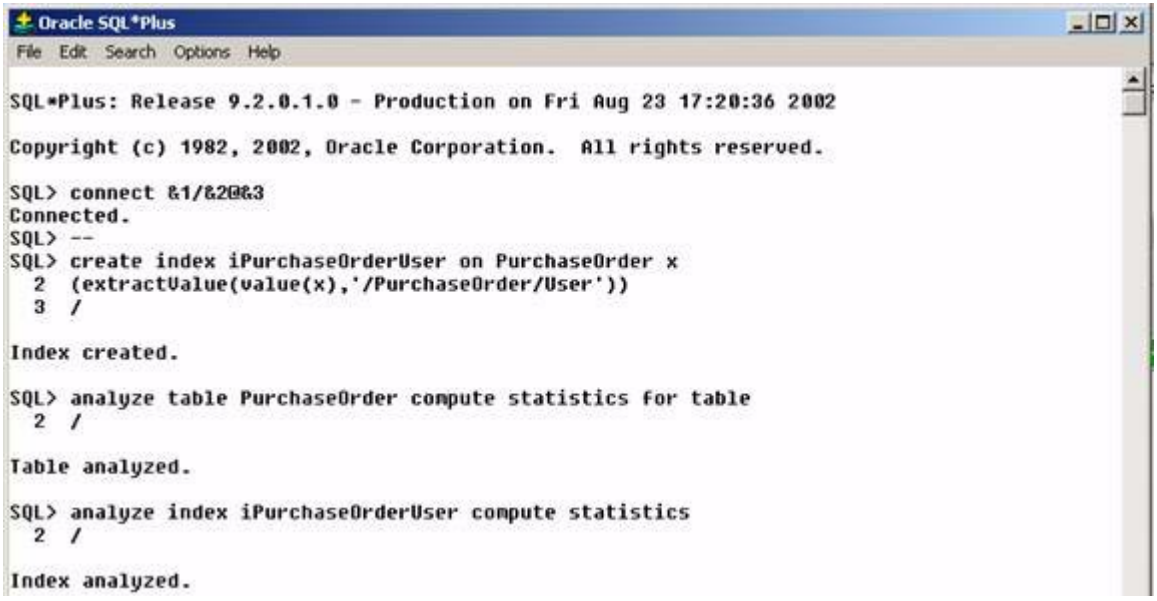
2. SQL> プロンプトに「QUIT」と入力して、SQL*Plus ウィンドウを閉じます。

4.3 extractValue() および XPath 式を使用した XML 索引の作成

この手順では、XPath 式を使用して索引を作成する方法が示されます。前述の例に示す 2 番目の問合せをサポートするための索引を作成すると、表スキャンが不要になります。

1. 「**4.3 Create XML Indexes**」アイコンまたは同様のアイコンをクリックして、SQL スクリプトを実行します。

図 26-27 extractValue() および XPath 表記法を使用した索引の作成



要点は次のとおりです。

- 表スキャンを不要にするには、問合せを解決するための索引を作成します。
- この索引は、問合せを表現するために使用した XPath 表記法と同じ表記法を使用して定義されます。
- この索引は、ファンクション索引ではありません。クエリー・リライトは、CREATE INDEX 文で指定された XPath 式を基礎となるオブジェクトの適切な属性にマップする場合に有効です。これによって、これらの属性に従来の B ツリー索引が作成されます。

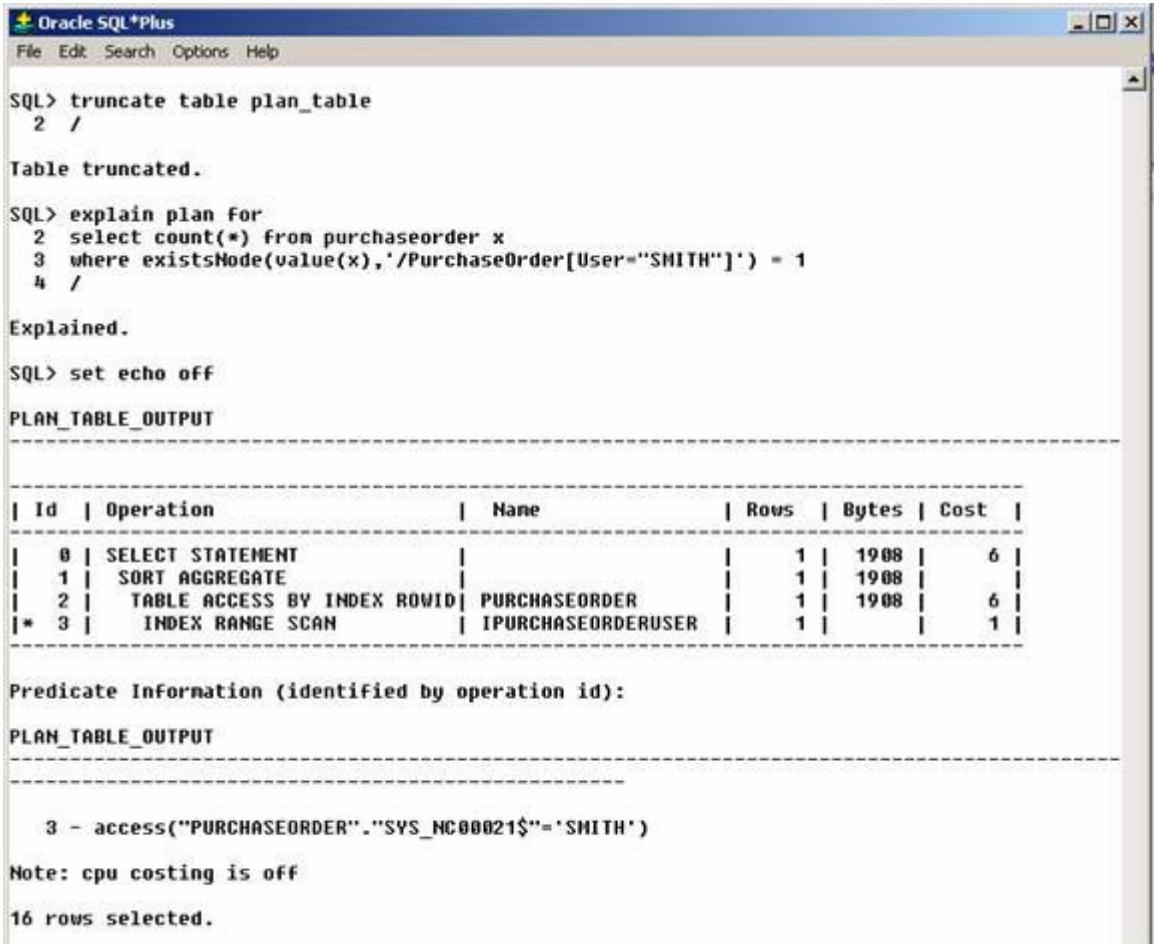
2. SQL> プロンプトに「QUIT」と入力して、SQL*Plus ウィンドウを閉じます。

4.4 実行計画を使用した、索引が使用されているかどうかの判断

この手順では、新しく作成された索引を使用して問合せが解決されることが示されます。

1. 「**4.4 Explain Plan (2)**」アイコンをクリックして、SQL スクリプトを実行します。

図 26-28 実行計画の実行による新しい索引が使用されているかどうかの判断



要点は次のとおりです。

- 新しい索引が自動的に使用されます。アプリケーションのリライトは必要ありません。
- データベース管理者にとって変更点はありません。必要な能力はこれまでと同じです。必要な索引を作成して、問合せを効率的に実行できるようにします。索引の使用を監視します。問合せパフォーマンスの向上に寄与していない索引を削除します。

2. SQL> プロンプトに「QUIT」と入力して、SQL*Plus ウィンドウを閉じます。

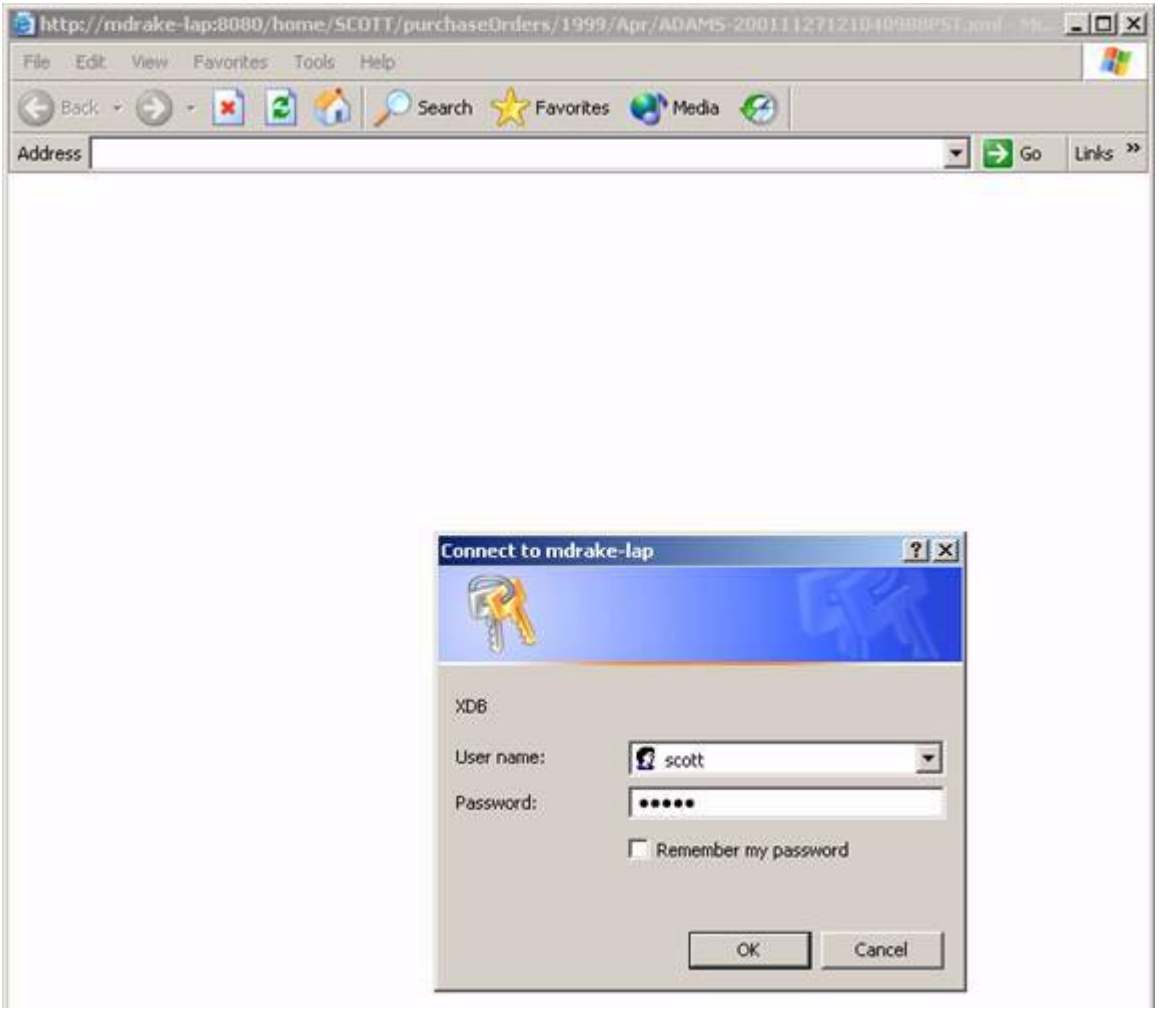
5.0 XML DB Demo: HTTP を使用した XML コンテンツへのアクセス

前述の手順では、FTP を使用して XML コンテンツを Oracle XML DB にロードする方法、および一般的な SQL の表 / 行隠喩を使用してコンテンツにアクセスする方法が示されました。この手順では、HTTP プロトコルを使用して、Oracle XML DB に格納されたコンテンツにパスベース（フォルダ / ファイル）の隠喩を使用してアクセスする方法が示されます。

参照： 第 4 章「XMLType の使用」を参照してください。

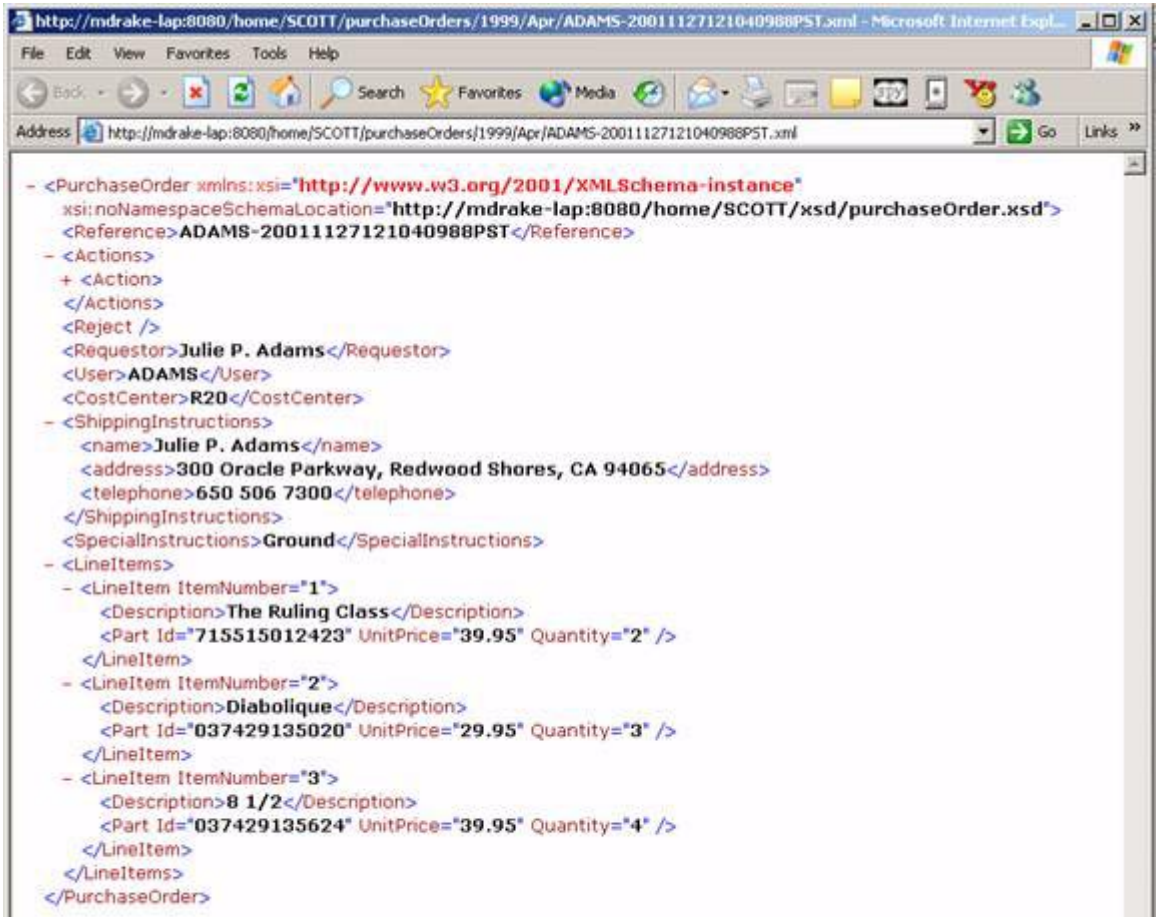
1. 「**5.0 ADAMS-2001...**」アイコンをクリックして、Internet Explorer を起動し、ターゲット・ドキュメントを表示します。データベース・ユーザー名およびパスワードを入力するためのプロンプトが表示された場合は入力します。

図 26-29 HTTP の使用 : XML コンテンツへのアクセス



PurchaseOrder 文書がブラウザに表示されます。図 26-30 を参照してください。

図 26-30 XML 文書 PurchaseOrder の表示



要点は次のとおりです。

- Oracle XML DB では、SQL 中心の表 / 行隠喩と文書中心のフォルダ / ファイル隠喩の両方を使用してコンテンツにアクセスできます。
- コンテンツには、単純な URL を使用して Web ブラウザから直接アクセスできます。
- Oracle XML DB は、HTTP プロトコルのネイティブなサポートを提供します。この機能を使用可能にするために、Web サーバー、プラグイン・テクノロジ、アダプタまたはコントロールは必要ありません。
- プロトコル・サポートは、必要に応じて無効にできます。

- プロトコル・サポートは、Oracle Net の共有サーバー・モード（Oracle リスナーおよび共有サーバー・モード・サーバー）と同じアーキテクチャに基づいています。

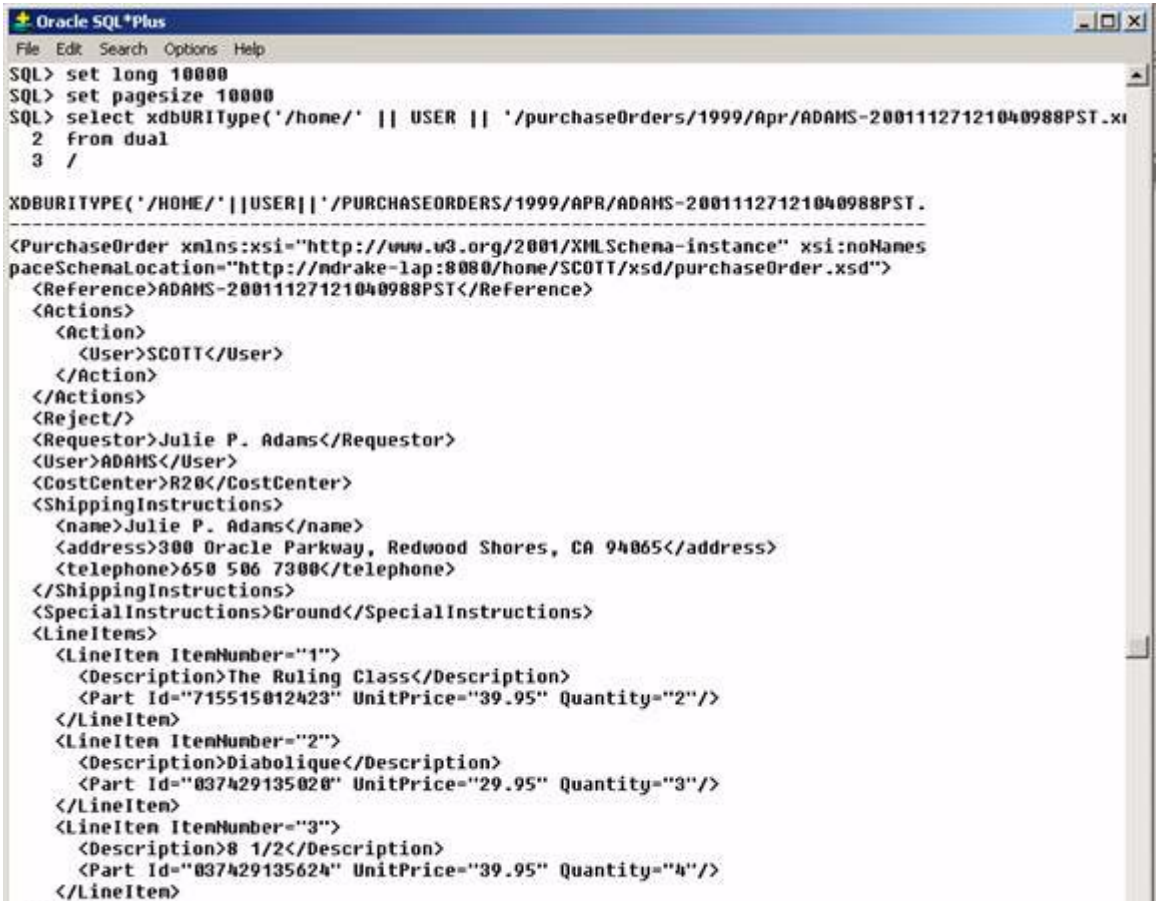
2. この時点では、このブラウザ・ウィンドウを閉じないでください。

5.1 XDBUri サーブレットを介して、取り出された XML 文書を表示できる SQL

この手順では、SQL で作業している場合でも、ファイル / フォルダ隠喩を使用してコンテンツにアクセスできることが示されます。

1. 「**5.1 Show Document(1)**」アイコンをクリックして、SQL スクリプトを実行します。

図 26-31 ファイル / フォルダの指定によるコンテンツへのアクセス



要点は次のとおりです。

- コンテンツへのパスベースのアクセスは、SQL からも可能です。`XDBUriType` を使用すると、パスベースの隠喩を使用して、Oracle XML DB Repository に格納されたコンテンツにアクセスできます。
- `XDBUriType` によって、様々なコンテンツにアクセスできる一連の方法が提供されます。`XDBUriType` に対して指定されたすべてのパスは、Oracle XML DB Repository のルートからのパスであると想定されます。

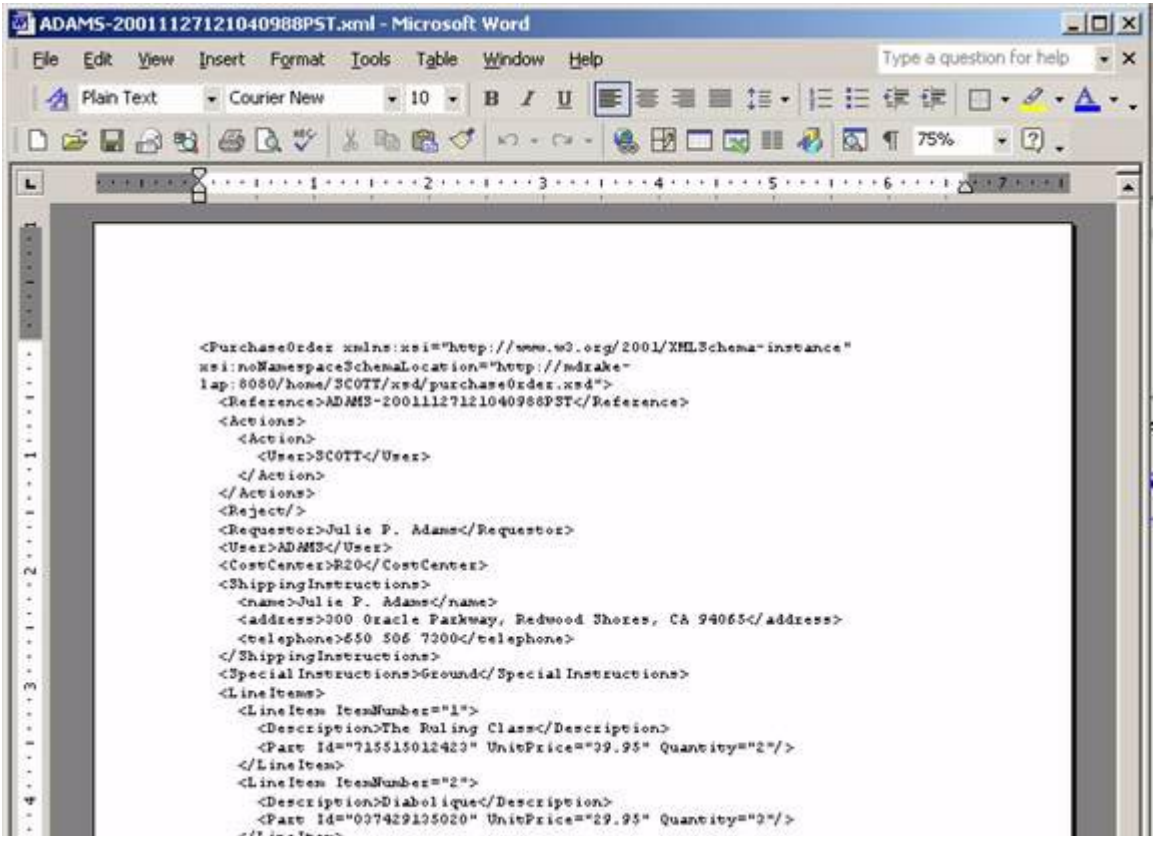
2. この時点では、この SQL*Plus ウィンドウを閉じないでください。

5.2 WebDAV 対応ツールを使用した XML 文書の編集

この手順では、Oracle XML DB の WebDAV サポート、および WebDAV 対応の標準のツールを使用して、リポジトリに格納されたコンテンツにアクセスおよび更新する方法が示されます。

1. 「**5.2 Edit Document**」アイコンをクリックして、Microsoft Word を起動し、ターゲット・ドキュメントを開きます。必要なユーザー名およびパスワードを入力するためのプロンプトが表示された場合は入力します。ファイル変換またはキャラクタ・セット変換を行うためのプロンプトが表示された場合は、Word が推奨するデフォルト値を選択します。文書が Word で表示されます。

図 26-32 Microsoft Word などの WebDAV ツールを使用した XML 文書の編集



Microsoft Word を使用して文書を編集します。
`/PurchaseOrder/Actions[1]/Action/User/text()` ノードの値を「VISHU」に更新します。これ以降のデモの手順はこの変更を正しく行ったかどうかに依存するため、値「VISHU」を入力するときは、慎重に行ってください。変更した文書を保存します。

注意： 現在、メモ帳またはワードパッドは使用できません。Word 2000 や Word XP など、WebDAV を認識するエディタを使用する必要があります。

要点は次のとおりです。

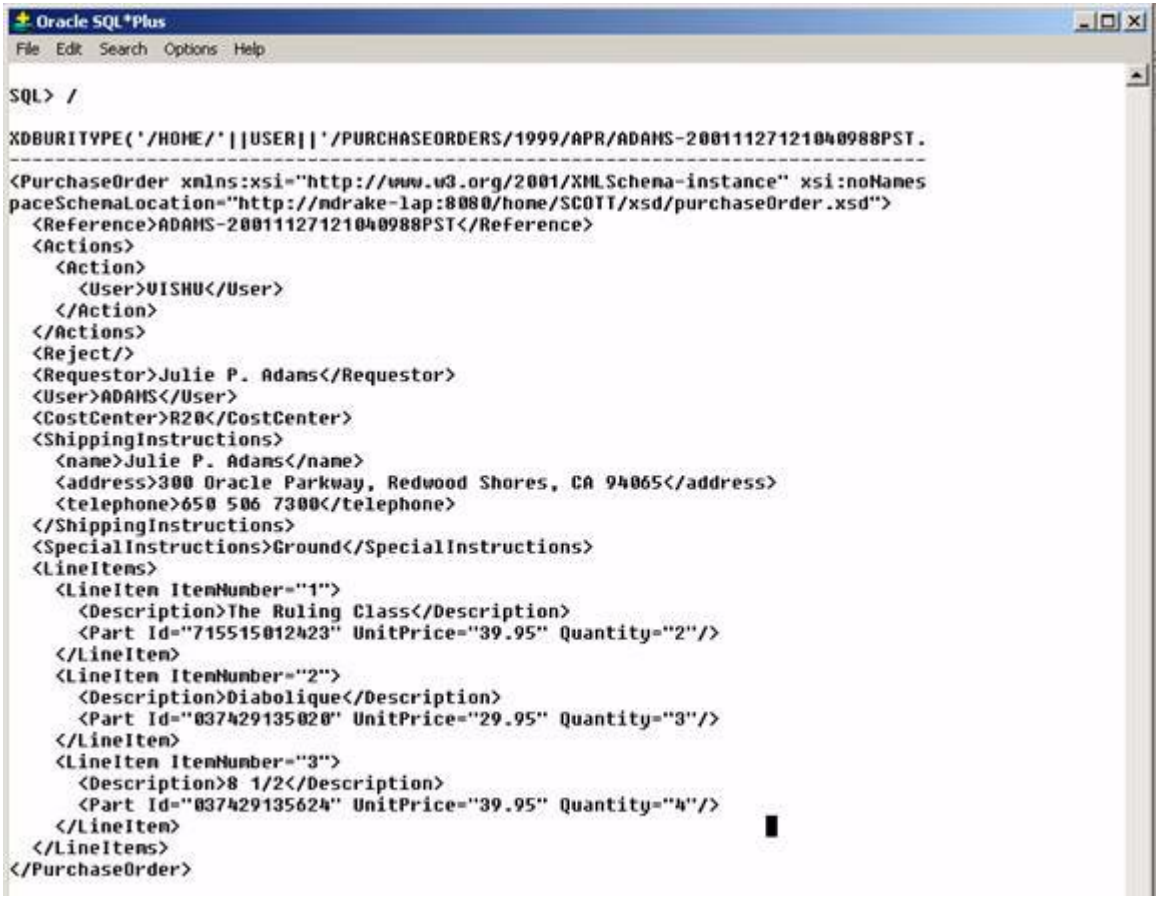
- Microsoft Word などの WebDAV 対応製品を使用して、Oracle XML DB Repository に格納されたコンテンツにアクセスおよび更新できます。オラクル社および Microsoft 社はオープンな業界標準でのサポートを選択しているため、これらの 2 製品は相互に機能します。
- Macromedia 社、Adobe 社、Altova 社などの他のベンダーは、WebDAV プロトコルのサポートを自社製品に取り込んでいます。これは、これらのすべての製品が Oracle XML DB Repository に格納されたコンテンツで機能することを意味します。

5.3 SQL を使用した、XML 文書の更新の表示および確認

この手順では、Microsoft Word を使用して行った変更を SQL から参照できることが示されます。

1. 手順 5.1 で表示されたウィンドウがまだ開いている場合は、「/」文字を入力して、問合せを再実行します。手順 5.1 で表示されたウィンドウが閉じている場合は、「**5.3 Show Document (2)**」アイコンをクリックして、問合せを実行します。

図 26-33 SQL でも参照可能な、Microsoft Word を使用して行った変更



要点は次のとおりです。

- Microsoft Word を使用して、Oracle XML DB に格納されたコンテンツを編集できます。
- Microsoft Word を使用して行った変更は、SQL からすぐに参照できます。DAV ベースのクライアントを使用して実行された各操作は、独立トランザクションです。

2. SQL> プロンプトに「QUIT」と入力して、SQL*Plus ウィンドウを閉じます。

5.4 SQL を使用した XML 文書の更新

この手順では、SQL を使用して XML 文書を更新する方法が示されます。この手順では、updateXML() 関数を使用して、XMLType として格納された XML 文書のコンテンツが更新されます。また、XPath 式を使用してターゲット・ノードが参照されます。

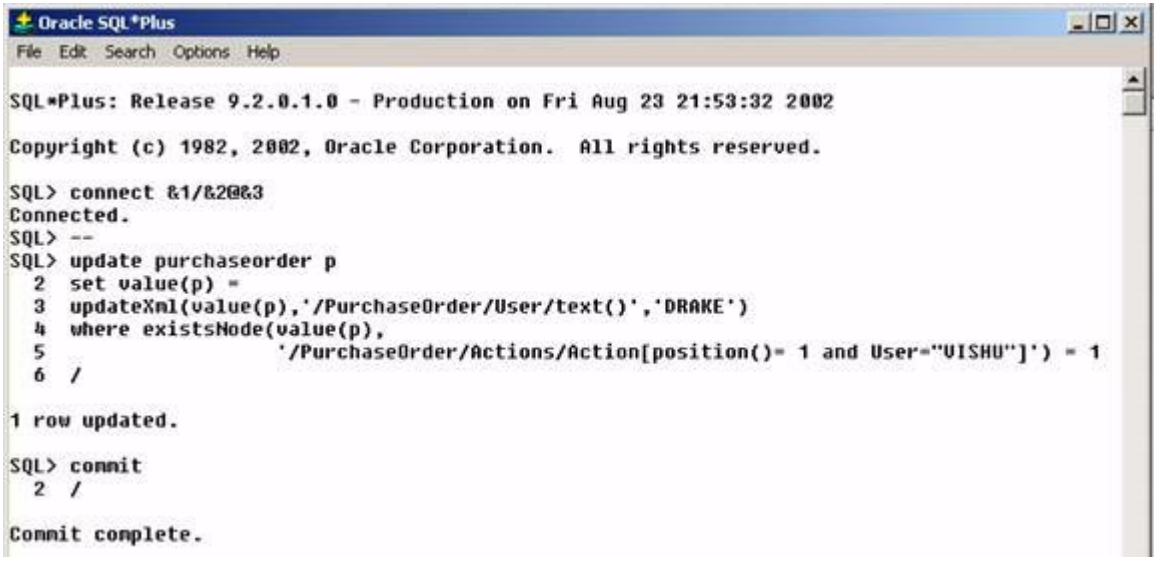
参照： 第 4 章「XMLType の使用」を参照してください。

注意： この手順を正常に動作させるには、手順 5.2 を正常に完了している必要があります。

---

1. 「**5.4 Update Document**」アイコンをクリックして、SQL スクリプトを実行します。

図 26-34 updateXML() および WHERE 句での XPath 式の使用による XML 文書の更新



要点は次のとおりです。

- updateXML() 関数を使用すると、XMLType として格納された XML 文書のコンテンツを更新できます。
- updateXML() では、XPath 式を使用して、更新する要素、属性またはノードが識別されます。

- updateXML() は、XML Schema に基づくコンテンツと基づかないコンテンツの両方で機能します。

  XML Schema に基づくコンテンツの場合、クエリー・リライトによって、updateXML() でインプレース更新を実行できるようになります。基礎となるいずれかの SQL オブジェクトの属性に XPath 式をマップできる場合、更新は SQL 操作として実行されます。

- updateXML() を使用すると、XML Schema に基づく文書をより効率的に更新できます。

  * Microsoft Word で文書を更新すると、Oracle XML DB は文書のどの部分が変更されたかを認識できません。そのため、強制的に文書全体が解析され、新しい文書に基づいてすべてのデータベース・オブジェクトが更新されます。

  * updateXML() を使用して文書を更新すると、文書の変更された部分のみが更新されます。

  * updateXML() を使用して XML Schema に基づかない XML を更新すると、DOM をインスタンス化し、その DOM に対して更新を実行することによって更新が行われます。その後、この DOM は出力され、基礎となる CLOB 記憶域に再書込みされます。

- updateXML() を使用して行った変更は、SQL を使用して行った他のすべての変更と同様です。これらの変更を他のデータベース・ユーザーが参照できるようにするには、それらをコミットする必要があります。

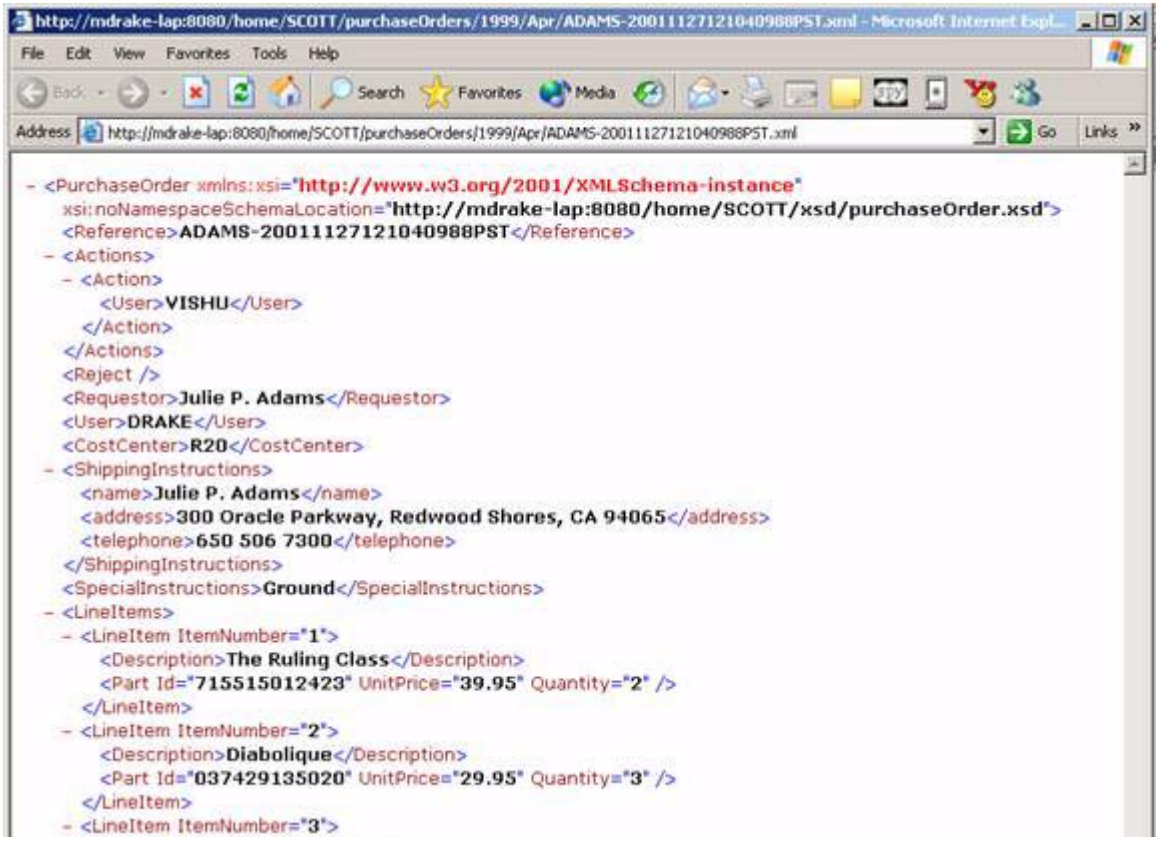
2. SQL> プロンプトに「QUIT」と入力して、SQL*Plus ウィンドウを閉じます。

5.5 XML と SQL の両方を使用して行った XML 文書の変更の表示

この手順では、SQL による方法と XML による方法の双対性が示されます。

1. 手順 5.0 で表示されたウィンドウがまだ開いている場合は、[Ctrl] キーを押したまま「更新」をクリックして、ブラウザのコンテンツを再ロードします。手順 5.1 で表示されたウィンドウが閉じている場合は、「**5.5 ADAMS-200111...**」アイコンをクリックして、ブラウザを開き、文書を表示します。

図 26-35 XML と SQL の相互運用性 : SQL を使用した XML コンテンツへのアクセス



要点は次のとおりです。

- Microsoft Word を使用して行った変更と SQL を使用して行った変更の両方を、更新されたページで参照できます。
- Oracle XML DB は、XML と SQL の完全な双対性および相互運用性を提供します。XML コンテンツは、文書中心のファイル / フォルダ隠喩と SQL 中心の表 / 行隠喩の両方を使用してアクセスおよび更新できます。
- いずれか一方の方法を使用して行われた変更は、そのトランザクションがコミットされた直後に、もう一方の方法でも使用できるようになります。

2. ブラウザ・ウィンドウを閉じます。

6.0 XML DB Demo: SQL を使用した RESOURCE_VIEW の問合せ

この手順では、Oracle XML DB Repository の詳細、および SQL プログラマが RESOURCE_VIEW を使用してリポジトリのコンテンツを問い合せる方法が示されます。

> **参照：** 第 15 章「RESOURCE_VIEW および PATH_VIEW」を参照してください。

要点は次のとおりです。

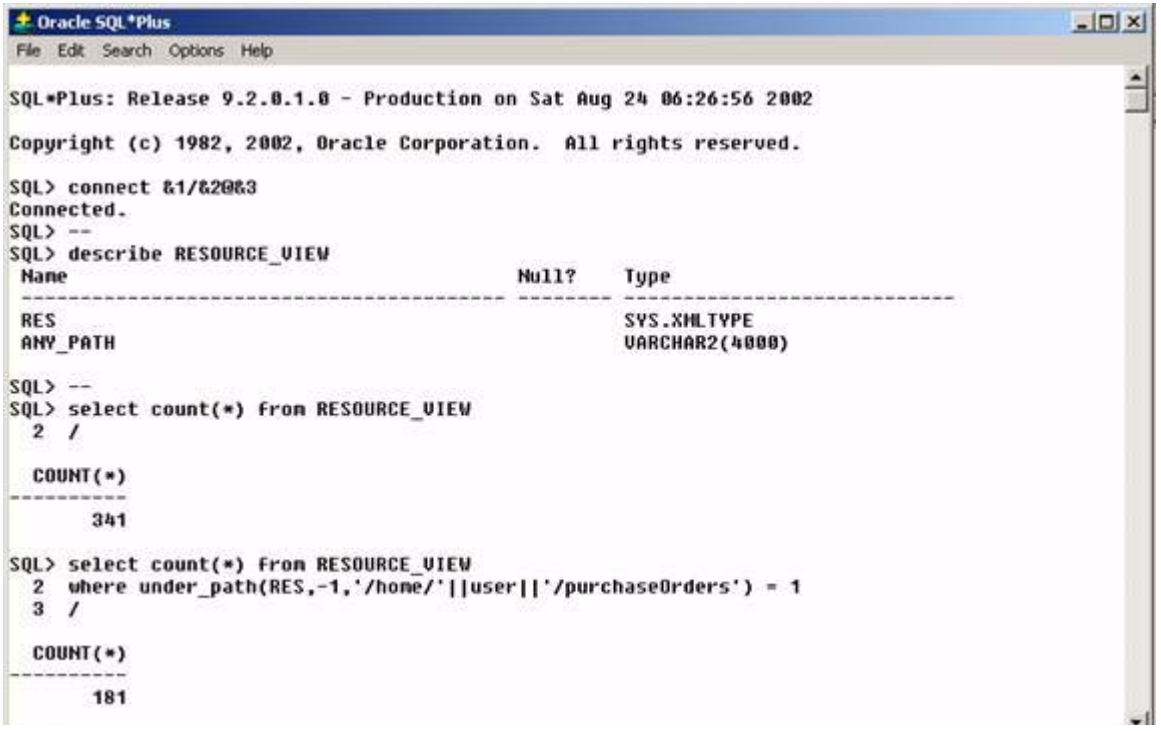
- Oracle XML DB Repository は、XML DB の XML Schema 内に存在する一連の表として管理されます。これらの表には、リポジトリに格納されたすべての文書のメタデータが含まれています。この XML DB の XML Schema は、ロックされたデータベース・アカウントです。この XML Schema 内の表には、直接アクセスできません。
- すべての非 XML 文書および XML Schema に基づかないすべての XML は、Oracle XML DB Repository の表に直接格納されます。
- XML Schema に基づく XML は、その XML Schema の登録時に指定されたデフォルトの表に格納されます。通常、これらの表は、その XML Schema を登録したユーザーが所有するデータベース・スキーマ内に存在します。
- Oracle XML DB Repository のコンテンツは、2 つのビュー（RESOURCE_VIEW および PATH_VIEW）を使用して SQL プログラマに公開されます。パブリック・シノニムによって、すべてのデータベース・ユーザーがこれらのビューを使用できるようになります。また、Oracle XML DB は、Oracle XML DB Repository のコンテンツに対する効率的なパスベースの問合せを可能にする一連の SQL 関数も提供します。
  - Oracle XML DB は、Oracle データベースの拡張索引作成機能を使用して、階層索引という新しいドメイン索引を定義します。
    - この索引を使用して、Oracle XML DB Repository に対するパスベースの問合せが効率的に解決されます。
    - 階層索引を使用すると、高コストな connect by 操作を使用せずにパスベースの問合せを解決できます。
- Oracle XML DB が保持するデフォルトのメタデータは、IETF の WebDAV 標準に準拠しています。
  - WebDAV では、WebDAV 準拠のサーバーが保持する必要がある一連のメタデータが定義されます。また、WebDAV クライアントと WebDAV サーバーが XML を使用して情報を交換することも定義されます。WebDAV サーバーがメタデータを永続的に保持する方法は定義されません。

  XML DB では、メタデータは Oracle XML DB の XML Schema XDBResource に準拠する一連の XML 文書として永続的に保持されます。
  - Oracle XML DB Repository 内のファイルまたはフォルダごとに 1 つのリソース・ドキュメントが存在します。

- Oracle XML DB Repository に格納されたドキュメントは、アクセス制御リスト（ACL）ベースのセキュリティ・メカニズムによって保護されます。ドキュメントにアクセスするには、そのドキュメントへの読込み権限以上が必要です。
  - Oracle XML DB における現在の ACL 実装は、WebDAV の ACL 仕様案に準拠していますが、その完全な実装ではありません。
  - SQL を使用してリポジトリにアクセスすると、ACL レベルのセキュリティがデータベースの行レベルのセキュリティ機能を使用して施行されます。
- XML DB リポジトリには、基礎的なバージョニング機能があります。リポジトリの将来のリリースでは、WebDAV のバージョニング仕様 DELTA-V が実装される予定です。

1. 「**5.0 Resource_Views Query (1)**」アイコンをクリックして、SQL スクリプトを実行します。

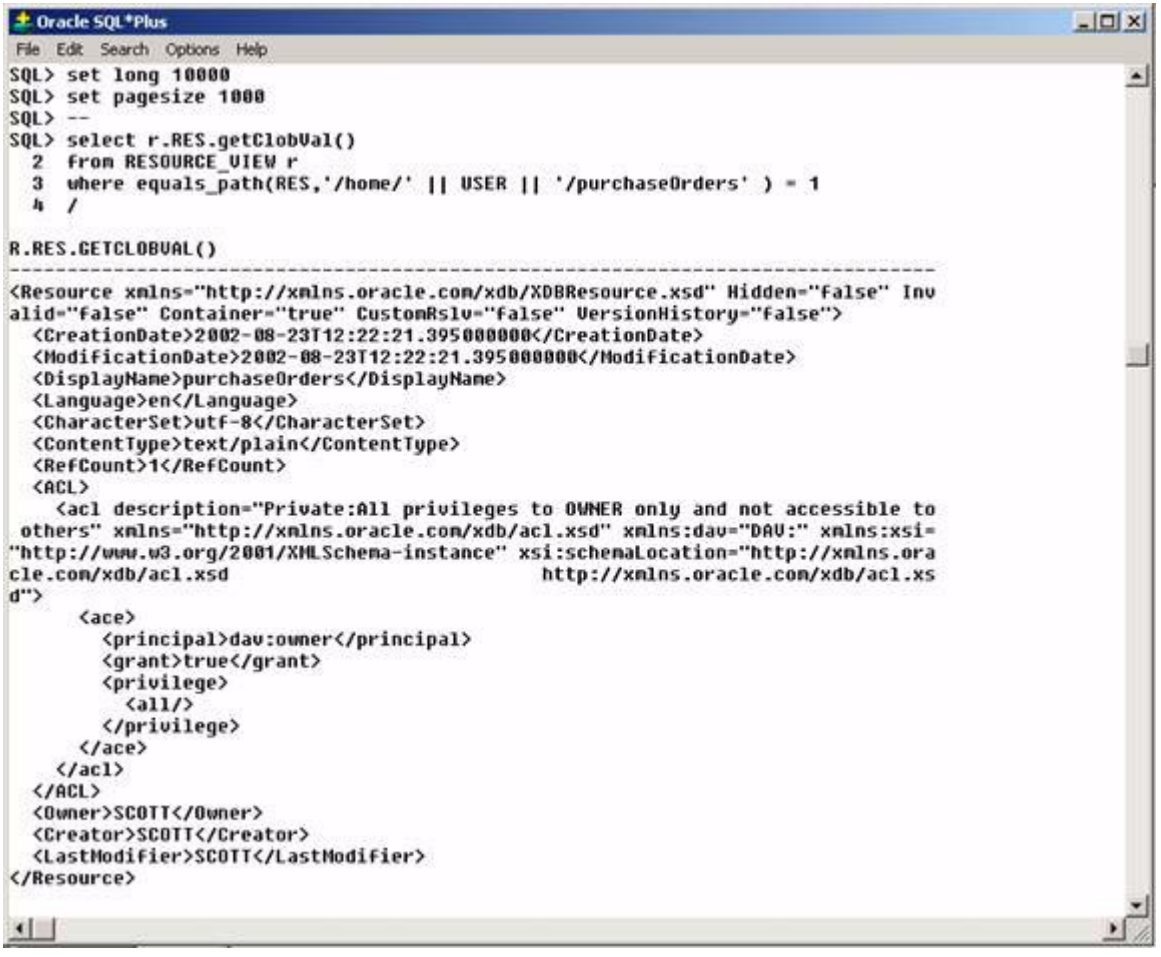
図 26-36 EQUALS_PATH() を使用した RESOURCE_VIEW の問合せ



要点は次のとおりです。

- RESOURCE_VIEW は、Oracle XML DB Repository の主パブリック・ビューを提供します。このビューには、リポジトリ内のドキュメントまたはフォルダごとに 1 行が含まれます。各行には、2 つの列（RES および ANY_PATH）が含まれます。
  - RES は、リポジトリに格納されたドキュメントに関するメタデータを含む XML 文書です。
  - ANY_PATH には、ドキュメントにアクセスするために使用可能な、リポジトリのルートからの有効なパスが含まれています。
- RESOURCE_VIEW および PATH_VIEW には、他のすべてのビューと同様にアクセスできます。たとえば、リポジトリ内のドキュメント数をカウントするには、RESOURCE_VIEW 内の行数をカウントします。
  - ACL ベースのセキュリティによって、「select count(*) from RESOURCE_VIEW」問合せで、ユーザーがアクセス権を持つドキュメントの数が戻されることが保証されます。
- UNDER_PATH などの関数を使用すると、問合せを効率的にリポジトリの特定のサブツリーに限定することが容易になります。

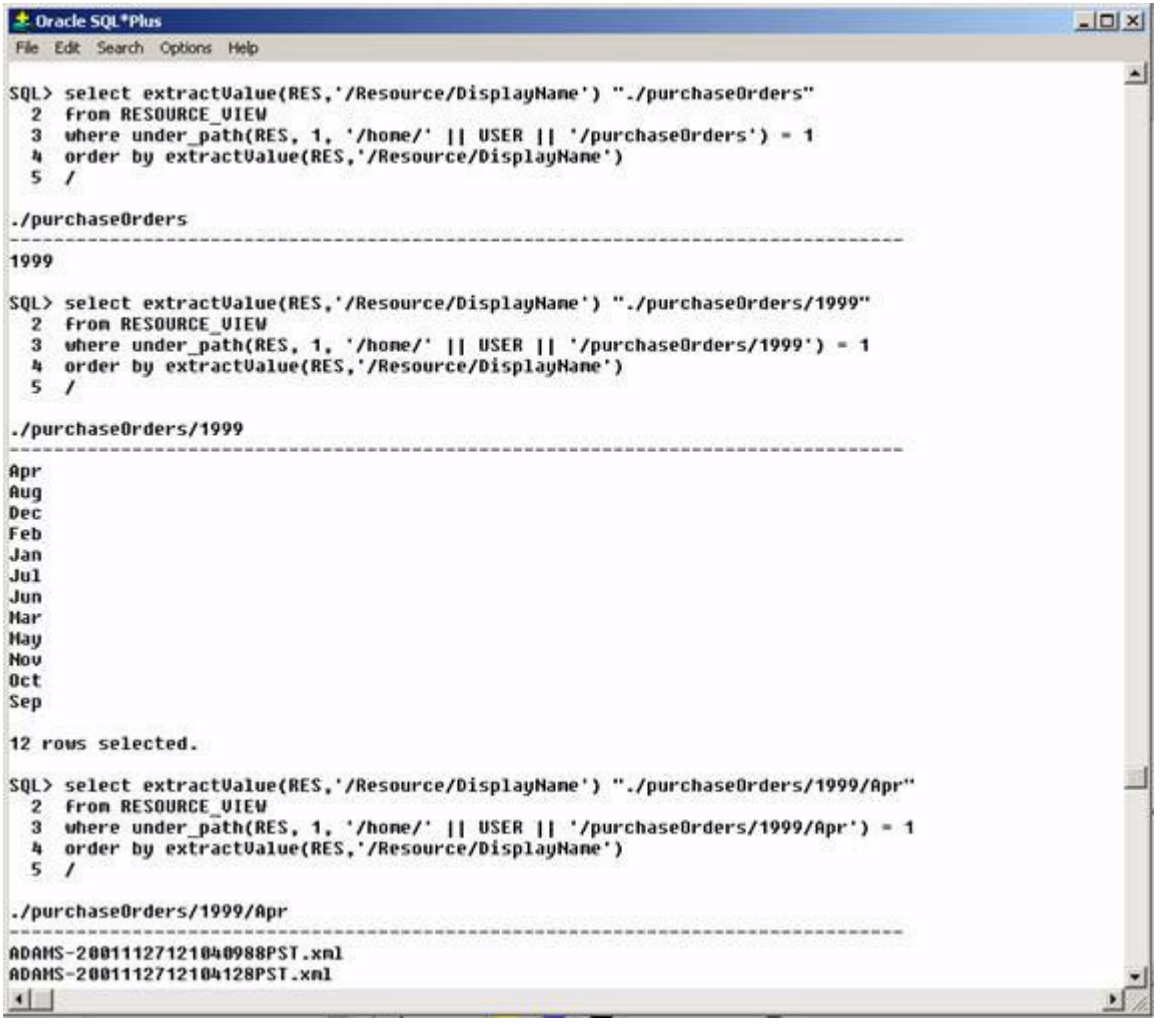
図 26-37 EQUALS_PATH() を使用した、RESOURCE_VIEW を介した purchaseOrders フォルダに関するメタデータの取出し



要点は次のとおりです。

- RES 列には、XML Schema XDBResource に準拠する XML 文書が含まれます。この XML Schema では、IETF の WebDAV 標準を実装するために必要な一連のメタデータが定義されます。
- この例では、EQUALS_PATH() 関数を使用して、ユーザーの purchaseOrders フォルダ用のメタデータを取り出しています。
- 使用可能なメタデータには、DisplayName、Creator、Owner、LastModifier、CreationDate、ModificationDate などの項目が含まれています。
- このリソースには、Oracle XML DB に格納された他のすべての XML 文書と同様にアクセスできます。extractValue() および existsNode() を使用して、RESOURCE_VIEW に対する問合せを実行できます。

図 26-38 Using extractValue() および UNDER_PATH() を使用した、purchaseOrders フォルダからのディレクトリ・ツリー内の移動



要点は次のとおりです。

- extractValue() を使用して、メタデータにアクセスできます。ドキュメントの所有者の変更など、メタデータの更新を伴う操作は、updateXML() を使用して実行できます。
- リポジトリで管理される一連のリソースに対する問合せは、existsNode()、UNDER_PATH()、EQUALS_PATH() などの関数を使用して実行できます。

この例では、extractValue() および UNDER_PATH() を使用して、ユーザーの purchaseOrders/ フォルダからディレクトリ・ツリー内を移動しています。

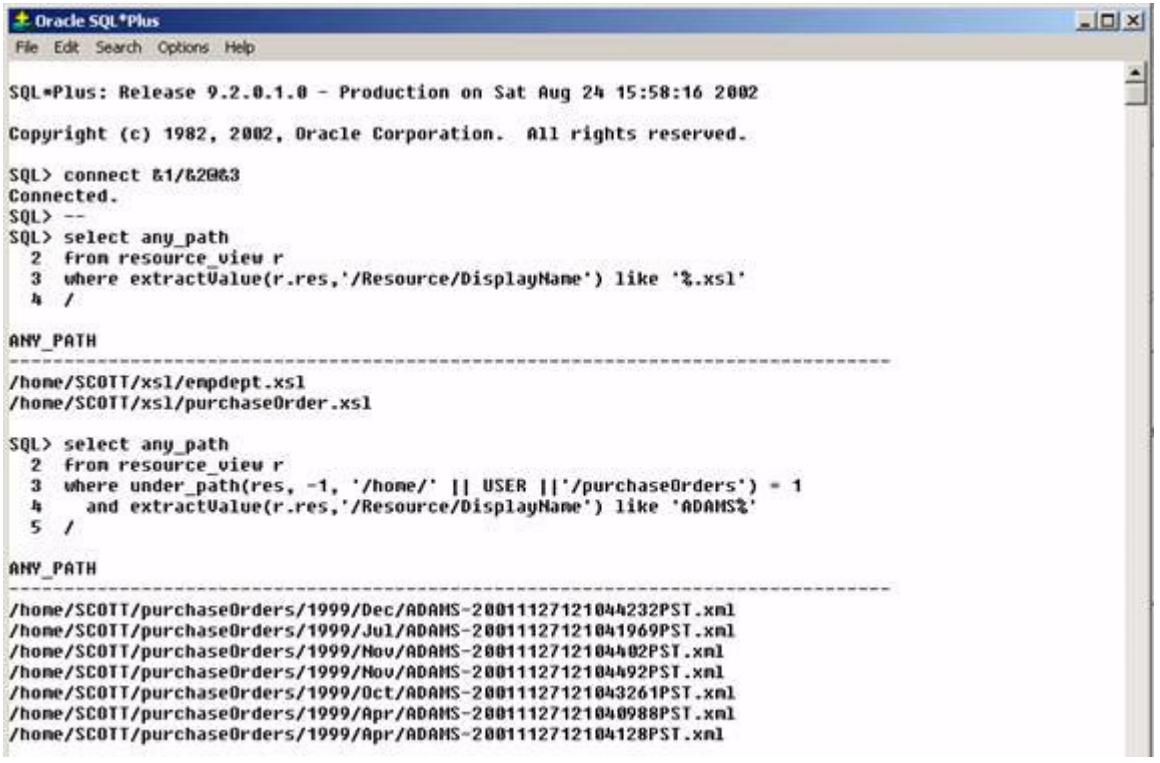
1. SQL> プロンプトに「QUIT」と入力して、SQL*Plus ウィンドウを閉じます。

6.1 階層的な索引付けを使用した RESOURCE_VIEWS の XPath ベースの問合せ

この手順では、XML DB リポジトリに対する問合せを行い、階層索引機能を使用して効率的にパスベースの問合せを解決する方法が示されます。

1. 「**6.1 Resource View Queries (2)**」アイコンをクリックして、SQL スクリプトを実行します。

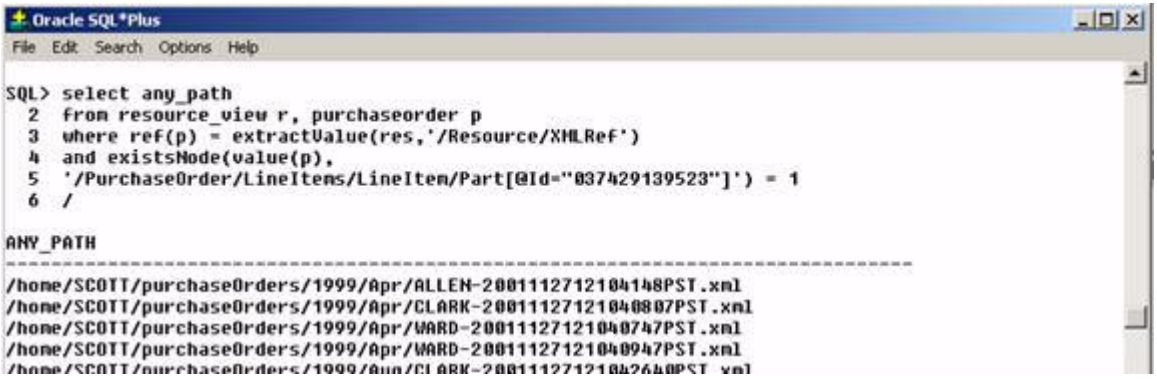
図 26-39 標準 SQL 構文を使用したリポジトリの検索



要点は次のとおりです。

- 標準 SQL 構文および演算子を使用して、リポジトリを検索できます。
- 最初の例は、SQL に類似した演算子を使用して、すべての XSL ドキュメント（ファイル拡張子 .xsl が付いたドキュメントなど）を検索する方法を示しています。
- 2 番目の例は、「ADAMS」で始まるすべての文書を検索し、その結果を `UNDER_PATH()` 関数を使用してユーザーの `purchaseOrders` フォルダに存在するドキュメントに限定する、より複雑な `WHERE` 句を示しています。

図 26-40 XML 表と結合された RESOURCE_VIEW の問合せによるメタデータの取出し

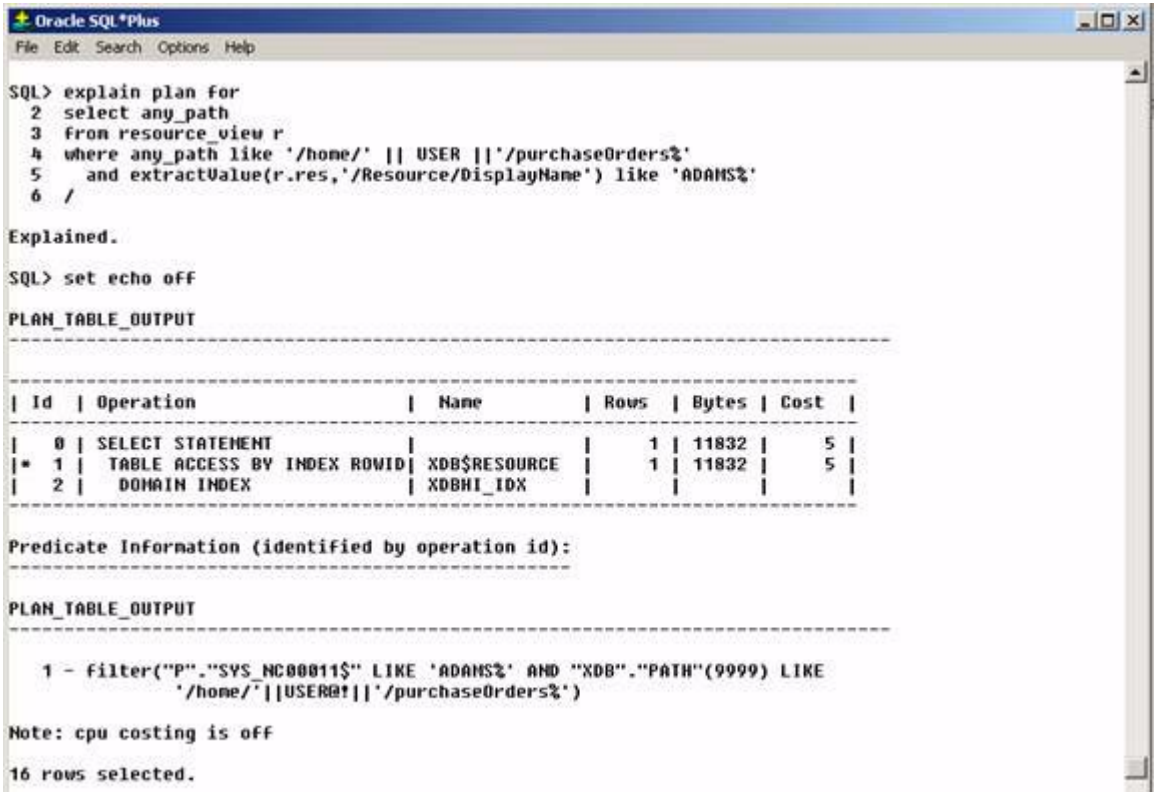


要点は次のとおりです。

- Oracle XML DB は、システムで保持されるメタデータ値（Owner や ModificationDate など）および XML コンテンツに対する操作を伴う問合せをサポートします。これは、XML コンテンツを含む表と RESOURCE_VIEW を結合することによって実現されます。
- この例では、PURCHASEORDER 表を RESOURCE_VIEW と結合して、問合せによって戻された表の各行のパスを取得しています。
  - この方法では、問合せによって戻された XMLType 表の各行の URL が効率的に生成されます。
  - この方法を Oracle XML DB の HTTP リクエスト処理機能と組み合せると、Web ベースのアプリケーションの作成プロセスが大幅に簡略化されます。

2. SQL> プロンプトに「QUIT」と入力して、SQL*Plus ウィンドウを閉じます。

図 26-41 階層索引によってパスベースの問合せが効率的に解決されたことを示す実行計画



要点は次のとおりです。

- この実行計画の出力は、階層索引を使用して、パスベースの問合せが connect-by 処理を使用せずに効率的に解決されていることを示しています。

7.0 XML DB Demo: ビューを使用した、関連ツールから XML へのアクセス

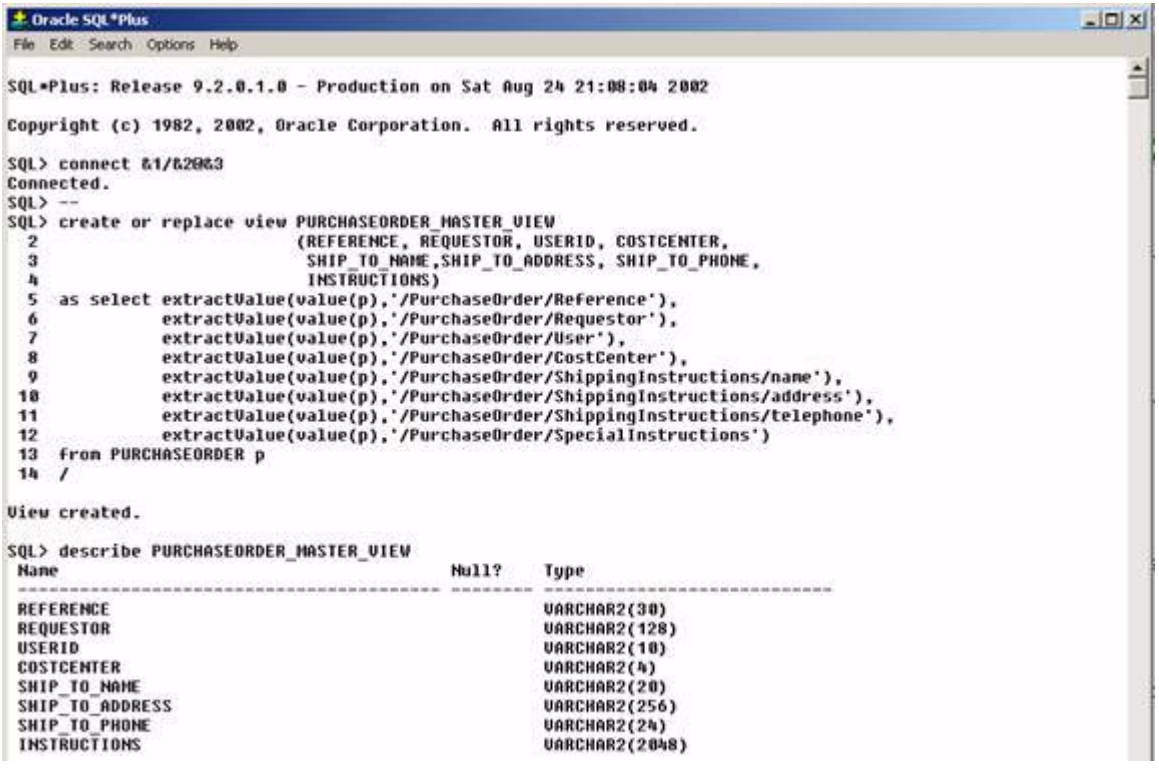
この手順では、データのリレーショナル・ビューのみを認識するツールおよび製品が Oracle XML DB で管理される XML コンテンツにアクセスする方法が示されます。

要点は次のとおりです。

- リレーショナル・データにアクセス可能なツール（およびアプリケーション開発者）と製品は数多く存在しますが、それらはデータが XML ファイルに格納される方法を認識しません。
- Oracle XML DB では、XML コンテンツのリレーショナル・ビューを定義でき、これらのツールで XML コンテンツを使用できます。

1. 「7.0 Make Views」アイコンをクリックして、SQL スクリプトを実行します。

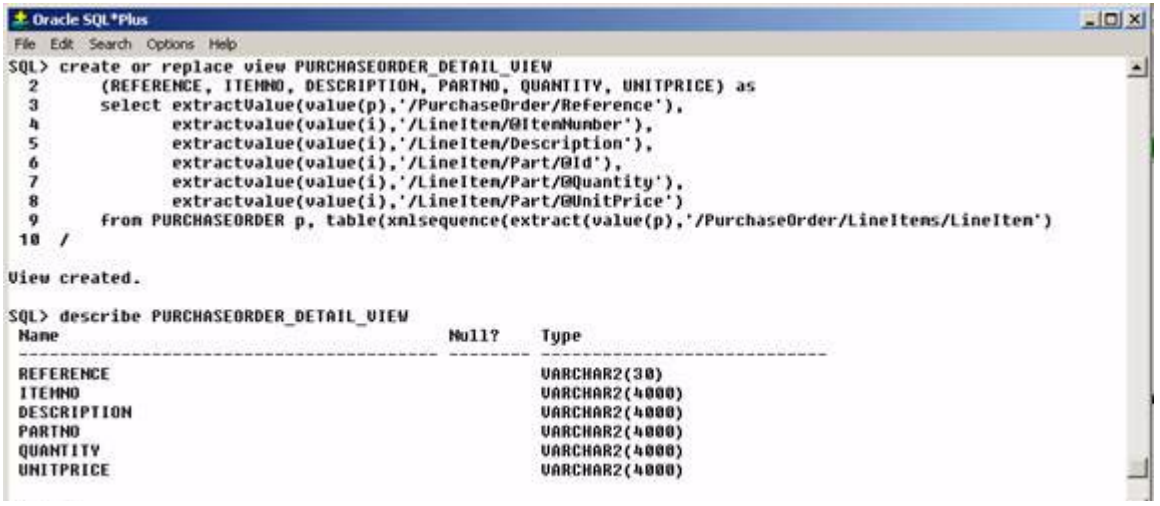
図 26-42 CREATE VIEW および XPath 式を使用したビューの作成



要点は次のとおりです。

- ビューは、単純な CREATE VIEW 文を使用して作成されます。この文では、XPath 式を使用して、XML 文書内のテキスト・ノードまたは属性値が CREATE VIEW 文で宣言された列にマップされます。
- この例では、PURCHASEORDER 表のドキュメントごとに 1 行を含む PURCHASE_ORDER_MASTER_VIEW が作成されています。

図 26-43 PurchaseOrder 表のドキュメントごとに 1 行を含む Purchase_Order_Master_View の作成



要点は次のとおりです。

- リレーショナル・ビューを使用して、要素のコレクションのメンバーを一連の行として公開することもできます。
- この例では、lineitem 要素のコンテンツを一連の行として公開する PURCHASE_ORDER_DETAIL_VIEW というビューが作成されています。このビューには、PURCHASEORDER 表の lineitem 要素ごとに 1 行が含まれます。
- 最初の手順では、extractValue() 関数を使用して、PURCHASEORDER 表の各ドキュメントから XML フラグメントが生成されます。XML フラグメントは、複数のルート・レベル・ノードを含む XML 文書です。この場合、XML フラグメントは一連の lineitem ノードで構成されます。そのフラグメントには、lineitem コレクションのメンバーごとに 1 つのルート・レベル・ノードが含まれます。
- 次の手順では、XMLSequence() 関数を使用して、フラグメントの各ルート・レベル・ノードから個別の行が作成されます。

- 最後の手順では、SQL 演算子 TABLE を使用して、これらの一連の行が SELECT 文の FROM 句で使用できる表に変換されます。
- PURCHASEORDER 表と、TABLE 演算子によって生成された一連の行の間には、暗黙的な相関結合が存在します。

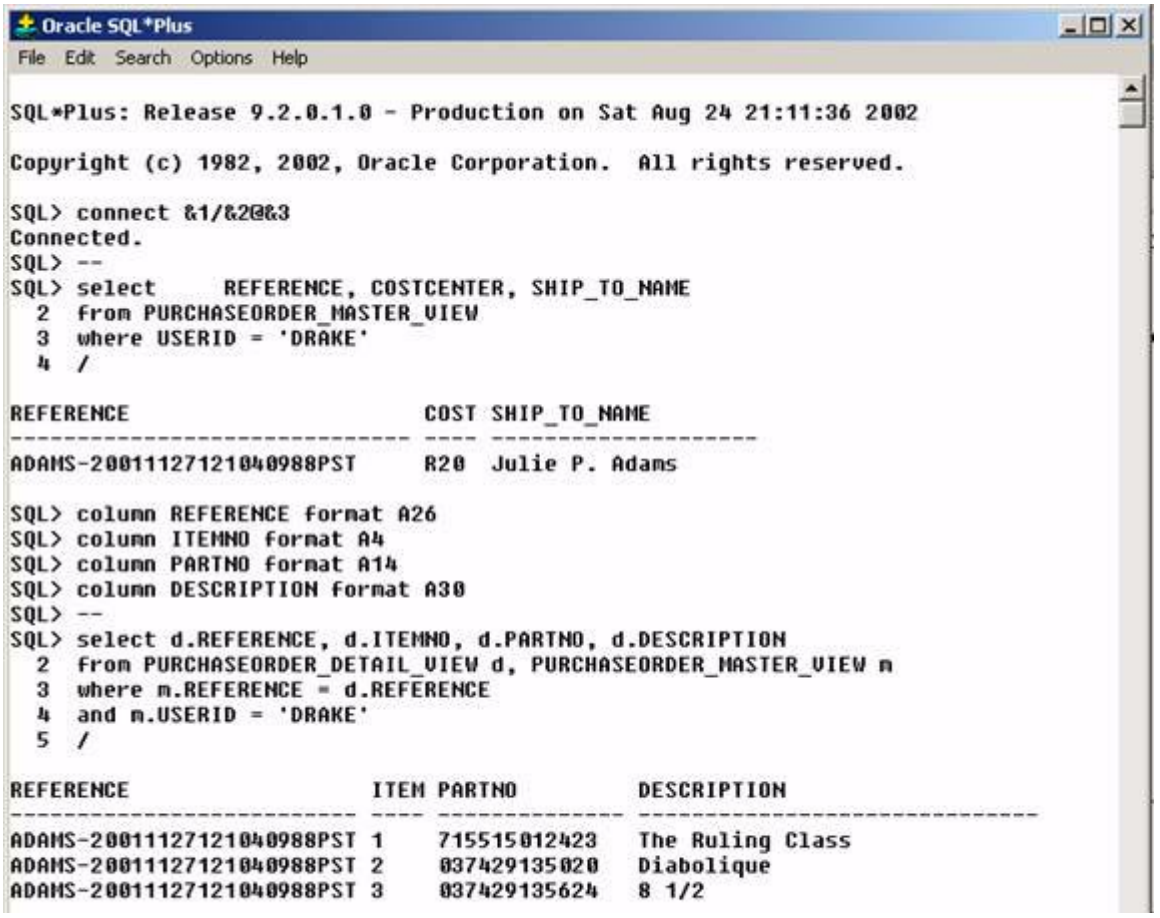
2. SQL> プロンプトに「QUIT」と入力して、SQL*Plus ウィンドウを閉じます。

7.1 他のビューと同様に動作する XML のリレーショナル・ビュー

この手順では、XML のリレーショナル・ビューが他のデータのリレーショナル・ビューと同じように見え、同様に動作することが示されます。

1. 「**7.1 Query Views**」アイコンをクリックして、SQL スクリプトを実行します。

図 26-44 標準 SQL を使用した XML のリレーショナル・ビューの問合せ



要点は次のとおりです。

- これらは、単純で基本的な SQL 問合せです。開発者やエンド・ユーザーは、これらのビューのコンテンツが XMLType 表からのものであることを認識している必要はありません。問合せを作成するツールやユーザーは、XML コンテンツにアクセスするために必要な XML 固有の演算子および構文を理解している必要はありません。
- クエリー・リライトによって、基礎となるオブジェクト・ストアに対して SQL 問合せが実行されることが保証されます。

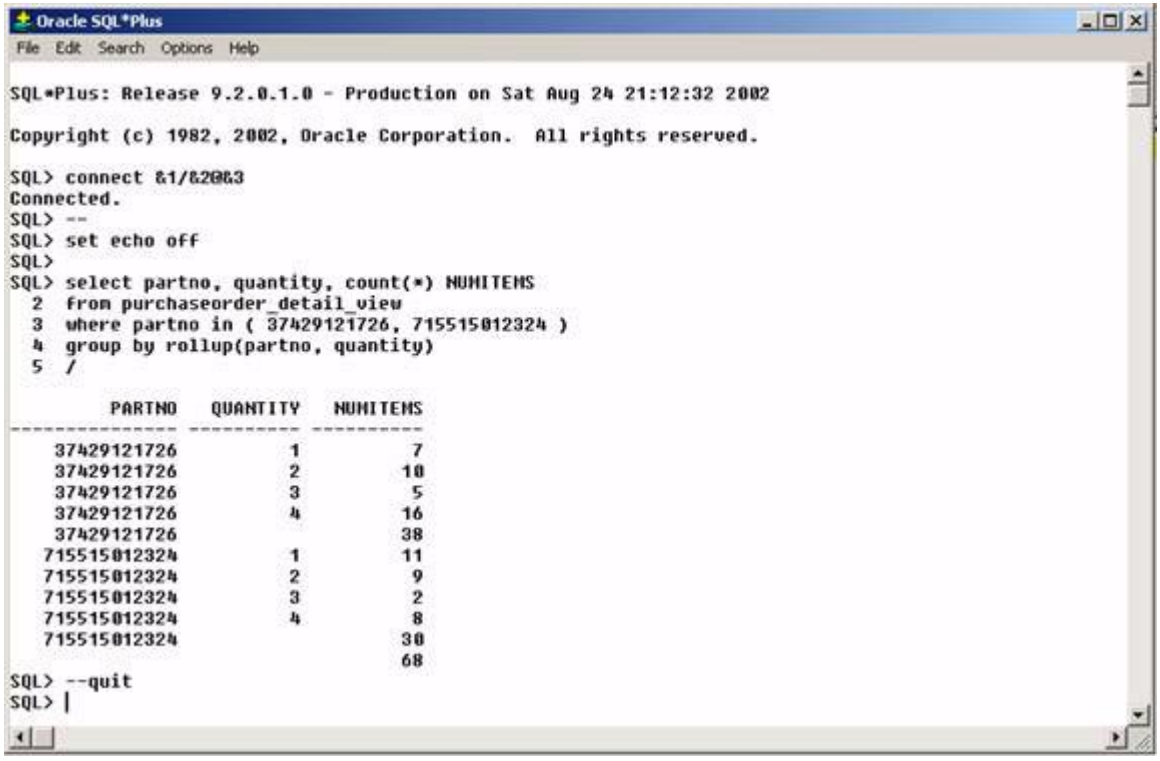
2. SQL> プロンプトに「QUIT」と入力して、SQL*Plus ウィンドウを閉じます。

7.2 ロールアップを使用した問合せ

この手順では、XML のリレーショナル・ビューを使用すると、Oracle XML DB で管理されるコンテンツに対して Oracle データベースの任意の SQL ベース機能を使用できることが示されます。

1. 「**7.2 Rollup Query**」アイコンをクリックして、SQL スクリプトを実行します。

図 26-45 XML コンテンツへのロールアップ問合せの適用



要点は次のとおりです。

- Oracle XML DB では、ロールアップ問合せなどの高度な SQL 機能を XML コンテンツに適用することが可能になります。
- ロールアップ機能自体は XML を認識しませんが、リレーショナル・ビューの作成機能によって、このような機能を XML コンテンツに使用できるようになります。

2. SQL> プロンプトに「QUIT」と入力して、SQL*Plus ウィンドウを閉じます。

8.0 XML DB Demo: DBUri サーブレットを使用したコンテンツへのアクセスおよび XSL を使用したコンテンツの変換

この手順では、DBUri サーブレットを使用して XML Schema/ 表隠喩によってコンテンツにアクセスする方法が示されます。また、Oracle XML DB の XSLT 変換実行機能も示されます。

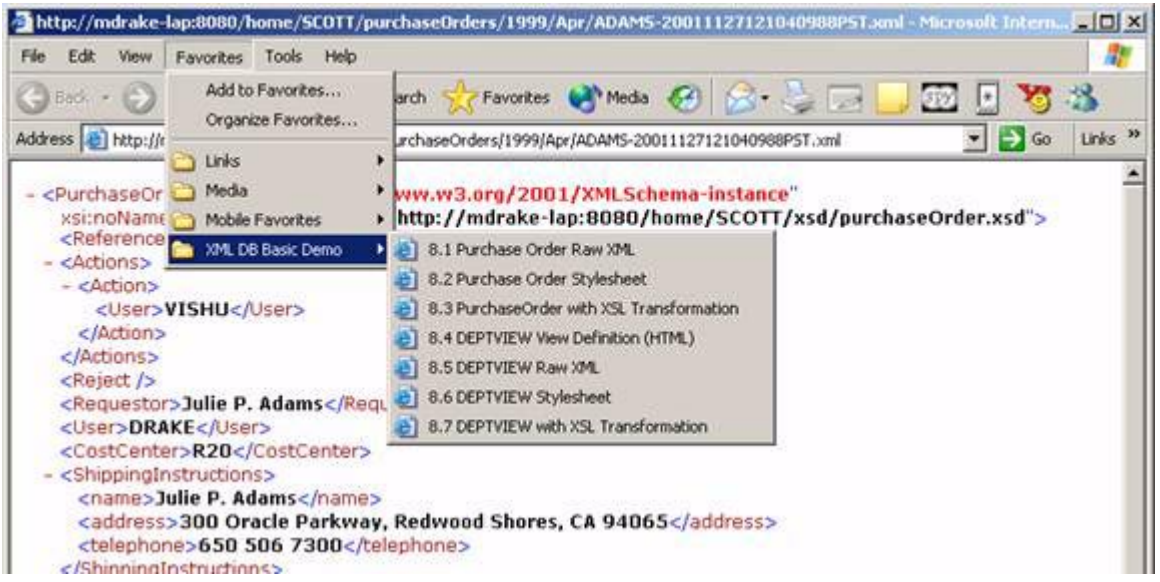
1. 「**8.0 DBUri and XSL Examples**」アイコンをクリックして、Internet Explorer を起動します。ユーザー名およびパスワードを入力するためのプロンプトが表示された場合は、データベース・ユーザーのユーザー名およびパスワードを入力します。

   Internet Explorer が起動され、リポジトリベースの URL を使用してドキュメント ADAMS-20011127121040988PST.xml のコンテンツが表示されます。この URL は、Oracle XML DB の HTTP サーバーを使用して、Oracle XML DB Repository 内のリソースに基づいてコンテンツを表示します。

2. 「**お気に入り**」をクリックし、「**XML DB Basic Demo**」をクリックします。

   デモの次のフェーズ中に使用する一連のインターネット・ショート・カットが表示されます。

図 26-46 Internet Explorer の起動およびリポジトリベースの URL による XML コンテンツの表示



要点は次のとおりです。

- コンテンツは、URL に基づいて表示されます。この URL は、フォルダ / ファイル隠喩を使用して、要求されたコンテンツを指すリソースを識別します。

3. この時点では、このブラウザ・ウィンドウを閉じないでください。

8.1 PurchaseOrder Raw XML

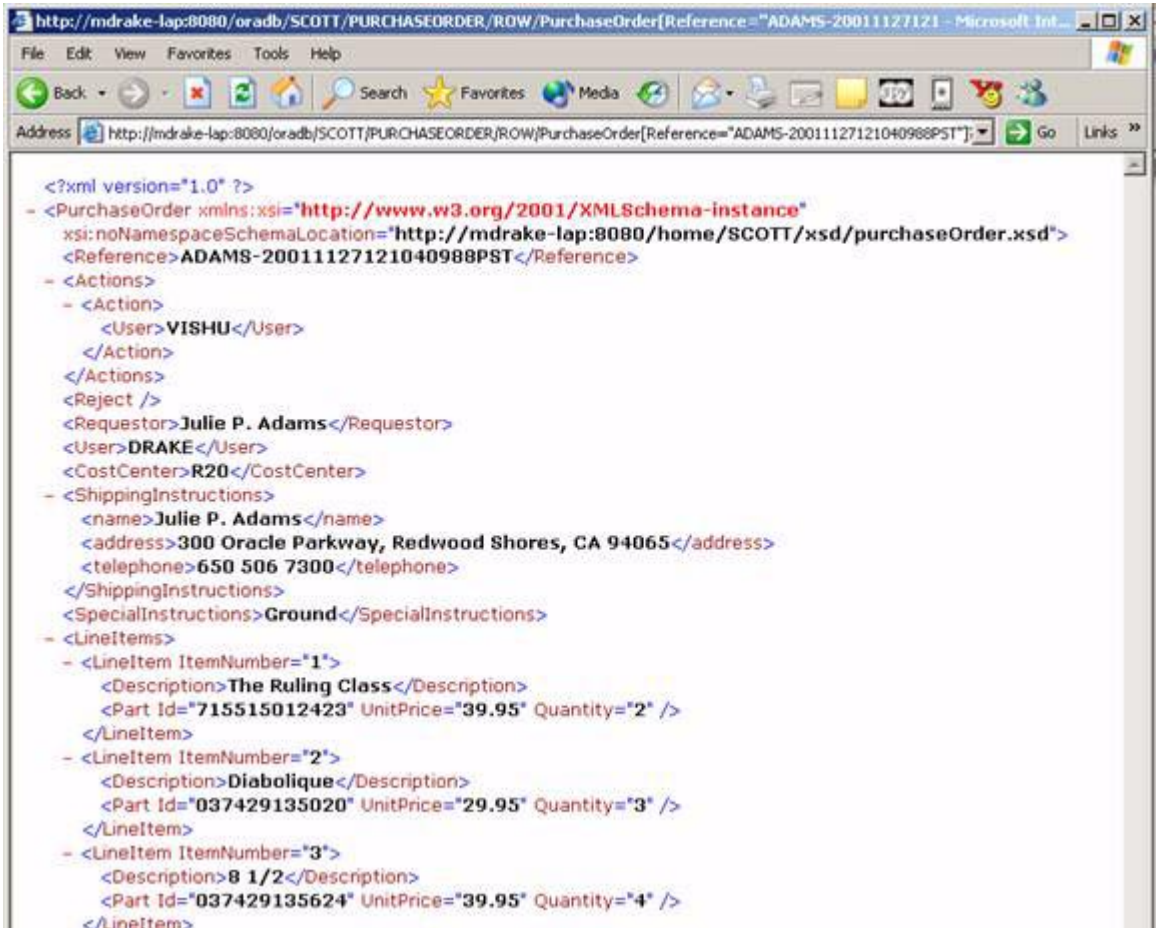
この手順では、Oracle XML DB の `DBUri` サーブレットが示されます。

> **参照：** 第 12 章「URL を介したデータの作成およびアクセス」を参照してください。

1. 「**お気に入り**」メニューの「XML DB Basic Demo」項目を開き、「**8.1 PurchaseOrder Raw XML**」項目を選択します。これによって、同じ発注書が表示されます。ただし、この文書は次のような DBUri ベースのパスを使用して識別されます。

```
http://mdrake-lap:8080/oradb/SCOTT/PURCHASEORDER/ROW
/PurchaseOrder[Reference="ADAMS-20011127121040988PST"]
?contenttype=text/xml
```

図 26-47 DBUri サーブレットおよび DBUriType を使用した、URL に基づいた XML 文書の行の選択



要点は次のとおりです。

- DBUri サーブレットは、Oracle XML DB の DBUriType 機能を使用します。DBUriType を使用すると、XML Schema、表、行および列で構成される URL を使用して表の行を識別できます。XPath に類似した構文を使用すると、この URL をサブセットに拡張できます。この場合、ターゲット表の行はその URL と一致します。

  * DBUriType は、選択された 1 つ以上の行を XML 文書として戻します。

    - DBUri サーブレットは、Oracle XML DB のネイティブな HTTP 機能を使用して、ブラウザで DBUri を使用してデータベースの任意の行にアクセスできるようにします。

- この例は、DBUri を使用して PurchaseOrder XMLType 表の行にアクセスする方法を示しています。URI は次の構成要素で構成されます。
    - /oradb: DBUri サーブレットのデフォルトのマウント・ポイント。
    - /SCOTT: データベース・スキーマ名。
    - /PURCHASEORDER: 表の名前。
    - /ROW: デフォルトの行セパレータ。
    - /PurchaseOrder: 対象のドキュメントのルート・ノード。
    - [Reference="ADAMS-20011127121040988PST"]: 戻す 1 つ以上の行を決定する XPath 式。
    - ?contenttype=text/xml: contenttype パラメータを使用すると、開発者はブラウザに戻す MIME タイプを指定できます。
- XMLType 表またはビューの場合、DBUri サーブレットでは、XPath 式を使用して、戻す表の行を決定できます。これは、機能上、W3C の XPointer 勧告に類似しています。
- リレーショナル表またはビューの場合、DBUri サーブレットでは、表の列に基づく、XPath に類似した式を使用して、結果のドキュメントに含める行を決定できます。

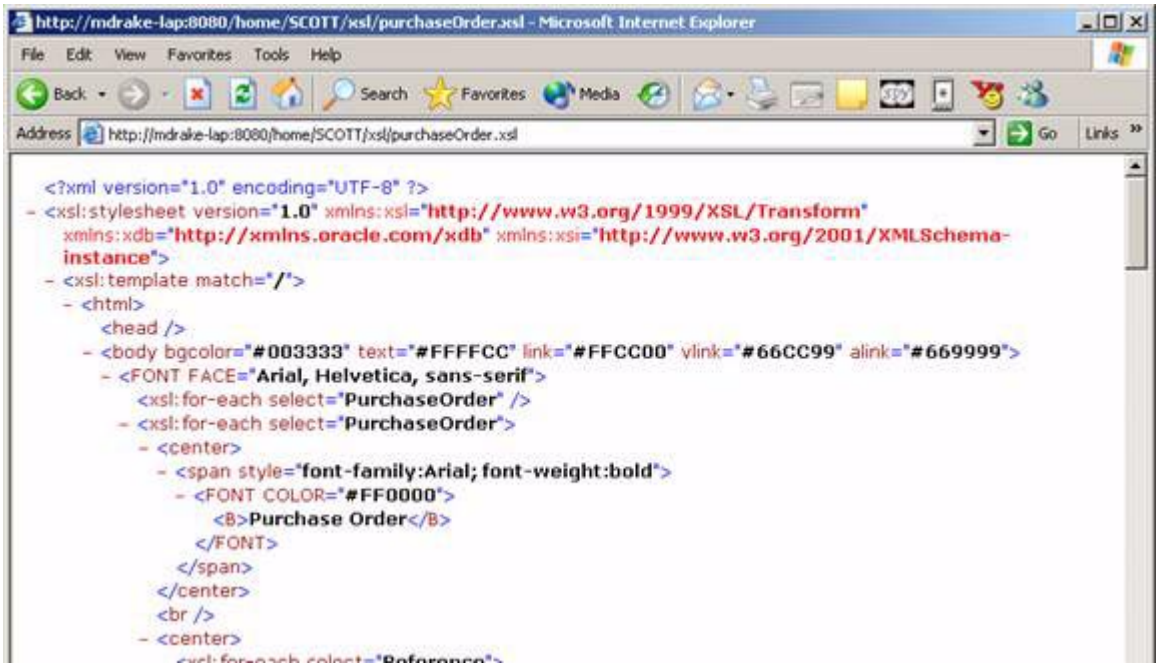
2. この時点では、このブラウザ・ウィンドウを閉じないでください。

8.2 標準の XSLT スタイルシートを使用した HTML への XML 文書の変換

この手順では、PurchaseOrder 文書を XML から HTML に変換するために使用可能な標準のスタイルシートが示されます。

1. 「お気に入り」メニューの「XML DB Basic Demo」項目を開き、「**8.2 PurchaseOrder XSL Style sheet**」項目を選択します。これによって、XML 文書 PurchaseOrder で使用可能な XSLT スタイルシートが表示されます。

図 26-48 XML 文書 PurchaseOrder で使用するためのスタイルシート



要点は次のとおりです。

- ほとんどのエンド・ユーザーは、XML を扱う必要がなく、情報を HTML 形式で参照することを希望します。XML 文書を HTML に変換するための標準メカニズムは、W3C の XSLT 勧告に準拠するスタイルシートです。
- このスタイルシートは、W3C の標準 XSLT スタイルシートです。このスタイルシートについては、Oracle 固有のものはありません。
- XML から HTML を作成するには、XSLT プロセッサが必要です。XSLT プロセッサは、XML およびスタイルシートに含まれる命令を受け入れ、それらを使用して HTML を生成します。

  XSL によって、表示ロジックを処理ロジックから分離することが可能になります。異なるスタイルシートを使用すると、任意の XML 文書を異なる方法でフォーマットできます。たとえば、あるスタイルシートでは文書を PC ブラウザで表示するためにフォーマットし、別のスタイルシートでは同じ文書を WAP 対応の電話で表示するためにフォーマットできます。

この例では、データベースが XSLT 処理を実行できるように、スタイルシートが Oracle XML DB Repository にロードされています。

2. この時点では、このブラウザ・ウィンドウを閉じないでください。

8.3 XSLT を使用した PurchaseOrder の変換

この手順では、PurchaseOrder 文書を XML から HTML に変換するために使用される Oracle XML DB のスタイルシート・プロセッサが示されます。

> **参照：** 第 6 章「XMLType データの変換および検証」を参照してください。

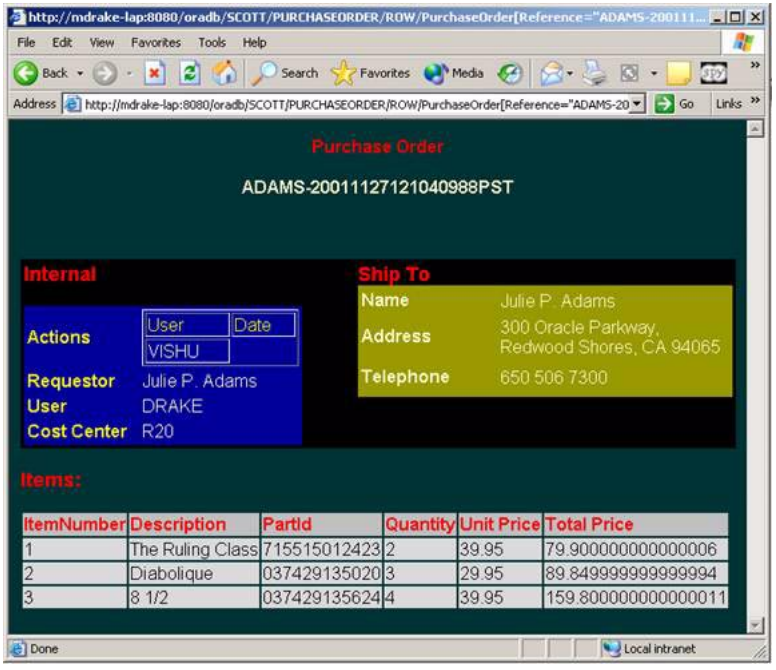
要点は次のとおりです。

- Oracle XML DB には、データベースにリンクされた XSLT スタイルシート処理エンジンが含まれています。
- XSLT プロセッサは、遅延ロードされた DOM などの、Oracle XML DB のパフォーマンスの最適化を使用できます。これらの最適化によって、XSLT 変換の実行に関連する解析およびメモリーのオーバーヘッドの量が大幅に削減されます。
- 従来の XSL 処理では、最初に DOM（XML 文書のメモリー内表現）が作成されます。次に、この DOM が XSLT プロセッサに渡されます。XSLT プロセッサは、DOM API を使用して、文書に含まれている情報を取得し、それを HTML としてフォーマットします。XSLT 変換の標準の実行方法には、関連する多くの問題があります。
  - DOM の作成は、プロセッサ集中型とメモリー集中型の両方になる場合があります。XML 文書のメモリー内表現が、元の文書の数倍のサイズになる可能性があります。
  - XSLT 処理を開始する前に、文書全体を解析し、DOM 構造に変換する必要があります。これは、生成される出力に文書の一部のみを含める場合、非常に非効率的であることがあります。
  - XML Schema に基づくコンテンツの場合、XML Schema を解析し、メモリーにロードする必要もあります。
- Oracle XML DB では、これらの問題が解決されています。XML Schema に基づくコンテンツを変換する場合、XSLT プロセッサは Oracle XML DB の遅延ロードされた仮想 DOM のインスタンスを使用して表されます。この XML DB DOM によって、XSLT プロセッサは従来の DOM API を使用して変換を実行できます。ただし、この XML DB DOM には、従来の DOM に比べて多くのメリットがあります。
  - DOM がディスクから直接ロードされます。DOM をロードするために、解析を行う必要がありません。文書の処理される部分のみが DOM にロードされるため、DOM のメモリー使用量が削減されます。
  - 処理の続行中に、文書の他の部分が必要に応じてロードされます。ページング・メカニズムを使用して、DOM 全体のサイズが管理されます。

- どの XML Schema も、Oracle XML DB のスキーマ・キャッシュからアクセスされます。

1. 「お気に入り」メニューの「XML DB Basic Demo」項目を開き、「**8.3 PurchaseOrder XSL Transform**」項目を選択します。これによって、XSLT スタイルシートを使用して PurchaseOrder 文書を変換した結果が表示されます。

図 26-49 XSL スタイルシートを使用した XML 文書 PurchaseOrder の変換



要点は次のとおりです。

- スタイルシート処理は、データベースに取り込まれている XSLT プロセッサによって実行されます。
- スタイルシート・プロセッサは、**transform=** パラメータを URL の一部として指定することによって、DBUri サーブレットから起動できます。**transform** パラメータの値は、変換の実行時に使用するスタイルシートを識別する URL です。

- XSLT プロセッサは、XMLType の transform() メソッドまたは SQL 関数 xmltransform() を使用して、SQL から起動することもできます。変換は、Oracle XML DB の XSLT プロセッサによって、データベース・レベルで実行されます。
- スタイルシート処理では、遅延ロードされた DOM などの Oracle XML DB の最適化を使用できます。これによって、XSLT 処理が大幅に効率化されます。
- 前述の例には、ターゲット・ドキュメントからのすべての情報が含まれています。ただし、生成されたイメージは XML 文書の XSLT 変換に基づくサマリー・ドキュメントです。
  * 従来のシステムでは、必要な各ドキュメントを解析し、DOM に変換してから、処理する必要があります。
  * Oracle XML DB では、サマリーに lineItem 要素の情報が含まれていない場合、これらは XSLT 処理の一部としてロードされません。

2. この時点では、このブラウザ・ウィンドウを閉じないでください。

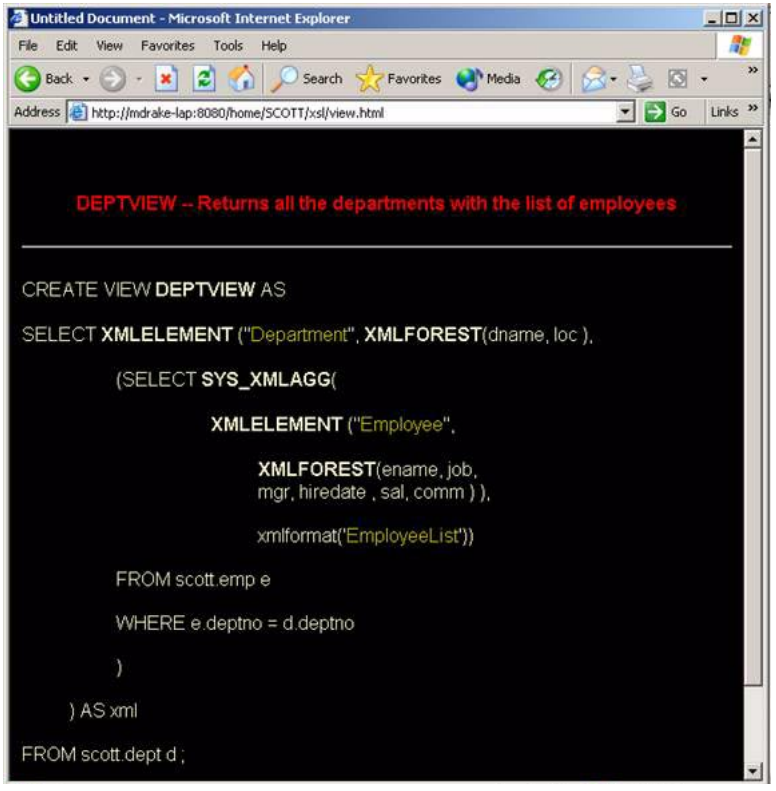
8.4 SQL を使用した XMLType ビューの作成

この手順では、SQL/XML（SQLX）演算子および関数を使用して、XMLType ビューを作成したり、SQL 問合せから XML を生成する方法が示されます。

参照： 第 11 章「XMLType ビュー」を参照してください。

1. 「**お気に入り**」メニューの「XML DB Basic Demo」項目を開き、「**8.4 DEPTVIEW Definition**」項目を選択します。
2. これによって、XMLType ビュー DEPTVIEW の定義を示す HTML ページが表示されます。

図 26-50 XMLType ビュー DEPTVIEW を示す HTML ページ



要点は次のとおりです。

- SQL/XML は ANSI/ISO 標準です。これは、2 つの主なセクションに分類されます。
  - 1 つ目のセクションでは、XML 文書を SQL 操作の一部として挿入、問合せおよび更新するための関数と演算子、および SQL 操作によって戻される結果セットに XML 文書またはその一部を含めるための関数と演算子が提供されます。
  - 2 つ目のセクションでは、リレーショナル表に対する問合せから XML 文書を生成するための関数が提供されます。
- SQL/XML 標準は、オラクル社、IBM 社、Microsoft 社およびこれに関心を持つ他のベンダーによって開発されています。
- この例は、SQL/XML 関数を使用して `XMLType` ビューを生成する方法を示しています。これによって、SCOTT サンプル・スキーマに含まれる EMP 表および DEPT 表のコンテンツの永続 XML ビューが提供されます。

- このビューは、Department ノードのコレクションで構成されます。各 Department には、DNAME 要素と LOC 要素、および部門の各従業員の詳細情報を含む Employee ノードのコレクションが含まれます。
- SQL/XML 演算子は、SQL 結果セットから任意の形式の XML を簡単に生成できるように設計されています。
- XMLForest() は、ツリーのコレクションを含んでいるため、この名前が付いています。
- 他のベンダーとは異なり、オラクル社では、問合せを解決するために使用する計画の決定時に SQL/XML 演算子を考慮します。このため、XML の生成時に選択された問合せ計画が、同等の表形式の結果セットの生成時に選択された問合せ計画と異なる場合があります。
- 他のベンダーは、SQL/XML 演算子に基づく DOM を作成した後に、従来のリレーショナル問合せを実行し、その結果を強制的に DOM に変換してから、その DOM を出力することによって、SQL/XML を実装します。これは、多くの理由から非効率的です。
  - データベースは、問合せ計画の決定時に必要な出力の形式を考慮できません。
  - 要求された結果セットを生成する前に、表形式の結果セットを強制的に DOM 構造に変換する必要があります。これによって、多くのメモリーおよび処理のオーバーヘッドが発生します。

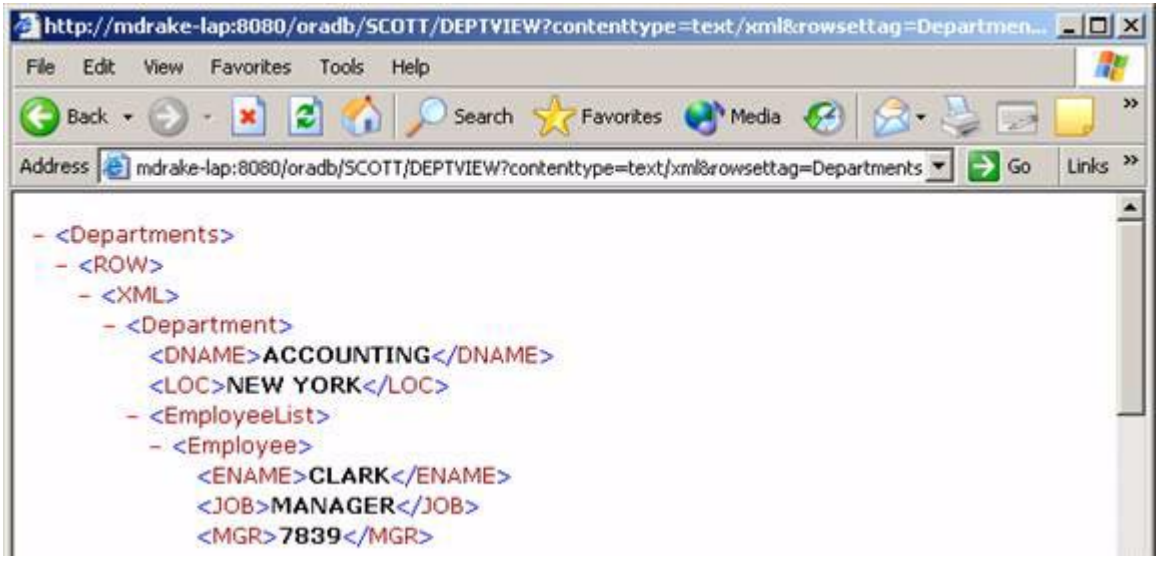
3. この時点では、このブラウザ・ウィンドウを閉じないでください。

8.5 DBUri サーブレットを使用した RAW XML DEPTVIEW の表示

この手順では、DBUri サーブレットを使用して DEPTVIEW のコンテンツを表示する方法が示されます。

1. 「**お気に入り**」メニューの「XML DB Basic Demo」項目を開き、「**8.5 Display DEPTVIEW**」項目を選択します。これによって、XMLType ビュー DEPTVIEW のコンテンツが表示されます。

図 26-51 DEPTVIEW のコンテンツ



要点は次のとおりです。

- この XMLType ビューでは、DEPT および EMP が単一の XML 文書として公開されます。
- 生成された文書には、DEPT 表から選択された行ごとに 1 つの <ROW> 要素が存在します。
- XML 文書の形式は、ビュー定義で使用される SQL/XML 演算子によって定義されます。

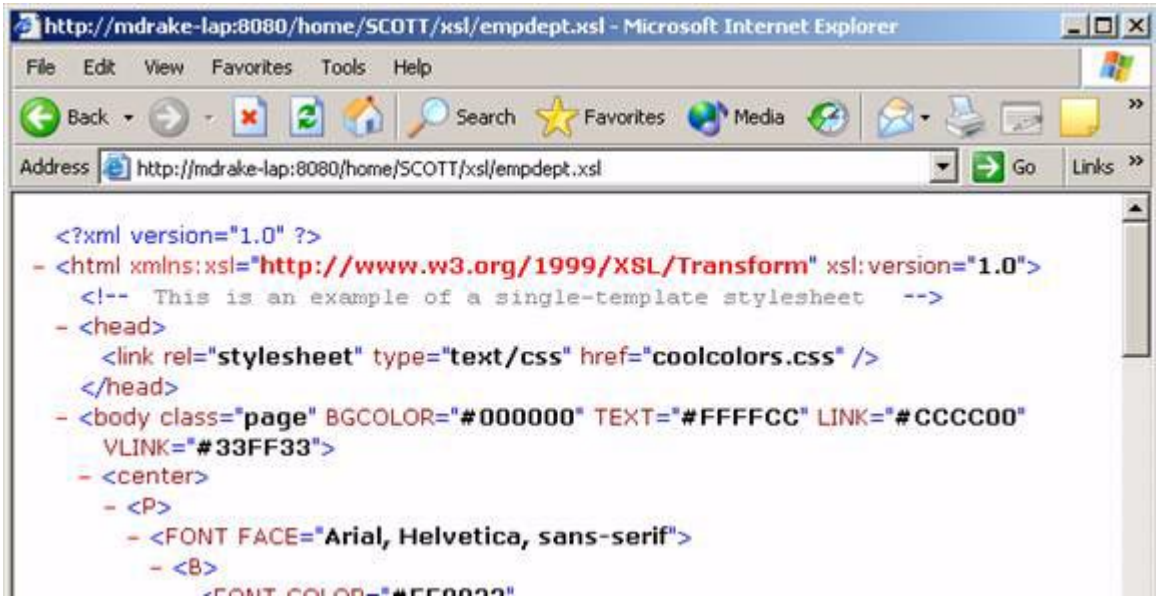
> **参照：** 第 12 章「URL を介したデータの作成およびアクセス」を参照してください。

8.6 スタイルシートを使用した XML から HTML への DEPTVIEW の変換

この手順では、スタイルシートを使用して PurchaseOrder 文書を XML から HTML に変換する方法が示されます。

1. 「お気に入り」メニューの「XML DB Basic Demo」項目を開き、「**8.6 DEPTVIEW style sheet**」項目を選択します。これによって、DEPTVIEW ビューで使用可能な XSLT スタイルシートが表示されます。

図 26-52 DEPTVIEW で使用するための XSL スタイルシート



要点は次のとおりです。

- このスタイルシートは、W3C 準拠の標準 XSLT スタイルシートです。このスタイルシートについては、Oracle 固有のものはありません。

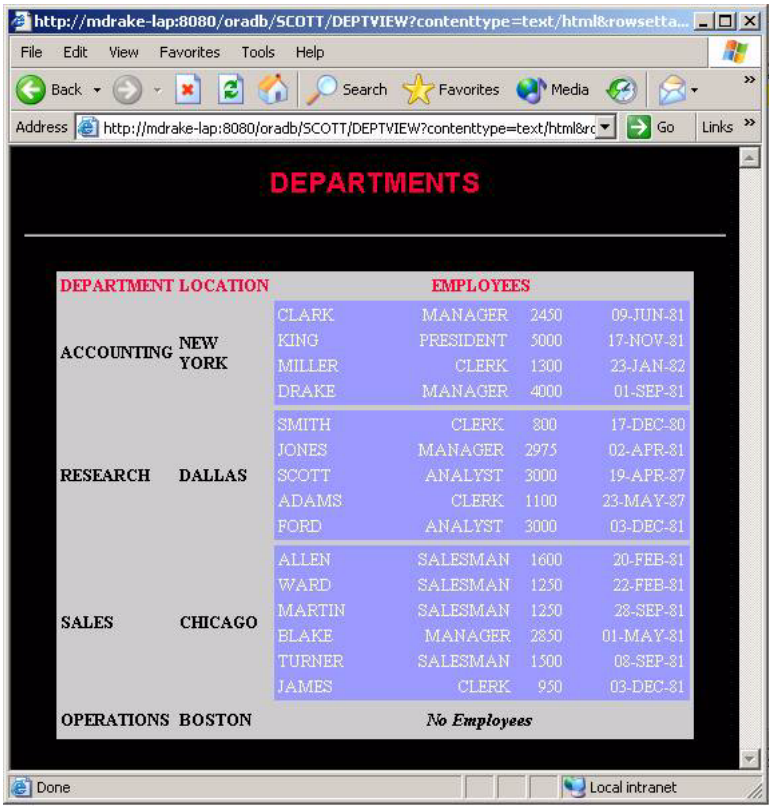
この時点では、このブラウザ・ウィンドウを閉じないでください。

8.7 XSLT 変換後の変換済 DEPTVIEW の表示

この手順では、スタイルシートを使用して PurchaseOrder 文書を XML から HTML に変換する方法が示されます。

1. 「お気に入り」メニューの「XML DB Basic Demo」項目を開き、「**8.7 DEPTVIEW with XSL Transformation**」項目を選択します。これによって、XSLT スタイルシートを使用して PurchaseOrder 文書を変換した結果が表示されます。

図 26-53 PurchaseOrder 文書を XML から HTML に変換するためのスタイルシート



要点は次のとおりです。

- Oracle XML DB を使用すると、DEPT や EMP などのリレーショナル表のコンテンツを HTML ドキュメントとして簡単に表示できます。
- SQL/XML 演算子を使用して、リレーショナル表のコンテンツを XML 文書としてフォーマットする XMLType ビューを作成します。次に、XSLT スタイルシートを適用して、その XML 文書を HTML に変換します。
- この機能を実現するために、プロシージャのコーディング、サーブレットまたは他のアプリケーション・コンポーネントの作成またはインストールは必要ありません。

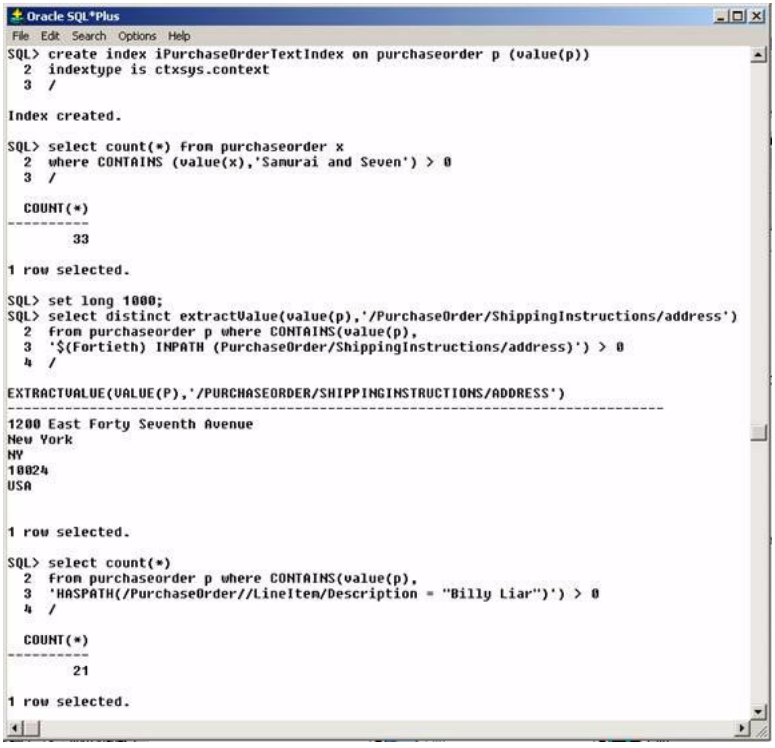
2. ブラウザ・ウィンドウを閉じます。

9.0 XML DB Demo: Oracle Text の例

この手順では、構造化記憶域方法を使用して Oracle XML DB に格納された XML コンテンツに Oracle Text 機能を適用する方法が示されます。

1. 「**9.0 Oracle Text Examples**」アイコンをクリックして、SQL スクリプトを実行します。

図 26-54 Oracle Text の contains() 演算子を使用した PurchaseOrders 表の検索



要点は次のとおりです。

- Oracle XML DB では、構造化記憶域方法を使用してデータベースに格納された XML コンテンツに Oracle Text 索引を作成できます。

- 次の 2 種類の Oracle Text 索引を作成できます。
  - `ctxsys.ctxxpath`: この索引は、`existsNode()` 関数の処理を高速化するために使用できます。
  - `ctxsys.conte`: この索引を使用すると、XML コンテンツの全文検索を実行できます。
- デフォルトでは、`existsNode()` は B ツリー索引またはファンクション索引を検索し、問合せを解決するために使用されます。これらの索引が存在しない場合、表内の各ドキュメントに対して機能上の評価が実行され、指定された XPath 式と一致するノードが各ドキュメントに含まれているかどうかが確認されます。
- `ctxsys.ctxxpath` 索引が作成されている場合、`existsNode()` は、B ツリー索引またはファンクション索引を使用して解決できない XPath 式を解決するときに、この索引を 1 次フィルタとして使用します。これは、表内のすべてのドキュメントに対して機能上の評価を実行するより高速です。
- この例では、`PURCHASEORDER` 表に全文（`ctxsys.context`）索引を作成し、Oracle Text の `contains()` 関数を使用して XML 文書 `PurchaseOrder` のテキストベース検索を実行しています。

2. SQL> プロンプトに「QUIT」と入力して、SQL*Plus ウィンドウを閉じます。

参照： 第 7 章「Oracle Text を使用した XML データの検索」を参照してください。

A

Oracle XML DB のインストールおよび構成

この付録では、Oracle XML DB アプリケーションを管理および構成する方法を説明します。この付録の内容は次のとおりです。

Oracle XML DB のインストール

Oracle XML DB は、次の条件に従ってインストールする必要があります。

- 「Oracle XML DB の新規のインストールまたは再インストール」（A-2 ページ）
- 「既存の Oracle XML DB のインストールのアップグレード」（A-5 ページ）

Oracle XML DB の新規のインストールまたは再インストール

Database Configuration Assistant（DBCA）の有無にかかわらず、Oracle XML DB の新規のインストールを実行できます。

次の手順に従って、Oracle XML DB を再インストールします。

1. ディスパッチャを削除します。
2. ユーザー xdb を削除します。
3. 表領域 xdb を削除します。
4. 表領域 xdb を再作成します。
5. `catnoqm.sql` を実行します。
6. `catqm.sql` を実行します。
7. `catxdbj.sql` を実行します。

DBCA を使用した Oracle XML DB の新規のインストール

Oracle XML DB はシード・データベースの一部であり、デフォルトで、データベースのインストールの一部として DBCA によってインストールされます。Oracle XML DB のインストールには、他の手順は必要ありません。ただし、カスタマイズされたデータベースのインストールを選択する場合、Oracle XML DB 表領域、および FTP、HTTP、WebDAV のポート番号を構成できます。

デフォルトでは、DBCA によって次のタスクが実行されます。

- Oracle XML DB Repository に対する Oracle XML DB 表領域を作成します。
- すべてのプロトコル・アクセスを有効にします。
- FTP をポート 2100 に構成します。
- HTTP、WebDAV をポート 8080 に構成します。

Oracle XML DB 表領域には、Oracle XML DB Repository に格納されるデータが保持されます。これには、次のものを使用してリポジトリに格納されるデータが含まれます。

- SQL（`resource_view`、`path_view` などを使用）

- FTP、HTTP、WebDAV などのプロトコル

この表領域外の表にデータを格納し、このデータに対する REF をこの表領域内の表に格納することによって、リポジトリを介してこのデータにアクセスできます。

注意： Oracle XML DB 表領域は削除しないでください。削除すると、すべてのリポジトリ・データにアクセスできなくなります。

動的プロトコル登録によるローカル・リスナーへの FTP および HTTP サービスの登録

Oracle XML DB のインストールには、FTP および HTTP サービスをローカル・リスナーに登録する動的プロトコル登録が含まれます。「lsnrctl」を使用して、開始、停止および問合せを実行できます。次に例を示します。

- start: `lsnrctl start`
- stop: `lsnrctl stop`
- query: `lsnrctl status`

FTP または HTTP のポート番号の変更

FTP または HTTP のポート番号を変更するには、Oracle XML DB Repository の `/xdbconfig.xml` ファイル内のタグ `<ftp-port>` および `<http-port>` を更新します。

参照： `/xdbconfig.xml` の更新方法の詳細は、第 19 章「FTP、HTTP および WebDAV プロトコルの 使用」を参照してください。

ポート番号を更新すると、動的プロトコル登録によって、更新前のポート番号の FTP または HTTP サービスが自動的に停止され、ローカル・リスナーが起動されている場合は、新しいポート番号で FTP または HTTP サービスが開始されます。ローカル・リスナーが起動されていない場合は、ポート番号を更新した後、リスナーを再起動します。

インストール後

前述の項で説明したとおり、Oracle XML DB では、動的プロトコル登録を使用して、FTP および HTTP リスナー・サービスがローカル・リスナーに設定されます。そのため、Oracle XML DB プロトコルにアクセスする場合、リスナーが起動されていることを確認してください。

HTTP を介した Oracle XML DB Repository データに対する非認証アクセスを許可するには、`ANONYMOUS` ユーザー・アカウントのロックを解除する必要があります。

注意： リスナーが標準ポート（1521）以外で起動中の場合、プロトコルを正しいリスナーに登録するには、初期化パラメータ・ファイルに TNSNAME エントリを参照する local_listener エントリを含める必要があります。これによって、正しいリスナーがポイントされます。初期化パラメータの編集後、CREATE SPFILE を使用して SPFILE エントリを再生成する必要があります。

DBCA を使用しない手動での新規の Oracle XML DB のインストール

データベースのインストール後、Oracle XML DB Repository 用の新しい表領域を作成した後に、rdbms/admin で SYS ユーザーで接続し、次の SQL スクリプトを実行して、Oracle XML DB をインストールする必要があります。構文は次のとおりです。

```
catqm.sql <xdb_pass> <XDB_TS_NAME> <TEMP_TS_NAME> #Create the tables and views
needed to run XML DB
```

次に例を示します。

```
catqm.sql change_on_install XDB TEMP
```

SYS に再度接続し、次のスクリプトを実行します。

```
catxdbj.sql          #Load xdb java library
```

注意： データベースが Oracle9*i* データベース リリース 2（9.2）以上で起動されていることを確認してください。

インストール後

手動でのインストール後、次のタスクを実行します。

1. init.ora ファイルに次のディスパッチャ・エントリを追加します。

   ```
   dispatchers="(PROTOCOL=TCP) (SERVICE=<sid>XDB)"
   ```

2. データベースおよびリスナーを再起動して、Oracle XML DB へのプロトコル・アクセスを有効にします。
3. HTTP を介した Oracle XML DB Repository データに対する非認証アクセスを許可するには、ANONYMOUS ユーザー・アカウントのロックを解除する必要があります。

Oracle XML DB の再インストール

Oracle XML DB を再インストールするには、次の SQL コマンドを実行して SYS に接続し、Oracle XML DB ユーザーおよび表領域を削除します。

> **注意：** XML DB ユーザーを削除すると、Oracle XML DB Repository に格納されたすべてのユーザー・データも失われます。

```
drop user xdb cascade;
alter tablespace <XDB_TS_NAME> offline;
drop tablespace <XDB_TS_NAME> including contents;
```

A-4 ページの「DBCA を使用しない手動での新規の Oracle XML DB のインストール」で説明するとおり、Oracle XML DB を手動でインストールします。

既存の Oracle XML DB のインストールのアップグレード

通常どおり、スクリプト `catproc.sql` を実行します。

アップグレード後の手順として、Oracle XML DB 機能が必要である場合、A-2 ページの「DBCA を使用しない手動での新規の Oracle XML DB のインストール」で説明するとおり、Oracle XML DB を手動でインストールする必要があります。

リリース 2（9.2.0.1）からリリース 2（9.2.0.2）への XML DB のアップグレード

Oracle9*i* データベース リリース 2（9.2）の Oracle9*i* データベース リリース 2（9.2.0.2）パッチ・セットは、Oracle XML DB リリース 2（9.2.0.1）のユーザーに必要なアップグレードです。Oracle XML DB の XML Schema に基づく `XMLType` の表および列を、リリース 2（9.2.0.1）からリリース 2（9.2.0.2）に移行する必要があります。この必須の移行は、データベース・アップグレード・プロセスの一部として自動的に行われます。この移行は透過的ですが、いくつかの制限事項があります。

- Database Upgrade Assistant を使用して、リリース 2（9.2.0.2）に直接アップグレードすることはできません。Database Upgrade Assistant は、メジャー・リリースまたはマイナー・リリースへのアップグレードのみに使用でき、パッチ・セット・リリースへのアップグレードには使用できません。そのため、アップグレード・プロセスは手動で実行する必要があります。
- 手動によるアップグレード手順の詳細は、『Oracle9*i* データベース移行ガイド』の第 3 章「Oracle9*i* の新しいリリースへのデータベースのアップグレード」を参照してください。リリース 2（9.2.0.2）へのアップグレードに必要な手順は、手順 14 を除き、これと同じです。

- 手順 1 ～ 13 は同じです。システム要件を確認し、データベースを MIGRATE モードで起動します。
- 手順 14 では、catpatch.sql を実行します。このスクリプトによって、特定のディクショナリ表が作成および変更されます。また、新しいリリース 2（9.2.0.2）に含まれている catalog.sql および catproc.sql スクリプトも実行します。これによって、システム・カタログ・ビュー、および PL/SQL を使用するために必要なすべてのパッケージが作成されます。
- 手順 15 ～ 21 は同じです。

この時点で、Oracle XML DB は自動的にアップグレードされ、XML Schema に基づく XML データは新しいリリースである Oracle9*i* データベース リリース 2（9.2.0.2）で使用可能な新しい形式に移行されています。次の文を実行し、すべての構成要素が妥当であり、新しいリリースにアップグレードされていることを確認します。

```
SQL> SELECT comp_id, version, status FROM dba_registry;
```

注意： すべての XML Schema および文書を最初からリストアできる場合、アップグレード前に XML DB を削除してから XML DB を再インストールし、アップグレードが正常に実行された後ですべてのデータを再ロードすることをお薦めします。この方法には、ユーザーの XML データの移行が失敗する可能性がなくなり、データをリリース 2（9.2.0.1）形式から移行するのではなく、移行後にデータベースに新規にロードした場合、リリース 2（9.2.0.2）に組み込まれているすべての最適化が最大限に使用されるという 2 つのメリットがあります。

リリース 2（9.2.0.1）からリリース 2（9.2.0.2）へのデータの移行

データを移行する場合は、次の重要な指示に従ってください。

アップグレード前

オブジェクト・リレーショナルとして格納された、XML Schema に基づくすべてのデータのバックアップを取る必要があります。これを行うと、破損行が削除され、XML が最初から再ロードされるため、移行プロセス中のデータ破損が最小化されます。

また、アップグレード前に、オブジェクト・リレーショナルとして格納された、XML Schema に基づくすべての XMLType の行および列がその XML Schema に対して妥当であることを確認する必要があります。リリース 2（9.2.0.1）では、Oracle XML DB は、表に挿入される XML 文書がその XML Schema に対して妥当であることを厳密に確認しませんでしたが、リリース 2（9.2.0.2）では、データ格納の特定の部分が、格納された XML 文書の XML Schema に対する妥当性に依存します。そのため、リリース 2（9.2.0.1）を使用して格納された、XML Schema に準拠しない XML 文書はリリース 2（9.2.0.2）に移行できません。

データをリリース 2（9.2.0.2）に移行できない場合

XML Schema に基づく文書をリリース 2（9.2.0.2）に移行できない例は 2 つあります。データがこれら 2 つのカテゴリのいずれかに該当する場合、次の手順を実行することをお薦めします。

1. 移行されない XML Schema に準拠するすべての文書を XML テキストとして保存します。
2. アップグレード前に、XML Schema を登録解除します。
3. XML Schema を再登録します。
4. アップグレード完了後に、文書を再ロードします。

次の 2 つのカテゴリのいずれかに該当するデータを移行しようとすると、移行に失敗した行ごとにトレース・ファイルにエラーが記録されます。

XML Schema に anyType 要素を含む文書 移行できない 1 つ目のタイプの XML 文書は、XML Schema に `anyType` 要素を含む XML 文書です。リリース 2（9.2.0.1）における `anyType` に対する格納制限のため、オラクル社では、NULL でない 1 つ以上の `anyType` 要素を含む XML 文書を移行できないように、リリース 2（9.2.0.2）では `anyType` の格納形式を変更しています。

親要素と異なる名前空間を持つサブタイプ要素を含む文書 移行できない 2 つ目のタイプの XML 文書は、親要素と異なる名前空間を持つサブタイプ要素を XML Schema に含む XML 文書です。これは、リリース 2（9.2.0.1）の場合、Oracle XML DB がサブタイプ要素の名前空間がその親要素の名前空間と同じであることを要求および想定していたため、サブタイプ要素の実際の名前空間が格納フェーズで失われたためです。そのため、これらの XML Schema およびそれに準拠する XML 文書はリリース 2（9.2.0.2）に移行できません。

リリース 2（9.2.0.2）への Oracle XML DB アップグレード・プロセス

`XMLType` データの移行は、アップグレード・スクリプト `catpatch.sql` 内で透過的に行われます。

> **参照：** このスクリプトの実行については、A-5 ページの「リリース 2（9.2.0.1）からリリース 2（9.2.0.2）への XML DB のアップグレード」を参照してください。

Oracle XML DB の移行中に発生したエラーは、移行が失敗した行の表名および ROWID とともにトレース・ファイルに記載されます。行の移行中にエラーが発生すると、移行スクリプトによって単にそのエラーがトレース・ファイルに記載され、次の行の移行が続行されます。移行スクリプトは、エラーが発生しても中断されません。

すべての行が正常に移行された場合、一度データベースを再起動すると、Oracle XML DB を使用できます。

Oracle XML DB: リリース 2（9.2.0.2）への移行時のエラー処理

1 つ以上の XMLType 行の移行中にエラーが発生したことをトレース・ファイルが示す場合、移行が完了していなくても、残りの Oracle XML DB は使用可能なままである必要があります。移行されていない行に後でアクセスしようとすると、ORA-01038 エラーが発生します。XMLType 表から移行されていない行を削除しても問題ありません。

アップグレード前にバックアップを取った場合は、行が移行され、エラーが発生していなくても、アップグレード完了後にその行を使用できないときは、その行を削除し、RAW XML データを使用してその行をリストアする必要があります。

アップグレード後も、すべての XMLType 表は、移行されたかどうかにかかわらず、移行エンジンを介して再実行できます。これは、特定の XMLType 行が catpatch.sql の一部として正常に移行されなくても、その後この移行を正常に実行するための処置を講じた場合に有効です。

表の移行をトリガーするファンクションの概要

表 A-1 に、リリース 2（9.2.0.2）へのアップグレード後に使用可能な XDB.DBMS_XDB パッケージの移行プロシージャを示します。

表 A-1 XDB.DBMS_XDB パッケージ リリース 2（9.2.0.2）の移行プロシージャ

プロシージャ	説明
MigrateColumnFrom9201	1 つの XMLType 列をリリース 2（9.2.0.1）形式からリリース 2（9.2.0.2）形式に移行します。この列は、オブジェクト表内に存在する必要があり、XMLType 型で、XML Schema に基づき、オブジェクト・リレーショナル形式で格納されている必要があります。列の移行前に、その表に排他ロックが設定されます。 パラメータ : owner (IN) - XMLType 表を所有するデータベース・ユーザー table_name (IN) - 表の名前 column_name (IN) - 表内の XMLType 列の名前（「FOO」、「BAR」など） 構文 : MigrateColumnFrom9201(owner IN VARCHAR2, table_name IN VARCHAR2, column_name IN VARCHAR2)
MigrateTableFrom9201	1 つの XMLType 表をリリース 2（9.2.0.1）形式からリリース 2（9.2.0.2）形式に移行します。この表はその行型として XMLType を持つ必要があり、XMLType 行は XML Schema に基づき、オブジェクト・リレーショナル形式で格納されている必要があります。表に対する操作の前に、その表に排他ロックが設定されます。 パラメータ : owner (IN) - XMLType 表を所有するデータベース・ユーザー table_name (IN) - XMLType 表の名前 構文 : MigrateTableFrom9201(owner IN VARCHAR2, table_name IN VARCHAR2);

表 A-1 XDB.DBMS_XDB パッケージ リリース 2（9.2.0.2）の移行プロシージャ（続き）

プロシージャ	説明
MigrateAllXmlFrom9201	すべてのオブジェクト・リレーショナルな XMLType 表および XMLType 列をリリース 2（9.2.0.1）形式からリリース 2（9.2.0.2）形式に移行します。各表に対する操作の前に、その表に排他ロックが設定されます。 パラメータ： なし 構文： MigrateAllXmlFrom9201;

Oracle XML DB の構成

次の項では、Oracle XML DB を構成する方法について説明します。Oracle Enterprise Manager を使用して Oracle XML DB を構成することもできます。

> **参照：** 第 21 章「Oracle Enterprise Manager を使用した Oracle XML DB の管理」を参照してください。

Oracle XML DB は、Oracle XML DB Repository に格納された構成リソース /xdbconfig.xml を介して管理できます。

Oracle XML DB の構成ファイルは、実行時に変更できます。構成ファイルを更新すると、新しいバージョンのファイルが生成されます。セッションが開始されると、現行のバージョンの構成がそのセッションにバインドされます。セッションでは、明示的なコールを行って最新の構成にリフレッシュしないかぎり、最後までこの構成が使用されます。

Oracle XML DB の構成ファイル xdbconfig.xml

Oracle XML DB の構成は、Oracle XML DB の構成 XML Schema http://xmlns.oracle.com/xdb/xdbconfig.xsd に準拠する XML リソース /xdbconfig.xml として格納されます。

Oracle XML DB の構成を変更するには、xdbconfig.xml に対して適切な XML 要素を挿入、削除または編集することによって、/xdbconfig.xml ファイルを更新します。

Oracle XML DB の構成 XML Schema の構造は、次のとおりです。

最上位タグ <xdbconfig>

最上位タグ <xdbconfig> は、2 つのセクションに分割されます。

- **<sysconfig>:** システム固有の組込みパラメータが保持されます。
- **<userconfig>:** ユーザーが新しいカスタム・パラメータを格納します。

次に構文を示します。

```
<xdbconfig>
    <sysconfig> ... </sysconfig>
    <userconfig>  ...  </userconfig>
</xdbconfig>
```

<sysconfig>

<sysconfig> セクションは、次のとおりさらに分割されます。

```
<sysconfig>
    General parameters
      <protocolconfig> ... </protocolconfig>
</sysconfig>
```

このセクションには、ACL の最大存続期間、Oracle XML DB で大 / 小文字が区別されるかどうかなど、すべての Oracle XML DB に適用されるいくつかの一般的なパラメータが格納されます。

プロトコル固有のパラメータは、<protocolconfig> タグ内でグループ化されます。

<userconfig>

<userconfig> セクションには、追加する必要がある場合があるすべてのパラメータが含まれます。

<protocolconfig>

<protocolconfig> セクションの構造は、次のとおりです。

```
<protocolconfig>
  <common> ... </common>
  <httpconfig> ... </httpconfig>
  <ftpconfig> ... </ftpconfig>
</protocolconfig>
```

<common> には、MIME タイプについての情報など、すべてのプロトコルに適用されるパラメータが Oracle9*i* データベースによって格納されます。<httpconfig> および <ftpconfig> セクションには、それぞれ HTTP および FTP 固有のパラメータが格納されます。

<httpconfig>

<httpconfig> 内には、Web ベースのアプリケーションに対応する <webappconfig> サブセクションも存在します。このサブセクションには、アイコンの名前、アプリケーションの表示名、Oracle XML DB Servlet のリストなど、Web アプリケーション固有のパラメータが格納されます。

参照：

- プロトコル構成パラメータのリストについては、第 19 章「FTP、HTTP および WebDAV プロトコルの 使用」の表 19-1、表 19-2 および表 19-3 を参照してください。
- 20-12 ページの「Oracle XML DB のサンプル・サーブレットの構成」も参照してください。

Oracle XML DB の構成の例

次に、Oracle XML DB の構成ファイルの例を示します。

例 A-1 Oracle XML DB の構成ファイル

```
<xdbconfig xmlns="http://xmlns.oracle.com/xdb/xdbconfig.xsd"
                 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                 xsi:schemaLocation="http://xmlns.oracle.com/xdb/xdbconfig.xsd
                                     http://xmlns.oracle.com/xdb/xdbconfig.xsd">
   <sysconfig>
      <acl-max-age>900</acl-max-age>
      <invalid-pathname-chars>,</invalid-pathname-chars>
      <call-timeout>300</call-timeout>
      <max-session-use>100</max-session-use>
      <default-lock-timeout>3600</default-lock-timeout>
      <xdbcore-logfile-path>/sys/log/xdblog.xml</xdbcore-logfile-path>
      <xdbcore-log-level>1</xdbcore-log-level>

      <protocolconfig>
          <common>
             <extension-mappings>
                <mime-mappings>
                  <mime-mapping>
                    <extension>au</extension>
```

```
            <mime-type>audio/basic</mime-type>
          </mime-mapping>
          <mime-mapping>
            <extension>avi</extension>
            <mime-type>video/x-msvideo</mime-type>
          </mime-mapping>
          <mime-mapping>
            <extension>bin</extension>
            <mime-type>application/octet-stream</mime-type>
          </mime-mapping>

        <lang-mappings>
          <lang-mapping>
            <extension>en</extension>
            <lang>english</lang>
          </lang-mapping>
        </lang-mappings>

        <charset-mappings>
        </charset-mappings>

        <encoding-mappings>
          <encoding-mapping>
            <extension>gzip</extension>
            <encoding>zip file</encoding>
          </encoding-mapping>
          <encoding-mapping>
            <extension>tar</extension>
            <encoding>tar file</encoding>
          </encoding-mapping>
        </encoding-mappings>
    </extension-mappings>

    <session-pool-size>50</session-pool-size>
    <session-timeout>6000</session-timeout>
</common>

<ftpconfig>
  <ftp-port>2100</ftp-port>
  <ftp-listener>local_listener</ftp-listener>
  <ftp-protocol>tcp</ftp-protocol>
  <logfile-path>/sys/log/ftplog.xml</logfile-path>
  <log-level>0</log-level>
  <session-timeout>6000</session-timeout>
</ftpconfig>

<httpconfig>
```

```
                <http-port>8080</http-port>
                <http-listener>local_listener</http-listener>
                <http-protocol>tcp</http-protocol>
                <session-timeout>6000</session-timeout>
                <server-name>XDB HTTP Server</server-name>
                <max-header-size>16384</max-header-size>
                <max-request-body>2000000000</max-request-body>
                <logfile-path>/sys/log/httplog.xml</logfile-path>
                <log-level>0</log-level>
                <servlet-realm>Basic realm="XDB"</servlet-realm>
                <webappconfig>
                  <welcome-file-list>
                    <welcome-file>index.html</welcome-file>
                    <welcome-file>index.htm</welcome-file>
                  </welcome-file-list>
                  <error-pages>
                  </error-pages>
                  <servletconfig>
                    <servlet-mappings>
                      <servlet-mapping>
                        <servlet-pattern>/oradb/*</servlet-pattern>
                        <servlet-name>DBURIServlet</servlet-name>
                      </servlet-mapping>
                    </servlet-mappings>

                    <servlet-list>
                      <servlet>
                        <servlet-name>DBURIServlet</servlet-name>
                        <display-name>DBURI</display-name>
                        <servlet-language>C</servlet-language>
                        <description>Servlet for accessing DBURIs</description>
                        <security-role-ref>
                          <role-name>authenticatedUser</role-name>
                          <role-link>authenticatedUser</role-link>
                        </security-role-ref>
                      </servlet>
                    </servlet-list>
                  </servletconfig>
                </webappconfig>
              </httpconfig>
            </protocolconfig>
          </sysconfig>

          <userconfig><numusers>40</numusers></userconfig>

        </xdbconfig>
```

Oracle XML DB Configuration API

Oracle XML DB Configuration API には、階層内の他の XML Schema に基づくリソースと同様にアクセスできます。この Configuration API は、FTP、HTTP、WebDAV、Oracle Enterprise Manager、Java DOM API、PL/SQL DOM API、Resource API for Java、Resource API for PL/SQL などを使用して、アクセスおよび操作できます。

構成用に、DBMS_XDB パッケージの一部として PL/SQL API が提供されています。この API によって、次のファンクションが公開されます。

構成の取得 : CFG_Get()

CFG_Get() ファンクションは、構成ファイルのコピーを XMLType として戻します。

```
DBMS_XDB.CFG_GET() RETURN SYS.XMLTYPE
```

CFG_Get() は、自動コミットされます。

構成の更新 : CFG_Update()

CFG_Update() ファンクションは、構成を新しい構成に更新します。

```
DBMS_XDB.CFG_UPDATE(newconfig SYS.XMLTYPE)
```

例 A-2 CFG_Update() および CFG_Get() を使用した構成ファイルの更新

構成ファイルでいくつかのパラメータを更新する場合、次の文を使用できます。

```
BEGIN
DBMS_XDB.CFG_UPDATE(UPDATEXML(UPDATEXML
    (DBMS_XDB.CFG_GET(),
        /xdbconfig/descendant::ftp-port/text()',   '2121'),
        '/xdbconfig/descendant::http-port/text()',
         19090'))
END;
/
```

多数のパラメータを更新する場合、前述の例では複雑になります。かわりに FTP、HTTP または Oracle Enterprise Manager を使用します。

構成のリフレッシュ : CFG_Refresh()

CFG_Refresh() ファンクションは、その時点におけるディスク内での最新のバージョンに対応するように、構成スナップショットを更新します。

```
DBMS_XDB.CFG_REFRESH()
```

通常、CFG_Refresh() は、次のいずれかの場合にコールされます。

- 構成の変更後、セッションで最新バージョンの構成情報を取得する必要がある場合。
- セッションの実行が長時間に及び、同時セッションによって構成が変更され、現行のセッションで最新バージョンの構成情報を取得する必要がある場合。
- 構成が更新され、Oracle XML DB Configuration API によってその変更が認識されている場合。

> **参照 :** 『Oracle9*i* XML API リファレンス - XDK および Oracle XML DB』の DBMS_XDB についての章を参照してください。

B

XML Schema の手引き

この付録では、W3C の XML Schema 勧告の概要について説明します。この付録の内容は次のとおりです。

XML Schema の概要

この概要の一部は、W3C の XML Schema ノートから抜粋したものです。

参照 :

- http://www.w3.org/TR/xmlschema-0/ の『Primer（入門書）』
- http://www.w3.org/TR/xmlschema-1/ の『Structures（構造）』
- http://www.w3.org/TR/xmlschema-2/ の『Datatypes（データ型）』
- http://w3.org/XML/Schema
- http://www.oasis-open.org/cover/schemas.html
- http://www.xml.com/pub/a/2000/11/29/schemas/part1.html

XML Schema（この付録ではスキーマという）は、XML 文書のクラスを定義します。「インスタンス・ドキュメント」という用語は、多くの場合、特定のスキーマに準拠する XML 文書を示すために使用されます。ただし、インスタンスおよびスキーマはドキュメントとして存在する必要はありません。これらは、アプリケーション間で送信されたバイトのストリーム、データベース・レコード内のフィールド、または XML Information Set の情報項目のコレクションとして存在する場合があります。ただし、この付録では説明を簡単にするために、インスタンスおよびスキーマをドキュメントおよびファイルと同様に扱います。

発注書 po.xml

XML ファイル po.xml 内の次のインスタンス・ドキュメントについて考えてみます。このインスタンス・ドキュメントは、家庭用品の発注および請求を処理するアプリケーションによって生成された発注書を記述しています。

```
<?xml version="1.0"?>
         <purchaseOrder orderDate="1999-10-20">
               <shipTo country="US">
                    <name>Alice Smith</name>
                    <street>123 Maple Street</street>
                    <city>Mill Valley</city>
                    <state>CA</state>
                    <zip>90952</zip>
                </shipTo>
                <billTo country="US">
                    <name>Robert Smith</name>
                    <street>8 Oak Avenue</street>
                    <city>Old Town</city>
                    <state>PA</state>
                    <zip>95819</zip>
```

```
                </billTo>
                <comment>Hurry, my lawn is going wild!</comment>
                <items>
                    <item partNum="872-AA">
                        <productName>Lawnmower</productName>
                        <quantity>1</quantity>
                        <USPrice>148.95</USPrice>
                        <comment>Confirm this is electric</comment>
                    </item>
                    <item partNum="926-AA">
                        <productName>Baby Monitor</productName>
                        <quantity>1</quantity>
                        <USPrice>39.98</USPrice>
                        <shipDate>1999-05-21</shipDate>
                    </item>
            </items>
</purchaseOrder>
```

この発注書は、主要素 purchaseOrder およびサブ要素 shipTo、billTo、comment および items で構成されます。これらのサブ要素（comment を除く）は他のサブ要素を含み、それらはさらに他のサブ要素を含みます。最後は、サブ要素ではなく、USPrice などの数値を含むサブ要素に到達します。

- **複合型要素 :** サブ要素または属性を持つ要素は、複合型を持つといいます。
- **単純型要素 :** サブ要素を含まず、数値（および文字列、日付など）を含む要素は、単純型を持つといいます。一部の要素は属性を持ち、属性は常に単純型を持ちます。

このインスタンス・ドキュメント内の複合型、および一部の単純型は、発注書スキーマで定義されています。その他の単純型は、XML Schema の組込み単純型レパートリの一部として定義されています。

インスタンス・ドキュメントと発注書スキーマの対応付け

発注書スキーマは、XML インスタンス・ドキュメント内で言及されていません。多くのインスタンスは XML Schema を参照しますが、実際は、XML Schema を参照する必要はありません。インスタンス・ドキュメントのすべてのプロセッサはインスタンス・ドキュメントからの情報なしでも発注書の XML Schema を取得できると想定しています。インスタンスと XML Schema を対応付ける明示的なメカニズムについては、この付録の後半を参照してください。

発注書スキーマ po.xsd

発注書スキーマは、po.xsd ファイルに含まれています。

発注書スキーマ po.xsd

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:annotation>
     <xsd:documentation xml:lang="en">
      Purchase order schema for Example.com.
      Copyright 2000 Example.com. All rights reserved.
     </xsd:documentation>
  </xsd:annotation>

<xsd:element name="purchaseOrder" type="PurchaseOrderType"/>

<xsd:element name="comment" type="xsd:string"/>

<xsd:complexType name="PurchaseOrderType">
 <xsd:sequence>
   <xsd:element name="shipTo" type="USAddress"/>
   <xsd:element name="billTo" type="USAddress"/>
   <xsd:element ref="comment" minOccurs="0"/>
   <xsd:element name="items"  type="Items"/>
 </xsd:sequence>
 <xsd:attribute name="orderDate" type="xsd:date"/>
</xsd:complexType>

<xsd:complexType name="USAddress">
  <xsd:sequence>
   <xsd:element name="name"   type="xsd:string"/>
   <xsd:element name="street" type="xsd:string"/>
   <xsd:element name="city"   type="xsd:string"/>
   <xsd:element name="state"  type="xsd:string"/>
   <xsd:element name="zip"    type="xsd:decimal"/>
 </xsd:sequence>
 <xsd:attribute name="country" type="xsd:NMTOKEN"fixed="US"/>
</xsd:complexType>

<xsd:complexType name="Items">
  <xsd:sequence>
   <xsd:element name="item" minOccurs="0" maxOccurs="unbounded">
    <xsd:complexType>
     <xsd:sequence>
       <xsd:element name="productName" type="xsd:string"/>
         <xsd:element name="quantity">
           <xsd:simpleType>
```

```
            <xsd:restriction base="xsd:positiveInteger">
              <xsd:maxExclusive value="100"/>
            </xsd:restriction>
          </xsd:simpleType>
        </xsd:element>
        <xsd:element name="USPrice"  type="xsd:decimal"/>
        <xsd:element ref="comment"   minOccurs="0"/>
        <xsd:element name="shipDate" type="xsd:date" minOccurs="0"/>
      </xsd:sequence>
      <xsd:attribute name="partNum" type="SKU" use="required"/>
    </xsd:complexType>
  </xsd:element>
 </xsd:sequence>
</xsd:complexType>

<!-- Stock Keeping Unit, a code for identifying products -->
<xsd:simpleType name="SKU">
   <xsd:restriction base="xsd:string">
      <xsd:pattern value="\d{3}-[A-Z]{2}"/>
   </xsd:restriction>
</xsd:simpleType>
</xsd:schema>
```

発注書スキーマは、スキーマ要素および様々なサブ要素、特に要素 `complexType` および `simpleType` で構成されます。これらのサブ要素は、XML インスタンス・ドキュメントにおける要素の出現方法およびその内容を決定します。

接頭辞 xsd:

スキーマ内の各要素には、接頭辞 `xsd:` が付いています。この接頭辞は、スキーマ要素に出現する宣言 `xmlns:xsd="http://www.w3.org/2001/XMLSchema"` によって XML Schema の名前空間と対応付けられています。任意の接頭辞を使用できますが、通常、XML Schema の名前空間を示すために接頭辞 `xsd:` が使用されます。同じ接頭辞（同じ対応付け）が `xsd:string` などの組込み単純型の名前にも出現しています。これによって、要素および単純型がスキーマ作者のボキャブラリではなく、XML Schema 言語のボキャブラリに属していることが識別されます。簡略化のため、この説明では要素および単純型の名前（`simpleType` など）を使用し、接頭辞は省略します。

XML Schema のコンポーネント

スキーマ・コンポーネントは、スキーマの抽象データ・モデルを構成するビルディング・ブロックに対する一般用語です。XML Schema は一連のスキーマ・コンポーネントです。全部で 13 種類のコンポーネントがあり、それらは 3 つのグループに分類されます。

プライマリ・コンポーネント

次に、プライマリ・コンポーネントを示します。このコンポーネントには、名前を持つ場合があるもの（型定義）と名前を持つ必要があるもの（要素宣言および属性宣言）があります。

- 単純型定義
- 複合型定義
- 属性宣言
- 要素宣言

セカンダリ・コンポーネント

次に、セカンダリ・コンポーネントを示します。このコンポーネントは、名前を持つ必要があります。

- 属性グループ定義
- ID 制約定義
- モデル・グループ定義
- 表記法宣言

ヘルパー・コンポーネント

ヘルパー・コンポーネントは他のコンポーネントの一部を提供し、それらのコンテキストに依存しています。

- 注釈
- モデル・グループ
- 小要素
- ワイルド・カード
- 属性の使用

複合型定義、要素宣言および属性宣言

XML Schema では、複合型と単純型の間に次の基本的な相違点があります。

- 複合型 : 要素を内容として持つことができ、属性を持つ場合があります。
- 単純型 : 要素を内容として持つことができず、属性を持つこともできません。

また、次のコンポーネントの間にも大きい相違点があります。

- 定義 : 新しい型（単純型と複合型の両方）を作成します。
- 宣言 : 特定の名前および型（単純型と複合型の両方）を持つ要素および属性がドキュメント・インスタンスで出現できるようにします。

この項では、複合型を定義し、その複合型内で出現する要素および属性を宣言します。

新しい複合型は、complexType 要素を使用して定義され、通常、そのような定義は一連の要素宣言、要素参照および属性宣言を含んでいます。宣言はそれ自体は型ではなく、名前と制約の対応付けです。制約は、対応付けられたスキーマによって制御されるドキュメント内でのその名前の出現方法を制御します。要素は element 要素を使用して宣言され、属性は attribute 要素を使用して宣言されます。

USAddress 型の定義

たとえば、USAddress は複合型として定義されており、USAddress の定義内に 5 つの要素宣言および 1 つの属性宣言が含まれています。

```
<xsd:complexType name="USAddress" >
   <xsd:sequence>
     <xsd:element name="name"   type="xsd:string"/>
     <xsd:element name="street" type="xsd:string"/>
     <xsd:element name="city"   type="xsd:string"/>
     <xsd:element name="state"  type="xsd:string"/>
     <xsd:element name="zip"    type="xsd:decimal"/>
   </xsd:sequence>
   <xsd:attribute name="country" type="xsd:NMTOKEN" fixed="US"/>
</xsd:complexType>
```

したがって、型が USAddress であると定義されている要素（po.xml 内の shipTo など）がインスタンス内に出現する場合、その要素は 5 つの要素と 1 つの属性で構成されている必要があります。これらの要素には、次の要件があります。

- 宣言の name 属性の値で指定されているとおり、name、street、city、state および zip という名前を持つ。
- 宣言された順序と同じ順序で出現する。これらの要素のうち最初の 4 つは文字列を含み、5 番目の要素は数値を含む。型が USAddress であると宣言されている要素は、文字列 US を含む country という属性を持つ場合があります。

USAddress の定義には、string、decimal および NMTOKEN という単純型を伴う宣言のみが含まれています。

PurchaseOrderType の定義

一方、PurchaseOrderType の定義には、USAddress などの複合型を伴う要素宣言が含まれています。ただし、型が単純型または複合型のどちらであるかに関係なく、どちらの宣言も同じ type 属性を使用して型を識別します。

```
<xsd:complexType name="PurchaseOrderType">
        <xsd:sequence>
            <xsd:element name="shipTo" type="USAddress"/>
             <xsd:element name="billTo" type="USAddress"/>
                  <xsd:element ref="comment" minOccurs="0"/>
              <xsd:element name="items"  type="Items"/>
         </xsd:sequence>
         <xsd:attribute name="orderDate" type="xsd:date"/>
  </xsd:complexType>
```

PurchaseOrderType の定義では、要素宣言のうちの 2 つ（shipTo および billTo に対するもの）は、異なる要素名を同じ複合型 USAddress に対応付けています。この定義の結果として、po.xml などのインスタンス・ドキュメント内に出現する、型が PurchaseOrderType であると宣言されている要素は shipTo および billTo という要素で構成されている必要があります。また、これらの各要素には、USAddress の一部で宣言された 5 つのサブ要素（name、street、city、state および zip）が含まれている必要があります。shipTo 要素および billTo 要素は、USAddress の一部として宣言された country 属性を持つこともできます。

PurchaseOrderType の定義には、orderDate 属性宣言が含まれています。この属性宣言は、country 属性宣言と同様に単純型を識別します。要素宣言とは異なり、属性は他の要素や他の属性を含むことができないため、すべての属性宣言は単純型を参照する必要があります。

前述の各要素宣言は、名前を既存の型定義と対応付けています。新しい要素を宣言するのではなく、既存の要素を使用する方が適切である場合があります。次に例を示します。

```
<xsd:element ref="comment" minOccurs="0"/>
```

この宣言は、発注書スキーマの他の箇所で宣言されている既存の要素 comment を参照しています。通常、ref 属性の値は、複合型定義の一部としてではなくスキーマの下で宣言されている、グローバル要素を参照する必要があります。この宣言の結果として、comment という要素がインスタンス・ドキュメント内に出現でき、その内容は要素の型（この場合は string）と一致している必要があります。

出現の制約 : minOccurs および maxOccurs

宣言内の minOccurs 属性の値が 0（ゼロ）であるため、PurchaseOrderType 内での comment 要素の出現はオプションです。通常、minOccurs の値が 1 以上である場合、その要素の出現は必須です。要素が出現できる最大回数は、宣言内の maxOccurs 属性の値によって決定されます。この値は 41 などの正の整数であるか、または出現回数に上限がないことを示す用語 unbounded です。minOccurs 属性と maxOccurs 属性のデフォルト値は、どちらも 1 です。

したがって、comment などの要素が maxOccurs 属性なしで宣言されている場合、その要素は複数回出現することはできません。minOccurs 属性に対してのみ値を指定する場合、その値が maxOccurs のデフォルト値以下（0 または 1）であることを確認してください。

同様に、maxOccurs 属性に対してのみ値を指定する場合、その値が minOccurs のデフォルト値以上（1 以上）であることを確認してください。両方の属性が省略されている場合、要素は 1 回のみ出現する必要があります。

属性は 1 回出現するか、または 1 回も出現しないかのどちらかです。そのため、属性の出現を指定する構文は、要素の構文とは異なります。特に、use 属性を使用して属性を宣言すると、その属性が required、optional または prohibited のいずれであるかを示すことができます。前述の po.xsd 内の partNum 属性宣言の例を示します。

```
<xsd:attribute name="partNum" type="SKU" use="required"/>
```

デフォルトの属性

属性および要素のデフォルト値は、どちらも default 属性を使用して宣言されます。ただし、属性の場合と要素の場合では、この属性を使用した結果が少し異なります。属性がデフォルト値を使用して宣言されている場合、その属性の値はインスタンス・ドキュメント内にその属性の値として出現する値です。その属性がインスタンス・ドキュメント内に出現しない場合は、スキーマ・プロセッサによって default 属性の値と同じ値を持つ属性が提供されます。

注意 : 属性のデフォルト値は、その属性自体がオプションである場合にのみ有効です。そのため、use に対してデフォルト値と optional 以外の値の両方を指定すると、エラーが発生します。

デフォルトの要素

スキーマ・プロセッサは、デフォルトの要素を少し異なる方法で処理します。要素がデフォルト値を使用して宣言されている場合、その要素の値はインスタンス・ドキュメント内にその要素の内容として出現する値です。

要素が内容を持たずに出現している場合は、スキーマ・プロセッサによって default 属性の値と等しい値を持つ要素が提供されます。ただし、要素がインスタンス・ドキュメント内に出現しない場合は、スキーマ・プロセッサによってその要素は提供されません。

次に、要素のデフォルトと属性のデフォルトの違いの概要を示します。

- デフォルトの属性値は、属性が欠落している場合に適用されます。
- デフォルトの要素値は、要素が空の場合に適用されます。

fixed 属性は、属性および要素が特定の値に設定されることを保証するために、属性宣言と要素宣言の両方で使用されます。たとえば、po.xsd には country 属性の宣言が含まれています。これは、固定値 US を使用して宣言されています。この宣言は、インスタンス・ドキュメント内での country 属性の出現がオプションである（use のデフォルト値が optional である）ことを意味します。ただし、属性が出現する場合、その値は US である必要があり、属性が出現しない場合は、スキーマ・プロセッサによって US という値を持つ country 属性が提供されます。

注意： 固定値およびデフォルト値という概念は、相互に排他的です。そのため、宣言に fixed 属性と default 属性の両方が含まれる場合は、エラーが発生します。

表 B-1 に、要素および属性の出現を制約するために要素宣言および属性宣言で使用される属性値の概要を示します。

表 B-1 XML Schema の要素および属性に対する出現制約

要素（minOccurs、maxOccurs）fixed、default	属性 use、fixed、default	備考
(1、1) -、-	required、-、-	要素 / 属性は 1 回のみ出現する必要があり、任意の値を持つことができます。
(1、1) 37、-	required、37、-	要素 / 属性は 1 回のみ出現する必要があり、その値は 37 である必要があります。
(2、unbounded) 37、-	利用不可	要素は 2 回以上出現する必要があり、その値は 37 である必要があります。通常、minOccurs および maxOccurs の値は正の整数である場合があり、maxOccurs の値は「unbounded」である場合があります。
(0、1) -、-	optional、-、-	要素 / 属性は 1 回出現でき、任意の値を持つことができます。
(0、1) 37、-	optional、37、-	要素 / 属性は 1 回出現できます。出現する場合、その値は 37 である必要があります。出現しない場合、その値は 37 です。

表 B-1 XML Schema の要素および属性に対する出現制約（続き）

要素 （minOccurs、maxOccurs） fixed、default	属性 use、fixed、default	備考
（0、1）-、37	optional、37、-	要素 / 属性は 1 回出現できます。出現しない場合、その値は 37 です。出現する場合、その値は指定された値です。
（0、2）-、37	利用不可	要素は 1 回または 2 回出現するか、あるいは 1 回も出現しない場合があります。要素が出現しない場合、要素は提供されません。要素が出現し、空である場合、その値は 37 です。それ以外の場合、その値は指定された値です。通常、minOccurs および maxOccurs の値は正の整数である場合があり、maxOccurs の値は「unbounded」である場合があります。
（0、0）-、-	prohibited、-、-	要素 / 属性は出現できません。

注意： minOccurs、maxOccurs または use のいずれも、グローバル要素およびグローバル属性の宣言には出現できません。

グローバル要素およびグローバル属性

グローバル要素およびグローバル属性は、schema 要素の子として出現する宣言によって作成されます。グローバル要素またはグローバル属性が宣言されると、それらは前述の項で説明した ref 属性を使用して 1 つ以上の宣言で参照できます。

グローバル要素を参照する宣言を使用すると、参照される要素がインスタンス・ドキュメント内の参照する宣言のコンテキストに出現できます。そのため、たとえば、comment 要素を参照する宣言が複合型定義内で shipTo、billTo および items 要素の宣言と同じレベルに出現することから、comment は po.xml 内で他の 3 要素と同じレベルに出現できます。

グローバル要素の宣言を使用すると、要素はインスタンス・ドキュメントの最上位に出現することもできます。そのため、po.xsd でグローバル要素として宣言されている purchaseOrder は、po.xml 内で最上位要素として出現できます。

注意： この理由から、comment 要素も po.xml などのドキュメント内に最上位要素として出現できます。

注意事項： 1 つ目の注意事項は、グローバル宣言は参照を含むことができず、単純型および複合型を直接識別する必要があることです。グローバル宣言は ref 属性を含むことができず、type 属性を使用するか、またはグローバル宣言の後に名前を持たない型が続く必要があります。

2 つ目の注意事項は、カーディナリティ制約はグローバル宣言を参照するローカル宣言には使用できますが、グローバル宣言には使用できないことです。したがって、グローバル宣言は、属性 minOccurs、maxOccurs または use を含むことができません。

ネーミングの競合

前述の項では、次の操作を行う方法について説明しました。

- PurchaseOrderType などの新しい複合型の定義
- purchaseOrder などの要素の宣言
- orderDate などの属性の宣言

これらの操作には、ネーミングを伴います。2 つのものに同じ名前が与えられた場合、通常、その 2 つが類似しているほど、ネーミングの競合が発生する可能性が高くなります。

次に例を示します。

2 つのものがどちらも型（USStates という複合型と USStates という単純型など）である場合は、競合が発生します。

2 つのものが型と要素または型と属性である場合（USAddress という複合型を定義し、USAddress という要素を宣言する場合など）、競合は発生しません。

2 つのものが異なる型内の要素であり、グローバル要素ではない場合（name という 1 つの要素を USAddress 型の一部として宣言し、name という 2 つ目の要素を Item 型の一部として宣言する場合など）、競合は発生しません。このような要素は、ローカル要素宣言といいます。

2 つのものがどちらも型であり、ユーザーがその 1 つを定義し、もう 1 つが XML Schema によって定義されている場合（ユーザーが decimal という単純型を定義した場合など）、競合は発生しません。最後の例で明らかな矛盾が存在する理由は、2 つの型が異なる名前空間に属していることです。名前空間については、C-17 ページの「W3C の XML Namespace 勧告の概要」を参照してください。

単純型

発注書スキーマは、単純型を持ついくつかの要素および属性を宣言します。これらの単純型の一部（string や decimal など）は XML Schema に組込み済であり、他の単純型は組込み済の単純型から派生したものです。

たとえば、partNum 属性は、string から派生した SKU（Stock Keeping Unit（在庫商品識別番号））という型を持ちます。組込み単純型とその派生型は、どちらもすべての要素宣言および属性宣言で使用できます。表 B-2 に、XML Schema に組込み済のすべての単純型と各型の例を示します。

表 B-2 XML Schema に組込み済の単純型

単純型	例	備考
string	Confirm this is electric	--
normalizedString	Confirm this is electric	(3)
token	Confirm this is electric	(4)
byte	-1、126	(2)
unsignedByte	0、126	(2)
base64Binary	GpM7	--
hexBinary	0FB7	--
integer	-126789、-1、0、1、126789	(2)
positiveInteger	1、126789	(2)
negativeInteger	-126789、-1	(2)
nonNegativeInteger	0、1、126789	(2)
nonPositiveInteger	-126789、-1、0	(2)
int	-1、126789675	(2)
unsignedInt	0、1267896754	(2)
long	-1、12678967543233	(2)
unsignedLong	0、12678967543233	(2)
short	-1、12678	(2)
unsignedShort	0、12678	(2)
decimal	-1.23、0、123.4、1000.00	(2)
float	-INF、-1E4、-0、0、12.78E-2、12、INF、NaN	単精度の 32 ビット浮動小数点と同等。NaN は数値以外を意味する。「注意」の（2）を参照。
double	-INF、-1E4、-0、0、12.78E-2、12、INF、NaN	倍精度の 64 ビット浮動小数点と同等。「注意」の（2）を参照。
boolean	true、false 1、0	--
time	13:20:00.000、13:20:00.000-05:00	(2)

表 B-2 XML Schema に組込み済の単純型（続き）

単純型	例	備考
dateTime	1999-05-31T13:20:00.000-05:00	協定世界時（UTC）から 5 時間遅れの東部標準時（EST）における 1999 年 5 月 31 日午後 1 時 20 分。「注意」の（2）を参照。
duration	P1Y2M3DT10H30M12.3S	1 年 2 か月 3 日と 10 時間 30 分 12.3 秒。
date	1999-05-31	(2)
gMonth	--05--	5 月。「注意」の（2）および（5）を参照。
gYear	1999	1999 年。「注意」の（2）および（5）を参照。
gYearMonth	1999-02	1999 年 2 月。日数は関係なし。「注意」の（2）および（5）を参照。
gDay	---31	31 日。「注意」の（2）および（5）を参照。
gMonthDay	--05-31	毎年 5 月 31 日。「注意」の（2）および（5）を参照。
Name	shipTo	XML 1.0 の Name 型。
QName	po:USAddress	XML Namespace の QName。
NCName	USAddress	XML Namespace の NCName（接頭辞とコロンのない QName）。
anyURI	http://www.example.com/、http://www.example.com/doc.html#ID5	--
language	en-GB、en-US、fr	XML 1.0 で定義されている xml:lang の有効値。
ID	--	XML 1.0 の属性の型 ID。「注意」の（1）を参照。

表 B-2 XML Schema に組込み済の単純型（続き）

単純型	例	備考
IDREF	--	XML 1.0 の属性の型 IDREF。「注意」の（1）を参照。
IDREFS	--	XML 1.0 の属性の型 IDREFS。（1）を参照。
ENTITY	--	XML 1.0 の属性の型 ENTITY。「注意」の（1）を参照。
ENTITIES	--	XML 1.0 の属性の型 ENTITIES。「注意」の（1）を参照。
NOTATION	--	XML 1.0 の属性の型 NOTATION。「注意」の（1）を参照。
NMTOKEN	US、Brasil	XML 1.0 の属性の型 NMTOKEN。「注意」の（1）を参照。
NMTOKENS	US UK、Brasil Canada Mexique	XML 1.0 の属性の型 NMTOKENS（空白で区切られた NMTOKEN のリスト）。「注意」の（1）を参照。

注意：

（1）XML Schema と XML 1.0 DTD の間の互換性を維持するために、単純型 `ID`、`IDREF`、`IDREFS`、`ENTITY`、`ENTITIES`、`NOTATION`、`NMTOKEN`、`NMTOKENS` は属性内でのみ使用する必要があります。

（2）この型の値は、複数の字句形式によって表すことができます。たとえば、100 と 1.0E2 はどちらも 100 を表す有効な浮動形式です。ただし、この型に対して、正規の字句形式を定義する規則が確立されています。『XML Schema Part 2』を参照してください。

（3）normalizedString 型内の改行、タブおよび復帰改行文字は、スキーマ処理の前に空白文字に変換されます。

（4）normalizedString と同様。また、連続した空白文字は単一の空白文字に置き換えられ、先行および後続空白文字は削除されます。

（5）「g」という接頭辞は、グレゴリオ暦における時間間隔を示します。

新しい単純型は、既存の単純型（組込み型および派生型）から派生させることによって定義されます。特に、既存の単純型を制限することによって、新しい単純型を派生させることができます。新しい型の値の有効範囲は、既存の型に対する値の範囲のサブセットになります。

新しい単純型を定義し、その名前を指定するには、simpleType 要素を使用します。既存の（ベース）型を示したり、値の範囲を制約する「ファセット」を識別するには、restriction 要素を使用します。ファセットの完全なリストについては、http://www.w3.org/TR/xmlschema-0/ にある『XML Schema Part 0: Primer（入門書）』の付録 B を参照してください。

値の範囲が 10000 ～ 99999（10000 と 99999 を含む）の myInteger という新しい整数型を作成するとします。これを組込み単純型 integer に基づいて定義します。integer の値の範囲には、10000 未満の整数および 99999 より大きい整数も含まれています。

myInteger を定義するには、minInclusive および maxInclusive という 2 つのファセットを使用して、integer ベース型の範囲を制限します。

範囲が 10000 ～ 99999 の myInteger の定義

```
<xsd:simpleType name="myInteger">
       <xsd:restriction base="xsd:integer">
           <xsd:minInclusive value="10000"/>
            <xsd:maxInclusive value="99999"/>
        </xsd:restriction>
</xsd:simpleType>
```

この例は、myInteger を定義するために使用されたベース型と 2 つのファセットの特定の組合せを示しています。その他の様々な組合せについては、組込み単純型とそのファセットのリストを参照してください。

発注書スキーマには、別のより複雑な単純型定義の例が含まれています。SKU という新しい単純型が（制限によって）単純型 string から派生しています。さらに、pattern というファセットを正規表現 "\d{3}-[A-Z]{2}" とともに使用すると、SKU の値を制約できます。この正規表現は、3 つの数字の後にハイフンが続き、その後に 2 つの大文字の ASCII 文字が続くことを示しています。

単純型 SKU の定義

```
<xsd:simpleType name="SKU">
    <xsd:restriction base="xsd:string">
      <xsd:pattern value="\d{3}-[A-Z]{2}"/>
    </xsd:restriction>
</xsd:simpleType>
```

この正規表現言語の詳細は、http://www.w3.org/TR/xmlschema-0/ にある付録 D を参照してください。

XML Schema は、http://www.w3.org/TR/xmlschema-0/ にある付録 B にリストされている 14 のファセットを定義しています。これらのファセットのうち、enumeration（列挙）ファセットは特に有効です。このファセットは、boolean 型を除く、ほぼすべての単純型の値を制約するために使用できます。enumeration ファセットは、単純型を一連の個別値に制限します。たとえば、enumeration ファセットを使用して、string から派生した USState という新しい単純型を定義できます。この単純型の値は、米国の州の標準的な略称である必要があります。

enumeration ファセットの使用

```
<xsd:simpleType name="USState">
       <xsd:restriction base="xsd:string">
           <xsd:enumeration value="AK"/>
           <xsd:enumeration value="AL"/>
            <xsd:enumeration value="AR"/>
             <!-- and so on ... -->
        </xsd:restriction>
</xsd:simpleType>
```

state 要素宣言で現在使用されている string 型は、USState で置き換えることをお薦めします。これによって、state 要素（billTo および shipTo の state サブ要素）の有効値は、AK、AL、AR などのいずれかに制限されます。特定の型に対して指定された enumeration の値は一意である必要があります。

リスト型

XML Schema には、表 B-3 に示すほとんどの型を構成するアトム型の他に、リスト型の概念があります。アトム型、リスト型および次の項で説明する共用体型は、総称して単純型といいます。アトム型の値は、XML Schema の観点から見ると、分割できないものです。たとえば、NMTOKEN の値 US は、文字「S」などの US の一部は単独では何の意味もないという点で、分割できないものです。一方、リスト型はアトム型のシーケンスで構成されています。したがって、シーケンスの一部（「アトム」）には意味があります。たとえば、NMTOKENS はリスト型であり、この型の要素は、「US UK FR」など、空白で区切られた NMTOKEN のリストになります。XML Schema には、次の 3 つの組込みリスト型があります。

- NMTOKENS
- IDREFS
- ENTITIES

組込みリスト型を使用する他に、既存のアトム型から派生させることによって、新しいリスト型を作成できます。ただし、既存のリスト型または複合型からリスト型を作成することはできません。次に、myInteger のリストの作成例を示します。

myInteger のリストの作成

```
<xsd:simpleType name="listOfMyIntType">
     <xsd:list itemType="myInteger"/>
</xsd:simpleType>
```

内容が listOfMyIntType に準拠しているインスタンス・ドキュメント内の要素は、次のとおりです。

```
<listOfMyInt>20003 15037 95977 95945</listOfMyInt>
```

リスト型には、ファセット length、minLength、maxLength および enumeration を適用できます。たとえば、米国の 6 つの州のみで構成されるリスト（SixUSStates）を定義するには、最初に USState の USStateList という新しいリスト型を定義し、次に USStateList を 6 つの項目のみに制限することによって SixUSStates を派生させます。

米国の 6 つの州を表すリスト型

```
<xsd:simpleType name="USStateList">
    <xsd:list itemType="USState"/>
</xsd:simpleType>
<xsd:simpleType name="SixUSStates">
   <xsd:restriction base="USStateList">
      <xsd:length value="6"/>
   </xsd:restriction>
</xsd:simpleType>
```

型が SixUSStates である要素には、6 つの項目が含まれている必要があります。この 6 項目のそれぞれは、列挙型 USState の（アトム）値の 1 つである必要があります。次に例を示します。

```
<sixStates>PA NY CA NY LA AK</sixStates>
```

リスト型をアトム型 string から派生させることができることに注意してください。ただし、string には空白が含まれる場合があり、空白はリスト型の項目を区切るため、ベース型が string であるリスト型を使用する場合には注意が必要です。たとえば、length ファセットが 3 で、ベース型が string であるリスト型を定義したとします。その場合、次の 3 項目リストは有効です。

Asia Europe Africa

ただし、次の 3 項目リストは無効です。

Asia Europe America Latin

「America Latin」はリスト外では単一の文字列として存在する場合がありますが、それがリスト内に挿入された場合、America と Latin の間の空白によって、事実上、4 番目の項目が作成されます。そのため、後者の例は 3 項目リスト型に準拠しません。

共用体型

アトム型およびリスト型によって、要素や属性値を 1 つのアトム型の 1 つ以上のインスタンスにすることができます。一方、共用体型によって、要素や属性値を複数のアトム型とリスト型の共用体から導出された 1 つの型の 1 つ以上のインスタンスにすることができます。参考に、米国の州を単一文字の略称または数値コードのリストとして表す共用体型を作成します。zipUnion 共用体型は、1 つのアトム型と 1 つのリスト型から作成されます。

zip コードのための共用体型

```
<xsd:simpleType name="zipUnion">
     <xsd:union memberTypes="USState listOfMyIntType"/>
</xsd:simpleType>
```

共用体型を定義する場合、memberTypes 属性値が共用体内のすべての型のリストです。

zipUnion 型の zips という要素を宣言したとすると、要素の妥当なインスタンスは次のとおりです。

```
<zips>CA</zips>
 <zips>95630 95977 95945</zips>
<zips>AK</zips>
```

2 つのファセット pattern および enumeration を共用体型に適用できます。

名前を持たない型定義

スキーマは、PurchaseOrderType などの名前が付いた型のセットを定義し、type= 構成を使用してその型を参照する要素（purchaseOrder など）を宣言することによって構成できます。このスタイルのスキーマ構成は単純ですが、特に、1 回のみ参照され、制約をほとんど含まない多くの型を定義する場合には効率的ではないことがあります。これらの場合、型を名前を持たない型としてより簡潔に定義できます。これによって、型に名前を付け、明示的に参照する必要があるというオーバーヘッドを回避できます。

po.xsd の Items 型の定義には、名前を持たない型を使用する 2 つの要素宣言（item および quantity）が含まれています。通常、要素（または属性）宣言内に type= が存在しないことによって、および名前のない（単純または複合）型定義が存在することによって、名前を持たない型を識別できます。

2 つの名前を持たない型の定義

```
<xsd:complexType name="Items">
    <xsd:sequence>
        <xsd:element name="item" minOccurs="0" maxOccurs="unbounded">
            <xsd:complexType>
              <xsd:sequence>
                <xsd:element name="productName" type="xsd:string"/>
                  <xsd:element name="quantity">
                   <xsd:simpleType>
                    <xsd:restriction base="xsd:positiveInteger">
                     <xsd:maxExclusive value="100"/>
                    </xsd:restriction>
                   </xsd:simpleType>
                  </xsd:element>
                <xsd:element name="USPrice"  type="xsd:decimal"/>
                <xsd:element ref="comment"   minOccurs="0"/>
                <xsd:element name="shipDate" type="xsd:date" minOccurs="0"/>
               </xsd:sequence>
              <xsd:attribute name="partNum" type="SKU" use="required"/>
            </xsd:complexType>
        </xsd:element>
    </xsd:sequence>
</xsd:complexType>
```

item 要素の場合、要素 productName、quantity、USPrice、comment および shipDate と、partNum という属性で構成される無名複合型を持ちます。quantity 要素の場合、値の範囲が 1 ～ 99 の integer から派生した無名単純型を持ちます。

要素内容

発注書スキーマには、他の要素を含む要素（items など）および属性を持ち、他の要素を含む要素（shipTo など）および単純型の値のみを含む要素（USPrice など）の多くの例が含まれています。ただし、属性を持つが単純型の値のみを含む要素、他の要素と文字内容が混在した要素、または内容を持たない要素は含まれていません。この項では、これらの要素の内容モデルについて示します。

単純型から派生した複合型

属性を持ち、単純値を含む要素を宣言する方法について考えてみます。インスタンス・ドキュメントでは、このような要素は次のような形式で出現する場合があります。

```
<internationalPrice currency="EUR">423.46</internationalPrice>
```

発注書スキーマは、開始点である USPrice 要素を次のとおり宣言しています。

```
<xsd:element name="USPrice" type="decimal"/>
```

この場合、属性をこの要素に追加するにはどうすればよいのでしょうか？前述のとおり、単純型は属性を持つことができず、decimal は単純型です。

したがって、属性宣言を行うには、複合型を定義する必要があります。また、内容は単純型 decimal にする必要があります。この場合、単純型 decimal に基づく複合型を定義するにはどうすればよいのでしょうか？これを行うには、単純型 decimal から新しい複合型を派生させます。

単純型からの複合型の派生

```
<xsd:element name="internationalPrice">
   <xsd:complexType>
        <xsd:simpleContent>
             <xsd:extension base="xsd:decimal">
                  <xsd:attribute name="currency" type="xsd:string"/>
             </xsd:extension>
         </xsd:simpleContent>
    </xsd:complexType>
</xsd:element>
```

新しい（名前を持たない）型を定義するには、complexType 要素を使用します。新しい型の内容モデルに文字データのみが含まれ、要素が含まれないことを指定するために、simpleContent 要素を使用します。最後に、単純型 decimal を拡張することによって、新しい型を派生させます。この拡張は、標準属性宣言を使用した currency 属性の追加で構成されます（型の派生の詳細は、『XML Schema Part 0: Primer（入門書）』の第 4 項を参照）。このように宣言された internationalPrice 要素は、この項の前半に記載されている例に示すインスタンス内に出現します。

混合内容

発注書スキーマの構成は、サブ要素を含む要素があるという特徴があります。最も深いサブ要素には、文字データが含まれます。XML Schema は、文字データがサブ要素とともに出現できるスキーマの構成も提供します。文字データの使用は、最も深いサブ要素には限定されません。

発注書と同じいくつかの要素を使用する次の顧客への手紙の一部について考えてみます。

顧客への手紙の一部

```
<letterBody>
     <salutation>Dear Mr.<name>Robert Smith</name>.</salutation>
     Your order of <quantity>1</quantity> <productName>Baby
      Monitor</productName> shipped from our warehouse on
      <shipDate>1999-05-21</shipDate>. ....
</letterBody>
```

要素の間とその子要素の間に出現しているテキストに注意してください。たとえば、テキストは、要素 salutation、quantity、productName および shipDate の間に出現しています。これらのすべての要素は letterBody の子です。また、テキストは、letterBody の孫である要素 name の周囲に出現しています。次のスキーマの一部は、letterBody を宣言します。

顧客への手紙のスキーマの一部

```
<xsd:element name="letterBody">
   <xsd:complexType mixed="true">
       <xsd:sequence>
           <xsd:element name="salutation">
              <xsd:complexType mixed="true">
                <xsd:sequence>
                  <xsd:element name="name" type="xsd:string"/>
                </xsd:sequence>
              </xsd:complexType>
           </xsd:element>
           <xsd:element name="quantity"    type="xsd:positiveInteger"/>
           <xsd:element name="productName" type="xsd:string"/>
           <xsd:element name="shipDate"    type="xsd:date" minOccurs="0"/>
           <!-- and so on -->
       </xsd:sequence>
   </xsd:complexType>
</xsd:element>
```

顧客への手紙に出現している要素が宣言され、その型が前述の element および complexType 要素構成を使用して定義されています。文字データが letterBody の子要素の間に出現できるようにするために、型定義の mixed 属性が TRUE に設定されています。

XML Schema の混合モデルが XML 1.0 の混合モデルとは基本的に異なることに注意してください。XML Schema の混合モデルでは、インスタンス内に出現する子要素の順序と数が、モデルで指定された子要素の順序と数と一致している必要があります。一方、XML 1.0 の混合モデルでは、インスタンス内に出現する子要素の順序と数を制約できません。XML 1.0 が提供する部分的なスキーマ検証とは対照的に、XML Schema は混合モデルの完全な検証を提供します。

空内容

internationalPrice 要素が、通貨単位と価格を属性と内容値に分けて持つのではなく、両方を属性値として持つようにする必要があるとします。次に例を示します。

```
<internationalPrice currency="EUR" value="423.46"/>
```

このような要素には内容がなく、その内容モデルは空です。

空複合型

内容が空である型を定義するには、基本的に、内容に要素のみを使用できる型を定義します。ただし、実際には、要素は 1 つも宣言しないため、その型の内容モデルは空になります。

```
<xsd:element name="internationalPrice">
    <xsd:complexType>
        <xsd:complexContent>
          <xsd:restriction base="xsd:anyType">
              <xsd:attribute name="currency" type="xsd:string"/>
              <xsd:attribute name="value"    type="xsd:decimal"/>
          </xsd:restriction>
        </xsd:complexContent>
    </xsd:complexType>
</xsd:element>
```

この例では、complexContent を持つ（名前を持たない）型、すなわち要素のみを定義しています。complexContent 要素は、複合型の内容モデルを制限または拡張することを意図していることを示しています。anyType の restriction は 2 つの属性を宣言していますが、要素内容は導入していません（制限の詳細は、『XML Schema Part 0: Primer（入門書）』の第 4.4 項を参照）。このように宣言された internationalPrice 要素は、前述の例に示すとおり、インスタンス内に正当に出現できます。

空複合型の略記

前述の空内容要素の構文は比較的冗長であるため、internationalPrice 要素をより簡潔に宣言できます。

```
<xsd:element name="internationalPrice">
    <xsd:complexType>
      <xsd:attribute name="currency" type="xsd:string"/>
        <xsd:attribute name="value"    type="xsd:decimal"/>
    </xsd:complexType>
</xsd:element>
```

simpleContent または complexContent なしに定義された複合型は、anyType を制限する複合内容の略記として解釈されるため、このように簡潔に構文を指定しても機能します。

AnyType

anyType は、ur-type という抽象的概念を表します。ur-type は、すべての単純型および複合型が派生するベース型です。anyType 型は、その内容を制約しません。anyType は、他の型と同様に使用できます。次に例を示します。

```
<xsd:element name="anything" type="xsd:anyType"/>
```

このように宣言された要素の内容は制約されないため、要素の値は 423.46、他の文字列、または文字と要素が混在していてもかまいません。実際に、anyType は何も指定されなかった場合のデフォルトの型であるため、前述の例は次のように書くこともできます。

```
<xsd:element name="anything"/>
```

たとえば、国際化をサポートするために埋込みマークアップを必要とする文章を含む要素の場合など、制約されない要素内容が必要な場合は、デフォルトの宣言またはそれを少し制限した形式が適していることがあります。『XML Schema Part 0: Primer（入門書）』の第 5.5 項に記載されている text 型は、このような目的に適した型の例です。

注釈

XML Schema は、ユーザーとアプリケーションの両方にメリットがあるように、スキーマに注釈を付けるための 3 つの要素を提供します。発注書スキーマでは、基本的なスキーマ記述および著作権情報が documentation 要素内に挿入されています。documentation 要素は、ユーザーが判読可能な情報を挿入する位置として推薦されています。情報の言語を示すために、すべての documentation 要素で xml:lang 属性を使用することをお薦めします。または、xml:lang 属性を schema 要素に置くことによって、スキーマ内のすべての情報の言語を示すこともできます。

発注書スキーマで使用されていない appInfo 要素を使用すると、ツール、スタイルシートおよび他のアプリケーションに情報を提供できます。appInfo を使用した例として、『XML Schema Part 2: Datatypes（データ型）』の単純型を記述するスキーマがあります。

このスキーマを記述する情報（特定の単純型に適用可能なファセットなど）は、appInfo 要素内で表現されています。この情報は、『XML Schema Part 2』ドキュメント用のテキストを自動的に生成するために、アプリケーションによって使用されています。

documentation と appInfo は、どちらも annotation のサブ要素として出現します。annotation 自体は、ほぼすべてのスキーマ構成の先頭に出現する場合があります。次の例は、要素宣言および複合型定義の先頭に出現する annotation 要素を示します。

要素宣言および複合型定義の注釈

```
<xsd:element name="internationalPrice">
   <xsd:annotation>
     <xsd:documentation xml:lang="en">
         element declared with anonymous type
     </xsd:documentation>
   </xsd:annotation>
   <xsd:complexType>
      <xsd:annotation>
        <xsd:documentation xml:lang="en">
            empty anonymous type with 2 attributes
        </xsd:documentation>
      </xsd:annotation>
      <xsd:complexContent>
        <xsd:restriction base="xsd:anyType">
            <xsd:attribute name="currency" type="xsd:string"/>
            <xsd:attribute name="value"    type="xsd:decimal"/>
        </xsd:restriction>
      </xsd:complexContent>
   </xsd:complexType>
</xsd:element>
```

annotation 要素は、要素 schema、simpleType および attribute によって示される他のスキーマ構成の先頭に出現する場合もあります。

内容モデルの構築

発注書スキーマ内のすべての複合型定義は、インスタンス・ドキュメント内に出現する必要がある要素のシーケンスを宣言します。これらの型の内容モデルで宣言された個々の要素の出現は、属性 minOccurs の値を（コメントなどで）0（ゼロ）にすることによって指定されるとおり、オプションである場合があります。それ以外の場合は、minOccurs および maxOccurs の値に応じて制約される場合があります。

XML Schema は、内容モデル内に出現する要素のグループに適用される制約も提供します。これらの制約は、XML 1.0 で使用可能な制約およびいくつかの追加制約を反映しています。制約は属性には適用されないことに注意してください。

XML Schema では、要素のグループを定義し、それに名前を付けることができます。そのため、要素を使用して、複合型の内容モデルを構築できます（これは、XML 1.0 のパラメータ・エンティティの一般的な使用方法の擬似実行です)。要素の名前のないグループも定義できます。名前のあるグループ内の要素とともに、要素の名前のないグループも宣言された順序と同じ順序で出現するように制約できます。また、要素の 1 つのみがインスタンス内に出現できるように制約することもできます。

発注書が納品先と請求先が異なる場合の住所、または納品先と請求先が同じである場合の単一の住所を含むことができるように、発注書スキーマの PurchaseOrderType 定義に 2 つのグループを導入する場合について考えてみます。

ネストした choice グループおよび sequence グループ

```
<xsd:complexType name="PurchaseOrderType">
  <xsd:sequence>
     <xsd:choice>
       <xsd:group   ref="shipAndBill"/>
          <xsd:element name="singleUSAddress" type="USAddress"/>
     </xsd:choice>
     <xsd:element ref="comment" minOccurs="0"/>
        <xsd:element name="items"  type="Items"/>
  </xsd:sequence>
     <xsd:attribute name="orderDate" type="xsd:date"/>
</xsd:complexType>

  <xsd:group name="shipAndBill">
     <xsd:sequence>
        <xsd:element name="shipTo" type="USAddress"/>
          <xsd:element name="billTo" type="USAddress"/>
     </xsd:sequence>
</xsd:group>
```

choice グループ要素を使用すると、その子の 1 つのみがインスタンス内に出現できます。1 つ目の子は、要素順序 shipTo、billTo で構成される名前付きのグループ shipAndBill を参照する内部 group 要素です。2 つ目の子は、asingleUSAddress です。したがって、インスタンス・ドキュメント内では、purchaseOrder 要素は、billTo 要素が後に続く shipTo 要素か、または singleUSAddress 要素のどちらかを含んでいる必要があります。choice グループの後には、comment および items 要素宣言が続きます。choice グループと要素宣言は、どちらも sequence グループの子です。これらの様々なグループが存在することによって、住所に関連する要素の後には comment 要素および items 要素がこの順序で続く必要があります。

グループ内の要素を制約する 3 番目のオプションが存在します。グループ内のすべての要素が 1 回のみ出現するか、または 1 回も出現しません。また、それらの要素はどのような順序でも出現できます。SGML &-Connector の簡略化バージョンを提供する all グループの使用は、任意の内容モデルの最上位に制限されています。

さらに、グループのすべての子は（グループではなく）個別の要素である必要があります。また、内容モデル内の要素は複数回出現することはできず、minOccurs および maxOccurs の許容値は 0（ゼロ）および 1 です。

たとえば、purchaseOrder の子要素がどのような順序でも出現できるようにするには、PurchaseOrderType を次のとおり再定義します。

all グループ

```
<xsd:complexType name="PurchaseOrderType">
    <xsd:all>
       <xsd:element name="shipTo" type="USAddress"/>
          <xsd:element name="billTo" type="USAddress"/>
            <xsd:element ref="comment" minOccurs="0"/>
             <xsd:element name="items"  type="Items"/>
     </xsd:all>
     <xsd:attribute name="orderDate" type="xsd:date"/>
</xsd:complexType>
```

この定義によって、comment 要素はオプションで purchaseOrder 内に出現できます。comment 要素は、どの shipTo、billTo および items 要素の前後にも出現できますが、出現できるのは 1 回のみです。さらに、all グループでは、comment などの要素が複数回出現できるようにする方法としてグループ外で宣言することは許可されていません。XML Schema では、all グループが内容モデルの最上位に唯一の子として出現する必要があると規定しています。したがって、次の例は無効です。

all グループを使用した無効な例

```
<xsd:complexType name="PurchaseOrderType">
   <xsd:sequence>
       <xsd:all>
          <xsd:element name="shipTo" type="USAddress"/>
             <xsd:element name="billTo" type="USAddress"/>
             <xsd:element name="items"  type="Items"/>
       </xsd:all>
    <xsd:sequence>
      <xsd:element ref="comment" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
   </xsd:sequence>
     <xsd:attribute name="orderDate" type="xsd:date"/>
</xsd:complexType>
```

また、内容モデル内に出現する名前付きのグループおよび名前のないグループ（それぞれ、group、choice、sequence、all で表されます）は、minOccurs 属性および maxOccurs 属性を持つ場合があります。XML Schema によって提供される様々なグループを組み合せたり、ネストすることによって、および minOccurs と maxOccurs の値を設定することに

よって、XML 1.0 DTD で表現可能なすべての内容モデルを表現できます。さらに、all グループは追加の表現機能を提供します。

属性グループ

発注書内の各商品のより詳細な情報（各商品の重さや指定の出荷方法など）を提供するには、weightKg および shipBy 属性宣言を item 要素の（名前を持たない）型定義に追加します。

インライン型定義への属性の追加

```
<xsd:element name="Item" minOccurs="0" maxOccurs="unbounded">
   <xsd:complexType>
      <xsd:sequence>
          <xsd:element   name="productName" type="xsd:string"/>
             <xsd:element   name="quantity">
                <xsd:simpleType>
                  <xsd:restriction base="xsd:positiveInteger">
                    <xsd:maxExclusive value="100"/>
                  </xsd:restriction>
                </xsd:simpleType>
             </xsd:element>
             <xsd:element name="USPrice"  type="xsd:decimal"/>
                <xsd:element ref="comment"   minOccurs="0"/>
                  <xsd:element name="shipDate" type="xsd:date" minOccurs="0"/>
      </xsd:sequence>
         <xsd:attribute name="partNum"  type="SKU" use="required"/>
           <!-- add weightKg and shipBy attributes -->
              <xsd:attribute name="weightKg" type="xsd:decimal"/>
              <xsd:attribute name="shipBy">
                <xsd:simpleType>
                  <xsd:restriction base="xsd:string">
                    <xsd:enumeration value="air"/>
                    <xsd:enumeration value="land"/>
                    <xsd:enumeration value="any"/>
                  </xsd:restriction>
                 </xsd:simpleType>
                </xsd:attribute>
               </xsd:complexType>
             </xsd:element>
```

また、item 要素の必要なすべての属性を含む名前付きの属性グループを作成し、このグループを item 要素宣言内にその名前を指定することによって、このグループを参照することもできます。

属性グループを使用した属性の追加

```
<xsd:element name="item" minOccurs="0" maxOccurs="unbounded">
     <xsd:complexType>
          <xsd:sequence>
               <xsd:element name="productName" type="xsd:string"/>
               <xsd:element name="quantity">
                <xsd:simpleType>
                 <xsd:restriction base="xsd:positiveInteger">
                  <xsd:maxExclusive value="100"/>
                 </xsd:restriction>
                </xsd:simpleType>
               </xsd:element>
               <xsd:element name="USPrice"  type="xsd:decimal"/>
               <xsd:element ref="comment"   minOccurs="0"/>
               <xsd:element name="shipDate" type="xsd:date" minOccurs="0"/>
              </xsd:sequence>

              <!-- attributeGroup replaces individual declarations -->
              <xsd:attributeGroup ref="ItemDelivery"/>
             </xsd:complexType>
            </xsd:element>

            <xsd:attributeGroup name="ItemDelivery">
              <xsd:attribute name="partNum"  type="SKU" use="required"/>
              <xsd:attribute name="weightKg" type="xsd:decimal"/>
              <xsd:attribute name="shipBy">
                <xsd:simpleType>
                 <xsd:restriction base="xsd:string">
                  <xsd:enumeration value="air"/>
                  <xsd:enumeration value="land"/>
                  <xsd:enumeration value="any"/>
          </xsd:restriction>
       </xsd:simpleType>
   </xsd:attribute>
</xsd:attributeGroup>
```

この方法で属性グループを使用すると、属性グループを 1 箇所で定義および編集し、複数の定義および宣言で参照できるため、スキーマの可読性が向上し、スキーマの更新が簡単になります。属性グループにはこれらの特長があるため、XML 1.0 のパラメータ・エンティティと類似しています。属性グループが他の属性グループを含むことができることに注意してください。また、属性宣言と属性グループ参照は、どちらも複合型定義の最後に出現する必要があることにも注意してください。

nil 値

po.xml に示されている発注書の商品の 1 つである Lawnmower（芝刈り機）には、shipDate 要素がありません。この使用例のコンテキストでは、スキーマの作者は、未出荷の商品を示すためにこの要素を含めなかった可能性があります。ただし、通常、要素の欠落は特定の意味を持ちません。情報が不明であるか、または情報を適用できないことを示している場合があります。または、その他の理由で要素が欠落している場合もあります。要素の欠落によってではなく、要素を使用して、未出荷の商品、不明な情報または適用できない情報を明示的に表す方が望ましい場合があります。

たとえば、存在する要素を使用して、リレーショナル・データベースとの間で送受信される NULL 値を表す方が望ましい場合があります。このような場合は、XML Schema の nil メカニズムを使用して表現できます。このメカニズムを使用すると、nil 以外の値を持つ要素や nil 以外の値を持たない要素が出現できるようになります。

XML Schema の nil メカニズムには、バンド外の nil シグナルが含まれます。そのため、要素内容として出現する実際の nil 値が存在するのではなく、要素内容が nil であることを示す属性が存在します。参考に、nil を通知できるように、shipDate 要素宣言を次のとおり変更します。

```
<xsd:element name="shipDate" type="xsd:date" nillable="true"/>
```

また、インスタンス・ドキュメント内で shipDate が nil 値を持つことを明示的に表現するために、インスタンス用の XML Schema 名前空間に由来する nil 属性を TRUE に設定します。

```
<shipDate xsi:nil="true"></shipDate>
```

nil 属性は、インスタンス用の XML Schema 名前空間である http://www.w3.org/2001/XMLSchema-instance の一部として定義されているため、その名前空間に関連付けられた接頭辞（xsi: など）を使用してインスタンス・ドキュメント内に出現する必要があります。（xsd: 接頭辞の場合と同様に、xsi: 接頭辞も慣習的にのみ使用されています。）nil メカニズムは要素の値にのみ適用され、属性値には適用されないことに注意してください。xsi:nil="true" を持つ要素は、要素内容は持たず、属性は持っている場合があります。

DTD と XML Schema との相違

DTD は、XML 1.0 が提供する、XML マークアップの制約を宣言するメカニズムです。DTD によって、次のものを指定できます。

- XML 文書に出現できる要素
- 要素に含めることができる要素（またはサブ要素）
- 要素が出現できる順序

XML Schema は DTD と同様の目的を果たしますが、XML 文書の制約の指定についてはより柔軟であり、特定のアプリケーションではより有効である可能性があります。

XML の例

次の XML 文書について考えてみます。

```
<?xml version="1.0">
<publisher pubid="ab1234">
  <publish-year>2000</publish-year>
  <title>The Cat in the Hat</title>
  <author>Dr. Seuss</author>
  <artist>Ms. Seuss</artist>
  <isbn>123456781111</isbn>
</publisher>
```

DTD の例

前述の XML 文書に対する典型的な DTD について考えてみます。

```
<!ELEMENT publisher (year,title, author+, artist?, isbn)>
<!ELEMENT publisher (year,title, author+, artist?, isbn)>
<!ELEMENT publish-year (#PCDATA)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT author (#PCDATA)>
<!ELEMENT artist (#PCDATA)>
<!ELEMENT isbn (#PCDATA)>
...
```

XML Schema の例

前述の DTD の例に相当する XML Schema 定義は次のとおりです。

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema">
 <element name="publisher">  <complexType>
   <sequence>
    <element name="publish-year" type="short"/>
    <element name="title" type="string"/>
    <element name="author" type="string" maxOccurs="unbounded"/>
    <element name="artist" type="string" nillable="true" minOccurs="0"/>
    <element name="isbn" type="long"/>
   </sequence>
   <attribute name="pubid" type="hexBinary" use="required"/>
  </complexType>
 </element>
</schema>
```

DTD の制限事項

XML マークアップ宣言で知られる DTD は次の処理を含む特定のアプリケーションには不十分であると考えられています。

- 文書の作成および公開
- メタデータの交換
- E-Commerce
- データベース間の操作

DTD には、次のような制限事項があります。

- DTD は名前空間テクノロジと統合されないため、ユーザーはコードをインポートおよび再利用できません。
- DTD は文字データ以外のデータ型をサポートしないため、メタデータ標準およびデータベース・スキーマの記述が制限されます。
- アプリケーションは、DTD が可能にする水準より柔軟に文書構造の制約を指定する必要があります。

XML Schema の機能と DTD の機能の比較

表 B-3 に、XML Schema の機能を示します。XML Schema の機能には DTD の機能が含まれていることに注意してください。

表 B-3 XML Schema の機能と DTD の機能の比較

XML Schema の機能	DTD の機能
組込みデータ型	
XML Schema では、一連の組込みデータ型を指定できます。これらのデータ型は、次の型体系の基礎を構成します。 STRING（文字列）、BOOLEAN（ブール）、FLOAT（浮動）、DECIMAL（小数）、DOUBLE（実数値）、DURATION（継続時間）、DATETIME（日付時刻）、TIME（時）、DATE（日付）、GYEARMONTH（年 / 月）、GYEAR（年）、GMONTHDAY（月 / 日）、GDAY（日）、BASE64BINARY（BASE64 エンコード・バイナリ）、HEXBINARY（16 進バイナリ）、ANYURI（URI 参照）、NOTATION（表記法）、QNAME その他のデータ型は、基本型で定義された導出データ型です。	DTD は、文字列以外のデータ型をサポートしません。
ユーザー定義データ型	
ユーザーは、組込みデータ型から独自のデータ型を導出できます。データ型を導出するには、制限、リストおよび共用体という 3 つの方法があります。制限は、制約ファセットをベース型に適用することによって、より制限されたデータ型を定義します。リストは、単純にその項目型の値のリストを許可します。共用体は新しい型を定義し、その値はメンバー型のいずれかにすることができます。	DTD の例では、publish-year 要素をさらに制約することはできません。

表 B-3 XML Schema の機能と DTD の機能の比較（続き）

XML Schema の機能	DTD の機能
たとえば、publish-year 型の値が特定の範囲内になるように指定するには、次のとおり指定します。 <element name="publish-year"> <simpleType> <restriction base="short" <minInclusive value="1970"/ <maxInclusive value="2000"/> </restriction> </simpleType> </element> 制約ファセットには、length（長さ）、minLength（最小長）、maxLength（最大長）、pattern（パターン）、enumeration（列挙）、whiteSpace（空白）、maxInclusive（最大内含値）、maxExclusive（最大排他値）、minInclusive（最小内含値）、minExclusive（最小排他値）、totalDigits（全桁数）、fractionDigits（小数桁数）があります。 一部のファセットは、特定のベース型にのみ適用されることに注意してください。	--
出現インジケータ（コンテンツ・モデルまたは構造） XML Schema では、インスタンス・ドキュメントまたは要素の構造（complexType）がモデル・グループおよび属性グループで定義されます。属性グループには属性が含まれますが、モデル・グループにはさらにモデル・グループまたは小要素が含まれる場合があります。 モデル・グループと属性グループの両方で、ワイルド・カードを使用できます。モデル・グループには順序、全体および選択という 3 つの種類があり、それぞれ小要素間の順序関係、結合関係および分離関係を表します。また、各小要素の出現回数の範囲も指定できます。	--
データ型と同様に、complexType も他の型から導出できます。導出方法には、制限または拡張があります。導出された型は、ベース型の内容および対応する変更を継承します。継承の他に、型定義は他のコンポーネントを参照することができます。この機能によって、一度定義したコンポーネントを他の様々な構造で使用できます。 XML Schema の型宣言および型定義メカニズムは、DTD より柔軟で強力です。	--

表 B-3 XML Schema の機能と DTD の機能の比較（続き）

XML Schema の機能	DTD の機能
minOccurs、maxOccurs	DTD によって制御される要素内の子要素数は、次の記号で割り当てられます。 ■ ? = 0（ゼロ）または 1。B-31 ページの「DTD の例」では、artist? はアーティストがオプションであることを示しています。 ■ * = 0（ゼロ）以上。 ■ + = 1 以上。B-31 ページの「DTD の例」では、author+ は 1 人以上の作者が存在する可能性を示しています。 ■ (none) = 1。
ID 制約 XML Schema は、一意性、キーおよびキー参照の宣言によって XML ID/IDREF メカニズムの概念を拡張します。これらは型定義の一部であり、属性のみでなく、要素内容もキーとして許可します。各制約には有効範囲があります。字句文字列ではなく、値で制約の比較が行われます。	なし。
インポート / エクスポート・メカニズム（スキーマのインポート、追加および変更） 単一スキーマ・ファイルでは、スキーマのすべてのコンポーネントを定義する必要はありません。XML Schema は、複数の XML Schema を作成するメカニズムを提供します。異なる名前空間を使用する XML Schema を統合する場合はインポート機能を使用し、同じ名前空間を持つコンポーネントを追加する場合は追加機能を使用します。コンポーネントは追加時に再定義して変更できます。	外部スキーマで定義された構造体は使用できません。

XML Schema を使用すると、XML 文書のクラスを定義できます。

XML インスタンス・ドキュメント

XML インスタンス・ドキュメントは、特定の XML Schema に準拠する XML 文書を示します。これらのインスタンスおよび XML Schema は、特にドキュメントとして存在する必要はありませんが、一般にファイルといいます。ただし、これらは次のいずれかとして存在する場合があります。

- バイトのストリーム
- データベース・レコード内のフィールド
- XML Information Set の情報項目のコレクション

Oracle XML DB は、`http://www.w3.org/2001/XMLSchema` にある 2001 年 5 月 2 日に公開された W3C の XML Schema 勧告の仕様をサポートします。

XML Schema への既存の DTD の変換

XMLSpy などの一部の XML エディタを使用すると、XML Schema への既存の DTD の変換が簡単になります。ただし、その場合でも、結果として生成される XML Schema 定義ファイルが有効になり、XML Schema として使用できるようにするには、さらに型指定宣言および検証宣言をそのファイルに追加する必要があります。

XML Schema の例 : PurchaseOrder.xsd

次の PurchaseOrder.xsd の例は、W3C の XML Schema での XML 文書の例に変更を加えずに記載したものです。この XML Schema は、第 3 章「Oracle XML DB の使用」にも例として記載されています。

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
        <xs:complexType name="ActionsType" >
                <xs:sequence>
                        <xs:element name="Action" maxOccurs="4" >
                                <xs:complexType >
                                        <xs:sequence>
                                                <xs:element ref="User"/>
                                                <xs:element ref="Date"/>
                                        </xs:sequence>
                                </xs:complexType>
                        </xs:element>
                </xs:sequence>
        </xs:complexType>
        <xs:complexType name="RejectType" >
                <xs:all>
                        <xs:element ref="User" minOccurs="0"/>
                        <xs:element ref="Date" minOccurs="0"/>
                        <xs:element ref="Comments" minOccurs="0"/>
                </xs:all>
```

```
</xs:complexType>
<xs:complexType name="ShippingInstructionsType" >
        <xs:sequence>
                <xs:element ref="name"/>
                <xs:element ref="address"/>
                <xs:element ref="telephone"/>
        </xs:sequence>
</xs:complexType>
<xs:complexType name="LineItemsType" >
        <xs:sequence>
                <xs:element name="LineItem"
                            type="LineItemType"
                            maxOccurs="unbounded"/>
        </xs:sequence>
</xs:complexType>
<xs:complexType name="LineItemType" >
        <xs:sequence>
                <xs:element ref="Description"/>
                <xs:element ref="Part"/>
        </xs:sequence>
        <xs:attribute name="ItemNumber" type="xs:integer"/>
</xs:complexType>
<!--

-->
<xs:element name="PurchaseOrder" >
        <xs:complexType>
                <xs:sequence>
                        <xs:element ref="Reference"/>
                        <xs:element name="Actions"
                            type="ActionsType"/>
                        <xs:element name="Reject"                I
                            type="RejectType"
                            minOccurs="0"/>
                        <xs:element ref="Requestor"/>
                        <xs:element ref="User"/>
                        <xs:element ref="CostCenter"/>
                        <xs:element name="ShippingInstructions"
                            type="ShippingInstructionsType"/>
                        <xs:element ref="SpecialInstructions"/>
                        <xs:element name="LineItems"
                            type="LineItemsType"/>
                </xs:sequence>
        </xs:complexType>
</xs:element>
<xs:simpleType name="money">
        <xs:restriction base="xs:decimal">
```

```
                <xs:fractionDigits value="2"/>
                <xs:totalDigits value="12"/>
        </xs:restriction>
</xs:simpleType>
<xs:simpleType name="quantity">
        <xs:restriction base="xs:decimal">
                <xs:fractionDigits value="4"/>
                <xs:totalDigits value="8"/>
        </xs:restriction>
</xs:simpleType>
<xs:element name="User"  >
        <xs:simpleType>
                <xs:restriction base="xs:string">
                        <xs:minLength value="1"/>
                        <xs:maxLength value="10"/>
                </xs:restriction>
        </xs:simpleType>
</xs:element>
<xs:element name="Requestor"  >
        <xs:simpleType>
                <xs:restriction base="xs:string">
                        <xs:minLength value="0"/>
                        <xs:maxLength value="128"/>
                </xs:restriction>
        </xs:simpleType>
</xs:element>
<xs:element name="Reference"  >
        <xs:simpleType>
                <xs:restriction base="xs:string">
                        <xs:minLength value="1"/>
                        <xs:maxLength value="26"/>
                </xs:restriction>
        </xs:simpleType>
</xs:element>
<xs:element name="CostCenter"  >
        <xs:simpleType>
                <xs:restriction base="xs:string">
                        <xs:minLength value="1"/>
                        <xs:maxLength value="4"/>
                </xs:restriction>
        </xs:simpleType>
</xs:element>
<xs:element name="Vendor"  >
        <xs:simpleType>
                <xs:restriction base="xs:string">
                        <xs:minLength value="0"/>
                        <xs:maxLength value="20"/>
```

```
                </xs:restriction>
        </xs:simpleType>
</xs:element>
<xs:element name="PONumber"  >
        <xs:simpleType>
                <xs:restriction base="xs:integer"/>
        </xs:simpleType>
</xs:element>
<xs:element name="SpecialInstructions"  >
        <xs:simpleType>
                <xs:restriction base="xs:string">
                        <xs:minLength value="0"/>
                        <xs:maxLength value="2048"/>
                </xs:restriction>
        </xs:simpleType>
</xs:element>
<xs:element name="name"  >
        <xs:simpleType>
                <xs:restriction base="xs:string">
                        <xs:minLength value="1"/>
                        <xs:maxLength value="20"/>
                </xs:restriction>
        </xs:simpleType>
</xs:element>
<xs:element name="address"  >
        <xs:simpleType>
                <xs:restriction base="xs:string">
                        <xs:minLength value="1"/>
                        <xs:maxLength value="256"/>
                </xs:restriction>
        </xs:simpleType>
</xs:element>
<xs:element name="telephone"  >
        <xs:simpleType>
                <xs:restriction base="xs:string">
                        <xs:minLength value="1"/>
                        <xs:maxLength value="24"/>
                </xs:restriction>
        </xs:simpleType>
</xs:element>
<xs:element name="Date" type="xs:date"  />
<xs:element name="Comments"  >
        <xs:simpleType>
                <xs:restriction base="xs:string">
                        <xs:minLength value="1"/>
                        <xs:maxLength value="2048"/>
                </xs:restriction>
```

```
                </xs:simpleType>
        </xs:element>
        <xs:element name="Description"  >
                <xs:simpleType>
                        <xs:restriction base="xs:string">
                                <xs:minLength value="1"/>
                                <xs:maxLength value="256"/>
                        </xs:restriction>
                </xs:simpleType>
        </xs:element>
        <xs:element name="Part"  >
                <xs:complexType>
                        <xs:attribute name="Id" >
                                <xs:simpleType>
                                        <xs:restriction base="xs:string">
                                            <xs:minLength value="12"/>
                                            <xs:maxLength value="14"/>
                                        </xs:restriction>
                                </xs:simpleType>
                        </xs:attribute>
                        <xs:attribute name="Quantity" type="money"/>
                        <xs:attribute name="UnitPrice" type="quantity"/>
                </xs:complexType>
        </xs:element>
</xs:schema>
```

C

XPath および Namespace の手引き

この付録では、W3C の XPath 勧告、Namespace 勧告および Information Set（情報セット）の概要を説明します。この付録の内容は次のとおりです。

- W3C の XPath 1.0 勧告の概要
- XPath 式
- ロケーション・パス
- XPath 1.0 のデータ・モデル
- W3C の XPath 2.0 草案の概要
- W3C の XML Namespace 勧告の概要
- W3C の XML Information Set の概要

W3C の XPath 1.0 勧告の概要

XPath は、XML 文書の一部を指定するための言語であり、XSLT と XPointer の両方で使用されるように設計されています。XPath は、検索言語または問合せ言語として使用できるのみでなく、ハイパー・テキスト・リンクでも使用できます。この付録で示す XPath の手引きは、W3C の XPath 勧告から抜粋したものです。

XPath は、文字列、数値およびブールの操作も簡単にします。

XPath は、簡潔な非 XML 構文を使用して、URI および XML 属性値内で XPath を簡単に使用できるようにします。XPath は、XML 文書の表面的な構文ではなく、その抽象的な論理構造で機能します。XPath の名前は、URL の場合と同様に、XML 文書の階層構造をナビゲートするためにパス表記法を使用することから付けられています。

XML 文書を指定するための使用に加えて、XPath は、照合（あるノードがあるパターンに一致しているかどうかのテスト）に使用できる一般的なサブセットを持つようにも設計されています。このような XPath の使用については、W3C の XSLT 勧告を参照してください。

注意： 今回のリリースでは、Oracle XML DB は、XPath 1.0 勧告のサブセットをサポートします。Oracle XML DB は、ブール、数値または文字列を戻す XPath をサポートしません。ただし、Oracle XML DB は、述語内でこれらの XPath 型をサポートします。

XPath による XML 文書のノードのツリーとしてのモデル化

XPath は、XML 文書をノードのツリーとしてモデル化します。ノードには、要素ノード、属性ノード、テキスト・ノードなどの様々な型があります。XPath は、それぞれの型のノードについて、その文字列値を計算する方法を定義します。ノードの型には、名前を持つものもあります。XPath は、XML 名前空間を完全にサポートします。そのため、ノードの名前は、1 つのローカル部分および NULL の可能性がある 1 つの名前空間 URI で構成されるペアとしてモデル化されます。これを拡張名といいます。データ・モデルの詳細は、C-10 ページの「XPath 1.0 のデータ・モデル」を参照してください。XML 名前空間の概要については、C-17 ページの「W3C の XML Namespace 勧告の概要」を参照してください。

参照： 次の URL を参照してください。

- `http://www.w3.org/TR/xpath`
- `http://www.w3.org/TR/xpath20/`
- `http://www.zvon.org/xxl/XPathTutorial/General/examples.html`
- `http://www.mulberrytech.com/quickref/XSLTquickref.pdf`
- `http://www.oreilly.com/catalog/xmlnut/chapter/ch09.html`
- XML での Unicode の使用については、`http://www.w3.org/TR/2002/NOTE-unicode-xml-20020218/` を参照してください。

XPath 式

XPath の構文は、主に式で構成されます。式は、生成規則 `Expr` と一致します。式は評価されてオブジェクトを生成しますが、このオブジェクトは次の 4 つの基本型の 1 つを持ちます。

- ノード・セット（重複していないノードの順序付けられていないコレクション）
- ブール（TRUE または FALSE）
- 数値（浮動小数点数）
- 文字列（UCS 文字列）

コンテキストに関連した式の評価

式の評価は、コンテキストに関連して行われます。XSLT および XPointer は、それぞれ XSLT および XPointer で使用される XPath 式に対するコンテキストを決定する方法を指定します。コンテキストは次のもので構成されます。

- **ノード**（コンテキスト・ノード）。
- **0（ゼロ）以外の正の整数の組**（コンテキスト位置およびコンテキスト・サイズ）。コンテキスト位置は、常にコンテキスト・サイズ以下です。
- **変数バインディングのセット**。これらのバインディングは、変数名から変数値へのマッピングで構成されます。変数の値はオブジェクトです。このオブジェクトは、式の値に使用可能などの型であることもでき、ここでは指定されていない追加の型であることもできます。
- **関数ライブラリ**。このライブラリは、関数名から関数へのマッピングで構成されます。各関数は、0（ゼロ）個以上の引数を取り、単一の結果を戻します。すべての XPath 実装がサポートする必要があるコア関数ライブラリの定義については、XPath 勧告を参照

してください。コア関数ライブラリの関数の場合、変数および結果は次の 4 つの基本型になります。

- ノード・セット関数
- 文字列関数
- ブール関数
- 数値関数

XSLT と XPointer は、両方とも追加の関数を定義することによって XPath を拡張します。これらの関数には、4 つの基本型に機能するものがあります。また、XSLT や XPointer によって定義された追加のデータ型に機能するものもあります。

- **式の有効範囲内にある名前空間宣言のセット**。これらの名前空間宣言は、接頭辞から名前空間 URI へのマッピングで構成されます。

部分式の評価

部分式を評価するために使用される変数バインディング、関数ライブラリおよび名前空間宣言は、その部分式を含む式を評価するために使用されるものと常に同じです。

部分式を評価するために使用されるコンテキスト・ノード、コンテキスト位置およびコンテキスト・サイズは、その部分式を含む式を評価するために使用されるものと異なる場合があります。いくつかの種類の式は、コンテキスト・ノードを変更します。コンテキスト位置およびコンテキスト・サイズを変更するのは、述語のみです。ある種類の式の評価を記述する場合、部分式の評価に対してコンテキスト・ノード、コンテキスト位置およびコンテキスト・サイズが変更されるときは、常に明示的に示します。コンテキスト・ノード、コンテキスト位置およびコンテキスト・サイズについて記述しない場合、それらは、その種類の式の部分式の評価に対して変更されないままです。

XML 属性内に頻繁に出現する XPath 式

この項に指定されている文法は、XML 1.0 正規化の後の属性値に適用されます。そのため、たとえば、文法で < という文字が使用される場合、この文字は XML ソース内では < として出現できませんが、XML 1.0 の規則に従って、`<` などとして入力することによって、引用される必要があります。

式内では、リテラル文字列は一重引用符または二重引用符によって区切られます。これらの引用符は、XML 属性の区切るためにも使用されます。XML プロセッサによって式内の引用符が属性値の終了として解析されることを回避するため、次の機能があります。

- 引用符は、文字参照（`&quot;` または `&apos;`）として入力できます。
- XML 属性が二重引用符で区切られている場合、式は一重引用符を使用でき、XML 属性が一重引用符で区切られている場合、式は二重引用符を使用できます。

ロケーション・パス

式の種類のうち重要なものの 1 つは、ロケーション・パスです。ロケーション・パスとは、取る必要があるルートです。ルートは、方向およびいくつかのステップで構成される場合があります。それぞれのステップは、「/」で区切られています。

ロケーション・パスは、コンテキスト・ノードに対する相対的なノードの集合を選択します。ロケーション・パスである式を評価した結果は、そのロケーション・パスによって選択されたノードを含むノードセットです。

ロケーション・パスは、ノードの集合をフィルタするために使用される式を再帰的に含むことができます。ロケーション・パスは、生成規則 `LocationPath` と一致します。

式の解析は、まず解析する文字列をトークンに分割し、結果の一連のトークンを解析することによって行われます。空白は、トークン間で自由に使用できます。

ロケーション・パスは XPath 内で最も一般的な文法構成メンバーではありません（LocationPath は Expr の特殊なケースです）が、最も重要な構成メンバーです。

ロケーション・パスの構文の省略形

どのロケーション・パスも、単純で冗長的な構文を使用して表現できます。また、一般的なケースを簡潔に表現できる多くの構文の省略形もあります。この項の内容は次のとおりです。

- C-5 ページの「非省略構文を使用したロケーション・パスの例」では、非省略構文を使用したロケーション・パスのセマンティクスについて説明します。
- C-7 ページの「省略構文を使用したロケーション・パスの例」では、省略構文について説明します。

非省略構文を使用したロケーション・パスの例

表 C-1 に、非省略構文を使用したロケーション・パスの例を示します。

表 C-1 XPath: 非省略構文を使用したロケーション・パスの例

非省略ロケーション・パス	説明
child::para	コンテキスト・ノードの para 子要素を選択します。
child::*	コンテキスト・ノードのすべての子要素を選択します。
child::text()	コンテキスト・ノードのすべての子テキスト・ノードを選択します。
child::node()	ノードの型に関係なく、コンテキスト・ノードのすべての子を選択します。
attribute::name	コンテキスト・ノードの name 属性を選択します。

表 C-1 XPath: 非省略構文を使用したロケーション・パスの例（続き）

非省略ロケーション・パス	説明
attribute::*	コンテキスト・ノードのすべての属性を選択します。
nodedescendant::para	コンテキスト・ノードの para 子孫要素を選択します。
ancestor::div	コンテキスト・ノードのすべての div 祖先要素を選択します。
ancestor-or-self::div	コンテキスト・ノードの div 祖先要素を選択し、コンテキスト・ノードが div 要素である場合は、そのコンテキスト・ノードも選択します。
descendant-or-self::para	コンテキスト・ノードの para 子孫要素を選択し、コンテキスト・ノードが para 要素である場合は、そのコンテキスト・ノードも選択します。
self::para	コンテキスト・ノードが para 要素である場合は、そのコンテキスト・ノードを選択します。それ以外の場合は、何も選択しません。
child::chapter/descendant::para	コンテキスト・ノードの chapter 子要素の para 子孫要素を選択します。
child::*/child::para	コンテキスト・ノードのすべての para 孫要素を選択します。
/	常に文書要素の親であるドキュメント・ルートを選択します。
/descendant::para	コンテキスト・ノードと同じドキュメント内に存在するすべての para 要素を選択します。
/descendant::olist/child::item	olist 親要素を持ち、コンテキスト・ノードと同じドキュメント内に存在するすべての item 要素を選択します。
child::para[position()=1]	コンテキスト・ノードの最初の para 子要素を選択します。
child::para[position()=last()]	コンテキスト・ノードの最後の para 子要素を選択します。
child::para[position()=last()-1]	コンテキスト・ノードの最後から 2 番目の para 子要素を選択します。
child::para[position()>1]	コンテキスト・ノードの最初の para 子要素以外の、コンテキスト・ノードのすべての para 子要素を選択します。
following-sibling::chapter[position()=1]	コンテキスト・ノードの次の chapter 兄弟要素を選択します。
preceding-sibling::chapter[position()=1]	コンテキスト・ノードの前の chapter 兄弟要素を選択します。
/descendant::figure[position()=42]	ドキュメントの 42 番目の figure 要素を選択します。
/child::doc/child::chapter[position()=5]/child::section [position()=2]	doc 文書要素の 5 番目の chapter の 2 番目の section を選択します。

表 C-1 XPath: 非省略構文を使用したロケーション・パスの例（続き）

非省略ロケーション・パス	説明
child::para[attribute::type="warning"]	コンテキスト・ノードの、warning という値を持つ type 属性を含むすべての para 子要素を選択できます。
child::para[attribute::type='warning'][position()=5]	コンテキスト・ノードの、warning という値を持つ type 属性を含む 5 番目の para 子要素を選択できます。
child::para[position()=5][attribute::type= "warning"]	コンテキスト・ノードの 5 番目の para 子要素が warning という値を持つ type 属性を含む場合、その para 子要素を選択します。
child::chapter[child::title='Introduction']	コンテキスト・ノードの、Introduction と等しい文字列値を持つ 1 つ以上の title 子要素を含む chapter 子要素を選択します。
child::chapter[child::title]	コンテキスト・ノードの、1 つ以上の title 子要素を含む chapter 子要素を選択します。
child::*[self::chapter or self::appendix]	コンテキスト・ノードの chapter 子要素および appendix 子要素を選択します。
child::*[self::chapter or self::appendix][position()=last()]	コンテキスト・ノードの最後の chapter 子要素または appendix 子要素を選択します。

省略構文を使用したロケーション・パスの例

表 C-2 に、省略構文を使用したロケーション・パスの例を示します。

表 C-2 XPath: 省略構文を使用したロケーション・パスの例

省略ロケーション・パス	説明
para	コンテキスト・ノードの para 子要素を選択します。
*	コンテキスト・ノードのすべての子要素を選択します。
text()	コンテキスト・ノードのすべての子テキスト・ノードを選択します。
@name	コンテキスト・ノードの name 属性を選択します。
@*	コンテキスト・ノードのすべての属性を選択します。
para[1]	コンテキスト・ノードの最初の para 子要素を選択します。
para[last()]	コンテキスト・ノードの最後の para 子要素を選択します。
*/para	コンテキスト・ノードのすべての para 孫要素を選択します。
/doc/chapter[5]/section[2]	doc の 5 番目の chapter の 2 番目の section を選択します。

表 C-2 XPath: 省略構文を使用したロケーション・パスの例（続き）

省略ロケーション・パス	説明
chapter//para	コンテキスト・ノードの chapter 子要素の para 子孫要素を選択します。
//para	ドキュメント・ルートのすべての para 子孫要素を選択します。したがって、コンテキスト・ノードと同じドキュメント内に存在するすべての para 要素を選択します。
//olist/item	コンテキスト・ノードと同じドキュメント内に存在する、olist 親要素を持つすべての item 要素を選択します。
.	コンテキスト・ノードを選択します。
.//para	コンテキスト・ノードの para 子孫要素を選択します。
..	コンテキスト・ノードの親を選択します。
../@lang	コンテキスト・ノードの親の lang 属性を選択します。
para[@type="warning"]	コンテキスト・ノードの、warning という値を持つ type 属性を含むすべての para 子要素を選択できます。
para[@type="warning"][5]	コンテキスト・ノードの、warning という値を持つ type 属性を含む 5 番目の para 子要素を選択できます。
para[5][@type="warning"]	コンテキスト・ノードの 5 番目の para 子要素が warning という値を持つ type 属性を含む場合、その para 子要素を選択します。
chapter[title="Introduction"]	コンテキスト・ノードの、Introduction と等しい文字列値を持つ 1 つ以上の title 子要素を含む chapter 子要素を選択します。
chapter[title]	コンテキスト・ノードの、1 つ以上の title 子要素を含む chapter 子要素を選択します。
employee[@secretary and @assistant]	コンテキスト・ノードの、secretary 属性と assistant 属性の両方を持つ employee 子要素を選択します。

最も重要な省略は、`child::` をロケーション・ステップから省略できるということです。実際には、child はデフォルトの軸です。たとえば、`div/para` というロケーション・パスは `child::div/child::para` の短縮形です。

属性の省略形 @

属性にも省略形があります。`attribute::` は **@** に省略できます。

たとえば、`para[@type="warning"]` というロケーション・パスは、`child::para[attribute::type="warning"]` の短縮形であるため、warning と等しい値を持つ type 属性を含む para 子要素を選択します。

パスの省略形 //

// は、/descendant-or-self::node()/ の短縮形です。たとえば、//para は、/descendant-or-self::node()/child::para の短縮形であるため、ドキュメント内の任意の para 要素を選択します（文書要素ノードはルート・ノードの子であるため、文書要素である para 要素でも //para によって選択されます）。

div//para は、div/descendant-or-self::node()/child::para の短縮形であるため、div 子要素のすべての para 子孫要素を選択します。

注意： //para[1] というロケーション・パスは、/descendant::para[1] というロケーション・パスと同じ意味ではありません。/descendant::para[1] は、最初の para 子孫要素を選択します。//para[1] は、その親の最初の para 子要素であるすべての para 子孫要素を選択します。

ロケーション・ステップの省略形 .

. というロケーション・ステップは、self::node() の短縮形です。これは、特に // とともに使用すると効果的です。たとえば、.//para というロケーション・パスは、次のロケーション・パスの短縮形です。

```
self::node()/descendant-or-self::node()/child::para
```

したがって、この短縮形は、コンテキスト・ノードのすべての para 子孫要素を選択します。

ロケーション・ステップの省略形 ..

同様に、.. というロケーション・ステップは、parent::node() の短縮形です。たとえば、../title は次のロケーション・パスの短縮形です。

```
parent::node()/child::title
```

したがって、この短縮形は、コンテキスト・ノードの親の title 子要素を選択します。

省略形の概要

省略形の絶対ロケーション・パス ::= '//' 相対ロケーション・パス

省略形の相対ロケーション・パス ::= 相対ロケーション・パス '//' ステップ

省略形のステップ ::= '.' | '..'

省略形の軸指定子 ::= '@'?

相対ロケーション・パスおよび絶対ロケーション・パス

ロケーション・パスには 2 つの種類があります。

- **相対ロケーション・パス :** 相対ロケーション・パスは、/ で区切られた 1 つ以上のロケーション・ステップのシーケンスで構成されます。相対ロケーション・パス内のステップは、左から右に合成されます。次に、各ステップは、コンテキスト・ノードに対して相対的にノードの集合を選択します。ステップのシーケンスの先頭部分は、次のとおり、以降のステップと合成されます。ステップのシーケンスの先頭部分は、コンテキスト・ノードに対して相対的にノードの集合を選択します。その集合内の各ノードは、次のステップのコンテキスト・ノードとして使用されます。そのステップによって識別されたノードの集合が 1 つに統合されます。ステップを合成したものによって識別されたノードの集合は、この共用体です。

  たとえば、`child::div/child::para` は、コンテキスト・ノードの div 子要素の para 子要素（div 親要素を持つ para 孫要素）を選択します。

- **絶対ロケーション・パス :** 絶対ロケーション・パスは、/ と、その後にオプションで続く相対ロケーション・パスで構成されます。/ 自体は、コンテキスト・ノードを含むドキュメントのルート・ノードを選択します。その後に相対ロケーション・パスが続いている場合、そのロケーション・パスは、コンテキスト・ノードを含むドキュメントのルート・ノードに対して相対的に相対ロケーション・パスによって選択されるノードの集合を選択します。

ロケーション・パスの構文の概要

ロケーション・パスは、ターゲット・ノードを検索する方法を提供します。次に、ロケーション・パスの一般的な構文を示します。

```
axisname :: nodetest expr1 expr2 ...

LocationPath            ::=   RelativeLocationPath
                            | AbsoluteLocationPath
AbsoluteLocationPath    ::=   '/' RelativeLocationPath?
                            | AbbreviatedAbsoluteLocationPath
RelativeLocationPath    ::=   Step
                            | RelativeLocationPath '/' Step
                            | AbbreviatedRelativeLocationPath
```

XPath 1.0 のデータ・モデル

XPath は、ツリーとしての XML 文書に機能します。この項では、XPath が XML 文書をツリーとしてモデル化する方法について説明します。XPath によって操作される XML 文書に対するこのモデルの関連性は、XML Namespace 勧告に準拠している必要があります。

> **参照 :** C-17 ページの「W3C の XML Namespace 勧告の概要」を参照してください。

ノード

ツリーにはノードが含まれます。ノードには 7 つのタイプがあります。

- ルート・ノード
- 要素ノード
- テキスト・ノード
- 属性ノード
- 名前空間ノード
- 処理命令ノード
- コメント・ノード

ルート・ノード

ルート・ノードは、ツリーのルートです。ルート・ノードは、ツリーのルートとして以外は出現しません。文書要素の要素ノードは、ルート・ノードの子です。ルート・ノードは、プロローグ内や、文書要素の末尾の後に出現する処理命令（PI）およびコメントを処理するための、子処理命令ノードおよび子コメント・ノードも持ちます。ルート・ノードの文字列は、ルート・ノードのすべての子孫テキスト・ノードの文字列値をドキュメント順に連結したものです。ルート・ノードは拡張名を持ちません。

要素ノード

ドキュメント内の要素ごとに 1 つの要素ノードが存在します。要素ノードは、XML Namespace 勧告に従って、タグ内に指定されている要素の `QName` を拡張することによって計算された拡張名を持ちます。要素の拡張名の名前空間 URI は、`QName` に接頭辞がなく、適用可能なデフォルトの名前空間がない場合、NULL になります。

注意： `http://www.w3.org/TR/REC-xml-names/` の付録 A.3 の表記法では、拡張名のローカル部分は ExpEType 要素の type 属性に対応します。拡張名の名前空間 URI は、ExpEType 要素の ns 属性に対応し、ExpEType 要素の ns 属性が省略されている場合は NULL です。

要素ノードの子は、要素ノード、コメント・ノード、処理命令ノード、およびその内容に対するテキスト・ノードです。内部エンティティと外部エンティティの両方に対する実体参照は、拡張されます。文字参照は解決されます。要素ノードの文字列は、要素ノードのすべての子孫テキスト・ノードの文字列値をドキュメント順に連結したものです。

一意の ID: 要素ノードは、一意の識別子（ID）を持つことができます。これは、DTD で ID 型として宣言されている属性の値です。1 つのドキュメント内で 2 つの要素が同じ一意の ID を持つことはできません。XML プロセッサが 1 つのドキュメント内の 2 つの要素が同じ一意の ID を持っていること（これは、ドキュメントが妥当でない場合にのみ可能です）を通

知した場合、そのドキュメント順の 2 番目の要素は一意の ID を持っていないものとして処理される必要があります。

注意： ドキュメントが DTD を持たない場合、そのドキュメント内のどの要素も一意の ID を持ちません。

テキスト・ノード

文字データは、テキスト・ノードにグループ化されます。各テキスト・ノードには、可能なかぎり多くの文字データがグループ化されます。テキスト・ノードが、テキスト・ノードである直前または直後の兄弟を持つことはありません。テキスト・ノードの文字列値は、文字データです。テキスト・ノードは、常に 1 文字以上のデータを持ちます。CDATA セクション内の各文字は、文字データとして処理されます。したがって、ソース・ドキュメント内の `<![CDATA[<]]>` は、`<` と同様に処理されます。どちらの結果も、ツリーのテキスト・ノードでは単一の < 文字になります。そのため、CDATA セクションは、`<![CDATA[` および `]]>` が削除され、< および & が出現するごとにそれぞれ `<` および `&` に置き換えられたときと同様に処理されます。

注意： < 文字を含むテキスト・ノードが XML として書き出される場合、< 文字は &lt を使用したり、それを CDATA セクション内に組み込むことによってエスケープする必要があります。コメント、処理命令および属性値内の文字は、テキスト・ノードを生成しません。外部エンティティ内の行端は、XML 勧告で指定されているとおり、#xA に正規化されます。テキスト・ノードは拡張名を持ちません。

属性ノード

各要素ノードは、関連付けられた属性ノードの集合を持ちます。要素は、これらの各属性ノードの親です。ただし、属性ノードはその親要素の子ではありません。

注意： これは DOM とは異なります。DOM は、属性を持つ要素をその属性の親として処理しません。

要素が属性ノードを共有することはありません。1 つの要素ノードが別の要素ノードと同じノードではない場合、片方の要素ノードのすべての属性ノードはもう 1 つの要素ノードの属性ノードと同じノードではありません。

注意： = 演算子は、2 つのノードが同じノードであるかどうかではなく、それらが同じ値を持つかどうかをテストします。したがって、2 つの異なる要素の属性は、同じノードでない場合でも、= を使用して等しいものとして比較される場合があります。

デフォルトの属性は、指定された属性と同様に処理されます。DTD で要素型に対して属性を宣言した場合でも、デフォルトを #IMPLIED として宣言し、その要素に対して属性を指定しなかった場合、その要素の属性セットにその属性のノードは含まれません。

`xml:lang` や `xml:space` などの一部の属性は、別の子孫要素に対する同じ属性のインスタンスによってオーバーライドされないかぎり、その属性を持つ要素の子孫であるすべての要素に適用されます。ただし、これはツリー内での属性ノードの出現位置に影響しません。要素は、その要素の開始タグまたは空要素タグで明示的に指定された属性、または DTD でデフォルト値を使用して明示的に宣言された属性に対してのみ、属性ノードを持ちます。

属性ノードは、拡張名および文字列値を持ちます。拡張名は、XML Namespace 勧告に従って、XML 文書のタグ内に指定されている `QName` を拡張することによって計算されます。属性の QName に接頭辞がない場合、属性名の名前空間 URI は NULL になります。

注意： XML Namespace 勧告の付録 A.3 の表記法では、拡張名のローカル部分は `ExpAName` 要素の name 属性に対応します。拡張名の名前空間 URI は、`ExpAName` 要素の ns 属性に対応し、`ExpAName` 要素の ns 属性が省略されている場合は NULL です。

属性ノードは文字列値を持ちます。この文字列値は、XML 勧告によって指定されているとおり、正規化された値です。正規化された値が長さ 0（ゼロ）の文字列である属性は、特別に処理されることはありません。この属性は、結果として文字列値に長さ 0（ゼロ）の文字列を持つ属性ノードになります。

注意： デフォルトの属性は、外部 DTD または外部パラメータ・エンティティで宣言できます。XML 勧告では、検証を行わない XML プロセッサは、外部 DTD または外部パラメータを読み込む必要はありません。一部の非検証 XML プロセッサでは、XPath ツリーに外部 DTD または外部パラメータ・エンティティで宣言されたデフォルトの属性値が含まれることを前提とするスタイルシートまたは他の機能が機能しない場合があります。

名前空間を宣言する属性に対応する属性ノードはありません。

名前空間ノード

各要素は、関連付けられた名前空間ノードの集合を持ちます。名前空間ノードは、要素の有効範囲内にある個別の名前空間接頭辞（XML Namespace 勧告によって暗黙的に宣言されている xml 接頭辞など）ごとに 1 つ、およびデフォルトの名前空間が要素の有効範囲内にある場合は、それに対して 1 つです。要素は、これらの各名前空間ノードの親です。ただし、名前空間ノードは、その親要素の子ではありません。

要素が名前空間ノードを共有することはありません。1 つの要素ノードが別の要素ノードと同じノードではない場合、片方の要素ノードのすべての名前空間ノードはもう 1 つの要素ノードの名前空間ノードと同じノードではありません。これは、要素が次のものに対して名前空間ノードを持つことを意味します。

- 名前が xmlns: で始まる要素の各属性
- 要素自体またはより近い祖先がその接頭辞を再宣言していないかぎり、名前が xmlns: で始まる祖先要素の各属性
- 要素または祖先が xmlns 属性を持ち、最も近いそのような要素の xmlns 属性の値が空でない場合、xmlns 属性

> **注意：** 属性 xmlns="" は、デフォルトの名前空間の宣言を取り消します。

名前空間ノードは拡張名を持ちます。ローカル部分は、名前空間接頭辞です（これは、名前空間ノードがデフォルトの名前空間用である場合は空です）。名前空間 URI は、常に NULL です。

名前空間ノードの文字列値は、名前空間接頭辞にバインドされている名前空間 URI です。その名前空間 URI が相対 URI である場合、拡張名の名前空間 URI と同様に解決される必要があります。

処理命令ノード

ドキュメント・タイプ宣言内に出現する処理命令を除き、処理命令ごとに処理命令ノードが存在します。処理命令は拡張名を持ちます。ローカル部分は、処理命令のターゲットです。名前空間 URI は NULL です。処理命令ノードの文字列値は、処理命令のうちのターゲットと空白に続く部分です。最後の ?> は、この文字列値に含まれません。

> **注意：** XML 宣言は、処理命令ではありません。したがって、XML 宣言に対応する処理命令ノードはありません。

コメント・ノード

ドキュメント・タイプ宣言内に出現するコメントを除き、コメントごとにコメント・ノードがあります。コメントの文字列値は、開始の <!-- または終了の --> を含まない、コメントの内容です。コメント・ノードは拡張名を持ちません。

ノードの文字列値を決定する方法は、ノードのタイプによって異なります。ノードのタイプによって、文字列値がそのノードの一部となるものもあり、文字列値が子孫ノードの文字列値から計算されるものもあります。

注意： 要素ノードおよびルート・ノードの場合、ノードの文字列値は、DOM nodeValue メソッドによって戻される文字列と同じではありません。

拡張名

ノードのタイプによっては、拡張名を持つものもあります。拡張名は、次のもので構成されます。

- ローカル部分。これは文字列です。
- 名前空間 URI。名前空間 URI は、NULL または文字列のいずれかです。XML 文書で指定された名前空間 URI は、RFC2396 で定義されているとおり、URI 参照として使用できます。これは、名前空間 URI がフラグメント識別子を持ち、相対 URI として使用できることを意味します。相対 URI は、名前空間の処理中に絶対 URI に解決される必要があります。データ・モデル内のノードが持つ拡張名の名前空間 URI は、絶対 URI である必要があります。

2 つの拡張名が同じローカル部分を持ち、どちらも NULL の名前空間 URI を持つか、または非 NULL の等しい名前空間 URI を持つ場合、その 2 つの拡張名は等しくなります。

ドキュメント順

ドキュメント順という順序付けがあります。これは、ドキュメント内のすべてのノードで定義され、汎用エンティティの拡張後にドキュメントの XML 表現内で、各ノードの XML 表現の最初の文字が出現する順序に対応しています。したがって、ルート・ノードが最初のノードになります。

要素ノードは、その子より前に出現します。したがって、ドキュメント順では、（エンティティの拡張後に）XML 内で要素ノードの開始タグが出現した順に要素ノードが順序付けされます。要素の属性ノードおよび名前空間ノードは、その要素の子より前に出現します。名前空間ノードは、属性ノードより前に出現するように定義されます。

名前空間ノードの相対的な順序は、実装に依存します。

属性ノードの相対的な順序は、実装に依存します。

逆ドキュメント順は、ドキュメント順を逆にしたものです。

ルート・ノードおよび要素ノードは、子ノードの順序付けられたリストを持ちます。ノードが子を共有することはありません。1 つのノードが別のノードと同じノードではない場合、片方のノードのすべての子はもう 1 つのノードのどの子とも同じノードではありません。

ルート・ノード以外のすべてのノードは、親を 1 つのみ持ちます。その親は、要素ノードまたはルート・ノードのどちらかです。ルート・ノードまたは要素ノードは、その各子ノードの親です。ノードの子孫とは、そのノードの子、およびそのノードの子の子孫です。

W3C の XPath 2.0 草案の概要

XPath 2.0 は、W3C の XSL ワーキング・グループと XML Query ワーキング・グループによる共同作業の成果です。XPath 2.0 は、XPath 1.0 と XQuery の両方から導出された言語です。XPath 2.0 草案および XQuery 1.0 草案は、共通のソースから作成されています。これらの言語は、密接に関係しており、同じ式構文およびセマンティクスの大部分を共有しています。この 2 つの草案には、同一の箇所があります。

XPath は、XSLT や XQuery などのホスト言語に埋め込まれるように設計されています。XPath は、照合（あるノードがあるパターンに一致しているかどうかのテスト）に使用できる一般的なサブセットを持ちます。

XQuery バージョン 1.0 には、サブセットとして XPath バージョン 2.0 が含まれています。構文的に有効で、XPath 2.0 と XQuery 1.0 の両方で正常に実行されるすべての式は、どちらの言語でも同じ結果を戻します。

XPath は、次の仕様にも依存し、それらと密接に関係しています。

- XPath データ・モデルは、XPath プロセッサが使用できる XML 文書内の情報を定義します。データ・モデルは、XQuery 1.0 and XPath 2.0 Data Model で定義されています。
- XPath の静的セマンティクスおよび動的セマンティクスは、XQuery 1.0 Formal Semantics で正式に定義されています。これは、完全 XPath 言語をセマンティクスの定義対象であるコア・サブセットにマッピングすることによって行われます。このドキュメントは、XPath の正確な定義を必要とする実装者などに有効です。
- XPath がサポートする関数および演算子のライブラリは、XQuery 1.0 and XPath 2.0 Functions and Operators で定義されています。

XPath 2.0 の式

XPath の基本ビルディング・ブロックは式です。この言語は、いくつかの種類の式を提供します。これらの式は、キーワード、記号およびオペランドで構成される場合があります。通常、式のオペランドは他の式です。

XPath は、様々な種類の式を完全な一般性を伴ってネストできるようにする機能的な言語です。また、XPath は強い型指定の言語でもあります。この言語では、様々な式、演算子および関数のオペランドが指定された型に準拠している必要があります。

XML と同様に、XPath は大 / 小文字を区別する言語です。XPath 内のすべてのキーワードには、小文字が使用されます。

```
Expr
                    ::=
                       OrExpr
                       |  AndExpr
                       |  ForExpr
                       |  QuantifiedExpr
                       |  IfExpr
                       |  GeneralComp
                       |  ValueComp
                       |  NodeComp
                       |  OrderComp
                       |  InstanceofExpr
                       |  RangeExpr
                       |  AdditiveExpr
                       |  MultiplicativeExpr
                       |  UnionExpr
                       |  IntersectExceptExpr
                       |  UnaryExpr
                       |  CastExpr
                       |  PathExpr
```

W3C の XML Namespace 勧告の概要

他のソフトウェア・パッケージ用のマークアップで同じ要素型または属性名が使用されている場合に名前の「衝突」が発生したときでも、ソフトウェア・モジュールは、それが、処理対象として設計されているタグおよび属性を認識できる必要があります。

文書の構成メンバーは、この汎用名を持つ必要があります。汎用名の有効範囲は、それらの構成メンバーを含む文書を含み、それ以外のものにも適用されます。これを実現する XML 名前空間というメカニズムについては、W3C の XML Namespace 勧告を参照してください。

参照： http://www.w3.org/TR/REC-xml-names/ を参照してください。

名前空間の概要

XML 名前空間とは、URI 参照（RFC2396）によって識別され、XML 文書内で要素型および属性名として使用される名前の集合です。XML 名前空間は、内部構造を持ち、数学的には集合ではないという点で、コンピューティング分野で一般的に使用される「名前空間」とは異なります。この問題については、W3C の Namespace 勧告の付録「A. The Internal Structure of XML Namespaces（XML 名前空間の内部構造）」を参照してください。

URI 参照

名前空間を識別する URI 参照は、文字ごとに完全に同じである場合、同一とみなされます。この意味で同一でない URI 参照も、実際には機能的に同等である場合があることに注意してください。この例には、大 / 小文字のみが異なる URI 参照、または異なる有効なベース URI を持つ外部エンティティ内の URI 参照があります。

XML 名前空間からの名前は、修飾名として出現できます。この修飾名には、その名前を名前空間接頭辞とローカル部分に分離する単一のコロンが含まれます。

URI 参照にマップされた接頭辞が、名前空間を選択します。普遍的に管理される URI 名前空間とドキュメント独自の名前空間の組合せによって、普遍的に一意の識別子が生成されます。また、接頭辞の有効範囲決定およびデフォルト設定のためのメカニズムが提供されます。

URI 参照は、名前には使用できない文字を含むことができるため、名前空間接頭辞として直接使用することはできません。そのため、名前空間接頭辞は URI 参照のプロキシとして機能します。名前空間接頭辞と URI 参照の関連付けを宣言するために、次の項に示す属性ベースの構文が使用されます。この名前空間提案をサポートするソフトウェアは、これらの宣言および接頭辞を認識し、それらに対して機能する必要があります。

表記法および使用方法

この仕様の生成規則内の多くの非終端記号は、この仕様ではなく、W3C の XML 勧告で定義されています。この仕様で定義される非終端記号が W3C の XML 勧告で定義されている非終端記号と同じ名前を持つ場合、この仕様での生成規則は、すべての場合で、W3C の XML 勧告の対応する生成規則に一致する文字列のサブセットと一致します。

この仕様の生成規則では、NSC は、この仕様に準拠するドキュメントが従う必要がある規則の 1 つである「名前空間制約」です。

例で使用するすべてのインターネット・ドメイン名は、w3.org 以外はランダムに選択されたものであり、特別な意味を持つものではありません。

名前空間の宣言

名前空間は、予約済属性のグループを使用して宣言されます。このような属性の名前は、`xmlns` であるか、または接頭辞として `xmlns:` を持つ必要があります。これらの属性は、他の XML 属性と同様に、直接またはデフォルトで提供されます。

名前空間宣言用の属性名

```
[1] NSAttName ::=    PrefixedAttName
                   | DefaultAttName
[2] PrefixedAttName ::= 'xmlns:' NCName

[NSC: Leading "XML" ]
[3] DefaultAttName ::= 'xmlns'
[4] NCName ::= (Letter | '_') (NCNameChar)*

/* An XML Name, minus the ":" */
[5] NCNameChar ::= Letter | Digit | '.' | '-' | '_' | CombiningChar
                   | Extender
```

URI 参照である属性の値は、名前空間を識別する名前空間名です。名前空間名は、その目的を果たすために、一意性および永続性という特長を持つ必要があります。名前空間名をスキーマ（存在する場合）の取出しに直接使用できることが目的ではありません。これらの目的を想定して設計された構文の例には、Uniform Resource Name（RFC2141）の構文があります。ただし、通常の URL も、同じこれらの目的を達成するように管理できることに注意してください。

属性名が PrefixedAttName と一致する場合

属性名が `PrefixedAttName` と一致する場合、`NCName` によって接頭辞が与えられます。この接頭辞は、宣言が指定されている要素の有効範囲内にある属性値の名前空間名に要素名および属性名を関連付けるために使用されます。このような宣言では、名前空間名は空であることはできません。

属性名が DefaultAttName と一致する場合

属性名が `DefaultAttName` と一致する場合、その属性値の名前空間名は、宣言が指定されている要素の有効範囲内にあるデフォルトの名前空間の名前空間名です。このようなデフォルトの宣言では、属性値は空であることができます。デフォルトの名前空間および宣言のオーバーライドについては、C-22 ページの「要素および属性への名前空間の適用」（W3C の Namespace 勧告）を参照してください。

次の名前空間宣言の例は、名前空間接頭辞 `edi` を名前空間名 `http://ecommerce.org/schema` に関連付けます。

```
<x xmlns:edi='http://ecommerce.org/schema'>
  <!-- the "edi" prefix is bound to http://ecommerce.org/schema
       for the "x" element and contents -->
</x>
```

名前空間制約 : 先頭の XML

3 つの文字 x、m および l の大 / 小文字を任意に組み合せたシーケンスで始まる接頭辞は、XML 仕様および XML 関連仕様で使用するために予約済です。

修飾名

W3C の Namespace 勧告に準拠する XML 文書では、いくつかの名前（非終端記号 Name に対応する構成メンバー）を、次のとおり定義される修飾名として与えることができます。

修飾名の構文

```
[6] QName ::= (Prefix ':')? LocalPart
[7] Prefix ::= NCName
[8] LocalPart::= NCName
```

接頭辞の概要

Prefix は、修飾名の名前空間接頭辞部分を提供し、名前空間宣言内の名前空間 URI 参照に関連付けられる必要があります。

LocalPart は、修飾名のローカル部分を提供します。接頭辞は名前空間名のプレースホルダとしてのみ機能することに注意してください。有効範囲がその名前を含む文書とそれ以外のものにも適用される名前を構成する場合、アプリケーションは、接頭辞ではなく名前空間名を使用する必要があります。

修飾名の使用

W3C の Namespace 勧告に準拠する XML 文書では、要素型は、次のとおり修飾名として与えられます。

要素型

```
[9]  STag ::= '<' QName (S Attribute)* S? '>' [NSC: Prefix Declared ]
[10] ETag:: = '</' QName S? '>'[NSC: Prefix Declared ]
[11] EmptyElemTag ::= '<' QName (S Attribute)* S? '/>' [NSC: Prefix Declared ]
```

次に、要素型として機能する修飾名の例を示します。

```
<x xmlns:edi='http://ecommerce.org/schema'>
  <!-- the 'price' element's namespace is http://ecommerce.org/schema -->
  <edi:price units='Euro'>32.18</edi:price>
</x>
```

属性は、名前空間宣言か、またはその属性の名前が修飾名として与えられたもののどちらかです。

属性

```
[12] Attribute::= NSAttName Eq AttValue|QName Eq AttValue [NSC:Prefix Declared]
```

次に、属性名として機能する修飾名の例を示します。

```
<x xmlns:edi='http://ecommerce.org/schema'>
   <!-- the 'taxClass' attribute's namespace is http://ecommerce.org/schema -->
   <lineItem edi:taxClass="exempt">Baby food</lineItem>
</x>
```

名前空間制約 : 宣言された接頭辞

名前空間接頭辞は、xml または xmlns でないかぎり、その接頭辞を使用する要素の開始タグまたは祖先要素（接頭辞付きのマークアップがその内容に出現する要素）の名前空間宣言属性で宣言されている必要があります。

接頭辞 xml は、定義上、名前空間名 http://www.w3.org/XML/1998/namespace にバインドされます。

接頭辞 xmlns は、名前空間バインディングのみに使用され、それ自体はどの名前空間名にもバインドされません。

この制約は、名前空間宣言属性が直接 XML 文書エンティティ内ではなく、外部エンティティ内で宣言されたデフォルトの属性を介して提供される場合、操作上の問題を発生させる場合があります。このような宣言は、非検証 XML プロセッサに基づくソフトウェアでは、読み込まれない場合があります。

多くの XML アプリケーションは、名前空間を認識できるものを含め、検証プロセッサを正常に要求できません。このようなアプリケーションで正しく操作するために、名前空間宣言は、直接提供されるか、または DTD の内部サブセットで宣言されたデフォルトの属性を介して提供される必要があります。

要素名および属性型は、DTD 内の宣言に出現する場合にも、修飾名として与えられます。

宣言内の修飾名

```
[13] doctypedecl::= '<!DOCTYPE' S QName (S ExternalID)? S? ('[' (markupdecl |
                    PEReference | S)* ']' S?)? '>'
[14] elementdecl::= '<!ELEMENT' S QName S contentspec S? '>'
[15] cp         ::= (QName | choice | seq) ('?' | '*' | '+')?
[16] Mixed      ::= '(' S? '#PCDATA' (S? '|' S? QName)* S? ')*'
                    | '(' S? '#PCDATA' S? ')'
[17] AttlistDecl::= '<!ATTLIST' S QName AttDef* S? '>'
[18] AttDef     ::=  S (QName | NSAttName) S AttType S DefaultDecl
```

要素および属性への名前空間の適用

名前空間の有効範囲決定

名前空間宣言は、同じ NSAttName 部分を持つ別の名前空間宣言によってオーバーライドされないかぎり、それが指定された要素、およびその要素の内容に存在するすべての要素に適用されます。

```
<?xml version="1.0"?>
  <!-- all elements here are explicitly in the HTML namespace -->
  <html:html xmlns:html='http://www.w3.org/TR/REC-html40'>
    <html:head><html:title>Frobnostication</html:title></html:head>
    <html:body><html:p>Moved to
      <html:a href='http://frob.com'>here.</html:a></html:p></html:body>
</html:html>
```

次の例に示すとおり、複数の名前空間接頭辞を単一の要素の属性として宣言できます。

```
<?xml version="1.0"?>
  <!-- both namespace prefixes are available throughout -->
  <bk:book xmlns:bk='urn:loc.gov:books'
           xmlns:isbn='urn:ISBN:0-395-36341-6'>
      <bk:title>Cheaper by the Dozen</bk:title>
      <isbn:number>1568491379</isbn:number>
  </bk:book>
```

名前空間のデフォルト設定

デフォルトの名前空間は、それが宣言された要素（その要素が名前空間接頭辞を持たない場合）、およびその要素の内容に存在する接頭辞を持たないすべての要素に適用されます。デフォルトの名前空間宣言内の URI 参照が空である場合、その宣言の有効範囲内の接頭辞がない要素は、どの名前空間にも存在しないとみなされます。デフォルトの名前空間は、直接属性には適用されないことに注意してください。

```
<?xml version="1.0"?>
  <!-- elements are in the HTML namespace, in this case by default -->
  <html xmlns='http://www.w3.org/TR/REC-html40'>
    <head><title>Frobnostication</title></head>
    <body><p>Moved to
      <a href='http://frob.com'>here</a>.</p></body>
  </html>

<?xml version="1.0"?>
  <!-- unprefixed element types are from "books" -->
  <book xmlns='urn:loc.gov:books'
        xmlns:isbn='urn:ISBN:0-395-36341-6'>
      <title>Cheaper by the Dozen</title>
```

```
      <isbn:number>1568491379</isbn:number>
  </book>
```

次に、名前空間の有効範囲決定のより大きい例を示します。

```
<?xml version="1.0"?>
  <!-- initially, the default namespace is "books" -->
  <book xmlns='urn:loc.gov:books'
        xmlns:isbn='urn:ISBN:0-395-36341-6'>
      <title>Cheaper by the Dozen</title>
      <isbn:number>1568491379</isbn:number>
      <notes>
        <!-- make HTML the default namespace for some commentary -->
        <p xmlns='urn:w3-org-ns:HTML'>
            This is a <i>funny</i> book!
        </p>
      </notes>
  </book>
```

デフォルトの名前空間は、空の文字列に設定できます。これは、その宣言の有効範囲内では、デフォルトの名前空間が存在しない場合と同じ効果があります。

```
<?xml version='1.0'?>
  <Beers>
    <!-- the default namespace is now that of HTML -->
    <table xmlns='http://www.w3.org/TR/REC-html40'>
     <th><td>Name</td><td>Origin</td><td>Description</td></th>
     <tr>
       <!-- no default namespace inside table cells -->
       <td><brandName xmlns="">Huntsman</brandName></td>
       <td><origin xmlns="">Bath, UK</origin></td>
       <td>
         <details xmlns=""><class>Bitter</class><hop>Fuggles</hop>
           <pro>Wonderful hop, light alcohol, good summer beer</pro>
           <con>Fragile; excessive variance pub to pub</con>
           </details>
          </td>
        </tr>
      </table>
    </Beers>
```

属性の一意性

この仕様に準拠する XML 文書では、どのタグも次のいずれかの条件に該当する 2 つの属性を含むことができません。

- 同一の名前を持つ 2 つの属性
- 同じローカル部分を含み、同一の名前空間名にバインドされている接頭辞を含む修飾名を持つ 2 つの属性

たとえば、次の例では、各 bad 開始タグは無効です。

```
<!-- http://www.w3.org is bound to n1 and n2 -->
  <x xmlns:n1="http://www.w3.org"
     xmlns:n2="http://www.w3.org" >
    <bad a="1"     a="2" />
    <bad n1:a="1"  n2:a="2" />
  </x>
```

ただし、デフォルトの名前空間は属性名に適用されないため、次の各 good 開始タグは有効です。

<!-- http://www.w3.org is bound to n1 and is the default -->

```
  <x xmlns:n1="http://www.w3.org"
     xmlns="http://www.w3.org" >
    <good a="1"     b="2" />
    <good a="1"     n1:a="2" />
  </x>
```

XML 文書の準拠

W3C の Namespace 勧告に準拠する XML 文書では、要素型および属性名は、QName に対する生成規則に適合し、「名前空間制約」を満たす必要があります。

XML 文書内に存在する、XML 準拠のために Name に対する XML 生成規則に適合する必要がある他のすべてのトークンが、この仕様の NCName に対する生成規則に適合する場合、その XML 文書はこの仕様に準拠します。

このような文書における準拠の効果は、次のとおりです。

- すべての要素型および属性名には、0（ゼロ）または 1 つのコロンが含まれます。
- エンティティ名、PI ターゲットまたは表記法名のいずれにも、コロンは含まれません。

厳密には、ID、IDREF（S）、ENTITY（IES）および NOTATION 型と宣言された属性値は、Name でもあるため、コロンを含まない値である必要があります。

ただし、属性値の宣言された型は、検証プロセッサなどの、マークアップ宣言を読み込むプロセッサのみで使用できます。そのため、検証プロセッサの使用が指定されないかぎり、属性値の内容がこの仕様に準拠しているかどうかが確認されるという保証はありません。

この手引きには、次に示す W3C の Namespace 勧告の付録は含まれていません。

- A. The Internal Structure of XML Namespaces (Non-Normative)（XML 名前空間の内部構造（非規範的））
- A.1 The Insufficiency of the Traditional Namespace（従来の名前空間の不十分さ）
- A.2 XML Namespace Partitions（XML 名前空間のパーティション）
- A.3 Expanded Element Types and Attribute Names（拡張要素型および拡張属性名）
- A.4 Unique Expanded Attribute Names（一意の拡張された属性名）

W3C の XML Information Set の概要

W3C の XML Information Set 仕様は、XML Information Set（情報セット）という抽象データ・セットを定義します。この仕様は、整形式の XML 文書内の情報を参照する必要がある他の仕様で使用するための一貫した一連の定義を示します。

情報項目またはプロパティを含める際の主な基準は、将来の仕様において有効であるかどうかです。この基準は、XML プロセッサによって戻される必要がある最小限の情報セットを構成するものではありません。

XML 文書は、整形式であり、次の項に示す名前空間制約を満たす場合、情報セットを持ちます。

情報セットを持つために、XML 文書が妥当である必要はありません。

参照： `http://www.w3.org/TR/xml-infoset/` を参照してください。

情報セットは、XML 文書の解析以外の方法（この仕様では説明しません）によって作成できます。C-27 ページの「総合情報セット」を参照してください。

XML 文書の情報セットは、多くの情報項目で構成されます。どの整形式の XML 文書の情報セットにも、1 つ以上のドキュメント情報項目および他のものが含まれます。情報項目は、XML 文書の一部の抽象的な記述です。各情報項目は、一連の関連付けられた名前付きのプロパティを持ちます。この仕様では、プロパティ名は大カッコで [このように] 示します。情報項目のタイプについては、W3C の XML Information Set の第 2 項を参照してください。

XML Information Set は、特定のインタフェースやインタフェースのクラスを要求または優先しません。この仕様は明確化および単純化のために情報セットを変更されたツリーとして表しますが、XML Information Set をツリー構造で使用できるようにする必要はありません。イベントベースおよび問合せベースのインタフェースなどを含む、他の種類のインタフェースも、XML Information Set に準拠する情報を提供できます。

「情報セット」および「情報項目」という用語は、コンピュータ業界の一般的な用語である「ツリー」および「ノード」と類似する意味を持ちます。ただし、この仕様では、考えられる他の特定のデータ・モデルとの混同を避けるために、「情報セット」および「情報項目」

という用語を使用します。情報項目は、DOM のノードまたは XPath データ・モデルの「ツリー」および「ノード」とは 1 対 1 でマップされません。

名前空間

W3C の Namespace 勧告に準拠しない XML 1.0 文書は、技術的には整形式であっても、意味のある情報セットを持つとはみなされません。この仕様は、W3C の Namespace 勧告で規定されている以外の方法で使用されるコロンを含む要素名または属性名を持つドキュメントに対する情報セットを定義しません。

また、XML Information Set 仕様は、名前空間宣言で相対 URI 参照を使用するドキュメントに対する情報セットも定義しません。これは、W3C の Namespace 勧告の「Relative Namespace URI References（相対名前空間 URI 参照）」に示される W3C XML Plenary Interest Group の決定に基づいています。

[namespace name] プロパティの値は、対応する名前空間属性の正規化された値です。プロセッサによって、この値に追加の URI エスケープが適用されることはありません。

エンティティ

情報セットは、実体参照がすでに拡張され、それらの置換テキストに対応する情報項目によって表現された状態で、その XML 文書を記述します。ただし、プロセッサがこの拡張を実行しない場合がある様々な環境が存在します。エンティティは、宣言または取出しされない場合があります。非検証プロセッサは、一部の宣言を読み込まない場合や、すべての宣言を読み込む場合でも、一部の外部エンティティを拡張しない場合があります。このような場合、拡張されない実体参照情報項目が、その実体参照を表すために使用されます。

行端の処理

情報セットのすべてのプロパティの値は、XML 勧告の第 2.11 項「End-of-Line Handling（行端の処理）」に示される行端の正規化を考慮したものです。

ベース URI

いくつかの情報項目は、[base URI] プロパティまたは [declaration base URI] プロパティを持ちます。これらのプロパティは、XML Base に従って計算されます。リソースの取出しには、パーサー・レベル（エンティティ・リゾルバ内など）以下のレベルで、リダイレクションが含まれる場合があります。この場合、ベース URI は、すべてのリダイレクション後にリソースを取り出すために使用される最後の URI です。

これらのプロパティの値は、リソースを取り出すために必要な場合がある URI エスケープを反映しませんが、ドキュメント内で指定されるか、またはリダイレクションの場合にサーバーによって戻された場合、エスケープされた文字を含むことができます。

場合によっては（文字列またはパイプから読み込まれたドキュメントなど）、XML Base の規則によって、アプリケーション依存のベース URI になる場合があります。これらの場合に

は、この仕様は、[base URI] プロパティまたは [declaration base URI] プロパティの値を定義しません。

相対 URI の解決時には、xml:base 属性の値よりも [base URI] プロパティを使用する必要があります。総合情報セットの場合、相対 URI には一貫性がない場合があります。

Unknown および No Value

一部のプロパティは、unknown または no value という値を持つ場合があり、それぞれ「プロパティ値が不明」または「プロパティが値を持たない」ことを意味します。これらの 2 つの値は、異なる値であり、他のすべての値とも異なります。特にこれらの値は、それぞれ単にメンバーを持たない空の文字列、空のセットおよび空のリストとは異なります。NULL という用語は特別な意味を持つ場合があり、ここで意図する意味とは一致しない場合があるため、この仕様では使用しません。

総合情報セット

この仕様では、XML 文書を解析した結果の情報セットについて説明します。情報セットは、DOM などの API を使用する方法、既存の情報セットを変換する方法など、その他の方法でも構成できます。

実際のドキュメントに対応する情報セットは、様々な方法で一貫している必要があります。たとえば、要素の [in-scope namespaces] プロパティは、その要素およびその祖先の [namespace attributes] プロパティと一貫しています。これは、その他の方法で構成された情報セットには該当しない場合があります。このような場合、情報セットに対応する XML 文書は存在せず、それをシリアル化するには、（有効範囲内の名前空間に対応する名前空間宣言の出力などによって）非一貫性を解決する必要があります。

D

XSLT の手引き

この付録では、W3C の XSL 勧告および XSLT 勧告の概要について説明します。この付録の内容は次のとおりです。

XSL の概要

XML 文書には、構造はありますがフォーマットはありません。eXtensible Stylesheet Language（XSL）は、XML 文書にフォーマットを追加します。XSL は、XML セマンティクスを表示する方法を提供します。XSL を使用すると、XML 要素を HTML などの他のフォーマット言語にマップできます。

参照： 次の章または URL を参照してください。

- 『Oracle9*i* ケース・スタディ - XML アプリケーション』のコンテンツのカスタマイズ、Oracle9*i*AS Wireless Edition、および XML と XSL を使用した表示のカスタマイズに関する章
- http://www.oasis-open.org/cover/xsl.html
- http://www.mulberrytech.com/xsl/xsl-list/
- http://www.builder.com/Authoring/XmlSpot/?tag=st.cn.sr1.ssr.bl_xml
- http://www.zvon.org/HTMLonly/XSLTutorial/Books/Book1/index.html
- 第 6 章「XMLType データの変換および検証」

W3C の XSL Transformation バージョン 1.0 勧告

この仕様は、XML 文書を他の XML 文書へ変換するための言語である XSLT の構文およびセマンティクスを定義します。

XSLT は、XML 用のスタイルシート言語である XSL の一部として使用するために設計されています。XSL には、XSLT の他に、書式設定用の XML ボキャブラリが含まれます。XSL は、XSLT を使用して XML 文書のスタイルを指定し、その文書を、書式設定用ボキャブラリを使用する別の XML 文書へ変換する方法を記述します。

XSLT は、XSL から独立して使用できるようにも設計されています。ただし、XSLT は、完全な XML 変換用の汎用言語ではありません。XSL の一部として XSLT を使用する場合に必要な変換処理を目的として設計されています。

参照： `http://www.w3.org/TR/xslt` を参照してください。

この仕様は、XSLT 言語の構文およびセマンティクスを定義します。XSLT 言語での変換は、XML Namespace 勧告に準拠する整形式の XML 文書として表現されます。この XML 文書には、XSLT によって定義された要素と XSLT によって定義されない要素の両方が含まれる場合があります。

XSLT によって定義された要素は、特定の XML 名前空間（「2.1 XSLT Namespace」を参照）に属することによって区別されます。この仕様では、この XML 名前空間を XSLT 名前空間といいます。この仕様は、XSLT 名前空間の構文およびセマンティクスを定義します。

XSLT で表現される変換は、ソース・ツリーから結果ツリーへ変換する規則を表します。この変換は、テンプレートとパターンを対応付けることによって実行されます。パターンは、ソース・ツリー内の要素と一致します。結果ツリーの一部を作成するために、テンプレートはインスタンス化されます。結果ツリーは、ソース・ツリーから切り離されています。結果ツリーは、ソース・ツリーの構造と異なる構造にできます。結果ツリーを構築するときに、ソース・ツリーの要素をフィルタしたり、再順序付けすることができます。また、任意の構造を追加することもできます。

XSLT で表現される変換を、スタイルシートといいます。これは、XSLT を XSL の書式設定用ボキャブラリに変換するときに、この変換がスタイルシートとして機能するためです。

このマニュアルでは、XSLT スタイルシートを XML 文書に対応付ける方法は指定しません。示されているメカニズムをサポートする XSLT プロセッサを使用することをお薦めします。このメカニズムや他のメカニズムで複数の XSLT スタイルシートのシーケンスが同時に 1 つの XML 文書に適用される場合、その効果はシーケンスの各メンバーを順番にインポートする単一のスタイルシートを適用した場合と同じになります。

スタイルシートには、一連のテンプレート規則が含まれます。テンプレート規則には、ソース・ツリー内のノードと一致するパターン、およびインスタンス化して結果ツリーの一部を形成するテンプレートという 2 つの部分があります。これによって、類似したソース・ツリー構造を持つ様々なドキュメントに、スタイルシートを適用できます。

W3C は、スタイルシート活動の一部として XSL 仕様を策定しています。XSL には、スタイルの指定以外に、ドキュメントを操作する機能もあります。XSL は、XML のスタイルシート言語です。

1999 年 7 月の W3C の XSL 仕様は、次の 2 つのドキュメントに分割されています。

- XSL の構文およびセマンティクス
- XSL を使用してスタイルシートを適用し、ドキュメントを別のドキュメントに変換する方法

XSL で使用される書式設定オブジェクトは、カスケーディング・スタイルシート（CSS）および文書スタイル意味指定言語（DSSSL）での作業に基づいています。XSL は、DSSSL より簡単に使用できるように設計されています。

XSL 提案に定義されている機能によって、次の機能が使用可能になります。

- 祖先と子孫、位置および一意性に基づいたソース要素のフォーマット
- フォーマット構造体の作成（生成されたテキストおよび図形を含む）
- 再利用可能な書式設定用マクロの定義
- 書込み先に依存しないスタイルシート
- 拡張可能な一連の書式設定オブジェクト

参照： `http://www.w3.org/Style/XSL/` を参照してください。

XML の名前空間

名前空間とは、一意の識別子または名前です。XML 文書は、異なる DTD または XML Schema を使用して個別に作成できるため、名前空間が必要です。名前空間は、タグが生成された DTD または XML Schema を識別することによって、マークアップ・タグの競合を回避します。また、XML 要素を特定の DTD または XML Schema へリンクします。

`rml:`、`xhtml:`、`xsl:` などの名前空間マーカーを使用する前に、次の段落で説明する名前空間インジケータ `xmlns` を使用して、マーカーを識別する必要があります。

> **参照：** http://w3.org/TR/REC-xml-names を参照してください。

XSLT スタイルシートのアーキテクチャ

XSLT スタイルシートには、次の構文を含める必要があります。

- スタイルシートを指定する開始タグ（`<xsl:stylesheet2>` など）。
- 名前空間インジケータ（XSL の名前空間インジケータ `xmlns:xsl="http//www.w3.org/TR/WD-xsl"`、書式設定オブジェクトの名前空間インジケータ `xmlns:fo="http//www.w3.org/TR/WD-xsl/FO"` など）。
- フォントの種類と太さ、色、改ページなどのテンプレート規則。テンプレートには、要素および要素値の制御指示が含まれます。
- スタイルシートの定義の終了を示すタグ（`</xsl:stylesheet2>`）。

XSL Transformation（XSLT）

XSLT は、XSL の 1 つ目の部分として使用されるように設計されています。XSL には、XSLT の他に、書式設定用の XML ボキャブラリが含まれます。XSL は、XSLT を使用して XML 文書のスタイルを指定し、その文書を、書式設定用ボキャブラリを使用する別の XML 文書へ変換する方法を記述します。

XSL の 2 つ目の部分では、XSL 書式設定オブジェクト、その属性、および書式設定オブジェクトを組み合せる方法を記述します。

> **参照：** 第 6 章「XMLType データの変換および検証」を参照してください。

XML Path Language（XPath）

XSL に関連する別の仕様が、XPath バージョン 1.0 として公開されています。XPath は、XML 文書の一部を指定するための言語で、XSL によって変換される文書の一部を正確に指定する場合に必須です。たとえば、XPath を使用すると、章の要素に属するすべての段落を選択したり、特記事項の要素を選択することができます。XPath は、XSLT と XPointer の両

方で使用できるように設計されています。XPath は、XSLT と XPointer の間で共有される機能に対して、共通の構文およびセマンティクスを提供するために設計されています。

> **参照：** 付録 C「XPath および Namespace の手引き」を参照してください。

CSS と XSL の比較

W3C は、相互運用可能な書式設定モデルの実装の実現に取り組んでいます。

CSS

カスケーディング・スタイルシート（CSS）は、HTML ドキュメントのスタイルを指定するために使用します。CSS は、W3C のスタイルに関するワーキング・グループにより策定されています。CSS2 は、作成者またはユーザーが HTML ドキュメントや XML アプリケーションなどの構造化ドキュメントにスタイル（たとえば、フォント、間隔、音声キュー）を指定できるスタイルシート言語です。

CSS2 によって、ドキュメントの表示スタイルがドキュメントのコンテンツから分離されるため、Web オーサリングおよび Web サイトのメンテナンスが簡単になります。

XSL

一方、XSL はドキュメントを変換できます。たとえば、XSL を使用して、XML データを Web サーバー上で HTML/CSS ドキュメントへ変換できます。このように、2 つの言語は相互に補足し合っており、ともに使用できます。どちらの言語を使用しても、XML 文書のスタイルを指定できます。CSS および XSL は、基礎となる同一の書式設定モデルを使用するため、設計者は、両方の言語で同じ書式設定機能を使用できます。

XSL が画面上のドキュメントのレンダリングに使用するモデルは、ISO 標準の複雑なスタイル言語である DSSSL への長年の取組みに基づいて構築されています。XSL は、主に複雑なドキュメント作成プロジェクトを目的としており、目次、索引およびレポートの自動生成、およびその他のより複雑な出版タスクにおいても、様々な用途があります。

XSLT スタイルシートの例 : PurchaseOrder.xsl

次の例に示す `PurchaseOrder.xsl` は、XSLT スタイルシートの例です。このスタイルシートの例は、第 3 章「Oracle XML DB の使用」の例で使用されています。

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns:xdb="http://xmlns.oracle.com/xdb"
                xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <xsl:template match="/">
    <html>
```

```
<head/>
<body bgcolor="#003333" text="#FFFFCC" link="#FFCC00"
      vlink="#66CC99" alink="#669999">
  <FONT FACE="Arial, Helvetica, sans-serif">
    <xsl:for-each select="PurchaseOrder"/>
    <xsl:for-each select="PurchaseOrder">
      <center>
        <span style="font-family:Arial; font-weight:bold">
          <FONT COLOR="#FF0000">
            <B>Purchase Order </B>
          </FONT>
        </span>
      </center>
      <br/>
      <center>
        <xsl:for-each select="Reference">
          <span style="font-family:Arial; font-weight:bold">
            <xsl:apply-templates/>
          </span>
        </xsl:for-each>
      </center>
    </xsl:for-each>
    <P>
      <xsl:for-each select="PurchaseOrder">
        <br/>
      </xsl:for-each>
      <P/>
      <P>
        <xsl:for-each select="PurchaseOrder">
          <br/>
        </xsl:for-each>
      </P>
    </P>
    <xsl:for-each select="PurchaseOrder"/>
    <xsl:for-each select="PurchaseOrder">
      <table border="0" width="100%" BGCOLOR="#000000">
        <tbody>
          <tr>
            <td WIDTH="296">
              <P>
                <B>
                  <FONT SIZE="+1" COLOR="#FF0000"
                   FACE="Arial, Helvetica, sans-serif">Internal
                  </FONT>
                </B>
              </P>
              <table border="0" width="98%" BGCOLOR="#000099">
```

```
<tbody>
  <tr>
    <td WIDTH="49%">
      <B>
        <FONT COLOR="#FFFF00">Actions</FONT>
      </B>
    </td>
    <td WIDTH="51%">
      <xsl:for-each select="Actions">
        <xsl:for-each select="Action">
          <table border="1" WIDTH="143">
            <xsl:if test="position()=1">
              <thead>
                <tr>
                  <td HEIGHT="21">
                    <FONT
                     COLOR="#FFFF00">User</FONT>
                  </td>
                  <td HEIGHT="21">
                    <FONT
                     COLOR="#FFFF00">Date</FONT>
                  </td>
                </tr>
              </thead>
            </xsl:if>
            <tbody>
              <tr>
                <td>
                  <xsl:for-each select="User">
                    <xsl:apply-templates/>
                  </xsl:for-each>
                </td>
                <td>
                  <xsl:for-each select="Date">
                    <xsl:apply-templates/>
                  </xsl:for-each>
                </td>
              </tr>
            </tbody>
          </table>
        </xsl:for-each>
      </xsl:for-each>
    </td>
  </tr>
  <tr>
    <td WIDTH="49%">
      <B>
```

```
                <FONT COLOR="#FFFF00">Requestor</FONT>
              </B>
            </td>
            <td WIDTH="51%">
              <xsl:for-each select="Requestor">
                <xsl:apply-templates/>
              </xsl:for-each>
            </td>
          </tr>
          <tr>
            <td WIDTH="49%">
              <B>
                <FONT COLOR="#FFFF00">User</FONT>
              </B>
            </td>
            <td WIDTH="51%">
              <xsl:for-each select="User">
                <xsl:apply-templates/>
              </xsl:for-each>
            </td>
          </tr>
          <tr>
            <td WIDTH="49%">
              <B>
                <FONT COLOR="#FFFF00">Cost Center</FONT>
              </B>
            </td>
            <td WIDTH="51%">
              <xsl:for-each select="CostCenter">
                <xsl:apply-templates/>
              </xsl:for-each>
            </td>
          </tr>
        </tbody>
      </table>
    </td>
    <td width="93"/>
    <td valign="top" WIDTH="340">
      <B>
        <FONT COLOR="#FF0000">
          <FONT SIZE="+1">Ship To</FONT>
        </FONT>
      </B>
      <xsl:for-each select="ShippingInstructions">
        <xsl:if test="position()=1"/>
      </xsl:for-each>
      <xsl:for-each select="ShippingInstructions">
```

```
<xsl:if test="position()=1">
  <table border="0" BGCOLOR="#999900">
    <tbody>
      <tr>
        <td WIDTH="126" HEIGHT="24">
          <B>Name</B>
        </td>
        <xsl:for-each
             select="../ShippingInstructions">
          <td WIDTH="218" HEIGHT="24">
            <xsl:for-each select="name">
              <xsl:apply-templates/>
            </xsl:for-each>
          </td>
        </xsl:for-each>
      </tr>
      <tr>
        <td WIDTH="126" HEIGHT="34">
          <B>Address</B>
        </td>
        <xsl:for-each
             select="../ShippingInstructions">
          <td WIDTH="218" HEIGHT="34">
            <xsl:for-each select="address">
              <span style="white-space:pre">
                <xsl:apply-templates/>
              </span>
            </xsl:for-each>
          </td>
        </xsl:for-each>
      </tr>
      <tr>
        <td WIDTH="126" HEIGHT="32">
          <B>Telephone</B>
        </td>
        <xsl:for-each
             select="../ShippingInstructions">
          <td WIDTH="218" HEIGHT="32">
            <xsl:for-each select="telephone">
              <xsl:apply-templates/>
            </xsl:for-each>
          </td>
        </xsl:for-each>
      </tr>
    </tbody>
  </table>
</xsl:if>
```

```
        </xsl:for-each>
      </td>
    </tr>
  </tbody>
</table>
<br/>
<B>
  <FONT COLOR="#FF0000" SIZE="+1">Items:</FONT>
</B>
<br/>
<br/>
<table border="0">
  <xsl:for-each select="LineItems">
    <xsl:for-each select="LineItem">
      <xsl:if test="position()=1">
        <thead>
          <tr bgcolor="#C0C0C0">
            <td>
              <FONT COLOR="#FF0000">
                <B>ItemNumber</B>
              </FONT>
            </td>
            <td>
              <FONT COLOR="#FF0000">
                <B>Description</B>
              </FONT>
            </td>
            <td>
              <FONT COLOR="#FF0000">
                <B>PartId</B>
              </FONT>
            </td>
            <td>
              <FONT COLOR="#FF0000">
                <B>Quantity</B>
              </FONT>
            </td>
            <td>
              <FONT COLOR="#FF0000">
                <B>Unit Price</B>
              </FONT>
            </td>
            <td>
              <FONT COLOR="#FF0000">
                <B>Total Price</B>
              </FONT>
            </td>
```

```
    </tr>
  </thead>
</xsl:if>
<tbody>
  <tr bgcolor="#DADADA">
    <td>
      <FONT COLOR="#000000">
        <xsl:for-each select="@ItemNumber">
          <xsl:value-of select="."/>
        </xsl:for-each>
      </FONT>
    </td>
    <td>
      <FONT COLOR="#000000">
        <xsl:for-each select="Description">
          <xsl:apply-templates/>
        </xsl:for-each>
      </FONT>
    </td>
    <td>
      <FONT COLOR="#000000">
        <xsl:for-each select="Part">
          <xsl:for-each select="@Id">
            <xsl:value-of select="."/>
          </xsl:for-each>
        </xsl:for-each>
      </FONT>
    </td>
    <td>
      <FONT COLOR="#000000">
        <xsl:for-each select="Part">
          <xsl:for-each select="@Quantity">
            <xsl:value-of select="."/>
          </xsl:for-each>
        </xsl:for-each>
      </FONT>
    </td>
    <td>
      <FONT COLOR="#000000">
        <xsl:for-each select="Part">
          <xsl:for-each select="@UnitPrice">
            <xsl:value-of select="."/>
          </xsl:for-each>
        </xsl:for-each>
      </FONT>
    </td>
    <td>
```

```
                        <FONT FACE="Arial, Helvetica, sans-serif"
                              COLOR="#000000">
                          <xsl:for-each select="Part">
                            <xsl:value-of select="@Quantity*@UnitPrice"/>
                          </xsl:for-each>
                        </FONT>
                      </td>
                    </tr>
                  </tbody>
                </xsl:for-each>
              </xsl:for-each>
            </table>
          </xsl:for-each>
        </FONT>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

E

Java DOM API for XMLType および Resource API for Java: クイック・リファレンス

この付録では、次の Oracle XML DB の Java API のクイック・リファレンスを示します。

- Java DOM API for XMLType
- Oracle XML DB Resource API for Java

Java DOM API for XMLType

oracle.xdb パッケージおよび oracle.xdb.dom パッケージは、Java DOM API for XMLType を実装します。Java DOM API for XMLType は、W3C の DOM レベル 1.0 および 2.0 コア勧告を実装し、Oracle 固有の拡張機能を提供します。

表 E–1 に、Java DOM API for XMLType（oracle.xdb.dom および oracle.xdb）のクラスを示します。XMLType クラスは、oracle.xdb.dom パッケージではなく oracle.xdb パッケージに存在することに注意してください。

参照： 次のマニュアルまたは章を参照してください。

- 『Oracle9*i* XML API リファレンス - XDK および Oracle XML DB』
- 第 9 章「Java API for XMLType」

サポートされない Java メソッド

次のメソッドは、リリース 2（9.2.0.1）のマニュアルには記載されていますが、リリース 2（9.2.0.2）では現在サポートされていません。

- XDBDocument.getElementByID
- XDBDocument.importNode
- XDBNode.normalize
- XDBNode.isSupported
- XDBDomImplementation.hasFeature

表 E-1 Java DOM API for XMLType（主に oracle.xdb.dom）のクラス

Java DOM API for XMLType	説明
XDBAttribute	XOB と対話するために W3C の DOM の Node インタフェースを実装します。
XDBCData	W3C の Text インタフェース org.w3c.dom.CData を実装します。
XDBCharData	W3C の CharacterData インタフェース org.w3c.dom.CharData を実装します。
XDBComment	org.w3c.dom.Comment インタフェースを実装します。

表 E-1 Java DOM API for XMLType（主に oracle.xdb.dom）のクラス（続き）

Java DOM API for XMLType	説明
XDBDocument	`org.w3c.dom.Document` インタフェースを実装します。 メソッド: XDBDocument() コンストラクタ: XDBDocument() - 新しいドキュメントを作成します。サーバーのみで使用できます。 XDBDocument(byte[ ] source) - ソースからドキュメントを移入します。サーバーのみで使用できます。 XDBDocument(Connection conn) - ドキュメント・ソースをキャッシュするために接続をオープンします。 XDBDocument(Connection conn, byte[] source) - ドキュメント・ソースにバイトをキャッシュするための接続です。 XDBDocument(Connection conn, String source) - XML テキストを含む文字列をキャッシュするために接続をオープンします。 XDBDocument(String source) - XML テキストを含む文字列です。サーバーのみで使用できます。 パラメータ: source - 含まれる XML テキスト。 conn - 使用される接続。
XDBDomImplementation	`org.w3c.dom.DomImplementation` を実装します。 メソッド: XDBDomImplementation() - サーバーへの JDBC 接続をオープンします。
XDBElement	`org.w3c.dom.Element` を実装します。
XDBEntity	`org.w3c.dom.Entity` を実装します。
XDBNodeMap	`org.w3c.dom.NamedNodeMap` を実装します。

表 E-1 Java DOM API for XMLType（主に oracle.xdb.dom）のクラス（続き）

Java DOM API for XMLType	説明
XDBNode	XOB と対話するために W3C の DOM の Node インタフェース `org.w3c.dom.Node` を実装します。 メソッド : write() - このノードおよびすべてのサブノードに対する XML を出力ストリームに書き込みます。出力ストリームが ServletOutputStream である場合、サーブレット出力がコミットされ、ネイティブなストリームを使用してデータが書き込まれます。 public void write(OutputStream s, String charEncoding, short indent); パラメータ : s - 出力される XML テキスト toContains を書き込むためのストリーム。 charEncoding - IANA 文字コード（ISO 8859 など）。 indent - ネストした要素をインデントする文字数。
XDBNodeList	`org.w3c.dom.NodeList` を実装します。
XDBNotation	`org.w3c.dom.Notation` を実装します。
XDBProcInst	W3C の DOM の ProcessingInstruction インタフェース `org.w3c.dom.ProcInst` を実装します。
XDBText	`org.w3c.dom.Text` を実装します。

表 E-1 Java DOM API for XMLType（主に oracle.xdb.dom）のクラス（続き）

Java DOM API for XMLType	説明
XMLType （oracle.xdb パッケージ）	SQL 型 SYS.XMLTYPE に対する Java メソッドを実装します。 メソッド: createXML() - XMLType を作成します。JDBC を介してデータにアクセスする場合、このメソッドを使用します。 getStringVal() - XMLType から XML データを含む文字列値を取得します。 getClobVal() - XMLType から XML データを含む CLOB 値を取得します。 extract() - XMLType から任意のノードのセットを抽出します。 existsNode() - XMLType に任意のノードのセットが存在するかどうかを確認します。 transform() - 任意の XSL ドキュメントを使用して XMLType を変換します。 isFragment() - XMLType が通常の文書であるか、または文書のフラグメントであるかを確認します。 getDOM() - XMLType に関連付けられた DOM 文書を取得します。

表 E-1 Java DOM API for XMLType（主に oracle.xdb.dom）のクラス（続き）

Java DOM API for XMLType	説明
createXML()	XMLType を作成します。XMLType を作成できない場合、java.sql.SQLException を発生させます。 public static XMLType createXML(OPAQUE opq) - XMLType のバイトを含む指定された OPAQUE 型を使用して XMLType を作成して戻します。 public static XMLType createXML(Connection conn, String xmlval) - XML データを含む文字列が指定された XMLType を作成して戻します。 public static XMLType createXML(Connection conn, CLOB xmlval) - XML データを含む CLOB が指定された XMLType を作成して戻します。 public static XMLType createXML(Connection conn, Document domdoc) - DOM 文書のインスタンスが指定された XMLType を作成して戻します。 パラメータ： opq - XMLType が構成される OPAQUE 型のオブジェクト。 conn - 使用される接続オブジェクト。 xmlval - 含まれる XML データ。 domdoc - DOM ツリーを表す DOM 文書。
getStringVal()	XMLType から XML データを含む文字列値を取得します。java.sql.SQLException を発生させます。 public String getStringVal();
getClobVal()	XMLType から XML データを含む CLOB 値を取得します。java.sql.SQLException を発生させます。 public CLOB getClobVal();
extract()	XMLType から任意のノードのセットを抽出して戻します。ノードのセットは、XPath 式によって指定されます。元の XMLType は変更されません。Thick の場合のみに機能します。指定された式に一致するノードが存在しない場合、NULL を戻します。java.sql.SQLException を発生させます。 public XMLType extract(String xpath, String nsmap); パラメータ： xpath - 検索するノードを指定する XPath 式。 nsmap - XPath 式の接頭辞を解決する名前空間のマッピング（フォーマットは「xmlns=a.com xmlns:b=b.com」）。

表 E-1 Java DOM API for XMLType（主に oracle.xdb.dom）のクラス（続き）

Java DOM API for XMLType	説明
existsNode()	XMLType に任意のノードのセットが存在するかどうかを確認します。このノードのセットは、XPath 式によって指定されます。XMLType に指定されたノードが存在する場合、TRUE を戻します。そうでない場合、FALSE を戻します。java.sql.SQLException を発生させます。 public boolean existsNode(String xpath, String nsmap); パラメータ： xpath - 検索するノードを指定する XPath 式。 nsmap - XPath 式の接頭辞を解決する名前空間のマッピング（フォーマットは「xmlns=a.com xmlns:b=b.com」）。
transform()	任意の XSL ドキュメントを使用して XMLType を変換して戻します。新しい（変換済）XML 文書が戻されます。java.sql.SQLException を発生させます。 public XMLType transform(XMLType xsldoc, String parammap); パラメータ： xsldoc - XMLType に適用される XSL ドキュメント。 parammap - XSLT 変換に渡される最上位のパラメータ（フォーマットは「a=b c=d e=f」、NULL に設定可能）。
isFragment()	XMLType が通常の文書であるか、または文書のフラグメントであるかを確認します。文書のフラグメントである場合、TRUE を戻します。そうでない場合、FALSE を戻します。java.sql.SQLException を発生させます。 public boolean isFragment();
getDOM()	XMLType に関連付けられた DOM 文書を取得します。この文書は、org.w3c.dom.Document です。コール元から文書に対するすべての DOM 操作を実行できます。文書がバイナリである場合、getDOM を実行すると NULL が戻されます。java.sql.SQLException を発生させます。 public org.w3c.dom.Document getDOM();

Oracle XML DB Resource API for Java

Oracle XML DB Resource API for Java の WebDAV のサポートは、サービス・プロバイダ・インタフェース（SPI）・ドライバをレンダリングする oracle.xdb.spi パッケージのクラスを使用して実装されます。oracle.xdb.spi のクラスは、Oracle XML DB に対して、コアである WebDAV のサポートを実装します。表 E-2 に、oracle.xdb.spi のクラスを示します。

> **参照：** 第 17 章「Oracle XML DB Resource API for Java」を参照してください。

表 E-2 Oracle XML DB Resource API for Java（oracle.xdb.spi）

oracle.xdb.spi のクラス	説明
XDBContext Class	Oracle XML DB に Java のネーミングおよびコンテキスト・インタフェースを実装します。これによって、javax.naming.context が拡張されます。今回のリリースでは、フェデレーションはサポートされません。他の名前空間が存在するかどうかは認識されません。 メソッド： XDBContext() - XDBContext クラスのコンストラクタです。 ■ public XDBContext(Hashtable env) - 環境が指定された XDBContext クラスのインスタンスを作成します。 ■ public XDBContext(Hashtable env, String path) - 環境およびパスが指定された XDBContext クラスのインスタンスを作成します。 パラメータ： env - コンテンツのプロパティを記述する環境。 path - コンテキストの初期パス。
XDBContextFactory Class	javax.naming.context を実装します。 メソッド： XDBContextFactory() - XDBContextFactory クラスのコンストラクタです。 public XDBContextFactory();
XDBNameParser Class	javax.naming.NameParser を実装します。
XDBNamingEnumeration Class	javax.naming.NamingEnumeration を実装します。

表 E-2 Oracle XML DB Resource API for Java（oracle.xdb.spi）（続き）

oracle.xdb.spi のクラス	説明
XDBResource Class	Oracle XML DB の JNDI のサービス・プロバイダ・インタフェース（SPI）の主要な機能を実装します。今回のリリースでは、フェデレーションはサポートされません。他の名前空間が存在するかどうかは認識されません。 public class XDBResource extends java.lang.Object. メソッド: XDBResource() - XDBResource の新しいインスタンスを作成します。 getAuthor() - リソースの作成者を戻します。 getComment() - リソースの DAV コメントを戻します。 getContent() - リソースのコンテンツを戻します。 getContentType() - リソースのコンテンツの型を戻します。 getCreateDate() - リソースの作成日を戻します。 getDisplayName() - リソースの表示名を戻します。 getLanguage() - リソースの言語を戻します。 getLastModDate() - リソースの最後の変更日を戻します。 getOwnerId() - リソースの所有者の ID を戻します。 setACL() - リソースに ACL を設定します。 setAuthor() - リソースの作成者を設定します。 setComment() - リソースの DAV コメントを設定します。 setContent() - リソースのコンテンツを設定します。 setContentType() - リソースのコンテンツの型を設定します。 setCreateDate() - リソースの作成日を設定します。 setDisplayName() - リソースの表示名を設定します。 setLanguage() - リソースの言語を設定します。 setLastModDate() - リソースの最後の変更日を設定します。 setOwnerId() - リソースの所有者の ID を設定します。

表 E-2 Oracle XML DB Resource API for Java（oracle.xdb.spi）（続き）

oracle.xdb.spi のクラス	説明
XDBResource()	XDBResource の新しいインスタンスを作成します。環境が指定された XDBResource の新しいインスタンスを作成します。 public XDBResource(Hashtable env, String path) - 環境およびパスが指定された XDBResource の新しいインスタンスを作成します。 パラメータ： env - 渡される環境。 path - リソースへのパス。
getAuthor()	リソースの作成者を取得します。 public String getAuthor();
getComment()	リソースの DAV（Web Distributed Authoring and Versioning）コメントを取得します。 public String getComment();
getContent()	リソースのコンテンツを戻します。 public Object getContent();
getContentType()	リソースのコンテンツの型を戻します。 public String getContentType();
getCreateDate()	リソースの作成日を戻します。 public Date getCreateDate();
getDisplayName()	リソースの表示名を戻します。 public String getDisplayName();
getLanguage()	リソースの言語を戻します。 public String getLanguage();
getLastModDate()	リソースの最後の変更日を戻します。 public Date getLastModDate();
getOwnerId()	リソースの所有者の ID を戻します。このメソッドに対して適切な値は、ALL_USERS などのカタログ・ビューによって提供されるデータベース・ユーザーのユーザー ID の値です。 public long getOwnerId();

表 E-2 Oracle XML DB Resource API for Java（oracle.xdb.spi）（続き）

oracle.xdb.spi のクラス	説明
setACL()	リソースに ACL を設定します。 public void setACL(String aclpath); パラメータ： aclpath - ACL リソースへのパス。
setAuthor()	リソースの作成者を設定します。 public void setAuthor(String authname); パラメータ： authname - リソースの作成者。
setComment()	リソースの DAV（Web Distributed Authoring and Versioning）コメントを設定します。 public void setComment(String davcom); パラメータ： davcom - リソースの DAV コメント。
setContent()	リソースのコンテンツを設定します。 public void setContent(Object xmlobj); パラメータ： xmlobj - リソースのコンテンツ。
setContentType()	リソースのコンテンツの型を設定します。 public void setContentType(String conttype); パラメータ： conttype - リソースのコンテンツの型。
setCreateDate()	リソースの作成日を設定します。 public void setCreateDate(Date credate); パラメータ： credate - リソースの作成日。
setDisplayName()	リソースの表示名を設定します。 public void setDisplayName(String dname); パラメータ： dname - リソースの表示名。

表 E-2 Oracle XML DB Resource API for Java（oracle.xdb.spi）（続き）

oracle.xdb.spi のクラス	説明
setLanguage()	リソースの言語を設定します。 public void setLanguage(String lang); パラメータ： lang - リソースの言語。
setLastModDate()	リソースの最後の変更日を設定します。 public void - setLastModDate(Date d); パラメータ： d - リソースの最後の変更日。
setOwnerId()	リソースの所有者の ID を設定します。このメソッドに対して適切な所有者 ID の値は、ALL_USERS などのカタログ・ビューによって提供されるデータベース・ユーザーのユーザー ID の値です。 public void setOwnerId(long ownerid); パラメータ： ownerid - リソースの所有者の ID。

F

Oracle XML DB の XMLType API、PL/SQL API および Resource PL/SQL API: クイック・リファレンス

この付録では、次の Oracle XML DB の SQL API および PL/SQL API の概要を説明します。

- XMLType API
- PL/SQL DOM API for XMLType（DBMS_XMLDOM）
- PL/SQL Parser for XMLType（DBMS_XMLPARSER）
- PL/SQL XSLT Processor for XMLType（DBMS_XSLPROCESSOR）
- DBMS_XMLSCHEMA
- Oracle XML DB の XML Schema カタログ・ビュー
- Resource API for PL/SQL（DBMS_XDB）
- RESOURCE_VIEW および PATH_VIEW
- DBMS_XDB_VERSION
- DBMS_XDBT

XMLType API

XMLType は、XML データを処理するためのシステム定義の OPAQUE 型です。XMLType には、XML ノードおよびフラグメントを抽出するための事前定義のメンバー関数が含まれます。XMLType の列を作成し、これらの列に XML 文書を挿入できます。SQL 関数の SYS_XMLGEN と SYS_XMLAGG、PL/SQL パッケージの DBMS_XMLGEN、および SQLX 関数を使用して、XML 文書を XMLType インスタンスとして動的に生成することもできます。

表 F-1 に、XMLType API 関数を示します。

参照： 次のマニュアルまたは章を参照してください。

- 『Oracle9*i* XML API リファレンス - XDK および Oracle XML DB』
- 第 4 章「XMLType の使用」

表 F-1 XMLType API

関数	説明
XMLType() constructor function XMLType(xmlData IN clob, schema IN varchar2 := NULL, validated IN number := 0, wellformed IN Number := 0) return self as result constructor function XMLType(xmlData IN varchar2, schema IN varchar2 := NULL,validated IN number := 0, wellformed IN number := 0) return self as result constructor function XMLType (xmlData IN "<ADT_1>", schema IN varchar2 := NULL, element IN varchar2 := NULL, validated IN number := 0) return self as result onstructor function XMLType(xmlData IN SYS_REFCURSOR, schema in varchar2 := NULL, element in varchar2 := NULL, validated in number := 0) return self as result	XMLType データ型のインスタンスを構成するコンストラクタです。コンストラクタは、CLOB または VARCHAR2 の XML またはオブジェクト型を取ることができます。 パラメータ : xmlData - CLOB、REF カーソル、VARCHAR2 またはオブジェクト型のデータ。 schema - 入力を指定されたスキーマに準拠させるために使用されるオプションのスキーマ URL。 validated - 指定された XML Schema に従ってインスタンスが妥当であることを示すフラグ（デフォルトは 0）。 wellformed - 入力が整形式であることを示すフラグ。このフラグが設定されている場合、データベースによって入力インスタンスが整形式であるかどうかの確認は行われません（デフォルトは 0）。 element - ADT_1 または REF CURSOR コンストラクタの場合の、オプションの要素名（デフォルトは NULL）。
--	--

表 F-1 XMLType API （続き）

関数	説明
createXML() STATIC FUNCTION createXML(xmlval IN varchar2) RETURN XMLType deterministic STATIC FUNCTION createXML(xmlval IN clob) RETURN XMLType STATIC FUNCTION createXML (xmlData IN clob, schema IN varchar2, validated IN number := 0, wellformed IN number := 0) RETURN XMLType deterministic STATIC FUNCTION createXML (xmlData IN varchar2, schema IN varchar2, validated IN number := 0, wellformed IN number := 0) RETURN XMLType deterministic STATIC FUNCTION createXML (xmlData IN "<ADT_1>", schema IN varchar2 := NULL, element IN varchar2 := NULL, validated IN NUMBER := 0) RETURN XMLType deterministic STATIC FUNCTION createXML (xmlData IN SYS_REFCURSOR, schema in varchar2 := NULL, element in varchar2 := NULL, validated in number := 0) RETURN XMLType deterministic	XMLType インスタンスを作成して戻すための静的関数です。日付を渡すために使用される文字列および CLOB パラメータには、整形式かつ妥当な XML 文書が含まれている必要があります。オプションについては、次の表を参照してください。 パラメータ : xmlData - CLOB、REF カーソル、VARCHAR2 またはオブジェクト型の実際のデータ。 schema - 入力を指定されたスキーマに準拠させるために使用されるオプションのスキーマ URL。 validated - 指定された XML Schema に従ってインスタンスが妥当であることを示すフラグ（デフォルトは 0）。 wellformed - 入力が整形式であることを示すフラグ。このフラグが設定されている場合、データベースによって入力インスタンスが整形式であるかどうかの確認は行われません（デフォルトは 0）。 element - ADT_1 または REF CURSOR コンストラクタの場合の、オプションの要素名（デフォルトは NULL）。
existsNode() MEMBER FUNCTION existsNode(xpath IN varchar2) RETURN number deterministic MEMBER FUNCTION existsNode(xpath in varchar2, nsmap in varchar2) RETURN number deterministic	XMLType インスタンスおよび XPath を取り、XPath を適用することによって空でないノードのセットが戻されるかどうかを示す 1 または 0（ゼロ）を戻します。XPath 文字列が NULL であるか、ドキュメントが空である場合は 0（ゼロ）、そうでない場合は 1 を戻します。 パラメータ : xpath - テストする XPath 式。 nsmap - オプションの名前空間のマッピング。

表 F-1 XMLType API （続き）

関数	説明
extract() MEMBER FUNCTION extract(xpath IN varchar2) RETURN XMLType deterministic MEMBER FUNCTION extract(xpath IN varchar2, nsmap IN varchar2) RETURN XMLType deterministic	XMLType フラグメントを抽出し、結果ノードを含む XMLType インスタンスを戻します。XPath がどのノードも戻さない場合、NULL になります。 パラメータ： xpath - 適用する XPath 式。 nsmap - 名前空間のマッピングについての情報に対するオプションの接頭辞。
isFragment() MEMBER FUNCTION isFragment() RETURN number deterministic	XMLType インスタンスが整形式の文書に対応しているか、またはフラグメントに対応しているかを決定します。XMLType インスタンスがフラグメントを含んでいるか、整形式文書を含んでいるかを示す 1 または 0（ゼロ）を戻します。
getClobVal() MEMBER FUNCTION getClobVal() RETURN clob deterministic	シリアル化 XML 表現を含む CLOB を戻します。一時 CLOB が戻される場合、使用後に解放する必要があります。
getNumberVal() MEMBER FUNCTION getNumberVal() RETURN number deterministic	XMLType インスタンスが指すテキスト値からフォーマットされる数値を戻します。XMLType は、数値を含む有効なテキスト・ノードを指す必要があります。
getStringVal() MEMBER FUNCTION getStringVal() RETURN varchar2 deterministic	シリアル化 XML 表現を含む文字列として文書を戻します。テキスト・ノードの場合、テキスト自体を戻します。XML 文書が VARCHAR2 の最大サイズ（4000 バイト）より大きい場合、実行時にエラーが発生します。
transform() MEMBER FUNCTION transform(xsl IN XMLType, parammap in varchar2 := NULL) RETURN XMLType deterministic	XSLT スタイルシート引数、および名前と値の組である文字列として渡される最上位のパラメータを使用して XML データを変換します。パラメータ以外の引数が NULL である場合、NULL を戻します。 パラメータ xsl - 変換を記述する XSLT スタイルシート。 parammap - XSL の最上位のパラメータ（名前と値の組である文字列）。

表 F-1 XMLType API （続き）

関数	説明
toObject() MEMBER PROCEDURE toObject(SELF in XMLType, object OUT "<ADT_1>", schema in varchar2 := NULL, element in varchar2 := NULL)	オプションのスキーマおよび最上位の要素の引数を使用して、XML データをユーザー定義の型のインスタンスに変換します。 パラメータ： SELF - 変換されるインスタンス。メンバー・プロシージャとして使用される場合、暗黙的です。 object - 必要な型の変換済オブジェクト・インスタンスが、この関数に渡される場合があります。 schema - スキーマ URL。XMLType インスタンスから変換済オブジェクト・インスタンスへのマッピングは、スキーマを使用して指定できます。 element - 最上位の要素名。XMLType インスタンスをマップする XML Schema 文書の最上位の要素名を指定します。
isSchemaBased() MEMBER FUNCTION isSchemaBased return number deterministic	XMLType インスタンスが XML Schema に基づくかどうかを決定します。XMLType インスタンスが XML Schema に基づく場合は 1、そうでない場合は 0（ゼロ）を戻します。
getSchemaURL() MEMBER FUNCTION getSchemaURL return varchar2 deterministic	XMLType インスタンスが XML Schema に基づく文書である場合は XMLType インスタンスに対応する XML Schema の URL を戻します。そうでない場合は NULL を戻します。
getRootElement() MEMBER FUNCTION getRootElement return varchar2 deterministic	XMLType インスタンスのルート要素を取得します。インスタンスがフラグメントの場合、NULL を戻します。
createSchemaBasedXML() MEMBER FUNCTION createSchemaBasedXML(schema IN varchar2 := NULL) return sys.XMLType deterministic	XML Schema に基づかない XML およびスキーマ URL から XML Schema に基づく XMLType インスタンスを作成します。 パラメータ： schema - スキーマ URL。NULL の場合、XMLType インスタンスにスキーマ URL を含める必要があります。
createNonSchemaBasedXML() MEMBER FUNCTION createNonSchemaBasedXML return XMLType deterministic	XML Schema に基づくインスタンスから XML Schema に基づかない XML 文書を作成します。
getNamespace() MEMBER FUNCTION getNamespace return varchar2 deterministic	インスタンスの最上位の要素の名前空間を戻します。入力がフラグメントまたは XML Schema に基づかないインスタンスである場合、NULL になります。

表 F-1　XMLType API （続き）

関数	説明
schemaValidate() MEMBER PROCEDURE schemaValidate	検証が行われていない場合、スキーマに対する XML インスタンスの検証を行います。XML Schema に基づかない文書の場合、エラーが発生します。検証に失敗した場合、エラーが発生します。検証が正常に終了した場合、文書の状態が、検証済に変更されます。
isSchemaValidated() MEMBER FUNCTION isSchemaValidated return NUMBER deterministic	XMLType がスキーマに対して検証済である場合、検証状態が戻されます。スキーマに対する検証が行われた場合は 1、そうでない場合は 0（ゼロ）を戻します。
setSchemaValidated() MEMBER PROCEDURE setSchemaValidated(flag IN BINARY_INTEGER := 1)	スキーマに対する検証が行われないようにするために、入力 XML インスタンスに検証状態を設定します。 パラメータ： フラグ - 0 = 未検証、1 = 検証済。デフォルト値は 1 です。
isSchemaValid() member function isSchemaValid(schurl IN VARCHAR2 := NULL, elem IN VARCHAR2 := NULL) return NUMBER deterministic	入力インスタンスが指定されたスキーマに準拠するかどうかを確認します。XML インスタンスの検証状態は変更されません。XML Schema URL が指定されておらず、XML 文書が XML Schema に基づく場合、XMLType インスタンスの自スキーマに準拠するかどうかが確認されます。 パラメータ： schurl - スキーマへの準拠の確認が行われる XML Schema URL。 elem - 検証の対象となる特定のスキーマの要素。複数の最上位の要素を定義する XML Schema が存在する場合、および特定の要素に準拠するかどうかを確認する必要がある場合に有効です。

PL/SQL DOM API for XMLType（DBMS_XMLDOM）

表 F-2 に、リリース 2（9.2.0.1）でサポートされている PL/SQL DOM API for XMLType（DBMS_XMLDOM）のメソッドを示します。これらのファンクションおよびプロシージャは、W3C の DOM 勧告に従ってグループ化されます。次の DBMS_XMLDOM のメソッドは、リリース 2（9.2.0.2）ではサポートされていません。

- hasFeature
- getVersion
- setVersion
- getCharset
- setCharset

- getStandalone
- setStandalone
- writeExternalDTDToFile
- writeExternalDTDToBuffer
- writeExternalDTDToClob

表 F-3 に、リリース 2（9.2.0.2）でサポートされているその他のメソッドを示します。

参照： 第 8 章「PL/SQL API for XMLType」を参照してください。

表 F-2 リリース 2（9.2.0.1）DBMS_XMLDOM メソッドの概要

グループ / メソッド	説明
ノード・メソッド	--
isNull()	ノードが NULL であるかどうかをテストします。
makeAttr()	ノードを属性にキャストします。
makeCDataSection()	ノードを CDATA セクションにキャストします。
makeCharacterData()	ノードを文字データにキャストします。
makeComment()	ノードをコメントにキャストします。
makeDocumentFragment()	ノードをドキュメント・フラグメントにキャストします。
makeDocumentType()	ノードをドキュメント・タイプにキャストします。
makeElement()	ノードを要素にキャストします。
makeEntity()	ノードをエンティティにキャストします。
makeEntityReference()	ノードを実体参照にキャストします。
makeNotation()	ノードを表記法にキャストします。
makeProcessingInstruction()	ノードを DOM 処理命令にキャストします。
makeText()	ノードを DOM テキストにキャストします。
makeDocument()	ノードを DOM 文書にキャストします。
writeToFile()	ファイルにノードのコンテンツを書き込みます。
writeToBuffer()	バッファにノードのコンテンツを書き込みます。
writeToClob()	CLOB にノードのコンテンツを書き込みます。
getNodeName()	ノードの名前を取得します。

表 F-2 リリース 2（9.2.0.1）DBMS_XMLDOM メソッドの概要 （続き）

グループ / メソッド	説明
getNodeValue()	ノードの値を取得します。
setNodeValue()	ノードの値を設定します。
getNodeType()	ノードのタイプを取得します。
getParentNode()	ノードの親ノードを取得します。
getChildNodes()	ノードの子ノードを取得します。
getFirstChild()	ノードの最初の子ノードを取得します。
getLastChild()	ノードの最後の子ノードを取得します。
getPreviousSibling()	ノードの以前の兄弟関係を取得します。
getNextSibling()	ノードの次の兄弟関係を取得します。
getAttributes()	ノードの属性を取得します。
getOwnerDocument()	ノードの所有者ドキュメントを取得します。
insertBefore()	参照先の子ノードの前に子を挿入します。
replaceChild()	古い子を新しい子ノードで置き換えます。
removeChild()	ノードから指定した子ノードを削除します。
appendChild()	ノードに新しい子ノードを追加します。
hasChildNodes()	ノードに子ノードが存在するかどうかをテストします。
cloneNode()	ノードを複製します。
名前付きノード・マップ・メソッド	--
isNull()	ノード・マップが NULL であるかどうかをテストします。
getNamedItem()	名前で指定された項目を取得します。
setNamedItem()	名前で指定された項目をマップに設定します。
removeNamedItem()	名前で指定された項目を削除します。
item()	マップ内の指定された索引の項目を取得します。
getLength()	マップ内の項目数を取得します。
ノード・リスト・メソッド	--
isNull()	ノード・リストが NULL であるかどうかをテストします。

表 F-2 リリース 2（9.2.0.1）DBMS_XMLDOM メソッドの概要 （続き）

グループ / メソッド	説明
item()	ノード・リスト内の指定された索引の項目を取得します。
getLength()	リスト内の項目数を取得します。
属性メソッド	--
isNull()	属性ノードが NULL であるかどうかをテストします。
makeNode()	属性をノードにキャストします。
getQualifiedName()	属性の修飾名を取得します。
getNamespace()	属性の名前空間 URI を取得します。
getLocalName()	属性のローカル名を取得します。
getExpandedName()	属性の拡張名を取得します。
getName()	属性の名前を取得します。
getSpecified()	属性が所有する要素内で指定されているかどうかをテストします。
getValue()	属性値を取得します。
setValue()	属性値を設定します。
CDATA セクション・メソッド	--
isNull()isNull()	CData セクションが NULL であるかどうかをテストします。
makeNode()makeNode()	CData セクションをノードにキャストします。
文字データ・メソッド	--
isNull()	文字データが NULL であるかどうかをテストします。
makeNode()	文字データをノードにキャストします。
getData()	ノードのデータを取得します。
setData()	ノードにデータを設定します。
getLength()	データの長さを取得します。
substringData()	データの部分文字列を取得します。
appendData()	指定されたデータをノード・データに追加します。
insertData()	データをノード内の指定されたオフセットに挿入します。

表 F-2 リリース 2（9.2.0.1）DBMS_XMLDOM メソッドの概要 （続き）

グループ / メソッド	説明
deleteData()	指定されたオフセットのデータを削除します。
replaceData()	指定されたオフセットのデータを置換します。
コメント・メソッド	--
isNull()	コメントが NULL であるかどうかをテストします。
makeNode()	コメントをノードにキャストします。
DOM インプリメンテーション・メソッド	--
isNull()	DOM インプリメンテーション・ノードが NULL であるかどうかをテストします。
hasFeature()	指定された機能を DOM が実装しているかどうかをテストします。（今回のリリースではサポートされていません。）
ドキュメント・フラグメント・メソッド	--
isNull()	ドキュメント・フラグメントが NULL であるかどうかをテストします。
makeNode()	ノードにドキュメント・フラグメントをキャストします。
ドキュメント・タイプ・メソッド	--
isNull()	ドキュメント・タイプが NULL であるかどうかをテストします。
makeNode()	ドキュメント・タイプをノードにキャストします。
findEntity()	ドキュメント・タイプ内で、指定されたエンティティを検索します。
findNotation()	ドキュメント・タイプ内で、指定された表記法を検索します。
getPublicId()	ドキュメント・タイプの公開識別子を取得します。
getSystemId()	ドキュメント・タイプのシステム識別子を取得します。
writeExternalDTDToFile()	DTD をファイルに書き込みます。
writeExternalDTDToBuffer()	DTD をバッファに書き込みます。
writeExternalDTDToClob()	DTD を CLOB に書き込みます。
getName()	ドキュメント・タイプの名前を取得します。

表 F-2 リリース 2（9.2.0.1）DBMS_XMLDOM メソッドの概要 （続き）

グループ / メソッド	説明
getEntities()	ドキュメント・タイプのエンティティのノード・マップを取得します。
getNotations()	ドキュメント・タイプの表記法のノード・マップを取得します。
要素メソッド	--
isNull()	要素が NULL であるかどうかをテストします。
makeNode()	要素をノードにキャストします。
getQualifiedName()	要素の修飾名を取得します。
getNamespace()	要素の名前空間 URI を取得します。
getLocalName()	要素のローカル名を取得します。
getExpandedName()	要素の拡張名を取得します。
getChildrenByTagName()	要素の子をタグ名で取得します。
getElementsByTagName()	サブツリーの要素を要素ごとに取得します。
resolveNamespacePrefix()	接頭辞を名前空間の URI に変換します。
getTagName()	要素のタグ名を取得します。
getAttribute()	名前で指定された属性ノードを取得します。
setAttribute()	名前で指定された属性を設定します。
removeAttribute()	名前で指定された属性を削除します。
getAttributeNode()	名前で指定された属性ノードを取得します。
setAttributeNode()	要素に属性ノードを設定します。
removeAttributeNode()	要素の属性ノードを削除します。
normalize()	要素の子テキストを正規化します。（今回のリリースではサポートされていません。）
エンティティ・メソッド	--
isNull()	エンティティが NULL であるかどうかをテストします。
makeNode()	エンティティをノードにキャストします。
getPublicId()	エンティティの公開識別子を取得します。
getSystemId()	エンティティのシステム識別子を取得します。
getNotationName()	エンティティの表記法名を取得します。

表 F-2 リリース 2（9.2.0.1）DBMS_XMLDOM メソッドの概要 （続き）

グループ / メソッド	説明
実態参照メソッド	--
isNull()	実体参照が NULL であるかどうかをテストします。
makeNode()	実体参照を NULL にキャストします。
表記法メソッド	--
isNull()	表記法が NULL であるかどうかをテストします。
makeNode()	表記法をノードにキャストします。
getPublicId()	表記法の公開識別子を取得します。
getSystemId()	表記法のシステム識別子を取得します。
処理命令メソッド	--
isNull()	処理命令が NULL であるかどうかをテストします。
makeNode()	処理命令をノードにキャストします。
getData()	処理命令のデータを取得します。
getTarget()	処理命令のターゲットを取得します。
setData()	処理命令のデータを設定します。
テキスト・メソッド	--
isNull()	テキストが NULL であるかどうかをテストします。
makeNode()	テキストをノードにキャストします。
splitText()	テキスト・ノードの内容を 2 つのテキスト・ノードに分割します。
文書メソッド	--
isNull()	文書が NULL であるかどうかをテストします。
makeNode()	文書をノードにキャストします。
newDOMDocument()	新しい文書を作成します。
freeDocument()	文書を解放します。
getVersion()	文書のバージョンを取得します。（今回のリリースではサポートされていません。）
setVersion()	文書のバージョンを設定します。（今回のリリースではサポートされていません。）

表 F-2 リリース 2（9.2.0.1）DBMS_XMLDOM メソッドの概要 （続き）

グループ / メソッド	説明
getCharset()	文書のキャラクタ・セットを取得します。（今回のリリースではサポートされていません。）
setCharset()	文書のキャラクタ・セットを設定します。（今回のリリースではサポートされていません。）
getStandalone()	文書が単独で指定されているかどうかを取得します。（今回のリリースではサポートされていません。）
setStandalone()	文書を単独で設定します。（今回のリリースではサポートされていません。）
writeToFile()	文書をファイルに書き込みます。
writeToBuffer()	文書をバッファに書き込みます。
writeToClob()	文書を CLOB に書き込みます。
writeExternalDTDToFile()	文書の DTD をファイルに書き込みます。（今回のリリースではサポートされていません。）
writeExternalDTDToBuffer()	文書の DTD をバッファに書き込みます。（今回のリリースではサポートされていません。）
writeExternalDTDToClob()	文書の DTD を CLOB に書き込みます。（今回のリリースではサポートされていません。）
getDoctype()	文書の DTD を取得します。
getImplementation()	DOM の実装を取得します。
getDocumentElement()	文書のルート要素を取得します。
createElement()	新しい要素を作成します。
createDocumentFragment()	新しい文書のフラグメントを作成します。
createTextNode()	テキスト・ノードを作成します。
createComment()	コメント・ノードを作成します。
createCDATASection()	CData セクション・ノードを作成します。
createProcessingInstruction()	処理命令を作成します。
createAttribute()	属性を作成します。
createEntityReference()	実体参照を作成します。
getElementsByTagName()	要素をタグ名で取得します。

表 F-3 リリース 2（9.2.0.2）に追加された DBMS_XMLDOM メソッド

メソッド	構文
createDocument	FUNCTION createDocument (namspaceURI IN VARCHAR2, qualifiedName IN VARCHAR2, doctype IN DOMType :=NULL) REURN DocDocument;
getPrefix	FUNCTION getPrefix(n DOMNode) RETURN VARCHAR2;
setPrefix	PROCEDURE setPrefix (n DOMNode) RETURN VARCHAR2;
hasAttributes	FUNCTION hasAttributes (n DOMNode) RETURN BOOLEAN;
getNamedItem	FUNCTION getNamedItem (nnm DOMNamedNodeMap, name IN VARCHAR2, ns IN VARCHAR2) RETURN DOMNode;
setNamedItem	FUNCTION getNamedItem (nnm DOMNamedNodeMap, arg IN DOMNode, ns IN VARCHAR2) RETURN DOMNode;
removeNamedItem	FUNCTION removeNamedItem (nnm DOMNamesNodeMap, name in VARCHAR2, ns IN VARCHAR2) RETURN DOMNode;
getOwnerElement	FUNCTION getOwnerElement (a DOMAttr) RETURN DOMElement;
getAttribute	FUNCTION getAttribute (elem DOMElement, name IN VARCHAR2, ns IN VARCHAR2) RETURN VARCHAR2;
hasAttribute	FUNCTION hasAttribute (elem DOMElement, name IN VARCHAR2) RETURN BOOLEAN;
hasAttribute	FUNCTION hasAttribute (elem DOMElement, name IN VARCHAR2, ns IN VARCHAR2) RETURN BOOLEAN;
setAttribute	PROCEDURE setAttribute (elem DOMElement, name IN VARCHAR2, newvalue IN VARCHAR2, ns IN VARCHAR2);
removeAttribute	PROCEDURE removeAttribute (elem DOMElement, name IN VARCHAR2, ns IN VARCHAR2);
getAttributeNode	FUNCTION getAttributeNode(elem DOMElement, name IN VARCHAR2, ns IN VARCHAR2) RETURN DOMAttr;
setAttributeNode	FUNCTION setAttributeNode(elem DOMElement, newAttr IN DOMAttr, ns IN VARCHAR2) RETURN DOMAttr;
createElement	FUNCTION createElement (doc DOMDocument, tagname IN VARCHAR2, ns IN VARCHAR2) RETURN DOMElement;
createAttribute	FUNCTION createAttribute (doc DOMDocument, name IN VARCHAR2, ns IN VARCHAR2) RETURN DOMAttr;

PL/SQL Parser for XMLType（DBMS_XMLPARSER）

PL/SQL Parser for XMLType（DBMS_XMLPARSER）を介して、XML 文書のコンテンツおよび構造にアクセスできます。

表 F-4 に、PL/SQL Parser for XMLType（DBMS_XMLPARSER）のファンクションおよびプロシージャを示します。

参照： 第 8 章「PL/SQL API for XMLType」を参照してください。

表 F-4　DBMS_XMLPARSER のファンクションおよびプロシージャ

ファンクションおよびプロシージャ	説明
parse()	指定した URL ファイルに格納された XML を解析します。
newParser()	新しいパーサーのインスタンスを戻します。
parseBuffer()	指定したバッファに格納された XML を解析します。
parseClob()	指定した CLOB に格納された XML を解析します。
parseDTD()	指定した URL ファイルに格納された DTD を解析します。
parseDTDBuffer()	指定したバッファに格納された DTD を解析します。
parseDTDClob()	指定した CLOB に格納された DTD を解析します。
setBaseDir()	関連 URL の解決に使用されるベース・ディレクトリを設定します。
showWarnings()	警告のオンまたはオフを切り替えます。
setErrorLog()	エラーが指定ファイルに送信されるように設定します。
setPreserveWhitespace()	空白保存モードを設定します。
setValidationMode()	検証モードを設定します。
getValidationMode()	検証モードを戻します。
setDoctype()	DTD を設定します。
getDoctype()	DTD Parser を取得します。
getDocument()	DOM 文書を取得します。
freeParser()	パーサー・オブジェクトを解放します。
getReleaseVersion()	Oracle XML Parser for PL/SQL のリリース・バージョンを戻します。

PL/SQL XSLT Processor for XMLType（DBMS_XSLPROCESSOR）

XSLT プロセッサの PL/SQL 実装は、W3C の XSLT 草案（WD-xslt-19990813 改訂）に準拠します。

表 F-5 に、PL/SQL XSLT Processor for XMLType（DBMS_XSLPROCESSOR）のファンクションおよびプロシージャの概要を示します。

参照： 第 8 章「PL/SQL API for XMLType」を参照してください。

表 F-5 PL/SQL XSLT Processor for XMLType（DBMS_XSLPROCESSOR）のファンクション

ファンクションおよびプロシージャ	説明
newProcessor()	新しいプロセッサ・インスタンスを戻します。
processXSL()	入力 XML 文書を変換します。
showWarnings()	警告のオンまたはオフを切り替えます。
setErrorLog()	エラーが指定ファイルに送信されるように設定します。
newStylesheet()	指定した入力および参照 URL を使用して、新しいスタイルシートを作成します。
transformNode()	指定したスタイルシートを使用して、DOM ツリー内のノードを変換します。
selectNodes()	指定したパターンに一致する DOM ツリーのノードを選択します。
selectSingleNodes()	指定したパターンに一致するツリーの最初のノードを選択します。
valueOf()	指定したパターンに一致するツリーの最初のノード値を取り出します。
setParam()	スタイルシートに最上位のパラメータを設定します。
removeParam()	最上位のスタイルシートのパラメータを削除します。
resetParams()	最上位のスタイルシートのパラメータをリセットします。
freeStylesheet()	スタイルシート・オブジェクトを解放します。
freeProcessor()	プロセッサ・オブジェクトを解放します。

DBMS_XMLSCHEMA

このパッケージは、Oracle XML DB のインストール時に dbmsxsch.sql によって作成されます。XML Schema を登録および削除するためのプロシージャを提供します。表 F-6 に、DBMS_XMLSCHEMA のファンクションおよびプロシージャの概要を示します。

> **参照：** 第 5 章「XMLType の構造化されたマッピング」を参照してください。

表 F-6 DBMS_XMLSCHEMA のファンクションおよびプロシージャ

定数	説明
registerSchema() procedure registerSchema(schemaURL IN VARCHAR2, schemaDoc IN VARCHAR2, local IN BOOLEAN := TRUE, genTypes IN BOOLEAN := TRUE, genbean IN BOOLEAN := FALSE, genTables IN BOOLEAN := TRUE, force IN BOOLEAN := FALSE, owner IN VARCHAR2 := null) procedure registerSchema(schemaURL IN VARCHAR2, schemaDoc IN CLOB, local IN BOOLEAN := TRUE, genTypes IN BOOLEAN := TRUE, genbean IN BOOLEAN := FASLE, force IN BOOLEAN := FALSE, owner IN VARCHAR2 := null) procedure registerSchema(schemaURL IN VARCHAR2, schemaDoc IN BFILE, local IN BOOLEAN := TRUE, genTypes IN BOOLEAN := TRUE, genbean IN BOOLEAN := FALSE, force IN BOOLEAN := FALSE, owner IN VARCHAR2 := null) procedure registerSchema(schemaURL IN VARCHAR2, schemaDoc IN SYS.XMLType, local IN BOOLEAN := TRUE, genTypes IN BOOLEAN := TRUE, genbean IN BOOLEAN := FALSE, force IN BOOLEAN := FALSE, owner IN VARCHAR2 := null) procedure registerSchema(schemaURL IN VARCHAR2, schemaDoc IN SYS.URIType, local IN BOOLEAN := TRUE, genTypes IN BOOLEAN := TRUE, genbean IN BOOLEAN := FALSE, force IN BOOLEAN := FALSE, owner IN VARCHAR2 := null)	Oracle XML DB によって使用される指定された XML Schema を登録します。このスキーマを使用して、このスキーマに準拠するドキュメントを格納できます。 パラメータ : schemaURL - スキーマ・ドキュメントを一意に識別する URL。この値を使用して、XML DB 階層内でのスキーマ・ドキュメントのパス名を導出します。 schemaDoc - 妥当な XML Schema ドキュメント local - ローカル・スキーマであるか、またはグローバル・スキーマであるか。デフォルトでは、すべてのスキーマはローカル・スキーマとして /sys/schemas/<username>/... に登録されます。グローバル・スキーマとして登録された場合、/sys/schemas/PUBLIC/.... に追加されます。グローバル・スキーマとして登録できるようにするには、前述のディレクトリに権限を書き込む必要があります。 genTypes - スキーマ・コンパイラによってオブジェクト型を生成する必要があるかどうか。デフォルトでは、TRUE です。 genbean - スキーマ・コンパイラによって JavaBean を生成する必要があるかどうか。デフォルトでは、FALSE です。 genTables - スキーマ・コンパイラによってデフォルト表を生成する必要があるかどうか。デフォルトでは、TRUE です。 force - このパラメータを TRUE に設定すると、スキーマの登録時にエラーが発生しないようにすることができます。エラーの場合、無効な XML Schema オブジェクトが作成されます。デフォルトでは、このパラメータの値は FALSE です。 owner - XML Schema オブジェクトを所有するデータベース・ユーザーの名前を指定します。デフォルトでは、XML Schema を登録するユーザーが XML Schema オブジェクトを所有します。このパラメータを使用して、異なるデータベース・ユーザーが XML Schema を所有するように登録できます。

表 F-6 DBMS_XMLSCHEMA のファンクションおよびプロシージャ （続き）

定数	説明
registerURI() procedure registerURI(schemaURL IN varchar2, schemaDocURI IN varchar2, local IN BOOLEAN := TRUE, genTypes IN BOOLEAN := TRUE, genbean IN BOOLEAN := FALSE, genTables IN BOOLEAN := TRUE, force IN BOOLEAN := FALSE, owner IN VARCHAR2 := null)	URI の名前で指定された XML Schema を登録します。
deleteSchema() procedure deleteSchema(schemaURL IN varchar2, delete_option IN pls_integer := DELETE_RESTRICT)	Oracle XML DB から XML Schema を削除します。
generateBean() procedure generateBean(schemaURL IN varchar2)	登録された XML Schema に対応する JavaBean コードを生成します。
compileSchema() procedure compileSchema(schemaURL IN varchar2)	登録済の XML Schema を再コンパイルします。無効なスキーマを有効な状態にするために有効です。
generateSchema() function generateSchemas(schemaName IN varchar2, typeName IN varchar2, elementName IN varchar2 := NULL, schemaURL IN varchar2 := NULL, annotate IN BOOLEAN := TRUE, embedColl IN BOOLEAN := TRUE) return sys.XMLSequenceType function generateSchema(schemaName IN varchar2, typeName IN varchar2, elementName IN varchar2 := NULL, recurse IN BOOLEAN := TRUE, annotate IN BOOLEAN := TRUE, embedColl IN BOOLEAN := TRUE) return sys.XMLType	Oracle の型名から XML Schema を生成します。

DBMS_XMLSCHEMA 定数

- DELETE_RESTRICT, CONSTANT NUMBER := 1
- DELETE_INVALIDATE,CONSTANT NUMBER := 2

- DELETE_CASCADE, CONSTANT NUMBER := 3
- DELETE_CASCADE_FORCE, CONSTANT NUMBER := 4

Oracle XML DB の XML Schema カタログ・ビュー

表 F-7 に、Oracle XML DB の XML Schema カタログ・ビューを示します。

表 F-7 Oracle XML DB: XML Schema カタログ・ビュー

スキーマ	説明
USER_XML_SCHEMAS	ユーザーが所有するすべての登録済の XML Schema を示します。
ALL_XML_SCHEMAS	現行のユーザーによって使用可能なすべての登録済の XML Schema を示します。
DBA_XML_SCHEMAS	Oracle XML DB に存在するすべての登録済の XML Schema を示します。
DBA_XML_TABLES	システムに存在するすべての XMLType 表を示します。
USER_XML_TABLES	現行のユーザーが所有するすべての XMLType 表を示します。
ALL_XML_TABLES	現行のユーザーによって使用可能なすべての XMLType 表を示します。
DBA_XML_TAB_COLS	システムに存在するすべての XMLType 表の列を示します。
USER_XML_TAB_COLS	現行のユーザーが所有するすべての XMLType 表の列を示します。
ALL_XML_TAB_COLS	現行のユーザーによって使用可能なすべての XMLType 表の列を示します。
DBA_XML_VIEWS	システムに存在するすべての XMLType ビューを示します。
USER_XML_VIEWS	現行のユーザーが所有するすべての XMLType ビューを示します。
ALL_XML_VIEWS	現行のユーザーによって使用可能なすべての XMLType ビューを示します。
DBA_XML_VIEW_COLS	システムに存在するすべての XMLType ビューの列を示します。
USER_XML_VIEW_COLS	現行のユーザーが所有するすべての XMLType ビューの列を示します。

表 F-7 Oracle XML DB: XML Schema カタログ・ビュー（続き）

スキーマ	説明
ALL_XML_VIEW_COLS	現行のユーザーによって使用可能なすべての XMLType ビューの列を示します。

Resource API for PL/SQL（DBMS_XDB）

Resource API for PL/SQL（DBMS_XDB）という PL/SQL パッケージは、次の Oracle XML DB タスクのためのファンクションを提供します。

- Oracle XML DB 階層のリソース管理。これらのファンクションは、リソース・ビューで提供される機能を補足します。
- セキュリティ管理のための Oracle XML DB のアクセス制御リスト（ACL）。ACL ベースのセキュリティ・メカニズムを次のいずれかの方法で使用できます。
  - 階層内 ACL: ACL は、Oracle XML DB Resource API を介して格納されます。
  - メモリー内 ACL: ACL は、Oracle XML DB の外部に格納されます。

  これらのメソッドのいくつかは、Oracle XML DB のリソースおよび任意のデータベース・オブジェクトの両方に対して使用できます。`AclCheckPrivileges()` を使用すると、データベース・ユーザーが、Oracle XML DB の階層にオブジェクトを格納することなく、Oracle XML DB の ACL ベースのセキュリティ・メカニズムにアクセスできるようになります。
- Oracle XML DB の構成セッションの管理。
- 階層索引の再作成。

表 F-8 に、DBMS_XDB のファンクションおよびプロシージャの概要を示します。

> **参照：** 第 16 章「Oracle XML DB Resource API for PL/SQL（DBMS_XDB）」を参照してください。

表 F-8 DBMS_XDB のファンクションおよびプロシージャ

ファンクションおよびプロシージャ	説明
getAclDocument() FUNCTION getAclDocument(abspath IN VARCHAR2) RETURN sys.xmltype	指定されたパス名を使用して、リソースを保護する ACL ドキュメントを取得します。
getPrivileges() FUNCTION getPrivileges(res_path IN VARCHAR2) RETURN sys.xmltype	指定された XML DB リソースに対して現行のユーザーに付与されたすべての権限を取得します。

表 F-8 DBMS_XDB のファンクションおよびプロシージャ（続き）

ファンクションおよびプロシージャ	説明
changePrivileges() FUNCTION changePrivileges(res_path IN VARCHAR2, ace IN xmltype) RETURN pls_integer	指定されたリソースの ACL に、指定された ACE を追加します。
checkPrivileges() FUNCTION checkPrivileges(res_path IN VARCHAR2, privs IN xmltype) RETURN pls_integer	指定された XML DB リソースに対して現行のユーザーに付与されたすべてのアクセス権限を確認します。
setacl() PROCEDURE setacl(res_path IN VARCHAR2, acl_path IN VARCHAR2)	指定された XML DB リソースに ACL が指定されるように設定します。
AclCheckPrivileges() FUNCTION AclCheckPrivileges(acl_path IN VARCHAR2, owner IN VARCHAR2, privs IN xmltype) RETURN pls_integer	所有者が「owner」パラメータによって指定されているリソースに、指定された ACL ドキュメントによって、現行のユーザーに付与されているアクセス権限を確認します。
LockResource() FUNCTION LockResource(path IN VARCHAR2, depthzero IN BOOLEAN, shared IN boolean) RETURN BOOLEAN	指定されたリソースのパスを使用して、WebDAV 形式のロックを取得します。
GetLockToken() PROCEDURE GetLockToken(path IN VARCHAR2, locktoken OUT VARCHAR2)	指定されたリソースのパスを使用して、現行のユーザーに対するリソースのロック・トークンを戻します。
UnlockResource() FUNCTION UnlockResource(path IN VARCHAR2, deltoken IN VARCHAR2) RETURN BOOLEAN	指定されたロック・トークンおよびリソースのパスを使用して、リソースのロックを解除します。

表 F-8 DBMS_XDB のファンクションおよびプロシージャ（続き）

ファンクションおよびプロシージャ	説明
CreateResource() FUNCTION CreateResource(path IN VARCHAR2,data IN VARCHAR2) RETURN BOOLEAN FUNCTION CreateResource(path IN VARCHAR2, data IN SYS.XMLTYPE) RETURN BOOLEAN FUNCTION CreateResource(path IN VARCHAR2, datarow IN REF SYS.XMLTYPE) RETURN BOOLEAN FUNCTION CreateResource(path IN VARCHAR2, data IN CLOB) RETURN BOOLEAN FUNCTION CreateResource(path IN VARCHAR2, data IN BFILE) RETURN BOOLEAN FUNCTION CreateResource(abspath IN VARCHAR2, data IN BFILE, csid IN NUMBER := 0) RETURN BOOLEAN; FUNCTION CreateResource(abspath IN VARCHAR2, data IN BLOB, csid IN NUMBER := 0) RETURN BOOLEAN;	新しいリソースを作成します。
CreateFolder() FUNCTION CreateFolder(path IN VARCHAR2) RETURN BOOLEAN	階層に新しいフォルダ・リソースを作成します。
DeleteResource() PROCEDURE DeleteResource(path IN VARCHAR2)	階層からリソースを削除します。
Link() PROCEDURE Link(srcpath IN VARCHAR2, linkfolder IN VARCHAR2, linkname IN VARCHAR2)	既存のリソースへのリンクを作成します。
CFG_Refresh() PROCEDURE CFG_Refresh	セッションの構成情報を最新の構成にリフレッシュします。

表 F-8 DBMS_XDB のファンクションおよびプロシージャ（続き）

ファンクションおよびプロシージャ	説明
CFG_Get() FUNCTION CFG_Get RETURN SYS.XMLType	セッションの構成情報を取得します。
CFG_Update() PROCEDURE CFG_Update(xdbconfig IN SYS.XMLTYPE)	構成情報を更新します。

DBMS_XMLGEN

PL/SQL パッケージの DBMS_XMLGEN は、SQL 問合せ結果を正規の XML 形式に変換します。このパッケージによって、任意の SQL 問合せが入力され、XML に変換された後、結果が CLOB として戻されます。DBMS_XMLGEN は、C で作成されてカーネルでコンパイルされることを除き、DBMS_XMLQUERY に類似しています。このパッケージは、データベースのみで実行できます。

表 F-9 に、DBMS_XMLGEN のファンクションおよびプロシージャの概要を示します。

> **参照：** 第 10 章「データベースからの XML データの生成」を参照してください。

表 F-9 DBMS_XMLGEN のファンクションおよびプロシージャ

ファンクションおよびプロシージャ	説明
newContext()	新しいコンテキスト・ハンドルを作成します。
setRowTag()	結果の各行を囲む要素名を設定します。デフォルトのタグは ROW です。
setRowSetTag ()	結果全体を囲む要素名を設定します。デフォルトのタグは ROWSET です。
getXML()	XML 文書を取得します。
getNumRowsProcessed()	getXML を最後にコールしたときに処理された SQL 行の数を取得します。
setMaxRows()	1 度にフェッチされる行の最大数を設定します。
setSkipRows()	XML を生成する前に 1 度にスキップする行の数を設定します。デフォルトは、0（ゼロ）です。
setConvertSpecialChars()	非 XML キャラクタである $ などの特殊文字をエスケープ済の表現に変換する必要があるかどうか。デフォルトでは、変換が実行されます。

表 F-9 DBMS_XMLGEN のファンクションおよびプロシージャ（続き）

ファンクションおよびプロシージャ	説明
convert()	XML を、エスケープ済またはエスケープされていない同等の XML 文書に変換します。
useItemTagsForColl()	コレクション要素に対して _ITEM タグが追加されたコレクション列名を強制的に使用します。デフォルトでは、コレクションのベースの要素に対して基礎となるオブジェクト型の名前が設定されます。
restartQUERY()	問合せを再度開始し、フェッチを最初から開始します。
closeContext()	コンテキストをクローズし、すべてのリソースを解放します。

RESOURCE_VIEW および PATH_VIEW

Oracle XML DB の RESOURCE_VIEW および PATH_VIEW は、Oracle XML DB Repository に格納されたデータへの SQL ベースのアクセスのためのメカニズムを提供します。FTP や WebDAV API などのプロトコルを介して Oracle XML DB Repository に格納されたデータには、RESOURCE_VIEW および PATH_VIEW を介して SQL でアクセスできます。

Oracle XML DB Resource API for PL/SQL は、RESOURCE_VIEW、PATH_VIEW およびいくつかの PL/SQL パッケージに基づいています。この API は、問合せおよび DML 機能を提供します。PATH_VIEW にはリポジトリの一意の各パスに対して 1 つの行が存在し、REOURCE_VIEW にはリポジトリの各リソースに対して 1 つの行が存在します。

表 F-10 に、Oracle XML DB Resource API for PL/SQL の演算子の概要を示します。

> **参照：** 第 15 章「RESOURCE_VIEW および PATH_VIEW」を参照してください。

表 F-10 RESOURCE_VIEW および PATH_VIEW の演算子

演算子	説明
UNDER_PATH INTEGER UNDER_PATH(resource_column, pathname) INTEGER UNDER_PATH(resource_column, depth, pathname) INTEGER UNDER_PATH(resource_column,pathname,correlation) INTEGER UNDER_PATH(resource_column, depth,pathname, correlation)	Oracle XML DB の階層索引を使用して、特定のパスのサブパスを戻します。 パラメータ： resource_column - path_view または resource_view の「resource」列の名前または別名。 pathname - 解決するパス名。 depth - 検索する深さの最大数。0（ゼロ）未満の深さは 0（ゼロ）として処理されます。 correlation - UNDER_PATH 演算子を補助演算子（PATH および DEPTH）に相関させるために使用可能な整数。
EQUALS_PATH EQUALS_PATH INTEGER EQUALS_PATH(resource_column, pathname)	指定されたパス名を持つリソースを検索します。
PATH PATH VARCHAR2 PATH(correlation)	指定されたパス名属性の下にあるリソースの絶対パス名を戻します。
DEPTH DEPTH INTEGER DEPTH(correlation)	指定された起動パスの下にあるリソースのフォルダ階層を戻します。

DBMS_XDB_VERSION

DBMS_XDB の他に DBMS_XDB_VERSION も、Oracle XML DB Versioning API を実装します。

表 F-11 に、DBMS_XDB_VERSION のファンクションおよびプロシージャの概要を示します。

参照： 第 14 章「Oracle XML DB Versioning」を参照してください。

表 F-11 DBMS_XDB_VERSION のファンクションおよびプロシージャ

ファンクションおよびプロシージャ	説明
MakeVersioned() FUNCTION MakeVersioned(pathname VARCHAR2) RETURN DBMS_XDB_VERSION.resid_type	パス名が指定された通常のリソースをバージョン管理されたリソースに変換します。

表 F-11 DBMS_XDB_VERSION のファンクションおよびプロシージャ （続き）

ファンクションおよびプロシージャ	説明
Checkout() PROCEDURE Checkout(pathname VARCHAR2)	VCR を更新または削除する前に、その VCR をチェックアウトします。
Checkin() FUNCTION Checkin(pathname VARCHAR2) RETURN DBMS_XDB_VERSION.resid_type	チェックアウトされた VCR をチェックインし、新しく作成されたバージョンのリソース ID を戻します。
Uncheckout() FUNCTION Uncheckout(pathname VARCHAR2) RETURN DBMS_XDB_VERSION.resid_type	チェックアウトされたリソースをチェックインし、リソースがチェックアウトされる前のバージョンのリソース ID を戻します。
GetPredecessors() FUNCTION GetPredecessors(pathname VARCHAR2) RETURN DBMS_XDB_VERSION. resid_list_type	パス名で先行リソースのリストを取得します。
GetPredsByResId() FUNCTION GetPredsByResId(resid DBMS_XDB.VERSION.resid_type) RETURN DBMS_XDB_VERSION. resid_list_type	リソース ID で先行リソースのリストを取得します。
GetResourceByResId() FUNCTION GetResourceByResId(DBMS_XDB_VERSION.resid DBMS_XDB_VERSION.resid_type) RETURN XMLType	指定されたリソースのオブジェクト ID を使用して、`XMLType` としてリソースを取得します。
GetSuccessors() FUNCTION GetSuccessors(pathname VARCHAR2) RETURN DBMS_XDB_VERSION. resid_list_type	パス名で後続リソースのリストを取得します。
GetSuccsByResId() FUNCTION GetSuccsByResId(resid DBMS_XDB_VERSION.resid_type) RETURN DBMS_XDB_VERSION. resid_list_type	リソース ID で後続リソースのリストを取得します。

DBMS_XDBT

DBMS_XDBT を使用して、Oracle XML DB Repository の階層に Oracle Text の ConText 索引を設定できます。DBMS_XDBT によって、デフォルトのプリファレンスおよび Oracle Text 索引が作成されます。また、ConText 索引の自動同期が設定されます。

DBMS_XDBT には、ConText 索引の構成設定を説明する変数が含まれます。これらの変数は、インストールに必要な場合がある基本的なカスタマイズに対応することを目的としています。ただし、これは完全なセットではありません。

次のタスクに対して DBMS_XDBT を使用します。

- 適切な構成を設定するためのパッケージのカスタマイズ
- dropPreferences() を使用した既存の索引プリファレンスの削除
- createPreferences() を使用した新しい索引プリファレンスの作成
- createIndex() を使用した ConText 索引の作成
- configureAutoSync() を使用した ConText 索引の自動同期の設定

表 F-12 に、DBMS_XDBT のファンクションおよびプロシージャの概要を示します。

> **参照：** 『Oracle9*i* XML API リファレンス - XDK および Oracle XML DB』を参照してください。

表 F-12 DBMS_XDBT のファンクションおよびプロシージャ

ファンクションおよびプロシージャ	説明
dropPreferences()	すべての既存のプリファレンスを削除します。
createPreferences()	XML DB 階層の ConText 索引に必要なプリファレンスを作成します。
createDatastorePref()	ConText 索引に対する USER データストア・プリファレンスを作成します。
createFilterPref()	ConText 索引に対するフィルタ・プリファレンスを作成します。
createLexerPref()	ConText 索引に対するレクサー・プリファレンスを作成します。
createWordlistPref()	ConText 索引に対するストップリストを作成します。
createStoplistPref()	ConText 索引に対するセクション・グループを作成します。
createStoragePref()	ConText 索引に対するワードリスト・プリファレンスを作成します。

表 F-12 DBMS_XDBT のファンクションおよびプロシージャ（続き）

ファンクションおよびプロシージャ	説明
createSectiongroupPref()	ConText 索引に対する格納プリファレンスを作成します。
createIndex()	XML DB 階層に ConText 索引を作成します。
configureAutoSync()	自動メンテナンス（SYNC）が実行されるように ConText 索引を構成します。

G

設定スクリプトの例および Oracle XML DB によって提供される XML Schema

この付録では、第 3 章「Oracle XML DB の使用」の例で使用できるいくつかの設定スクリプトの例について説明します。また、リソース・ビューおよびパス・ビューの構造、および Oracle XML DB によって提供される XML Schema を示します。

- 設定スクリプトの例
- RESOURCE_VIEW および PATH_VIEW のデータベースおよび XML Schema
- XDBResource.xsd: Oracle XML DB リソースを表すための XML Schema
- acl.xsd: Oracle XML DB の ACL を表すための XML Schema
- xdbconfig.xsd: Oracle XML DB を構成するための XML Schema

設定スクリプトの例

次の項では、第 3 章「Oracle XML DB の使用」の例で使用された設定スクリプトおよびサンプル・ファイルを示します。

第 3 章の設定スクリプトの例 : ユーザーおよびディレクトリの作成

```
set echo on
connect / as sysdba
drop directory DIR;
drop user &1 cascade;
create user &1 identified by &2;
grant create any directory, drop any directory to &1;
grant connect, resource to &1;
connect &1/&2
create or replace function getFileContent(file bfile)
return CLOB deterministic
is
   charContent     CLOB := ' ';
   targetFile      bfile;
   warning         number;
begin
  targetFile := file;
  DBMS_LOB.fileopen(targetFile, DBMS_LOB.file_readonly);
  DBMS_LOB.loadfromFile(charContent,targetFile,
  DBMS_LOB.getLength(targetFile),1,1);
  DBMS_LOB.fileclose(targetFile);
  return charContent;
end;
/
show errors;
drop directory DIR;
create directory DIR as '&3';
create or replace function getDocument(filename varchar2)
return CLOB deterministic
is
   file            bfile := bfilename('DIR',filename);
   charContent     CLOB := ' ';
   targetFile      bfile;
   warning         number;
begin
  targetFile := file;
  DBMS_LOB.fileopen(targetFile, DBMS_LOB.file_readonly);
  DBMS_LOB.loadfromFile(charContent,targetFile,
  DBMS_LOB.getLength(targetFile),1,1);
  DBMS_LOB.fileclose(targetFile);
  return charContent;
```

```
end;
/
show errors
declare
  result boolean;
begin
  result := dbms_xdb.createfolder('/public/&4');
end;
/
commit;
quit
```

第 3 章の設定スクリプトの例 : 権限の付与および表の作成

このスクリプトでは、適切な権限の付与、XMLType 列が含まれる表の作成、XMLType 表の作成、および表の挿入、問合せおよび更新が行われます。

```
set echo on
connect scott/tiger
grant all on emp to &1;
connect &1/&2
--
-- Table Creation Examples
--
create table EXAMPLE1
(
 KEYVALUE  varchar2(10)  primary key,
     XMLCOLUMN xmltype
);
create table XMLTABLE of XMLType;
--
-- Insert Example
--
describe getDocument;
insert into XMLTABLE
values (xmltype(getDocument('purchaseorder.xml')))
/
commit
/
--
-- Valid existsNode operations
--
select existsNode(value(X),'/PurchaseOrder/Reference')
from XMLTABLE X
/
select existsNode(value(X),
                  '/PurchaseOrder[Reference="ADAMS-20011127121040988PST"]')
```

```
from XMLTABLE X
/
select existsNode(value(X),
          '/PurchaseOrder/LineItems/LineItem[2]/Part[@Id="037429135020"]')
from XMLTABLE X
/
select existsNode(value(X),
          '/PurchaseOrder/LineItems/LineItem[Description="8 1/2"]')
from XMLTABLE X
/
--
-- Invalid existsNode() operations
--
select existsNode(value(X),'/PurchaseOrder/UserName')
from XMLTABLE X
/
select existsNode(value(X),
                  '/PurchaseOrder[Reference="ADAMS-XXXXXXXXXXXXXXXXXXXX"]')
from XMLTABLE X
/
select existsNode(value(X),
          '/PurchaseOrder/LineItems/LineItem[3]/Part[@Id="037429135020"]')
from XMLTABLE X
/
select existsNode(value(X),
          '/PurchaseOrder/LineItems/LineItem[Description="Snow White"]')
from XMLTABLE X
/
--
-- existsNode() in where clause examples
--
select count(*)
from XMLTABLE x
where existsNode(value(x),'/PurchaseOrder[User="ADAMS"]') = 1
/
delete from XMLTABLE x
where existsNode(value(x),'/PurchaseOrder[User="ADAMS"]') = 1
/
commit
/
--
-- Reload the Sample Document.
--
insert into XMLTABLE
values (xmltype(getDocument('purchaseorder.xml')))
/
commit
```

```
/
--
-- Valid extractValue() operiations
--
select extractValue(value(x),'/PurchaseOrder/Reference')
from XMLTABLE X
/
select extractValue(value(x),
       '/PurchaseOrder/LineItems/LineItem[2]/Part/@Id')
from XMLTABLE X
/
--
-- Invalid extractValue() operations
--
select extractValue(value(X),
                   '/PurchaseOrder/LineItems/LineItem/Description')
from XMLTABLE X
/
select extractValue(value(X),
                   '/PurchaseOrder/LineItems/LineItem[1]')
from XMLTABLE X
/
select extractValue(value(x),'/PurchaseOrder/Reference')
from XMLTABLE X, SCOTT.EMP
where extractValue(value(x),'/PurchaseOrder/User') = EMP.ENAME
and EMP.EMPNO = 7876
/
--
-- extract() operations
--
set long 10000
select extract(value(X),
                   '/PurchaseOrder/LineItems/LineItem/Description')
from XMLTABLE X
/
select extract(value(X),
                   '/PurchaseOrder/LineItems/LineItem[1]')
from XMLTABLE X
/
set long 10000
set feedback on
select extractValue(value(t),'/Description')
from XMLTABLE X,
table ( xmlsequence (
        extract(value(X),
                '/PurchaseOrder/LineItems/LineItem/Description')
        )
```

```
) t
/
update XMLTABLE t
set value(t) = updateXML(value(t),
'/PurchaseOrder/Reference/text()',
'MILLER-20020331120000000PST')
where existsNode(value(t),
                 '/PurchaseOrder[Reference="ADAMS-20011127121040988PST"]') = 1
/
select value(t)
from XMLTABLE t
/
update XMLTABLE t
set value(t) =
updateXML(value(t),
          '/PurchaseOrder/LineItems/LineItem[2]',
          xmltype('<LineItem ItemNumber="4">
                      <Description>Andrei Rublev</Description>
                      <Part Id="715515009928" UnitPrice="39.95"
                         Quantity="2"/>
                   </LineItem>'
           )
)
where existsNode(value(t),
                 '/PurchaseOrder[Reference="MILLER-20020331120000000PST"]'
) = 1
/
select value(t)
from XMLTABLE t
where existsNode(value(t),
                 '/PurchaseOrder[Reference="MILLER-20020331120000000PST"]'
) = 1
/
select value(t).transform(xmltype(getDocument('purchaseOrder.xsl')))
from XMLTABLE t
where existsNode(value(t),
                 '/PurchaseOrder[Reference="MILLER-20020331120000000PST"]'
) = 1
/
begin
  dbms_xmlschema.registerSchema(
                 'http://www.oracle.com/xsd/purchaseOrder.xsd',
                 getDocument('purchaseOrder.xsd'),
                 TRUE, TRUE, FALSE, FALSE
       );
end;
/
```

```
create table XML_PURCHASEORDER of XMLType
XMLSCHEMA "http://www.oracle.com/xsd/purchaseOrder.xsd"
ELEMENT "PurchaseOrder"
/
describe XML_PURCHASEORDER
insert into XML_PURCHASEORDER
values (xmltype(getDocument('Invoice.xml')))
/
alter table XML_PURCHASEORDER
add constraint VALID_PURCHASEORDER
check (XMLIsValid(sys_nc_rowinfo$)=1)
/
insert into XML_PURCHASEORDER
values (xmltype(getDocument('InvalidPurchaseOrder.xml')))
/
alter table XML_PURCHASEORDER
drop constraint VALID_PURCHASEORDER
/
create trigger VALIDATE_PURCHASEORDER
before insert on XML_PURCHASEORDER
for each row
declare
  XMLDATA xmltype;
begin
   XMLDATA := :new.sys_nc_rowinfo$;
   xmltype.schemavalidate(XMLDATA);
end;
/
insert into XML_PURCHASEORDER
values (xmltype(getDocument('InvalidPurchaseOrder.xml')))
/
drop table XML_PURCHASEORDER;
begin
  dbms_xmlSchema.deleteSchema('http://www.oracle.com/xsd/purchaseOrder.xsd',4);
end;
/
begin
  dbms_xmlschema.registerSchema(
                'http://www.oracle.com/xsd/purchaseOrder.xsd',
                getDocument('purchaseOrder1.xsd'),
                TRUE, TRUE, FALSE, FALSE
       );
end;
/
describe XML_PURCHASEORDER_TYPE
drop table XML_PURCHASEORDER;
begin
```

```
  dbms_xmlSchema.deleteSchema('http://www.oracle.com/xsd/purchaseOrder.xsd',4);
end;
/
begin
  dbms_xmlschema.registerSchema(
                'http://www.oracle.com/xsd/purchaseOrder.xsd',
                getDocument('purchaseOrder2.xsd'),
                TRUE, TRUE, FALSE, FALSE
       );
end;
/
describe XML_PURCHASEORDER_TYPE
quit
```

ファイルのロード

```
set echo on
connect &1/&2
declare
  result boolean;
begin
  result := dbms_xdb.createResource('/public/&3/&4',
                                    getFileContent(bfilename('DIR','&4')));
end;
/
commit;
quit
```

第 3 章のスクリプトの例 : invoice.xml

```
<Invoice
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="http://www.oracle.com/xsd/purchaseOrder.xsd">
  <Reference>ADAMS-20011127121040988PST</Reference>
  <Actions>
    <Action>
      <User>SCOTT</User>
      <Date xsi:nil="true"/>
    </Action>
  </Actions>
  <Reject/>
  <Requestor>Julie P. Adams</Requestor>
  <CostCenter>R20</CostCenter>
  <ShippingInstructions>
    <name>Julie P. Adams</name>
    <address>300 Oracle Parkway, Redwood Shores, CA 94065</address>
```

```
    <telephone>650 506 7300</telephone>
  </ShippingInstructions>
  <SpecialInstructions>Ground</SpecialInstructions>
  <LineItems>
    <LineItem ItemNumber="1">
      <Description>The Ruling Class</Description>
      <Part Id="715515012423" UnitPrice="39.95" Quantity="2"/>
    </LineItem>
    <LineItem ItemNumber="2">
      <Description>Diabolique</Description>
      <Part Id="037429135020" UnitPrice="29.95" Quantity="3"/>
    </LineItem>
    <LineItem ItemNumber="3">
      <Description>8 1/2</Description>
      <Part Id="037429135624" UnitPrice="39.95" Quantity="4"/>
    </LineItem>
  </LineItems>
</Invoice>
```

第 3 章のスクリプトの例 : PurchaseOrder.xml

```
<PurchaseOrder
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="http://www.oracle.com/xdb/po.xsd">
  <Reference>ADAMS-20011127121040988PST</Reference>
  <Actions>
    <Action>
      <User>SCOTT</User>
      <Date xsi:nil="true"/>
    </Action>
  </Actions>
  <Reject/>
  <Requestor>Julie P. Adams</Requestor>
  <User>ADAMS</User>
  <CostCenter>R20</CostCenter>
  <ShippingInstructions>
    <name>Julie P. Adams</name>
    <address>300 Oracle Parkway, Redwood Shores, CA 94065</address>
    <telephone>650 506 7300</telephone>
  </ShippingInstructions>
  <SpecialInstructions>Ground</SpecialInstructions>
  <LineItems>
    <LineItem ItemNumber="1">
      <Description>The Ruling Class</Description>
      <Part Id="715515012423" UnitPrice="39.95" Quantity="2"/>
    </LineItem>
    <LineItem ItemNumber="2">
```

```
      <Description>Diabolique</Description>
      <Part Id="037429135020" UnitPrice="29.95" Quantity="3"/>
    </LineItem>
    <LineItem ItemNumber="3">
      <Description>8 1/2</Description>
      <Part Id="037429135624" UnitPrice="39.95" Quantity="4"/>
    </LineItem>
  </LineItems>
</PurchaseOrder>
```

注意 :

- XML Schema の例「XML Schema の例 : PurchaseOrder.xsd」は、付録 B「XML Schema の手引き」に記載しています。
- XSL ファイルの例「XSLT スタイルシートの例 : PurchaseOrder.xsl」は、付録 D「XSLT の手引き」に記載しています。

第 3 章のスクリプトの例 : FTP スクリプト

```
#! /usr/bin/ksh
TESTDIR=$1
TESTFILENAME=$2
. ./config.sh
SCRIPTFILE='date '+%Y%m%d%H%M%S''
SCRIPTFILE=/tmp/$SCRIPTFILE.cmd
mkdir /tmp/$TESTDIR
echo "open $ORAHOSTNAME $ORAFTPPORT" > $SCRIPTFILE
echo "user $ORASQLUSER $ORASQLPASSWORD" >> $SCRIPTFILE
echo "cd public" >>  $SCRIPTFILE
echo "cd $TESTDIR" >>  $SCRIPTFILE
echo "put $TESTFILENAME" >>  $SCRIPTFILE
echo "ls -l" >> $SCRIPTFILE
echo "get $TESTFILENAME /tmp/$TESTDIR/$TESTFILENAME" >> $SCRIPTFILE
echo "quit" >>  $SCRIPTFILE
ftp -v -n < $SCRIPTFILE
rm $SCRIPTFILE
echo "Diff Results for $TESTFILENAME"
diff -b $TESTFILENAME /tmp/$TESTDIR/$TESTFILENAME
rm -rf /tmp/$TESTDIR
```

第 3 章のスクリプトの例 : FTP ポートおよび HTTP ポートの構成

```
#! /usr/bin/ksh
ORAHOSTNAME='hostname'
ORAFTPPORT=2100
```

```
ORAHTTPPORT=8080

if [ "$LOGNAME" = "oracle2" ]
then
  ORAFTPPORT=2122
  ORAHTTPPORT=8088
fi
echo "FTP  Port = $ORAFTPPORT"
echo "HTTP Port = $ORAHTTPPORT"

ORASQLUSER=DOC92
ORASQLPASSWORD=DOC92
```

RESOURCE_VIEW および PATH_VIEW のデータベースおよび XML Schema

次の項では、RESOURCE_VIEW および PATH_VIEW の構造について説明します。

RESOURCE_VIEW の定義および構造

RESOURCE_VIEW には、リポジトリの各リソースに対して 1 つの行が含まれます。次に、RESOURCE_VIEW の構造を示します。

Column	Datatype	Description
RES	XMLTYPE	A resource in Oracle XML Repository
ANY_PATH	VARCHAR2	A path that can be used to access the resource in the Repository

参照： 第 15 章「RESOURCE_VIEW および PATH_VIEW」を参照してください。

PATH_VIEW の定義および構造

PATH_VIEW には、リポジトリの一意の各パスに対して 1 つの行が含まれます。次に PATH_VIEW の構造を示します。

Column	Datatype	Description
PATH	VARCHAR2	Path name of a resource
RES	XMLTYPE	The resource referred by PATH
LINK	XMLTYPE	Link property

参照： 第 15 章「RESOURCE_VIEW および PATH_VIEW」を参照してください。

XDBResource.xsd: Oracle XML DB リソースを表すための XML Schema

この項では、Oracle XML DB リソースを表すために使用される XML Schema XDBResource.xsd を示します。

XDBResource.xsd

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://xmlns.oracle.com/xdb/XDBResource.xsd"
version="1.0" elementFormDefault="qualified"
xmlns:res="http://xmlns.oracle.com/xdb/XDBResource.xsd">

  <simpleType name="OracleUserName">
    <restriction base="string">
      <minLength value="1" fixed="false"/>
      <maxLength value="4000" fixed="false"/>
    </restriction>
  </simpleType>

  <simpleType name="ResMetaStr">
    <restriction base="string">
      <minLength value="1" fixed="false"/>
      <maxLength value="128" fixed="false"/>
    </restriction>
  </simpleType>

  <simpleType name="SchElemType">
    <restriction base="string">
      <minLength value="1" fixed="false"/>
      <maxLength value="4000" fixed="false"/>
    </restriction>
  </simpleType>

  <simpleType name="GUID">
    <restriction base="hexBinary">
      <minLength value="8" fixed="false"/>
      <maxLength value="32" fixed="false"/>
    </restriction>
  </simpleType>

  <simpleType name="LocksRaw">
    <restriction base="hexBinary">
      <minLength value="0" fixed="false"/>
      <maxLength value="2000" fixed="false"/>
    </restriction>
```

```
  </simpleType>

  <simpleType name="LockScopeType">
    <restriction base="string">
      <enumeration value="Exclusive" fixed="false"/>
      <enumeration value="Shared" fixed="false"/>
    </restriction>
  </simpleType>

  <complexType name="LockType" mixed="false">
    <sequence>
      <element name="owner" type="string"/>
      <element name="expires" type="dateTime"/>
      <element name="lockToken" type="hexBinary"/>
    </sequence>
    <attribute name="LockScope" type="res:LockScopeType" />
  </complexType>

  <complexType name="ResContentsType" mixed="false">
    <sequence >
      <any name="ContentsAny" />
    </sequence>
  </complexType>

  <complexType name="ResAclType" mixed="false">
    <sequence >
      <any name="ACLAny"/>
    </sequence>
  </complexType>

  <complexType name="ResourceType" mixed="false">
    <sequence >
      <element name="CreationDate" type="dateTime"/>
      <element name="ModificationDate" type="dateTime"/>
      <element name="Author" type="res:ResMetaStr"/>
      <element name="DisplayName" type="res:ResMetaStr"/>
      <element name="Comment" type="res:ResMetaStr"/>
      <element name="Language" type="res:ResMetaStr"/>
      <element name="CharacterSet" type="res:ResMetaStr"/>
      <element name="ContentType" type="res:ResMetaStr"/>
      <element name="RefCount" type="nonNegativeInteger"/>
      <element name="Lock" type="res:LocksRaw"/>
      <element pname="ACL" type="res:ResAclType"
minOccurs="0" maxOccurs="1"/>
      <element name="Owner" type="res:OracleUserName"
minOccurs="0" maxOccurs="1"/>
      <element name="Creator" type="res:OracleUserName"
```

```
minOccurs="0" maxOccurs="1"/>
      <element name="LastModifier" type="res:OracleUserName"
minOccurs="0" maxOccurs="1"/>
      <element name="SchemaElement" type="res:SchElemType"
minOccurs="0" maxOccurs="1"/>
      <element name="Contents" type="res:ResContentsType"
minOccurs="0" maxOccurs="1"/>
      <element name="VCRUID" type="res:GUID"/>
      <element name="Parents" type="hexBinary" minOccurs="0"
maxOccurs="1000"/>      <any name="ResExtra"
namespace="##other" minOccurs="0" maxOccurs="65535"/>
    </sequence>

    <attribute name="Hidden" type="boolean"/>
    <attribute name="Invalid" type="boolean"/>
    <attribute name="VersionID" type="integer"/>
    <attribute name="ActivityID" type="integer"/>
    <attribute name="Container" type="boolean"/>
    <attribute name="CustomRslv" type="boolean"/>
    <attribute name="StickyRef" type="boolean"/>

  </complexType>

  <element name="Resource" type="res:ResourceType"/>

</schema>
```

acl.xsd: Oracle XML DB の ACL を表すための XML Schema

この項では、Oracle XML DB の ACL を表すために使用される XML Schema について説明します。

ACL を表す XML Schema: acl.xsd

この項では、Oracle XML DB の ACL を表すために使用される XML Schema `acl.xsd` を示します。

acl.xsd

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"

targetNamespace="http://xmlns.oracle.com/xdb/acl.xsd"
version="1.0"
        xmlns:xdb="http://xmlns.oracle.com/xdb"
        xmlns:xdbacl="http://xmlns.oracle.com/xdb/acl.xsd"
```

```
        elementFormDefault="qualified">

   <annotation>
     <documentation>
        This XML schema describes the structure of XML DB ACL
documents.

        Note : The following "systemPrivileges" element lists
all supported
          system privileges and their aggregations.
          See dav.xsd for description of DAV privileges
        Note : The elements and attributes marked "hidden" are for
          internal use only.
     </documentation>
     <appinfo>
       <xdb:systemPrivileges>
        <xdbacl:all>
          <xdbacl:read-properties/>
          <xdbacl:read-contents/>
          <xdbacl:read-acl/>
          <xdbacl:update/>
          <xdbacl:link/>
          <xdbacl:unlink/>
          <xdbacl:unlink-from/>
          <xdbacl:write-acl-ref/>
          <xdbacl:update-acl/>
          <xdbacl:link-to/>
          <xdbacl:resolve/>
        </xdbacl:all>
       </xdb:systemPrivileges>
     </appinfo>
   </annotation>

  <!-- privilegeNameType (this is an emptycontent type) -->
  <complexType name = "privilegeNameType"/>

  <!-- privilegeName element
       All system and user privileges are in the
substitutionGroup
       of this element.
    -->
  <element name = "privilegeName"
type="xdbacl:privilegeNameType"
           xdb:defaultTable=""/>

  <!-- all system privileges in the XML DB ACL namespace -->
  <element name = "read-properties"
```

```
type="xdbacl:privilegeNameType"
           substitutionGroup="xdbacl:privilegeName"
xdb:defaultTable=""/>
  <element name = "read-contents"
type="xdbacl:privilegeNameType"
           substitutionGroup="xdbacl:privilegeName"
xdb:defaultTable=""/>
  <element name = "read-acl" type="xdbacl:privilegeNameType"

           substitutionGroup="xdbacl:privilegeName"
xdb:defaultTable=""/>
  <element name = "update" type="xdbacl:privilegeNameType"
           substitutionGroup="xdbacl:privilegeName"
xdb:defaultTable=""/>
  <element name = "link" type="xdbacl:privilegeNameType"
           substitutionGroup="xdbacl:privilegeName"
xdb:defaultTable=""/>
  <element name = "unlink" type="xdbacl:privilegeNameType"
           substitutionGroup="xdbacl:privilegeName"
xdb:defaultTable=""/>
  <element name = "unlink-from" type="xdbacl:privilegeNameType"
           substitutionGroup="xdbacl:privilegeName"
xdb:defaultTable=""/>
  <element name = "write-acl-ref"
type="xdbacl:privilegeNameType"
           substitutionGroup="xdbacl:privilegeName"
xdb:defaultTable=""/>
  <element name = "update-acl"
type="xdbacl:privilegeNameType"
           substitutionGroup="xdbacl:privilegeName"
xdb:defaultTable=""/>
  <element name = "link-to" type="xdbacl:privilegeNameType"
           substitutionGroup="xdbacl:privilegeName"
xdb:defaultTable=""/>
  <element name = "resolve" type="xdbacl:privilegeNameType"
           substitutionGroup="xdbacl:privilegeName"
xdb:defaultTable=""/>
  <element name = "all" type="xdbacl:privilegeNameType"
           substitutionGroup="xdbacl:privilegeName"
xdb:defaultTable=""/>

  <!-- privilege element -->
  <element name = "privilege" xdb:SQLType = "XDB$PRIV_T"
xdb:defaultTable="">
    <complexType>
      <choice maxOccurs="unbounded">
        <any xdb:transient="generated"/>
```

```
        <!-- HIDDEN ELEMENTS -->
        <element name = "privNum" type = "hexBinary" xdb:baseProp="true"
                 xdb:hidden="true"/>
      </choice>
    </complexType>
  </element>

  <!-- ace element -->
  <element name = "ace" xdb:SQLType = "XDB$ACE_T"
xdb:defaultTable="">
     <complexType>
      <sequence>
       <element name = "grant" type = "boolean"/>
       <element name = "principal" type = "string"
                xdb:transient="generated"/>
       <element ref="xdbacl:privilege"
minOccurs="1"/>
       <!-- HIDDEN ELEMENTS -->
       <element name = "principalID" type = "hexBinary"
minOccurs="0"
                xdb:baseProp="true" xdb:hidden="true"/>
       <element name = "flags" type = "unsignedInt"
minOccurs="0"
                xdb:baseProp="true" xdb:hidden="true"/>
      </sequence>
     </complexType>
  </element>

  <!-- acl element -->
  <element name = "acl" xdb:SQLType = "XDB$ACL_T" xdb:defaultTable = "XDB$ACL">
    <complexType>
     <sequence>
      <element name = "schemaURL" type = "string" minOccurs="0"
               xdb:transient="generated"/>
      <element name = "elementName" type = "string" minOccurs="0"
               xdb:transient="generated"/>
      <element ref = "xdbacl:ace" minOccurs="1" maxOccurs = "unbounded"
               xdb:SQLCollType="XDB$ACE_LIST_T"/>

      <!-- HIDDEN ELEMENTS -->
      <element name = "schemaOID" type = "hexBinary" minOccurs="0"
               xdb:baseProp="true" xdb:hidden="true"/>
      <element name = "elementNum" type = "unsignedInt" minOccurs="0"
               xdb:baseProp="true" xdb:hidden="true"/>
     </sequence>
     <attribute name = "shared" type = "boolean"
default="true"/>
```

```
     <attribute name = "description" type = "string"/>
    </complexType>
  </element>

 </schema>';
```

xdbconfig.xsd: Oracle XML DB を構成するための XML Schema

この項では、Oracle XML DB を構成するための XML Schema xdbconfig.xsd を示します。

xdbconfig.xsd

```
<schema
targetNamespace="http://xmlns.oracle.com/xdb/xdbconfig.xsd"
        xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:xdbc="http://xmlns.oracle.com/xdb/xdbconfig.xsd"
        xmlns:xdb="http://xmlns.oracle.com/xdb"
        version="1.0" elementFormDefault="qualified">

  <element name="xdbconfig" xdb:defaultTable="XDB$CONFIG">

  <complexType><sequence>

    <!-- predefined XML DB properties - these should NOT be
changed -->
    <element name="sysconfig">
    <complexType><sequence>
        <!-- generic XML DB properties -->
        <element name="acl-max-age" type="unsignedInt"
default="1000"/>
        <element name="acl-cache-size" type="unsignedInt"
default="32"/>
        <element name="invalid-pathname-chars" type="string"
default=""/>
        <element name="case-sensitive" type="boolean"
default="true"/>
        <element name="call-timeout"   type="unsignedInt"
default="300"/>
        <element name="max-link-queue" type="unsignedInt"
default="65536"/>
        <element name="max-session-use" type="unsignedInt"
default="100"/>
        <element name="persistent-sessions" type="boolean"
default="false"/>
        <element name="default-lock-timeout" type="unsignedInt"
```

```
default="3600"/>
        <element name="xdbcore-logfile-path" type="string"
default="/sys/log/xdblog.xml"/>
        <element name="xdbcore-log-level" type="unsignedInt"
default="0"/>
        <element name="resource-view-cache-size" type="unsignedInt"
default="1048576"/>
        <element name="case-sensitive-index-clause" type="string" minOccurs="0"/>

        <!-- protocol specific properties -->
        <element name="protocolconfig">
        <complexType><sequence>

            <!-- these apply to all protocols -->
            <element name="common">
            <complexType><sequence>
                <element name="extension-mappings">
                <complexType><sequence>
                  <element name="mime-mappings"
type="xdbc:mime-mapping-type"/>
                  <element name="lang-mappings"
type="xdbc:lang-mapping-type"/>
                  <element name="charset-mappings"
type="xdbc:charset-mapping-type"/>
                  <element name="encoding-mappings"
type="xdbc:encoding-mapping-type"/>
                </sequence></complexType>
                </element>
                <element name="session-pool-size"
type="unsignedInt"

default="50"/>
                <element name="session-timeout"
type="unsignedInt"

default="6000"/>

            </sequence></complexType>
            </element>

            <!-- FTP specific -->
            <element name="ftpconfig">
            <complexType><sequence>
              <element name="ftp-port" type="unsignedShort" default="2100"/>
              <element name="ftp-listener" type="string"/>
              <element name="ftp-protocol" type="string"/>
              <element name="logfile-path" type="string"
```

```
default="/sys/log/ftplog.xml"/>
              <element name="log-level" type="unsignedInt" default="0"/>
              <element name="session-timeout"  type="unsignedInt"
default="6000"/>
              <element name="buffer-size" default="8192">
                <simpleType>
                  <restriction base="unsignedInt">
                    <minInclusive value="1024"/>       <!-- 1KB -->
                    <maxInclusive value="1048496"/>    <!-- 1MB -->
                  </restriction>
                </simpleType>
              </element>
            </sequence></complexType>
            </element>

            <!-- HTTP specific -->
            <element name="httpconfig">
            <complexType><sequence>
                <element name="http-port" type="unsignedShort"
default="8080"/>
                <element name="http-listener" type="string"/>
                <element name="http-protocol" type="string"/>
                <element name="max-http-headers" type="unsignedInt"
default="64"/>
                <element name="max-header-size" type="unsignedInt"
default="4096"/>
                <element name="max-request-body" type="unsignedInt"
default="2000000000" minOccurs="1"/>
                <element name="session-timeout"  type="unsignedInt"
default="6000"/>
                <element name="server-name" type="string"/>
                <element name="logfile-path" type="string"
default="/sys/log/httplog.xml"/>
                <element name="log-level" type="unsignedInt"
default="0"/>
                <element name="servlet-realm" type="string"
minOccurs="0"/>

                <element name="webappconfig">
                <complexType><sequence>
                <element name="welcome-file-list"
type="xdbc:welcome-file-type"/>
                  <element name="error-pages"
type="xdbc:error-page-type"/>
                  <element name="servletconfig"
type="xdbc:servlet-config-type"/>
                </sequence></complexType>
```

```
                </element>
                <element name="default-url-charset" type="string"
minOccurs="0"/>
             </sequence></complexType>
             </element>

         </sequence></complexType>
         </element>

     </sequence></complexType>
     </element>


    <!-- users can add any properties they want here -->
    <element name="userconfig" minOccurs="0">
         <complexType><sequence>
            <any maxOccurs="unbounded" namespace="##other"/>
         </sequence></complexType>
    </element>

  </sequence></complexType>

  </element>

  <complexType name="welcome-file-type">
     <sequence>
         <element name="welcome-file" minOccurs="0" maxOccurs="unbounded">
          <simpleType>
           <restriction base="string">
            <pattern value="[^/]*"/>
           </restriction>
          </simpleType>
         </element>
     </sequence>
  </complexType>

  <!-- customized error pages -->
  <complexType name="error-page-type">
  <sequence>
        <element name="error-page" minOccurs="0" maxOccurs="unbounded">
        <complexType><sequence>
              <choice>
                <element name="error-code">
                  <simpleType>
                    <restriction base="positiveInteger">
                      <minInclusive value="100"/>
                      <maxInclusive value="999"/>
```

```
                    </restriction>
                  </simpleType>
                </element>

                <!-- Fully qualified classname of a Java exception type -->
                <element name="exception-type" type="string"/>

                <element name="OracleError">
                 <complexType><sequence>
                    <element name="facility" type="string" default="ORA"/>
                    <element name="errnum" type="unsignedInt"/>
                 </sequence></complexType>
                </element>
              </choice>

              <element name="location" type="anyURI"/>

        </sequence></complexType>
        </element>
  </sequence>
  </complexType>


  <!-- parameter for a servlet: name, value pair and a
description  -->
  <complexType name="param">
    <sequence>
      <element name="param-name" type="string"/>
      <element name="param-value" type="string"/>
      <element name="description" type="string"/>
    </sequence>
  </complexType>

  <complexType name="servlet-config-type">
    <sequence>
         <element name="servlet-mappings">
           <complexType><sequence>
              <element name="servlet-mapping" minOccurs="0"
                       maxOccurs="unbounded">
              <complexType><sequence>
                  <element name="servlet-pattern"
type="string"/>
                  <element name="servlet-name"
type="string"/>
              </sequence></complexType>
              </element>
           </sequence></complexType>
```

```
          </element>

      <element name="servlet-list">
        <complexType><sequence>
             <element name="servlet" minOccurs="0"
maxOccurs="unbounded">
               <complexType><sequence>
                 <element name="servlet-name"
type="string"/>
                 <element name="servlet-language">
                   <simpleType>
                     <restriction base="string">
                       <enumeration value="C"/>
                       <enumeration value="Java"/>
                       <enumeration value="PL/SQL"/>
                     </restriction>
                   </simpleType>
                 </element>
                 <element name="icon" type="string"
minOccurs="0"/>
                 <element name="display-name"
type="string"/>
                 <element name="description" type="string"
minOccurs="0"/>
                 <choice>
                   <element name="servlet-class" type="string"
minOccurs="0"/>
                   <element name="jsp-file" type="string"
minOccurs="0"/>
                 </choice>
                 <element name="servlet-schema" type="string"
minOccurs="0"/>
                 <element name="init-param" minOccurs="0"
                          maxOccurs="unbounded" type="xdbc:param"/>
                 <element name="load-on-startup" type="string"
minOccurs="0"/>
                 <element name="security-role-ref" minOccurs="0"
                       maxOccurs="unbounded">
                 <complexType><sequence>
                     <element name="description"
type="string" minOccurs="0"/>
                     <element name="role-name"
type="string"/>
                     <element name="role-link"
type="string"/>
                 </sequence></complexType>
                 </element>
```

```
                </sequence></complexType>
              </element>
        </sequence></complexType>
      </element>
  </sequence>
  </complexType>


  <complexType name="lang-mapping-type"><sequence>
      <element name="lang-mapping" minOccurs="0"
maxOccurs="unbounded">
        <complexType><sequence>
        <element name="extension" type="xdbc:exttype"/>
        <element name="lang" type="string"/>
        </sequence></complexType>
      </element></sequence>
  </complexType>


  <complexType name="charset-mapping-type"><sequence>
      <element name="charset-mapping" minOccurs="0"
maxOccurs="unbounded">
        <complexType><sequence>
        <element name="extension" type="xdbc:exttype"/>
        <element name="charset" type="string"/>
        </sequence></complexType>
      </element></sequence>
  </complexType>

  <complexType name="encoding-mapping-type"><sequence>
      <element name="encoding-mapping" minOccurs="0"
maxOccurs="unbounded">
        <complexType><sequence>
        <element name="extension" type="xdbc:exttype"/>
        <element name="encoding" type="string"/>
        </sequence></complexType>
      </element></sequence>
  </complexType>

 <complexType name="mime-mapping-type"><sequence>
      <element name="mime-mapping" minOccurs="0"
maxOccurs="unbounded">
      <complexType><sequence>
        <element name="extension" type="xdbc:exttype"/>
        <element name="mime-type" type="string"/>
      </sequence></complexType>
      </element></sequence>
```

```
  </complexType>

  <simpleType name="exttype">
      <restriction base="string">
        <pattern value="[^\*\./]*"/>
      </restriction>
  </simpleType>

</schema>
```

用語集

ACE

アクセス制御エントリ。「アクセス制御エントリ（Access Control Entry: ACE）」を参照。

ACL

アクセス制御リスト。「アクセス制御リスト（Access Control List: ACL）」を参照。

API

Application Program Interface。「Application Program Interface（API）」を参照。

Application Program Interface（API）

一連のパブリック・プログラム・インタフェース。オペレーティング・システム、またはデータベース、Web サーバー、JVM などの他のプログラム環境と通信するための言語およびメッセージ形式で構成される。通常、これらのメッセージは、アプリケーション開発に使用可能なファンクションおよびメソッドをコールする。

BC4J

Business Components for Java。JDeveloper に付属する J2EE アプリケーション開発フレームワーク。BC4J は、J2EE Design Patterns を実装するオブジェクト・リレーショナル・マッピング・ツールである。

BFILE

オペレーティング・システム内に常駐するデータベース表領域の外に存在する外部バイナリ・ファイル。BFILE はデータベース・セマンティクスから参照され、外部 LOB ともいう。

BLOB

「バイナリ・ラージ・オブジェクト（Binary Large Object: BLOB）」を参照。

Business-to-Business（B2B）

商品販売およびサービス提供における企業間の相互のコミュニケーションを示す用語。これを実現するソフトウェア・インフラストラクチャが取引である。

Business-to-Consumer（B2C）

商品販売およびサービス提供における企業と顧客間のコミュニケーションを示す用語。

CDATA

「文字データ（character Data: CDATA）」を参照。

CDF

チャネル定義形式。インターネット上のチャネルに関する情報を交換する方法を提供する。

CGI

「Common Gateway Interface（CGI）」を参照。

Class Generator

入力ファイルを受け入れ、対応する機能を持つ一連の出力クラスを作成するユーティリティ。XML Class Generator の場合、入力ファイルは DTD であり、出力は、その DTD に準拠する XML 文書を作成するために使用できる一連のクラスである。

CLASSPATH

JVM がアプリケーションの実行に必要なクラスを検索するために使用する、オペレーティング・システムの環境変数。

CLOB

「キャラクタ・ラージ・オブジェクト（Character Large Object: CLOB）」を参照。

Common Gateway Interface（CGI）

Web サーバーが他のプログラムを実行し、ブラウザに送信された HTML ページ、図形、オーディオおよびビデオに出力を渡すことを可能にするプログラム・インタフェース。

Common Object Request Broker Architecture（CORBA）

ネットワーク全体の分散オブジェクト間の通信用の、Object Management Group による標準。これらの自己完結型ソフトウェア・モジュールは、異なるプラットフォームまたはオペレーティング・システム上で実行しているアプリケーションで使用できる。CORBA オブジェクトとそのデータ形式、およびファンクションは、Java、C、C++、Smalltalk、COBOL などの様々な言語にコンパイルできるインタフェース定義言語（IDL）で定義される。

Common Oracle Runtime Environment（CORE）

C で作成された関数のライブラリ。これによって開発者は、事実上すべてのプラットフォームおよびオペレーティング・システムに簡単に移植可能なコードを作成できる。

CORBA

「Common Object Request Broker Architecture（CORBA）」を参照。

CSS

「カスケーディング・スタイルシート（Cascading Style Sheets: CSS）」を参照。

DBUriType

データ型のインスタンスを格納するために使用される Oracle9*i* データベースのデータ型。このデータ型では、データベース・スキーマの XPath を使用したナビゲーションが可能になる。

DOCTYPE

XML 文書内に DTD またはその参照を指定するタグ名として使用される用語。たとえば、`<!DOCTYPE person SYSTEM "person.dtd">` では、ルート要素名が person として、また外部 DTD が person.dtd としてファイル・システム内に宣言される。内部 DTD は、DOCTYPE 宣言内で宣言される。

Document Type Definition（DTD）

XML 文書の使用可能な構造を定義する一連の規則。DTD は、SGML から書式を導出し、DOCTYPE 要素を使用するか、または DOCTYPE 参照を介して外部ファイルを使用して XML 文書内に含めることができるテキスト・ファイルである。

DOM

「ドキュメント・オブジェクト・モデル（Document Object Model: DOM）」を参照。

DOM 再現性（DOM fidelity）

データの整合性および精度を確認するため、Oracle XML DB に格納された XML 文書を再生成するときなどに、Oracle XML DB では DOM 再現性と呼ばれるデータ整合メカニズムが使用される。DOM 再現性とは、特に DOM 検索の目的で、戻された XML 文書が元の XML 文書と同一であることを意味する。Oracle XML DB は、バイナリ属性 `SYS_XDBPD$` を使用して DOM 再現性を確保する。

DTD

「Document Type Definition（DTD）」を参照。

EDI

電子データ交換。

Enterprise JavaBeans

サーバー上の JVM 内で実行する独立プログラム・モジュール。CORBA によって Enterprise JavaBeans のインフラストラクチャが提供され、コンテナ・レイヤーによって、サポートされたサーバーにセキュリティ、トランザクション・サポートおよびその他の共通機能が提供される。

existsNode

`XMLType` 内に `XPath` が存在するかどうかに基づいて TRUE または FALSE を戻す SQL 演算子。

eXtensible Markup Language（XML）

データ記述のオープン標準。SGML 構文のサブセットを使用して World Wide Web Consortium（W3C）によって開発され、インターネットでの使用目的で設計された。

eXtensible Stylesheet Language Formatting Object（XSLFO）

書式設定セマンティクスを指定するための XML 用語を定義する、W3C の標準仕様。「FOP」を参照。

eXtensible Stylesheet Language Transformation（XSLT）

XSLT とも書く。XML 文書を別のドキュメントに変換する変換言語を定義する、W3C の XSL 標準仕様。

eXtensible Stylesheet Language（XSL）

XML 文書を変換またはレンダリングするために、スタイルシート内で使用される言語。W3C の XSL は、XSL Transformations および XSL Formatting Objects という 2 つの W3C 勧告で構成されている。XSL Transformations は 1 つの XML 文書を別の XML 文書に変換し、XSL Formatting Objects は XML 文書の表示を指定する。XSL は、スタイルシートを表す言語である。XSL は、次の 2 つで構成される。

- XML 文書を変換するための言語（XSLT）
- 書式設定セマンティクスを指定するための XML ボキャブラリ（XSLFO）

XSLT スタイルシートは、書式設定用ボキャブラリを使用する XML 文書へのクラスのインスタンスの変換方法を記述して、XML 文書のクラス表示を指定する。

extract

`XMLType` として格納された XML 文書のフラグメントを取り出す SQL 演算子。

FOP

XSL Formatting Objects によって駆動される出力フォーマッタ。FOP は、書式設定オブジェクト・ツリーを読み込んで、結果ページを指定された出力にレンダリングする Java アプリケーションである。出力形式には、現在、PDF、PCL、PS、SVG、XML（領域ツリー表現）、Print、AWT、MIF および TXT がサポートされている。主な出力ターゲットは PDF である。

HASPATH

Oracle Text に含まれ、特定の XPath の存在を確認するために XMLType データ型の問合せに使用される SQL 演算子。

HTML

「Hypertext Markup Language（HTML）」を参照。

HTTP

「Hypertext Transfer Protocol（HTTP）」を参照。

HTTPUriType

データ型のインスタンスを格納するために使用されるデータ型。このデータ型では、リモート・データベース内でデータベース・スキーマの XPath ベースのナビゲーションが可能になる。

Hypertext Markup Language（HTML）

Web ブラウザに送信するファイルを作成するために使用し、Web の基礎として機能するマークアップ言語。HTML の次のバージョンは xHTML と呼ばれ、XML アプリケーションになる予定である。

Hypertext Transfer Protocol（HTTP）

インターネットを介して、Web サーバーとブラウザ間で HTML ファイルを転送するために使用するプロトコル。

***i*AS**

「Oracle9i Application Server（Oracle9iAS）」を参照。

IDE

「統合開発環境（Integrated Development Environment: IDE）」を参照。

INPATH

Oracle Text に含まれ、特定の XPath の特定のテキストを検索するために XMLType データ型の問合せに使用される SQL 演算子。

***inter*Media**

複合データ型のコレクションおよびコレクションの Oracle 内でのアクセス。これには、テキスト、ビデオ、時系列および空間データ型が含まれる。

Internet Inter-ORB Protocol（IIOP）

インターネットなどの TCP/IP ネットワーク上で、メッセージを交換するために CORBA が使用するプロトコル。

J2EE

「Java 2 Platform Enterprise Edition（J2EE)」を参照。

Java

Sun 社によって開発およびメンテナンスされた高水準のプログラミング言語。Java では、アプリケーションが JVM という仮想マシン内で実行される。JVM は、オペレーティング・システムに対するすべてのインタフェースの役割を担う。開発者は、このアーキテクチャによって、JVM を搭載するすべてのオペレーティング・システムまたはプラットフォームで実行する Java アプリケーションおよびアプレットを開発できる。

Java 2 Platform Enterprise Edition（J2EE)

複数層のエンタープライス・コンピューティングを定義する Java プラットフォーム（Sun 社)。

Java API for XML Processing（JAXP）

特定の XML プロセッサの実装に依存しない API を使用して、アプリケーションで XML 文書の解析および変換を可能にする。

JavaBean

JVM 内で実行する独立プログラム・モジュール。通常、クライアント側のユーザー・インタフェースを作成するために使用される。Java Bean ともいう。サーバー側の同等のモジュールは、Enterprise JavaBeans という。「Enterprise JavaBeans」を参照。

Java Database Connectivity（JDBC）

Java アプリケーションが、SQL 言語を介してデータベースにアクセスできるようにするプログラム API。JDBC ドライバは、プラットフォームに依存しないように Java で作成されるが、各データベースに固有である。

Java Development Kit（JDK）

Java 開発環境を確立する Java バージョン用の、Java クラス、ランタイム、コンパイラ、デバッガおよびソース・コードのコレクション。JDK はバージョンで指定され、Java 2 はバージョン 1.2 以上を指定するために使用される。

Java Naming and Directory Interface（JNDI）

Sun 社が提供するプログラム・インタフェース。Java プログラムを、DNS、LDAP、NDS などのネーミング・サービスおよびディレクトリ・サービスに接続する。

Java Runtime Environment（JRE）

プラットフォーム上で Java Virtual Machine を構成する、コンパイル済クラスのコレクション。JRE はバージョンで指定され、Java 2 はバージョン 1.2 以上を指定するために使用される。

JavaServer Pages（JSP）

Web ページへの単純なプログラム・インタフェースを可能にする、サーブレットの拡張機能。JSP は、特殊タグ、および Web サーバーまたはアプリケーション・サーバーで実行される埋込み Java コードを含む HTML ページであり、HTML ページに動的機能を提供する。JSP は、サーバーの JVM で最初に要求および実行されるときに、サーブレットにコンパイルされる。

Java Virtual Machine（JVM）

コンパイル済 Java バイトコードをプラットフォームのマシン言語に変換し、それを実行する Java インタプリタ。JVM は、クライアント側、ブラウザ内、中間層内、イントラネット上、Oracle9iAS などのアプリケーション・サーバー上、または Oracle などのデータベース・サーバー内で実行できる。

JAXP

「Java API for XML Processing（JAXP）」を参照。

JDBC

「Java Database Connectivity（JDBC）」を参照。

JDeveloper

アプリケーション、アプレットおよびサーブレットの開発を可能にする Oracle の Java IDE。エディタ、コンパイラ、デバッガ、構文チェッカ、ヘルプ・システム、統合 UML クラス・モデラーなどを含む。JDeveloper は、エディタで XML がサポートされるとともに、簡単に使用できるように統合された Oracle XDK for Java を搭載して、XML ベースの開発をサポートするように拡張されている。

JDK

「Java Development Kit（JDK）」を参照。

JNDI

「Java Naming and Directory Interface（JNDI）」を参照。

JVM

「Java Virtual Machine（JVM）」を参照。

LAN

「Local Area Network（LAN）」を参照。

LOB

「ラージ・オブジェクト（Large Object: LOB）」を参照。

Local Area Network（LAN）

限定された地域内のユーザー用のコンピュータ通信ネットワーク。LAN は、サーバー、ワークステーション、通信ハードウェア（ルーター、ブリッジ、ネットワーク・カードなど）およびネットワーク・オペレーティング・システムで構成される。

NCLOB

「各国語キャラクタ・ラージ・オブジェクト（National Character Large Object: NCLOB）」を参照。

N 層（N-tier）

クライアントおよびサーバーで構成される 1 つ以上の層で構成される、コンピュータ通信ネットワーク・アーキテクチャの指定。通常、2 層システムは 1 つのクライアント・レベルおよび 1 つのサーバー・レベルで構成される。3 層システムは、通常、1 つのクライアント層とともに、データベース・サーバーと Web またはアプリケーション・サーバーの 2 つのサーバー層を使用する。

OAG

Open Applications Group。

OAI

Oracle Applications Integrator。CRM アプリケーションを Oracle ERP に加えて他の ERP システムと統合するための、Oracle iStudio 開発ツールを含むランタイム。固有の API は、「メッセージ対応」である必要がある。標準の拡張フックを使用して、他のアプリケーション・システムと交換された XML ストリームを生成または解析する。開発中。

OASIS

「Organization for the Advancement of Structured Information（OASIS）」を参照。

Object Request Broker（ORB）

クライアント側の要求元プログラムとサーバー側のオブジェクト間のメッセージ通信を管理するソフトウェア。ORB は、アクション要求およびそのパラメータをオブジェクトに渡し、結果を戻す。共通の実装は、JCORB および Enterprise JavaBeans である。「Common Object Request Broker Architecture（CORBA）」を参照。

OC4J

Oracle9*i*AS Containers for J2EE。JDeveloper に付属する J2EE 開発ツール製品。

OCT

「Ordered Collection in Tables（OCT）」を参照。

OE

Oracle Exchange。

Oracle9*i* Application Server（Oracle9*i*AS）

Oracle アプリケーション・サーバー。オープン標準フレームワーク内で高パフォーマンスの N 層トランザクション指向 Web アプリケーションを構築、配置および管理するために必要な、すべての主要なサービスおよび機能を統合する。

Oracle 9*i*FS

データベース内または中間層上で実行する、Oracle のファイル・システムおよび Java ベースの開発環境。単一のデータベース・リポジトリに複数の型のドキュメントを作成、格納および管理する方法を提供する。

ORACLE_HOME

アプリケーションで使用するために、Oracle データベースのインストール場所を識別するオペレーティング・システムの環境変数。

Oracle JVM

Oracle データベースのメモリー領域内で実行する JVM。JVM は、Oracle8*i* リリース 8.1.5 では Java1.1 準拠であり、リリース 8.1.6 以降では Java1.2 準拠である。

Oracle Text

Oracle のツール製品。XPath に類似した検索機能とともに、文書の全文索引および複数の文書にまたがった SQL 問合せを実行する機能を提供する。

Oracle XML DB

Oracle データベース・サーバーが提供する、高パフォーマンスな XML 格納および取出しテクノロジ。W3C の XML データ・モデルに基づく。

ORB

「Object Request Broker（ORB）」を参照。

Ordered Collection in Tables（OCT）

VARRAY の要素が異なる表に格納されている場合、これらの要素を Ordered Collection in Tables という。

Organization for the Advancement of Structured Information（OASIS）

会議、セミナー、展示会およびその他の教育イベントを通じて、パブリック情報標準の普及促進を目的として設立されたメンバーの組織。XML は、SGML と同様に、OASIS が活発に普及を促進している標準である。

PCDATA

「解析対象文字データ（Parsed Character Data: PCDATA）」を参照。

PDA

Palm Pilot などのパーソナル・デジタル・アシスタント。

PL/SQL

SQL を拡張した、Oracle 手続き型言語。データベース内で実行可能なプログラムを作成するために使用する。

PUBLIC

後に続く参照の、インターネット上の場所を指定する用語。

RDF

Resource Definition Framework。

SAX

「Simple API for XML（SAX）」を参照。

Secure Sockets Layer（SSL）

インターネット上のプライマリ・セキュリティ・プロトコル。ブラウザとサーバー間の暗号化形式に、公開鍵 / 秘密鍵を使用する。

SGML

「Standard Generalized Markup Language（SGML）」を参照。

Simple API for XML（SAX）

XML Parser によって提供され、イベント駆動型のアプリケーションによって使用される XML 標準インタフェース。

Simple Object Access Protocol（SOAP）

非集中型の分散環境で情報を交換するための XML ベースのプロトコル。

SOAP

「Simple Object Access Protocol（SOAP）」を参照。

SQL

「Structured Query Language（SQL）」を参照。

SSI

「サーバー側インクルード（Server-Side Include: SSI）」を参照。

SSL

「Secure Sockets Layer（SSL）」を参照。

Standard Generalized Markup Language（SGML）

マークアップおよび DTD を使用して実装された、テキスト・ドキュメントの書式を定義するための ISO 標準。

Structured Query Language（SQL）

リレーショナル・データベース内のデータをアクセスおよび処理するために使用する標準言語。

SYSTEM

後に続く参照の、ホスト・オペレーティング・システム上の場所を指定する。

SYS_XMLAGG

後に続く参照の、ホスト・オペレーティング・システム上の場所を指定する用語。

SYS_XMLGEN

渡された SQL 問合せの結果を XML として戻すシステム固有の SQL 関数。これは、`XMLType` のインスタンス化にも使用できる。

TCP/IP

「Transmission Control Protocol/Internet Protocol（TCP/IP）」を参照。

Transmission Control Protocol/Internet Protocol（TCP/IP）

TCP で構成される通信ネットワーク・プロトコル。TCP は、トランスポート機能、およびルーティング・メカニズムを提供する IP を制御する。TCP/IP は、インターネット通信の標準である。

TransXUtility

変換されたシード・データおよびメッセージのデータベースへのロードを簡単にする Java API。

UDDI

「Universal Description, Discovery and Integration（UDDI）」を参照。

UIX

「User Interface XML（UIX）」を参照。

Uniform Resource Identifier（URI）

URL および XPath を作成するために使用するアドレス構文。

Uniform Resource Locator（URL）

インターネット上のファイルの場所およびルートを定義するアドレス。URL は、Web をナビゲートするためにブラウザによって使用され、プロトコル接頭辞、ポート番号、ドメイン名、ディレクトリ名とサブディレクトリ名、およびファイル名で構成される。たとえば、http://technet.oracle.com:80/tech/xml/index.htm では、Web 上の OTN の XML サイトを検索するためにブラウザが移動する、場所およびパスが指定されている。

Universal Description, Discovery and Integration（UDDI）

この仕様は、XML を使用してサービスの記述、ビジネスの検出およびインターネット上のビジネス・サービスの統合を行う、プラットフォーム非依存フレームワークを提供する。

URI

「Uniform Resource Identifier（URI）」を参照。

URL

「Uniform Resource Locator（URL）」を参照。

User Interface XML（UIX）

Web アプリケーションを構築するフレームワークを構成する一連のテクノロジ。

W3C

「World Wide Web Consortium（W3C）」を参照。

WAN

「Wide Area Network（WAN）」を参照。

WebDAV

「World Wide Web Distributed Authoring and Versioning（WebDAV）」を参照。

Web Request Broker

URL を処理し、適切なカートリッジに送信する OAS 内のカートリッジ。

Web サービス記述言語（Web Services Description Language: WSDL）

Web サービスのインタフェース、プロトコル・バインドおよび配置の詳細を記述するための汎用 XML 言語。

Wide Area Network（WAN）

州や国などの広域内のユーザー用のコンピュータ通信ネットワーク。WAN は、サーバー、ワークステーション、通信ハードウェア（ルーター、ブリッジ、ネットワーク・カードなど）およびネットワーク・オペレーティング・システムで構成される。

World Wide Web Consortium（W3C）

1994 年に設立された、Web の標準を確立するための国際的な産業組合。W3C のサイトは、www.w3c.org である。

World Wide Web Distributed Authoring and Versioning（WebDAV）

Web 上での協調的なオーサリングのための Internet Engineering Task Force（IETF）標準。Oracle XML DB Foldering およびセキュリティ機能は、WebDAV に準拠している。

WSDL

「Web サービス記述言語（Web Services Description Language: WSDL）」を参照。

XDBBinary

バイナリ・データを含む Oracle XML DB スキーマで定義された XML 要素。`XDBBinary` 要素は、完全に構造化されていないバイナリ・データが Oracle XML DB にアップロードされたときに、リポジトリに格納される。

XDK

「XML Developer's Kit（XDK）」を参照。

XLink

XML 文書内でのハイパーリンクの使用を制御する規則で構成された XML Linking 言語。これらの規則は、W3C の勧告プロセス下の XML Linking Group によって開発されている。XLink は、XML がドキュメントの表示およびハイパーリンクの管理にサポートする 3 つの言語（XLink、XPointer および XPath）の 1 つである。

XML

「eXtensible Markup Language（XML）」を参照。

XML Developer's Kit（XDK）

ソフトウェア開発者に、アプリケーションを XML 対応にするための標準ベースの機能を提供する、一連のライブラリ、コンポーネントおよびユーティリティ。Oracle XDK for Java には、XML Parser、XSLT プロセッサ、XML Class Generator、Transviewer JavaBeans および XSQL Servlet が含まれる。

XML Gateway

ビジネス・イベントによってトリガーされる XML メッセージを作成および使用するために、Oracle E-Business Suite との統合を簡単にするための一連のサービス。

XML Query

W3C が取り組む、XML 文書を問い合せるための言語および構文の標準。

XML Schema

W3C が取り組む、XML 文書内で単純なデータ型および複合構造を表すための標準。データ型の定義や妥当性など、現在 DTD で不足している領域に取り組んでいる。Oracle XML Schema プロセッサは、オンライン取引などの E-Business アプリケーションで使用される、XML 文書およびデータの妥当性を自動的に検証する。XML Schema は、XML 文書に単純型および複合型を追加し、DTD の機能を XML Schema 定義の XML 文書に置き換える。

XML Transviewer Beans

XDK for Java に含まれる Oracle XML JavaBeans を示す Oracle 用語。

XMLType

`XMLType` 列は、データベース内の基礎となる CLOB 列を使用して XML データを格納する。

XMLType ビュー（XMLType views）

Oracle XML DB は、既存のリレーショナル・データおよびオブジェクト・リレーショナル・データベースを XML 形式にラップする方法を提供する。特に、レガシー・データが XML ではなく、それを XML 形式に移行する必要がある場合などに有効。

XPath

XSL および XPointer で使用されるドキュメント内で要素を指定するための、オープン標準の構文。XPath は、現在 W3C 勧告である。XSLT、XLink および XML Query に使用される XML 文書を操作するためのデータ・モデルおよび文法を指定する。

XPointer

XML 文書のフラグメントへの参照を記述するための W3C 勧告。XPointer は、XPath 形式の URI の終わりに使用できる。XPath ナビゲーションを使用して、XML 文書内の個別のエンティティまたはフラグメントの識別を指定する。

XSL

「eXtensible Stylesheet Language（XSL）」を参照。

XSLFO

「eXtensible Stylesheet Language Formatting Object（XSLFO）」を参照。

XSLT

「eXtensible Stylesheet Language Transformation（XSLT）」を参照。

XSQL

Oracle XSQL Servlet によって使用される指定。1 つ以上の SQL 問合せから動的 XML 文書を生成し、XML スタイルシートを使用して、サーバー内のドキュメントを変換する（オプション）機能を提供する。

アクセス制御エントリ（Access Control Entry: ACE）

アクセス制御リスト内のエントリ。指定されたプリンシパルへのアクセスを許可または禁止する。

アクセス制御リスト（Access Control List: ACL）

アクセス制御エントリのリスト。指定されたリソースへのアクセス権を所有するプリンシパルを決定する。

アプリケーション・サーバー（Application Server）

アプリケーションおよびその環境をホストするために設計されたサーバーであり、サーバー・アプリケーションの実行を許可する。代表的な例は Oracle9*i*AS で、リモート・クライアントがインタフェースを制御する場合に、Java、C、C++ および PL/SQL アプリケーションをホストできる。「Oracle9i Application Server（Oracle9iAS）」を参照。

インスタンス化（instantiate）

Java や C++ などのオブジェクト・ベース言語で使用される用語で、特定のクラスのオブジェクト作成を示す。

エンティティ（entity）

別の文字列、またはドキュメントのキャラクタ・セットに属さない特殊文字を表すことができる文字列。エンティティ、およびパーサーによってエンティティの代替となるテキストは、DTD に宣言される。

オブジェクト・ビュー（Object View）

1 つ以上のオブジェクト表または他のビューに含まれるデータの、調整された外観。オブジェクト・ビュー問合せの出力は、表として扱われる。オブジェクト・ビューは、表が使用されているほとんどの場所で使用できる。

オブジェクト・リレーショナル（object-relational）

テキスト・ドキュメント、オーディオ・ファイル、ビデオ・ファイル、ユーザー定義オブジェクトなどの高順序のデータ型を格納および操作できるリレーショナル・データベース・システムを示す用語。

親要素（parent element）

子要素という別の要素を囲む要素。たとえば、<Parent><Child></Child></Parent> は、Parent 要素がその Child 要素をラップしていることを示す。

カートリッジ（cartridge）

Java または PL/SQL のストアド・プログラム。データベースが新しいデータ型を理解および処理するために必要な機能を追加する。カートリッジは、Oracle8 以上の Oracle 内の拡張フレームワークでインタフェースの役割を担う。Oracle Text はこの種類のカートリッジであり、データベース内に格納されたテキスト・ドキュメントの読込み、書込みおよび検索のサポートを追加する。

解析対象文字データ（Parsed Character Data: PCDATA）

解析する必要があるが、タグまたは解析対象外データの一部ではないテキストで構成される要素内容。

階層的な索引付け（hierarchical indexing）

フォルダをその子に関連付けているデータは、Oracle XML DB の階層索引によって管理される。この索引によって、オペレーティング・システムのファイル・システムで使用されるディレクトリ・メカニズムに類似した、パス名の評価を高速化するメカニズムが提供される。パス名ベースのアクセスでは、通常、Oracle XML DB の階層索引が使用される。

カスケーディング・スタイルシート（Cascading Style Sheets: CSS）

スタイル（フォント、カラー、間隔など）を Web ドキュメントに追加するための単純なメカニズム。

各国語キャラクタ・ラージ・オブジェクト（National Character Large Object: NCLOB）

データベースの各国語キャラクタ・セットに対応する文字データで構成された値を持つ LOB データ型。

空要素（empty element）

テキスト内容または子要素のない要素。属性およびその値のみを含む場合がある。空要素の書式は、`<name/>`、または `<name></name>`（タグの間には空白なし）である。

キャラクタ・ラージ・オブジェクト（Character Large Object: CLOB）

データベース・キャラクタ・セットに対応する文字データで構成された値を持つ LOB データ型。CLOB は、Oracle Text の検索エンジンを使用して索引付けおよび検索できる。

クライアント / サーバー（client/server）

実際のアプリケーションはクライアント側で実行するが、ネットワークを介して、サーバー側のデータまたは他の外部プロセスにアクセスするアプリケーション・アーキテクチャを表す用語。

結果セット（result set）

1 行以上のデータで構成される SQL 問合せの出力。

コールバック（callback）

1 つのプロセスに他のプロセスを開始させ、それを継続させるプログラム方法。2 番目のプロセスは、アクションの結果、値または他のイベントとして 1 番目のプロセスをコールする。この方法は、継続的な対話を許可するユーザー・インタフェースを持つほとんどのプログラムに使用されている。

コマンドライン（command line）

ユーザーがコマンド・インタプリタのプロンプトにコマンドを入力するインタフェース・メソッド。

子要素（child element）

親要素という別の要素内に完全に含まれた要素。たとえば、
<Parent><Child></Child></Parent> は、Child 要素がその Parent 要素内にネストされていることを示す。

コンテンツ（Content）

Oracle XML DB 内のリソースの本体、およびファイルなどリソースを処理する場合にその内容を要求すると戻されるもの。コンテンツは、常に `XMLType` である。

サーバー側インクルード（Server-Side Include: SSI）

データまたは他の内容を、要求元ブラウザに送信する前に Web ページに置く HTML コマンド。

サーブレット（servlet）

サーバー（通常は Web またはアプリケーション・サーバー）内で実行する Java アプリケーション。サーブレットは、CGI スクリプトに相当する Java である。

スキーマ（schema）

データベース内の構造およびデータ型の定義。スキーマは、XML Schema の W3C 勧告をサポートする XML 文書も示す。

スタイルシート（Stylesheet）

XML では、XML 処理命令で構成される XML 文書を示す用語。XML 処理命令は、入力 XML 文書を出力 XML 文書に変換またはフォーマットするために、XML プロセッサによって使用される。

スレッド（thread）

プログラミングにおける、同時実行（マルチスレッド）をサポートするオペレーティング・システム内の、単一のメッセージまたはプロセス実行パス。

整形式（well-formed）

XML 文書が、XML 宣言で宣言された XML バージョンの構文に準拠している状態を示す用語。これには、ルート要素が単一か、タグが適切にネストされているかなどが含まれる。

セッション（session）

2 つの層の間のアクティブ接続。

属性（attribute）

要素のプロパティ。等号で区切られた名前および値で構成され、開始タグ内の要素名の後に含まれる。たとえば、`<Price units='USD'>5</Price>` では、units（単位）が属性で USD がその値である。値は、一重または二重引用符で囲む必要がある。属性は、ドキュメントまたは DTD 内に格納できる。要素には多くの属性を指定できるが、その取得順序は定義されない。

タグ（tag）

XML マークアップの単一のピース。要素の開始または終了を指定する。タグは、< で始まり > で終わる。XML には、開始タグ（<name>）、終了タグ（</name>）および空タグ（<name/>）がある。

妥当（valid）

XML 文書の構造および要素内容が、参照 DTD または内部 DTD で宣言されたものと一貫している状態を示す用語。

遅延型変換（lazy type conversions）

Oracle XML DB によって使用されるメカニズム。Java アプリケーションが最初に Java 用の XML データを要求したときに、そのデータのみを変換する。これによって、JDBC での型変換による一般的なボトルネックが低減される。

データグラム（datagram）

XSQL Servlet が処理した SQL 問合せから、HTML ページに埋め込まれたリクエスタに戻されるテキストのフラグメント。XML 形式の場合もある。

データベース・アクセス記述子（Database Access Descriptor: DAD）

データベース・アクセスに使用される、名前付きの一連の構成値。DAD は、データベース名や Oracle Net サービス名などの情報、`ORACLE_HOME` ディレクトリ、および言語、ソート型、日付言語などのグローバリゼーション・サポート構成情報を指定する。

統合開発環境（Integrated Development Environment: IDE）

ソフトウェアの開発を支援するために設計された、単一のユーザー・インタフェースから実行されるプログラム・セット。JDeveloper は、Java 開発用の IDE であり、エディタ、コンパイラ、デバッガ、構文チェッカ、ヘルプ・システムなどを含む。JDeveloper を使用すると、単一のユーザー・インタフェースを介して Java ソフトウェアを開発できる。

ドキュメント・オブジェクト・モデル（Document Object Model: DOM）

XML 文書のメモリー内ツリーベースのオブジェクト表現。要素および属性へのプログラム・アクセスを可能にする。DOM オブジェクトおよびそのインタフェースは、W3C 勧告である。プログラム・アクセス用の API など、XML 文書の DOM を指定する。DOM は、解析対象ドキュメントをオブジェクトのツリーとして表示する。

名前空間（namespace）

XML 文書内にある、関連する一連の要素名または属性を示す用語。名前空間の構文およびその使用方法は、W3C 勧告によって定義されている。たとえば <xsl:apply-templates/ > 要素は、XSL 名前空間の一部として識別される。名前空間は、XML 文書または DTD 内で、属性の構文 `xmlns:xsl="http://www.w3.org/TR/WD-xsl"` を宣言してから宣言される。

名前レベル・ロック（name-level locking）

Oracle XML DB では、コレクション・レベル・ロックではなく名前レベル・ロックが提供される。名前をコレクションに追加すると、コレクションには排他書込みロックが設定されず、コレクション内の名前のみがロックされる。名前の変更はキューに入れられ、コレクションはコミット時にのみロックされ、変更される。

ノード（node）

XML では、DOM ツリー内のアドレス指定可能な各エンティティを示す用語。

パーサー（parser）

XML で、XML 文書を入力として受け入れ、ドキュメントが整形式であり、また妥当（オプション）であるかどうかを判断するソフトウェア・プログラム。Oracle XML Parser は、SAX および DOM インタフェースの両方をサポートする。

バイナリ・ラージ・オブジェクト（Binary Large Object: BLOB）

内容がバイナリ・データで構成されたラージ・オブジェクト・データ型。このデータは、データ構造がデータベースに認識されないため、RAW 型とみなされる。

ハイパー・テキスト（hypertext）

ユーザーがハイパーリンクとして指定された文字列または句を選択して、他のドキュメントまたは図形間を操作できるテキスト・ドキュメントを作成および公開する方法。

パス名（path name）

リポジトリ階層内でのリソースの位置を反映したリソースの名前。パス名は、ルート要素（最初の /）、要素セパレータ（/）および様々なサブ要素（またはパス要素）で構成される。パス要素は、（\ または /）を除いて、データベース・キャラクタ・セットのすべての文字で構成できる。これらの文字は、Oracle XML DB で特別な意味を持つ。スラッシュは、パス名内のデフォルトの名前セパレータであり、バックスラッシュは、エスケープ文字として使用される。

表記法宣言（Notation Attribute Declaration）

XML では、パーサーが理解できない内容の型の宣言。これらの型には、オーディオ、ビデオおよび他のマルチメディアが含まれる。

ファンクション索引（function-based index）

データベース索引。これを作成すると、一般的な問合せの結果をより高速に戻すことができる。

フォルダ（Folder）

リソースを含む、または含むことができる Oracle XML DB Repository 内のディレクトリまたはノード。フォルダは、リソースでもある。

フォルダリング（Foldering）

コンテンツをリソースの階層構造に格納できるようにする Oracle XML DB の機能。

プリンシパル（principal）

Oracle XML DB のリソースへのアクセス制御権限を付与されたエンティティ。Oracle XML DB は、次のプリンシパルをサポートする。

- データベース・ユーザー。
- データベース・ロール。データベース・ロールは、グループとして認識される。たとえば、DBA ロールは、DBA ロールを付与されたすべてのユーザーの DBA グループを表す。

LDAP サーバーからインポートされたユーザーおよびロールも、データベースの通常の認証モデルの一部としてサポートされる。

プロローグ（prolog）

XML 宣言および DTD、またはドキュメントを処理するために必要な他の宣言を含む、XML 文書の最初の部分。

文字データ（character Data: CDATA）

ドキュメント内の解析対象外のテキストは、CDATA セクションに格納される。これによって、&、<、> などの、他に特別な機能を持つ文字を含めることができる。CDATA セクションは、要素の内容または属性内で使用できる。

ユーザー・インタフェース（user interface: UI）

メニュー、スクリーン、キーボード・コマンド、マウス・クリック、およびユーザーによるソフトウェア・アプリケーションとの対話方法を定義するコマンド言語の組合せ。

要素（element）

XML 文書の基本論理単位。子、データ、属性とその値などの他の要素に対するコンテナとして機能する。要素は、<name> などの開始タグおよび </name> などの終了タグ、または空要素の場合、<name/> によって識別される。

ラージ・オブジェクト（Large Object: LOB）

内部 LOB および外部 LOB に分割された SQL データ型のクラス。内部 LOB には BLOB、CLOB および NCLOB が含まれ、外部 LOB には BFILE が含まれる「BFILE」、「バイナリ・ラージ・オブジェクト（Binary Large Object: BLOB）」および「キャラクタ・ラージ・オブジェクト（Character Large Object: CLOB）」を参照。

ラッパー（Wrapper）

通常、汎用またはオブジェクト・インタフェースを提供するために、他のデータまたはソフトウェアをラップするデータ構造またはソフトウェアを示す用語。

リスナー（listener）

入力プロセスを監視する個別のアプリケーション・プロセス。

リソース（resource）

リポジトリ階層内のオブジェクト。

リソース名（resource name）

親フォルダ内のリソースの名前。リソース名は、フォルダ内で一意（大 / 小文字が区別されない場合もある）である必要がある。リソース名のキャラクタ・セットは、常に UTF-8（NVARCHAR）である。

リポジトリ（repository）

パス名にマップされる、スキーマ内の一連のデータベース・オブジェクト。リポジトリには、それぞれパス名を持つ、一連のリソースを含む 1 つのルート（「/」）がある。

ルート要素（root element）

XML 文書内にある他のすべての要素を囲む要素。オプションのプロローグとエピローグの間に存在する。XML 文書には、1 つのルート要素のみ置くことができる。

レンダラ（renderer）

ドキュメントを指定された形式に出力するソフトウェア・プロセッサ。

ワーキング・グループ（Working Group: WG）

特定のインターネット・テクノロジ分野における勧告を実行する業界のメンバーで構成された W3C の委員会。

索引

記号

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

R

S

T

U

V

W

X

あ

い

え

お

か

き

く

け

こ

さ

し

す

せ

そ

ち

つ

て

と

な

ね

は

ひ

れ

ろ