DB2® ユニバーサル・データベース

XML エクステンダー 管理およびプログラミングの手引き

バージョン 7

DB2® ユニバーサル・データベース

XML エクステンダー 管理およびプログラミングの手引き

バージョン 7

ご注意!

本書、および本書がサポートする製品をご使用になる前に、 313ページの『特記事項』にある一般的な情報を必ずお読みください。

本書において、日本では発表されていない IBM 製品 (機械およびプログラム)、プログラミング、またはサービスについて言及または説明する場合があります。しかし、このことは、弊社がこのような IBM 製品、プログラミング、またはサービスを、日本で発表する意図があることを必ずしも示すものではありません。

本マニュアルに関するご意見やご感想は、次の URL からお送りください。今後の参考にさせていただきます。

http://www.ibm.com/jp/manuals/main/mail.html

なお、日本 IBM 発行のマニュアルはインターネット経由でもご購入いただけます。詳しくは

http://www.ibm.com/jp/manuals/ の「ご注文について」をご覧ください。

(URL は、変更になる場合があります)

原 典:	IBM® DB2® Universal Database XML Extender Administration and Programming Version 7
発 行:	日本アイ・ビー・エム株式会社
担 当:	ナショナル・ランゲージ・サポート

第1刷 2000.12

この文書では、平成明朝体™W3、平成明朝体™W9、平成角ゴシック体™W3、平成角ゴシック体™W5、および平成角ゴシック体™W7を使用しています。この(書体*)は、(財) 日本規格協会と使用契約を締結し使用しているものです。フォントとして無断複製することは禁止されています。

注* 平成明朝体™W3、平成明朝体™W9、平成角ゴシック体™W3、
平成角ゴシック体™W5、平成角ゴシック体™W7

目次

表

本書について

この節では、以下の情報を示します。

- 『本書の対象読者』
- 『本書の使用方法』
- xiページの『強調表示の表記規則』
- xiiページの『構文図の読み方』
- xivページの『関連情報』

本書の対象読者

本書は以下の読者を対象としています。

- DB2 アプリケーション内で XML データを扱う方で、XML の概念を理解している方。本書の読者には、XML および DB2 についての一般的な知識が必要です。 XML および関連トピックについてさらに知るためには、以下の Web サイトを参照してください。

  http://www.w3c.org/XML

  DB2 についてさらに知るためには、以下の Web サイトを参照してください。

  http://www.ibm.com/software/data/db2/library
- DB2 管理の概念および技法を理解している DB2 データベース管理者。
- SQL、および DB2 アプリケーションに使用できる 1 つ以上のプログラム言語を理解している DB2 アプリケーション・プログラマー。

本書の現行バージョンを入手する方法

本書の最新バージョンは、以下の XML エクステンダー Web サイトから入手できます。

http://www.ibm.com/software/data/db2/extenders/xmlext/library.html

本書の使用方法

本書は以下のように構成されています。

第 1 部 概要

ここでは、XML エクステンダーおよびそれをビジネス・アプリケーション内で使用する方法についての概要を示します。これにはインストールと使用開始に役立つ、入門用のシナリオが含まれています。

第 2 部 管理

ここでは、XML データ用に DB2 データベースを準備して保守する方法を示します。 XML データを含む DB2 データベースを管理する必要がある場合、ここをお読みください。

第 3 部 プログラミング

ここでは、XML データの管理方法を示します。 DB2 アプリケーション・プログラム内で XML データにアクセスして操作する必要がある場合、ここをお読みください。

第 4 部 参照情報

ここでは、XML エクステンダー管理コマンド、ユーザー定義タイプ、ユーザー定義の関数、およびストアード・プロシージャーの使用方法を示します。さらに、XML エクステンダーが発行するメッセージおよびコードをリストします。 XML エクステンダーの概念およびタスクを熟知している場合に、ユーザー定義タイプ (UDT)、ユーザー定義関数 (UDF)、コマンド、メッセージ、メタデータ表、制御表、またはコードについての情報が必要であれば、第 4 部をお読みください。

第 5 部 付録

付録では、文書アクセス定義の DTD、例および入門用シナリオのサンプル、および他の IBM XML 製品について説明します。

本書における DB2 UDB フィックスパック 2 バージョンの新規点

本書には、以下の新規情報または再提供情報を記載しています。

- 管理ウィザードのセットアップおよび開始
- 更新された UDF が XML 文書を更新する方法
- XMLClob UDF が結果を戻す方法
- ストアード・プロシージャー用の CLOB 制限の引き上げ
- DAD 要件には、以下の変更があります。
  - 新規 XML 文書の作成時のスタイル・シート処理命令の組み込み
  - 1 つだけの表を指定した場合の、最初の RDB ノードに対する欠落または空条件の使用
- JDBC でのパラメーター・マーカーのキャスト方法

- 以下のコード・ページの処理
  - 正しいエンコード宣言
  - 移行前提事項
  - 変換のシナリオ
- XML エクステンダーのパラメーター制限の付録

変更情報には、垂直変更バー 『|』 がマークされています。

このフィックスパックのすべての障害修正項目をお知りになりたい場合は、README ファイルを参照してください。

XML エクステンダー Web サイトのサポート・ページをチェックすると、FAQ および既知の問題についてのリストから、一般的な質問に対する修正やソリューションについての追加情報を参照することができます。そのページのアドレスは、http://www.ibm.com/software/data/db2/extenders/xmlext/support.html です。

この資料を DB2 UDB バージョン 7 情報センターに組み込む

XML エクステンダーの資料を、以下の手順で DB2 情報センターに組み込むことができます。

- この資料の HTML ファイルすべてを、インストールしてある XML 製品のご使用の言語の DOC サブディレクトリーから、ご使用の言語の DB2 UDB 資料ディレクトリーの以下のサブディレクトリーにコピーします。

  Windows の場合、情報センターのディレクトリーは sqllib¥doc¥html¥db2sx です。

  UNIX の場合、情報センターのディレクトリーは *install directory*/doc/html/db2sx です。
- 情報センターを再起動すると、この資料が**「資料 (Books)」**タブに組み込まれます。

強調表示の表記規則

本書では、以下の表記規則を使用します。

太字 (Bold)

太字体のテキストは、以下のものを示します。

- コマンド
- フィールド名
- メニュー名

	• プッシュボタン
イタリック *(Italic)*	イタリック体のテキストは、以下のものを示します。 • 値で置き換える変数パラメーター • 強調された語 • 用語集の用語が最初に使用される個所
英大文字 (UPPERCASE)	英大文字は、以下のものを示します。 • データ・タイプ • 列名 • 表名
例 (Example)	例の字体のテキストは、以下のものを示します。 • システム・メッセージ • 入力する値 • コーディング例 • ディレクトリー名 • ファイル名 • パス名

構文図の読み方

本書全体を通して、コマンドおよび SQL ステートメントの構文は、構文図を使用して説明されます。

構文図は以下のように読んでください。

- 構文図は左から右へ、上から下へ、線に沿って読みます。

  ▶▶── 記号は、ステートメントの開始を示します。

  ──▶ 記号は、構文図が次の行に続くことを示します。

  ▶── 記号は、ステートメントが前の行から続いていることを示します。

  ──▶◀ 記号は、ステートメントの終了を示します。

  完全なステートメント以外の、構文単位の図は、 ▶── 記号から開始して ──▶ 記号で終了します。
- 必須項目は、水平線 (メインパス) の線上に示されます。

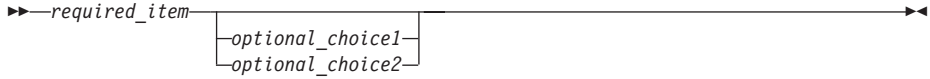
- オプション項目は、メインパスの下に示されます。

オプション項目がメインパスの上に示されている場合、その項目は読みやすくするためだけに示されているのであり、ステートメントの実行には影響を与えません。

- 2 つ以上の項目から選択できる場合、それらの項目はスタックに入れられて縦に並べられます。

  項目から 1 つを選択*しなければならない* 場合、スタック内の項目の 1 つがメインパスの線上に示されます。

  項目の 1 つを選択することが任意である場合、スタック全体がメインパスの下に示されます。



  項目の 1 つがデフォルト値である場合、その項目はメインパスの上に示されて、残りの項目は下に示されます。

- メインの線の上にある、左に戻る矢印は、項目を反復して指定できることを示します。

  反復の矢印に句読点が含まれる場合、反復する項目を指定の句読点で区切らなければなりません。

スタックの上にある反復の矢印は、そのスタック内の項目を反復できることを示します。

- キーワードは、英大文字 (FROM など) で示されます。 XML エクステンダーでは、キーワードを大文字でも小文字でも指定できます。キーワードではない用語は、英小文字 (*column-name* など) で示されます。それらは、ユーザーが供給する名前または値を示します。
- 句読記号、括弧、算術演算子、または同様の他の記号が示されている場合、それらを構文の一部として入力しなければなりません。

関連情報

XML エクステンダーおよび関連製品を使用する際に、以下の資料が役立つことがあります。

資料	資料番号	説明
DB2 コール・レベル・インターフェースの手引きおよび解説書	SC88-8517	CLI を使用して DB2 サーバーにアクセスするアプリケーションの作成方法を示します。
DB2 アプリケーション開発の手引き	SC88-8516	アプリケーション開発のプロセスについて説明し、組み込み SQL および API を使用してデータベースにアクセスするアプリケーション・プログラムをコーディング、コンパイル、および実行する方法について解説します。
DB2 エクステンダー・ページ	資料番号なし	DB2 エクステンダー、およびそのエクステンダーに関連した技術についての情報を示します。 DB2 エクステンダーのページの Web アドレスは、http://www.software.ibm.com/data/db2/extenders です。

資料	資料番号	説明
DB2 ユニバーサル・データベース SQL 解説書 (第 1 巻、第 2 巻)	• 第 1 巻: SC88-8540 • 第 2 巻: SC88-8657	SQL の構文、セマンティクス、およびこの言語の規則を説明しています。また、リリース間の非互換性、製品の制限事項、およびカタログ視点についての情報も提供しています。
• *DB2 ユニバーサル・データベース 管理の手引き: インプリメンテーション* • *DB2 ユニバーサル・データベース 管理の手引き: パフォーマンス* • *DB2 ユニバーサル・データベース 管理の手引き: 計画*	• インプリメンテーション: SC88-8511 • パフォーマンス: SC88-8512 • 計画: SC88-8513	DB2 データベースの設計、実装、および保守を行う方法を示します。
DB2 ユニバーサル・データベース イメージ、オーディオ、およびビデオ・エクステンダー 管理およびプログラミングの手引き	SC88-8609	イメージ、オーディオ、およびビデオ・データを扱う DB2 データベースを管理する方法を示します。さらに、エクステンダーに備わっているアプリケーション・プログラミング・インターフェースを使用して、それらのタイプのデータにアクセスし、操作する方法を示します。
DB2 ユニバーサル・データベース テキスト・エクステンダー 管理およびプログラミング	SC88-8610	テキストを扱う DB2 データベースを管理する方法を示します。さらに、エクステンダーに備わっているアプリケーション・プログラミング・インターフェースを使用して、それらのタイプのデータにアクセスし、操作する方法を示します。

第1部 概要

ここでは、XML エクステンダーおよびそれをビジネス・アプリケーション内で使用する方法についての概要を示します。

第1章 XML エクステンダーの概要

IBM® DB2® エクステンダー™・ファミリーは、伝統的なデータと非伝統的なデータの両方を処理するための、データおよびメタデータ管理のソリューションを提供します。 XML エクステンダーは、IBM の DB2 ユニバーサル・データベース (DB2 UDB)™ と XML の柔軟性を統合するのに役立ちます。

DB2 の XML エクステンダーには、XML 文書の保管とアクセスを行ったり、既存のリレーショナル・データから XML 文書を生成したり、 XML 文書をリレーショナル・データに細分化したりする (保管されているタグなしのエレメントまたは属性内容を分解する) 機能が備わっています。また XML エクステンダーには、DB2 で XML データを管理するための新しいデータ・タイプ、関数、およびストアード・プロシージャーも備わっています。

XML エクステンダーは、次のようなオペレーティング・システムで使用することができます。

- Windows NT
- AIX
- Sun Solaris
- Linux
- NUMA-Q

XML 文書

コンピューター産業には多数のアプリケーションがあり、それぞれに固有の長所と短所があります。今日のユーザーは、個々のタスクに最適なアプリケーションを選択することができます。しかし、別々のアプリケーションでデータが共用されることはよく行われており、データを他のアプリケーションにインポートするために、別の形式に複製、変換、エクスポート、または保管するという課題が絶えず発生しています。その場合、そのような変換プロセスの多くで、データの部分的な喪失が不可避であったり、またはデータの整合性を保つための面倒なプロセスの実行が最低限必要であったりするので、ビジネス・アプリケーションにおける重大な問題を生じることがあります。これは、時間と経費の浪費につながります。

今日、この問題に対処する方法の 1 つは、アプリケーション開発者が *Open Database Connectivity (ODBC)* アプリケーションを作成して、データをデータ

ベース管理システムに保管するようにすることです。そこからデータを操作して、他のアプリケーションの規定の形式で提供することができます。短期間で変更されてすぐに古くなってしまうアプリケーションの場合でも、その規定の形式にデータを変換できるように、データベース・アプリケーションを作成しておく必要があります。データを HTML に変換するアプリケーションでは表示ソリューションが用意されていますが、その表示用のデータを、他の用途に使えることはまずありません。データを表示から分離する何らかの手段があったら、その手段を使って、アプリケーション相互でデータをやりとりすることができるはずです。

XML は、この問題に対処するために開発されました。 XML は、*eXtensible Markup Language (拡張可能マークアップ言語)* の頭字語です。拡張可能と呼べるのは、その言語自体がメタ言語であり、企業の必要に応じて独自の言語を作成できるからです。 XML を使用すると、特定のアプリケーション用のデータだけでなく、データ構造をも取り込むことができます。 XML は唯一の交換形式というわけではありません。しかし、XML はデータ交換のための受け入れられる規格となってきました。この規格に従えば、それぞれのアプリケーションは、適切な形式を使ったデータ変換を行わなくても、互いにデータを共用できるようになります。

XML アプリケーション

XML はデータ交換のための受け入れられる規格となっているので、それを活用できる多数のアプリケーションが出現しています。

特定のプロジェクト管理アプリケーションを使用していて、そのデータの一部を予定表アプリケーションにも使用したいと仮定します。 XML を使用すると、このことを簡単に行えます。互いに連結している今日の世界でアプリケーションのベンダーが競争してゆくためには、アプリケーションに XML 交換ユーティリティーを組み入れることが必要になっています。そのためここの例では、プロジェクト管理アプリケーションは XML でタスクをエクスポートし、それをそのまま予定表アプリケーションにインポートすることができます (ただし、認められている DTD に情報が準拠していなければなりません)。

XML と DB2 の使用理由

XML には、データ交換のための標準形式が用意されていて、それによって多くの問題が解決されますが、解決すべき問題はまだ他にもあります。企業のデータ・アプリケーションを構築するとき、以下のような質問に答えなければなりません。

- どれほどの頻度でデータを複製するか。
- どのような種類の情報をアプリケーション間で共用する必要があるか。
- どのようにして必要な情報を素早く検索できるか。
- 新規項目が追加されるなどの特定のアクションが、全アプリケーション間の自動データ交換を起動するようにするにはどうすればよいか。

これらの種類の問題は、データベース管理システムによってのみ解決できます。 XML 情報およびメタ情報をデータベースに直接組み入れることによって、他のアプリケーションが特定の目的のために必要とする XML 結果を、より直接的に (そしてより早く) 取得することができます。そこで、XML エクステンダーが役立ちます。 XML エクステンダーを使用すると、多くの XML アプリケーションで DB2 の能力を生かすことができます。

DB2 データベース内にある構造化 XML 文書の内容を使用して、構造化 XML 情報と伝統的リレーショナル・データとを結合することができます。アプリケーションに応じて、 XML 文書全体を非伝統的なユーザー定義データ・タイプとして DB2 内に保管するか、または XML の内容を伝統的なデータとしてリレーショナル表内にマップするかを選択できます。非伝統的な XML データ・タイプの場合、DB2 UDB テキスト・エクステンダーに備わっている構造化テキスト検索の他に、 XML エレメントまたは属性値の豊富なデータ・タイプを検索する機能が XML エクステンダーに付け加えられます。

XML エクステンダーを使うと、アプリケーションで以下のことが行えます。

- 検索対象の任意の XML エレメントまたは属性値を*サイド表* に抽出する一方で、XML 文書全体を、アプリケーション表内の*列データ* として保管したり、外部的にローカル・ファイルとして保管したりします。 XML 列方式を使って以下のことを行うことができます。
  - サイド表に抽出されて索引付けされている SQL 汎用データ・タイプ の XML エレメントまたは属性を高速検索します。
  - *XML エレメント* の内容、または *XML 属性* の値を更新します。
  - SQL 照会を使って XML エレメントまたは属性を動的に抽出します。
  - 追加と更新の際に XML 文書の妥当性を検査します。
  - テキスト・エクステンダーを使って構造化テキスト検索を行います。
- XML コレクション保管方式およびアクセス方式を使って、 1 つ以上のリレーショナル表から成る XML 文書の内容を合成または分解します。

DB2 への XML の組み込み

XML エクステンダーには、 DB2 を使って XML データを管理および活用するのに役立つ次のような機能が用意されています。

- リレーショナル表への XML データの組み込みを管理するのに役立つ管理ツール
- XML データの保管方式と使用方式
- XML データの検査に使用する DTD を保管するための DTD リポジトリー (DTD_REF)
- リレーショナル・データに対する XML 文書のマップ用の、文書アクセス定義 (DAD) と呼ばれるマッピング方式

管理ツール

XML エクステンダーの管理ツールを使うと、データベースおよび表列を XML 対応にしたり、 DB2 リレーショナル構造に対して XML データをマップしたりするのに役立ちます。 XML エクステンダーには、管理タスクを実行するためのアプリケーションを自分で開発したり、単にウィザードを使用したりするためのいくつかの管理ツールがあります。次のようなツールを使用して、XML エクステンダーの管理タスクを実行できます。

- XML エクステンダーの管理ウィザードには、管理タスク用のグラフィカル・ユーザー・インターフェースが備わっています。
- **dxxadm** コマンドには、管理タスク用のコマンド行オプションが備わっています。
- XML エクステンダーの管理ストアード・プロシージャーには、管理タスク用のアプリケーション開発オプションが用意されています。

保管およびアクセスの方式

XML エクステンダーには、XML 列および XML コレクションという、 DB2 に XML 文書を組み込むための 2 つの保管およびアクセスの方式が備えられています。これらの方式は、それぞれかなり異なる用途に使われますが、同一のアプリケーション内で使用することができます。

XML 列

この方式は、変更前の XML 文書を DB2 に保管するのに役立ちます。 XML 列は、文書をアーカイブするのに適した働きをします。 XML 対応の列に文書を挿入した後、これを更新、取り出し、および検索することができます。エレメントおよび属性のデータを DB2 表 (サイド表) にマップしてから、これに索引を付けて高速構造検索を行えるようにすることができます。

XML コレクション

この方式は、DB2 表に対して XML 文書をマップするのに役立ちます。それによって、既存の DB2 データから XML 文書を合成したり、 XML 文書を DB2 データに分解 (タグなしのエレメントまたは属性の内容を保管) したりすることができます。この方式は、データ交換のアプリケーションに適しています。 XML 文書の内容を頻繁に更新する場合は特にそうです。

DTD リポジトリー

XML エクステンダーには、XML の*文書タイプ定義 (DTD)* リポジトリー (複数の XML エレメントおよび属性の宣言の集合) が用意されています。データベースが XML 用に*使用可能にされる* と、 *DTD 参照表 (DTD_REF)* が作成されます。この表の各行は、追加のメタデータ情報のある DTD を表します。ユーザーはこの表にアクセスして、独自の DTD を挿入できます。 DTD_REF 表内の *DTD* を使用して XML 文書の妥当性検査を行います。

文書アクセス定義 (DAD)

構造化 XML 文書の処理方法は、*文書アクセス定義 (DAD)* に指定します。DAD 自体は、XML に形式化された文書です。これは、XML 列または XML コレクションの使用時に XML 文書構造を DB2 データベースに関連付けます。 XML コレクションの場合とは対照的に、XML 列の定義時には、DAD の構造は異なります。

DAD ファイルの管理は、データベースを XML 対応にするときに作成した XML_USAGE 表を使って行います。

XML 列: 構造化文書の保管と検索

XML には文書のセットを作成するために必要な情報がすべて含まれるので、文書構造を現状のままで保管して保守したい場合があるでしょう。

たとえば、Web に記事を供給するニュース発行会社であれば、発行済みの記事のアーカイブを保守したいと思うことがあります。そのようなシナリオでは、XML エクステンダーによって XML 記事の全部または一部を DB2 表の列に保管することができます。 8ページの図1 に示すように、この種類の XML 文書記憶を *XML 列* と呼びます。

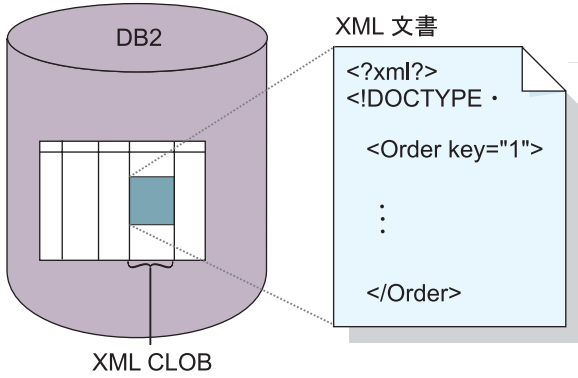


図 1. 構造化 XML 文書を DB2 表の列に保管する

XML エクステンダーには、XML 列で使用する数種類のユーザー定義タイプ (UDT) が備わっています。

- XMLVarchar
- XMLCLOB
- XMLFILE

XML エクステンダーの UDT にはすべて、接頭部 db2xml が付いています。これは DB2 XML エクステンダー UDT のスキーマ名 です。これらのデータ・タイプは、アプリケーション表内にある XML 文書のストレージ・タイプを識別するために使用されます。 XML エクステンダーは、従来のフラット・ファイルをサポートします。 XML 文書を DB2 内に保管する必要はありません。さらに、XML 文書をローカル・ファイル名の指定にしたがって、*ローカル・ファイル・システム* として保管することもできます。

DB2 XML エクステンダーには強力な*ユーザー定義機能 (UDF)* が備わっていて、 XML 列内の XML 文書を検索したり、XML エレメントまたは属性値を抽出することができます。 UDF は、データベース管理システムに定義して、その後 SQL 照会で参照できるようになる機能です。 XML エクステンダーには次のようなタイプの UDF が備えられています。

- 保管: XML データ・タイプの XML 対応列に、未変更の XML 文書を保管します。
- 抽出: XML 文書または値指定のエレメントおよび属性をデータ・タイプとして抽出します。
- 更新: XML 文書全体または指定のエレメントと属性値を更新します。

抽出機能を使うと、汎用 SQL データ・タイプで高効率の検索を行うことができます。さらに、DB2 UDB テキスト・エクステンダーを XML エクステンダーと共に使用して、構造化テキスト検索と全テキスト検索 を XML 文書内のテキストで実行できます。この強力な検索機能を使用して、新聞の記事や*電子データ交換 (EDI)* アプリケーションなどの、頻繁に検索可能なエレメントまたは属性を持つ読み取り可能テキストを大量に発行する Web サイトを使いやすくすることができます。

XML エクステンダーの UDF にはすべて、接頭部 db2xml が付いています。これは DB2 XML エクステンダー UDF のスキーマ名です。 UDF は XML UDT に適用されて、XML 列のために使用されます。

ロケーション・パス

ロケーション・パス とは、 XML エレメントまたは属性を識別する一連の *XML タグ* です。 XML エクステンダーはロケーション・パスを使って XML 文書の構造を識別し、エレメントまたは属性のコンテキストを示します。単一スラッシュ (/) のパスは、コンテキストが文書全体であることを示します。ロケーション・パスは以下の状況で使用されます。

- UDF の抽出時に抽出対象のエレメントおよび属性を識別するため
- DAD での XML 列の索引付け体系の定義の際に、 XML エレメント (または属性) と DB2 列とのマッピング・ファイルを指定するため
- 構造化テキストの検索でのテキスト・エクステンダーの使用時に XML エレメントまたは属性を識別するため

10ページの図2 は、ロケーション・パスおよび XML 文書との関係を示しています。

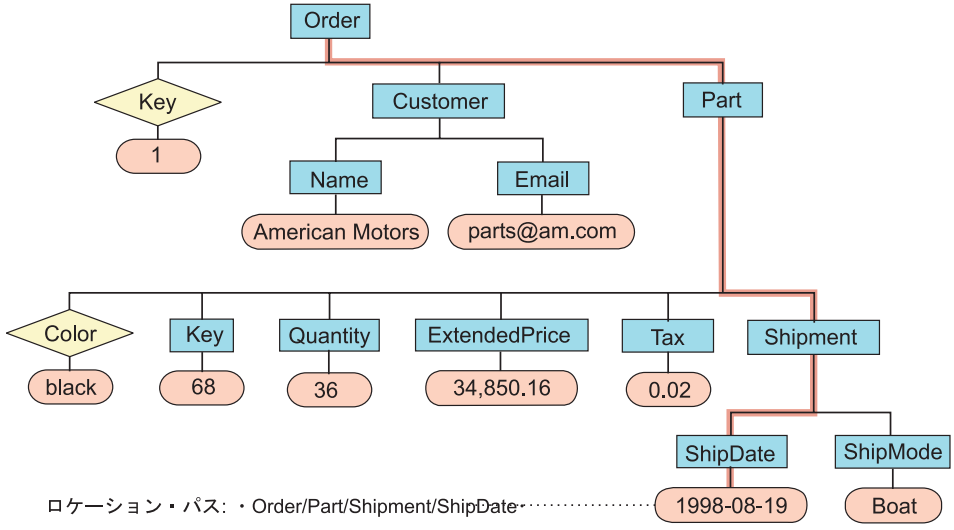


図 2. 文書を構造化 XML 文書として DB2 表の列に保管する

ロケーション・パスを指定するために、 XML エクステンダーは *XML Stylesheet Language Transformation (XSLT)* および *XML Path 言語 (XPath)* のサブセットを使用します。本書では、XPath の仕様で定義されている用語である*ロケーション・パス* を使用します。ロケーション・パスとは、XML エレメントまたは属性を識別する一連の XML タグです。本書ではさらに、XPath の仕様で指定されている、*絶対ロケーション・パス* の XSLT または XPath の省略形の構文も使用します。絶対ロケーション・パスとは、オブジェクトの全パス名です。

XSLT は、XML 文書を他の XML 文書に変換するための言語です。これは XML のスタイル・シート言語である *XSL (XML Stylesheet Language)* の一部として使用されるように設計されています。 XSLT に加えて、XSL にはフォーマット化を指定するための XML 用語も含まれています。 XSL は XSLT を使用して XML 文書のスタイル設定を指定し、フォーマット化の語いを使用する他の XML 文書に変換する方法を示します。

XPath は、XSLT が使用するために設計された、XML 文書のアドレス部を指定する言語です。すべてのロケーション・パスは、XPath 用に定義された構文を使用して表現できます。

XSLT および XPath についての詳細は、以下の Web ページを参照してください。

- XSLT については、http://www.w3.org/TR/WD-xslt

- XPath については、http://www.w3.org/TR/xpath

構文と制約事項については、58ページの『ロケーション・パス』を参照してください。

XML 列の用語

ここでは、XML の概念と、本書で使用される用語について解説します。

文書アクセス定義 (DAD)
XML 列では、構造化照会のために索引付けされる DB2 サイド表に対する XML 文書構造のマッピング。

DXX_INSTALL
XML エクステンダーのインストール・ディレクトリー。

サイド表
XML 列内でエレメントまたは属性を検索するときのパフォーマンスを改善するために XML エクステンダーが作成する追加の表。

XML 列
XML データ・タイプの DB2 列を使用可能にし、使用可能になった列に原形の XML 文書を保管することで XML 文書を保管およびアクセスする方式。管理ツールの 1 つを使って XML 対応にされた列を指すこともあります。

XML 表
管理ツールの 1 つを使って XML 対応にされた 1 つ以上の列の入ったアプリケーション表。

XML UDF
XML エクステンダーに備わっている DB2 ユーザー定義関数。

XML UDT
XML エクステンダーに備わっている DB2 ユーザー定義タイプ。

XML コレクション: 統合データ管理

伝統的な SQL データは、着信した XML 文書から*分解される* か、またはそれが*構成されて* 発信する文書が作成されます。データを他のアプリケーションと共用する場合に、 DB2 のリレーショナル機能の利点を生かすには、やりとりする XML 文書を必要に応じて合成および分解してデータを管理できるとよいと考えられます。このような XML 文書記憶を、*XML コレクション* と呼びます。

XML コレクションの例を 12ページの図3 に示してあります。

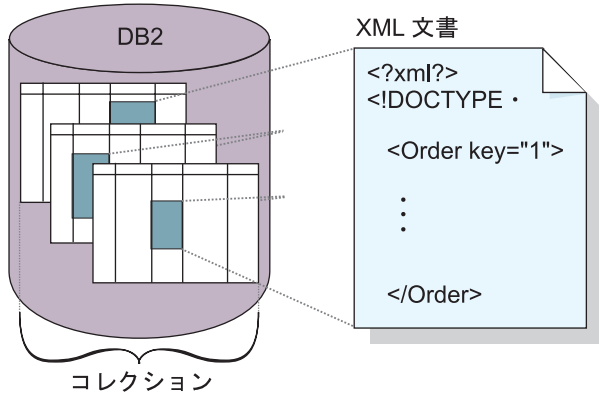


図 3. タグなしのデータとしての文書の DB2 表内での保管

XML コレクションは、DAD ファイルに定義しますが、これは、 1 つ以上のリレーショナル表に対するエレメントおよび属性のマップの方法を指定するものです。コレクション名を使用可能にすることでこれを定義してから、ストアード・プロシージャーと一緒に使用して、XML 文書を合成または分解することができます。

DAD ファイル内にコレクションを定義する場合、 SQL マッピングまたは RDB_node マッピングの 2 つのマッピング体系のうちのいずれかを使います。SQL マッピングでは、SQL SELECT ステートメントを使って、コレクションに使用する DB2 表と条件を定義します。 RDB_node マッピングでは、XPath ベースの RDB_node を使って、表、列、および条件を定義します。

XML 文書の合成または分解用のストアード・プロシージャーが用意されています。ストアード・プロシージャーは接頭部 db2xml を使います。これは XML エクステンダーのスキーマ名 です。 XML コレクションには次のようなストアード・プロシージャーを使います。

- 合成
  - dxxGenXML(): XML コレクション用の DAD ファイルを使って XML 文書を合成します。
  - dxxRetrieveXML(): 使用可能にされた XML コレクションを使って XML 文書を合成します。
- 分解
  - dxxShredXML(): XML コレクション用の DAD ファイルを使って XML 文書を分解します。

– dxxInsertXML(): 使用可能にされた XML コレクションを使って XML 文書を分解します。

XML コレクションの用語

以下の用語は XML エクステンダーに固有のものであり、本書で頻繁に使用されます。

合成 (composition)
DAD ファイルに定義されているとおりに、既存のリレーショナル・データから XML 文書を生成すること。

分解 (decomposition)
DAD ファイルに定義されているとおりに、タグなしのリレーショナル・データとして XML 文書を保管すること。

文書アクセス定義 (DAD)
XML コレクションでは、 XML 文書の合成または分解用に DB2 データ構造に対して XML 文書をマッピングすること。

DXX_INSTALL
XML エクステンダーのインストール・ディレクトリー。

XML コレクション
XML データを保管し、そこにアクセスするために、リレーショナル表のセットを使用する方式。タグなしのデータを合成して XML 文書にしたり、 XML 文書からこのデータを分解したりすることができます。また、XML 文書に合成したりこの文書から分解したりする表セットも指します。

XML ストアード・プロシージャー (XML stored procedures)
XML 文書を合成または分解するストアード・プロシージャー。

第2章 XML エクステンダー入門

この章では、XML エクステンダーを使用してアプリケーション用の XML データにアクセスし、それを変更する方法を示します。ここに示されたチュートリアル学習に従って、準備されたサンプル・データを使用してのデータベースのセットアップ、 SQL データの XML 文書へのマップ、XML 文書のデータベースへの保管、および XML 文書からデータを検索および抽出することができます。

管理についての演習では、管理コマンドと一緒に DB2 コマンド行を使用します。これらのタスクは、本書で説明されている XML エクステンダーの管理ウィザードを使用して実行できます。 XML データ管理の学習では、 XML エクステンダーに備わっている UDF およびストアード・プロシージャーを使用します。本書の残りの部分にある例のほとんどは、この章で使用されるサンプル・データを利用しています。

必須: この章の演習を完了するには、 DB2 UDB バージョン 6.1 以上が必要です。バージョン 6.1 を使用する場合、フィックスパック 2 をインストールする必要があります。さらにこの演習ステップでは、Windows NT® の使用が前提になります。

演習は次のとおりです。

- 原形の XML 文書を DB2 表の列に保管します。
  - 文書の保管先の XML UDT と、頻繁に検索する XML のエレメントおよび属性に関する計画を立てる
  - データベースおよび表をセットアップする
  - データベースを XML に関して使用可能にする
  - DTD を DTD リポジトリー表に挿入する
  - XML 列のための DAD を準備する
  - XML タイプの既存の表に列を追加する
  - XML 用に新規の列を使用可能にする
  - サイド表に索引を作成する
  - XML 列内に XML 文書を保管する
  - XML エクステンダー UDF を使用して XML 列を検索する
- 既存のデータから XML 文書を作成します。

- XML 文書のデータ構造を計画する
- データベースおよび表をセットアップする
- データベースを XML に関して使用可能にする
- XML コレクションの文書アクセス定義 (DAD) ファイルを準備する
- 既存のデータから XML 文書を合成する
- データベースから XML 文書を取り出す

• データベースを終結処理します。

演習のシナリオ

この演習では、販売代理店に自動車やトラックを納入する会社である ACME Auto Direct の社員が想定されています。与えられた任務として 2 つのタスクがあります。まず、営業部から照会できるように SALES_DB データベース内に注文をアーカイブするためのシステムをセットアップします。次のタスクでは、既存の注文データベース SALES_DB から情報を取り出し、かぎとなる情報をその中から抽出して XML 文書に保管します。

演習: XML 列への XML 文書の保管

シナリオ

サービス部門用の営業データをアーカイブするタスクを任務として与えられたとします。そのデータは、同じ DTD を使用する複数の XML 文書内に保管されます。サービス部門は、カスタマーからの要求および苦情を処理する際にそれらの XML 文書を使用します。

サービス部門から、XML 文書の望ましい構造が伝えられ、どのエレメント・データが最も照会頻度が高いと考えられるかが通知されています。 XML 文書は、SALES_DB データベース内の SALES_TAB 表に保管される必要があり、迅速に検索できなければなりません。 SALES_DB 表は、各販売に関するデータを記入する 2 つの列と、 XML 文書が入る 3 番目の列で構成されることになります。この列を ORDER と名付けます。

XML 文書を保管する XML データ・タイプと、どの XML エレメントおよび属性がひんぱんに照会されるかを判別します。次に、XML 用の SALES_DB データベースをセットアップし、 SALES_TAB 表を作成してから、ORDER 列を使用可能にして、原形の文書を DB2 に保管できるようにします。また、妥当性検査のために XML 文書用の DTD を挿入してから、その文書を XMLVARCHAR データ・タイプで保管します。列を使用可能にするときに、

文書アクセス定義 (DAD) ファイル内の文書 (サイド表の構造を指定する XML 文書) の構造検索用に索引を付けるサイド表を定義します。 DAD ファイル、DTD、および XML 文書のサンプルについては、 291ページの『付録B. サンプル』 を参照してください。

SALES_TAB に関しては表1 に説明されています。

表 1. SALES_TAB 表

列名	データ・タイプ
INVOICE_NUM	CHAR(6) NOT NULL PRIMARY KEY
SALES_PERSON	VARCHAR(20)
ORDER	XMLVARCHAR

計画

XML エクステンダーを使用して文書を保管する前に、XML 文書の構造を解明して、文書を検索する方法を決める必要があります。文書の検索方法を計画するときに、以下のことを決める必要があります。

- XML 文書を保管するのに使う XML ユーザー定義タイプ
- パフォーマンスの向上のために索引を付ける対象の、サービス部門がひんぱんに検索する XML エレメントと属性

以下の項に、これらの決定を下す方法について説明します。

XML 文書の構造

この章の演習の XML 文書の構造では、発注キーを使って構造化されている個々の注文に関する情報を最上レベルとし、次にカスタマー、パーツ、および出荷情報を下位レベルとして取り込みます。 XML 文書に関しては 18ページの図4 に説明されています。

またこの演習では、 XML 文書の構造を理解および検査するのに使用するサンプル DTD も用意されています。 DTD ファイルは、291ページの『付録B. サンプル』 にあります。これは、18ページの図4 の中の構造に一致します。

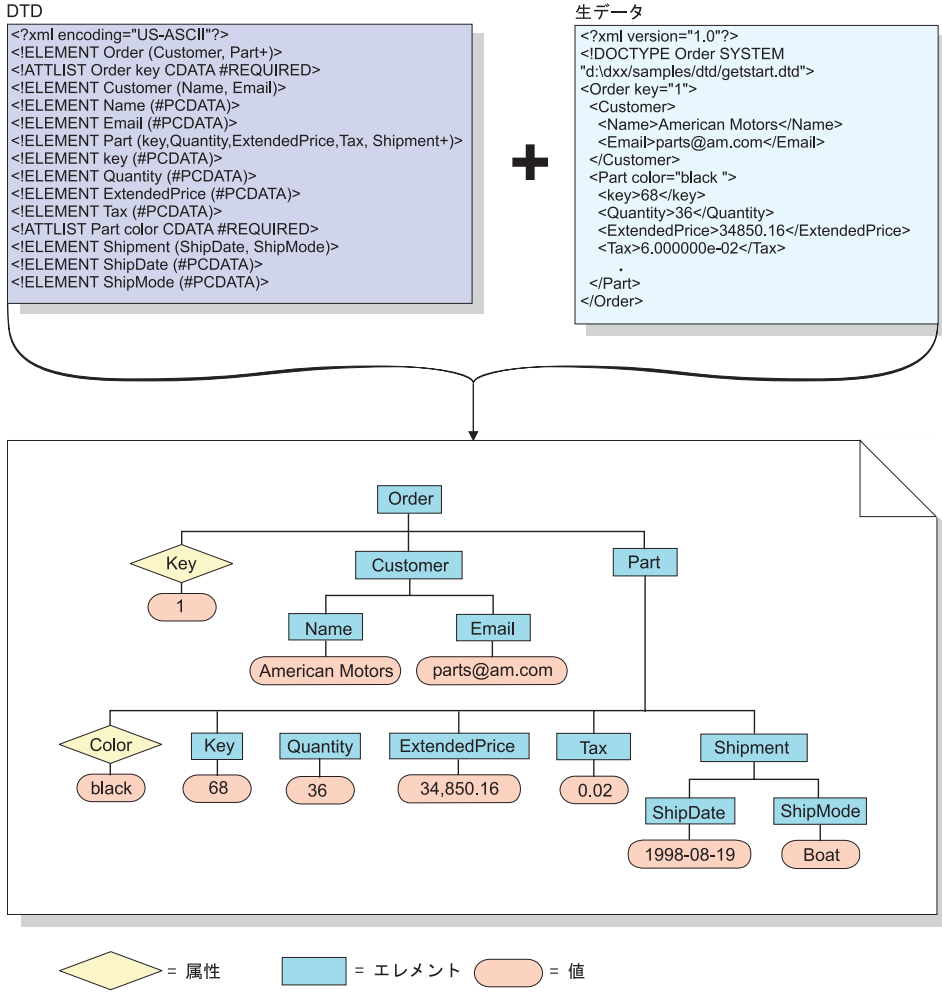


図 4. DTD および XML 文書の階層構造

XML 列の XML データ・タイプの判別

XML エクステンダーには、 XML 文書を入れる列を定義するのに使う XML ユーザー・データ・タイプが備わっています。それらのデータ・タイプは次のとおりです。

- XMLVarchar: 小さい文書を DB2 に保管するのに使用
- XMLCLOB: 大きい文書を DB2 に保管するのに使用
- XMLFILE: DB2 以外に文書を保管するのに使用

この演習では小さい文書を DB2 に保管するので、XMLVarchar データ・タイプを使います。

検索対象のエレメントと属性の判別

XML 文書の構造とアプリケーションの必要事項を理解すれば、どのエレメントと属性を検索すればよいか、また、最も頻繁に検索または抽出されるエレメントと属性、あるいは照会に最も経費のかかるエレメントと属性を判別することができます。サービス部門からの通知によれば、注文キー、顧客名、価格、および注文品の出荷日付が最も頻繁に照会され、その検索での迅速なパフォーマンスが必要です。この情報は、XML 文書構造のエレメントと属性に入ります。 表2 は、各エレメントと属性のロケーション・パスを説明しています。

表 2. 検索するエレメントと属性

データ	ロケーション・パス
注文キー	/Order/@key
カスタマー	/Order/Customer/Name
価格	/Order/Part/ExtendedPrice
出荷日付	/Order/Part/Shipment/ShipDate

サイド表に対する XML 文書のマッピング

このチュートリアルでは、XML 列用の DAD ファイルを作成し、 XML 文書を DB2 内に保管するのにそれを使用します。また、パフォーマンスを向上するため、索引付けに使われる DB2 サイド表に対して XML エレメントと属性をマップします。前の項で、どのエレメントと属性が検索対象になるかを判別しました。この項ではさらに、索引付け可能な DB2 表に対するそのようなエレメントと属性値のマッピングを習得します。

検索するエレメントと属性を確認し終わったら、サイド表内でのそれらの編成の仕方、表の数、およびどの表内にどの列が入るかを判別します。通常、似通った情報を同じ表内に入れてサイド表を編成します。いずれかのエレメントのロケーション・パスを文書内で複数回反復できるかどうかによっても構造は決まります。たとえばここの文書では、パーツ・エレメントが複数回繰り返されることがあるので、価格とエレメントも複数出現する可能性があります。複数出現する可能性のあるエレメントは、それ専用の表に入っていなければなりません。

さらに、どの DB2 基本タイプをエレメントまたは属性値に使用すべきかも判別する必要があります。通常それは、データの形式によって簡単に判別するこ

とができます。データがテキストであれば VARCHAR を選び、データが整数であれば INTEGER を選び、データが日付の場合に範囲検索を行いたければ DATE を選びます。

このチュートリアルではエレメントと属性を、次のようなサイド表にマップします。

ORDER_SIDE_TAB

列名	データ・タイプ	ロケーション・パス	複数出現するかどうか
ORDER_KEY	INTEGER	/Order/@key	いいえ
CUSTOMER	VARCHAR(16)	/Order/Customer/Name	いいえ

PART_SIDE_TAB

列名	データ・タイプ	ロケーション・パス	複数出現するかどうか
PRICE	DECIMAL(10,2)	/Order/Part/ExtendedPrice	はい

SHIP_SIDE_TAB

列名	データ・タイプ	ロケーション・パス	複数出現するかどうか
DATE	DATE	/Order/Part/Shipment/ShipDate	はい

このチュートリアル用に、環境を設定するためのスクリプトのセットが用意されています。そのスクリプトは *DXX_INSTALL*¥samples¥cmd ディレクトリーにあります。ここで、*DXX_INSTALL* は、たとえば c:¥dxx¥samples¥cmd などの、XML エクステンダーをインストールしたドライブおよびディレクトリーです。それは以下のとおりです。

getstart_db.cmd
データベースを作成して、4 つの表にデータを取り込みます。

getstart_prep.cmd
データベースを XML エクステンダーのストアード・プロシージャーと DB2 CLI にバインドします。

getstart_insertDTD.cmd
XML 文書を妥当性検査するのに使用する DTD を XML 列に挿入します。

getstart_createTabCol.cmd
XML 対応の列をもつアプリケーション表を作成します。

getstart_alterTabCol.cmd
XML 対応にする列の追加によってアプリケーション表を更新します。

getstart_enableCol.cmd
XML 列を使用可能にします。

getstart_createIndex.cmd
XML 列用にサイド表に索引を作成します。

getstart_insertXML.cmd
XML 文書を XML 列に挿入します。

getstart_queryCol.cmd
アプリケーション表に対して SELECT ステートメントを実行し、XML 文書を戻します。

getstart_clean.cmd
チュートリアル環境を終結処理します。

セットアップ

この項では、XML エクステンダーで使用できるようデータベースを準備します。以下のことを行います。

1. データベースを作成する
2. データベースを使用可能にする

データベースの作成

この節では、コマンドを使用してデータベースをセットアップします。このコマンドは、サンプルのデータベースを作成して、それに接続し、データを保持するための表を作成して、データを挿入します。

データベースを作成するには、以下のように行います。

1. *DXX_INSTALL*¥samples¥cmd ディレクトリーに変更します。ここで、*DXX_INSTALL* は XML エクステンダーをインストールしたドライブおよびディレクトリーです。この演習ではディレクトリーとして c:¥dxx を使用します。別のドライブおよびディレクトリーを使用する場合、この値を変更してください。

2. Windows NT の「スタート」メニューから「DB2 コマンド・ウィンドウ (DB2 Command Window)」をオープンするか、または Windows NT のコマンド・プロンプトに以下のコマンドを入力します。

   ```
   DB2CMD
   ```

3. DB2 コマンド・ウィンドウから、以下のコマンドを実行します。

   ```
   getstart_db.cmd
   ```

データベースの使用可能化

XML 情報をデータベースに保管するには、それを XML エクステンダーに関して使用可能にしなければなりません。データベースを XML に関して使用可能にするとき、XML エクステンダーは以下の事柄を行います。

- すべてのユーザー定義タイプ (UDT) およびユーザー定義関数 (UDF) を作成します。
- 制御表を作成して、そこに XML エクステンダーが必要とするメタデータを取り込みます。
- db2xml スキーマを作成して、必要な特権を割り当てます。

データベースを XML に関して使用可能にするには、以下のように行います。

DB2 コマンド・ウィンドウから以下のスクリプトを実行して、 SALES_DB データベースを使用可能にします。

```
getstart_prep.cmd
```

このスクリプトは、データベースを XML エクステンダーのストアード・プロシージャーおよび DB2 CLI にバインドします。さらに、**dxxadm** コマンド・オプションを実行して、以下のデータベースを使用可能にします。

```
dxxadm enable_db SALES_DB
```

XML 列の作成

XML エクステンダーには、データベース内の XML 文書全体の保管とアクセスのための方式が備わっています。それは XML 列と呼ばれます。 XML 列方式を使用すると、XML ファイル・タイプを使用して文書を保管し、サイド表内の列に索引付けし、その後で XML 文書を照会または検索することができます。この保管方式は、文書が頻繁に更新されないアーカイブ・アプリケーションで特に役立ちます。このチュートリアルの目的上、用意されている XML 文書を XML 列に保管します。

この演習では、文書を SALES_TAB 表に保管します。文書を保管するには、以下のように行います。

1. XML 文書用の DTD を DTD 参照テーブル (DTD_REF) に挿入します。
2. 構造化検索用に XML 文書の位置およびサイド表を指定する DAD ファイルを準備します。
3. SALES_TAB 表内に、XML ユーザー定義タイプ XMLVARCHAR を持つ列を追加します。
4. XML に関して列を使用可能にします。
5. 構造化検索用のサイド表に索引を付けます。
6. XML エクステンダーに備わっているユーザー定義関数を使用して文書を保管します。

DTD リポジトリーへの DTD の保管

DTD を使用して XML 列内の XML データを妥当性検査できます。 XML エクステンダーは、XML 対応のデータベース (DTD_REF) 内に表を作成します。この表は DTD 参照と呼ばれ、DTD を保管するのに使用することができます。 XML 文書を妥当性検査することにした場合、DTD をこのリポジトリーに保管しなければなりません。このチュートリアルの DTD は `c:¥dxx¥samples¥dtd¥getstart.dtd` です。

DTD を挿入するには、以下のように行います。

DB2 コマンド・ウィンドウから、以下の SQL INSERT コマンドを入力します。

```
DB2 CONNECT TO SALES_DB
DB2 INSERT into db2xml.dtd_ref values('c:¥dxx¥samples¥dtd¥getstart.dtd',
    db2xml.XMLClobFromFile('c:¥dxx¥samples¥dtd¥getstart.dtd'), 0, 'user1',
    'user1', 'user1')
```

また、次のようなコマンド・ファイルを実行して DTD を挿入することもできます。

```
getstart_insertDTD.cmd
```

DAD ファイルを準備する

XML 列の DAD ファイルは、単純な構造になっています。保管モードが XML 列であることを指定して、索引付けのための表および列を定義します。

以下のステップで、DAD 内のエレメントはタグ と呼ばれ、 XML 文書構造のエレメントはエレメント と呼ばれます。作成する DAD ファイルに類似したサンプルは、 `c:¥dxx¥samples¥dad¥getstart_xcolumn.dad` にあります。このサンプルは、以下のステップで生成するファイルとは若干異なります。それを学

習で使用する場合、ファイル・パスがご使用の環境のものと異なる可能性があることと、 <validation> 値が YES ではなく NO に設定されていることに注意してください。

DAD ファイルを準備するには、以下のように行います。

1. テキスト・エディターをオープンして、ファイルに getstart_xcolumn.dad という名前を付けます。

   DAD ファイル内で使うどのタグでも、大文字小文字の区別があることに注意してください。

2. XML および Doctype 宣言を使って DAD ヘッダーを作成します。

   ```
   <?xml version="1.0"?>
   <!DOCTYPE DAD SYSTEM "c:¥dxx¥dtd¥dad.dtd">
   ```

   DAD ファイルは XML 文書であり、XML 宣言を必要とします。

3. 開始と終了の <DAD></DAD> タグを挿入します。他のすべてのタグは、これらのタグの内側に配置されます。

4. 開始と終了の <DTDID></DTDID> タグを挿入して、 XML 文書の DTD に DAD を関連付ける DTD ID を指定し、クライアント上での DTD の位置を指定します。

   ```
   <dtdid>c:¥dxx¥samples¥dtd¥getstart.dtd</dtdid>
   ```

   このストリングが、23ページの『DTD リポジトリーへの DTD の保管』の中の DTD 参照表への DTD の挿入時の最初のパラメーター値として使った値に一致することを確認します。たとえば、別のマシン・ドライブで作業していれば、 DTDID に使用するパスは上記のストリングとは異なることがあります。

5. 開始と終了の <validation></validation> タグを指定します。それによって、XML エクステンダーが、DTD リポジトリー表に挿入された DTD を使用して、 XML 文書構造の妥当性検査を行うことを指示します。

   ```
   <validation>YES</validation>
   ```

   <validation> の値は英大文字で指定しなければなりません。

6. 開始と終了の <Xcolumn></Xcolumn> タグを挿入して、XML 列を保管方式と定義します。この方式は、XML データが XML 列に保管されることを定義します。

   ```
   <Xcolumn>
   </Xcolumn>
   ```

7. 生成されるサイド表ごとに、開始と終了の <table></table> タグを入力します。

```
<Xcolumn>
<table name="order_side_tab">
</table>
<table name="part_side_tab">
</table>
<table name="ship_side_tab">
</table>
</Xcolumn>
```

8. サイド表に入れる列ごとに、開始と終了の <column></column> タグを入力します。各 <column> タグには、以下の 4 つの属性があります。
   - **name:** 列の名前
   - **type:** 列のデータ・タイプ
   - **path:** XPath 構文を使用する XML 文書内での対応するエレメントのロケーション・パス。ロケーション・パスの構文の詳細は、58ページの『ロケーション・パス』を参照してください。
   - **multi-occurrence:** エレメントのロケーション・パスが XML 文書構造内に複数出現できるかどうかの指示

```
<Xcolumn>
<table name="order_side_tab">
  <column name="order_key"
          type="integer"
          path="/Order/@key"
          multi_occurrence="NO"/>
  <column name="customer"
          type="varchar(50)"
          path="/Order/Customer/Name"
          multi_occurrence="NO"/>
</table>
<table name="part_side_tab">
  <column name="price"
          type="decimal(10,2)"
          path="/Order/Part/ExtendedPrice"
          multi_occurrence="YES"/>
</table>
<table name="ship_side_tab">
  <column name="date"
          type="DATE"
          path="/Order/Part/Shipment/ShipDate"
          multi_occurrence="YES"/>
</table>
</Xcolumn>
```

9. 終了のための </Xcolumn> が最後の </table> タグの後にあることを確認します。
10. 終了のための </DAD> が最後の </Xcolumn> タグの後にあることを確認します。
11. getstart_xcolumn.dad という名前を付けてファイルを保管します。

作成したばかりのファイルを、サンプル・ファイル c:¥dxx¥samples¥dad¥getstart_xcolumn.dad と比較することができます。このファイルは、XML 列を使用可能にしてサイド表を作成するのに必要な DAD ファイルの作業用コピーです。このサンプル・ファイルには、個々の環境に合わせて変更しないと実行しても正常に完了しない可能性のあるパス・ステートメントが入っています。

SALES_TAB 表を作成する

この節では、SALES_TAB 表を作成します。そこには初期状態で、注文についての販売情報を含む 2 つの列があります。

表を作成するには、以下のように行います。

DB2 コマンド・ウィンドウで、以下の CREATE TABLE ステートメントを入力します。

```
DB2 CONNECT TO SALES_DB

DB2 CREATE TABLE SALES_TAB(INVOICE_NUM CHAR(6) NOT NULL PRIMARY KEY,
         SALES_PERSON VARCHAR(20))
```

あるいは、次のようなコマンド・ファイルを実行して表を作成することもできます。

```
getstart_createTabCol.cmd
```

XML タイプの列を追加する

ここで、新規の列を SALES_TAB 表に追加します。この列には以前に生成した原形の XML 文書が含まれることになり、それは XML UDT のものでなければなりません。 XML エクステンダーには、185ページの『第8章 XML エクステンダーのユーザー定義タイプ』に説明されている複数のデータ・タイプが備わっています。このチュートリアルでは、文書を XMLVARCHAR として保管します。

XML タイプの列を追加するには、以下のように行います。

DB2 コマンド・ウィンドウで、以下の SQL ステートメントを入力します。

```
DB2 ALTER TABLE SALES_TAB ADD ORDER DB2XML.XMLVARCHAR
```

あるいは、次のようなコマンド・ファイルを実行して表を更新することもできます。

```
getstart_alterTabCol.cmd
```

XML 列を使用可能にする

XML タイプの列を作成した後、それを XML エクステンダーに関して使用可能にします。列を使用可能にすると、XML エクステンダーは DAD ファイルを読み込んで、サイド表を作成します。列を使用可能にする前に、以下のことを行う必要があります。

- XML 文書を入れる XML 列およびサイド表列のデフォルト視点を作成するかどうかを決めます。 XML 文書を照会するときにデフォルト視点を指定することができます。この演習では、視点を指定するには、`-v` パラメーターを使用します。
- 基本キーを *ROOT ID* と指定し、基本キーの列名をアプリケーション表内に指定し、そしてすべてのサイド表をアプリケーション表に関連付ける固有 ID を指定するかどうかを決めます。基本キーを指定しないと、XML エクステンダーはアプリケーション表とサイド表に DXXROOT_ID 列を追加します。 ROOT_ID 列は、アプリケーションとサイド表とを結び付けます。それによって、XML エクステンダーは XML 文書の更新時にサイド表を自動的に更新することができます。この演習では基本キーの名前は、コマンド (INVOICE_NUM) の `-r` パラメーターで指定します。すると XML エクステンダーは、指定された列を ROOT_ID として使用して、その列をサイド表に追加します。
- 表スペースの指定またはデフォルトの表スペースの使用のどちらを行うかを決めます。この演習では、デフォルトの表スペースを使います。

列を XML に関して使用可能にするには、以下のように行います。

DB2 コマンド・ウィンドウで、以下のコマンドを入力します。

```
dxxadm enable_column SALES_DB SALES_TAB ORDER GETSTART_XCOLUMN.DAD
        -v SALES_ORDER_VIEW -r INVOICE_NUM
```

あるいは、次のようなコマンド・ファイルを実行して XML の列を使用可能にすることもできます。

```
getstart_enableCol.cmd
```

XML エクステンダーは、INVOICE_NUM 列の入ったサイド表を作成し、デフォルト視点を作成します。

重要: サイド表は決して変更しないでください。 XML エクステンダーに付属している UDF を使って XML 文書の更新だけを行うようにしてください。XML 列内の XML 文書を更新すると、XML エクステンダーはサイド表を自動的に更新します。

列およびサイド表を表示する

XML 列を使用可能にしたとき、XML 列およびサイド表のビューが作成されています。このビューを使用して、XML 列を扱うことができます。

XML 列およびサイド表の列を表示するには、以下のように行います。

DB2 コマンド・ウィンドウで、以下の SQL SELECT ステートメントを入力します。

```
DB2 SELECT * FROM SALES_ORDER_VIEW
```

この視点は、getstart_xcolumn.dad ファイルに指定されているとおり、サイド表内の列を示します。

サイド表での索引の作成

サイド表で索引を作成すると、XML 文書の構造の高速検索が可能になります。このステップでは、 XML 列 ORDER を使用可能にしたときに作成されたサイド表内のキー列に索引を作成します。社員がどの列を最も頻繁に照会するかは、サービス部門から知らされています。 表3 はそのような列を示しています。これらに索引を付けます。

表 3. 索引を付けるサイド表の列

列	サイド表
ORDER_KEY	ORDER_SIDE_TAB
CUSTOMER	ORDER_SIDE_TAB
PRICE	PART_SIDE_TAB
DATE	SHIP_SIDE_TAB

サイド表に索引を付けるには、次のようにします。

DB2 コマンド・ウィンドウで次のような SQL コマンドを入力します。

```
DB2 CREATE INDEX KEY_IDX
      ON ORDER_SIDE_TAB(ORDER_KEY)

DB2 CREATE INDEX CUSTOMER_IDX
      ON ORDER_SIDE_TAB(CUSTOMER)

DB2 CREATE INDEX PRICE_IDX
     ON PART_SIDE_TAB(PRICE)

DB2 CREATE INDEX DATE_IDX
      ON SHIP_SIDE_TAB(DATE)
```

あるいは、次のようなコマンド・ファイルを実行して索引を作成することもできます。

```
getstart_createIndex.cmd
```

XML 文書の保管

XML 文書を入れる列を使用可能にし、サイド表に索引を付け終わったので、XML エクステンダーに備わっている関数を使用して文書を保管できるようになりました。データを XML 列に保管するとき、デフォルト・キャスト関数または XML エクステンダー UDF のどちらかを使用します。基本タイプ VARCHAR のオブジェクトを XML UDT XMLVARCHAR の列に保管することになるので、デフォルト・キャスト関数を使用します。保管のデフォルト・キャスト関数および XML エクステンダーに備わっている UDF についての詳細は、 132ページの『データを保管する』を参照してください。

XML 文書を保管するには、以下のように行います。

重要: XML 文書 `c:¥dxx¥samples¥xml¥getstart.xml` を開きます。 DOCTYPE にあるファイル・パスが、 DTD リポジトリーに DTD を挿入するときに DAD に指定する DTD ID と一致するかどうかを確認してください。この確認は、 db2xml.DTD_REF 表を照会し、 DAD ファイルの DTDID エレメントを調べることによって行えます。デフォルトとは異なるドライブおよびディレクトリー構造を使用している場合、 DOCTYPE 宣言のパスを変更する必要があります。

DB2 コマンド・ウィンドウで、以下の SQL INSERT コマンドを入力します。

```
DB2 INSERT INTO SALES_TAB (INVOICE_NUM, SALES_PERSON, ORDER) VALUES('123456',
   'Sriram Srinivasan', db2xml.XMLVarcharFromFile('c:¥dxx¥samples¥cmd¥getstart.xml'))
```

XML 文書を保管するとき、XML エクステンダーは自動的にサイド表を更新します。

あるいは、次のようなコマンド・ファイルを実行して文書を保管することもできます。

```
getstart_insertXML.cmd
```

表が更新されたことを確認するには、 DB2 コマンド・ウィンドウでその表に対して以下の SELECT ステートメントを実行します。

```
DB2 SELECT * FROM SALES_TAB

DB2 SELECT * FROM PART_SIDE_TAB
```

```
DB2 SELECT * FROM ORDER_SIDE_TAB

DB2 SELECT * FROM SHIP_SIDE_TAB
```

XML 文書の検索

サイド表を直接照会することによって、XML 文書を検索できます。このステップでは、価格が 2500.00 を超えるすべての注文を検索します。

サイド表を照会するには、以下のように行います。

DB2 コマンド・ウィンドウで次のような SELECT ステートメントを入力します。

```
DB2 "SELECT DISTINCT SALES_PERSON FROM SALES_TAB S, PART_SIDE_TAB P
   WHERE PRICE > 2500.00 AND
   S.INVOICE_NUM=P.INVOICE_NUM"
```

結果セットには、価格が 2500.00 を超える品目を販売した販売員の名前が示されているはずです。

あるいは、次のようなコマンド・ファイルを実行して文書を検索することもできます。

```
getstart_queryCol.cmd
```

以上で、XML 文書を DB2 表に保管する入門用のチュートリアルを完了しました。本書に示す例の多くは、これらの学習課程に基づいています。

演習: XML 文書の合成

チュートリアルのシナリオ

与えられた任務では、既存の注文データベース SALES_DB から情報を取り出し、そこからかぎとなる情報を抽出して XML 文書に保管します。次にサービス部門は、カスタマーからの要求および苦情を処理する際にそれらの XML 文書を使用します。どのデータを取り込めばよいかと、XML 文書の望ましい構造は、サービス部門から知らされています。

既存のデータを使用して、これらの表にあるデータから、 XML 文書 `getstart.xml` を合成します。

さらに、関連する表の列を、注文レコードを提供する XML 文書構造にマップするための、 DAD ファイルを計画および作成します。この文書は複数の表から構成されるため、 XML 構造および DTD によってこれらの表を関連付ける

ための XML コレクションを作成します。この DTD を使って、XML 文書の構造を定義します。また、これを使って、アプリケーションで合成した XML 文書を検査することもできます。

XML 文書用の既存のデータベースは、以下の表で説明されています。*イタリック* の列名は、サービス部門が XML 文書構造内に要求した列です。

ORDER_TAB

列名	データ・タイプ
ORDER_KEY	INTEGER
CUSTOMER	VARCHAR(16)
CUSTOMER_NAME	VARCHAR(16)
CUSTOMER_EMAIL	VARCHAR(16)

PART_TAB

列名	データ・タイプ
PART_KEY	INTEGER
COLOR	CHAR(6)
QUANTITY	INTEGER
PRICE	DECIMAL(10,2)
TAX	REAL
ORDER_KEY	INTEGER

SHIP_TAB

列名	データ・タイプ
DATE	DATE
MODE	CHAR(6)
COMMENT	VARCHAR(128)
PART_KEY	INTEGER

計画

XML エクステンダーを使用して文書を構成する前に、XML 文書の構造と、それをデータベース・データに対応させる方法について決めなければなりませ

ん。この項では、サービス部門が指定した XML 文書の構造、 XML 文書の構造の定義に使う DTD、および文書に取り込むデータの入った列にこの文書をマップする方法について概略します。

文書構造の決定

XML 文書構造は、複数の表から特定の注文に関する情報を取得して、その注文についての XML 文書を作成します。これらの各表には注文についての関連情報が含まれていて、それぞれのキー列によって結合させることができます。注文番号を最上レベルとし、カスタマー、パーツ、および出荷情報をその下に置く構造の文書がサービス部門にとって望ましいものです。文書構造は直感的かつ柔軟で、文書の構造ではなくエレメントがデータを説明するようにします。 (たとえば、カスタマーの名前は段落にではなく、『customer』と呼ばれるエレメントに含めます。) サービス部門の要求どおりにすると、 DTD および XML 文書の階層構造は
33ページの図5 で説明されているようなものになります。

文書構造を設計し終わったら、 XML 文書の構造を記述する DTD を作成しなければなりません。このチュートリアルには、XML 文書および DTD が備わっています。 DTD ファイルは、291ページの『付録B. サンプル』 にあります。 それが 33ページの図5 における構造と一致することが分かります。

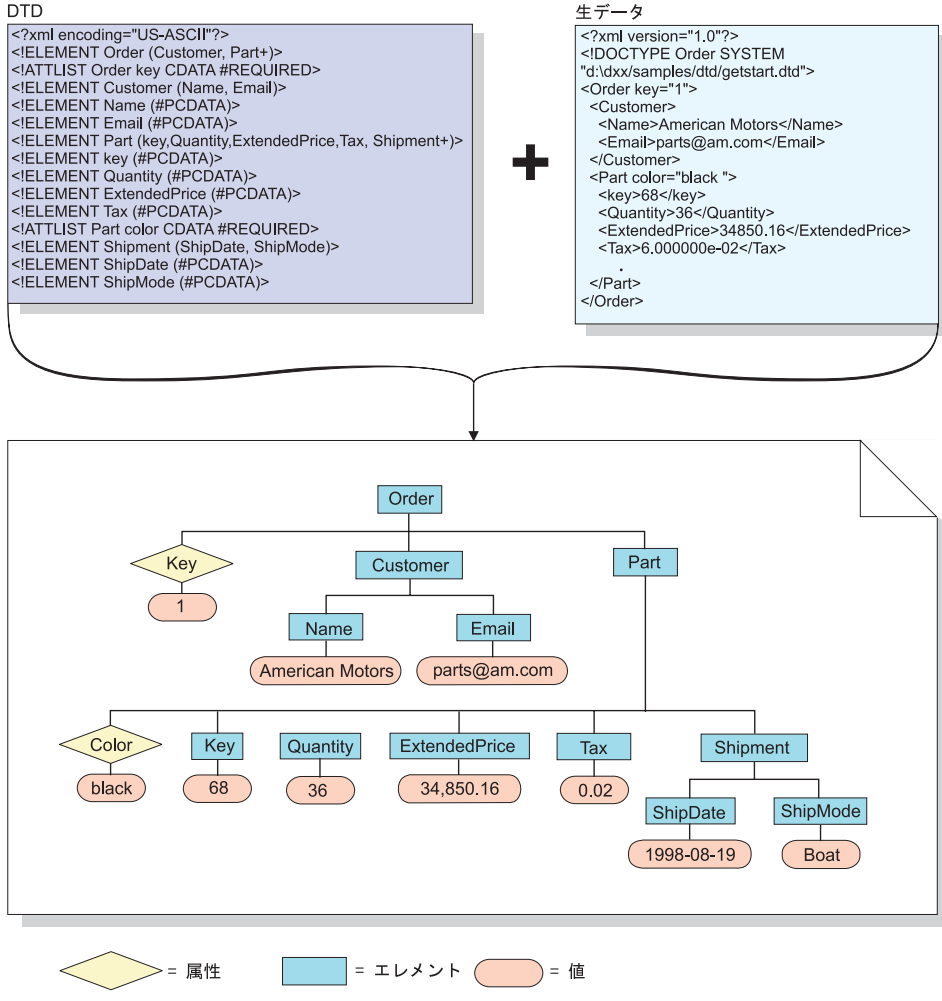


図 5. DTD および XML 文書の階層構造

XML 文書とデータベース関係とのマッピング

構造を設計して DTD を作成した後、文書の構造とエレメントおよび属性にデータを取り込むために使用する DB2 表とがどのように関連するかを示さなければなりません。 34ページの図6 に示すように、階層構造をリレーショナル表内の特定の列にマップすることができます。

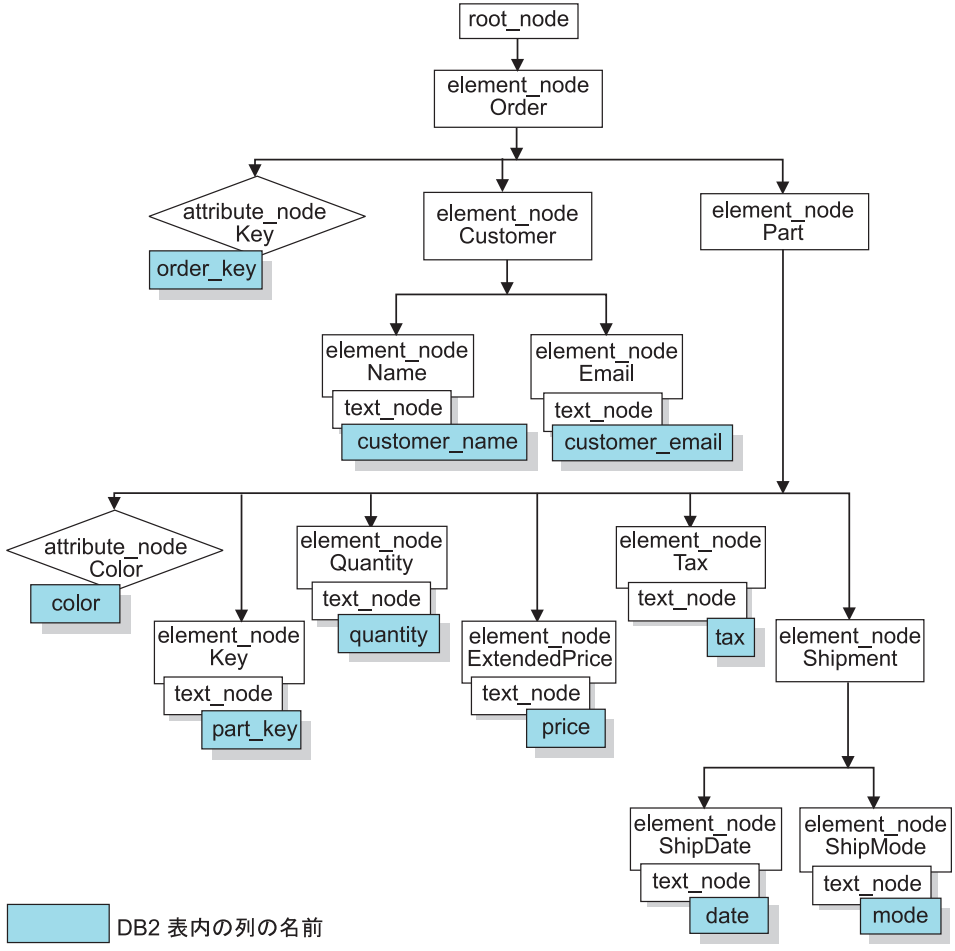


図 6. リレーショナル表の列にマップされた XML 文書

この関係記述を使用して、リレーショナル・データと XML 文書構造との関係を定義する DAD ファイルを作成します。

XML コレクションの DAD ファイルを作成するには、 図6 で説明されているように、 XML 文書がデータベース構造に対応する方法を理解して、 XML 文書がエレメントおよび属性のためのデータをどの表および列から引き出すかを記述できるようにしなければなりません。この情報は、XML コレクション用の DAD ファイルを作成するために使用します。

このチュートリアル用に、環境を設定するためのスクリプトのセットが用意されています。そのスクリプトは *DXX_INSTALL*¥samples¥cmd ディレクトリーにあ

ります。ここで、*DXX_INSTALL* は、たとえば c:¥dxx¥samples¥cmd などの、XML エクステンダーをインストールしたドライブおよびディレクトリーです。それは以下のとおりです。

getstart_db.cmd
データベースを作成して、4 つの表にデータを取り込みます。

getstart_prep.cmd
データベースを XML エクステンダーのストアード・プロシージャーと DB2 CLI にバインドします。

getstart_stp.cmd
ストアード・プロシージャーを実行して XML コレクションを合成します。

getstart_exportXML.cmd
アプリケーションで使用する XML 文書をデータベースからエクスポートします。

getstart_clean.cmd
チュートリアル環境を終結処理します。

セットアップ

データベースの作成

この節では、コマンドを使用してデータベースをセットアップします。このコマンドは、サンプルのデータベースを作成して、それに接続し、データを保持するための表を作成して、データを挿入します。

重要: XML 列の演習を完了した後で、環境の終結処理を行っていない場合、このステップを省いてもかまいません。 SALES_DB データベースがあることを確かめてください。

データベースを作成するには、以下のように行います。

1. *DXX_INSTALL*¥samples¥cmd ディレクトリーに変更します。ここで、*DXX_INSTALL* は XML エクステンダーをインストールしたドライブおよびディレクトリーです。この演習ではディレクトリーとして c:¥dxx を使用します。別のドライブおよびディレクトリーを使用する場合、この値を変更してください。
2. Windows NT の「スタート」メニューから「DB2 コマンド・ウィンドウ (DB2 Command Window)」をオープンするか、または Windows NT のコマンド・プロンプトに以下のコマンドを入力します。

   ```
   DB2CMD
   ```

3. DB2 コマンド・ウィンドウから、以下のコマンドを実行します。

```
getstart_db.cmd
```

データベースの使用可能化

XML 情報をデータベースに保管するには、それを XML エクステンダーに関して使用可能にしなければなりません。データベースを XML に関して使用可能にするとき、XML エクステンダーは以下の事柄を行います。

- すべてのユーザー定義タイプ (UDT) およびユーザー定義関数 (UDF) を作成します。
- 制御表を作成して、そこに XML エクステンダーが必要とするメタデータを取り込みます。
- db2xml スキーマを作成して、必要な特権を割り当てます。

重要: XML 列の演習を完了した後で、環境の終結処理を行っていない場合、このステップを省いてもかまいません。

データベースを XML に関して使用可能にするには、以下のように行います。

DB2 コマンド・ウィンドウから以下のスクリプトを実行して、 SALES_DB データベースを使用可能にします。

```
getstart_prep.cmd
```

このスクリプトは、データベースを XML エクステンダーのストアード・プロシージャーおよび DB2 CLI にバインドします。さらに、**dxxadm** コマンド・オプションを実行して、以下のデータベースを使用可能にします。

```
dxxadm enable_db SALES_DB
```

XML コレクションの作成: DAD ファイルの準備

複数の表内に既にデータが存在するので、それらの表を XML 文書に関連付ける XML コレクションを作成します。 XML コレクションを作成するには、DAD ファイルを準備してコレクションを定義します。

31ページの『計画』では、データが存在するリレーショナル・データベース内の列をどれにするか、および表からのデータを構造化して XML 文書とする方法を決めました。この節では、表と XML 文書の構造との間の関係を指定する DAD ファイル内のマッピング体系を作成します。

以下のステップで、DAD 内のエレメントはタグ と呼ばれ、 XML 文書構造のエレメントはエレメント と呼ばれます。作成する DAD ファイルに類似したサンプルが、 `c:¥dxx¥samples¥dad¥getstart_xcollection.dad` にあります。こ

のサンプルは、以下のステップで生成するファイルとは若干異なります。サンプルを演習で使用する場合、環境によってはファイル・パスが異なる可能性があることに注意してください。

XML 文書を構成するための DAD ファイルを作成するには、以下のように行います。

1. c:¥dxx¥samples¥cmd ディレクトリーから、テキスト・エディターをオープンして、 getstart_xcollection.dad という名前のファイルを作成します。
2. 以下のテキストを使用して、DAD ヘッダーを作成します。

   ```
   <?xml version="1.0"?>
   <!DOCTYPE DAD SYSTEM "c:¥dxx¥dtd¥dad.dtd">
   ```

   XML エクステンダーは、製品が c:¥dxx にインストールされていると想定します。そうでない場合、ここおよび以下のステップで、この値をインストール時に指定したドライブおよびディレクトリーに変更してください。
3. <DAD></DAD> タグを挿入します。他のすべてのタグは、これらのタグの内側に配置されます。
4. <validation></validation> タグを指定して、 DTD リポジトリー表に挿入した DTD を使用して XML エクステンダーが XML 文書構造の妥当性検査を行うかどうかを示します。

   ```
   <validation>NO</validation>
   ```

5. <Xcollection></Xcollection> タグを使用して、 XML コレクションとしてのアクセスおよび保管の方式を定義します。アクセスおよび保管の方式は、XML データが DB2 表のコレクション内に保管されることを定義します。

   ```
   <Xcollection>
   </Xcollection>
   ```

6. SQL ステートメントを指定して、XML コレクションに使用する表および列を指定します。この方式は SQL マッピングと呼ばれ、リレーショナル・データを XML 文書構造にマップする 2 つの方法の 1 つです。 (マッピング体系について詳しくは、66ページの『マッピング体系の種類』を参照してください。) 以下のステートメントを入力します。

   ```
   <SQL_stmt>
      SELECT o.order_key, customer_name, customer_email, p.part_key, color, quantity,
      price, tax, ship_id, date, mode from order_tab o, part_tab p,
      table (select substr(char(timestamp(generate_unique())),16)
        as ship_id, date, mode, part_key from ship_tab) s
            WHERE o.order_key = 1 and
                  p.price > 20000 and
   ```

```
                p.order_key = o.order_key and
                s.part_key = p.part_key
          ORDER BY order_key, part_key, ship_id
</SQL_stmt>
```

この SQL ステートメントは、SQL マッピングの使用時に次のようなガイドラインに準じます。文書構造の詳細は、34ページの図6 を参照してください。

- 列は、トップダウンの順序で、XML 文書構造の階層どおりに指定します。たとえば、注文とカスタマーのエレメントの列が第 1、パーツ・エレメントが第 2、出荷が第 3 になります。
- 1 つのエンティティーの列はひとまとめのグループにして、グループごとにオブジェクト ID 列 (ORDER_KEY、PART_KEY、および SHIP_ID) を付けます。
- オブジェクト ID 列を、それぞれのグループ内の第 1 列にします。たとえば、キー属性に関連した列よりも O.ORDER_KEY を前に置き、パーツ・エレメントの列よりも p.PART_KEY を前に置きます。
- SHIP_TAB 表には単一キー条件列はないので、 generate_unique DB2 組み込み関数を使って SHIP_ID 列を生成します。
- こうすれば、ORDER BY ステートメント内でオブジェクト ID 列はトップダウンの順序でリストされます。 ORDER BY 内の列は、スキーマや表名で修飾してはならず、 SELECT 文節内の列名に一致しなければなりません。

SQL ステートメントの作成時の要件の詳細は、68ページの『マッピング体系の要件』を参照してください。

7. 合成した XML 文書で使用する次のようなプロローグ情報を追加します。

   ```
   <prolog>?xml version="1.0"?</prolog>
   ```

   これと全く同じテキストがすべての DAD ファイルに必要です。

8. 合成しようとする XML 文書で使う <doctype></doctype> タグを追加します。 <doctype> タグには、クライアントで保管される DTD のパスが入ります。

   ```
   <doctype>!DOCTYPE Order SYSTEM "c:¥dxx¥samples¥dtd¥getstart.dtd"</doctype>
   ```

9. <root_node></root_node> タグを使用して、XML 文書のルート・エレメントを定義します。 root_node 内で、XML 文書を形成するエレメントおよび属性を指定します。
10. 以下の 3 つのタイプのノードを使用して、 XML 文書構造を DB2 リレーショナル表構造にマップします。

element_node
XML 文書内のエレメントを指定します。 element_node には子の element_node があってもかまいません。

attribute_node
XML 文書内のエレメントの属性を指定します。

text_node
エレメントのテキスト内容とリレーショナル表内の列データを、最下位の element_node について指定します。

これらのノードについての詳細は、63ページの『DAD ファイル』 を参照してください。 34ページの図6 は、XML 文書および DB2 表列の階層構造を図示し、使用されるノードの種類を示しています。陰影のあるボックスは、XML 文書を構成するためにデータが取り出される DB2 表の列名を示します。

以下のステップでは、ノードの各タイプを 1 タイプずつ追加することになります。

a. XML 文書の各エレメントについて、<element_node> タグを定義します。

```
<root_node>
<element_node name="Order">
  <element_node name="Customer">
     <element_node name="Name">
     </element_node>
     <element_node name="Email">
     </element_node>
  </element_node>
  <element_node name="Part">
     <element_node name="key">
     </element_node>
     <element_node name="Quantity">
     </element_node>
     <element_node name="ExtendedPrice">
     </element_node>
     <element_node name="Tax">
     </element_node>
     <element_node name="Shipment" multi_occurrence="YES">
        <element_node name="ShipDate">
        </element_node>
        <element_node name="ShipMode">
        </element_node>
     </element_node> <!-- end Shipment -->
  </element_node> <!-- end Part -->
</element_node> <!-- end Order -->
</root_node>
```

<Shipment> 子エレメントには、multi_occurrence=″YES″ の属性があることに注意してください。この属性は属性を持たないエレメントのために使用され、それは文書内で繰り返されています。 <Part> には色の属性があり、そのため固有のものとなっているので、multi-occurrence 属性は使用しません。

b. XML 文書内の各属性について <attribute_node> タグを定義します。これらの属性は、element_node 内でネストしています。追加された attribute_nodes は太字で強調表示されます。

```
<root_node>
<element_node name="Order">
  <attribute_node name="key">
  </attribute_node>
  <element_node name="Customer">
     <element_node name="Name">
     </element_node>
     <element_node names"Email">
     </element_node>
  </element_node>
  <element_node name="Part">
    <attribute_node name="color">
    </attribute_node>
    <element_node name="key">
    </element_node>
    <element_node name="Quantity">
    </element_node>

...

  </element_node> <!-- end Part -->
</element_node> <!-- end Order -->
</root_node>
```

c. 最下位の element_node ごとに、<text_node> タグを定義します。それは文書構成時に DB2 から取り出される文字データが XML エレメントに含まれることを示しています。

```
<root_node>
<element_node name="Order">
  <attribute_node name="key">
  </attribute_node>
  <element_node name="Customer">
    <element_node name="Name">
      <text_node>
      </text_node>
    </element_node>
    <element_node name="Email">
      <text_node>
      </text_node>
    </element_node>
  </element_node>
  <element_node name="Part">
```

```
    <attribute_node name="color">
    </attribute_node>
    <element_node name="key">
       <text_node>
       </text_node>
    </element_node>
    <element_node name="Quantity">
      <text_node>
      </text_node>
    </element_node>
    <element_node name="ExtendedPrice">
      <text_node>
      </text_node>
    </element_node>
    <element_node name="Tax">
      <text_node>
      </text_node>
    </element_node>
    <element_node name="Shipment" multi-occurrence="YES">
      <element_node name="ShipDate">
        <text_node>
        </text_node>
      </element_node>
      <element_node name="ShipMode">
        <text_node>
        </text_node>
      </element_node>
    </element_node> <!-- end Shipment -->
  </element_node> <!-- end Part -->
</element_node> <!-- end Order -->
</root_node>
```

d. 最下位の element_node ごとに、<column> タグを定義します。これらのタグは、XML 文書の構成時にどの列からデータを取り出すかを指定し、通常は <attribute_node> または <text_node> タグ内にあります。ここで定義される列は、 <SQL_stmt> SELECT 文節内になければならないことを覚えておいてください。

```
<root_node>
<element_node name="Order">
  <attribute_node name="key">
    <column name="order_key"/>
  </attribute_node>
  <element_node name="Customer">
    <element_node name="Name">
      <text_node>
        <column name="customer_name"/>
      </text_node>
    </element_node>
    <element_node name="Email">
      <text_node>
        <column name="customer_email"/>
     </text_node>
    </element_node>
```

```
</element_node>
<element_node name="Part">
  <attribute_node name="color">
     <column name="color"/>
  </attribute_node>
  <element_node name="key">
    <text_node>
      <column name="part_key"/>
    </text_node>
  <element_node name="Quantity">
    <text_node>
      <column name="quantity"/>
    </text_node>
  </element_node>
  <element_node name="ExtendedPrice">
    <text_node>
      <column name="price"/>
    </text_node>
  </element_node>
  <element_node name="Tax">
    <text_node>
      <column name="tax"/>
    </text_node>
  </element_node>
  <element_node name="Shipment" multi-occurrence="YES">
    <element_node name="ShipDate">
      <text_node>
        <column name="date"/>
      </text_node>
    </element_node>
    <element_node name="ShipMode">
      <text_node>
        <column name="mode"/>
      </text_node>
    </element_node>
  </element_node> <!-- end Shipment -->
 </element_node> <!-- end Part -->
</element_node> <!-- end Order -->
</root_node>
```

11. 終了のための </root_node> タグが最後の </element_node> タグの後にあることを確認します。
12. 終了のための </Xcollection> タグが </root_node> タグの後にあることを確認します。
13. 終了のための </DAD> タグが </Xcollection> タグの後にあることを確認します。
14. ファイルを getstart_xcollection.dad として保管します。

作成したばかりのファイルを、サンプル・ファイル c:¥dxx¥samples¥dad¥getstart_xcollection.dad と比較することができます。このファイルは、XML 文書を合成するのに必要な DAD ファイルの作業用コ

ピーです。このサンプル・ファイルには、個々の環境に合わせて変更しないと実行しても正常に完了しない可能性のあるパス・ステートメントが入っています。

アプリケーションで頻繁に XML コレクションを使って文書を合成する場合、そのコレクションを使用可能にすることでコレクション名を定義することができます。コレクションを使用可能にすると、それは XML_USAGE 表に登録されるので、ストアード・プロシージャーの実行時にそのコレクション名を (DAD ファイル名の代わりに) 指定して、パフォーマンスを向上させることができます。この演習では、コレクションを使用可能化しません。コレクションの使用可能化の詳細は、123ページの『XML コレクションの使用可能化』を参照してください。

XML 文書を構成する

このステップでは、dxxGenXML() ストアード・プロシージャーを使用して、DAD ファイルによって指定された XML 文書を構成します。このストアード・プロシージャーは、文書を XMLVARCHAR UDT として戻します。

XML 文書を構成するには、以下のように行います。

1. DB2 コマンド・ウィンドウで、以下のコマンドを入力してストアード・プロシージャーを実行します。

   ```
   getstart_stp.cmd
   ```

   XML 文書は構成されて、RESULT_TAB 表に保管されています。

   このステップで使えるストアード・プロシージャーのサンプルは、次に示すファイル内にあります。

   - `c:¥dxx¥samples¥c¥tests2x.sqc` は、組み込み SQL を使ってストアード・プロシージャーを呼び出す方法を示しており、 `getstart_stp.cmd` で使われる `texts2x` 実行可能ファイルを生成します。
   - `c:¥dxx¥samples¥cli¥sql2xml.c` は、 CLI を使ってストアード・プロシージャーを呼び出す方法を示しています。

2. XML エクステンダーの検索関数 Content() を使用して、XML 文書を表からファイルにエクスポートします。

   ```
   DB2 CONNECT TO SALES_DB

   DB2 SELECT db2xml.Content(db2xml.xmlvarchar(doc),
      'c:¥dxx¥samples¥cmd¥getstart.xml') FROM RESULT_TAB
   ```

あるいは、次のようなコマンドを実行してファイルをエクスポートすることもできます。

```
getstart_exportXML.cmd
```

この演習では、DB2 ストアード・プロシージャーの結果セット機能を使って 1 つ以上の合成 XML 文書を入手して、 1 行取り出すごとに 1 つの文書を入手する方法を習得しました。 1 行の文書を入手するたびに、それをファイルにエクスポートしますが、これがこの機能を例示する最も簡単な方法です。より効率的なデータの取り出し方の詳細は、 c:¥dxx¥samples¥cli の中の CLI の例を参照してください。

チュートリアル環境の終結処理

チュートリアル環境を終結処理するには、 getstart_clean.cmd ファイルを実行します。このファイルを実行すると、以下のようになります。

- XML 列 ORDER が使用不可になります。
- チュートリアルで作成された表が除去されます。
- DTD 参照表から DTD が削除されます。

このコマンド・ファイルが SALES_DB データベースを使用不可にしたり除去したりすることはありません。このデータベースは、そのまま XML エクステンダーで使うことができます。この章の両方の演習を完了していないと、エラー・メッセージが出されることがあります。その場合のエラーは無視してもかまいません。

チュートリアル環境を終結処理するには、次のようにします。

1. DB2 コマンド・ウィンドウから、以下のコマンドを実行します。

   ```
   getstart_clean.cmd
   ```

2. データベースを使用不可にしたい場合、 DB2 コマンド・ウィンドウで次のような XML エクステンダー・コマンドを実行します。

   ```
   dxxadm disable_db SALES_DB
   ```

   このコマンドは、管理制御表 DTD_REF および XML_USAGE を除去するほか、 XML エクステンダーに備えられているユーザー定義タイプと関数を除去します。

3. データベースを除去したい場合、 DB2 コマンド・ウィンドウで次のようなコマンドを実行します。

   ```
   db2 drop database SALES_DB
   ```

このコマンドは、SALES_DB を除去します。

第2部 管理

ここでは、XML エクステンダーの管理タスクを実行する方法を示します。

第3章 XML エクステンダーを使用するための準備: 管理

この章では、XML エクステンダーのセットアップおよび計画に必要な要件について説明します。

セットアップの要件

以下の節では、XML エクステンダーをセットアップするための要件について説明します。

ソフトウェア要件

XML エクステンダーは、AIX、Windows NT、および Sun Solaris 上で使用可能です。

必要なソフトウェア: XML エクステンダーには、 DB2 ユニバーサル・データベースのバージョン 7.1 以降が必要です。

オプションのソフトウェア:

- 構造化テキスト検索のために、DB2 ユニバーサル・データベース テキスト・エクステンダーのバージョン 7.1 以降
- XML エクステンダー管理ツールのために、
  - DB2 UDB JDBC (DB2 UDB バージョン 7.1 または以降で入手可能)
  - JDK 1.1.7 または JRE 1.1.7 (DB2 UDB コントロール・センターに備わっている)
  - JFC 1.1 および Swing 1.1 (DB2 UDB コントロール・センターに備わっている)

インストール要件

以下のタスクに関しては、ご使用のオペレーティング・システムの README ファイルを参照してください。

- XML エクステンダーを DB2 UDB データベースにバインドする。

  セキュリティーのために、XML エクステンダーを各データベースにバインドしなければなりません。バインドを完了する方法についての詳細は 227ページの『作業を始める前に』を、例については以下を参照してください。

  DXX_INSTALL¥samples¥cmd¥getstart_prep.cmd

- UNIX 上のセットアップ指示を表示する。
- XML アクセスのためのデータベースを作成する。

権限に関する要件

管理タスクを実行するには、DB2ADM 権限が必要です。

管理ツール

XML エクステンダーには、管理のための 3 つの方式が備わっています。それらは、XML エクステンダー管理ウィザード、XML エクステンダー管理コマンド、および XML エクステンダーのストアード・プロシージャー です。

- 管理ウィザードは、プロンプトを出して管理タスクを案内するものであり、これが管理のための推奨される方式です。このツールの使用方法は、75ページの『第4章 XML データの管理』 の管理タスクで説明されています。
- 管理コマンド **dxxadm** は、種々の管理タスクのためのオプション・を提供します。このコマンドの使用方法は、 75ページの『第4章 XML データの管理』および 171ページの『第7章 XML エクステンダー管理コマンド: **dxxadm**』 の管理タスクで説明されています。
- 管理ストアード・プロシージャーも、種々の管理タスクのためのオプションを提供します。ストアード・プロシージャーについては、 228ページの『管理ストアード・プロシージャー』で説明されています。

管理の計画

XML 文書を使用するアプリケーションについて計画するとき、設計に関する以下の決定を最初に行う必要があります。

- XML 文書をデータベース内のデータから構成するかどうか
- 既存の XML 文書を保管するかどうか、およびそれらを原形の XML 文書として列に保管するか、それとも通常の DB2 データに分解して保管するか

これらの決定を行った後、残りの管理タスクについて計画することができます。

- XML 文書の妥当性検査を行うかどうか
- 高速検索および取得のために XML 列データに索引付けをするかどうか
- XML 文書の構造を DB2 リレーショナル表にマップする方法

XML エクステンダーをどのように使用するかは、アプリケーションが何を必要とするかに依存します。 3ページの『第1章 XML エクステンダーの概要』に示されているように、XML 文書を既存の DB2 データから構成して、 XML

文書を原形の文書または DB2 データとして DB2 内に保管することができます。これらの保管方式とアクセス方式のおのおので、計画上の要件は異なります。以下の節では、これらの計画それぞれについての考慮事項を説明します。

アクセスおよび保管の方式を選択する

XML エクステンダーには、DB2 を XML リポジトリーとして使用するための 2 つのアクセスおよび保管の方式として、 XML 列と XML コレクションが備わっています。最初に、どちらの方式が XML データにアクセスして操作するアプリケーションの必要に最も適しているかを判別しなければなりません。

XML 列
: XML 文書の全体を DB2 列データとして保管および取得します。XML データは XML 列によって表されます。

XML コレクション
: XML 文書を分解してリレーショナル表のコレクションにするか、または XML 文書をリレーショナル表のコレクションから構成します。

ご使用のアプリケーションの性質によって、使用するアクセスおよび保管のタイプ、および XML データを構成する方式が決まります。以下のシナリオは、それぞれのアクセスおよび保管方式が最も適切な状況を説明しています。

XML 列をいつ使用するか

XML 列は以下の状況で使用します。

- XML 文書がすでに存在するか、外部ソースから送られてきて、その文書をネイティブの XML 形式で保管したいとき。整合性のため、およびアーカイブと監査のために、それらを DB2 に保管したいと考えるかもしれません。
- XML 文書が一般に読み取られるものの、更新されないとき。
- ファイル名データ・タイプを使用して DB2 の外部にある XML 文書をローカルまたはリモート・ファイル・システムに保管し、 DB2 を管理および検索操作のために使用したいとき。
- XML エレメントまたは属性に基づく範囲検索を行う必要があり、頻繁に検索引き数となるエレメントまたは属性が何かを知っているとき。
- 文書に大きなテキスト・ブロックを伴うエレメントがあり、文書全体を原形のままに保ちながら DB2 テキスト・エクステンダーを使用して構造テキスト検索を行いたいとき。

XML コレクションをいつ使用するか

XML コレクションは以下の状況で使用します。

- 既存のリレーショナル表にデータがあり、特定の DTD に基づいて XML 文書を構成したいとき。
- リレーショナル表に正しくマップされるデータのコレクションと共に保管しなければならない XML 文書があるとき。
- 別々のマッピング体系を使用して、別々のリレーショナル・データ視点を作成したいとき。
- 他のデータ・ソースから送られてくる XML 文書があるとき。データは重要だがタグは重要ではなく、データベースに純粋なデータを保管したいとき。データを既存の表に保管するか、新しい表に保管するかに関して柔軟でありたいとき。
- XML 文書の小さなサブセットを頻繁に更新することが必要で、更新のパフォーマンスが重要であるとき。
- 着信 XML 文書の全体のデータを保管する必要があるものの、取り出すのはそのサブセットだけであることが多いとき。
- XML 文書が 2 ギガバイトを超えていて、それを分解しなければならないとき。

文書アクセス定義 (DAD) ファイルを使用して、これら 2 つのアクセスおよび保管方式によって XML データを DB2 表に関連付けます。 53ページの図7 は、DAD がアクセスおよび保管の方式を指定する方法を示しています。

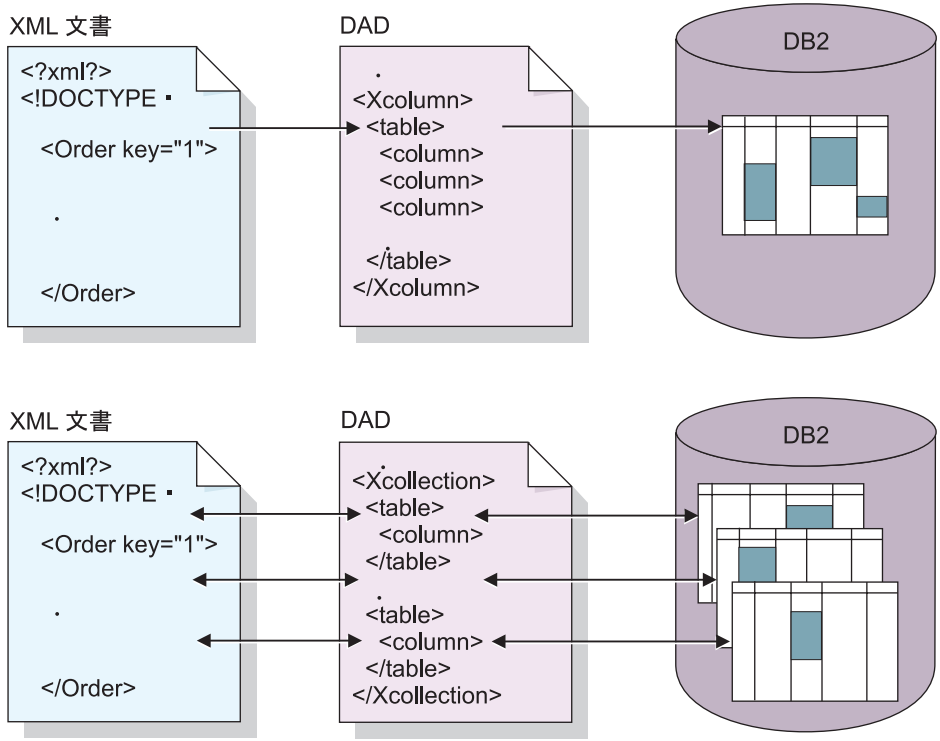


図 7. DAD ファイルは XML 文書構造を DB2 にマップして、アクセスおよび保管の方式を指定します。

DAD ファイルは、XML エクステンダー管理の重要な一部です。それは DTD などの主要ファイルの位置を定義して、 XML 文書構造が DB2 データにどのように関係するかを指定します。最も大切なこととして、それはアプリケーションで使用するアクセスおよび保管の方式を定義します。

XML 列について計画する

以下の節では、XML 列についての計画タスクを解説します。

妥当性検査

アクセスおよび保管の方式を選択した後、データの妥当性検査 を行うかどうかを決めることができます。 DTD を使用して XML データを妥当性検査し、XML 文書が有効であることを確認し、 XML データに対して構造化検索を実行します。 DTD は DTD リポジトリーに保管されていますが、 DB2 サーバーのアクセス先のファイル・システムに保管することもできます。

複数の異なる DTD を使用して、同じ XML 列内の文書の妥当性検査を行うことができます。つまり、類似のエレメントを持つ類似の構造の文書が、異なる DTD を呼び出すようにすることができます。複数の DTD を参照するためには、以下の指針に従ってください。

- DOCTYPE 定義内の XML 文書のシステム ID は、絶対パス名を使用して DTD ファイルを指定しなければなりません。
- DAD ファイル内で、妥当性検査について YES を指定しなければなりません。
- 少なくとも 1 つの DTD が DTD_REF 表に保管されている必要があります。すべての DTD をこの表に保管することができます。
- それら複数の DTD は、サブエレメントにおける相違だけをもつ共通の構造である必要があります。
- DAD ファイルは、その列内の文書によって参照されるすべての DTD に共通のエレメントまたは属性を指定しなければなりません。

重要: XML データを DB2 に挿入する前に、妥当性検査を行うかどうかを決めてください。 XML エクステンダーは、すでに DB2 に挿入されているデータの妥当性検査はサポートしません。

考慮事項:

- XML 文書をアーカイブ目的で保管する場合を除き、 XML データは DTD を使用して妥当性検査を行うことを推奨します。妥当性検査を行うためには、DTD が XML エクステンダー・リポジトリーに存在しなければなりません。 DTD をリポジトリーに挿入する方法については、80ページの『DTD リポジトリーへの DTD の保管』を参照してください。
- XML 文書の保管またはアーカイブには、DTD は必要ありません。
- XML データの妥当性検査を行うと、パフォーマンスに多少の影響を与える可能性があります。
- 複数の DTD を使用できますが、索引付けできるのは共通のエレメントおよび属性だけです。
- 文書の妥当性検査を選択しない場合、 XML 文書によって指定された DTD は処理されません。妥当性検査できない文書の部分の処理時にも、 DTD を処理してエンティティーと属性のデフォルトを解決することは重要です。

XML ユーザー定義タイプ

XML 文書は UDT として XML 列に保管します。使用可能な UDT については、55ページの表4 を参照してください。

表 4. XML エクステンダー UDT

ユーザー定義タイプ列	ソース・データ・タイプ	使用法の説明
XMLVARCHAR	VARCHAR(*varchar_len*)	XML 文書の全体を VARCHAR として DB2 内に保管します。
XMLCLOB	CLOB(*clob_len*)	XML 文書の全体を *CLOB* として DB2 内に保管します。
XMLFILE	VARCHAR(1024)	XML 文書のファイル名を DB2 内に保管して、 XML 文書を DB2 サーバーにとってローカルのファイル内に保管します。

サイド表

サイド表の計画を立てるとき、この表の編成方法、いくつの表を作成するか、およびサイド表のデフォルト視点を作成するかどうかを考察しなければなりません。その場合の決断は、エレメントと属性が複数回出現するかどうかや、照会のパフォーマンス上の要件などのいくつかの考慮事項をベースに下します。

複数オカレンス: ロケーション・パスが複数出現する文書の場合、 XML エクステンダーは、複数出現するエレメントの順序を追跡記録するために、各サイド表内にタイプ INTEGER の列 DXX_SEQNO を追加します。DXX_SEQNO を使うと、SQL 照会内に ORDER BY DXX_SEQNO と指定することで、元の XML 文書と同じ順序でエレメントを取り出すことができます。

デフォルト視点および照会のパフォーマンス: XML 列を使用可能にするときに、 ROOT ID という名前の固有 ID を使ってアプリケーション表をサイド表に結合するためのデフォルトの読み取り専用視点を指定することができます。デフォルト視点を使うと、サイド表を照会することで XML 文書を検索することができます。たとえば、アプリケーション表が SALES_TAB、サイド表が ORDER_TAB、PART_TAB、および SHIP_TAB であるとすると、次のようにします。

```
SELECT sales_person FROM sales_order_view
      WHERE  price > 2500.00
```

この SQL ステートメントは、 PRICE が 2500.00 より大きい列 ORDER に注文を保管している SALES_TAB 内の営業担当者の名前を戻します。

デフォルト視点を照会すると、アプリケーション表とサイド表の 1 つの視点が提示されるという利点があります。しかし、作成されるサイド表の数が多い場合、照会の負荷は大きくなります。したがって、デフォルト視点の作成をお勧めするのは、サイド表の合計列数が少ない場合のみです。アプリケーションで、サイド表の重要な列を結合して独自の視点を作成することができます。

XML 列データの索引

XML 列文書に索引を付けるかどうかは、計画時の重要な決断事項です。この決断は、データへのアクセスが必要になる頻度、および構造化検索の際のパフォーマンスの重要性に基づいて下します。

XML 文書の全体を含む XML 列を使用するとき、 XML エレメントまたは属性値を含めるためのサイド表を作成してから、これらの列に索引を作成することができます。どのエレメントおよび属性に索引を作成するのかを決めなければなりません。

XML 列を索引付けすることによって、頻繁に照会される整数、10 進数、または日付などの汎用データ・タイプのデータを、データベース・エンジンのネイティブ DB2 索引サポートを使用して索引付けすることができます。 XML エクステンダーは XML エレメントまたは属性の値を XML 文書から抽出して、それらを*サイド表* に保管し、それらのサイド表に索引を作成できるようにします。

サイド表の各列は、XML エレメントまたは属性および SQL データ・タイプを識別するロケーション・パスによって指定できます。 57ページの図8 は、サイド表のある XML 列を示しています。

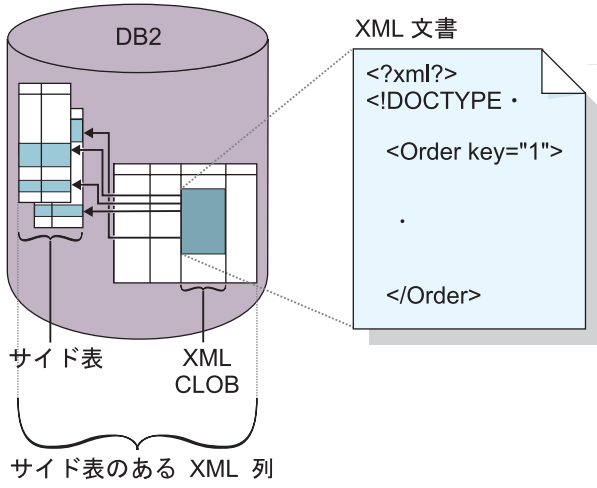


図 8. サイド表のある XML 列

XML エクステンダーは、XML 文書を XML 列に保管するときに自動的にサイド表にデータを取り入れます。

高速検索のために、DB2 *B-tree 索引付け* テクノロジーを使用してこれらの列に索引を作成します。索引の作成に使用される方式はオペレーティング・システムによって異なり、 XML エクステンダーはそれらの方式をサポートします。

考慮事項:

- XML 文書内のエレメントまたは属性で*複数オカレンス* のあるものについては、 XML 文書の複合構造のために、複数オカレンスのある XML エレメントまたは属性ごとに別個のサイド表を作成しなければなりません。

  たとえば、`/Order/Part/ExtendedPrice` に索引を作成して、`/Order/Part/ExtendedPrice` がデータ・タイプ REAL になるように指定したいと仮定します。この場合、XML エクステンダーは `/Order/Part/ExtendedPrice` の値をサイド表内の PRICE 列に保管します。
- 1 つの XML 列に複数の索引を作成することができます。前述の例では、2 つのサイド表内に 2 つの列、つまり `ExtendedPrice` のために 1 つと `ShipDate` のために 1 つを作成できます。
- サイド表をアプリケーション表に関連付けるには、ROOT ID、アプリケーション表内の基本キーの列名、そしてすべてのサイド表をアプリケーション表に関連付ける固有 ID を使います。アプリケーション表の基本キーを ROOT

ID にするかどうかを決めることができます。ただし、それを複合キーにすることはできません。この方法をお勧めします。

アプリケーション表に単一の基本キーが存在しない場合、または何かの理由でそれを使用したくない場合、XML エクステンダーはアプリケーション表を変更して、挿入時に作成される固有の ID を保管する列 DXXROOT_ID を追加します。すべてのサイド表には、固有の ID を持つ DXXROOT_ID 列があります。基本キーを ROOT ID として使用すると、アプリケーション表内の基本キー列と同じ名前とタイプの列が、すべてのサイド表に入り、基本キーの値が保管されます。

- XML 列を DB2 テキスト・エクステンダーについて使用可能にする場合、テキスト・エクステンダーの構造化テキスト機能も使用できます。テキスト・エクステンダーには「*セクション検索*」サポートがあり、ロケーション・パスによって指定された特定の文書コンテキストで検索語の一致を調べることにより、従来の全テキスト検索の機能を拡張します。*構造化テキスト索引* は、汎用 SQL データ・タイプでの XML エクステンダーの索引付けと共に使用できます。

ロケーション・パス

ロケーション・パス とは、 XML エレメントまたは属性を識別する一連の *XML タグ* です。 XML エクステンダーは、以下の状況でロケーション・パスを使用します。

- 抽出するエレメントおよび属性を識別する UDF を抽出しているとき
- DAD での XML 列の索引付け体系の定義の際に、 XML エレメント (または属性) と DB2 列とのマッピング・ファイルを指定するため
- 構造化テキスト検索のために テキスト・エクステンダーによって

59ページの図9 は、ロケーション・パスおよび XML 文書との関係を示しています。

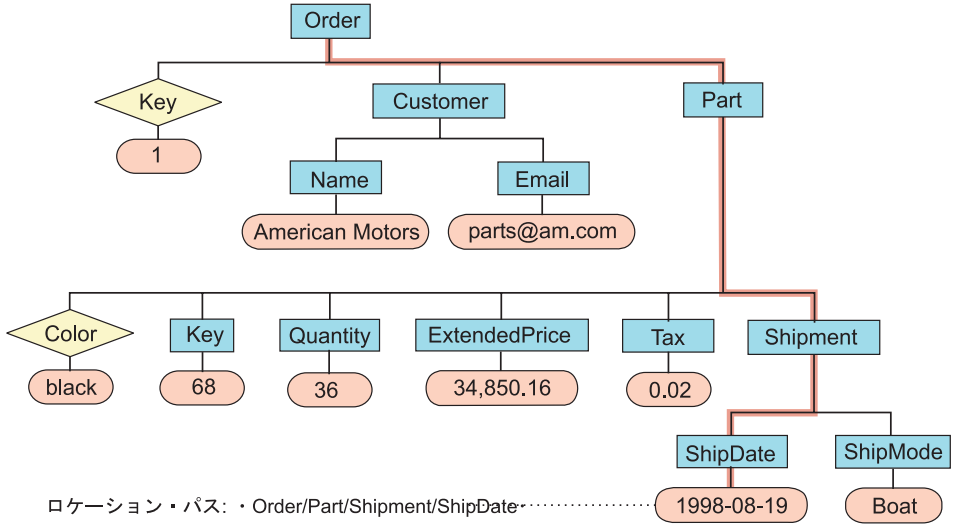


図 9. 文書を構造化 XML 文書として DB2 表の列に保管する

ロケーション・パスの構文: 以下のリストは、XML エクステンダーでサポートされるロケーション・パスの構文を説明しています。単一スラッシュ (/) のパスは、コンテキストが文書全体であることを示します。

1. / XML のルート・エレメント を表します。
2. */tag1*
   ルートの下のエレメント *tag1* を表します。
3. */tag1/tag2/..../tagn*
   ルートからの降順のチェーン、*tag1*、*tag2*、そして *tagn*-1 の子である、*tagn* という名前のエレメントを表します。
4. *//tagn*
   名前が *tagn* であるエレメントを表します。ダブルスラッシュ (//) は、0 個以上の任意のタグを示します。
5. */tag1//tagn*
   名前が *tagn* であるエレメントを表します。それはルートの下にある *tag1* という名前のエレメントの子であり、ダブルスラッシュ (//) は 0 個以上の任意のタグを示します。
6. */tag1/tag2/@attr1*
   tag2 という名前のエレメントの属性 *attr1* を表します。それはルートの下にあるエレメント *tag1* の子です。

7. */tag1/tag2[@attr1="5"]*

名前が *tag2* で属性 *attr1* の値が 5 であるエレメントを表します。 *tag2* はルートの下にある *tag1* という名前のエレメントの子です。

8. */tag1/tag2[@attr1="5"]/.../tagn*

ルートからの降順のチェーン、*tag1*、*tag2*、そして *tagn*-1 の子である、*tagn* という名前のエレメントを表します。 *tag2* の属性 *attr1* の値は 5 です。

ワイルドカード: ロケーション・パス内のエレメントをアスタリスクに置き換えて、任意のストリングに一致させることができます。

単純ロケーション・パス: *単純ロケーション・パス* は、XML 列 DAD ファイルで定義される、サイド表のエレメントおよび属性を指定するために使用する、ロケーション・パス構文です。単純ロケーション・パスは、エレメント・タイプ名を順番に並べてシングルスラッシュ (/) で区切ったものとして表されます。属性値は、エレメント・タイプの後で大括弧で囲まれて示されます。 表 5 は、単純ロケーション・パスの構文を要約しています。

表 5. 単純ロケーション・パスの構文

対象	ロケーション・パス	説明
XML エレメント	*/tag1/tag2/..../tagn-1/tagn*	*tagn* という名前のエレメントおよびその親によって識別されるエレメントの内容
XML 属性	*/tag_1/tag_2/..../tag_n-1 /tag_n/@attr1*	*tagn* およびその親によって識別されるエレメントの、*attr1* という名前の属性

XML エクステンダーの制約事項: XML エクステンダーには、DAD 内でのエレメントまたは属性の定義にロケーション・パスを使用することに関して制限があります。 XML エクステンダーは、エレメントまたは属性と DB2 列との間で 1 対 1 マッピングを使用するので、 DAD ファイル内および関数内で許可される構文規則を制限します。 表6 には、ロケーション・パスについての制限が説明されています。サポートされるロケーション・パスの列に指定されている数字は、 59ページの『ロケーション・パスの構文』での構文表示を示します。

表 6. ロケーション・パスの使用に関する XML エクステンダーの制限

ロケーション・パスの使用	サポートされるロケーション・パス
サイド表の XML 列 DAD マッピングのエレメント	3、6 (表5 で説明されている単純ロケーション・パス)

表 6. ロケーション・パスの使用に関する XML エクステンダーの制限 (続き)

ロケーション・パスの使用	サポートされるロケーション・パス
UDF の抽出	1-8[1]
Update UDF	1-8[1]
テキスト・エクステンダーによる UDF の検索	1-8

[1] UDF の抽出および更新は、属性がある述部を持つロケーション・パスはサポートしますが、エレメントがあるものはサポートしません。

DAD ファイル

XML 列では、DAD は基本的に、XML 列に保管された文書を索引付けする方法を指定します。 DAD は XML 形式の文書で、クライアントに存在します。XML 文書を DTD を使用して妥当性検査することを選択した場合、 DAD ファイルをその DTD に関連付けることができます。 DAD ファイルのデータ・タイプは CLOB です。 このファイルは最大 100 KB まで可能です。

XML 列の DAD ファイルには XML ヘッダーが含まれ、クライアント上の DAD ファイルおよび DTD へのディレクトリー・パスを指定し、索引付けのためにサイド表に保管する XML データのマップを提供します。

XML 列のアクセスおよび保管の方法を指定するには、 DAD ファイル内で以下のタグを使用します。

<Xcolumn>

XML データを XML 文書全体として、 XML データについて使用可能にされている DB2 列内で保管および検索するように指定します。

XML について使用可能にされた列は、XML エクステンダーの UDT です。アプリケーションには、任意のユーザー表 内の列を含めることができます。 XML 列データには、主に SQL ステートメントおよび XML エクステンダーの UDF を介してアクセスします。

XML エクステンダー管理ウィザードまたはエディターを使用して、DAD の作成および更新を行うことができます。

XML コレクションについて計画する

XML コレクションの計画を立てるとき、 DB2 データからの文書の合成と DB2 データへの XML 文書の分解の、一方または両方に関していくつかの考慮事項があります。以下の項では、XML コレクションを計画する上での懸案事項について述べ、合成および分解に関する考慮事項を示します。

妥当性検査
アクセスおよび保管の方式を選択した後、データの妥当性検査を行うかどうかを決めることができます。 DTD を使用して XML データの妥当性検査を行います。 DTD を使用すると、XML 文書が有効であることを確認し、 XML データを構造化検索できるようになります。 DTD は DTD リポジトリーに保管されています。

推奨:XML データは DTD を使用して妥当性検査を行ってください。妥当性検査を行うためには、DTD が XML エクステンダー・リポジトリーに存在しなければなりません。 DTD をリポジトリーに挿入する方法については、80ページの『DTD リポジトリーへの DTD の保管』を参照してください。 DTD の要件は、XML 文書を構成しているのか分解しているのかによって異なります。

- 構成の場合、生成された XML 文書を 1 つの DTD に関してのみ妥当性検査することができます。使用される DTD は、DAD ファイルで指定されます。
- 分解の場合、異なる DTD を使用して構成のための文書を妥当性検査することができます。つまり、同じ DAD ファイルを使用して文書を分解しますが、異なる DTD を呼び出すことができます。複数の DTD を参照するためには、以下の指針に従ってください。
  - 少なくとも 1 つの DTD が DTD_REF 表に保管されている必要があります。すべての DTD をこの表に保管することができます。
  - それら複数の DTD は、サブエレメントにおける相違をもつ共通の構造である必要があります。
  - DAD ファイル内で、妥当性検査を指定しなければなりません。
  - XML 文書のシステム ID は、絶対パス名を使用して DTD ファイルを指定しなければなりません。
  - 文書を分解する方法についての指定は DAD ファイルに含まれるので、指定できるのは分解のための共通のエレメントおよび属性だけです。 DTD に固有のエレメントおよび属性は、分解できません。

重要: XML データを DB2 に挿入する前に、 XML データの妥当性検査を行うかどうかを決めてください。 XML エクステンダーは、すでに DB2 に挿入されているデータの妥当性検査はサポートしません。

考慮事項:

- XML を交換形式として使用する場合は、DTD を使用してください。

- XML データの妥当性検査を行うと、パフォーマンスに多少の影響を与える可能性があります。
- 分解のために複数の DTD を使用する場合、分解できるのは共通のエレメントおよび属性だけです。
- 1 つの DTD を使用する場合は、すべてのエレメントおよび属性を分解することができます。
- 構成に使用できる DTD は 1 つだけです。

DAD ファイル

XML コレクションの場合、DAD ファイルは XML 文書の構造を、文書の構成元または分解先である DB2 表にマップします。

たとえば、XML 文書内に <Tax> と呼ばれるエレメントがある場合、 <Tax> を TAX と呼ばれる列にマップしなければならないという具合です。 XML データとリレーショナル・データとの関係を DAD 内で定義します。

DAD ファイルは、コレクションを使用可能にする際に指定するか、または DAD ファイルを XML コレクションのストアード・プロシージャー で使用する際に指定します。 DAD は XML 形式の文書で、クライアントに存在します。 XML 文書を DTD を使用して妥当性検査することを選択した場合、DAD ファイルをその DTD に関連付けることができます。 XML エクステンダーのストアード・プロシージャーの入力パラメーターとして使用される場合、 DAD ファイルのデータ・タイプは CLOB です。 このファイルは最大 100 KB まで可能です。

XML コレクションのアクセスおよび保管の方法を指定するには、 DAD ファイル内で以下のタグを使用します。

<Xcollection>

XML データを XML 文書から分解してリレーショナル表のコレクションにするか、またはリレーショナル表のコレクションから構成して XML 文書にするかを指定します。

XML コレクションは、XML データを含むリレーショナル表のセットの仮想名です。アプリケーションは任意のユーザー表の XML コレクションを使用可能にすることができます。これらのユーザー表は、既存の業務データ用の既存の表、または XML エクステンダーが最近作成した表などです。 XML コレクション・データへのアクセスには、主に XML エクステンダーに備わっているストアード・プロシージャーを使用します。

DAD ファイルは、以下の種類のノードを使用して XML 文書のツリー構造を定義します。

root_node
文書のルート・エレメントを指定します。

element_node
エレメントを識別すると同時に、ルート・エレメントまたは子エレメントとなることができます。

text_node
エレメントの CDATA テキストを表します。

attribute_node
エレメントの属性を表します。

図10 は、DAD ファイルで使用されているマッピングの一部を示しています。このノードは、XML 文書の内容をリレーショナル表内の表列にマップします。

```
<?xml version="1.0"?>
<!DOCTYPE DAD SYSTEM "c:¥dtd¥dad.dtd">
<DAD>
  ...
<Xcollection>
<SQL_stmt>
      ...
</SQL_stmt>
<prolog>?xml version="1.0"?</prolog>
<doctype>!DOCTYPE DAD SYSTEM "c:¥dxx¥sample¥dtd¥getstart.dtd"</doctype>
<root_node>
   <element_node name="Order">       --> Identifies the element <Order>
     <attribute_node name="key">     --> Identifies the attribute "key"
        <column name="order_key"/>   --> Defines the name of the column, "order_key",
                                          to which the element and attribute are
                                          mapped
     </attribute_node>
     <element_node name="Customer">  --> Identifies a child element of <Order> as
                                          <Customer>
        <text_node>                  --> Specifies the CDATA text for the element
                                          <Customer>
          <column name="customer">   --> Defines the name of the column, "customer",
                                          to which the child element is mapped
        </text_node>
     </element_node>
       ...
   </element_node>
     ...
<root_node>
</Xcollection>
</DAD>
```

図 10. ノード定義

この例では、SQL ステートメント内の最初の 2 列に対してエレメントおよび属性がマップされます。

XML エクステンダー は、<stylesheet> エレメントを使用して、スタイル・シートの処理命令もサポートします。これは、DAD ファイルのルート・ノードの内部でなければならず、 XML 文書用に定義した文書タイプおよびプロローグがなければなりません。たとえば次のようにします。

```
<Xcollection>
...
<prolog>...</prolog>
<doctype>...</doctype>
<stylesheet>?xml-stylesheet type="text/css" href="order.css"?</stylesheet>
<root_node>...</root_node>
...
  </Xcollection>
```

XML エクステンダー管理ウィザードまたはエディターを使用して、DAD ファイルの作成および更新を行うことができます。 <stylesheet> エレメントは、現在のところ XML エクステンダー管理ウィザードではサポートしていません。

XML コレクションのマッピング体系

XML コレクションを使用している場合、 XML データをリレーショナル・データベース内で表す方法を定義するマッピング体系 を選択しなければなりません。 XML コレクションはリレーショナル構造の XML 文書内で使用される階層構造と一致しなければならないので、それら 2 つの構造の相違点を理解している必要があります。 66ページの図11 は、階層構造がリレーショナル表の列にどのようにマップされるかを示しています。

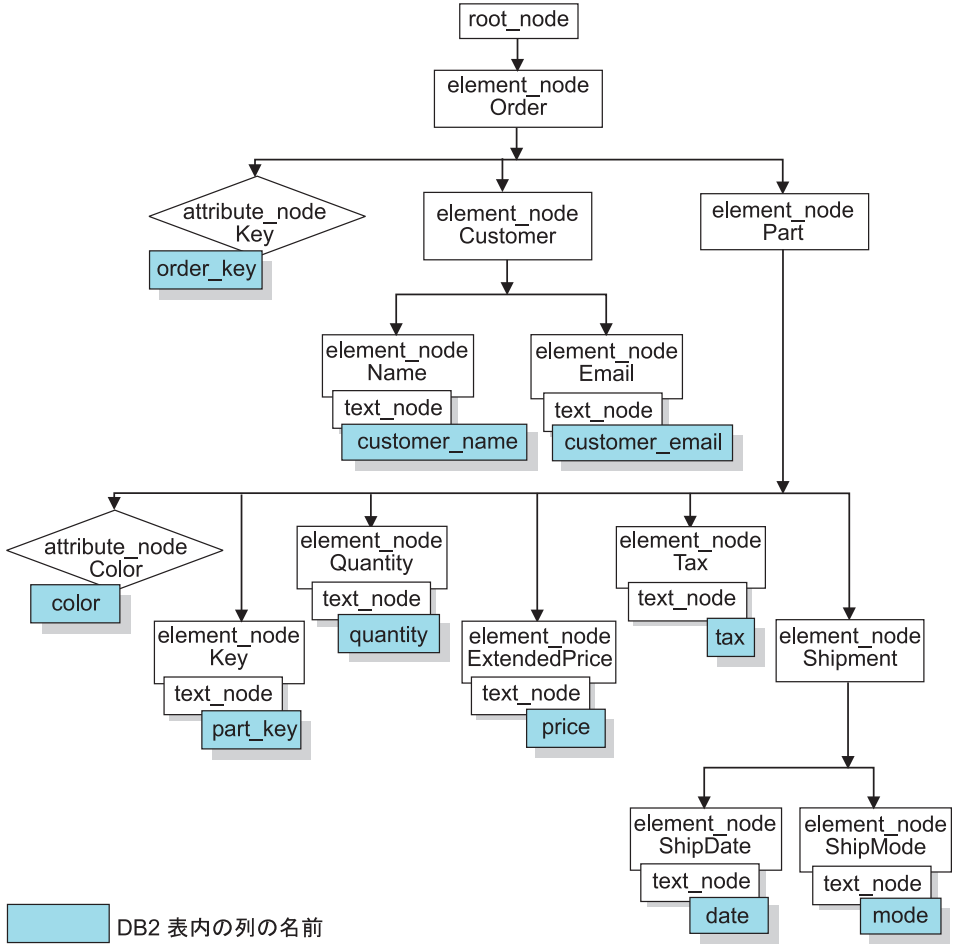


図 11. リレーショナル表の列にマップされた XML 文書構造

XML エクステンダーは、複数のリレーショナル表内にある XML 文書を構成または分解するとき、マッピング体系を使用します。 XML エクステンダーには、DAD ファイルの作成に役立つウィザードが備わっています。しかし、DAD ファイルを作成する前に、 XML データを XML コレクションにマップする方法を考えなければなりません。

マッピング体系の種類: マッピング体系は、DAD ファイル内の <Xcollection> エレメントに指定されます。 XML エクステンダーには、*SQL* マッピング と *リレーショナル・データベース (RDB_node)* マッピング という、 2 種類のマッピング体系が備わっています。どちらの方式も XSLT モデルを使用して XML 文書の階層を定義します。

SQL マッピング

リレーショナル・データから XML 文書への簡単で直接的なマッピングを可能にします。これは、単一の SQL ステートメントと *XSLT* データ・モデル を介して行います。 SQL マッピングは構成に使用され、分解には使用されません。 SQL マッピングは DAD ファイル内の SQL_stmt エレメントによって定義されます。 SQL_stmt の内容は、有効な SQL ステートメントです。 SQL_stmt は、SELECT 文節の列を、XML 文書で使用される XML エレメントまたは属性にマップします。 XML 文書の構成のために定義されたとき、SQL ステートメントの SELECT 文節にある列名が、 *attribute_node* の値または *text_node* の内容を定義するために使用されます。 FROM 文節は、データを含む表を定義します。 WHERE 文節は、*結合* と検索*条件* を指定します。

SQL マッピングにより、DB2 ユーザーは SQL を使用してデータをマップすることができます。 SQL マッピングを使用するとき、 1 つの SELECT ステートメント内ですべての表を結合して照会を形成しなければなりません。 1 つの SQL ステートメントでは十分でない場合、RDB_node マッピングの使用を考慮してください。すべての表を結び合わせるため、これらの表に*基本キー* と*外部キー* の関係があることが推奨されます。

RDB_node マッピング

XML エレメントの内容または XML 属性の値の位置を定義して、XML エクステンダーが XML データを保管または取得する場所を判別できるようにします。

RDB_node には、表、任意指定の列、および任意指定の条件についてのノード定義が 1 つ以上含まれます。表および列は、XML データをデータベースに保管する方法を定義するために使用されます。条件は、XML データの選択基準、または XML コレクション表を結合する方法を指定します。

マッピング体系を定義するには、<Xcollection> エレメントのある DAD を作成します。 68ページの図12 は、 3 つのリレーショナル表内のデータから XML 文書のセットを構成する、 XML コレクション SQL マッピングのあるサンプルの DAD ファイルの一部を示しています。

```
<?xml version="1.0"?>
<!DOCTYPE DAD SYSTEM "c:¥dtd¥dad.dtd">
<DAD>
  <dtdid>c:¥dxx¥samples¥dad¥getstart.dtd</dtdid>
  <validation>YES</validation>
  <Xcollection>
    <SQL_stmt>
       SELECT o.order_key, customer, p.part_key, quantity, price, tax, date,
              mode, comment
         FROM order_tab o, part_tab p,
               table(select substr(char(timestamp(generate_unique())),
               as ship_id, date, mode, from ship_tab) as s
        WHERE p.price > 2500.00 and s.date > "1996-06-01" AND
              p.order_key = o.order_key and s.part_key = p.part_key
    </SQL_stmt>
    <prolog>?xml version="1.0"?</prolog>
    <doctype>!DOCTYPE DAD SYSTEM "c:¥dxx¥samples¥dtd¥getstart.dtd"</doctype>
    <root_node>
   <element_node name="Order">
  <attribute_node name="key">
<column_name="order_key"/>
  </attribute_node>
  <element_node name="Customer">
  <text_node>
  <column name="customer"/>
  </text_node>
  <element_node>

...
  </element_node><!-end Part->
   </element_node><!-end Order->
    </root_node>
  </Xcollection>
</DAD>
```

図 12. SQL マッピング体系

XML エクステンダーには、XML コレクション内のデータを管理するいくつかのストアード・プロシージャーが備わっています。これらのストアード・プロシージャーは両方のタイプのマッピングをサポートしますが、 DAD ファイルは 『マッピング体系の要件』で説明されている規則に従う必要があります。

マッピング体系の要件: 以下の節では、それぞれの XML コレクションのマッピング体系の要件について解説します。

SQL マッピングを使用する際の要件

このマッピング体系では、SQL_stmt エレメントを DAD <Xcollection> エレメントに指定しなければなりません。 SQL_stmt には、複数のリレーショナル表を照会*述部* に結び合わせる単一の SQL ステートメントが含まれていなければなりません。さらに、以下の文節が必要です。

- **SELECT 文節**
  - 列の名前が固有のものであることを確認してください。 2 つの表に同じ列名がある場合、AS キーワードを使用してどちらか一方に別名を作成します。
  - 同じ表の列をグループ化して、リレーショナル表の論理階層レベルを使用します。つまり、XML 文書の階層構造にマップしたときの重要度のレベルに応じて、表をグループ化するということです。 SELECT 文節では、より高いレベルの表の列はより低いレベルの表の列より前に指定します。以下の例は、複数の表の間での階層関係を例示しています。

    ```
    SELECT o.order_key, customer, p.part_key, quantity, price, tax,
           ship_id, date, mode
    ```

    この例で、表 ORDER_TAB の order_key および customer は、XML 文書の階層ツリーで高いレベルにあるために、最高のリレーショナル・レベルを持ちます。表 SHIP_TAB の ship_id、date、および mode は、最低のリレーショナル・レベルにあります。
  - 単一列候補キーを使用して、各レベルを開始します。そのようなキーが表にない場合、照会では表式および組み込み関数 generate_unique() を使用してそのキーを生成しなければなりません。上記の例で、o.order_key は ORDER_TAB の基本キー、そして part_key は PART_TAB の基本キーです。それらは、選択されるそれぞれの列グループの先頭にあります。 SHIP_TAB には基本キーがないので、それを生成しなければなりません。この場合、ship_id となります。それは SHIP_TAB 表グループの最初の列としてリストされています。以下の例に示されているように、FROM 文節を使用して基本キー列を生成します。
- **FROM 文節**
  - 表式および組み込み関数 generate_unique() を使用して、基本単一キーのない表に単一キーを生成します。たとえば次のようにします。

    ```
    FROM order_tab as o, part_tab as p,
           table(select substr(char(timestamp(generate_unique())),16) as
           ship_id, date, mode from ship_tab) as s
    ```

    この例で、generate_unique() 関数は CHAR データ・タイプの TIMESTAMP に適用され、別名 ship_id が与えられます。

– 列を明瞭にするために必要な場合、別名を使用します。たとえば、ORDER_TAB に対して o、PART_TAB に対して p、そして SHIP_TAB に対して s を使用できます。

- **WHERE 文節**
  – 基本キーと外部キーとの関係を、表とコレクションを結び合わせる結合条件として指定します。たとえば次のようにします。

```
WHERE p.price > 2500.00 AND s.date > "1996-06-01" AND
      p.order_key = o.order_key AND s.part_key = p.part_key
```

  – その他の検索条件を述部に指定します。任意の有効な述部を使用できます。

- **ORDER BY 文節**
  – ORDER BY 文節を SQL_stmt の最後に定義します。
  – 列名が SELECT 文節内の列名に一致していることを確認します。
  – データベースのエンティティー関連 (ER) 設計でエンティティーを固有に識別する列名または ID を指定します。 ID は、表式および組み込み関数 generate_unique またはユーザー定義関数 (UDF) を使用して生成することができます。
  – エンティティーの階層のトップダウン順序を保持します。ORDER BY 文節で指定される列は、各エンティティーについてリストされる最初の列でなければなりません。順序を維持すれば、生成される XML 文書に誤った重複が入らないようにすることができます。
  – ORDER BY 内の列をスキーマ名または表名で修飾しないでください。

SQL_stmt には前述の要件がありますが、述部の式が表内に列を使用する限り WHERE 文節に任意の述部を指定できるので、たいへん強力です。

RDB_node マッピングを使用する際の要件

このマッピング方式を使用するとき、エレメント SQL_stmt を DAD ファイルの <Xcollection> エレメント内で使用しないでください。その代わりに、RDB_node エレメントを *element_node* の各トップ・ノードに対して、および各 attribute_node と text_node に対して使用します。

- **先頭 element_node に対する RDB_node**

  DAD ファイル内の*先頭 element_node* は、 XML 文書のルート・エレメントを表します。先頭 element_node に対する RDB_node を以下のように指定します。

– XML 文書に関連したすべての表を指定します。たとえば、以下のマッピングは、先頭 element_node である element_node <Order> の RDB_node 内に 3 つの表を指定します。

```
<element_node name="Order">
    <RDB_node>
        <table name="order_tab"/>
        <table name="part_tab"/>
        <table name="ship_tab"/>
        <condition>
           order_tab.order_key = part_tab.order_key AND
           part_tab.part_key = ship_tab.part_key
        </condition>
    </RDB_node>
```

コレクション内の表が 1 つしかない場合には、条件エレメントは空にしておくか、ないままにしておくことができます。

– 文書を分解する場合、または DAD ファイルに指定されている XML コレクションを使用可能にする場合、表ごとに基本キーを指定しなければなりません。基本キーは、単一の列または複数の列 (複合キーと呼ばれる) から構成されます。基本キーは、RDB_node の表エレメントに属性キーを追加することによって指定されます。複合キーが提供された場合、キー属性はスペースで区切られた複数のキー列名で指定されます。たとえば次のようにします。

```
<table name="part_tab" key="part_key, price"/>
```

分解のために指定された情報は、文書を構成するときには無視されます。

– orderBy 属性を使用して、複数オカレンスのあるエレメントまたは属性を含む XML 文書を再構成して元の構造に戻します。この属性により、文書の順序の保持に使用されるキーとなる列名を指定できます。 orderBy 属性は DAD ファイル内の表エレメントの一部であり、オプションの属性です。

表名および列名は、明示的にスペルアウトしなければなりません。

- **各 attribute_node および text_node に対する RDB_node**

  このマッピング体系では、データは各 element_node に対する attribute_node および text_node 内に存在します。そのため、XML エクステンダーはデータベースのどこからデータを見つけるかを知る必要があります。 attribute_node および text_node ごとに RDB_node を指定して、ストアード・プロシージャーにどの表、ど

の列、そしてどの照会条件からデータを取得するかを指定します。表および列の値は指定しなければなりません。条件の値は任意指定です。

– 列データを含む表の名前を指定します。表の名前は、先頭 element_node の RDB_node に含まれていなければなりません。この例では、エレメント <Price> の text_node に対して、表は PART_TAB として指定されます。

```
<element_node name="Price">
    <text_node>
        <RDB_node>
            <table name="part_tab"/>
            <column name="price"/>
            <condition>
                 price > 2500.00
            </condition>
        </RDB_node>
    </text_node>
</element_node>
```

– エレメント・テキストのデータを含む列の名前を指定します。前述の例では、その列は PRICE として指定されています。

– 照会条件を使用して XML 文書を生成したい場合、条件を指定します。上記の例では、条件は `price > 2500.00` として指定されています。条件に適合するデータだけが、生成される XML 文書に含まれます。条件は有効な WHERE 文節でなければなりません。

– 文書を分解する場合、または DAD ファイルに指定されている XML コレクションを使用可能にする場合、 attribute_node および text_node ごとに列タイプを指定しなければなりません。そうすれば、XML コレクションの使用可能化時に新規の表が作成されたときに、どの列のデータ・タイプも確実に正しいものになります。列タイプは、属性タイプを列エレメントに追加することによって指定されます。 たとえば、次のようになります。

```
<column name="order_key" type="integer"/>
```

分解のために指定された情報は、文書を構成するときには無視されます。

RDB_node マッピング・アプローチでは、SQL ステートメントを与える必要はありません。しかし、RDB_node エレメントに複合的な照会条件を入れることはさらに難しくなります。たとえば、*共用体*、式、または演算を使用しても、 SQL-to-XML アプローチほど強力な機能は得られません。

分解表サイズの要件

分解では RDB_node マッピングを使って、表行へのエレメントと属性値の抽出によって XML 文書を DB2 表に分解する方法を指定します。各 XML 文書の値は、1 つ以上の DB2 表に保管されます。どの表にも、各文書から分解した最大 1024 行までを入れることができます。

たとえば、XML 文書を 5 つの表に分解する場合、その 5 つの表のおのおのに、該当する文書中の 1024 行までを入れることができます。複数の文書用の行をもつ表でも、各文書につき 1024 行までを入れることができます。 20 個の文書をもつ表の場合、各文書につき 1024 行ずつ、20,480 行を入れることができます。

複数出現エレメント (XML 構造内で複数出現する可能性のあるロケーション・パスをもつエレメント) を使うと、行数が影響を受けます。たとえば、20 回出現するエレメント <Part> の入った文書は、表内で 20 行として分解されることがあります。複数出現エレメントを使用する場合、このような表サイズの制限事項に配慮してください。

第4章 XML データの管理

XML エクステンダーの管理タスクは、データベースおよび表列を XML 使用可能にすること、および XML データを DB2 リレーショナル構造にマップすることです。 XML エクステンダーには利用できるいくつかの管理ツールがあります。管理タスクを実行するためのアプリケーションを自分で開発することも、あるいは単にウィザードを使用することもできます。次のようなツールを使用して、XML エクステンダーの管理タスクを実行できます。

- XML エクステンダー管理ウィザード
- **dxxadm** コマンド
- XML エクステンダー管理ストアード・プロシージャー

この章では、管理ウィザードおよび **dxxadm** コマンドに関連した管理タスクについて説明します。管理ストアード・プロシージャーについては、228ページの『管理ストアード・プロシージャー』で説明します。

この章のタスクを完全に実行するには、 50ページの『管理の計画』に説明されている概念および計画タスクに精通している必要があります。

以下の節で、XML エクステンダー管理タスクを説明します。

1. 『管理ウィザードの開始』
2. 79ページの『XML 用のデータベースの使用可能化』
3. 80ページの『DTD リポジトリーへの DTD の保管』
4. 82ページの『XML 列またはコレクションの定義』
5. 83ページの『XML 列を処理する』
6. 96ページの『XML コレクションの処理』

管理ウィザードの開始

この節では、XML エクステンダー管理ウィザードのセットアップおよび起動に関する情報を示します。

管理ウィザードのセットアップ

ご使用のオペレーティング・システム用の readme ファイルに書かれている、管理ウィザードのインストールおよび構成のステップを完了したことを確認し

てください。その際、バインド・ステートメントを実行して、必要なソフトウェアを CLASSPATH ステートメント内に含めたことを確認してください。

- バインド・ステートメントは、ウィザードの README ファイルおよび入門サンプル・ファイルで提供されています。

  /dxx_install/samples/cmd/getstart_prep.cmd

- CLASSPATH ステートメントは以下のようになります (見やすくするために改行を入れています)。

```
.;C:¥java¥db2java.zip;C:¥java¥runtime.zip;C:¥java¥sqlj.zip;
C:¥dxx¥dxxadmin¥dxxadmin.jar;C:¥dxx¥dxxadmin¥dxxadmin.cmd;
C:¥dxx¥dxxadmin¥html¥dxxahelp*.htm;C:¥java¥jdk¥lib¥classes.zip;
C:¥java¥swingall.jar
```

  重要 : ウィザードでは、スペースを入れずにパス名を入力する必要があります。 IBM DB2 ユニバーサル・データベース V7.1 がデフォルトでインストールされている場合には、 SQLLIB¥java が Program Files ディレクトリーにあるので、 Java コードをさらに単純なパスにコピーしてください 。 Java コードを移動させたり、CLASSPATH を変更したりしないでください。コントロール・センターは、インストール時に指定された CLASSPATH を必要とします。

XML エクステンダー管理ウィザードは、クラス・ファイルを使用します。メインの XML エクステンダー管理クラス・ファイルの完全なファイル名は、以下のとおりです。

```
com.ibm.dxx.admin.Admin.
```

このファイルをご使用のシステムに合わせて変更し、ウィザードを起動します。

- JDK を使用して起動するには、以下のように入力します。

```
java -classpath classpath com.ibm.dxx.admin.Admin
```

- JRE を使用して起動するには、以下のように入力します。

```
jre -classpath classpath com.ibm.dxx.admin.Admin
```

  classpath は、以下のいずれかを指定します。

  – 管理ウィザード・クラス・ファイルがある場所を指定する %CLASSPATH% 環境変数。このオプションを使用する場合は、ご使用のシステムの CLASSPATH は、 *dxx_install*/dxxadmin ディレクトリーを指していなければなりません。このディレクトリーには、dxxadmin.jar、xml4j.jar、および db2java.zip ファイルが含まれています。たとえば次のようにします。

```
java -classpath %CLASSPATH% com.ibm.dxx.admin.Admin
```

– XML エクステンダー管理ウィザードを実行元になる、*dxx_install*/dxxadmin ディレクトリー内のファイルを指すポインターで、 %CLASSPATH% 環境変数をオーバーライドします。たとえば次のようにします。

```
java -classpath dxxadmin.jar;xml4j.jar;db2java.zip com.ibm.dxx.admin.Admin url=jdbc:db2:mydb
userid=db2xml password=db2xml driver=COM.ibm.db2.jdbc.app.DB2Driver
```

オプションで、実行時に以下のパラメーターを指定することができます。

url 接続する IBM DB2 UDB データ・ソースへの完全修飾 URL パス。たとえば、jdbc:db2://dxx.stl.ibm.com:8080/guidb。ウィザードでは『Address』 というラベルが付いています。

userid 上記データ・ソースにアクセスするために使用するユーザー ID。たとえば、db2guest。

password
上記ユーザー ID 用のパスワード。たとえば、guest。

driver 上記 URL 用の JDBC ドライバー名。デフォルトは、COM.ibm.db2.jdbc.net.DB2Driver。ウィザードでは 『JDBC driver』というラベルが付いています。

これらの値の詳細については、『管理ウィザードの起動』を参照してください。

管理ウィザードの起動

XML エクステンダー管理ウィザードを起動するには、以下のステップに従います。

1. ウィザードを起動します。

   Windows NT の場合:

   デスクトップから、XML エクステンダー管理ウィザードのアイコンをダブルクリックします。

   AIX、Sun Solaris、および Linux の場合:

   dxxadmin ファイルを実行します。

   管理ウィザードの「ログオン (Logon)」ウィンドウが開きます。

XML エクステンダー管理ウィザードを起動すると、「ログオン (Logon)」ウィンドウが表示されます。 XML データを処理するときに使用したいデータベースにログインします。 XML エクステンダーは、現行のインスタンスに接続します。

2. **「アドレス (Address)」**フィールドに、接続する IBM DB2 UDB データ・ソースへの完全修飾 JDBC URL を入力します。アドレスの構文は次のとおりです。

   スタンドアロン構成の場合 (推奨) :

   ```
   jdbc:db2:database_name
   ```

   ここで、

   database_name
   接続して XML 文書を保管するデータベース。

   たとえば、次のようになります。

   ```
   jdbc:db2:sales_db
   ```

   ネットワーク構成の場合:

   ```
   jdbc:db2://server_name:port_number/database_name
   ```

   ここで、

   server_name
   XML エクステンダーのあるサーバーの名前。

   port_number
   このサーバーへの接続に使用するポート番号。ポート番号を判別するには、サーバー・マシンで、DB2 コマンド行から以下のコマンドを入力します。

   ```
   db2jstrt port#
   ```

   Windows NT ユーザーは、ファイル ¥winnt¥system32¥driver¥etc¥services でポート番号をチェックすることができます。

   database_name
   接続して XML 文書を保管するデータベース。

   たとえば、次のようになります。

   ```
   jdbc:db2://host1.ibm.com:8080/sales_db
   ```

3. **「ユーザー ID (User ID)」**および**「パスワード (Password)」**フィールドでは、接続先データベースの DB2 ユーザー ID とパスワードを入力するか、またはすでに表示されている値を確認します。
4. **「JDBC ドライバー (JDBC Driver)」**フィールドでは、指定されたアドレス用の JDBC ドライバー名を確認します。以下の値を使用します。

   スタンドアロン構成の場合 (デフォルトおよび推奨) :

COM.ibm.db2.jdbc.app.DB2DRIVER

ネットワーク構成の場合:

COM.ibm.db2.jdbc.net.DB2DRIVER

5. **「終了 (Finish)」**をクリックしてウィザードに接続し、「ランチパッド (LaunchPad)」ウィンドウに進みます。

「ランチパッド (LaunchPad)」ウィンドウは、5 つの管理ウィザードへのアクセスを提供します。これらのウィザードでは、以下のことを行うことができます。

- データベースを使用可能にする
- DTD リポジトリーに DTD を追加する
- 以下の DAD ファイルを処理する。
  - XML 列
  - XML コレクション
- XML 列を処理する
- XML コレクションを処理する

XML 用のデータベースの使用可能化

XML エクステンダーを使用して DB2 から XML 文書を保管または検索するには、 XML 用のデータベースを使用可能にします。 XML エクステンダーは、現行のインスタンスを使用して接続されるデータベースを使用可能にします。

データベースを XML に関して使用可能にするとき、XML エクステンダーは以下の事柄を行います。

- すべてのユーザー定義タイプ (UDT) およびユーザー定義関数 (UDF) を作成します。
- 制御表を作成して、そこに XML エクステンダーが必要とするメタデータを取り込みます。
- db2xml スキーマを作成して、必要な特権を割り当てます。

  XML 関数のフルネームは *schema-name.function-name* です (ここで、*schema-name* は SQL オブジェクト の論理グループ化を行うための ID です)。 UDF または UDT を参照する時には、どこでもフルネームを使用できます。また、UDF や UDT を参照する時にスキーマ名を省略することもできます。その場合、DB2 は関数パスを使用して、必要な関数またはデータ・タイプを判別します。

管理ウィザードの使用

データベースで XML データを使用可能にするには、以下のステップに従います。

1. 管理ウィザードをセットアップして開始します。詳しくは 75ページの『管理ウィザードの開始』を参照してください。
2. 「ランチパッド (LaunchPad)」ウィンドウから**「データベースを使用可能にする (Enable database)」**を選択して、現行のデータベースを使用可能にします。

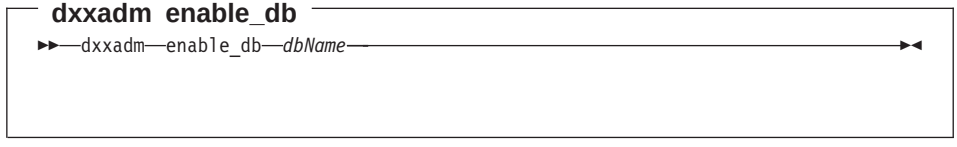
   データベースがすでに使用可能になっていれば、**「データベースを使用不可にする (Disable a database)」**のみ選択できます。

   データベースを使用可能にすると、「ランチパッド (LaunchPad)」ウィンドウに戻ります。

DB2 コマンド・シェルから

コマンド行から、使用可能にするデータベースを指定して **dxxadm** と入力します。

構文:



パラメーター:

dbName
使用可能にするデータベースの名前。

例: SALES_DB という既存のデータベースを使用可能にします。

```
dxxadm enable_db SALES_DB
```

DTD リポジトリーへの DTD の保管

DTD を使用して XML 列内または XML コレクション内の XML データを妥当性検査できます。 DTD は XML 列の妥当性検査を行います。また、XML 構造化検索およびコレクションの合成と分解に使われる DAD を定義するためにも使用されます。

すべての DTD は DTD リポジトリー (つまり DTD_REF という DB2 表) に保管されます。これには db2xml というスキーマ名があります。 DTD_REF 表内のそれぞれの DTD には、一意的な ID があります。あるデータベースを XML に関して使用可能にすると、XML エクステンダーは DTD_REF 表を作成します。

DTD の使用方法について、詳しくは 53ページの『XML 列について計画する』 および 61ページの『XML コレクションについて計画する』を参照してください。

DTD を挿入するには、DB2 コマンド・シェルまたは管理ウィザードのいずれも使用できます。

管理ウィザードの使用

DTD を挿入するには、以下のステップに従います。

1. 管理ウィザードをセットアップして開始します。詳しくは 75ページの『管理ウィザードの開始』を参照してください。
2. 「ランチパッド (LaunchPad)」ウィンドウから**「DTD のインポート (Import a DTD)」**を選択して、既存の DTD ファイルを現在のデータベースの DTD リポジトリーにインポートします。「DTD のインポート (Import a DTD)」ウィンドウが表示されます。
3. **「DTD ファイル名 (DTD file name)」**フィールドに DTD ファイル名を入力するか、または**「...」** をクリックして既存の DTD ファイルをブラウズします。
4. **DTD ID** フィールドに DTD ID を入力します。

   DTD ID は、DTD の ID で、ローカル・システム上の DTD の場所を指定するパスです。 DTD ID は、DAD ファイル内で <DTDID> エレメントに指定されている値と同じでなければなりません。
5. オプションで、**「作成者 (Author)」**フィールドに DTD 作成者名を入力します。

   作成者名が DTD に指定されていれば、XML エクステンダーは自動的にそれを表示します。
6. **「終了 (Finish)」**をクリックして、 DTD を DTD リポジトリー表 DB2XML.DTD_REF に挿入し、「ランチパッド (LaunchPad)」ウィンドウに戻ります。

DB2 コマンド・シェルから

以下の表7 に示すスキーマを使用して、 DTD_REF 表の SQL INSERT ステートメントを発行します。

表 7. DTD_REF DTD 表のスキーマ

列名	データ・タイプ	説明
DTDID	VARCHAR(128)	(一意的で、NULL ではない) 基本キー。妥当性検査の時、基本キーは DTD の識別に使われます。基本キーは各 XML 文書の DOCTYPE 行の SYSTEM ID と同じでなければなりません。基本キーが DAD ファイル内に指定されている場合、 DAD ファイルは DTD の定義したスキーマに従う必要があります。
CONTENT	XMLCLOB	DTD の内容。
USAGE_COUNT	INTEGER	データベース内において、この DTD を使って DAD を定義している XML 列および XML コレクションの数。
AUTHOR	VARCHAR(128)	DTD の作成者 (この情報の入力はオプションです)。
CREATOR	VARCHAR(128)	最初にデータ挿入を行ったユーザー ID。
UPDATOR	VARCHAR(128)	最後に更新を行ったユーザー ID。

たとえば次のようにします。

```
DB2 INSERT into db2xml.dtd_ref values('c:¥dxx¥samples¥dtd¥getstart.dtd',
    db2xml.XMLClobFromFile('c:¥dxx¥samples¥dtd¥getstart.dtd'), 0, 'user1',
    'user1', 'user1')
```

XML コレクションの場合の重要事項: DTD ID は、ローカル・システム上の DTD の場所を指定するパスです。 DTD ID は、DAD ファイル内で <DTDID> エレメントに指定されている値と同じでなければなりません。

XML 列またはコレクションの定義

以下の節では、データベースを XML 列または XML コレクション用にセットアップして定義する方法と、必要なデータ・マッピング体系を準備する方法を説明します。

次のような節があります。

- 83ページの『XML 列を処理する』

- 96ページの『XML コレクションの処理』

XML 列を処理する

XML 列をセットアップするには、XML データにアクセスするための DAD ファイルを定義して、 XML 表内の列で XML データを使用可能にする必要があります。 DAD を作成する上での重要な概念は、ロケーション・パス構文 (DB2 表に索引付けしたいエレメントおよび属性値をマップするために使用される) を理解しておくことです。ロケーション・パスとその構文の詳細については、 58ページの『ロケーション・パス』を参照してください。

DAD ファイルの作成と編集

DAD ファイルを指定する時には、検索対象データの属性とかぎとなるエレメントを定義します。データの索引を使用してデータをす早く検索するために、XML エクステンダーはこの情報を使ってサイド表を作成します。 DAD の作成計画についての詳細は、61ページの『DAD ファイル』を参照してください。

作業を始める前に

- データの索引を使用してデータをす早く検索するためにかぎとなるエレメントと属性を定義するには、 XML データの階層構造を理解しておく必要があります。
- XML 文書の DTD を用意して、DTD_REF 表に挿入します。妥当性検査を行うには、このステップが必須です。

管理ウィザードの使用

DAD ファイルを作成するには、以下のステップに従います。

1. 管理ウィザードをセットアップして開始します。詳しくは 75ページの『管理ウィザードの開始』を参照してください。
2. XML DAD ファイルを編集または作成するために、「ランチパッド (LaunchPad)」ウィンドウから**「DAD ファイルを処理する (Work with DAD files)」**をクリックします。 「DAD ファイルの指定 (Specify a DAD file)」ウィンドウが開きます。
3. 既存の DAD ファイルを編集するか、または新規の DAD ファイルを作成するかを指定します。
   - **既存の DAD を編集するには、以下のようにします。**
     a. **「...」**をクリックしてプルダウン・メニューで既存の DAD ファイルをブラウズするか、または DAD ファイル名を**「ファイル名 (File name)」**フィールドに入力します。

b. 指定した DAD ファイルがウィザードに認識されるかどうかを確認します。
   - 指定した DAD ファイルをウィザードが認識した場合は**「次へ (Next)」**が選択可能になり、 XML 列が**「タイプ (Type)」**フィールドに表示されます。
   - 指定した DAD ファイルをウィザードが認識しない場合は、**「次へ (Next)」**が選択不可になります。 DAD ファイル名を**「ファイル名 (File name)」**フィールドに再び入力するか、または**「開く (Open)」**をクリックして既存の DAD ファイルを再びブラウズします。 **「次へ (Next)」**が選択可能になるまでこれを繰り返します。

c. **「次へ (Next)」**をクリックします。

- **新規の DAD を作成するには、以下のようにします。**
  a. **「ファイル名 (File name)」**フィールドをブランクのままにしておきます。
  b. **「タイプ (Type)」**メニューから**「XML 列 (XML column)」**をクリックします。
  c. **「次へ (Next)」**をクリックします。

4. 「妥当性検査の選択 (Select Validation)」ウィンドウで、 DTD を使用して XML を妥当性検査するかどうかを選択します。
   - 妥当性検査を行うには、以下のようにします。
     a. **「DTD を使って XML 文書の妥当性検査を行う (Validate XML documents with the DTD)」**をクリックします。
     b. **「DTD ID」**メニューから、使用する DTD を選択します。

     データベース用の DTD リポジトリーに DTD をまだインポートしていない場合、 XML 文書の妥当性検査を行うことはできません。
   - **「DTD を使って XML 文書の妥当性検査を行わない (Do NOT Validate XML documents with the DTD)」**をクリックすると、 XML 文書の妥当性検査を行わないで続行します。
5. **「次へ (Next)」**をクリックします。
6. 「サイド表 (Side tables)」ウィンドウから、新規のサイド表の追加、既存のサイド表の編集、または既存のサイド表の除去を選択します。
   - **新規のサイド表またはサイド表の列を追加するには、以下のようにします。**

     新規のサイド表を追加するには、表内の列を定義します。サイド表内の列ごとに以下のステップを完了してください。

a. 「サイド表 (Side tables)」ウィンドウの**「詳細 (Details)」**ボックスの各フィールドに入力します。

1) **「表名 (Table name)」**: その列を含む表の名前を入力します。たとえば次のようにします。

```
ORDER_SIDE_TAB
```

2) **「列名 (Column name)」**: 列の名前を入力します。たとえば次のようにします。

```
CUSTOMER_NAME
```

3) **「タイプ (Type)」:** 列のタイプをメニューから選択します。 たとえば次のようにします。

```
XMLVARCHAR
```

4) **「長さ (Length)」(VARCHAR タイプのみ):** VARCHAR 文字の最大数を入力します。 たとえば次のようにします。

```
30
```

5) **「パス (Path)」**: エレメントまたは属性のロケーション・パスを入力します。たとえば次のようにします。

```
/ORDER/CUSTOMER/NAME
```

ロケーション・パス構文については、58ページの『ロケーション・パス』を参照してください。

6) **「複数出現 (Multi occur)」**: メニューから**「いいえ (No)」**または**「はい (Yes)」**を選択します。

1 つの文書内でこのエレメントまたは属性のロケーション・パスを 2 度以上使用できるかどうかを示します。

重要: 複数オカレンスに列を指定する場合、列を含むサイド表に 1 つの列のみを指定することができます。

b. **「追加 (Add)」**をクリックして、列を追加します。

c. サイド表の列の追加、編集、または除去を繰り返すか、または**「次へ (Next)」**をクリックします。

- **既存のサイド表の列を編集するには、以下のようにします。**

既存の列の定義を変更することによってサイド表を更新することができます。

a. 編集したいサイド表および列名をクリックします。

b. **「詳細 (Details)」**ボックスの各フィールドを編集します。

c. **「変更 (Change)」**をクリックして変更内容を保管します。

d. サイド表ごとに列の追加、編集、または除去を繰り返すか、または **「次へ (Next)」** をクリックします。

- **既存のサイド表の列を除去するには、以下のようにします。**
  a. 除去したいサイド表および列をクリックします。
  b. **「除去 (Remove)」** をクリックします。
  c. サイド表の列の追加、編集、または除去を繰り返すか、または **「次へ (Next)」** をクリックします。
- **既存のサイド表を除去するには、以下のようにします。**
  サイド表全体を除去するには、表内の各列を削除します。
  a. 除去したい表の各サイド表の列をクリックします。
  b. **「除去 (Remove)」** をクリックします。
  c. サイド表の列の追加編集、または除去を繰り返すか、または **「次へ (Next)」** をクリックします。

7. 変更後の DAD の出力ファイル名を、「DAD の指定 (Specify a DAD)」ウィンドウの **「ファイル名 (File name)」** フィールドに入力します。
8. **「終了 (Finish)」** をクリックして DAD ファイルを保管し、「ランチパッド (LaunchPad)」ウィンドウに戻ります。

DB2 コマンド・シェルから

DAD ファイルは、任意のテキスト・エディターで作成できる XML ファイルです。

DAD ファイルを作成するには、以下のステップに従います。

1. テキスト・エディターを開きます。
2. 以下の構文を使用して、DAD ファイル・ヘッダーを作成します。

```
<?xml version="1.0"?>
<!DOCTYPE DAD SYSTEM "path¥dtd¥dad.dtd"> --> the path and file name of
       the DTD for the DAD file
```

3. <DAD></DAD> タグを挿入します。
4. オプションで、<DAD> タグの間に、妥当性検査のためにこの DAD ファイルを XML 文書 DTD に関連付ける DTD ID を次のように指定します。

```
<dtdid>path¥dtd_name.dtd</dtdid> --> the path and file name of the DTD
       for your application
```

DTD ID は妥当性検査のために必要であり、 DTD 参照表 (db2xml.DTD_REF) に DTD を挿入するときに使用される DTD ID 値と一致していなければなりません。

5. 妥当性検査 (つまり XML 文書が有効かどうかを DTD を使って検査すること) を行うかどうかを指定します。たとえば次のようにします。

```
<validation>YES</validation>   --> specify YES or NO
```

YES を指定するには、すでに前のステップで DTD ID を指定して、 DTD を DTD_REF 表に挿入していなければなりません。

6. <Xcolumn> エレメントを使用して、アクセスおよび保管の方式を XML 列と定義します。

```
<Xcolumn>
</Xcolumn>
```

7. それぞれのサイド表を定義して、階層の検索用に索引を付ける対象となる重要なエレメントと属性を定義します。各表ごとに、以下のステップを行います。なお、以下のステップでは、293ページの『DAD ファイル: XML 列』の DAD ファイルの例を使用しています。

   a. <TABLE></TABLE> タグおよび名前属性を挿入します。

```
<table name="order_tab">
</table>
```

   b. <TABLE> タグの後に、<COLUMN> タグと、表内のそれぞれの列についての属性を挿入します。

      - ***name***: 列の名前
      - ***type***: 列のタイプ
      - ***「パス (path)」***: エレメントまたは属性のロケーション・パス。ロケーション・パス構文については、58ページの『ロケーション・パス』を参照してください。
      - ***multi_occurrence***: 1 つの文書内でこのエレメントまたは属性を 2 度以上使用可能にするかどうかを示します。

```
<table ...>
   <column name="order_key"
           type="integer"
           path="/Order/@key"
           multi_occurrence="NO"/>
   <column name="customer"
           type="varchar(50)"
           path="/Order/Customer/Name"
           multi_occurrence="NO"/>
</table>
```

8. 最後の列定義の後に、終了タグ </TABLE> があることを確認します。
9. 最後の </TABLE> の後に、終了タグ </Xcolumn> があることを確認します。
10. </Xcolumn> タグの後に、終了タグ </DAD> があることを確認します。

XML 表の作成または更新

表の中に完全な XML 文書を保管するには、 XML ユーザー定義タイプ (UDT) の列を含むように表を作成または更新しなければなりません。この表を *XML 表* といいます。これは XML 文書を含んでいる表です。 この表は、更新された表または新規作成のどちらでもかまいません。表に XML タイプの列が 1 つあれば、その列を XML 用に使用可能にすることができます。

既存の表に XML タイプの列を含むように変更するには、管理ウィザードまたは DB2 コマンド・シェルを使用します。

管理ウィザードの使用

1. 管理ウィザードをセットアップして開始します。詳しくは 75ページの『管理ウィザードの開始』を参照してください。
2. 「ランチパッド (LaunchPad)」ウィンドウから**「XML 列を処理する (Work with XML columns)」**をクリックします。「タスクの選択 (Select a task)」ウィンドウが開きます。
3. **「XML 列を追加する (Add an XML Column)」**をクリックします。「XML 列の追加 (Add an XML column)」ウィンドウが開きます。
4. **「表名 (Table name)」**プルダウン・メニューから表名を選択するか、変更したい表の名前を入力します。たとえば次のようにします。

   ```
   SALES_DB
   ```

5. **「列名 (Column name)」**フィールド内の表に追加される列の名前を入力します。たとえば次のようにします。

   ```
   ORDER
   ```

6. **「列タイプ (Column type)」**プルダウン・メニューから列用の UDT を選択します。たとえば次のようにします。

   ```
   XMLVARCHAR
   ```

7. **「終了 (Finish)」**をクリックして、 XML タイプの列を追加します。

DB2 コマンド・シェルから

CREATE TABLE または ALTER TABLE ステートメントの列文節で、 XML タイプの列を含む表を作成または変更します。

例: これは、sales というアプリケーションで、アプリケーション表 SALES_TAB の ORDER という列の中に、 XML 形式の行項目 order を保管する場合です。 この表には、INVOICE_NUM および SALES_PERSON という列もあります。 注文数が多くないため、XMLVARCHAR タイプを使って保管

するとします。 基本キーは INVOICE_NUM です。以下の CREATE TABLE ステートメントによって、XML タイプの列を 1 つ含んだ表を作成します。

```
CREATE TABLE sales_tab(
             invoice_num  char(6) NOT NULL PRIMARY KEY,
             sales_person     varchar(20),
             order            XMLVarchar);
```

XML 列の使用可能化

XML 文書を DB2 データベースに保管するには、 1 つの列を XML に関して使用可能にする必要があります。列を使用可能して索引付けできるようにすると、す早く検索することができます。列を使用可能にするには、XML エクステンダー管理ウィザードまたは DB2 コマンド・シェルを使用します。列のタイプは XML でなければなりません。

XML エクステンダーは、XML 列を使用可能にすると、以下の操作を行います。

- オプションで、以下の目的で DAD ファイルを読み取ります。
  - DAD ファイル用の DTD に照らして、その DAD ファイルを妥当性検査する。
  - 指定されていれば、DTD_REF 表から DTD ID を検索する。
  - XML 列の索引用にサイド表を作成する。
  - XML データを入れる列を用意する。
- オプションで、XML およびサイド定義のデフォルト視点 が定義されていればそれらを作成します。
- まだルート ID 値が指定されていなければ、これを指定します。

XML 列が使用可能になったら、以下のことを行うことができます。

- サイド表に索引を作成する
- XML 列に XML 文書を挿入する
- XML 列内の XML 文書を照会、更新、または検索する

作業を始める前に

XML UDT の列を含む DB2 表を作成または更新することによって、XML 表を作成します。

管理ウィザードの使用

XML 列を使用可能にするには、以下のステップに従います。

1. 管理ウィザードをセットアップして開始します。詳しくは 75ページの『管理ウィザードの開始』を参照してください。

2. 「ランチパッド (LaunchPad)」ウィンドウから**「XML 列を処理する (Work with XML Columns)」**をクリックし、 XML エクステンダーの列関連タスクを表示します。「タスクの選択 (Select a Task)」ウィンドウが開きます。
3. データベース内の既存の表列を使用可能にするために、**「列を使用可能にする (Enable a Column)」**をクリックしてから、**「次へ (Next)」**をクリックします。
4. **「表名 (Table name)」**フィールドから、 XML 列を含む表を選択します。たとえば次のようにします。

   ```
   SALES_TAB
   ```

5. **「列名 (Column name)」**フィールドから、使用可能にする列を選択します。 たとえば次のようにします。

   ```
   ORDER
   ```

   列のタイプは XML でなければなりません。
6. DAD パスおよびファイル名を**「DAD ファイル名 (DAD file name)」**フィールドに入力するか、または**「...」**をクリックして既存の DAD ファイルをブラウズします。たとえば次のようにします。

   ```
   c:¥dxx¥samples¥dad¥getstart.dad
   ```

7. オプションで、既存の表の名前を**「表スペース (Table space)」**フィールドに入力します。

   表スペースには XML エクステンダーの作成したサイド表が含まれます。表スペースが指定されている場合、サイド表は指定された表スペースの中に作成されます。表スペースが指定されていない場合、サイド表はデフォルト表スペースの中に作成されます。
8. オプションで、デフォルト視点の名前を**「デフォルト視点 (Default view)」**フィールドに入力します。

   指定される場合、デフォルト視点は、列が使用可能になるときに自動的に作成され、 XML 表と関連するすべてのサイド表を結合します。
9. オプションで、アプリケーション表内の基本キーの列名を「**ルート ID (Root ID)**」フィールドに入力します。これを入力することを推奨します。

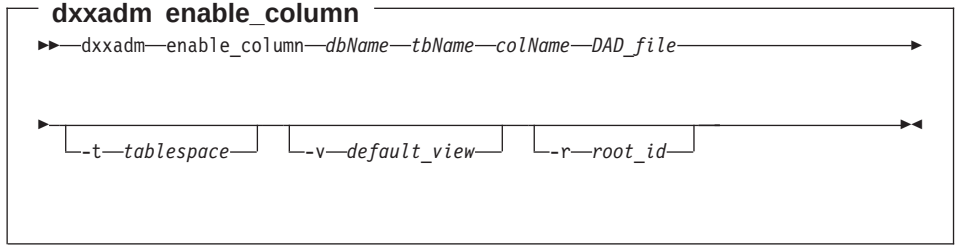
   XML エクステンダーはすべてのサイド表をアプリケーション表に関連付けるために、一意的な ID としてルート ID の値を使用します。これが指定されていない場合、 XML エクステンダーは DXXROOT_ID 列をアプリケーション表に追加して ID を生成します。
10. **「終了 (Finish)」**をクリックして、 XML 列を使用可能にし、サイド表を作成し、「ランチパッド (LaunchPad)」ウィンドウに戻ります。

- 列が正常に使用可能化されると、 `Enabled column is successful` というメッセージが表示されます。
- 列を正常に使用可能化できなかった場合は、エラー・ボックスが表示されます。列が正常に使用可能になるまで入力フィールドの値を訂正してください。

DB2 コマンド・シェルから

XML 列を使用可能にするには、以下のようなコマンドを入力します。

構文:



パラメーター:

dbName
: データベースの名前。

tbName
: 使用可能にする列を含む表の名前。

colName
: 使用可能にする XML 列の名前。

DAD_file
: 文書アクセス定義 (DAD) が入っているファイルの名前。

tablespace
: XML エクステンダーの作成したサイド表が含まれている、作成済みの表スペース。 これを指定しない場合、デフォルトの表スペースが使用されます。

default_view
: 任意指定。関連するすべてのサイド表をアプリケーション表に結合するために XML エクステンダーが作成した、デフォルト視点の名前。

root_id
: 任意指定。アプリケーション表内の基本キーの列名、およびすべてのサイド表をアプリケーション表に関連付ける一意的な ID。 XML エクス

テンダーはすべてのサイド表をアプリケーション表に関連付けるために、一意的な ID として *root_id* 値を使用します。 ルート ID を指定することを推奨します。ルート ID が指定されていない場合、 XML エクステンダーは DXXROOT_ID 列をアプリケーション表に追加して ID を生成します。

制限: アプリケーション表の列名の 1 つが DXXROOT_ID で、この列に *root_id* の値が入っていない場合は、ユーザーが *root_id* パラメーターを指定しなければなりません。こうしないとエラーが発生します。

例: 以下の例では、 DB2 コマンド・シェルを使用して列を使用可能にします。 この DAD ファイルおよび XML 文書は、291ページの『付録B. サンプル』に出ています。

```
dxxadm enable_column SALES_DB sales_tab order getstart.dad
            -v sales_order_view -r invoice_num
```

この例では、列 ORDER が表 SALES_DB.SALES_TAB 内で使用可能になります。 DAD ファイルは getstart.dad、デフォルト視点は sales_order_view、ルート ID は INVOICE_NUM です。

この例を使用すると、表 SALES_TAB のスキーマは次のとおりです。

列名	INVOICE_NUM	SALES_PERSON	ORDER
データ・タイプ	CHAR(6)	VARCHAR(20)	XMLVARCHAR

DAD 指定に基づいて、以下のサイド表が作成されます。

ORDER_SIDE_TAB:

列名	ORDER_KEY	CUSTOMER	INVOICE_NUM
データ・タイプ	INTEGER	VARCHAR(50)	CHAR(6)
パス式	/Order/@key	/Order/Customer/Name	N/A

PART_SIDE_TAB:

列名	PART_KEY	PRICE	INVOICE_NUM
データ・タイプ	INTEGER	DOUBLE	CHAR(6)
パス式	/Order/Part/@key	/Order/Part/ExtendedPrice	N/A

SHIP_SIDE_TAB:

列名	DATE	INVOICE_NUM
データ・タイプ	DATE	CHAR(6)
パス式	/Order/Part/Shipment/ShipDate	N/A

すべてのサイド表には同じタイプの列 INVOICE_NUM があります。これは、アプリケーション表の基本キー INVOICE_NUM によってルート ID が指定されているためです。 列が使用可能になった後、INVOICE_NUM の値がサイド表に挿入されます。 XML 列 ORDER を使用可能にする際に *default_view* パラメーターを指定したため、デフォルト視点 sales_order_view が作成されます。この視点は、以下のステートメントを使って上記の表をすべて結合します。

```
CREATE VIEW sales_order_view(invoice_num, sales_person, order,
                                order_key, customer, part_key, price, date)
     AS
     SELECT sales_tab.invoice_num, sales_tab.sales_person, sales_tab.order,
            order_tab.order_key, order_tab.customer,
            part_tab.part_key, part_tab.price,
            ship_tab.date
     FROM sales_tab, order_tab, part_tab, ship_tab
     WHERE sales_tab.invoice_num = order_tab.invoice_num
               AND sales_tab.invoice_num = part_tab.invoice_num
               AND sales_tab.invoice_num = ship_tab.invoice_num
```

enable_column コマンドによって表スペースが指定されている場合、サイド表は指定された表スペースの中に作成されます。 表スペースが指定されていない場合、サイド表はデフォルト表スペースの中に作成されます。

サイド表の索引付け

XML 列を使用可能にし、サイド表を作成した後、サイド表を索引付けすることができます。サイド表には、DAD ファイルの作成時に指定した列内に XML データが入ります。これらの表の索引付けを行うと、XML 文書に対する照会のパフォーマンスを改善するのに助けになります。

作業を始める前に

- XML 文書構造用のサイド表を指定する DAD ファイルを作成します。
- DAD ファイルを使用して XML 列を使用可能にします。これにより、サイド表が作成されます。

DB2 CREATE INDEX コマンド

DB2 CREATE INDEX コマンドを使用します。

例:

以下の例では、4 つのサイド表に索引を作成します。

```
DB2 CREATE INDEX KEY_IDX
      ON ORDER_SIDE_TAB(ORDER_KEY)

DB2 CREATE INDEX CUSTOMER_IDX
      ON ORDER_SIDE_TAB(CUSTOMER)

DB2 CREATE INDEX PRICE_IDX
      ON PART_SIDE_TAB(PRICE)

DB2 CREATE INDEX DATE_IDX
      ON SHIP_SIDE_TAB(DATE)
```

XML 列を使用不可にする

XML 列用の DAD ファイルを更新する必要がある場合や、 XML 列または列を含む表を削除したい場合に、列を使用不可にします。列が使用不可になったら、更新された DAD ファイルを含む列を再び使用可能にし、列またはその他のタスクを削除することができます。列を使用不可にするには、 XML エクステンダー管理ウィザードまたは DB2 コマンド・シェルを使用することができます。

XML エクステンダーは、XML 列を使用可能にすると、以下の操作を行います。

- XML_USAGE 表から列の項目を削除する
- この列と関連したサイド表を除去する

重要: 最初に列を使用不可にしないで XML 列を含む表を除去する場合、XML エクステンダーは XML 列と関連したいかなるサイド表も除去できず、予期しない結果が生じることがあります。

作業を始める前に

使用不可にする XML 列が現行の DB2 データベース内に存在することを確認してください。

管理ウィザードの使用

XML 列を使用不可にするには、以下のステップに従います。

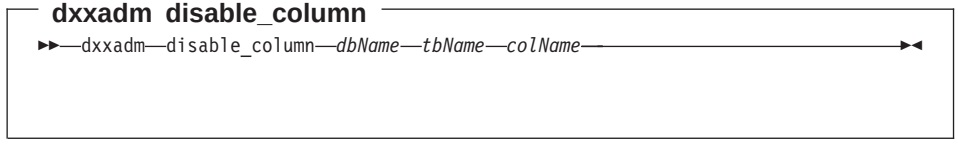
1. 管理ウィザードをセットアップして開始します。詳しくは 75ページの『管理ウィザードの開始』を参照してください。

2. 「ランチパッド (LaunchPad)」ウィンドウから**「XML 列を処理する (Working with XML Columns)」**をクリックし、 XML エクステンダーの列関連タスクを表示します。「タスクの選択 (Select a Task)」ウィンドウが開きます。
3. **「列を使用不可にする (Disable a Column)」**をクリックしてから、**「次へ (Next)」**をクリックし、データベース内の既存の表列を使用不可にします。
4. **「表名 (Table name)」**フィールドから、 XML 列を含む表を選択します。
5. **「列名 (Column name)」**フィールドから、使用不可にする列を選択します。
6. **「終了 (Finish)」**をクリックします。
   - 列が正常に使用不可にされると、 `Disabled column is successful` というメッセージが表示されます。
   - 列を正常に使用不可にできない場合は、エラー・ボックスが表示されます。列が正常に使用不可になるまで入力フィールドの値を訂正してください。

DB2 コマンド・シェルから

XML 列を使用不可にするには、以下のコマンドを入力します。

構文:



パラメーター:

dbName
データベースの名前。

tbName
使用不可にする列を含む表の名前。

colName
使用不可にする XML 列の名前。

例: 以下の例では、 DB2 コマンド・シェルを使用して列を使用不可にします。この DAD ファイルおよび XML 文書は、291ページの『付録B. サンプル』に出ています。

```
dxxadm disable_column SALES_DB sales_tab order
```

この例では、列 ORDER が表 SALES_DB.SALES_TAB 内で使用不可になります。

列が使用不可になると、サイド表が除去されます。

XML コレクションの処理

XML コレクションをセットアップするには、マッピング体系を作成する必要があります。またオプションで、DB2 表を DAD ファイルに関連付ける仮想名を使用してコレクションを使用可能にします。

XML コレクションの使用可能化は必須ではありませんが、これによってパフォーマンスが改善されます。

マッピング体系用の DAD ファイルの作成および編集

XML コレクションを使用する時には、DAD ファイルを作成する必要があります。 DAD ファイルは、XML データと複数のリレーショナル表との関連を定義します。 XML エクステンダーは DAD を使用して以下のことを行います。

- リレーショナル・データから XML 文書を構成する
- XML 文書をリレーショナル・データに分解する

XML 表と DB2 表の間でデータをマップする方式には SQL マッピングと RDB_node マッピングがあり、このどちらかを使用できます。

SQL マッピング

SQL ステートメント・エレメントを使用して、 XML データを入れる表と列の SQL 照会を指定します。 SQL マッピングは、XML 文書を合成するためにのみ使用できます。

RDB_node マッピング

XML エクステンダー独特のエレメントであるリレーショナル・データベース・ノードまたは RDB_node を使用します。このエレメントは XML データ用の表、列、条件、および順序を指定します。 RDB_node マッピングでは、 SQL ステートメントによるマッピングよりも複雑なマッピングが可能です。 RDB_node マッピングは、XML 文書の合成と分解の両方を行うために使用できます。

これら 2 つのマッピング方式は、どちらも *XPath* データ・モデル を使用しています (63ページの『DAD ファイル』を参照)。

作業を始める前に

- DB2 表と XML 文書の間の関連をマップします。このステップには、XML 文書の階層をマップして、文書内のデータが DB2 表にマップされる方法を指定することが含まれます。
- XML 文書の妥当性検査を行う場合、合成または分解している XML 文書用の DTD を DTD 参照表 db2xml.DTD_REF に挿入します。

SQL マッピングを使った XML 文書の合成

XML 文書を合成し、SQL を使用したいときには、SQL マッピングを使用してください。

管理ウィザードの使用: 以下のステップに従って、XML コレクションの SQL マッピングを使用して DAD ファイルを作成します。

SQL マッピングを使用して合成用の DAD ファイルを作成するには、以下のようにします。

XML 文書内のデータを取り出す表および列を定義するために XML 文書を合成し、 SQL ステートメントを使用したいときには、SQL マッピングを使用してください。

1. 管理ウィザードをセットアップして開始します。詳しくは 75ページの『管理ウィザードの開始』を参照してください。
2. 「ランチパッド (LaunchPad)」ウィンドウから**「DAD ファイルを処理する (Work with DAD files)」**をクリックします。「DAD の指定 (Specify a DAD)」ウィンドウが表示されます。
3. 既存の DAD ファイルを編集するか、または新規の DAD ファイルを作成するかを指定します。

   新規の DAD ファイルを作成するには、以下のようにします。

   a. **「ファイル名 (File name)」**フィールドをブランクのままにしておきます。

   b. **「タイプ (Type)」**メニューから、**「XML コレクションの SQL マッピング (XML collection SQL mapping)」**を選択します。

   c. **「次へ (Next)」**をクリックして、「妥当性検査の選択 (Select Validation)」ウィンドウを開きます。

   既存の DAD ファイルを編集するには、以下のようにします。

a. DAD ファイル名を**「ファイル名 (File name)」**フィールドに入力するか、または**「...」**をクリックして既存の DAD ファイルをブラウズします。

b. 指定した DAD ファイルがウィザードに認識されるかどうかを確認します。

- 指定した DAD ファイルをウィザードが認識した場合は**「次へ (Next)」**が選択可能になり、**「タイプ (Type)」**フィールドに、XML コレクションの SQL マッピング (XML collection SQL mapping) が表示されます。
- 指定した DAD ファイルをウィザードが認識しない場合は、**「次へ (Next)」**が選択不可になります。 DAD ファイル名を再入力するか、**「...」**をクリックして既存の DAD ファイルを再びブラウズします。**「次へ (Next)」**が選択可能になるまで入力フィールドの値を訂正してください。

c. **「次へ (Next)」**をクリックして、「妥当性検査の選択 (Select Validation)」ウィンドウを開きます。

4. 「妥当性検査の選択 (Select Validation)」ウィンドウで、 DTD を使用して XML を妥当性検査するかどうかを選択します。

- 妥当性検査を行うには、以下のようにします。

  a. **「DTD を使って XML 文書の妥当性検査を行う (Validate XML documents with the DTD)」**をクリックします。

  b. **「DTD ID」**メニューから、使用する DTD を選択します。

  データベース用の DTD リポジトリーに DTD をまだインポートしていない場合、 XML 文書の妥当性検査を行うことはできません。

- **「DTD を使って XML 文書の妥当性検査を行わない (Do NOT Validate XML documents with the DTD)」**をクリックすると、 XML 文書の妥当性検査を行わないで続行します。

5. **「次へ (Next)」**をクリックして、「テキストの指定 (Specify Text)」ウィンドウを開きます。

6. **「プロローグ (Prolog)」**フィールドにプロローグ名を入力し、合成される XML 文書のプロローグを指定します。

```
<?xml version="1.0"?>
```

既存の DAD を編集している場合は、**「プロローグ (Prolog)」**フィールドにプロローグが自動的に表示されます。

7. 「テキストの指定 (Specify Text)」ウィンドウの**「文書タイプ (Doctype)」**フィールドに、 XML 文書のタイプを入力します。 たとえば次のようにします。

```
! DOCTYPE DAD SYSTEM "c:¥dxx¥samples¥dtd¥getstart.dtd"
```

既存の DAD を編集している場合は、**「文書タイプ (Doctype)」**フィールドに文書タイプが自動的に表示されます。

8. **「次へ (Next)」**をクリックして、「SQL ステートメントの指定 (Specify SQL Statement)」ウィンドウを開きます。
9. **「SQL ステートメント (SQL statement)」**フィールドに有効な SQL SELECT ステートメントを入力します。たとえば次のようにします。

```
SELECT o.order_key, customer_name, customer_email, p.part_key, color, quantity,
price, tax, ship_id, date, mode from order_tab o, part_tab p,
table (select substr(char(timestamp(generate_unique())),16)
  as ship_id, date, mode, part_key from ship_tab) s
      WHERE o.order_key = 1 and
            p.price > 20000 and
            p.order_key = o.order_key and
            s.part_key = p.part_key
      ORDER BY order_key, part_key, ship_id
```

既存の DAD を編集している場合は、**「SQL ステートメント (SQL statement)」**フィールドに SQL ステートメントが自動的に表示されます。

10. **「SQL のテスト (Test SQL)」**をクリックして、 SQL ステートメントの妥当性をテストします。
    - SQL ステートメントが有効であれば、結果のサンプルが**「結果のサンプル (Sample results)」**フィールドに表示されます。
    - SQL ステートメントが無効であれば、エラー・メッセージが**「結果のサンプル (Sample results)」**フィールドに表示されます。エラー・メッセージは SQL SELECT ステートメントを訂正して再試行するよう指示します。
11. **「次へ (Next)」**をクリックして、「SQL マッピング (SQL Mapping)」ウィンドウを開きます。
12. 「SQL マッピング (SQL Mapping)」ウィンドウの左側のフィールドで、マップ元のエレメントまたは属性ノードをクリックしてこれを選択します。

    XML 文書内のエレメントおよび属性を、DB2 データに対応するエレメントおよび属性のノードにマップします。 これらのノードは、XML データから DB2 データへのパスを提供します。

    - **ルート・ノードを追加するには、以下のようにします。**
      a. **「ルート (Root)」**アイコンを選択します。

b. **「新規エレメント (New Element)」**をクリックして、新規ノードを定義します。

c. **「詳細 (Details)」**ボックスで、**「ノード・タイプ (Node type)」**を**「エレメント (Element)」**として指定します。

d. **「ノード名 (Node name)」**フィールドに最上位ノードの名前を入力します。

e. **「追加 (Add)」**をクリックして、新規ノードを作成します。

これで、ルート・ノードまたはエレメント (マップ内のその他すべてのエレメントおよび属性ノードの親である) の作成が完了しました。このノードに子エレメントおよび子属性を追加することができます。

- **子エレメントまたは子属性ノードを追加するには、以下のようにします。**

a. 左側のフィールドで親ノードをクリックし、子エレメントまたは子属性を追加します。

親ノードを選択していない間は、**「新規エレメント (New Element)」**が選択可能になりません。

b. **「新規エレメント (New Element)」**をクリックします。

c. **「詳細 (Details)」**ボックスの**「ノード・タイプ (Node type)」**メニューから、ノード・タイプを選択します。

「ノード・タイプ (Node type)」メニューには、マップ内のその場所で有効なノード・タイプのみが表示されます。

エレメント (Element)
XML 文書と関連した DTD 内に定義されている XML エレメントを表します。 XML エレメントを DB2 表内の列に関連付けるために使用されます。エレメント・ノードには、属性ノード、子エレメント・ノード、またはテキスト・ノードを含めることができます。最下位ノードには、ツリー表示内のノードと関連したテキスト・ノードおよび列名があります。

属性 (Attribute)
XML 文書と関連した DTD 内に定義されている XML 属性を表します。 XML 属性を DB2 表内の列に関連付けるために使用されます。属性ノードには、テキスト・ノードおよびツリー表示内のノードと関連した列名を含めることができます。

テキスト (Text)
リレーショナル表にマップされる内容を持つエレメントまたは属性ノードのテキストの内容を指定します。テキスト・ノードには、ツリー表示内のノードと関連した列名があります。

表 (Table)
リレーショナル表にマップされるエレメントまたは属性値の表名を指定します。

列 (Column)
リレーショナル表にマップされるエレメントまたは属性値の列名を指定します。

条件 (Condition)
列の条件を指定します。

d. **「詳細 (Details)」**ボックスの**「ノード名 (Node name)」**フィールドにノード名を入力します。 たとえば次のようにします。

```
Order
```

e. ノード・タイプとして基本レベルのエレメントに**「属性 (Attribute)」**、**「エレメント (Element)」**、または**「テキスト (Text)」**を指定した場合は、**「詳細 (Details)」**ボックスの**「列 (Column)」**フィールドから 1 つの列を選択します。たとえば次のようにします。

```
Customer_Name
```

制限: 新規の列は、管理ウィザードを使って作成することができません。ノード・タイプとして **「列 (Column)」**を指定した場合には、ご使用の DB2 データベースにすでに存在する列のみを選択できます。

f. **「追加 (Add)」**をクリックして、新規ノードを追加します。

あとでノードを変更するには、左側のフィールドでノードをクリックし、**「詳細 (Details)」**ボックスに必要な変更を行います。 エレメントを更新するには、**「変更 (Change)」**をクリックします。

追加プロセスを繰り返すノードを強調表示することによって、ノードに子エレメントまたは子属性を追加することもできます。

g. SQL マップの編集を続けるか、または**「次へ (Next)」**をクリックして「DAD の指定 (Specify a DAD)」ウィンドウを開きます。

- **ノードを除去するには、以下のようにします。**

a. 左側のフィールドで、ノードをクリックします。

b. **「除去 (Remove)」**をクリックします。

c. SQL マップの編集を続けるか、または**「次へ (Next)」**をクリックして「DAD の指定 (Specify a DAD)」ウィンドウを開きます。

最下位レベルのノードを除去する場合、別のエレメントが最下位レベルのノードになり、そのノード用に定義された列名が必要になります。

13. 変更後の DAD の出力ファイル名を、「DAD の指定 (Specify a DAD)」ウィンドウの**「ファイル名 (File name)」**フィールドに入力します。
14. **「終了 (Finish)」**をクリックして、「ランチパッド (LaunchPad)」ウィンドウに戻ります。

DB2 コマンド・シェルから: XML 文書を合成し、SQL を使用したいときには、SQL マッピング表記を使用します。

DAD ファイルは、任意のテキスト・エディターで作成できる XML ファイルです。以下のステップは、292ページの『文書アクセス定義ファイル』の付録サンプルの一部です。さらに包括的な情報やコンテキストについては、これらの例を参照してください。

1. テキスト・エディターを開きます。
2. 次のようにして DAD ヘッダーを作成します。

```
<?xml version="1.0"?>
<!DOCTYPE DAD SYSTEM "path¥dad.dtd"> --> the path and file name of the DTD
       for the DAD
```

3. <DAD></DAD> タグを挿入します。
4. <DAD> タグの後に、DAD ファイルを XML 文書 DTD に関連付ける DTD ID を指定します。

```
<dtdid>path¥dtd_name.dtd> --> the path and file name
       of the DTD for your application
```

5. 妥当性検査 (つまり XML 文書が有効かどうかを DTD を使って検査すること) を行うかどうかを指定します。たとえば次のようにします。

```
<validation>NO</validation>   --> specify YES or NO
```

6. <Xcollection> エレメントを使用して、アクセスおよび保管の方式を XML コレクションと定義します。アクセスおよび保管の方式は、 XML 文書の内容が DB2 表内に保管されているデータから派生することを定義します。

```
<Xcollection>
</Xcollection>
```

7. DB2 表からの照会を行う SQL ステートメント、または DB2 表にデータを挿入する SQL ステートメントを 1 つまたは複数指定します。 詳しく

は、68ページの『マッピング体系の要件』を参照してください。たとえば、以下の例のように、1 つの SQL 照会を指定します。

```
<SQL_stmt>
  SELECT o.order_key, customer_name, customer_email, p.part_key, color, quantity,
  price, tax, ship_id, date, mode from order_tab o, part_tab p,
  table (select substr(char(timestamp(generate_unique())),16)
    as ship_id, date, mode, part_key from ship_tab) s
        WHERE o.order_key = 1 and
              p.price > 20000 and
              p.order_key = o.order_key and
              s.part_key = p.part_key
        ORDER BY order_key, part_key, ship_id
</SQL_stmt>
```

8. 以下のプロローグ情報を追加します。

   ```
   <prolog>?xml version="1.0"?</prolog>
   ```

   上記と全く同じテキストが必要です。

9. <doctype></doctype> タグを追加します。たとえば次のようにします。

   ```
   <doctype>! DOCTYPE Order SYSTEM "c:¥dxx¥samples¥dtd¥getstart.dtd"</doctype>
   ```

10. <root_node></root_node> タグを使用してルート・ノードを定義します。root_node 内で、XML 文書を形成するエレメントおよび属性を指定します。

11. XML 文書内のエレメントおよび属性を、DB2 データに対応するエレメントおよび属性のノードにマップします。 これらのノードは、XML データから DB2 データへのパスを提供します。

    a. DB2 表内の列にマップする XML 文書内のそれぞれのエレメントごとに <element_node> を 1 つ定義します。

       ```
       <element_node name="name"></element_node>
       ```

       element_node には、以下のノードを入れることができます。

       - attribute_node
       - child element_node
       - text_node

    b. XML 文書内のそれぞれの属性ごとに、 DB2 表の 1 つの列にマップする <attribute_node> を 1 つ定義します。 DTD ファイルの例については、このセクションの最初に SQL マッピングを示しています。また、DAD ファイルの構文を詳細に説明した 283ページの『付録A. DAD ファイル用の DTD』では、 DAD ファイル用の DTD を示しています。

       たとえば、エレメント <Order> の属性キーが必要です。キーの値は列 PART_KEY に保管されています。

DAD ファイル: DAD ファイル内で、このキーの属性ノードを作成して、値 1 が保管される表の場所を指定します。

```
<attribute_node name="key">
    <column name="part_key"/>
</attribute_node>
```

合成された XML 文書: キーの値は PART_KEY 列から取られています。

```
<Order key="1">
```

12. DB2 表から派生している内容を持つそれぞれのエレメントまたは属性ごとに <text_node> を作成します。テキスト・ノードには、内容が提供されている列を指定する <column> エレメントが含まれています。

    たとえば、TAX という列から取られている値を持つ XML エレメント <Tax> を持っているとしましょう。

    DAD エレメント:

    ```
    <element_node name="Tax">
       <text_node>
       <column name="tax"/>
       </text_node>
    </element_node>
    ```

    列名は、DAD ファイルの最初にある SQL ステートメント内になければなりません。

    合成された XML 文書:

    ```
    <Tax>0.02</Tax>
    ```

    0.02 という値は、列 TAX から派生されます。

13. 終了のための </root_node> タグが最後の </element_node> タグの後にあることを確認します。
14. 終了のための </Xcollection> タグが </root_node> タグの後にあることを確認します。
15. 終了のための </DAD> タグが </Xcollection> タグの後にあることを確認します。

RDB_node マッピングを使った XML 文書の合成

XML 類似構造を使って XML 文書を合成するには、RDB_node マッピングを使用します。

この方式では、<RDB_node> を使用して、 DB2 表、列、および条件にエレメントまたは属性ノードを指定します。 <RDB_node> は、以下のエレメントを使用します。

- <table>: エレメントに対応する表を定義します。
- <column>: 対応するエレメントを含む列を定義します。
- <condition>: オプションで、列上に条件を指定します。

<RDB_node> 内に使われる子エレメントはノードのコンテキストに依存し、以下の規則に従います。

ノード・タイプ:	**RDB 子エレメントが使用されます。**		
	表	**列**	**条件 [1]**
ルート・エレメント	Y	N	Y
属性	Y	Y	オプション
テキスト	Y	Y	オプション

(1) 複数の表を使用する場合に必要

管理ウィザードの使用: **RDB_node マッピングを使用して合成用の DAD を作成するには、以下のようにします。**

1. 管理ウィザードをセットアップして開始します。詳しくは 75ページの『管理ウィザードの開始』を参照してください。
2. 「ランチパッド (LaunchPad)」ウィンドウから**「DAD ファイルを処理する (Work with DAD files)」**をクリックします。「DAD の指定 (Specify a DAD)」ウィンドウが表示されます。
3. 既存の DAD ファイルを編集するか、または新規の DAD ファイルを作成するかを指定します。

   既存の DAD を編集するには、以下のようにします。

   a. DAD ファイル名を**「ファイル名 (File name)」**フィールドに入力するか、または**「...」**をクリックして既存の DAD ファイルをブラウズします。

   b. 指定した DAD ファイルがウィザードに認識されるかどうかを確認します。

      - ウィザードが指定した DAD ファイルを認識する場合、**次へ** が選択可能になり、 XML コレクション RDB_node マッピングが **タイプ**・フィールドに表示されます。
      - 指定した DAD ファイルをウィザードが認識しない場合は、**「次へ (Next)」**が選択不可になります。 DAD ファイル名を**「ファイル名**

(File name)」フィールドに再び入力するか、または「**...**」をクリックして既存の DAD ファイルを再びブラウズします。 「**次へ (Next)**」が選択可能になるまで、これらのステップを繰り返します。

c. 「**次へ (Next)**」をクリックして、「妥当性検査の選択 (Select Validation)」ウィンドウを開きます。

新規の DAD を作成するには、以下のようにします。

a. 「**ファイル名 (File name)**」フィールドをブランクのままにしておきます。

b. 「**タイプ (Type)**」メニューから、「XML コレクションの RDB_node マッピング (XML collection RDB node mapping)」を選択します。

c. 「**次へ (Next)**」をクリックして、「妥当性検査の選択 (Select Validation)」ウィンドウを開きます。

4. 「妥当性検査の選択 (Select Validation)」ウィンドウで、 DTD を使用して XML を妥当性検査するかどうかを選択します。

- 妥当性検査を行うには、以下のようにします。

  a. 「**DTD を使って XML 文書の妥当性検査を行う (Validate XML documents with the DTD)**」をクリックします。

  b. 「**DTD ID**」メニューから、使用する DTD を選択します。

  データベース用の DTD リポジトリーに DTD をまだインポートしていない場合、 XML 文書の妥当性検査を行うことはできません。

- 「**DTD を使って XML 文書の妥当性検査を行わない (Do NOT Validate XML documents with the DTD)**」をクリックすると、 XML 文書の妥当性検査を行わないで続行します。

5. 「**次へ (Next)**」をクリックして、「テキストの指定 (Specify Text)」ウィンドウを開きます。

6. 「テキストの指定 (Specify Text)」ウィンドウの「**プロローグ (Prolog)**」フィールドにプロローグ名を入力します。

```
<?xml version="1.0"?>
```

既存の DAD を編集している場合は、「**プロローグ (Prolog)**」フィールドにプロローグが自動的に表示されます。

7. 「テキストの指定 (Specify Text)」ウィンドウの「**文書タイプ (Doctype)**」フィールドに、 XML 文書のタイプを入力します。

既存の DAD を編集している場合は、「**文書タイプ (Doctype)**」フィールドに文書タイプが自動的に表示されます。

8. **「次へ (Next)」**をクリックして、「RDB マッピング (RDB Mapping)」ウィンドウを開きます。
9. 「RDB マッピング (RDB Mapping)」ウィンドウの左側のフィールドで、マップ元のエレメントまたは属性ノードをクリックしてこれを選択します。

   XML 文書内のエレメントおよび属性を、DB2 データに対応するエレメントおよび属性のノードにマップします。 これらのノードは、XML データから DB2 データへのパスを提供します。
10. **ルート・ノードを追加するには、以下のようにします。**
    a. **「ルート (Root)」**アイコンを選択します。
    b. **「新規エレメント (New Element)」**をクリックして、新規ノードを定義します。
    c. **「詳細 (Details)」**ボックスで、**「ノード・タイプ (Node type)」**を**「エレメント (Element)」**として指定します。
    d. **「ノード名 (Node name)」**フィールドに最上位ノードの名前を入力します。
    e. **「追加 (Add)」**をクリックして、新規ノードを作成します。

       これで、ルート・ノードまたはエレメント (マップ内のその他すべてのエレメントおよび属性ノードの親である) の作成が完了しました。ルート・ノードには、表の子エレメントおよび結合条件があります。
    f. コレクションの部分であるそれぞれの表ごとに表ノードを追加します。
       1) ルート・ノード名を強調表示して、**「新規エレメント (New Element)」**を選択します。
       2) **「詳細 (Details)」**ボックスで、**「ノード・タイプ (Node type)」**を**「表 (Table)」**として指定します。
       3) **「表名 (Table name)」**から表名を選択します。表はすでに存在していなければなりません。
       4) **「追加 (Add)」**をクリックして、表ノードを追加します。
       5) 各表ごとに、以下のステップを繰り返します。
    g. 表ノード用の結合条件を追加します。
       1) ルート・ノード名を強調表示して、**「新規エレメント (New Element)」**を選択します。
       2) **「詳細 (Details)」**ボックスで、**「ノード・タイプ (Node type)」**を**「条件 (Condition)」**として指定します。
       3) **「条件 (Condition)」**フィールドで、以下の構文を使って結合条件を入力します。

```
table_name.table_column = table_name.table_column AND
table_name.table_column = table_name.table_column ...
```

4) **「追加 (Add)」**をクリックして、条件を追加します。

11. **エレメントまたは属性ノードを追加するには、以下のようにします。**
    a. 左側のフィールドで親ノードをクリックし、子エレメントまたは子属性を追加します。
    b. **「新規エレメント (New Element)」**をクリックします。親ノードを選択していない間は、**「新規エレメント (New Element)」**が選択可能になりません。
    c. **「詳細 (Details)」**ボックスの**「ノード・タイプ (Node type)」**メニューから、ノード・タイプを選択します。

       「ノード・タイプ (Node type)」メニューには、マップ内のその場所で有効なノード・タイプのみが示されます。**「エレメント (Element)」**または**「属性 (Attribute)」**。
    d. **「ノード名 (Node name)」**フィールドにノード名を指定します。
    e. **「追加 (Add)」**をクリックして、新規ノードを追加します。
    f. **エレメントまたは属性ノードの内容をリレーショナル表にマップするには、以下のようにします。**
       1) テキスト・ノードを指定します。
          a) 親ノードをクリックします。
          b) **「新規エレメント (New Element)」**をクリックします。
          c) **「ノード・タイプ (Node type)」**フィールドで、**「テキスト (Text)」**を選択します。
          d) **「追加 (Add)」**を選択して、ノードを追加します。
       2) 表ノードを追加します。
          a) 作成したばかりのテキスト・ノードを選択し、**「新規エレメント (New Element)」**をクリックします。
          b) **「ノード・タイプ (Node type)」**フィールドで、**「表 (Table)」**を選択し、表名にエレメントを指定します。
          c) **「追加 (Add)」**をクリックして、ノードを追加します。
       3) 列ノードを追加します。
          a) テキスト・ノードを再び選択し、**「新規エレメント (New Element)」**をクリックします。
          b) **「ノード・タイプ (Node type)」**フィールドで、**「列 (Column)」**を選択し、列名にエレメントを指定します。

c) **「追加 (Add)」**をクリックして、ノードを追加します。

制限: 新規の列は、管理ウィザードを使って作成することができません。 ノード・タイプとして Column (列) を指定した場合には、ご使用の DB2 データベースにすでに存在する列のみを選択できます。

4) オプションで、列の条件を追加します。

a) テキスト・ノードを再び選択し、**「新規エレメント (New Element)」**をクリックします。

b) **「ノード・タイプ (Node type)」**フィールドで、**「条件 (Condition)」**および以下の構文を持つ条件を選択します。

operator LIKE|<|>|= *value*

c) **「追加 (Add)」**をクリックして、ノードを追加します。

g. RDB マップの編集を続けるか、または**「次へ (Next)」**をクリックして「DAD の指定 (Specify a DAD)」ウィンドウを開きます。

12. **ノードを除去するには、以下のようにします。**

a. 左側のフィールドで、ノードをクリックします。

b. **「除去 (Remove)」**をクリックします。

c. RDB_node マップの編集を続けるか、または**「次へ (Next)」**をクリックして「DAD の指定 (Specify a DAD)」ウィンドウを開きます。

13. 変更後の DAD の出力ファイル名を、「DAD の指定 (Specify a DAD)」ウィンドウの**「ファイル名 (File name)」**フィールドに入力します。

14. **「終了 (Finish)」**をクリックして、ノードを除去し、「ランチパッド (LaunchPad)」ウィンドウに戻ります。

DB2 コマンド・シェルから: DAD ファイルは、任意のテキスト・エディターで作成できる XML ファイルです。以下のステップは、292ページの『文書アクセス定義ファイル』の付録サンプルの一部です。さらに包括的な情報やコンテキストについては、これらの例を参照してください。

1. テキスト・エディターを開きます。
2. 次のようにして DAD ヘッダーを作成します。

```
<?xml version="1.0"?>
<!DOCTYPE DAD SYSTEM "path¥dad.dtd"> --> the path and file name of the DTD
       for the DAD
```

3. <DAD></DAD> タグを挿入します。
4. <DAD> タグの後に、DAD ファイルを XML 文書 DTD に関連付ける DTD ID を指定します。

```
<dtdid>path¥dtd_name.dtd> --> the path and file name of the DTD
      for your application
```

5. 妥当性検査 (つまり XML 文書が有効かどうかを DTD を使って検査すること) を行うかどうかを指定します。たとえば次のようにします。

   ```
   <validation>NO</validation>   --> specify YES or NO
   ```

6. <Xcollection> エレメントを使用して、アクセスおよび保管の方式を XML コレクションと定義します。アクセスおよび保管の方式は、XML データが DB2 表のコレクション内に保管されることを定義します。

   ```
   <Xcollection>
   </Xcollection>
   ```

7. 以下のプロローグ情報を追加します。

   ```
   <prolog>?xml version="1.0"?</prolog>
   ```

   上記と全く同じテキストが必要です。

8. <doctype></doctype> タグを追加します。たとえば次のようにします。

   ```
   <doctype>! DOCTYPE Order SYSTEM "c:¥dxx¥samples¥dtd¥getstart.dtd"</doctype>
   ```

9. <root_node> を使用してルート・ノードを定義します。root_node 内で、XML 文書を形成するエレメントおよび属性を指定します。
10. XML 文書内のエレメントおよび属性を、DB2 データに対応するエレメントおよび属性のノードにマップします。これらのノードは、XML データから DB2 データへのパスを提供します。
    a. ルート・ノード element_node を定義します。この element_node には、以下のものが含まれています。
       - コレクションを指定するための結合条件を持つ table_nodes を指定する RDB_node
       - 子エレメント
       - 属性

       表ノードおよび条件を指定するには、以下のようにします。

       1) RDB_node エレメントを作成します。たとえば、次のようにします。

          ```
          <RDB_node>
          </RDB_node>
          ```

       2) XML 文書内に組み込まれるデータを含むそれぞれの表ごとに <table_node> を定義します。たとえば、列データが文書内にある 3 つの表 (ORDER_TAB、PART_TAB、および SHIP_TAB) を持っている場合には、それぞれの表ごとに表ノードを作成します。たとえば次のようにします。

```
<RDB_node>
<table name="ORDER_TAB">
<table name="PART_TAB">
<table name="SHIP_TAB"></RDB_node>
```

3) このコレクションを使用可能にする計画がある場合は、オプションで、表それぞれのキー列を指定してください。通常、キー属性は合成のためには必要とされません。しかし、コレクションを使用可能にする場合は、使用される DAD ファイルが合成と分解の両方をサポートする必要があります。たとえば次のようにします。

```
<RDB_node>
<table name="ORDER_TAB" key="order_key">
<table name="PART_TAB" key="part_key">
<table name="SHIP_TAB" key="date mode">
</RDB_node>
```

4) コレクション内に表の結合条件を定義します。構文は、以下のとおりです。

```
expression = expression AND
expression = expression
```

たとえば次のようにします。

```
<RDB_node>
<table name="ORDER_TAB">
<table name="PART_TAB">
<table name="SHIP_TAB">
<condition>
   order_tab.order_key = part_tab.order_key AND
   part_tab.part_key = ship_tab.part_key
</condition>
</RDB_node>
```

b. DB2 表内の列にマップする XML 文書内のそれぞれのエレメントごとに <element_node> タグを定義します。たとえば次のようにします。

```
<element_node name="name">
</element_node>
```

エレメント・ノードは、以下のタイプのエレメントのいずれかを持つことができます。

- <text_node>: エレメントが DB2 表に内容を持つことを指定します。エレメントには子エレメントがありません。
- <attribute_node>: 属性を指定します。属性ノードは、次のステップで定義されます。

text_node には、内容を DB2 表および列名にマップするための <RDB_node> が含まれています。

RDB_nodes は、内容を DB2 表にマップする最下位レベルのエレメントに使用されます。 RDB_node には次のような子エレメントがあります。

- <table>: エレメントに対応する表を定義します。
- <column>: 対応するエレメントを含む列を定義し、タイプ属性を持つ列タイプを指定します。
- <condition>: オプションで、列上に条件を指定します。

たとえば、TAX という列にマップする XML エレメント <Tax> を以下に示します。

XML 文書:

```
<Tax>0.02</Tax>
```

この場合、TAX 列の値は 0.02 になります。

```
<element_node name="Tax">
   <text_node>
      <RDB_node>
         <table name="part_tab"/>
         <column name="tax"/>
      </RDB_node>
   </text_node>
</element_node>
```

この例では、<RDB_node> が、<Tax> エレメントの値がテキスト値であることと、データの保管場所が PART_TAB 表の TAX 列であることを指定します。 DAD ファイルの例については、 292ページの『文書アクセス定義ファイル』で RDB_node マッピングを示しています。また、DAD ファイルの構文を詳細に説明した 283ページの『付録A. DAD ファイル用の DTD』では、 DAD ファイル用の DTD を示しています。

c. このコレクションを使用可能にする計画がある場合には、オプションで、各 <column> エレメントにタイプ属性を追加してください。通常、タイプ属性は合成のためには必要とされません。しかし、コレクションを使用可能にする場合は、使用される DAD ファイルが合成と分解の両方をサポートする必要があります。 たとえば次のようにします。

```
<column name="tax" type="real"/>
```

d. XML 文書内のそれぞれの属性ごとに、 DB2 表の 1 つの列にマップする <attribute_node> を 1 つ定義します。たとえば次のようにします。

```
<attribute_node name="key">
</attribute_node>
```

attribute_node には、属性値を DB2 表および列にマップするための <RDB_node> が含まれています。 <RDB_node> には次のような子エレメントがあります。

- <table>: エレメントに対応する表を定義します。
- <column>: 対応するエレメントを含む列を定義します。
- <condition>: オプションで、列上に条件を指定します。

たとえば、エレメント <Order> に属性 key を持ちたいとします。ここでは、キーの値が列 PART_KEY にすでに保管されていなければなりません。 DAD ファイルで、このキーの <attribute_node> を作成して、値が保管される表の場所を指定します。

DAD ファイル

```
<attribute_node name="key">
  <RDB_node>
    <table name="part_tab">
    <column name="part_key"/>
  <RDB_node>
</attribute_node>
```

合成された XML 文書:

```
<Order key="1">
```

11. 終了のための </root_node> タグが最後の </element_node> タグの後にあることを確認します。
12. 終了のための </Xcollection> タグが </root_node> タグの後にあることを確認します。
13. 終了のための </DAD> タグが </Xcollection> タグの後にあることを確認します。

XML 文書用のスタイル・シートの指定

文書を構成するときには、XML エクステンダー は、<stylesheet> エレメントを使用して、スタイル・シートの処理命令もサポートします。処理命令は、<Xcollection> ルート・エレメント内になければならず、 XML 文書構造用に定義した <doctype> および <prolog> とともに配置されていなければなりません。たとえば次のようにします。

```
<?xml version="1.0"?>
<!DOCTYPE DAD SYSTEM "c:¥dtd¥dad.dtd">
<DAD>
<SQL_stmt>
      ...
</SQL_stmt>
<Xcollection>
...
```

```
<prolog>...</prolog>
<doctype>...</doctype>
<stylesheet>?xml-stylesheet type="text/css" href="order.css"?</stylesheet>
<root_node>...</root_node>
...
</Xcollection>
...
</DAD>
```

RDB_node マッピングを使った XML 文書の分解

XML 文書を分解するには、RDB_node マッピングを使用します。この方式では、<RDB_node> を使用して、 DB2 表、列、および条件にエレメントまたは属性ノードを指定します。 <RDB_node> は、以下のエレメントを使用します。

- <table>: エレメントに対応する表を定義します。
- <column>: 対応するエレメントを含む列を定義します。
- <condition>: オプションで、列上に条件を指定します。

<RDB_node> 内に使われる子エレメントはノードのコンテキストに依存し、以下の規則に従います。

ノード・タイプ:	RDB 子エレメントが使用されます。		
	表	列	条件 [1]
ルート・エレメント	Y	N	Y
属性	Y	Y	オプション
テキスト	Y	Y	オプション

(1) 複数の表を使用する場合に必要

管理ウィザードの使用: **分解用の DAD を作成するには、以下のようにします。**

1. 管理ウィザードをセットアップして開始します。詳しくは 75ページの『管理ウィザードの開始』を参照してください。
2. 「ランチパッド (LaunchPad)」ウィンドウから**「DAD ファイルを処理する (Work with DAD files)」**をクリックします。「DAD の指定 (Specify a DAD)」ウィンドウが表示されます。
3. 既存の DAD ファイルを編集するか、または新規の DAD ファイルを作成するかを指定します。

   既存の DAD を編集するには、以下のようにします。

a. DAD ファイル名を**「ファイル名 (File name)」**フィールドに入力するか、または**「...」**をクリックして既存の DAD ファイルをブラウズします。

b. 指定した DAD ファイルがウィザードに認識されるかどうかを確認します。

- 指定した DAD ファイルをウィザードが認識した場合は、**「次へ (Next)」**が選択可能になり、**「タイプ (Type)」**フィールドに、XML コレクションの RDB ノード・マッピング (XML collection RDB node mapping) が表示されます。
- 指定した DAD ファイルをウィザードが認識しない場合は、**「次へ (Next)」**が選択不可になります。 DAD ファイル名を**「ファイル名 (File name)」**フィールドに再び入力するか、または**「...」**をクリックして既存の DAD ファイルを再びブラウズします。 **「次へ (Next)」**が選択可能になるまで、これらのステップを繰り返します。

c. **「次へ (Next)」**をクリックして、「妥当性検査の選択 (Select Validation)」ウィンドウを開きます。

新規の DAD を作成するには、以下のようにします。

a. **「ファイル名 (File name)」**フィールドをブランクのままにしておきます。

b. **「タイプ (Type)」**メニューから、「XML コレクションの RDB_node マッピング (XML collection RDB node mapping)」を選択します。

c. **「次へ (Next)」**をクリックして、「妥当性検査の選択 (Select Validation)」ウィンドウを開きます。

4. 「妥当性検査の選択 (Select Validation)」ウィンドウで、 DTD を使用して XML を妥当性検査するかどうかを選択します。

- 妥当性検査を行うには、以下のようにします。

  a. **「DTD を使って XML 文書の妥当性検査を行う (Validate XML documents with the DTD)」**をクリックします。

  b. **「DTD ID」**メニューから、使用する DTD を選択します。

  データベース用の DTD リポジトリーに DTD をまだインポートしていない場合、 XML 文書の妥当性検査を行うことはできません。

- **「DTD を使って XML 文書の妥当性検査を行わない (Do NOT Validate XML documents with the DTD)」**をクリックすると、 XML 文書の妥当性検査を行わないで続行します。

5. **「次へ (Next)」**をクリックして、「テキストの指定 (Specify Text)」ウィンドウを開きます。

6. XML 文書のみを分解する場合には、**「プロローグ (Prolog)」**フィールドを無視します。合成用と分解用の両方の DAD ファイルを使用している場合には、「テキストの指定 (Specify Text)」ウィンドウの**「プロローグ (Prolog)」**フィールドにプロローグ名を入力します。 XML 文書を DB2 データに分解している場合には、プロローグは必要ありません。

   ```
   <?xml version="1.0"?>
   ```

   既存の DAD を編集している場合は、**「プロローグ (Prolog)」**フィールドにプロローグが自動的に表示されます。

7. XML 文書のみを分解する場合には、**「文書タイプ (Doctype)」**フィールドを無視します。合成用と分解用の両方の DAD ファイルを使用している場合には、**「文書タイプ (Doctype)」**フィールドに XML 文書のタイプを入力します。

   既存の DAD を編集している場合は、**「文書タイプ (Doctype)」**フィールドに文書タイプが自動的に表示されます。

8. **「次へ (Next)」**をクリックして、「RDB マッピング (RDB Mapping)」ウィンドウを開きます。
9. 「RDB マッピング (RDB Mapping)」ウィンドウの左側のフィールドで、マップ元のエレメントまたは属性ノードをクリックしてこれを選択します。

   XML 文書内のエレメントおよび属性を、DB2 データに対応するエレメントおよび属性のノードにマップします。 これらのノードは、XML データから DB2 データへのパスを提供します。

10. **ルート・ノードを追加するには、以下のようにします。**
    a. **「ルート (Root)」**アイコンを選択します。
    b. **「新規エレメント (New Element)」**をクリックして、新規ノードを定義します。
    c. **「詳細 (Details)」**ボックスで、**「ノード・タイプ (Node type)」**を**「エレメント (Element)」**として指定します。
    d. **「ノード名 (Node name)」**フィールドに最上位ノードの名前を入力します。
    e. **「追加 (Add)」**をクリックして、新規ノードを作成します。

       これで、ルート・ノードまたはエレメント (マップ内のその他すべてのエレメントおよび属性ノードの親である) の作成が完了しました。ルート・ノードには、表の子エレメントおよび結合条件があります。

    f. コレクションの部分であるそれぞれの表ごとに表ノードを追加します。
       1) ルート・ノード名を強調表示して、**「新規エレメント (New Element)」**を選択します。

2) **「詳細 (Details)」**ボックスで、**「ノード・タイプ (Node type)」**を**「表 (Table)」**として指定します。

3) **「表名 (Table name)」**から表名を選択します。表はすでに存在していなければなりません。

4) **「表のキー (Table key)」**フィールド内に表のキー列を指定します。

5) **「追加 (Add)」**をクリックして、表ノードを追加します。

6) 各表ごとに、以下のステップを繰り返します。

g. 表ノード用の結合条件を追加します。

1) ルート・ノード名を強調表示して、**「新規エレメント (New Element)」**を選択します。

2) **「詳細 (Details)」**ボックスで、**「ノード・タイプ (Node type)」**を**「条件 (Condition)」**として指定します。

3) **「条件 (Condition)」**フィールドで、以下の構文を使って結合条件を入力します。

```
table_name.table_column = table_name.table_column AND
table_name.table_column = table_name.table_column ...
```

4) **「追加 (Add)」**をクリックして、条件を追加します。

このノードに子エレメントおよび子属性を追加することができます。

11. **エレメントまたは属性ノードを追加するには、以下のようにします。**

a. 左側のフィールドで親ノードをクリックし、子エレメントまたは子属性を追加します。

親ノードを選択していない間は、**「新規 (New)」**が選択可能になりません。

b. **「新規エレメント (New Element)」**をクリックします。

c. **「詳細 (Details)」**ボックスの**「ノード・タイプ (Node type)」**メニューから、ノード・タイプを選択します。

「ノード・タイプ (Node type)」メニューには、マップ内のその場所で有効なノード・タイプのみが示されます。**「エレメント (Element)」**または**「属性 (Attribute)」**。

d. **「ノード名 (Node name)」**フィールドにノード名を指定します。

e. **「追加 (Add)」**をクリックして、新規ノードを追加します。

f. **エレメントまたは属性ノードの内容をリレーショナル表にマップするには、以下のようにします。**

1) テキスト・ノードを指定します。

a) 親ノードをクリックします。

b) **「新規エレメント (New Element)」**をクリックします。

c) **「ノード・タイプ (Node type)」**フィールドで、**「テキスト (Text)」**を選択します。

d) **「追加 (Add)」**を選択して、ノードを追加します。

2) 表ノードを追加します。

a) 作成したばかりのテキスト・ノードを選択し、**「新規エレメント (New Element)」**をクリックします。

b) **「ノード・タイプ (Node type)」**フィールドで、**「表 (Table)」**を選択し、表名にエレメントを指定します。

c) **「追加 (Add)」**をクリックして、ノードを追加します。

3) 列ノードを追加します。

a) テキスト・ノードを再び選択し、**「新規エレメント (New Element)」**をクリックします。

b) **「ノード・タイプ (Node type)」**フィールドで、**「列 (Column)」**を選択し、列名にエレメントを指定します。

c) **「タイプ (Type)」**フィールド内に列の基本データ・タイプを指定し、タグなしデータを保管するためにはどの列タイプでなければならないかを指定します。

d) **「追加 (Add)」**をクリックして、ノードを追加します。

制限: 新規の列は、管理ウィザードを使って作成することができません。 ノード・タイプとして Column (列) を指定した場合には、ご使用の DB2 データベースにすでに存在する列のみを選択できます。

4) オプションで、列の条件を追加します。

a) テキスト・ノードを再び選択し、**「新規エレメント (New Element)」**をクリックします。

b) **「ノード・タイプ (Node type)」**フィールドで、**「条件 (Condition)」**および以下の構文を持つ条件を選択します。

operator LIKE|<|>|= *value*

c) **「追加 (Add)」**をクリックして、ノードを追加します。

これらのノードを変更するには、ノードを選択し、**「詳細 (Details)」**ボックス内のフィールドを変更し、**「変更 (Change)」**をクリックします。

g. RDB マップの編集を続けるか、または**「次へ (Next)」**をクリックして「DAD の指定 (Specify a DAD)」ウィンドウを開きます。

12. **ノードを除去するには、以下のようにします。**
    a. 左側のフィールドで、ノードをクリックします。
    b. **「除去 (Remove)」**をクリックします。
    c. RDB_node マップの編集を続けるか、または**「次へ (Next)」**をクリックして「DAD の指定 (Specify a DAD)」ウィンドウを開きます。
13. 変更後の DAD の出力ファイル名を、「DAD の指定 (Specify a DAD)」ウィンドウの**「ファイル名 (File name)」**フィールドに入力します。
14. **「終了 (Finish)」**をクリックして、ノードを除去し、「ランチパッド (LaunchPad)」ウィンドウに戻ります。

DB2 コマンド・シェルから: DAD ファイルは、任意のテキスト・エディターで作成できる XML ファイルです。以下のステップは、292ページの『文書アクセス定義ファイル』の付録サンプルの一部です。さらに包括的な情報やコンテキストについては、これらの例を参照してください。

1. テキスト・エディターを開きます。
2. 次のようにして DAD ヘッダーを作成します。

```
<?xml version="1.0"?>
<!DOCTYPE DAD SYSTEM "path¥dad.dtd"> --> the path and file name of the DTD
       for the DAD
```

3. <DAD></DAD> タグを挿入します。
4. <DAD> タグの後に、DAD ファイルを XML 文書 DTD に関連付ける DTD ID を指定します。

```
<dtdid>path¥dtd_name.dtd> --> the path and file name of the DTD
       for your application
```

5. 妥当性検査 (つまり XML 文書が有効かどうかを DTD を使って検査すること) を行うかどうかを指定します。たとえば次のようにします。

```
<validation>NO</validation>   --> specify YES or NO
```

6. <Xcollection> エレメントを使用して、アクセスおよび保管の方式を XML コレクションと定義します。アクセスおよび保管の方式は、XML データが DB2 表のコレクション内に保管されることを定義します。

```
<Xcollection>
</Xcollection>
```

7. 以下のプロローグ情報を追加します。

```
<prolog>?xml version="1.0"?</prolog>
```

上記と全く同じテキストが必要です。

8. <doctype></doctype> タグを追加します。たとえば次のようにします。

   ```
   <doctype>! DOCTYPE Order SYSTEM "c:¥dxx¥samples¥dtd¥getstart.dtd"</doctype>
   ```

9. <root_node></root_node> タグを使用して root_node を定義します。root_node 内で、XML 文書を形成するエレメントおよび属性を指定します。
10. <root_node> タグの後に、XML 文書内のエレメントおよび属性を、 DB2 データに対応するエレメントおよび属性のノードにマップします。 これらのノードは、XML データから DB2 データへのパスを提供します。
    a. 最上位のルート element_node を定義します。この element_node には、以下のものが含まれています。
       - コレクションを指定するための結合条件を持つ表ノード
       - 子エレメント
       - 属性

       表ノードおよび条件を指定するには、以下のようにします。

       1) RDB_node エレメントを作成します。たとえば、次のようにします。

          ```
          <RDB_node>
          </RDB_node>
          ```

       2) XML 文書内に組み込まれるデータを含むそれぞれの表ごとに <table_node> を定義します。たとえば、列データが文書内にある 3 つの表 (ORDER_TAB、PART_TAB、および SHIP_TAB) を持っている場合には、それぞれの表ごとに表ノードを作成します。たとえば次のようにします。

          ```
          <RDB_node>
          <table name="ORDER_TAB">
          <table name="PART_TAB">
          <table name="SHIP_TAB"></RDB_node>
          ```

       3) コレクション内に表の結合条件を定義します。構文は、以下のとおりです。

          ```
          expression = expression AND
          expression = expression ...
          ```

          たとえば次のようにします。

          ```
          <RDB_node>
          <table name="ORDER_TAB">
          <table name="PART_TAB">
          <table name="SHIP_TAB">
          <condition>
          ```

```
   order_tab.order_key = part_tab.order_key AND
   part_tab.part_key = ship_tab.part_key
</condition>
</RDB_node>
```

4) 各表ごとに基本キーを指定します。基本キーは、単一の列または複合キーという複数の列から構成されます。基本キーを指定するには、属性キーを RDB_node の表エレメントに追加します。以下の例では、 Order というルート element_node の RDB_node 内のそれぞれの表ごとに基本キーを定義します。

```
<element_node name="Order">
    <RDB_node>
         <table name="order_tab" key="order_key"/>
         <table name="part_tab" key="part_key price"/>
         <table name="ship_tab" key="date mode"/>
         <condition>
             order_tab.order_key = part_tab.order_key AND
             part_tab.part_key = ship_tab.part_key
         </condition>
    <RDB_node>
```

分解の際に指定された情報は、XML 文書の構成の時には無視されます。

コレクションを使用可能にする場合、使用される DAD ファイルは合成と分解の両方をサポートする必要があるため、キー属性は分解に必要です。

b. DB2 表内の列にマップする XML 文書内のそれぞれのエレメントごとに <element_node> タグを定義します。たとえば次のようにします。

```
<element_node name="name">
</element_node>
```

エレメント・ノードは、以下のタイプのエレメントのいずれかを持つことができます。

- <text_node>: エレメントが DB2 表に内容を持つことを指定します。エレメントには子エレメントがありません。
- <attribute_node>: 属性を指定します。属性ノードは次のステップで定義されます。
- 子エレメント

text_node には、内容を DB2 表および列名にマップするための RDB_node が含まれています。

RDB_nodes は、内容を DB2 表にマップする最下位レベルのエレメントに使用されます。 RDB_node には次のような子エレメントがあります。

- <table>: エレメントに対応する表を定義します。
- <column>: 対応するエレメントを含む列を定義します。
- <condition>: オプションで、列上に条件を指定します。

たとえば、TAX という列にタグなしの内容を保管したい XML エレメント <Tax> を持っているとしましょう。

XML 文書:

```
<Tax>0.02</Tax>
```

この場合、TAX 列に保管される値は `0.02` になります。

DAD ファイルでは、<RDB_node> を指定して、XML エレメントを DB2 表および列にマップします。

DAD ファイル:

```
<element_node name="Tax">
   <text_node>
      <RDB_node>
         <table name="part_tab"/>
         <column name="tax"/>
      </RDB_node>
   </text_node>
</element_node>
```

<RDB_node> は、<Tax> エレメントの値がテキスト値であることと、データの保管場所が PART_TAB 表の TAX 列であることを指定します。

c. XML 文書内のそれぞれの属性ごとに、 DB2 表の 1 つの列にマップする <attribute_node> を 1 つ定義します。たとえば次のようにします。

```
<attribute_node name="key">
</attribute_node>
```

attribute_node には、属性値を DB2 表および列にマップするための RDB_node が含まれています。 RDB_node には次のような子エレメントがあります。

- <table>: エレメントに対応する表を定義します。
- <column>: 対応するエレメントを含む列を定義します。
- <condition>: オプションで、列上に条件を指定します。

たとえば、エレメント <Order> の属性キーの例を以下に示します。ここでは、キーの値が列 PART_KEY にすでに保管されていなければなりません。

XML 文書:

```
<Order key="1">
```

DAD ファイル内でこのキーの attribute_node を作成して、値 1 が保管される表の場所を指定します。

DAD ファイル:

```
<attribute_node name="key">
  <RDB_node>
    <table name="part_tab">
    <column name="part_key"/>
  <RDB_node>
</attribute_node>
```

11. それぞれの attribute_node および text_node の RDB_node の列タイプを指定します。これにより、タグなしのデータが保管される各列のデータ・タイプが適切であることが保証されます。列タイプを指定するには、属性タイプを列エレメントに追加します。 以下の例では、列のタイプを INTEGER と定義します。

```
<attribute_node name="key">
    <RDB_node>
        <table name="order_tab"/>
        <column name="order_key" type="integer"/>
    </RDB_node>
</attribute_node>
```

12. 終了のための </root_node> タグが最後の </element_node> タグの後にあることを確認します。
13. 終了のための </Xcollection> タグが </root_node> タグの後にあることを確認します。
14. 終了のための </DAD> タグが </Xcollection> タグの後にあることを確認します。

XML コレクションの使用可能化

XML コレクションを使用可能にする時には、 XML 文書に関連付けられた表および列を識別するために DAD ファイルを構文解析して、制御情報を XML_USAGE 表に記録します。以下の目的のために、XML コレクションを使用可能化することができます。

- XML 文書を分解して、新規の DB2 表にデータを保管する
- 複数の DB2 表にすでに存在するデータから XML 文書を構成する

構成と分解に同じ DAD ファイルを使用すると、コレクションを構成と分解の両方に使用可能化できます。

XML コレクションを使用可能にするには、XML エクステンダー管理ウィザード、 **dxxadm** コマンド (enable_collection オプションを指定)、または XML エクステンダー・ストアード・プロシージャー dxxEnableCollection() を使用します。

管理ウィザードの使用

XML コレクションを使用可能にするには、以下のステップに従います。

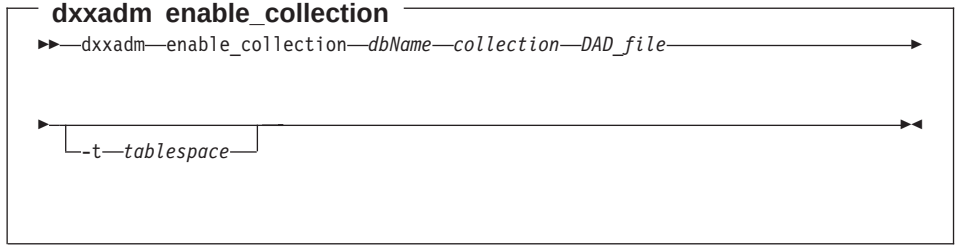
1. 管理ウィザードをセットアップして開始します。詳しくは 75ページの『管理ウィザードの開始』を参照してください。
2. 「ランチパッド (LaunchPad)」ウィンドウから**「XML コレクションを処理する (Work with XML Collections)」**をクリックします。「タスクの選択 (Select a Task)」ウィンドウが表示されます。
3. **「コレクションを使用可能にする (Enable a Collection)」**をクリックしてから、**「次へ (Next)」**をクリックします。「コレクションを使用可能にする (Enable a Collection)」ウィンドウが表示されます。
4. プルダウン・メニューから**「コレクション名 (Collection name)」**フィールド内で、使用可能にするコレクションの名前を選択します。
5. DAD ファイル名を**「DAD ファイル名 (DAD file name)」**フィールドに入力するか、または**「...」**をクリックして既存の DAD ファイルをブラウズします。
6. オプションで、すでに作成した表スペースを「**表スペース (Table space)**」フィールドに入力します。

   この表スペースには、分解の際に生成される新しい DB2 表が入ります。
7. **「終了 (Finish)」**をクリックして、コレクションを使用可能にし、「ランチパッド (LaunchPad)」ウィンドウに戻ります。
   - コレクションが正常に使用可能化されると、 `Enabled collection is successful` というメッセージが表示されます。
   - コレクションを正常に使用可能化できない場合は、エラー・ボックスが表示されます。コレクションが正常に使用可能化されるまで、上記のステップを繰り返してください。

DB2 コマンド・シェルから

XML コレクションを使用可能にするには、以下のような **dxxadm** コマンドを入力します。

構文:



パラメーター:

dbName
データベースの名前。

collection
XML コレクションの名前。この値は、XML コレクション・ストアード・プロシージャーのパラメーターとして使用されます。

DAD_file
文書アクセス定義 (DAD) が入っているファイルの名前。

tablespace
分解の際に生成された新規 DB2 表を保管する、既存の表スペース。これを指定しない場合、デフォルトの表スペースが使用されます。

例: 以下の例では、DB2 コマンド・シェルを使用して、データベース SALES_DB 内の sales_ord というコレクションを使用可能にします。 DAD ファイルは SQL マッピングを使用します (このファイルは、 294ページの『DAD ファイル: XML コレクション - SQL マッピング』に示されています)。

```
dxxadm enable_collection SALES_DB sales_ord getstart.dad
```

XML コレクションを使用可能にした後、 XML エクステンダー・ストアード・プロシージャーを使用して XML 文書を構成または分解することができます。

XML コレクションを使用不可にする

XML コレクションを使用不可にすると、表および列をコレクションの一部として識別するレコードを XML_USAGE 表から除去します。ただし、いかなる

データ表も除去されません。 DAD を更新してからコレクションを再び使用可能にする必要のあるとき、またはコレクションを除去するときに、コレクションを使用不可にします。

XML コレクションを使用不可にするには、XML エクステンダー管理ウィザード、 **dxxadm** コマンド (disable_collection オプションを指定)、または XML エクステンダー・ストアード・プロシージャー dxxDisableCollection() を使用します。

管理ウィザードの使用

XML コレクションを使用不可にするには、以下のステップに従います。

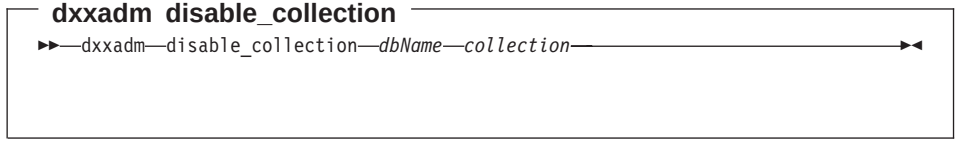
1. 管理ウィザードをセットアップして開始します。詳しくは 75ページの『管理ウィザードの開始』を参照してください。
2. 「ランチパッド (LaunchPad)」ウィンドウから**「XML コレクションを処理する (Work with XML Collections)」**をクリックし、 XML エクステンダーのコレクション関連タスクを表示します。「タスクの選択 (Select a Task)」ウィンドウが表示されます。
3. XML コレクションを使用不可にするには、**「XML コレクションを使用不可にする (Disable an XML Collection)」**をクリックしてから、**「次へ (Next)」**をクリックします。「コレクションを使用不可にする (Disable a Collection)」ウィンドウが表示されます。
4. **「コレクション名 (Collection name)」**フィールドに、使用不可にするコレクションの名前を入力します。
5. **「終了 (Finish)」**をクリックして、コレクションを使用不可にし、「ランチパッド (LaunchPad)」ウィンドウに戻ります。
   - コレクションが正常に使用不可になると、 `Disabled collection is successful` というメッセージが表示されます。
   - コレクションを正常に使用不可にできない場合は、エラー・ボックスが表示されます。コレクションが正常に使用不可になるまで、上記のステップを繰り返してください。

DB2 コマンド・シェルから

XML コレクションを使用不可にするには、以下のような **dxxadm** コマンドを入力します。

構文:

dxxadm disable_collection



パラメーター:

dbName
データベースの名前。

collection
XML コレクションの名前。この値は、XML コレクション・ストアード・プロシージャーのパラメーターとして使用されます。

例:

```
dxxadm disable_collection SALES_DB sales_ord
```

データベースを XML に関して使用不可にする

XML エクステンダー環境をクリーンアップし、XML エクステンダー UDT、UDF、ストアード・プロシージャー、および管理サポート表を除去したいときには、データベースを使用不可にします。 XML エクステンダーは、現行のインスタンスを使用して、接続先となるデータベースを使用不可にします。

あるデータベースを XML に関して使用不可にすると、 XML エクステンダーはそのデータベースに関して次のようなアクションを行います。

- すべてのユーザー定義タイプ (UDT) およびユーザー定義関数 (UDF) を削除します。
- XML エクステンダー用のメタデータを含む制御表を削除します。
- db2xml スキーマを削除します。

作業を始める前に

使用不可にするデータベース内の XML 列またはコレクションを使用不可にします。

管理ウィザードの使用

データベースで XML データを使用不可にするには、以下のステップに従います。

1. 管理ウィザードをセットアップして開始します。詳しくは 75ページの『管理ウィザードの開始』を参照してください。
2. 「ランチパッド (LaunchPad)」ウィンドウから**「データベースを使用不可にする (Disable database)」**をクリックして、現行のデータベースを使用不可にします。

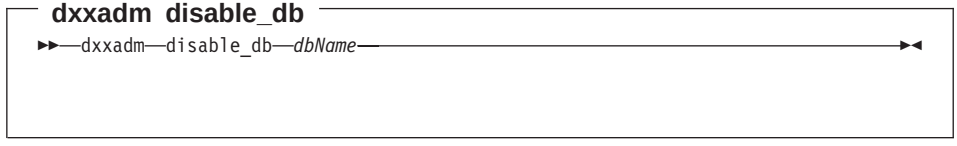
   データベースが現在使用可能になっていない場合、**「データベースを使用可能にする (Enable a database)」**のみ選択できます。

   データベースが使用不可になると、「ランチパッド (LaunchPad)」ウィンドウに戻ります。

DB2 コマンド・シェルから

コマンド行から、使用不可にするデータベースを指定して **dxxadm** と入力します。

構文:



パラメーター:

dbName
使用不可にするデータベースの名前。

例: SALES_DB という既存のデータベースを使用不可にします。

```
dxxadm disable_db SALES_DB
```

第3部 プログラミング

ここでは、XML データを管理するためのプログラミング技法を示します。

第5章 XML 列データの管理

XML 列を使用するとき、XML 文書の全体を列データとして保管します。このアクセスおよび保管の方式により、XML 文書を原形のまま保持しながら、文書に索引付けをして検索を行うこと、文書からデータを取り出すこと、および文書を更新することが可能になります。 XML 列には、XML 文書が DB2 におけるネイティブ形式で列データとして含まれています。データベースを XML について使用可能にした後、以下のユーザー定義タイプ (UDT) が使用できるようになります。

XMLCLOB
DB2 内に文字ラージ・オブジェクト (CLOB) として保管される XML 文書内容

XMLVARCHAR
DB2 内に VARCHAR として保管される XML 文書内容

XMLFile
ローカル・ファイル・システム上にファイルとして保管される XML 文書

XML UDT を列データ・タイプとして使用して、アプリケーション表を作成または更新することができます。 これらの表は、XML 表として知られています。 XML 用の表を作成または更新する方法を学ぶには、 88ページの『XML 表の作成または更新』 を参照してください。

列を XML について使用可能にした後、XML 列の内容の管理を開始できます。 XML 列を作成した後、以下の管理タスクを実行することができます。

- XML 文書を DB2 内に保管する
- XML データまたは文書を DB2 から取り出す
- XML 文書を更新する
- XML データまたは文書を削除する

これらのタスクを実行するには、以下の 2 つの方法を使用できます。

- *デフォルト・キャスト関数*、これは SQL 基本タイプを XML UDT に変換します。
- XML エクステンダーに備わっているユーザー定義関数 (UDF)。

本書では、タスクごとに両方の方法について解説します。

UDT 名および UDF 名

DB2 関数の完全な名前は、*schema-name.function-name* です。ここで、*schema-name* は SQL オブジェクトの論理グループ化を行うための ID です。XML エクステンダー UDF のためのスキーマ名は、db2xml です。 db2xml スキーマ名は、XML エクステンダー UDT の修飾子でもあります。本書では、関数名だけを参照します。

関数パスは、スキーマ名の順番付きリストです。 DB2 はリスト内のスキーマ名の順序を使用して、関数および UDT への参照を解決します。 SQL ステートメント SET CURRENT FUNCTION PATH を指定することにより、関数パスを指定できます。これにより、関数パスが CURRENT FUNCTION PATH 特殊レジスター内に設定されます。

XML エクステンダーでは、db2xml スキーマを関数パスに追加するのは良い考えです。これにより、XML エクステンダー UDF および UDT 名を、先頭に db2xml を付けずに入力することができます。以下の例は、db2xml スキーマを関数パスに追加する方法を示しています。

```
SET CURRENT FUNCTION PATH = db2xml, CURRENT FUNCTION PATH
```

重要: db2xml としてログオンする場合は、 db2xml を関数パス内の最初のスキーマとして追加しないでください。 db2xml としてログオンするときには、db2xml が自動的に最初のスキーマとして設定されます。その結果、関数パスが 2 つの db2xml スキーマで開始することになるので、エラー状態が生じます。

データを保管する

XML エクステンダーを使用して、原形の XML 文書を XML 列に挿入することができます。サイド表を定義する場合、XML エクステンダーは自動的にこれらの表を更新します。 XML 文書を直接保管する場合、XML エクステンダーは基本タイプを XML タイプとして保管します。

タスクの概説:

1. DAD ファイルを作成または更新したことを確認します。
2. 文書を保管するときに使用するデータ・タイプを決めます。
3. データを DB2 表に保管するための方式 (キャスト関数または UDF) を選択します。
4. XML 表および列が XML 文書を含むように指定する SQL INSERT ステートメントを指定します。

XML エクステンダーには、XML 文書を保管するための 2 つの方式が備わっています。それらは、デフォルト・キャスト関数と保管 UDF です。 表8 は、それぞれの方式をいつ使用するかを示しています。

表 8. XML エクステンダーの保管関数

基本タイプ	次のものとして DB2 に保管する		
	XMLVARCHAR	**XMLCLOB**	**XMLFILE**
VARCHAR	XMLVARCHAR()	N/A	XMLFileFromVarchar()
CLOB	N/A	XMLCLOB()	XMLFileFromCLOB()
FILE	XMLVarcharFromFile()	XMLCLOBFromFile()	XMLFILE

デフォルト・キャスト関数を使用する

UDT ごとに、SQL 基本タイプを UDT にキャストするためのデフォルト・キャスト関数 が存在します。 XML エクステンダーに備わっているキャスト関数を VALUES 文節内で使用して、データを挿入することができます。 表9 は、備わっているキャスト関数を示しています。

表 9. XML エクステンダーのデフォルト・キャスト関数

SELECT 文節内で使用されるキャスト	戻りタイプ	説明
XMLVARCHAR(VARCHAR)	XMLVARCHAR	VARCHAR のメモリー・バッファーからの入力
XMLCLOB(CLOB)	XMLCLOB	CLOB または CLOB ロケーターのメモリー・バッファーからの入力
XMLFILE(VARCHAR)	XMLFILE	ファイル名だけを保管する

例: 以下のステートメントは、キャストされた VARCHAR タイプを XMLVARCHAR タイプに挿入します。

```
INSERT INTO sales_tab
VALUES('123456', 'Sriram Srinivasan', db2xml.XMLVarchar(:xml_buff))
```

保管 UDF を使用する

XML エクステンダー UDT ごとに、基本タイプ以外のリソースから DB2 にデータをインポートするために保管 UDF が存在します。たとえば、XML ファイル文書を DB2 に XMLCLOB としてインポートしたい場合、関数 XMLCLOBFromFile() を使用できます。

134ページの表10 は、XML エクステンダーに備わっている保管関数を示しています。

表 10. XML エクステンダーの保管 UDF

保管ユーザー定義関数	戻りタイプ	説明
XMLVarcharFromFile()	XMLVARCHAR	XML 文書をサーバー上のファイルから読み取り、XMLVARCHAR タイプの値を戻します。
XMLCLOBFromFile()	XMLCLOB	XML 文書をサーバー上のファイルから読み取り、XMLCLOB タイプの値を戻します。
XMLFileFromVarchar()	XMLFILE	XML 文書をメモリーから VARCHAR として読み取り、それを外部ファイルに書き込んで、 XMLFILE タイプの値 (ファイル名) を戻します。
XMLFileFromCLOB()	XMLFILE	XML 文書をメモリーから CLOB または CLOB ロケーターとして読み取り、それを外部ファイルに書き込んで、XMLFILE タイプの値 (ファイル名) を戻します。

例: 以下のステートメントは、 XMLCLOBFromFile() 関数を XMLCLOB として使用して、レコードを XML 表に保管します。

```
EXEC SQL INSERT INTO sales_tab(ID, NAME, ORDER)
    VALUES( '1234', 'Sriram Srinivasan,
      XMLCLOBFromFile('c:¥dxx¥samples¥cmd¥getstart.xml'))
```

上記の例では、 XML オブジェクトが `c:¥dxx¥samples¥cmd¥getstart.xml` という名前のファイルから、表 SALES_TAB 内の列 ORDER にインポートされます。

データを取り出す

XML エクステンダーを使用して、文書全体、またはエレメントおよび属性の内容を取り出すことができます。 XML 列を直接取り出すと、XML エクステンダーは UDT を列タイプとして戻します。 データの取り出しの詳細については、以下の節を参照してください。

- 『文書全体を取り出す』
- 137ページの『エレメント内容および属性値を取り出す』

XML エクステンダーには、データを取り出すための 2 つの方式が備わっています。それらは、デフォルト・キャスト関数と Content() 多重定義 UDF です。 表11 は、それぞれの方式をいつ使用するかを示しています。

表 11. XML エクステンダーの検索関数

XML タイプ	次のものとして DB2 から取り出す		
	VARCHAR	CLOB	FILE
XMLVARCHAR	VARCHAR	N/A	Content()
XMLCLOB	N/A	XMLCLOB	Content()
XMLFILE	N/A	Content()	FILE

文書全体を取り出す

タスクの概説:

1. XML 文書を XML 表に保管したことを確認して、取り出したいデータを決めます。
2. DB2 表内のデータを取り出すための方式 (キャスト関数または UDF) を選択します。
3. 多重定義 Content() を使用する場合、UDF は取り出すデータに関連したデータ・タイプ、およびエクスポートされるデータ・タイプを判別することになります。
4. XML 文書の検索元となる XML 表および列を指定する SQL 照会を指定します。

XML エクステンダーには、データを取り出すための 2 つの方法が備わっています。

デフォルト・キャスト関数を使用する

UDT が XML UDT を SQL 基本タイプに変換して、その後それを処理するための、 DB2 に備わっているデフォルト・キャスト関数を使用します。 XML エクステンダーに備わっているキャスト関数を SELECT ステートメント内で使用して、データを取り出すことができます。 136ページの表12 は、備わっているキャスト関数を示しています。

表 12. XML エクステンダーのデフォルト・キャスト関数

SELECT 文節内で使用されるキャスト	戻りタイプ	説明
varchar(XMLVARCHAR)	VARCHAR	VARCHAR による XML 文書
clob(XMLCLOB)	CLOB	CLOB による XML 文書
varchar(XMLFile)	VARCHAR	VARCHAR による XML ファイル名

例: 以下の例では、XMLVARCHAR を取り出し、 VARCHAR データ・タイプとしてメモリーに保管します。

```
EXEC SQL SELECT db2xml.varchar(order) from sales_tab
```

Content() 多重定義 UDF を使用する

Content() UDF を使用して、外部記憶装置から文書内容を取り出してメモリーに入れるか、または内部記憶装置から文書をエクスポートして DB2 サーバー上の*外部ファイル* に入れます。

たとえば、XML 文書を XMLFILE として保管していて、メモリー内でそれを処理したい場合、Content() UDF を使用することができます。それは XMLFILE データ・タイプを入力として、CLOB を戻します。

Content() UDF は、指定されたデータ・タイプに応じて 2 つの異なる検索機能を実行します。それは次のとおりです。

外部記憶装置から文書を取り出して、メモリーに入れる

文書が外部ファイルとして保管されている場合、Content() を使用して XML 文書を取り出し、それをメモリー・バッファーまたは CLOB ロケーターに入れることができます。以下の関数構文を使用します。ここで、*xmlobj* は照会されている XML 列です。

XMLFILE から CLOB へ: ファイルからデータを取り出して、 CLOB ロケーターにエクスポートします。

```
Content(xmlobj XMLFile)
```

内部記憶装置から文書を取り出して、それを外部ファイルにエクスポートする さらに、Content() を使用して、DB2 内に保管された XML 文書を取り出し、データベース・サーバーのファイル・システム上にあるファイルにエクスポートすることができます。 それは、VARCHAR タイプのファイルの名前を戻します。 以下の関数構文を使用します。ここで、*xmlobj* は照会されている

XML 列で、*filename* は外部ファイルです。 *XML タイプ* は、XMLVARCHAR または XMLCLOB データ・タイプとすることができます。

XML タイプ **から外部ファイルへ:** XML データ・タイプとして保管されている XML の内容を取り出して、外部ファイルにエクスポートします。

```
Content(xmlobj XML type, filename varchar(512))
```

ここで、

xmlobj XML 内容が取り出される XML 列の名前です。 *xmlobj* はタイプ XMLVARCHAR または XMLCLOB です。

filename
XML データを保管するファイルの名前です。

下記の例では、*組み込み SQL* のある小さな C プログラムの一部によって、どのようにして XML 文書がファイルから取り出されてメモリーに入れられるかを示しています。この例では、列 BOOK は XMLFILE タイプであると想定します。

```
EXEC SQL BEGIN DECLARE SECTION;
             SQL TYPE IS CLOB_LOCATOR xml_buff;
          EXEC SQL END DECLARE SECTION;
          EXEC SQL CONNECT TO SALES_DB
          EXEC SQL DECLARE c1 CURSOR FOR
              SELECT Content(order) from sales_tab
                         EXEC SQL OPEN c1;
          do {
             EXEC SQL FETCH c1 INTO :xml_buff;
             if (SQLCODE != 0) {
                break;
                }
             else {
               /* do whatever you need to do with the XML doc in buffer */
               }
             }
          EXEC SQL CLOSE c1;
          EXEC SQL CONNECT RESET;
```

エレメント内容および属性値を取り出す

1 つ以上の XML 文書から、*エレメント* または*属性* 値の内容を取り出し (抽出) することができます (単一文書または集合文書検索)。 XML エクステンダーには、SQL データ・タイプごとに SQL SELECT 文節内で指定できるユーザー定義の抽出関数が備わっています。

XML データにリレーショナル・データとしてアクセスできるので、エレメントと属性の内容および値の取り出しは、アプリケーションの開発に役立ちま

す。たとえば、表 SALES_TAB 内の列 ORDER に 1000 の XML 文書が保管されていると仮定します。 SELECT 文節に抽出のための UDF を指定して情報を取り出す次の SQL ステートメントを使用して、品目を注文したカスタマーすべての名前を取り出すことができます。

```
SELECT extractVarchar(Order, '/Order/Customer/Name') from sales_order_view
        WHERE price > 2500.00
```

この例では、抽出 UDF はエレメント <customer> を列 ORDER から VARCHAR データ・タイプとして取り出します。ロケーション・パスは /Order/Customer/Name です (ロケーション・パス構文については、58ページの『ロケーション・パス』を参照)。さらに、サブエレメント <ExtendedPrice> の値が 2500.00 を超える <customer> エレメントの内容だけを指定する WHERE 文節を使用することにより、戻り値の数が減少します。

エレメントの内容または属性値を抽出する: 以下の構文を表関数またはスカラー*関数* として使用することにより、 139ページの表13 にリストされている抽出 UDF を使用します。

extract*retrieved_datatype*(*xmlobj*, *path*)

ここで、

retrieved_datatype
抽出関数から戻されるデータ・タイプ。以下のタイプのいずれかです。

- INTEGER
- SMALLINT
- DOUBLE
- REAL
- CHAR
- VARCHAR
- CLOB
- DATE
- TIME
- TIMESTAMP
- FILE

xmlobj エレメントまたは属性が抽出される XML 列の名前です。この列は、以下の XML ユーザー定義タイプのいずれかとして定義しなければなりません。

- XMLVARCHAR

- XMLCLOB を LOCATOR として
- XMLFILE

path XML 文書内のエレメントまたは属性のロケーション・パス (/Order/Customer/Name など)。ロケーション・パス構文については、58ページの『ロケーション・パス』を参照してください。

重要: UDF の抽出は、属性がある述部を持つロケーション・パスはサポートしますが、エレメントがあるものはサポートしないことに注意してください。たとえば、以下の述部はサポートされます。

```
'/Order/Part[@color="black "]/ExtendedPrice'
```

以下の述部はサポートされません。

```
'/Order/Part/Shipment/[Shipdate < "11/25/00"]'
```

表13 はスカラーと表の両方の形式による抽出関数を示しています。

表 13. XML エクステンダーの抽出関数

スカラー関数	表関数	戻される列名 (表関数)	戻りタイプ
extractInteger()	extractIntegers()	returnedInteger	INTEGER
extractSmallint()	extractSmallints()	returnedSmallint	SMALLINT
extractDouble()	extractDoubles()	returnedDouble	DOUBLE
extractReal()	extractReals()	returnedReal	REAL
extractChar()	extractChars()	returnedChar	CHAR
extractVarchar()	extractVarchars()	returnedVarchar	VARCHAR
extractCLOB()	extractCLOBs()	returnedCLOB	CLOB
extractDate()	extractDates()	returnedDate	DATE
extractTime()	extractTimes()	returnedTime	TIME
extractTimestamp()	extractTimestamps()	returnedTimestamp	TIMESTAMP

スカラー関数の例:

以下の例では、キーの属性値が ″1″ のときに 1 つの値が戻されます。この値は、自動的に DECIMAL タイプに変換される整数として抽出されます。

```
CREATE TABLE t1(key decimal(3,2));
INSERT into t1 values
SELECT * from table(db2xml.extractInteger(db2xml.XMLFile
   ('c:¥dxx¥samples¥xml¥getstart.xml'), '/Order/[@key="1"]'));
SELECT * from t1;
```

表関数の例:

以下の例では、販売注文のそれぞれのキー値が INTEGER として抽出されます。

```
SELECT * from table(db2xml.extractIntegers(db2xml.XMLFile
   ('c:¥dxx¥samples¥xml¥getstart.xml'), '/Order/@key')) as x;
```

XML データを更新する

XML エクステンダーによって、XML 列データを置換して XML 文書全体を更新すること、または指定したエレメントまたは属性の値を更新することができます。

タスクの概説:

1. XML 文書を XML 表に保管したことを確認して、取り出したいデータを決めます。
2. DB2 表内のデータを更新するための方式 (キャスト関数または UDF) を選択します。
3. 更新する XML 表および列を指定する SQL 照会を指定します。

重要: XML について使用可能にされた列を更新するとき、 XML エクステンダーは自動的にサイド表を更新して、変更が反映されるようにします。しかし、対応する XML エレメントまたは属性値を変更することにより、 XML 列に保管された元の XML 文書を更新するのではなく、これらの表を直接更新するということはしないでください。 そのような更新は、データ不整合の問題を生じさせることがあります。

XML 文書を更新するには、以下のように行います。

以下の方式のいずれかを使用します。

デフォルト・キャスト関数を使用する

ユーザー定義タイプ (UDT) ごとに、 SQL 基本タイプを UDT にキャストするためのデフォルト・キャスト関数が存在します。 XML エクステンダーに備わっているキャスト関数を使用して、XML 文書を更新することができます。 133ページの表9 は備わっているキャスト関数を示しており、列 ORDER が XML エクステンダーに備わっている各種の UDT から作成されたと想定しています。

例: キャストされた VARCHAR タイプから XMLVARCHAR タイプを更新します。 xml_buf は VARCHAR タイプとして定義されたホスト変数であると想定します。

```
UPDATE sales_tab VALUES('123456', 'Sriram Srinivasan',
  db2xml.XMLVarchar(:xml_buff))
```

保管 UDF を使用する

XML エクステンダー UDT ごとに、基本タイプ以外のリソースから DB2 にデータをインポートするために保管 UDF が存在します。保管 UDF を使用して、XML 文書全体を置換することにより、それを更新することができます。

例: 以下の例では、 XMLVarcharFromFile() 関数を使用して XML 文書を更新します。

```
UPDATE sales_tab
       set order = XMLVarcharFromFile('c:¥dxx¥samples¥cmd¥getstart.xml')
     WHERE sales_person = 'Sriram Srinivasan'
```

上記の例では、 XML オブジェクトが c:¥dxx¥samples¥cmd¥getstart.xml という名前のファイルから表 SALES_TAB 内の列 ORDER に更新されます。

XML エクステンダーに備わっている保管関数のリストについては、134ページの表10 を参照してください。

XML 文書の特定のエレメントおよび属性を更新する方法:

Update() UDF を使用して、文書全体を更新する代わりに、一度に 1 文書の 1 つの値だけを更新します。 UDF を使用して、ロケーション・パス、およびそのロケーション・パスが示す、置換するエレメントまたは属性の値を指定します。 (ロケーション・パス構文については、58ページの『ロケーション・パス』を参照してください。) XML 文書を編集する必要はありません。XML エクステンダーがユーザーに代わって変更を行います。

Update UDF は、XML ファイル全体を更新し、 XML 構文解析プログラムからの情報に基づいてファイルを再構築します。 Update UDF が文書を処理する方法と、更新前と更新後の文書の例を調べたい場合は、 219ページの『Update 関数が XML 文書を処理する方法』を参照してください。

構文:

```
Update(xmlobj, path, value)
```

ここで、

xmlobj エレメントまたは属性の値が更新される XML 列の名前です。

path 更新されるエレメントまたは属性のロケーション・パスです。ロケーション・パス構文については、

58ページの『ロケーション・パス』を参照してください。複数オカレンスの考慮事項について調べるには、221ページの『複数オカレンス』を参照してください。

value 更新される値です。

例: 以下のステートメントは、Update() UDF を使用して、 <Customer> エレメントの値を文字ストリング IBM に更新します。

```
UPDATE sales_tab
       set order = Update(order, '/Order/Customer/Name', 'IBM')
   WHERE sales_person = 'Sriram Srinivasan'
```

複数オカレンス:

Update() UDF にロケーション・パスが備えられている場合、一致するパスがあるすべてのエレメントまたは属性の内容は、提供された値で更新されます。これは、文書に複数出現のロケーション・パスがある場合、 Update 関数は、既存の値を *value* パラメーターで提供された値で置換するということです。

XML 文書を検索する

XML データの検索 (サーチ) は、XML データの取り出し (リトリーブ) と似ています。どちらの技法もデータを取得して処理できるようにしますが、検索では WHERE 文節を使用して述部を検索基準として定義します。

XML エクステンダーは、ご使用のアプリケーションでの必要に応じて、 XML 列内で XML 文書を検索するための数種類の方式を提供します。 それには、エレメントの内容および属性値に基づいて文書構造を検索し、結果を戻す機能が備わっています。 XML 列およびそのサイド表のビューを検索すること、パフォーマンス向上のためにサイド表を直接検索すること、または抽出 UDF に WHERE 文節を指定して使用することができます。さらに、DB2 テキスト・エクステンダーを使用して、構造内容に含まれている検索列データでテキスト・ストリングを検索することもできます。

XML エクステンダーによって、XML 文書から抽出された XML エレメントの内容または属性値を含むサイド表列の索引を使用して、高速検索を行うことができます。エレメントまたは属性のデータ・タイプを指定することにより、SQL 汎用データ・タイプを検索したり、範囲検索を行うことができます。たとえば、この注文の例で、合計価格が 2500.00 を超える注文すべてを検索できます。

さらに、DB2 UDB テキスト・エクステンダーを使用して構造化テキスト検索または全テキスト検索を行うことができます。たとえば、XML 形式の履歴書

を含む列 RESUME があるとします。ここで、Java のスキルのある応募者すべての名前を知りたいと仮定します。その場合、DB2 テキスト・エクステンダーを使用して XML 文書を検索し、 <skill> エレメントに文字ストリング JAVA が含まれるすべての履歴書を見つけることができます。

以下の節では、次の検索方式について解説します。

- 『構造に基づいて XML 文書を検索する』
- 145ページの『テキスト・エクステンダーを使用して構造化テキスト検索を行う』

構造に基づいて XML 文書を検索する

XML エクステンダーの検索機能を使用して、文書構造つまりエレメントと属性に基づいて列内の XML データを検索することができます。列データを検索するには、いくつかの方法で SELECT ステートメントを使用して、文書のエレメントおよび属性との一致に基づく*結果セット* を戻します。以下の方式を使用して、列データを検索することができます。

- サイド表上での直接照会による検索
- *結合ビュー* からの検索
- 抽出 UDF による検索
- 複数オカレンスのあるエレメントまたは属性の検索

これらの方式については以降の節で解説され、次のシナリオに基づく例が使用されます。 アプリケーション表 SALES_TAB には ORDER という名前の XML 列があります。 この列には 3 つのサイド表、ORDER_SIDE_TAB、PART_SIDE_TAB、および SHIP_SIDE_TAB があります。 デフォルト・ビューの sales_order_view は、ORDER 列が使用可能にされたときに指定され、以下の CREATE VIEW ステートメントを使用してこれらの表を結合します。

```
CREATE VIEW sales_order_view(invoice_num, sales_person, order,
                             order_key, customer, part_key, price, date)
     AS
     SELECT sales_tab.invoice_num, sales_tab.sales_person, sales_tab.order,
            order_side_tab.order_key, order_side_tab.customer,
            part_side_tab.part_key, ship_side_tab.date
     FROM sales_tab, order_side_tab, part_side_tab, ship_side_tab
     WHERE sales_tab.invoice_num = order_side_tab.invoice_num
                AND sales_tab.invoice_num = part_side_tab.invoice_num
                AND sales_tab.invoice_num = ship_side_tab.invoice_num
```

サイド表上での直接照会による検索

副照会検索を伴う直接照会により、サイド表が索引付けされている場合の構造化検索において、最高のパフォーマンスを得ることができます。照会または副照会を使用して、サイド表を適切に検索できます。

例: 以下のステートメントは照会または副照会を使用して、サイド表を直接検索します。

```
SELECT sales_person from sales_tab
   WHERE invoice_num in
     (SELECT invoice_num from part_side_tab
      WHERE price > 2500.00)
```

この例で、invoice_num は SALES_TAB 表の基本キーです。

結合ビューからの検索

XML エクステンダーが固有の ID を使用して、アプリケーション表とサイド表とを結合するデフォルトのビューを作成するようにすることができます。このデフォルト・ビューか、アプリケーション表とサイド表とを結合する任意のビューを使用して、列データの検索とサイド表の照会を行うことができます。この方式は、アプリケーション表とそのサイド表のための、単一の仮想ビューを提供します。しかし、作成されるサイド表の数が多い場合、照会の負荷は大きくなります。

ヒント: *root_id* または DXXROOT_ID (XML エクステンダーによって作成される) を使用して、独自のビューを作成するときに複数の表を結合することができます。

例: 以下のステートメントはビューを検索します。

```
SELECT sales_person from sales_order_view
   WHERE price > 2500.00
```

この SQL ステートメントは、在庫品目の注文価格が 2500.00 を超える sales_person の値を、結合ビュー sales_order_view 表から戻します。

抽出 UDF による検索

さらに、アプリケーション表に索引またはサイド表を作成していないとき、XML エクステンダーの抽出 UDF を使用してエレメントおよび属性を検索することもできます。抽出 UDF を使用して XML データを走査することは負荷がとても大きいので、検索に含められる XML 文書の数を制限する WHERE 文節を必ず使用してください。

例: 以下のステートメントは、抽出 XML エクステンダー UDF を使用して検索を行います。

```
SELECT sales_person from sales_tab
     WHERE extractVarchar(order, '/Order/Customer/Name')
    like '%IBM%'
AND invoice_num > 100
```

この例では、抽出 UDF は 値が IBM である </Order/Customer/Name> エレメントを抽出します。

複数オカレンスのあるエレメントまたは属性の検索

複数オカレンスのあるエレメントまたは属性を検索するときは、 DISTINCT 文節を使用して値の重複を回避してください。

例: 以下のステートメントは、DISTINCT 文節を使用して検索を行います。

```
SELECT sales_person from sales_tab
            WHERE invoice_num in
              (SELECT DISTINCT invoice_num from part_side_tab
               WHERE price > 2500.00 )
```

この例では、DAD ファイルにより /Order/Part/Price に複数オカレンスがあることが指定され、そのためのサイド表 PART_SIDE_TAB が作成されます。 PART_SIDE_TAB 表には、同一の invoice_num を持つ行が複数存在する可能性があります。 DISTINCT を使用すると、固有の値だけが戻されます。

テキスト・エクステンダーを使用して構造化テキスト検索を行う

XML 文書構造を検索しているとき、 XML エクステンダーは汎用データ・タイプに変換されたエレメントおよび属性値を検索しますが、テキストは検索しません。 DB2 UDB テキスト・エクステンダーを使用して、XML について使用可能にされた列上で、構造化または全テキスト検索を行うことができます。 テキスト・エクステンダーは、DB2 UDB バージョン 6.1 以降で XML 文書検索をサポートします。 テキスト・エクステンダーは、Windows オペレーティング・システム、AIX、および Sun Solaris 上で使用可能です。

構造化テキスト検索

XML 文書のツリー構造に基づいてテキスト・ストリングを検索します。 たとえば、文書構造 /Order/Customer/Name があり、文字ストリング 『IBM』 を <Customer> サブエレメント内で検索したい場合、構造化テキスト検索を使用できます。 その文書にはさらに、ストリング IBM が <Comment> サブエレメント内に、または製品の一部の名前として使用されている可能性があります。 構造化テキスト検索は、指定したエレメント内だけでストリングを検索します。 この例では、</Order/Customer/Name> サブエレメント内に IBM がある文書だけが見つかります。他のエレメント内に IBM があっても </Order/Customer/Name> サブエレメントにはない文書は戻されません。

全テキスト検索

エレメントまたは属性に関係なく、文書全体でテキスト・ストリングを

検索します。 前述の例を使用すると、文字ストリング IBM がどこに出現するかには関係なく、ストリング IBM のあるすべての文書が戻されます。

テキスト・エクステンダー検索を使用するには、DB2 テキスト・エクステンダーをインストールして、データベースおよび表を以下に示すように使用可能にしなければなりません。テキスト・エクステンダー検索の使用方法を学ぶには、 *DB2 ユニバーサル・データベース テキスト・エクステンダー 管理およびプログラミング* でテキスト・エクステンダーの UDF による検索に関する章を参照してください。

テキスト・エクステンダーについて XML 列を使用可能にする

XML について使用可能にされているデータベースがあると仮定して、以下のステップを使用して、 XML について使用可能にされた列の内容を検索するためにテキスト・エクステンダーを使用可能にします。例を示すために、データベースの名前が SALES_DB、表の名前が ORDER、そして XML 列の名前が XVARCHAR および XCLOB であるとします。

1. エクステンダー CD の `install.txt` ファイルを参照して、テキスト・エクステンダーのインストール方法を学んでください。
2. 以下の場所のいずれかから、 **txstart** コマンドを入力します。
   - UNIX オペレーティング・システムでは、インスタンス所有者のコマンド・プロンプトからコマンドを入力します。
   - Windows NT では、 DB2INSTANCE が指定されたコマンド・ウィンドウからコマンドを入力します。
3. テキスト・エクステンダーコマンド行ウィンドウをオープンします。このステップでは、SALES_DB という名前のデータベース、および ORDER という名前の表があり、それには XVARCHAR および XCLOB という名前の 2 つの XML 列があると仮定します。 `dxx¥samples¥c` 内のサンプル・プログラムを実行しなければならない場合があります。
4. データベースに接続します。 **db2tx** コマンド・プロンプトから、以下のようにタイプします。

   ```
   'connect to SALES_DB'
   ```

5. テキスト・エクステンダーに関してデータベースを使用可能にします。

   db2tx コマンド・プロンプトから、以下のようにタイプします。

   ```
   'enable database'
   ```

6. テキスト・エクステンダーに関して XML 表内の列を使用可能にして、XML 文書のデータ・タイプ、言語、コード・ページ、および列に関するその他の情報を定義します。

- VARCHAR 列 XVARCHAR については、次のようにタイプします。

```
'enable text column order xvarchar function db2xml.varchartovarchar handle
varcharhandle ccsid 850 language us_english format xml indextype precise
    indexproperty sections_enabled
documentmodel (Order) updateindex update'
```

- CLOB 列 XCLOB については、次のようにタイプします。

```
'enable text column order xclob function db2xml.clob handle clobhandle
 ccsid 850 language us_english indextype precise updateindex update'
```

7. 索引の状況を検査します。
   - 列 XVARCHAR については、次のようにタイプします: `get index status order handle varcharhandle`
   - 列 XCLOB については、次のようにタイプします: `get index status order handle clobhandle`
8. XML 文書モデルを、 `desmodel.ini` と呼ばれる文書モデル INI ファイルで定義します。このファイルは、UNIX では `/db2tx/txins000`、 Windows NT では `¥instance¥¥db2tx¥txins000`、そして初期設定ファイル内のセクションにあります。たとえば、`textmodel.ini` の場合、以下のようになります。

```
;list of document models
[MODELS]
modelname=Order

; an 'Order' document model definition
; left side = section name identifier
; right side = section name tag

[Order]
Order = /Order
Order/Customer/Name = /Order/Customer/Name
Order/Customer/Email = /Order/Customer/Email
Order/Part/@color = /Order/Part/@color
Order/Part/Shipment/ShipMode = /Order/Part/Shipment/ShipMode
```

テキスト・エクステンダーを使用してテキストを検索する

テキスト・エクステンダーの検索機能は、XML エクステンダーの文書の構造化検索と一緒に使用できます。推奨される方式は、文書エレメントまたは属性を検索する照会を作成して、テキスト・エクステンダーを使用してエレメント内容または属性値を検索することです。

例: 以下のステートメントは、テキスト・エクステンダーを使用して XML 文書テキストを検索します。 DB2 コマンド・ウィンドウから、以下のようにタイプします。

```
'connect to SALES_DB'
'select xvarchar from order where db2tx.contains(varcharhandle,
 'model Order section(Order/Customer/Name) "Motors"')=1'
'select xclob from order where db2tx.contains(clobhandle,
 'model Order section(Order/Customer/Name) "Motors"')=1'
```

テキスト・エクステンダー Contains() UDF は検索を実行します。

この例には、テキスト・エクステンダーを使用して列データを検索するために必要なすべてのステップは含まれていません。テキスト・エクステンダー検索の概念および機能を学ぶには、 *DB2 ユニバーサル・データベース テキスト・エクステンダー 管理およびプログラミング* でテキスト・エクステンダーの UDF による検索に関する章を参照してください。

XML 文書を削除する

SQL DELETE ステートメントを使用して、XML 列から XML 文書を削除します。 WHERE 文節を指定して、削除する文書を厳選することができます。

例: 以下のステートメントは、 <ExtendedPrice> の値が 2500.00 を超えるすべての文書を削除します。

```
DELETE from sales_tab
       WHERE invoice_num in
          (SELECT invoice_num from part_side_tab
           WHERE price > 2500.00)
```

JDBC から関数を呼び出すときの制限

関数でパラメーター・マーカーを使用する場合、JDBC 制限の要件として、関数用のパラメーター・マーカーを、戻されるデータが挿入される列のデータ・タイプにキャストすることが必要です。関数選択論理では、引き数の変換後のデータ・タイプが分からないので、参照を解決することができません。

結果として、JDBC は以下のコードを解決できません。

```
db2xml.XML_default_casting_function(length)
```

CAST 指定を使用して、VARCHAR などのパラメーター・マーカー用のタイプを提供すると、関数選択論理は以下のように先に進むことができます。

```
db2xml.XML_default_casting_function(CAST(? AS cast_type(length))
```

例 1: 以下の例で、パラメーター・マーカーは VARCHAR としてキャストされます。渡されるパラメーターは XML 文書であり、これは VARCHAR(1000) としてキャストされ、列 ORDER に挿入されます。

```
String query = "insert into sales_tab(invoice_num, sales_person, order) values
    (?,?,db2xml.XMLVarchar(cast (? as varchar(1000))))";
```

例 2: 以下の例で、パラメーター・マーカーは VARCHAR としてキャストされます。渡されるパラメーターはファイル名であり、その内容は VARCHAR に変換され、列 ORDER に挿入されます。

```
String query = "insert into sales_tab(invoice_num, sales_person, order) values
    (?,?,db2xml.XMLVarcharfromFILE(cast (? as varchar(1000))))";
```

第6章 XML コレクション・データの管理

XML コレクションは、XML 文書にマップされるデータを含むリレーショナル表のセットです。アクセスおよび保管のためのこの方式によって、既存のデータから XML 文書を合成したり、 XML 文書を分解したり、XML を交換の方式として使用することができます。

リレーショナル表は、XML 文書を分解するときに XML エクステンダーが生成する新しい表、またはアプリケーション用の XML 文書を生成するために XML エクステンダーと共に使用するデータを含む既存の表であることがあります。 これらの表にある列データには、XML タグが含まれません。それにはエレメントおよび属性にそれぞれ関連した内容および値が含まれます。 ストアード・プロシージャーは、XML コレクション・データを保管、取り出し、更新、検索、および削除するための、アクセスおよび保管の方式として機能します。

XML コレクションのストアード・プロシージャーによって使用されるパラメーター制限は、311ページの『付録D. XML エクステンダーの制限』で説明しています。

ストアード・プロシージャー結果用の CLOB のサイズは、引き上げることができます。これについては 227ページの『CLOB 制限の引き上げ』で説明しています。

XML コレクションを管理するための情報については、以下の節を参照してください。

- 『XML 文書を DB2 データから合成する』
- 160ページの『XML 文書を分解して DB2 データにする』

XML 文書を DB2 データから合成する

合成とは、XML コレクション内のリレーショナル・データから XML 文書のセットを生成することです。ストアード・プロシージャーを使用して XML 文書を合成できます。これらのストアード・プロシージャーを使用するためには、 XML 文書と DB2 表構造との間のマッピングを指定する DAD ファイルを作成しなければなりません。ストアード・プロシージャーは DAD ファイル

を使用して、XML 文書を合成します。 DAD ファイルの作成方法については、61ページの『XML コレクションについて計画する』を参照してください。

作業を始める前に

- XML 文書の構造をエレメントおよび属性値の内容を含むリレーショナル表にマップします。
- マッピング方式を選択します: SQL マッピングまたは RDB_node マッピング。
- DAD ファイルを準備します。完全な詳細については、61ページの『XML コレクションについて計画する』を参照してください。
- オプションとして、XML コレクションを使用可能にします。

XML 文書を合成する

XML エクステンダーには XML 文書を合成するための 2 つのストアード・プロシージャー、 dxxGenXML() および dxxRetrieveXML() が備わっています。

dxxGenXML()

このストアード・プロシージャーは、頻繁に更新されるアプリケーション、または XML を管理するためのオーバーヘッドを回避したいアプリケーションのために使用されます。ストアード・プロシージャー dxxGenXML() では、使用可能にされたコレクションは必要ありません。代わりにそれは DAD ファイルを使用します。

ストアード・プロシージャー dxxGenXML() は、 DAD ファイル内の <Xcollection> エレメントによって指定された XML コレクション表に保管されているデータを使用して、 XML 文書を構築します。このストアード・プロシージャーは、各 XML 文書を *結果表* 内に行として挿入します。また、結果表でカーソルを開いて結果セットを取り出すこともできます。結果表はアプリケーションによって作成され、VARCHAR、 CLOB、 XMLVARCHAR、または XMLCLOB タイプの列が必ず 1 つ必要です。

さらに、DAD ファイル内で妥当性検査エレメントを YES に指定した場合、 XML エクステンダーは INTEGER タイプの列 DXX_VALID を結果表に追加して、有効な XML 文書には値 1、無効な文書には 0 を挿入します。

さらに、ストアード・プロシージャー dxxGenXML() によって、結果表に生成する行の最大数を指定できます。これにより、処理時間が短縮

されます。ストアード・プロシージャーは表にある実際の行数を、戻りコードおよびメッセージと共に戻します。

分解のための対応するストアード・プロシージャーは dxxShredXML() です。これはさらに、DAD を入力パラメーターとして取り、 XML コレクションが使用可能になっていることを必要としません。

XML コレクションを合成する: dxxGenXML()

以下のストアード・プロシージャー宣言を使用して、ストアード・プロシージャー呼び出しをアプリケーションに組み込みます。

```
dxxGenXML(CLOB(100K)     DAD,             /* input */
          char(resultTabName) resultTabName,   /* input */
          integer        overrideType,    /* input */
          varchar(1024)  override,        /* input */
          integer        maxRows,         /* input */
          integer        numRows,         /* output */
          long           returnCode,      /* output */
          varchar(1024)  returnMsg)       /* output */
```

完全な構文および例については、237ページの『dxxGenXML()』 を参照してください。

例: 以下の例では、XML 文書が合成されます。

```
#include "dxx.h"
#include "dxxrc.h"

EXEC SQL INCLUDE SQLCA;
EXEC SQL BEGIN DECLARE SECTION;
     SQL TYPE is CLOB(100K) dad;              /* DAD */
     SQL TYPE is CLOB_FILE  dadFile;          /* dad file */
     char                   result_tab[32];   /* name of the result table */
     char                   override[2];      /* override, will set to NULL*/
     short                  overrideType;     /* defined in dxx.h */
     short                  max_row;          /* maximum number of rows */
     short                  num_row;          /* actual number of rows */
     long                   returnCode;       /* return error code */
     char                   returnMsg[1024];  /* error message text */
     short                  dad_ind;
     short                  rtab_ind;
     short                  ovtype_ind;
     short                  ov_inde;
     short                  maxrow_ind;
     short                  numrow_ind;
     short                  returnCode_ind;
     short                  returnMsg_ind;

     EXEC SQL END DECLARE SECTION;

     /* create table */
     EXEC CREATE TABLE xml_order_tab (xmlorder XMLVarchar);

     /* read data from a file to a CLOB */
     strcpy(dadfile.name,"c:¥dxx¥samples¥dad¥getstart_xcollection.dad");
     dadfile.name_length = strlen("c:¥dxx¥samples¥dad¥getstart_xcollection.dad");
     dadfile.file_options = SQL_FILE_READ;
     EXEC SQL VALUES (:dadfile) INTO :dad;
     strcpy(result_tab,"xml_order_tab");
     override[0] = '¥0';
     overrideType = NO_OVERRIDE;
```

```
max_row = 500;
num_row = 0;
returnCode = 0;
msg_txt[0] = '¥0';
collection_ind = 0;
dad_ind = 0;
rtab_ind = 0;
ov_ind = -1;
ovtype_ind = 0;
maxrow_ind = 0;
numrow_ind = -1;
returnCode_ind = -1;
returnMsg_ind = -1;

/* Call the store procedure */
EXEC SQL CALL dxxGenXML(:dad:dad_ind;
     :result_tab:rtab_ind,
     :overrideType:ovtype_ind,:override:ov_ind,
     :max_row:maxrow_ind,:num_row:numrow_ind,
     :returnCode:returnCode_ind,:returnMsg:returnMsg_ind);
```

DAD ファイルに指定された SQL 照会は 250 の XML 文書を生成するため、ストアード・プロシージャーが呼び出された後の結果表には 250 行が含まれます。

dxxRetrieveXML()

このストアード・プロシージャーは、定期的な更新を行うアプリケーションのために使用されます。同じタスクが繰り返されるので、パフォーマンスの改善が重要になります。 XML コレクションを使用可能にして、ストアード・プロシージャー内でコレクション名を使用すると、パフォーマンスが改善されます。

ストアード・プロシージャー dxxRetrieveXML() は、ストアード・プロシージャー dxxGenXML() と同様の働きをしますが、 DAD ファイルの代わりに XML コレクションの名前を取ります。 XML コレクションが使用可能にされると、DAD ファイルは XML_USAGE 表に保管されます。そのため、XML エクステンダーは DAD ファイルを検索して、それ以降 dxxRetrieveXML() は dxxGenXML() と同じになります。

dxxRetrieveXML() によって、同じ DAD ファイルを合成と分解の両方に使用できます。このストアード・プロシージャーはさらに、分解された XML 文書の検索にも使用できます。

分解のための対応するストアード・プロシージャーは dxxInsertXML() です。これはさらに、使用可能にされた XML コレクションの名前を取ります。

XML コレクションを合成する: dxxRetrieveXML()

以下のストアード・プロシージャー宣言を使用して、ストアード・プロシージャー呼び出しをアプリケーションに組み込みます。

```
dxxRetrieveXML(char(collectionName) collectionName,  /* input */
               char(resultTabName) resultTabName,   /* input */
               integer        overrideType,    /* input */
               varchar(1024)  override,        /* input */
               integer        maxRows,         /* input */
               integer        numRows,         /* output */
               long           returnCode,      /* output */
               varchar(1024)  returnMsg)       /* output */
```

完全な構文および例については、241ページの『dxxRetrieveXML()』 を参照してください。

例: 以下に dxxRetrieveXML() への呼び出しの例を示します。 ここでは、結果表が名前 XML_ORDER_TAB で作成され、 XMLVARCHAR タイプの列が 1 つあると想定しています。

```
#include "dxx.h"
#include "dxxrc.h"

           EXEC SQL INCLUDE SQLCA;
           EXEC SQL BEGIN DECLARE SECTION;
             char    collection[32];   /* dad buffer */
             char    result_tab[32];   /* name of the result table */
             char    override[2];      /* override, will set to NULL*/
             short   overrideType;     /* defined in dxx.h */
             short   max_row;          /* maximum number of rows */
             short   num_row;          /* actual number of rows */
             long    returnCode;       /* return error code */
             char    returnMsg[1024];  /* error message text */
             short   dadbuf_ind;
             short   rtab_ind;
             short   ovtype_ind;
             short   ov_inde;
             short   maxrow_ind;
             short   numrow_ind;
             short   returnCode_ind;
             short   returnMsg_ind;
             EXEC SQL END DECLARE SECTION;

             /* create table */
             EXEC SQL CREATE TABLE xml_order_tab (xmlorder XMLVarchar);

             /* initialize host variable and indicators */
             strcpy(collection,"sales_ord");
             strcpy(result_tab,"xml_order_tab");
             override[0] = '¥0';
             overrideType = NO_OVERRIDE;
             max_row = 500;
             num_row = 0;
             returnCode = 0;
             msg_txt[0] = '¥0';
             collection_ind = 0;
             rtab_ind = 0;
             ov_ind = -1;
             ovtype_ind = 0;
             maxrow_ind = 0;
             numrow_ind = -1;
             returnCode_ind = -1;
             returnMsg_ind = -1;

             /* Call the store procedure */
```

```
EXEC SQL CALL dxxRetrieve(:collection:collection_ind;
       :result_tab:rtab_ind,
       :overrideType:ovtype_ind,:override:ov_ind,
       :max_row:maxrow_ind,:num_row:numrow_ind,
       :returnCode:returnCode_ind,:returnMsg:returnMsg_ind);
```

DAD ファイル内の値を動的にオーバーライドする

動的な照会のためには、2 つの任意指定パラメーター *override* および *overrideType* を使用して、 DAD ファイル内の条件をオーバーライドすることができます。 *overrideType* からの入力に基づいて、アプリケーションは DAD 内にある SQL マッピング用の <SQL_stmt> タグ値または RDB_node マッピング用の RDB_nodes の条件をオーバーライドできます。

これらのパラメーターには、以下の値および規則があります。

overrideType
このパラメーターは必須入力パラメーター (IN) であり、 *override* パラメーターのタイプにフラグを付けます。 *overrideType* には以下の値があります。

NO_OVERRIDE
DAD ファイル内の条件をオーバーライドしないことを指定します。

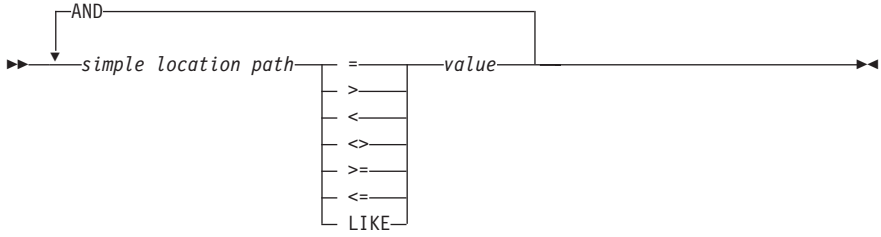
SQL_OVERRIDE
DAD ファイル内の条件を SQL ステートメントによってオーバーライドすることを指定します。

XML_OVERRIDE
DAD ファイル内の条件を XPath に基づく条件によってオーバーライドすることを指定します。

override
このパラメーターは必須入力パラメーター (IN) であり、 DAD ファイルをオーバーライドする条件を指定します。入力値構文は、*overrideType* で指定した値に対応します。

- NO_OVERRIDE を指定した場合、入力値は NULL ストリングです。
- SQL_OVERRIDE を指定した場合、入力値は有効な SQL ステートメントです。***必須:*** SQL_OVERRIDE および SQL ステートメントを使用する場合、 SQL マッピング体系を DAD ファイルで使用しなければなりません。入力 SQL ステートメントは、DAD ファイル内の <SQL_stmt> エレメントで指定された SQL ステートメントをオーバーライドします。
- XML_OVERRIDE を使用する場合、入力値は 1 つ以上の式を含むストリングです。***必須:*** XML_OVERRIDE および式を使用する場合、 XML

マッピング体系を DAD ファイルで使用しなければなりません。入力 XML 式は DAD ファイルで指定された RDB_node 条件をオーバーライドします。その式は、以下の構文を使用します。



ここで、

simple location path
XPATH によって定義された構文を使用する単純ロケーション・パス。構文については、60ページの表5 を参照してください。

演算子
式のその他の部分から演算子を分けるためにスペースを入れることができます。

value
数値または単一引用符で囲まれたストリング。

任意に、演算子をスペースで囲むことができます。 LIKE 演算子をスペースで囲むことは必須です。

XML_OVERRIDE 値を指定した場合、単純ロケーション・パスと一致する text_node または attribute_node 内の RDB_node の条件は、指定した式によってオーバーライドされます。

XML_OVERRIDE は、完全に XPath 準拠ではありません。単純ロケーション・パスは、列にマップされるエレメントまたは属性を識別するためにのみ使用されます。

例:

以下の例は、SQL_OVERRIDE および XML_OVERRIDE を使用する動的なオーバーライドを示しています。本書でのストアード・プロシージャー例のほとんどは、NO_OVERRIDE を使用します。

例: SQL_OVERRIDE を使用するストアード・プロシージャー

```
include "dxx.h"
include "dxxrc.h"

       EXEC SQL INCLUDE SQLCA;
       EXEC SQL BEGIN DECLARE SECTION;
         char    collection[32];    /* dad buffer */
         char    result_tab[32];    /* name of the result table */
         char    override[256];     /* override, SQL_stmt */
         short   overrideType;      /* defined in dxx.h */
         short   max_row;           /* maximum number of rows */
         short   num_row;           /* actual number of rows */
         long    returnCode;        /* return error code */
         char    returnMsg[1024];   /* error message text */
         short   rtab_ind;
         short   ovtype_ind;
         short   ov_inde;
         short   maxrow_ind;
         short   numrow_ind;
         short   returnCode_ind;
         short   returnMsg_ind;

         EXEC SQL END DECLARE SECTION;

         /* create table */
         EXEC CREATE TABLE xml_order_tab (xmlorder XMLVarchar);

         /* initialize host variable and indicators */
         strcpy(collection,"sales_ord");
         strcpy(result_tab,"xml_order_tab");
         sprintf(override,"%s %s %s %s %s %s %s",
                   "SELECT o.order_key, customer, p.part_key, quantity, price,",
                   "tax, ship_id, date, mode ",
                   "FROM order_tab o, part_tab p,",
                   "table(select substr(char(timestamp(generate_unique())),16",
                   "as ship_id,date,mode from ship_tab)as s",
                   "WHERE p.price > 50.00 and s.date >'1998-12-01' AND",
                   "p.order_key = o.order_key and s.part_key = p.part_key");
        overrideType = SQL_OVERRIDE;
         max_row = 500;
         num_row = 0;
         returnCode = 0;
         msg_txt[0] = '¥0';
         collection_ind = 0;
         rtab_ind = 0;
         ov_ind = 0;
         ovtype_ind = 0;
         maxrow_ind = 0;
         numrow_ind = -1;
         returnCode_ind = -1;
         returnMsg_ind = -1;

         /* Call the store procedure */
         EXEC SQL CALL dxxRetrieve(:collection:collection_ind;
                :result_tab:rtab_ind,
```

```
:overrideType:ovtype_ind,:override:ov_ind,
:max_row:maxrow_ind,:num_row:numrow_ind,
:returnCode:returnCode_ind,:returnMsg:returnMsg_ind);
```

この例で、DAD ファイル内の <xcollection> エレメントには <SQL_stmt> エレメントが必要です。 *override* パラメーターは、価格を 50.00 より大に変更し、日付を 1998-12-01 より大に変更して、<SQL_stmt> の値をオーバーライドします。

例: XML_OVERRIDE を使用するストアード・プロシージャー

```
include "dxx.h"
include "dxxrc.h"

       EXEC SQL INCLUDE SQLCA;
       EXEC SQL BEGIN DECLARE SECTION;
         char    collection[32];  /* dad buffer */
         char    result_tab[32];  /* name of the result table */
         char    override[256];   /* override, SQL_stmt */
         short   overrideType;    /* defined in dxx.h */
         short   max_row;         /* maximum number of rows */
         short   num_row;         /* actual number of rows */
         long    returnCode;      /* return error code */
         char    returnMsg[1024]; /* error message text */
         short   dadbuf_ind;
         short   rtab_ind;
         short   ovtype_ind;
         short   ov_inde;
         short   maxrow_ind;
         short   numrow_ind;
         short   returnCode_ind;
         short   returnMsg_ind;

         EXEC SQL END DECLARE SECTION;

         /* create table */
         EXEC CREATE TABLE xml_order_tab (xmlorder XMLVarchar);

         /* initialize host variable and indicators */
         strcpy(collection,"sales_ord");
         strcpy(result_tab,"xml_order_tab");
         sprintf(override,"%s %s",
                   "/Order/Part/Price > 50.00 AND ",
                   "Order/Part/Shipment/ShipDate > '1998-12-01'");
        overrideType = XML_OVERRIDE;
         max_row = 500;
         num_row = 0;
         returnCode = 0;
         msg_txt[0] = '¥0';
         collection_ind = 0;
         rtab_ind = 0;
         ov_ind = 0;
         ovtype_ind = 0;
         maxrow_ind = 0;
```

```
numrow_ind = -1;
returnCode_ind = -1;
returnMsg_ind = -1;

/* Call the store procedure */
EXEC SQL CALL dxxRetrieve(:collection:collection_ind;
       :result_tab:rtab_ind,
       :overrideType:ovtype_ind,:override:ov_ind,
       :max_row:maxrow_ind,:num_row:numrow_ind,
       :returnCode:returnCode_ind,:returnMsg:returnMsg_ind);
```

この例で、DAD ファイル内の <collection> エレメントには、ルートの element_node に対して RDB_node があります。 *override* の値は XML の内容に基づいています。 XML エクステンダーは単純ロケーション・パスをマップ先 DB2 列に変更します。

XML 文書を分解して DB2 データにする

XML 文書の分解とは、XML 文書内のデータを細かくしてリレーショナル表に保管することです。 XML エクステンダーには、 XML データをソースの XML 文書から分解してリレーショナル表に入れるストアード・プロシージャーが備わっています。これらのストアード・プロシージャーを使用するためには、 XML 文書と DB2 表構造との間のマッピングを指定する DAD ファイルを作成しなければなりません。 ストアード・プロシージャーは DAD ファイルを使用して、XML 文書を分解します。 DAD ファイルの作成方法については、61ページの『XML コレクションについて計画する』を参照してください。

分解のために XML コレクションを使用可能にする

たいてい、ストアード・プロシージャーを使用する前に XML コレクションを使用可能にしなければなりません。 以下の場合には、XML コレクションを使用可能にする必要があります。

- XML 文書を分解して新規の表に入れる場合、 XML コレクション内のすべての表はコレクションが使用可能になるときに XML エクステンダーによって作成されるので、 XML コレクションを使用可能にしなければなりません。
- 複数出現するエレメントおよび属性の順序を保持することが重要である場合。 XML エクステンダーが表の複数オカレンスのエレメントまたは属性の順序を保持するのは、それらがコレクションの使用可能化の際に作成された場合だけです。 XML 文書を分解して既存のリレーショナル表に入れる場合、順序が保持される保証はありません。

データベース内にすでに表が存在するとき任意で DAD ファイルを渡すようにしたい場合は、 XML コレクションを使用可能にする必要はありません。

分解する表サイズの制限

分解では RDB_node マッピングを使って、表行へのエレメントと属性値の抽出によって XML 文書を DB2 表に分解する方法を指定します。各 XML 文書の値は、1 つ以上の DB2 表に保管されます。どの表にも、各文書から分解した最大 1024 行までを入れることができます。

たとえば、XML 文書を 5 つの表に分解する場合、その 5 つの表のおのおのに、該当する文書中の 1024 行までを入れることができます。複数の文書用の行をもつ表でも、各文書につき 1024 行までを入れることができます。 20 個の文書をもつ表の場合、各文書につき 1024 行ずつ、20,480 行を入れることができます。

複数出現エレメント (XML 構造内で複数出現する可能性のあるロケーション・パスをもつエレメント) を使うと、行数が影響を受けます。たとえば、20 回出現するエレメント <Part> の入った文書は、表内で 20 行として分解されることがあります。複数出現エレメントを使用する場合、このような表サイズの制限事項に配慮してください。

作業を始める前に

- XML 文書の構造をエレメントおよび属性値の内容を含むリレーショナル表にマップします。
- RDB_node マッピングを使用して DAD ファイルを準備します。詳しくは 61ページの『XML コレクションについて計画する』を参照してください。
- オプションとして、XML コレクションを使用可能にします。

XML 文書の分解

XML エクステンダーには XML 文書を分解するための 2 つのストアード・プロシージャー、 dxxShredXML() および dxxInsertXML が備わっています。

dxxShredXML()

このストアード・プロシージャーは、頻繁に更新されるアプリケーション、または XML を管理するためのオーバーヘッドを回避したいアプリケーションのために使用されます。ストアード・プロシージャー dxxShredXML() では、使用可能にされたコレクションは必要ありません。代わりにそれは DAD ファイルを使用します。

ストアード・プロシージャー dxxShredXML() は、 DAD ファイルおよび分解する XML 文書の 2 つを入力パラメーターとします。それは戻りコードと戻りメッセージの 2 つを出力パラメーターとして戻します。

ストアード・プロシージャー dxxShredXML() は、 DAD ファイル内の <Xcollection> 指定に応じて、 XML 文書を XML コレクションに挿入します。 DAD ファイル内の <Xcollection> で使用される表は存在することが想定され、列は DAD マッピングで指定されるデータ・タイプに適合することが想定されます。それ以外の場合、エラー・メッセージが戻されます。ストアード・プロシージャー dxxShredXML() はその後、XML 文書を分解して、タグのない XML データを DAD ファイル内で指定された表に挿入します。

合成のための対応するストアード・プロシージャーは dxxGenXML() です。これはさらに、DAD を入力パラメーターとして取り、 XML コレクションが使用可能になっていることを必要としません。

XML コレクションを分解する: dxxShredXML()

以下のストアード・プロシージャー宣言を使用して、ストアード・プロシージャー呼び出しをアプリケーションに組み込みます。

```
dxxShredXML(CLOB(100K)    DAD,            /* input */
            CLOB(1M)      xmlobj,         /* input */
            long          returnCode,     /* output */
            varchar(1024) returnMsg)      /* output */
```

完全な構文および例については、247ページの『dxxShredXML()』 を参照してください。

例: dxxShredXML() への呼び出しの例を以下に示します。

```
#include "dxx.h"
#include "dxxrc.h"

EXEC SQL INCLUDE SQLCA;
        EXEC SQL BEGIN DECLARE SECTION;
         SQL TYPE is CLOB      dad;            /* DAD*/
         SQL TYPE is CLOB_FILE dadFile;        /* DAD file*/
         SQL TYPE is CLOB      xmlDoc;         /* input XML document */
         SQL TYPE is CLOB_FILE xmlFile;        /* input XMLfile */
         long                  returnCode;     /* error code */
         char                  returnMsg[1024]; /* error message text */
         short                 dad_ind;
         short                 xmlDoc_ind;
         short                 returnCode_ind;
         short                 returnMsg_ind;
        EXEC SQL END DECLARE SECTION;

        /* initialize host variable and indicators */
        strcpy(dadFile.name,"c:¥dxx¥samples¥dad¥getstart_xcollection.dad");
        dadFile.name_length=strlen("c:¥dxx¥samples¥dad¥getstart_xcollection.dad");
        dadFile.file_option=SQL_FILE_READ;
        strcpy(xmlFile.name,"c:¥dxx¥samples¥cmd¥getstart_xcollection.xml");
```

```
xmlFile.name_length=strlen("c:¥dxx¥samples¥cmd¥getstart_xcollection.xml");
xmlFile.file_option=SQL_FILE_READ;
SQL EXEC VALUES (:dadFile) INTO :dad;
SQL EXEC VALUES (:xmlFile) INTO :xmlDoc;
returnCode = 0;
returnMsg[0] = '¥0';
dad_ind = 0;
xmlDoc_ind = 0;
returnCode_ind = -1;
returnMsg_ind = -1;

/* Call the store procedure */
EXEC SQL CALL db2xml.dxxShredXML(:dad:dad_ind;
             :xmlDoc:xmlDoc_ind,
             :returnCode:returnCode_ind,:returnMsg:returnMsg_ind);
```

dxxInsertXML()

このストアード・プロシージャーは、定期的な更新を行うアプリケーションのために使用されます。同じタスクが繰り返されるので、パフォーマンスの改善が重要になります。 XML コレクションを使用可能にして、ストアード・プロシージャー内でコレクション名を使用すると、パフォーマンスが改善されます。 ストアード・プロシージャー dxxInsertXML() は dxxShredXML() と同様の働きをしますが、dxxInsertXML() は使用可能にされた XML コレクションを最初の入力パラメーターとして取ります。

ストアード・プロシージャー dxxInsertXML() は XML 文書を DAD ファイルに関連した、使用可能にされた XML コレクションに挿入します。 DAD ファイルには、コレクション表およびマッピングの指定が含まれています。コレクション表は、<Xcollection> での指定に応じて検査または作成されます。ストアード・プロシージャー dxxInsertXML() はその後、マッピングに従って XML 文書を分解し、タグのない XML データを、名前の指定された XML コレクションに挿入します。

合成のための対応するストアード・プロシージャーは dxxRetrieveXML() です。これはさらに、使用可能にされた XML コレクションの名前を取ります。

XML コレクションを分解する: dxxInsertXML()

以下のストアード・プロシージャー宣言を使用して、ストアード・プロシージャー呼び出しをアプリケーションに組み込みます。

```
dxxInsertXML(char(collectionName) collectionName,   /* input */
             CLOB(1M)       xmlobj,           /* input */
             long           returnCode,       /* output */
             varchar(1024)  returnMsg)        /* output */
```

完全な構文および例については、249ページの『dxxInsertXML()』 を参照してください。

例: dxxInsertXML() への呼び出しの例を以下に示します。

```
#include "dxx.h"
#include "dxxrc.h"

EXEC SQL INCLUDE SQLCA;
EXEC SQL BEGIN DECLARE SECTION;
         char                  collection[64];   /* name of an XML collection */
         SQL TYPE is CLOB_FILE xmlFile;          /* input XML file */
         SQL TYPE is CLOB      xmlDoc;           /* input XML doc */
         long                  returnCode;       /* error code */
         char                  returnMsg[1024];  /* error message text */
         short                 collection_ind;
         short                 xmlDoc_ind;
         short                 returnCode_ind;
         short                 returnMsg_ind;

         EXEC SQL END DECLARE SECTION;

         /* initialize host variable and indicators */
         strcpy(collection,"sales_ord")
         strcpy(xmlobj.name,"c:¥dxx¥samples¥cmd¥getstart_xcollection.xml");
         xmlobj.name_length=strlen("c:¥dxx¥samples¥cmd¥getstart_xcollection.xml");
         xmlobj.file_option=SQL_FILE_READ;
         SQL EXEC VALUES (:xmlFile) INTO (:xmlDoc);
         returnCode = 0;
         returnMsg[0] = '¥0';
         collection_ind = 0;
         xmlobj_ind = 0;
         returnCode_ind = -1;
         returnMsg_ind = -1;
         /* Call the store procedure */
         EXEC SQL CALL db2xml.dxxInsertXML(:collection:collection_ind;
                     :xmlDoc:xmlDoc_ind,
                     :returnCode:returnCode_ind,:returnMsg:returnMsg_ind);
```

XML コレクションにアクセスする

XML コレクションに対して、更新、削除、検索、および取り出しを行うことができます。しかし、XML コレクションを使用する目的は、タグのない純粋なデータをデータベース表に保管することであることを覚えておいてください。既存のデータベース表内のデータは、着信する XML 文書とは無関係です。更新、削除、および検索操作は、実際にはこれらの表に対する通常の SQL アクセスで構成されます。データが着信した XML 文書から分解されたものである場合、元の XML 文書は存在していません。

XML エクステンダーには、XML コレクションの視点からデータを操作する機能が備わっています。 UPDATE および DELETE SQL ステートメントを使用して、 XML 文書の合成に使用されるデータを変更し、その結果 XML コレクションを更新することができます。

考慮事項:

- 文書を更新するとき、他のコレクション表にとっては外部キーである、表の基本キーを含む行を削除しないでください。基本キーおよび外部キーの行が削除されると、その文書は削除されます。

- エレメントおよび属性値を置き換えるためには、文書を削除することなく、低レベルの表の行を削除および挿入することができます。
- 文書を削除するには、DAD に指定された先頭の element_node を合成する行を削除します。

XML コレクション内のデータを更新する

XML エクステンダーによって、 XML コレクション表に保管されたタグのないデータを更新することができます。 XML コレクション表の値を更新することにより、XML エレメントのテキスト、または XML 属性の値を更新することになります。さらに、更新によって、複数出現のエレメントまたは属性からデータのインスタンスを削除することができます。

SQL の観点からは、エレメントまたは属性の値を変更することは更新操作であり、エレメントまたは属性のインスタンスを削除することは削除操作です。 XML の観点からは、ルート element_node のエレメント・テキストまたは属性値が存在する限り、その XML 文書はまだ存在しているため、それは更新操作となります。

要件: XML コレクション内でデータを更新するときには、以下の規則に従ってください。

- 既存の表に基本キーと外部キーとの関係が存在する場合、コレクション表の間にもその関係を指定します。その関係が存在しない場合、結合される列が存在することを確認してください。
- DAD ファイルに指定された結合条件を、以下の場所に指定します。
  - SQL マッピングの場合、<SQL_stmt> エレメント
  - RDB_node マッピングの場合、ルート・エレメント・ノードの RDB_node

エレメントおよび属性値を更新する

XML コレクションでは、エレメント・テキストおよび属性値はすべてデータベース表内の列にマップされます。 列データがすでに存在しているか、または着信 XML 文書から分解されたものかに関係なく、そのデータを通常の SQL 更新技法を使用して置き換えます。

エレメントまたは属性値を更新するには、SQL UPDATE ステートメント内に WHERE 文節を指定し、 DAD ファイルに指定された結合条件をそこに含めます。

たとえば次のようにします。

```
UPDATE SHIP_TAB
    set MODE = 'BOAT'
  WHERE MODE='AIR' AND PART_KEY in
    (SELECT PART_KEY from PART_TAB WHERE ORDER_KEY=68)
```

<ShipMode> エレメントの値が SHIP_TAB 表で AIR から BOAT に更新されます。キーは 68 です。

エレメントおよび属性のインスタンスを削除する

複数出現のエレメントまたは属性を除去することによって合成済み XML 文書を更新するには、 WHERE 文節を使用して、エレメントまたは属性値に対応するフィールド値を含む行を削除します。先頭の element_node の値を含む行を削除しない限り、エレメントの値の削除は XML 文書の更新と見なされます。

たとえば、以下の DELETE ステートメントでは、そのサブエレメントのいずれかの固有値を指定することにより <shipment> が削除されます。

```
DELETE from SHIP_TAB
   WHERE DATE='1999-04-12'
```

DATE 値を指定すると、その値に対応する行が削除されます。合成済み文書には、最初に 2 つの <shipment> エレメントが含まれていましたが、現在は 1 つだけです。

XML 文書を XML コレクションから削除する

合成された XML 文書をコレクションから削除することができます。つまり、複数の XML 文書を合成する XML コレクションがある場合、それらの合成済み文書の 1 つを削除できるということです。

文書を削除するには、DAD ファイルに指定された先頭の element_node を合成する表内の行を削除します。この表には、最上位のコレクション表の基本キー、および低レベルの表の外部キーが含まれます。

たとえば、以下の DELETE ステートメントは基本キー列の値を指定します。

```
DELETE from order_tab
   WHERE order_key=1
```

ORDER_KEY は表 ORDER_TAB 内の基本キーであり、 XML 文書が合成されるときに先頭の element_node となります。この行を削除すると、合成の際に生成された 1 つの XML 文書が削除されます。そのため、XML の観点からは、1 つの XML 文書が XML コレクションから削除されます。

XML 文書を XML コレクションから取り出す

XML 文書を XML コレクションから取り出すことは、文書をコレクションから合成することと似ています。

XML 文書を取り出すには、ストアード・プロシージャー dxxRetrieveXML() を使用します。構文および例については、241ページの『dxxRetrieveXML()』 を参照してください。

DAD ファイルの考慮事項: XML 文書を分解して XML コレクションにする場合、 DAD ファイル内で順序を指定していない限り、複数出現のエレメントおよび属性値の順序が保持されないことがあります。この順序を保持するためには、RDB_node マッピング体系を使用してください。このマッピング体系により、RDB_node 内のルート・エレメントを含む表の orderBy 属性を指定することができます。

XML コレクションを検索する

この節では、以下の目的で XML コレクションを検索する方法を解説します。

- **検索基準を使用して XML 文書を生成する**

  このタスクは実際には条件を使用した合成です。以下の検索基準を使用して検索基準を指定できます。

  – DAD ファイルの text_node および attribute_node に条件を指定します。

  – dxxGenXML() および dxxRetrieveXML() ストアード・プロシージャーを使用しているとき、 *overwrite* パラメーターを指定します。

    たとえば、DAD ファイル `order.dad` を使用して XML コレクションの sales_ord を使用可能にしたものの、 Web から導出したデータを使用して価格をオーバーライドしたい場合、以下のようにして <SQL_stmt> DAD エレメントの値をオーバーライドすることができます。

```
EXEC SQL INCLUDE SQLCA;
EXEC SQL BEGIN DECLARE SECTION;
       ...

EXEC SQL END DECLARE SECTION;

         float    price_value;

         /* create table */
         EXEC SQL CREATE TABLE xml_order_tab (xmlorder XMLVarchar);

         /* initialize host variable and indicators */
         strcpy(collection,"sales_ord");
         strcpy(result_tab,"xml_order_tab");
         overrideType = SQL_OVERRIDE;
         max_row = 20;
         num_row = 0;
         returnCode = 0;
```

```
msg_txt[0] = '¥0';
override_ind = 0;
overrideType_ind = 0;
rtab_ind = 0;
maxrow_ind = 0;
numrow_ind = -1;
returnCode_ind = -1;
returnMsg_ind = -1;

/* get the price_value from some place, such as form data */
price_value = 1000.00         /* for example*/

/* specify the overwrite */
sprintf(overwrite,
       "SELECT o.order_key, customer, p.part_key, quantity, price,
              tax, ship_id, date, mode
       FROM order_tab o, part_tab p,
           table(select substr(char(timestamp(generate_unique())),16)
           as ship_id, date, mode from ship_tab)as s
       WHERE p.price > %d and s.date >'1996-06-01' AND
           p.order_key = o.order_key and s.part_key = p.part_key",
           price_value);

/* Call the store procedure */
EXEC SQL CALL db2xml.dxxRetrieve(:collection:collection_ind,
       :result_tab:rtab_ind,
       :overrideType:overrideType_ind,:overwrite:overwrite_ind,
       :max_row:maxrow_ind,:num_row:numrow_ind,
       :returnCode:returnCode_ind,:returnMsg:returnMsg_ind);
```

order.dad 内の price > 2500.00 の条件は、 price > ? によってオーバーライドされます。ここで ? は、入力変数 *price_value* に基づいています。

- **分解された XML データを検索する:**

  コレクション表の検索には通常の SQL 照会操作を使用できます。コレクション表を結合するか、または副照会を使用してから、テキスト列に対して構造化テキスト検索を実行することができます。構造化検索の結果を使用して、指定した XML 文書の取り出しまたは生成にそのデータを適用することができます。

第4部 参照情報

ここでは、XML エクステンダー管理コマンド、ユーザー定義のデータ・タイプ (UDT)、ユーザー定義の関数 (UDF)、およびストアード・プロシージャーについての構文情報を示します。さらに、問題判別のためのメッセージ・テキストも示します。

第7章 XML エクステンダー管理コマンド: dxxadm

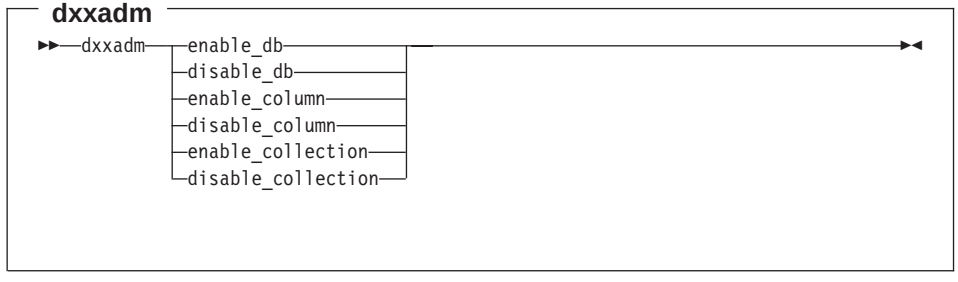
XML エクステンダーには、以下の管理タスクを完了するための管理コマンド **dxxadm** が備わっています。さまざまなコマンド・オプションを使用して **dxxadm** を呼び出すことによって、以下のXML エクステンダー管理タスクを実行します。

- データベースを XML エクステンダーに対して使用可能または使用不可にする
- XML 列を使用可能または使用不可にする
- XML コレクションを使用可能または使用不可にする

XML エクステンダー管理ウィザードまたはストアード・プロシージャーを使用して、それぞれの管理タスクを実行することもできます。

高レベルの構文

以下の構文図は、**dxxadm** コマンドの高レベルの構文を示しています。各オプションについては、以降の節で説明します。



管理コマンド・オプション

以下の節では、システム・プログラマーが使用できる各 **dxxadm** コマンド・オプションについて説明します。

- 173ページの『enable_db』
- 175ページの『disable_db』
- 177ページの『enable_column』
- 179ページの『disable_column』

- 181ページの『enable_collection』
- 183ページの『disable_collection』

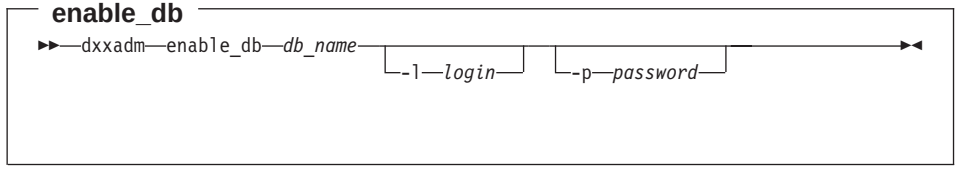
enable_db

目的

データベースに接続して使用可能にし、XML エクステンダーで使用できるようにします。データベースを使用可能にすると、XML エクステンダーは以下のオブジェクトを作成します。

- XML エクステンダー・ユーザー定義タイプ (UDT)
- XML エクステンダー・ユーザー定義関数 (UDF)
- XML エクステンダー DTD 参照表 (DTD_REF)。これは DTD およびそれぞれの DTD に関する情報を保管します。 DTD_REF 表についての詳細は、251ページの『DTD 参照表』を参照してください。
- XML エクステンダー使用状況表 (XML_USAGE)。これは XML に使用できるそれぞれの列およびコレクションに関する一般情報を保管します。 XML_USAGE 表についての詳細は、252ページの『XML 使用状況表』を参照してください。

形式



パラメーター

表 14. enable_db のパラメーター

パラメーター	説明
db_name	XML データが存在するデータベースの名前。
-l *login*	指定した場合にデータベースへの接続に使用される、任意指定のユーザー ID。 指定しない場合、現行のユーザー ID が使用されます。
-p *password*	指定した場合にデータベースへの接続に使用される、任意指定のパスワード。 指定しない場合、現行のパスワードが使用されます。

例

以下の例では、データベース SALES_DB が使用可能にされます。

```
dxxadm enable_db SALES_DB
```

disable_db

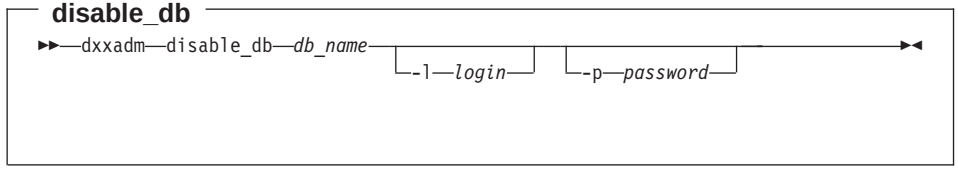
目的

XML が使用可能となっているデータベースに接続して、それを使用不可にします。データベースが使用不可になると、XML エクステンダーがそのデータベースを使用することはできなくなります。 XML エクステンダーがデータベースを使用不可にすると、以下のオブジェクトを除去します。

- XML エクステンダー・ユーザー定義タイプ (UDT)
- XML エクステンダー・ユーザー定義関数 (UDF)
- XML エクステンダー DTD 参照表 (DTD_REF)。これは DTD およびそれぞれの DTD に関する情報を保管します。 DTD_REF 表についての詳細は、251ページの『DTD 参照表』を参照してください。
- XML エクステンダー使用状況表 (XML_USAGE)。これは XML に使用できるそれぞれの列およびコレクションに関する一般情報を保管します。 XML_USAGE 表についての詳細は、252ページの『XML 使用状況表』を参照してください。

重要: データベースを使用不可にする前に、すべての XML 列を使用不可にする必要があります。 XML エクステンダーは、XML に使用できる列またはコレクションを含んでいるデータベースを使用不可にできません。

形式



パラメーター

表 15. disable_db のパラメーター

パラメーター	説明
db_name	XML データが存在するデータベースの名前。
-l *login*	指定した場合にデータベースへの接続に使用される、任意指定のユーザー ID。 指定しない場合、現行のユーザー ID が使用されます。

表 15. *disable_db* のパラメーター *(続き)*

パラメーター	説明
-p *password*	指定した場合にデータベースへの接続に使用される、任意指定のパスワード。 指定しない場合、現行のパスワードが使用されます。

例

以下の例では、データベース SALES_DB が使用不可にされます。

```
dxxadm disable_db SALES_DB
```

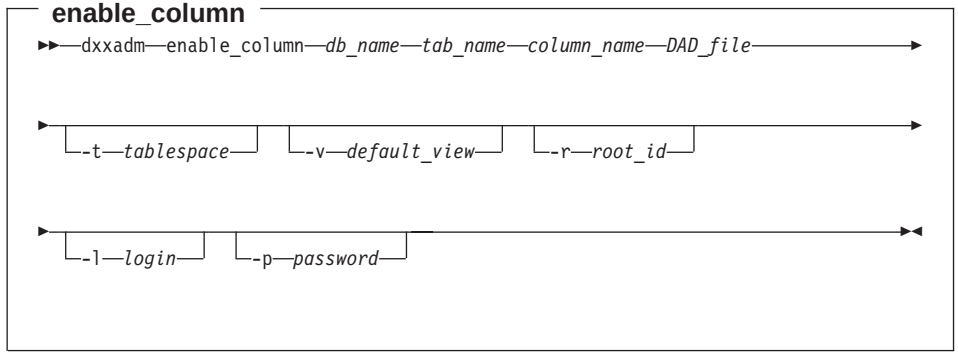
enable_column

目的

データベースに接続して XML 列を使用可能にし、そこに XML エクステンダー UDT を含めることができるようにします。 XML エクステンダーは列を使用可能にする時に以下のタスクを行います。

- XML 表に基本キーがあるかどうかを判別します。ない場合は、XML エクステンダーが XML 表を更新して DXXROOT_ID という列を追加します。
- DAD ファイルの中で指定されたサイド表を作成します。この表には XML 表内の各行の固有な ID を含む列があります。 この列は、ユーザーが指定した *root_id* か、または XML エクステンダーが命名する DXXROOT_ID です。
- XML 表およびそのサイド表のデフォルト視点を作成します。オプションで、ユーザーが名前を指定できます。

形式



パラメーター

表 16. enable_column のパラメーター

パラメーター	説明
db_name	XML データが存在するデータベースの名前。
tab_name	XML 列が存在する表の名前。
column_name	XML 列の名前。
DAD_file	XML 文書を XML 列およびサイド表にマップする DAD ファイルの名前。

表 16. *enable_column* のパラメーター *(続き)*

パラメーター	説明
-t *tablespace*	任意指定であり、XML 列に関連したサイド表を含む、表スペース。 これを指定しない場合、デフォルトの表スペースが使用されます。
-v *default_view*	XML 列とサイド表を結合する、任意指定のデフォルト・ビューの名前。
-r *root_id*	表テーブルの root_id として使用される、XML 列表内の基本キーの名前。 root_id は任意指定です。
-l *login*	指定した場合にデータベースへの接続に使用される、任意指定のユーザー ID。 指定しない場合、現行のユーザー ID が使用されます。
-p *password*	指定した場合にデータベースへの接続に使用される、任意指定のパスワード。 指定しない場合、現行のパスワードが使用されます。

例

以下の例では、XML 列が使用可能にされます。

```
dxxadm enable_column SALES_DB SALES_TAB ORDER -v sales_order_view -r INVOICE_NUMBER
```

disable_column

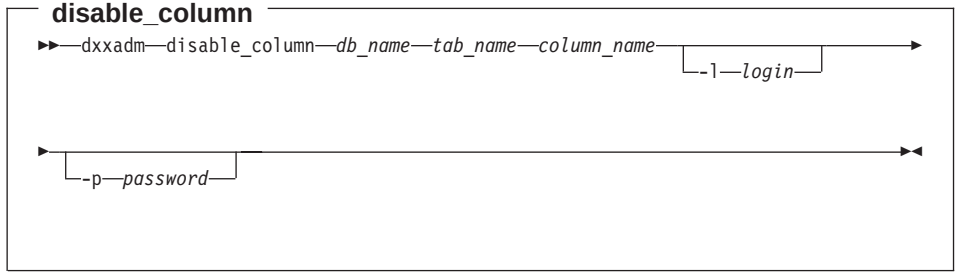
目的

データベースに接続して、XML が使用可能となっている列を使用不可にします。列が使用不可になると、そこに XML データ・タイプを含めることはできなくなります。 XML が使用可能になっている列が使用不可になると、以下の処置が実行されます。

- XML 列の使用項目が XML_USAGE 表から削除されます。
- USAGE_COUNT が DTD_REF 表内で減分されます。
- この列に関連したすべてのトリガーが除去されます。
- この列に関連したすべてのサイド表が除去されます。

重要: XML 表を除去する前に、XML 列を使用不可にしなければなりません。XML 表が除去されてもその XML 列が使用不可にされていない場合、 XML エクステンダーは作成したサイド表および XML 列項目の両方を XML_USAGE 表内に保ちます。

形式



パラメーター

表 17. disable_column のパラメーター

パラメーター	説明
db_name	データが存在するデータベースの名前。
tab_name	XML 列が存在する表の名前。
column_name	XML 列の名前。
-l *login*	指定した場合にデータベースへの接続に使用される、任意指定のユーザー ID。 指定しない場合、現行のユーザー ID が使用されます。

表 17. *disable_column* のパラメーター *(続き)*

パラメーター	説明
-p *password*	指定した場合にデータベースへの接続に使用される、任意指定のパスワード。 指定しない場合、現行のパスワードが使用されます。

例

以下の例では、XML が使用可能とされている列を使用不可にします。

```
dxxadm disable_column SALES_DB SALES_TAB ORDER
```

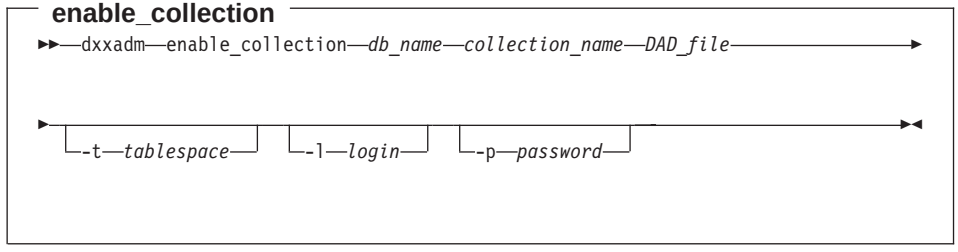
enable_collection

目的

指定した DAD に従って、データベースに接続して XML コレクションを使用可能にします。 コレクションを使用可能にするとき、XML エクステンダーは以下のタスクを実行します。

- XML コレクションの使用項目を XML_USAGE 表内に作成します。
- RDB_node マッピングでは、表がデータベースに存在しない場合、 DAD で指定されたコレクション表を作成します。

形式



パラメーター

表 18. enable_collection のパラメーター

パラメーター	説明
db_name	データが存在するデータベースの名前。
collection_name	XML コレクションの名前。
DAD_file	XML 文書をコレクション内のリレーショナル表にマップする DAD ファイルの名前。
-t *tablespace*	任意指定であり、コレクションに関連した表スペースの名前。 これを指定しない場合、デフォルトの表スペースが使用されます。
-l *login*	指定した場合にデータベースへの接続に使用される、任意指定のユーザー ID。 指定しない場合、現行のユーザー ID が使用されます。

表 18. enable_collection のパラメーター (続き)

パラメーター	説明
-p *password*	指定した場合にデータベースへの接続に使用される、任意指定のパスワード。 指定しない場合、現行のパスワードが使用されます。

例

以下の例では、XML コレクションが使用可能にされます。

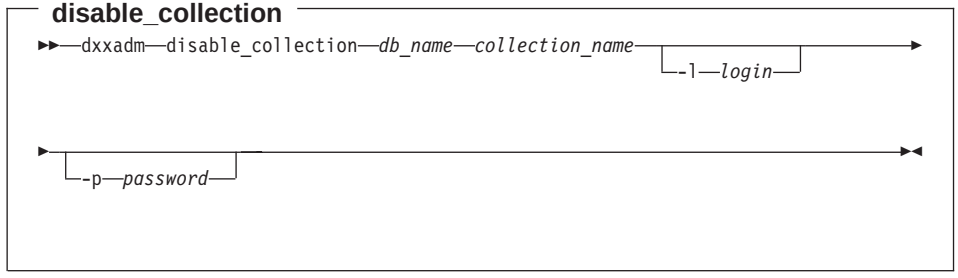
```
dxxadm enable_collection SALES_DB sales_ord getstart_xcollection.dad -t orderspace
```

disable_collection

目的

データベースに接続して、XML が使用可能となっているコレクションを使用不可にします。 そのコレクション名は、合成 (dxxRetrieveXML) および分解 (dxxInsertXML) ストアード・プロシージャーで使用できなくなります。 XML コレクションが使用不可にされると、関連したコレクション項目が XML_USAGE 表から削除されます。 コレクションを使用不可にしても、enable_collection ステップで作成したコレクション表は除去されないことに注意してください。

形式



パラメーター

表 19. disable_collection のパラメーター

パラメーター	説明
db_name	データが存在するデータベースの名前。
collection_name	XML コレクションの名前。
-l *login*	指定した場合にデータベースへの接続に使用される、任意指定のユーザー ID。 指定しない場合、現行のユーザー ID が使用されます。
-p *password*	指定した場合にデータベースへの接続に使用される、任意指定のパスワード。 指定しない場合、現行のパスワードが使用されます。

例

以下の例では、XML コレクションが使用不可にされます。

```
dxxadm disable_collection SALES_DB sales_ord
```

第8章 XML エクステンダーのユーザー定義タイプ

XML エクステンダーのユーザー定義タイプ (UDT) は、 XML 列および XML コレクションに使用されるデータ・タイプです。 UDT はすべて、スキーマ名 db2xml を持ちます。 XML エクステンダーは、XML 文書を保管および検索するための UDT を作成します。 表20 は、UDT の概要を示しています。

表 20. XML エクステンダー UDT

ユーザー定義タイプ列	ソース・データ・タイプ	使用法の説明
XMLVARCHAR	VARCHAR(*varchar_len*)	XML 文書の全体を VARCHAR として DB2 内に保管します。
XMLCLOB	CLOB(*clob_len*)	XML 文書の全体を文字ラージ・オブジェクト (CLOB) として DB2 内に保管します。
XMLFILE	VARCHAR(512)	ローカル・ファイル・サーバーのファイル名を指定します。 XMLFILE を XML 列に指定した場合、XML エクステンダーは XML 文書を外部サーバー・ファイルに保管します。テキスト・エクステンダーを XMLFILE によって使用可能にすることはできません。ファイル内容と DB2、および索引付け用に作成されたサイド表との間の整合性は、ユーザーの責任で保持してください。

ここで、*varchar_len* および *clob_len* はオペレーティング・システムによって決まります。

DB2 UDB では、*varchar_len* = 3K および *clob_len* = 2G となります。

これらの UDT は、アプリケーション列のタイプを指定するためだけに使用されます。それらは、XML エクステンダーが作成するサイド表には適用されません。

第9章 XML エクステンダーのユーザー定義関数

XML エクステンダーには、XML 文書を保管、取り出し、検索、および更新する関数、そして XML エレメントまたは属性を抽出する関数が備わっています。 XML ユーザー定義 (UDF) を XML 列に使用することはできますが、XML コレクションには使用できません。 すべての UDF にはスキーマ名 db2xml があり、これは UDF の前で省略できます。

4 種類の XML エクステンダー関数は、保管関数、検索関数、抽出関数、および更新関数です。

保管関数

保管関数は、XML 文書を DB2 データベースに挿入します。構文および例については、189ページの『保管関数』を参照してください。

検索関数

検索関数は、XML 文書を DB2 データベース内の XML 列から検索します。構文および例については、194ページの『検索関数』を参照してください。

抽出関数

抽出関数は、XML 文書からエレメント内容または属性値を抽出して、関数名によって指定されたデータ・タイプに変換します。 XML エクステンダーには、種々の SQL データ・タイプ用の抽出関数のセットが備わっています。 構文および例については、201ページの『抽出関数』を参照してください。

更新関数

Update() 関数は、エレメント内容または属性値を変更して、ロケーション・パスによって指定された更新値を持つ XML 文書のコピーを戻します。 Update() 関数では、アプリケーション・プログラマーが更新するエレメントまたは属性を指定できます。構文および例については、218ページの『更新関数』を参照してください。

表21 に、XML エクステンダー関数の要約を示します。

表 21. XML エクステンダーのユーザー定義関数

種類	関数
保管関数	XMLVarcharFromFile()
	XMLCLOBFromFile()
	XMLFileFromVarchar()
	XMLFileFromCLOB()
検索関数	Content(): XMLFile から取り出して CLOB に入れます
	Content(): XMLVarchar から取り出して外部サーバー・ファイルに入れます
	Content(): XMLCLOB から取り出して外部サーバー・ファイルに入れます
抽出関数	extractInteger() および extractIntegers()
	extractSmallint() および extractSmallints()
	extractDouble() および extractDoubles()
	extractReal() および extractReals()
	extractChar() および extractChars()
	extractVarchar() および extractVarchars()
	extractCLOB() および extractCLOBs()
	extractDate() および extractDates()
	extractTime() および extractTimes()
	extractTimestamp() および extractTimestamps()
更新関数	Update()

UDF でパラメーター・マーカーを使用するときには、JDBC 制限の要件として、 UDF 用のパラメーター・マーカーを、戻りデータが挿入される列のデータ・タイプにキャストすることが必要です。パラメーター・マーカーのキャスト方法を学ぶには、 148ページの『JDBC から関数を呼び出すときの制限』を参照してください。

保管関数

保管関数を使用して、XML 文書を DB2 データベースに挿入します。 UDT ディレクトリーのデフォルト・キャスト関数を INSERT または SELECT ステートメント内に使用できます。 これについては132ページの『データを保管する』で説明しています。さらに、XML エクステンダーには XML 文書を UDT 基本データ・タイプ以外のソースからアクセスして、指定した UDT に変換する UDF が備わっています。

XML エクステンダーには、以下の保管関数が備わっています。

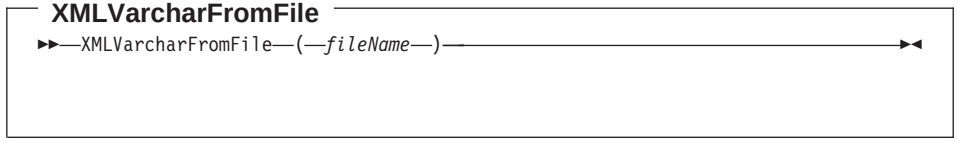
- 190ページの『XMLVarcharFromFile()』
- 191ページの『XMLCLOBFromFile()』
- 192ページの『XMLFileFromVarchar()』
- 193ページの『XMLFileFromCLOB()』

XMLVarcharFromFile()

目的

XML 文書をサーバー・ファイルから読み取り、文書を XMLVARCHAR タイプで戻します。

構文



パラメーター

表 22. XMLVarcharFromFile パラメーター

パラメーター	データ・タイプ	説明
fileName	VARCHAR(512)	完全修飾のサーバー・ファイル名。

戻りタイプ

XMLVARCHAR

例

以下の例では、XML 文書をサーバー・ファイルから読み取り、文書を XMLVARCHAR タイプとして XML 列に挿入します。

```
EXEC SQL INSERT INTO sales_tab(ID, NAME, ORDER)
   VALUES('1234', 'Sriram Srinivasan',
          XMLVarcharFromFile('c:¥dxx¥samples¥cmd¥getstart.xml'))
```

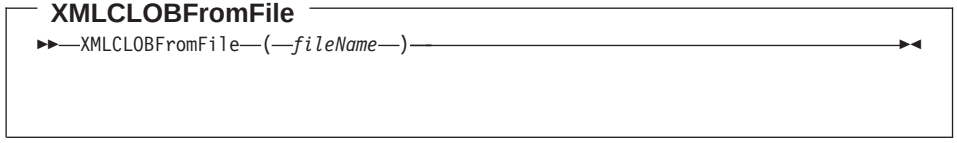
この例では、レコードが SALES_TAB 表に挿入されます。 関数 XMLVarcharFromFile() は、XML 文書をファイルから DB2 にインポートして、それを XMLVARCHAR として保管します。

XMLCLOBFromFile()

目的

XML 文書をサーバー・ファイルから読み取り、文書を XMLCLOB タイプで戻します。

構文



パラメーター

表 23. XMLCLOBFromFile パラメーター

パラメーター	データ・タイプ	説明
fileName	VARCHAR(512)	完全修飾のサーバー・ファイル名。

戻りタイプ

XMLCLOB を LOCATOR として

例

以下の例では、XML 文書をサーバー・ファイルから読み取り、文書を XMLCLOB タイプとして XML 列に挿入します。

```
EXEC SQL INSERT INTO sales_tab(ID, NAME, ORDER)
   VALUES('1234', 'Sriram Srinivasan',
          XMLCLOBFromFile('c:¥dxx¥samples¥cmd¥getstart.xml'))
```

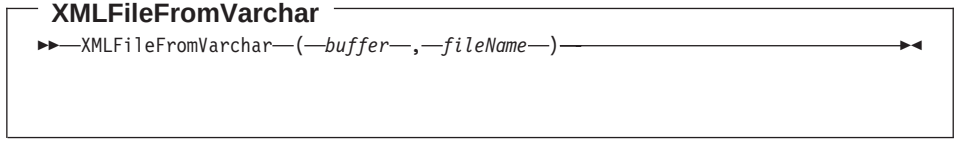
SALES_TAB 表の列 ORDER は、XMLCLOB タイプとして定義されます。上記の例は、列 ORDER が SALES_TAB 表にどのように挿入されるかを示しています。

XMLFileFromVarchar()

目的

XML 文書をメモリーから VARCHAR として読み取り、それを外部サーバー・ファイルに書き込んで、ファイル名およびパスを XMLFILE タイプで戻します。

構文



パラメーター

表 24. XMLFileFromVarchar パラメーター

パラメーター	データ・タイプ	説明
buffer	VARCHAR(3K)	メモリー・バッファー。
fileName	VARCHAR(512)	完全修飾のサーバー・ファイル名。

戻りタイプ

XMLFILE

例

以下の例では、XML 文書をメモリーから VARCHAR として読み取り、それを外部サーバー・ファイルに書き込んで、ファイル名およびパスを XMLFILE タイプとして XML 列に挿入します。

```
EXEC SQL BEGIN DECLARE SECTION;
      struct { short len; char data[3000]; } xml_buff;
EXEC SQL END DECLARE SECTION;

EXEC SQL INSERT INTO sales_tab(ID, NAME, ORDER)
   VALUES('1234', 'Sriram Srinivasan',
       XMLFileFromVarchar(:xml_buf, 'c:¥dxx¥samples¥cmd¥getstart.xml'))
```

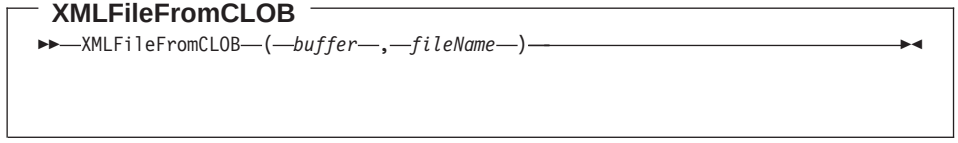
SALES_TAB 表の列 ORDER は、XMLFILE タイプとして定義されます。上記の例は、バッファー内に XML 文書があれば、それをサーバー・ファイルに保管できることを示しています。

XMLFileFromCLOB()

目的

XML 文書を CLOB ロケーターとして読み取り、それを外部サーバー・ファイルに書き込んで、ファイル名およびパスを XMLFILE タイプで戻します。

構文



パラメーター

表 25. XMLFileFromCLOB() パラメーター

パラメーター	データ・タイプ	説明
buffer	CLOB を LOCATOR として	XML 文書を含むバッファー。
fileName	VARCHAR(512)	完全修飾のサーバー・ファイル名。

戻りタイプ

XMLFILE

例

以下の例では、XML 文書を CLOB ロケーターとして読み取り、それを外部サーバー・ファイルに書き込んで、ファイル名およびパスを XMLFILE タイプとして XML 列に挿入します。

```
EXEC SQL BEGIN DECLARE SECTION;
     SQL TYPE IS CLOB_LOCATOR xml_buff;
EXEC SQL END DECLARE SECTION;

EXEC SQL INSERT INTO sales_tab(ID, NAME, ORDER)
   VALUES('1234', 'Sriram Srinivasan',
           XMLFileFromCLOB(:xml_buf, 'c:¥dxx¥samples¥cmd¥getstart.xml'))
```

SALES_TAB 表の列 ORDER は、XMLFILE タイプとして定義されます。バッファー内に XML 文書があれば、それをサーバー・ファイルに保管することができます。

検索関数

デフォルト・キャスト関数を使用して、XML UDT を基本データ・タイプに変換できます。 これについては 135ページの『文書全体を取り出す』で説明しています。 XML エクステンダーにはさらに、検索に使用する*多重定義関数* Content() が備わっています。

Content() 関数は、以下の種類の検索を実行します。

- **サーバーの外部ストレージからの、クライアントのホスト変数の検索**

  Content() を使用して、XML 文書が外部サーバー・ファイルに保管されたときに、それを取り出してメモリー・バッファーに入れることができます。この目的のために 195ページの『Content(): XMLFILE から取り出して CLOB に入れます』を使用できます。

- **内部記憶装置から取り出して外部サーバー・ファイルに入れる**

  さらに、Content() を使用して、DB2 内に保管された XML 文書を取り出し、 DB2 サーバーのファイル・システム上にあるサーバー・ファイルに保管することができます。以下の Content() 関数は、外部サーバー・ファイル上に情報を保管するために使用されます。

  - 197ページの『Content(): XMLVARCHAR から取り出して外部サーバー・ファイルに入れます』
  - 199ページの『Content(): XMLCLOB から取り出して外部サーバー・ファイルに入れます』

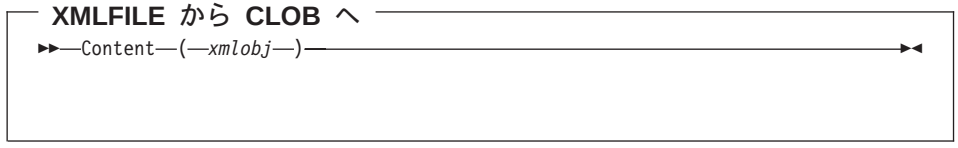
以下の節に示す例では、各コマンドの先頭に 『DB2』 とタイプする必要のない、 DB2 コマンド・シェルを使用していると仮定します。

Content(): XMLFILE から取り出して CLOB に入れます

目的

サーバー・ファイルからデータを検索して、CLOB LOCATOR に保管します。

構文



パラメーター

表 26. XMLFILE から CLOB へのパラメーター

パラメーター	データ・タイプ	説明
xmlobj	XMLFILE	XML 文書。

戻りタイプ

CLOB (*clob_len*) を LOCATOR として

DB2 UDB の *clob_len* は 2G です。

例

以下の例では、サーバー・ファイルからデータを検索して、CLOB ロケーターに保管します。

```
EXEC SQL BEGIN DECLARE SECTION;
     SQL TYPE IS CLOB_LOCATOR xml_buff;
EXEC SQL END DECLARE SECTION;

EXEC SQL CONNECT TO SALES_DB

EXEC SQL DECLARE c1 CURSOR FOR

     SELECT Content(order) from sales_tab
     WHERE sales_person = 'Sriram Srinivasan'

EXEC SQL OPEN c1;

do {
   EXEC SQL FETCH c1 INTO :xml_buff;
   if (SQLCODE != 0) {
      break;
      }
   else {
```

```
     /* do with the XML doc in buffer */
     }
   }
SQL CLOSE c1;

EXEC SQL CONNECT RESET;
```

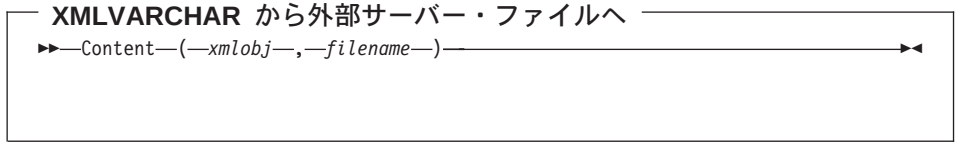
SALES_TAB 内の列 ORDER は XMLFILE タイプなので、 Content() UDF はサーバー・ファイルからデータを検索してそれを CLOB ロケーターに保管します。

Content(): XMLVARCHAR から取り出して外部サーバー・ファイルに入れます

目的

XMLVARCHAR タイプとして保管されている XML の内容を検索して、それを外部サーバー・ファイルに保管します。

構文



重要: 指定された名前のファイルがすでに存在する場合、コンテンツ関数はその内容を上書きします。

注:

パラメーター

表 27. XMLVarchar から外部サーバー・ファイルへのパラメーター

パラメーター	データ・タイプ	説明
xmlobj	XMLVARCHAR	XML 文書。
filename	VARCHAR(512)	完全修飾のサーバー・ファイル名。

戻りタイプ

VARCHAR(512)

例

以下の例では、XMLVARCHAR タイプとして保管されている XML の内容を検索して、それを外部サーバー・ファイルに保管します。

```
CREATE table app1 (id int NOT NULL, order db2xml.XMLVarchar);
INSERT into app1 values (1, '<?xml version="1.0"?>
  <!DOCTYPE SYSTEM c:¥dxx¥samples¥dtd¥getstart.dtd"->
    <Order key="1">
      <Customer>
          <Name>American Motors</Name>
          <Email>parts@am.com</Email>
      </Customer>
      <Part color="black">
```

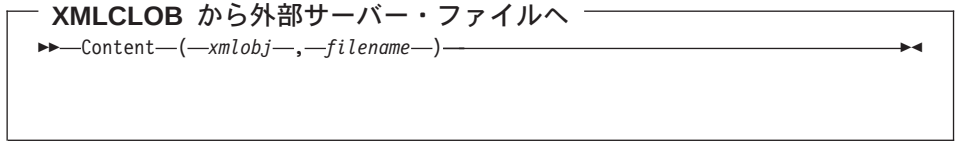
```
            <key>68</key>
            <Quantity>36</Quantity>
            <ExtendedPrice>34850.16</ExtendedPrice>
            <Tax>6.000000e-02</Tax>
            <Shipment>
                <ShipDate>1998-08-19</ShipDate>
                <ShipMode>AIR </ShipMode>
            </Shipment>
            <Shipment>
                <ShipDate>1998-08-19</ShipDate>
                <ShipMode>BOAT </ShipMode>
            </Shipment>
        </Order>');

SELECT db2xml.Content(order, 'c:¥dxx¥samples¥cmd¥getstart_.dad')
  from app1 where ID=1;
```

Content(): XMLCLOB から取り出して外部サーバー・ファイルに入れます

目的

XMLCLOB タイプとして保管されている XML の内容を検索して、それを外部サーバー・ファイルに保管します。

構文



重要: 指定された名前のファイルがすでに存在する場合、コンテンツ関数はその内容を上書きします。

パラメーター

表 28. XMLCLOB から外部サーバー・ファイルへのパラメーター

パラメーター	データ・タイプ	説明
xmlobj	XMLCLOB を LOCATOR として	XML 文書。
filename	VARCHAR(512)	完全修飾のサーバー・ファイル名。

戻りタイプ

VARCHAR(512)

例

以下の例では、XMLCLOB タイプとして保管されている XML の内容を検索して、それを外部サーバー・ファイルに保管します。

```
CREATE table app1 (id int NOT NULL, order db2xml.XMLCLOB not logged);
INSERT into app1 values (1, '<?xml version="1.0"?>
  <!DOCTYPE SYSTEM c:¥dxx¥samples¥dtd¥getstart.dtd"->
    <Order key="1">
      <Customer>
          <Name>American Motors</Name>
          <Email>parts@am.com</Email>
      </Customer>
      <Part color="black">
          <key>68</key>
          <Quantity>36</Quantity>
          <ExtendedPrice>34850.16</ExtendedPrice>
```

```
            <Tax>6.000000e-02</Tax>
            <Shipment>
                <ShipDate>1998-08-19</ShipDate>
                <ShipMode>AIR </ShipMode>
            </Shipment>
            <Shipment>
                <ShipDate>1998-08-19</ShipDate>
                <ShipMode>BOAT </ShipMode>
            </Shipment>
        </Order>');

SELECT db2xml.Content(order, 'c:¥dxx¥samples¥cmd¥getstart.xml')
from app1 where ID=1;
```

抽出関数

抽出関数は、XML 文書からエレメント内容または属性値を抽出して、要求された SQL データ・タイプを戻します。 XML エクステンダーには、種々の SQL データ・タイプ用の抽出関数のセットが備わっています。 抽出関数には、2 つの入力パラメーターがあります。 1 番目のパラメーターは XML エクステンダー UDT で、それには XML UDT の 1 つを指定できます。 2 番目のパラメーターは、XML エレメントまたは属性を指定するロケーション・パスです。それぞれの抽出関数は、ロケーション・パスによって指定される値または内容を戻します。

エレメントまたは属性値には複数出現するものもあるので、抽出関数はスカラー値または表値を戻します。その表値は、表関数と呼ばれます。

XML エクステンダーには、以下の抽出関数が備わっています。

- 202ページの『extractInteger() および extractIntegers()』
- 204ページの『extractSmallint() および extractSmallints()』
- 205ページの『extractDouble() および extractDoubles()』
- 207ページの『extractReal() および extractReals()』
- 208ページの『extractChar() および extractChars()』
- 209ページの『extractVarchar() および extractVarchars()』
- 211ページの『extractCLOB() および extractCLOBs()』
- 213ページの『extractDate() および extractDates()』
- 214ページの『extractTime() および extractTimes()』
- 216ページの『extractTimestamp() および extractTimestamps()』

以下の節に示す例では、各コマンドの先頭に 『DB2』 とタイプする必要のない、 DB2 コマンド・シェルを使用していると仮定します。

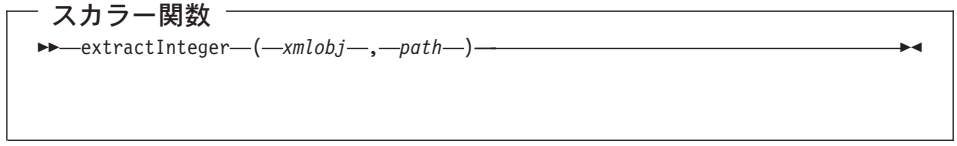
extractInteger() および extractIntegers()

目的

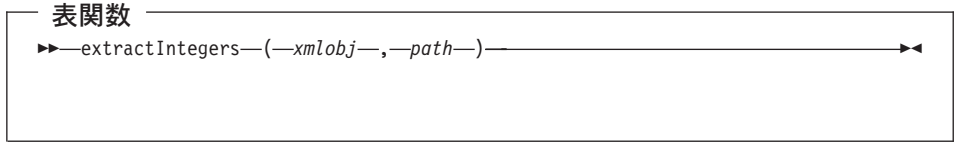
エレメント内容または属性値を XML 文書から抽出して、データを INTEGER タイプで戻します。

構文

スカラー関数



表関数



パラメーター

表 29. extractInteger および extractIntegers 関数のパラメーター

パラメーター	データ・タイプ	説明
xmlobj	XMLVARCHAR、XMLFILE、または XMLCLOB	列名。
path	VARCHAR	エレメントまたは属性のロケーション・パス。

戻りタイプ

INTEGER

戻り列名 (表関数)

returnedInteger

例

スカラー関数の例:

以下の例では、キーの属性値が ″1″ のときに 1 つの値が戻されます。この値は、自動的に DECIMAL タイプに変換される整数として抽出されます。

```
CREATE TABLE t1(key decimal(3,2));
INSERT into t1 values
SELECT * from table(db2xml.extractInteger(db2xml.XMLFile
   ('c:¥dxx¥samples¥xml¥getstart.xml'), '/Order/[@key="1"]'));
SELECT * from t1;
```

表関数の例:

以下の例では、販売注文のそれぞれのキー値が INTEGER として抽出されます。

```
SELECT * from table(db2xml.extractIntegers(db2xml.XMLFile
   ('c:¥dxx¥samples¥xml¥getstart.xml'), '/Order/@key')) as x;
```

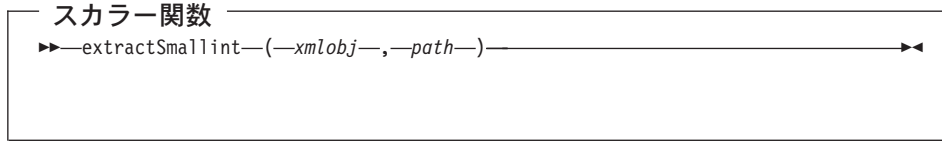
extractSmallint() および extractSmallints()

目的

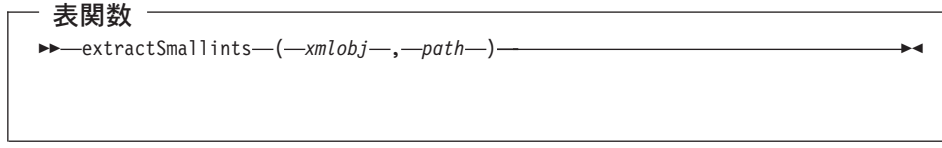
エレメント内容または属性値を XML 文書から抽出して、データを SMALLINT タイプで戻します。

構文

スカラー関数



表関数



パラメーター

表 30. extractSmallint および extractSmallints 関数のパラメーター

パラメーター	データ・タイプ	説明
xmlobj	XMLVARCHAR、XMLFILE、または XMLCLOB	列名。
path	VARCHAR	エレメントまたは属性のロケーション・パス。

戻りタイプ

SMALLINT

戻り列名 (表関数)

returnedSmallint

例

以下の例では、すべての販売注文における数量の値が SMALLINT として抽出されます。

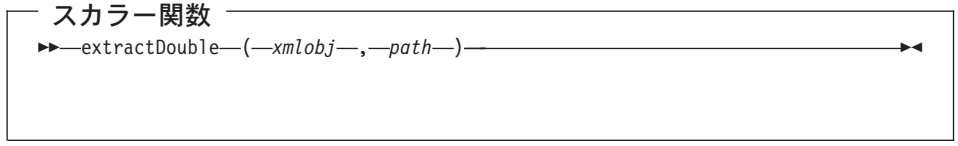
```
SELECT * from table(db2xml.extractSmallints(db2xml.File
   ('c:¥dxx¥samples¥xml¥getstart.xml'), '/Order/Part/Quantity')) as x;
```

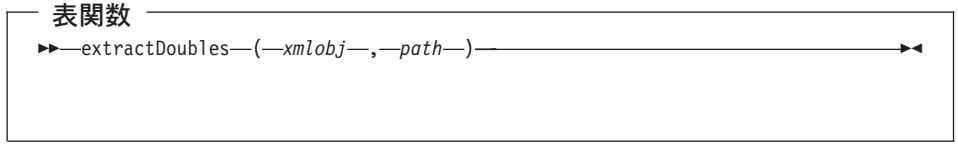
extractDouble() および extractDoubles()

目的

エレメント内容または属性値を XML 文書から抽出して、データを DOUBLE タイプで戻します。

構文





パラメーター

表 31. extractDouble および extractDoubles 関数のパラメーター

パラメーター	データ・タイプ	説明
xmlobj	XMLVARCHAR、XMLFILE、または XMLCLOB	列名。
path	VARCHAR	エレメントまたは属性のロケーション・パス。

戻りタイプ

DOUBLE

戻り列名 (表関数)

returnedDouble

例

スカラー関数の例:

以下の例では、注文価格を DOUBLE タイプから DECIMAL タイプに自動的に変換します。

```
CREATE TABLE t1(price, DECIMAL(5,2));
INSERT into t1 values (db2xml.extractDouble(db2xml.XMLFile
   ('c:¥dxx¥samples¥xml¥getstart.xml'),
    '/Order/Part[@color="black "]/ExtendedPrice'));
SELECT * from t1;
```

表関数の例:

以下の例では、販売注文のそれぞれの部分における ExtendedPrice の値が DOUBLE として抽出されます。

```
SELECT * from table(db2xml.extractDoubles(db2xml.XMLFile
   ('c:¥dxx¥samples¥xml¥getstart.xml'), '/Order/Part/ExtendedPrice')) as x;
```

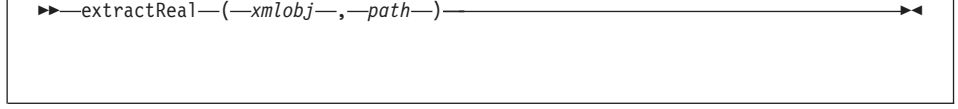
extractReal() および extractReals()

目的

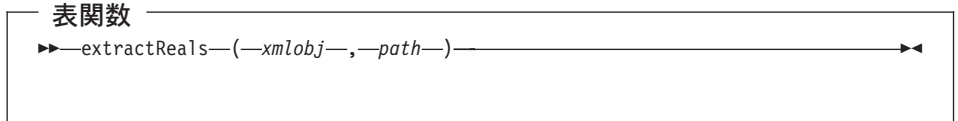
エレメント内容または属性値を XML 文書から抽出して、データを REAL タイプで戻します。

構文

スカラー関数



表関数



パラメーター

表 32. extractReal および extractReals 関数のパラメーター

パラメーター	データ・タイプ	説明
xmlobj	XMLVARCHAR、XMLFILE、または XMLCLOB	列名。
path	VARCHAR	エレメントまたは属性のロケーション・パス。

戻りタイプ

REAL

戻り列名 (表関数)

returnedReal

例

以下の例では、Tax のそれぞれの値が REAL として抽出されます。

```
SELECT * from table(db2xml.extractReals(db2xml.XMLFile
   ('c:¥dxx¥samples¥xml¥getstart.xml'), '/Order/Part/Tax')) as x;
```

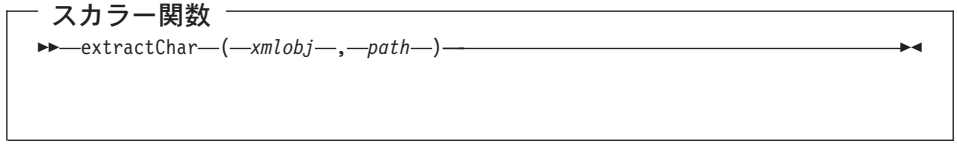
extractChar() および extractChars()

目的

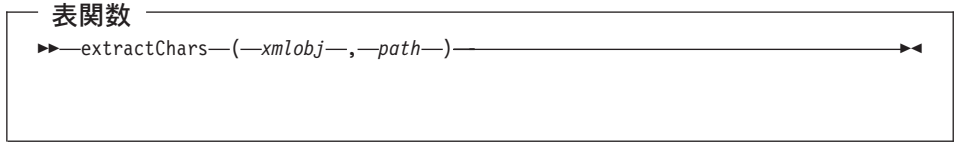
エレメント内容または属性値を XML 文書から抽出して、データを CHAR タイプで戻します。

構文

スカラー関数



表関数



パラメーター

表 33. extractChar および extractChars 関数のパラメーター

パラメーター	データ・タイプ	説明
xmlobj	XMLVARCHAR、XMLFILE、または XMLCLOB	列名。
path	VARCHAR	エレメントまたは属性のロケーション・パス。

戻りタイプ

CHAR

戻り列名 (表関数)

returnedChar

例

以下の例では、Color の値が CHAR として抽出されます。

```
SELECT * from table(db2xml.extractChars(Order,
   ('c:¥dxx¥samples¥xml¥getstart.xml'), '/Order/Part/@Color')) as x;
```

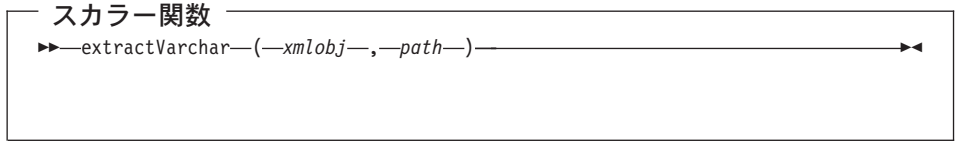
extractVarchar() および extractVarchars()

目的

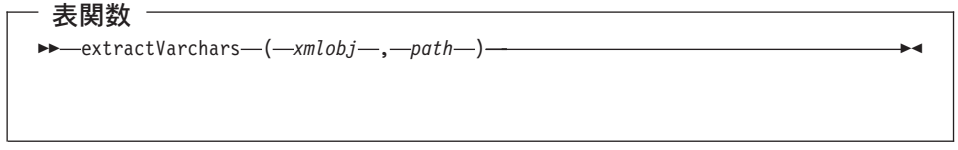
エレメント内容または属性値を XML 文書から抽出して、データを VARCHAR タイプで戻します。

構文

スカラー関数



表関数



パラメーター

表 34. extractVarchar および extractVarchars 関数のパラメーター

パラメーター	データ・タイプ	説明
xmlobj	XMLVARCHAR、XMLFILE、または XMLCLOB	列名。
path	VARCHAR	エレメントまたは属性のロケーション・パス。

戻りタイプ

VARCHAR(4K)

戻り列名 (表関数)

returnedVarchar

例

SALES_TAB 表の列 ORDER に 1000 を超える XML 文書が保管されているデータベースで、 ExtendedPrice が 2500.00 を超える品目を注文したすべてのカスタマーを検索したいと想定します。以下の SQL ステートメントでは、抽出 UDF を SELECT 文節内で使用します。

```
SELECT extractVarchar(Order, '/Order/Customer/Name') from sales_order_view
   WHERE price > 2500.00
```

UDF extractVarchar() は列 ORDER を入力、ロケーション・パス /Order/Customer/Name を選択 ID としています。 この UDF はカスタマーの名前を戻します。 WHERE 文節によって、抽出関数は ExtendedPrice が 2500.00 を超える注文だけを評価します。

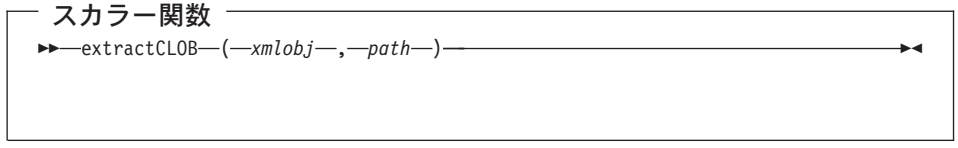
extractCLOB() および extractCLOBs()

目的

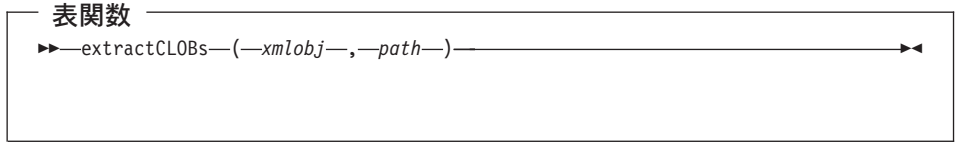
XML 文書の部分を、サブエレメントを含む、エレメントおよび属性マークアップや、エレメントおよび属性の内容とともに抽出します。この関数は、他の抽出関数とは異なります。他の抽出関数はエレメントと属性の内容だけしか戻しません。 extractClob(s) 関数は文書の部分を抽出するために使用するのに対し、 extractVarchar(s) および extractChar(s) は値だけを抽出するために使用します。

構文

スカラー関数



表関数



パラメーター

表 35. extractCLOB および extractCLOBs 関数のパラメーター

パラメーター	データ・タイプ	説明
xmlobj	XMLVARCHAR、XMLFILE、または XMLCLOB	列名。
path	VARCHAR	エレメントまたは属性のロケーション・パス。

戻りタイプ

CLOB(10K)

戻り列名 (表関数)

returnedCLOB

例

この例では、パーツ・エレメントのすべてが購入注文から抽出されます。

```
SELECT returnedCLOB as part
   from table(db2xml.extractCLOBs(db2xml.XMLFile('c:¥dxx¥samples¥xml¥getstart.xml'),
   '/Order/Part')) as x;
```

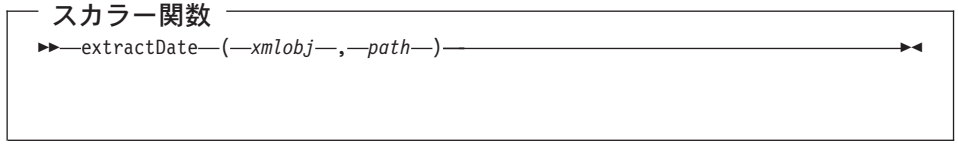
extractDate() および extractDates()

目的

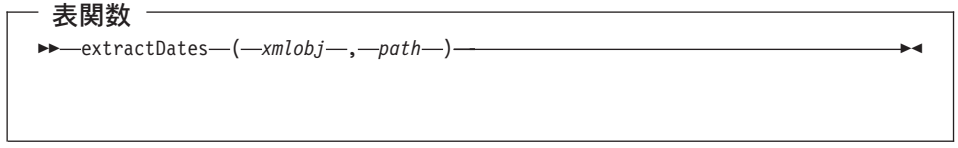
エレメント内容または属性値を XML 文書から抽出して、データを DATE タイプで戻します。

構文

スカラー関数



表関数



パラメーター

表 36. extractDate および extractDates 関数のパラメーター

パラメーター	データ・タイプ	説明
xmlobj	XMLVARCHAR、XMLFILE、または XMLCLOB	列名。
path	VARCHAR	エレメントまたは属性のロケーション・パス。

戻りタイプ

DATE

戻り列名 (表関数)

returnedDate

例

以下の例では、ShipDate の値が DATE として抽出されます。

```
SELECT * from table(db2xml.extractDates(db2xml.XMLFile
   ('c:¥dxx¥samples¥xml¥getstart.xml'), '/Order/Part/Shipment/ShipDate')) as x;
```

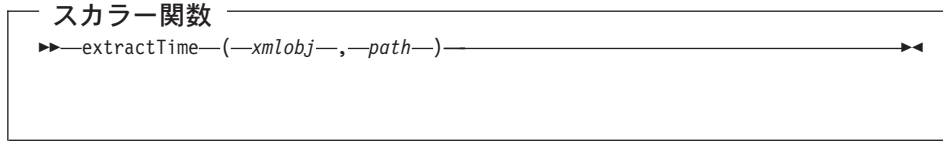
extractTime() および extractTimes()

目的

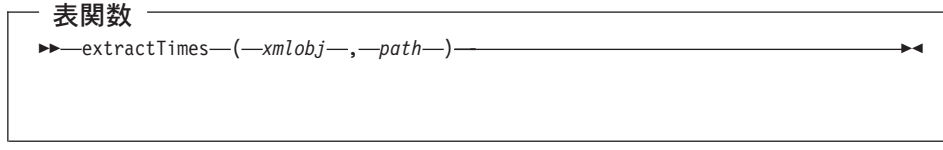
エレメント内容または属性値を XML 文書から抽出して、データを TIME タイプで戻します。

構文

スカラー関数



表関数



パラメーター

表 37. extractTime および extractTimes 関数のパラメーター

パラメーター	データ・タイプ	説明
xmlobj	XMLVARCHAR、XMLFILE、または XMLCLOB	列名。
path	VARCHAR	エレメントまたは属性のロケーション・パス。

戻りタイプ

TIME

戻り列名 (表関数)

returnedTime

例

この例では、本のサンプル・ファイルを使用します。この例では、本の値段が付けられた時刻に関して XML ファイル `book.xml` を検索し、その値を TIME として戻します。

XML ファイル:

```
<?xml version="1.0">
<DOCTYPE book SYSTEM "c:¥dxx¥samples¥book.dtd">
<book>
<chapter id="1" date="07/01/97">
<section>This is a section in Chapter One.</section>
<chapter id="2" date="01/02/1997">
<section>This is a section in Chapter Two.</section>
</chapter>
<price date="12/22/1998" time="11.12.13" timestamp="1998-12-22-11.12.13.888888">
38.281
</price>
</book>
```

抽出例:

```
CREATE TABLE t1(SaleTime TIME);
INSERT INTO t1 values(db2xml.extractTime(doc, '/book/price/@time'));
SELECT * from t1;
```

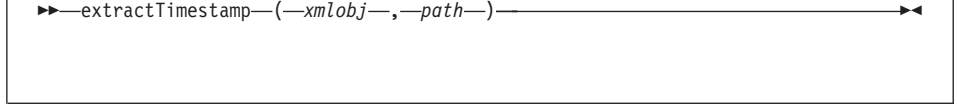
extractTimestamp() および extractTimestamps()

目的

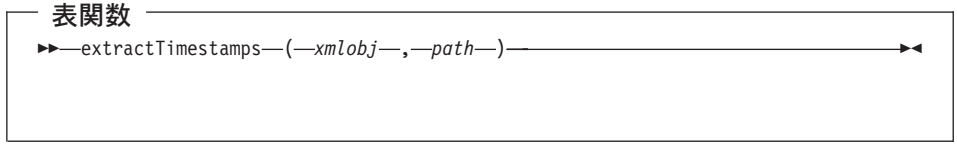
エレメント内容または属性値を XML 文書から抽出して、データを TIMESTAMP タイプで戻します。

構文

スカラー関数



表関数



パラメーター

表 38. extractTimestamp および extractTimestamps 関数のパラメーター

パラメーター	データ・タイプ	説明
xmlobj	XMLVARCHAR、XMLFILE、または XMLCLOB	列名。
path	VARCHAR	エレメントまたは属性のロケーション・パス。

戻りタイプ

TIMESTAMP

戻り列名 (表関数)

returnedTimestamp

例

この例では、本のサンプル・ファイルを使用します。この例では、それぞれの本の値段が付けられる時刻の指定に関して XML ファイル `book.xml` を検索し、その値を TIMESTAMP として戻します。

XML ファイル:

```
<?xml version="1.0">
<DOCTYPE book SYSTEM "c:¥dxx¥samples¥book.dtd">
<book>
<chapter id="1" date="07/01/97">
<section>This is a section in Chapter One.</section>
<chapter id="2" date="01/02/1997">
<section>This is a section in Chapter Two.</section>
</chapter>
<price date="12/22/1998" time="11.12.13" timestamp="1998-12-22-11.12.13.888888">
38.281
</price>
</book>
```

抽出例:

```
CREATE TABLE t1(SaleTime TIMESTAMP);
INSERT INTO t1 values(db2xml.extractTimestamp(doc, '/book/price/@timestamp'));
SELECT * from t1;
```

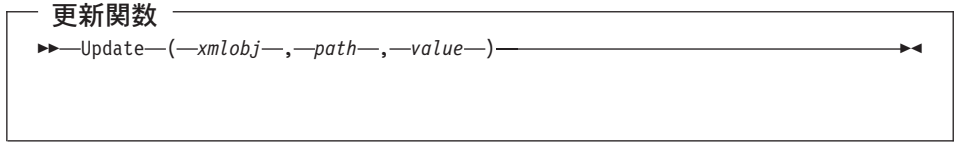
更新関数

Update() 関数は、XML 列に保管されている 1 つまたは複数の XML 文書内で指定されたエレメントまたは属性値を更新します。デフォルト・キャスト関数を使用して、SQL 基本タイプを XML UDT に変換することもできます。 これについては140ページの『XML データを更新する』で説明しています。

目的

XML UDT の列名、ロケーション・パス、および更新値のストリングを取得して、最初の入力パラメーターと同じ XML UDT を戻します。 Update() 関数によって、更新するエレメントまたは属性を指定できます。

構文



パラメーター

表 39. UDF Update のパラメーター

パラメーター	データ・タイプ	説明
xmlobj	XMLVARCHAR、XMLCLOB を LOCATOR として	列名。
path	VARCHAR	エレメントまたは属性のロケーション・パス。
value	VARCHAR	更新ストリング。

重要: Update UDF は、属性がある述部を持つロケーション・パスはサポートしますが、エレメントがあるものはサポートしないことに注意してください。たとえば、以下の述部はサポートされます。

```
'/Order/Part[@color="black "]/ExtendedPrice'
```

以下の述部はサポートされていません。

```
'/Order/Part/Shipment/[Shipdate < "11/25/00"]'
```

戻りタイプ

データ・タイプ	戻りタイプ
XMLVARCHAR	XMLVARCHAR
XMLCLOB を LOCATOR として	XMLCLOB

例

以下の例では、販売員 Sriram Srinivasan によって扱われた購入注文を更新します。

```
UPDATE sales_tab
   set order = Update(order, '/Order/Customer/Name', 'IBM')
   WHERE sales_person = 'Sriram Srinivasan'
```

この例では、/Order/Customer/Name の内容が IBM に更新されます。

使用法

Update 関数を使用して 1 つまたはそれ以上の XML 文書内の値を変更する場合、それが実際に置換するのは、XML 列内の XML 文書です。 XML 構文解析プログラムからの出力に基づいて、元の文書の一部は保存され、他の部分は消去または変更されます。以降の節では、文書が処理される方法を解説し、更新の前後で文書がどのように変更されるかの例を示します。

Update 関数が XML 文書を処理する方法

Update 関数が XML 文書を置換する場合、 XML 構文解析プログラムの出力に基づいて文書を再構築する必要があります。 220ページの表40 では、文書の部分を処理する方法を例とともに示しています。更新の前後の XML 文書の比較例については、222ページの『例』を参照してください。

表 40. Update 関数の規則

項目またはノード・タイプ	XML 文書のコード例	更新後の状況
XML 宣言	`<?xml version='1.0' encoding='utf-8' standalone='yes' >`	XML 宣言は以下のものを保存します。 • バージョン情報を保存します。 • エンコード宣言を保存して、元の文書で指定された場合に表示します。 • 独立の宣言を保存し、元の文書で指定された場合に表示します。 • 更新後に、単一引用符を使用して値が区切られます。
DOCTYPE 宣言	`<!DOCTYPE books SYSTEM "http://dtds.org/books.dtd" > <!DOCTYPE books PUBLIC "local.books.dtd" "http://dtds.org/books.dtd" > <!DOCTYPE books> -Any of <!DOCTYPE books ( S ExternalID ) ? [ internal-dtd-subset ] > -Such as <!DOCTYPE books [ <!ENTITY mydog "Spot"> ] >? [ internal-dtd-subset ] >`	文書タイプ宣言は以下のものを保存します。 • ルート・エレメント名をサポートします。 • 共通およびシステム `ExternalID` を保存し、元の文書で指定された場合に表示します。 • 内部 DTD サブセットは保存しません。エンティティーが置換され、属性のデフォルトが処理されて、出力文書に表示されます。 • 更新後には、共通およびシステム URI 値は二重引用符で区切られています。 • 現行の XML4c 構文解析プログラムは、`ExternalID` または内部 DTD サブセットを含まない XML 宣言は報告しません。この場合は、更新後に DOCTYPE 宣言が欠落します。

表 40. Update 関数の規則 (続き)

項目またはノード・タイプ	XML 文書のコード例	更新後の状況
処理命令	`<?xml-stylesheet title="compact" href="datatypes1.xsl" type="text/xsl"?>`	処理命令は保存されます。
コメント	`<!-- comment -->`	ルート・エレメントの内部のコメントは保存されます。 ルート・エレメントの外部のコメントは廃棄されます。

エレメント	`<books>`
	content
	`</books>`
		エレメントは保存されます。

属性	`id='1' date='01/02/1997"`	エレメントの属性は保存されます。 • 更新後には、値は二重引用符で区切られています。 • 属性の内部のデータは拡張されます。 • エンティティーは置換されます。
テキスト・ノード	`This chapter is about my dog &mydoc;.` `This chapter is about my dog Spot.`	テキスト・ノード (エレメントの内容) は保存されます。 • テキスト・ノードの内部のデータは拡張されます。 • エンティティーは置換されます。

複数オカレンス

Update() UDF にロケーション・パスが備えられている場合、一致するパスがあるすべてのエレメントまたは属性の内容は、提供された値で更新されます。これは、文書に複数出現するロケーション・パスがある場合、 Update 関数は、既存の値を *value* パラメーターで提供された値で置換するということです。

path パラメーターに述部を指定して、明確なロケーション・パスを提供することによって、意図しない更新を防止することができます。 Update UDF は、属性がある述部を持つロケーション・パスをサポートしますが、エレメントがあ

るものはサポートしないことに注意してください。詳細については、218ページの『パラメーター』を参照してください。

例

以下の例は、更新の前後の XML 文書のインスタンスを示しています。

例 1:

更新前 :

```
<?xml version='1.0'   encoding='utf-8'    standalone="yes"?>
<!DOCTYPE book PUBLIC "public.dtd" "system.dtd">
<?pitarget option1='value1' option2='value2'?>
<!-- comment -->
<book>
  <chapter id="1" date='07/01/1997'>
       <!-- first section -->
        <section>This is a section in Chapter One.</section>
  </chapter>
  <chapter id="2" date="01/02/1997">
        <section>This is a section in Chapter Two.</section>
        <footnote>A footnote in Chapter Two is here.</footnote>
  </chapter>
  <price date="12/22/1998" time="11.12.13"
       timestamp="1998-12-22-11.12.13.888888">38.281</price>
</book>
```

- XML 宣言に空白文字を含んでいます。
- 処理命令を指定します。
- ルート・ノードの外側にコメントがあります。
- PUBLIC ExternalID を指定します。
- ルート・ノードの内側にコメントがあります。

更新後 :

```
<?xml version='1.0' encoding='utf-8' standalone='yes'?>
<!DOCTYPE book PUBLIC "public.dtd" "system.dtd">
<?pitarget option1='value1' option2='value2'?><book>
  <chapter id="1" date="07/01/1997">
       <!-- first section -->
        <section>This is a section in Chapter One.</section>
  </chapter>
  <chapter id="2" date="01/02/1997">
        <section>This is a section in Chapter Two.</section>
        <footnote>A footnote in Chapter Two is here.</footnote>
  </chapter>
  <price date="12/22/1998" time="11.12.13"
       timestamp="1998-12-22-11.12.13.888888">60.02</price>
</book>
```

- マークアップの内側の空白は除去されています。
- 処理命令は保存されています。
- ルート・ノードの外側のコメントは保存されていません。
- PUBLIC ExternalID は保存されています。
- ルート・ノードの内側のコメントは保存されています。
- 変更された値は <price> エレメントの値です。

例 2:

更新前 :

```
<?xml version='1.0'?>
<!DOCTYPE book>
<!-- comment -->
<book>
  ...
</book>
```

ExternalID または内部 DTD サブセットのない DOCTYPE 宣言が含まれています。サポートされていません。

更新後 :

```
<?xml version='1.0'?>
<book>
  ...
</book>
```

DOCTYPE 宣言は、XML 構文解析プログラムによって報告されず、保存されません。

例 3:

更新前 :

```
<?xml version='1.0'?>
<!DOCTYPE book [ <!ENTITY myDog "Spot"> ]>
<!-- comment -->
<book>
  <chapter id="1" date='07/01/1997'>
       <!-- first section -->
       <section>This is a section in Chapter
         One about my dog &;myDog;.</section>
       ...
  </chapter>
  ...
</book>
```

- マークアップに空白文字を含んでいます。
- 内部 DTD サブセットを指定します。
- テキスト・ノードにエンティティーを指定します。

更新後 :

```
<?xml version='1.0'?>
<!DOCTYPE book>
<book>
  <chapter id="1" date="07/01/1997">
       <!-- first section -->
       <section>This is a section in Chapter
         One about my dog Spot.</section>
          ...
  </chapter>
  ...
</book>
```

- マークアップ内の空白は除去されています。
- 内部 DTD サブセットは保存されていません。
- テキスト・ノード内のエンティティーは解決され、置換されています。

第10章 XML エクステンダーのストアード・プロシージャー

XML エクステンダーは、XML 列やコレクションの管理に役立つストアード・プロシージャーを提供します。これらのストアード・プロシージャーは、DB2 クライアントから呼び出すことができます。 クライアント・インターフェースは、SQL、ODBC、または JDBC に組み込むことができます。 ストアード・プロシージャーの呼び出し方法について、詳しくは *DB2 UDB 管理の手引き* のストアード・プロシージャーに関するセクションを参照してください。

ストアード・プロシージャーは、db2xml を使用します。これは XML エクステンダーのスキーマ名です。

XML エクステンダーのストアード・プロシージャーには、次の 3 つのタイプがあります。

- 228ページの『管理ストアード・プロシージャー』は、管理用タスクの実行を援助します。
- 236ページの『合成ストアード・プロシージャー』は、既存のデータベース表のデータを使用して XML 文書を生成します。
- 246ページの『分解ストアード・プロシージャー』は、受け取った XML 文書を分解または断片化して、新規または既存のデータベース表にデータを保管します。

XML コレクションのストアード・プロシージャーによって使用されるパラメーター制限は、311ページの『付録D. XML エクステンダーの制限』で説明しています。

組み込みファイルの指定

ストアード・プロシージャーを呼び出すプログラムには、必ず XML エクステンダーの外部ヘッダー・ファイルを組み込んでください。ヘッダー・ファイルは、 `DXX_INSTALL/include` ディレクトリーにあります。 `DXX_INSTALL` は XML エクステンダーのインストール先ディレクトリーです。正確な場所はオペレーティング・システムによって異なります。ヘッダー・ファイルは次のとおりです。

dxx.h XML エクステンダーの定義する定数およびデータのタイプ

dxxrc.h XML エクステンダーの戻りコード

これらのヘッダー・ファイルを組み込む構文は次のとおりです。

```
#include "dxx.h"
#include "dxxrc.h"
```

組み込みファイルのパスが、 MAKE ファイル内でコンパイル・オプションとともに指定されていることを確認してください。

XML エクステンダーのストアード・プロシージャーの呼び出し

通常は、以下の構文を使って XML エクステンダーを呼び出します。

```
CALL db2xml.function_entry_point
```

ここで、

db2xml
XML エクステンダー ストアード・プロシージャーのライブラリーを指定します。 UNIX オペレーティング・システム上では、ライブラリーは sqllib/function ディレクトリーに保管されています。 Windows オペレーティング・システム上では、これは *DXX_INSTALL*/bin ディレクトリーに保管されています。

function_entry_point
ストアード・プロシージャーに渡される引き数を指定します。

CALL ステートメントの中で、ストアード・プロシージャーに渡される引き数は定数や式ではなく、ホスト変数でなければなりません。ホスト変数にはヌル標識を指定できます。例として、DXX_INSTALL/samples/c および DXX_INSTALL/samples/cli ディレクトリー内のストアード・プロシージャーを呼び出すサンプル、および本書の以下のセクションを参照してください。 43ページの『XML 文書を構成する』および 151ページの『第6章 XML コレクション・データの管理』。

DXX_INSTALL/samples/c ディレクトリーには、組み込み SQL を使用して XML コレクションのストアード・プロシージャーを呼び出すため SQC コード・ファイルが入っています。 DXX_INSTALL/samples/cli ディレクトリー内のサンプル・ファイルは、コール・レベル・インターフェース (CLI) を使ってストアード・プロシージャーを呼び出す方法を示します。

CLOB 制限の引き上げ

ストアード・プロシージャーに渡される場合の CLOB パラメーターのデフォルト制限は 1 MB です。以下のステップを実行して、この制限を引き上げることができます。

1. ストアード・プロシージャーを除去します。たとえば次のようにします。

```
db2   "drop procedure db2xml.dxxShredXML"
```

2. 新しいプロシージャーを、引き上げた CLOB 制限を指定して作成します。たとえば次のようにします。

```
db2   "create procedure db2xml.dxxShredXML(in     dadBuf      clob(100K),
                                            in     XMLObj      clob(10M),
                                            out    returnCode  integer,
                                            out    returnMsg   varchar(1024)
                                            )
       external name 'db2xml.dxxShredXML'
       language C
       parameter style DB2DARI
       not deterministic
       fenced
       null call;
```

作業を始める前に

データベースを、XML エクステンダー ストアード・プロシージャーおよび DB2 CLI バインド・ファイルとバインドします。ファイルをバインドするには、サンプル・コマンド・ファイル getstart_prep.cmd を使用することができます。このコマンド・ファイルは DXX_INSTALL/samples/cmd ディレクトリーにあります。

1. データベースに接続します。たとえば次のようにします。

```
db2 "connect to SALES_DB"
```

2. DXX_INSTALL/bnd ディレクトリーに移って、 XML エクステンダーをデータベースにバインドします。

```
db2 "bind @dxxbind.lst"
```

3. sqllib/bnd ディレクトリーに移って、 CLI をデータベースにバインドします。

```
db2 "bind @db2cli.lst"
```

4. 接続を終了します。

```
db2 "terminate"
```

管理ストアード・プロシージャー

これらのストアード・プロシージャーは、 XML 列やコレクションを使用可能または使用不可にする場合などの管理タスクに使用します。 これらを呼び出すには、XML エクステンダー管理ウィザードおよび管理コマンド **dxxadm** を使用します。

dxxEnableDB()

目的

データベースを使用可能にします。データベースを使用可能にすると、XML エクステンダーは以下のオブジェクトを作成します。

- XML エクステンダー・ユーザー定義タイプ (UDT)
- XML エクステンダー・ユーザー定義関数 (UDF)
- XML エクステンダー DTD 参照表 (DTD_REF)。これは DTD およびそれぞれの DTD に関する情報を保管します。 DTD_REF 表についての詳細は、251ページの『DTD 参照表』を参照してください。
- XML エクステンダー使用状況表 (XML_USAGE)。これは XML に使用できるそれぞれの列およびコレクションに関する一般情報を保管します。 XML_USAGE 表についての詳細は、252ページの『XML 使用状況表』を参照してください。

```
dxxEnableDB(char(dbName) dbName,       /* input */
            long         returnCode,   /* output */
            varchar(1024) returnMsg)   /* output */
```

パラメーター

表 41. dxxEnableDB() パラメーター

パラメーター	説明	IN/OUT パラメーター
dbName	データベース名	IN
returnCode	ストアード・プロシージャーからの戻りコード	OUT
returnMsg	エラー発生時に戻されるメッセージ・テキスト	OUT

dxxDisableDB()

目的

データベースを使用不可にします。 XML エクステンダーがデータベースを使用不可にすると、以下のオブジェクトを除去します。

- XML エクステンダー・ユーザー定義タイプ (UDT)
- XML エクステンダー・ユーザー定義関数 (UDF)
- XML エクステンダー DTD 参照表 (DTD_REF)。これは DTD およびそれぞれの DTD に関する情報を保管します。 DTD_REF 表についての詳細は、251ページの『DTD 参照表』を参照してください。
- XML エクステンダー使用状況表 (XML_USAGE)。これは XML に使用できるそれぞれの列およびコレクションに関する一般情報を保管します。 XML_USAGE 表についての詳細は、252ページの『XML 使用状況表』を参照してください。

重要: データベースを使用不可にする前に、すべての XML 列を使用不可にする必要があります。 XML エクステンダーは、XML に使用できる列またはコレクションを含んでいるデータベースを使用不可にできません。

```
dxxDisableDB(char(dbName)     dbName,       /* input */
                 long           returnCode,   /* output */
                 varchar(1024) returnMsg)    /* output */
```

パラメーター

表 42. dxxDisableDB() パラメーター

パラメーター	説明	IN/OUT パラメーター
dbName	データベース名	IN
returnCode	ストアード・プロシージャーからの戻りコード	OUT
returnMsg	エラー発生時に戻されるメッセージ・テキスト	OUT

dxxEnableColumn()

目的

XML 列を使用可能にします。 XML エクステンダーは列を使用可能にする時に以下のタスクを行います。

- XML 表に基本キーがあるかどうかを判別します。ない場合は、XML エクステンダーが XML 表を更新して DXXROOT_ID という列を追加します。
- DAD ファイルの中で指定されたサイド表を作成します。この表には XML 表内の各行の固有な ID を含む列があります。 この列はユーザーが指定する *root_id* であるか、または XML エクステンダーが命名する DXXROOT_ID です。
- XML 表およびそのサイド表のデフォルト視点を作成します。オプションで、ユーザーが名前を指定できます。

```
dxxEnableColumn(char(dbName) dbName,      /* input */
                char(tbName) tbName,      /* input */
                char(colName) colName,     /* input */
                CLOB(100K)     DAD,         /* input */
                char(tablespace) tablespace,  /* input */
                char(defaultView) defaultView, /* input */
                char(rootID) rootID,      /* input */
                long           returnCode,  /* output */
                varchar(1024)  returnMsg)   /* output */
```

パラメーター

表 43. dxxEnableColumn() パラメーター

パラメーター	説明	IN/OUT パラメーター
dbName	データベース名	IN
tbName	XML 列を含む表の名前	IN
colName	XML 列の名前	IN
DAD	DAD ファイルを含む CLOB	IN
tablespace	デフォルトの表スペース以外のサイド表を含む表スペース。 これを指定しない場合、デフォルトの表スペースが使用されます。	IN
defaultView	アプリケーション表とサイド表を結合するデフォルト視点の名前	IN
rootID	アプリケーション表における、サイド表の *root_id* として使われるただ 1 つの基本キーの名前	IN

表 43. *dxxEnableColumn()* パラメーター *(続き)*

パラメーター	説明	IN/OUT パラメーター
returnCode	ストアード・プロシージャーからの戻りコード	OUT
returnMsg	エラー発生時に戻されるメッセージ・テキスト	OUT

dxxDisableColumn()

目的

XML に使用できる列を使用不可にします。 XML 列が使用不可になると、XML データ・タイプを保管できなくなります。

```
dxxDisableColumn(char(dbName) dbName,      /* input */
                 char(tbName) tbName,      /* input */
                 char(colName) colName,     /* input */
                 long           returnCode,  /* output */
                 varchar(1024)  returnMsg)   /* output */
```

パラメーター

表 44. dxxDisableColumn() パラメーター

パラメーター	説明	IN/OUT パラメーター
dbName	データベース名	IN
tbName	XML 列を含む表の名前	IN
colName	XML 列の名前	IN
returnCode	ストアード・プロシージャーからの戻りコード	OUT
returnMsg	エラー発生時に戻されるメッセージ・テキスト	OUT

dxxEnableCollection()

目的

アプリケーション表に関連した XML コレクションを使用可能にします。

```
dxxEnableCollection(char(dbName) dbName,      /* input */
                    char(colName) colName,     /* input */
                    CLOB(100K)     DAD,         /* input */
                    char(tablespace) tablespace,  /* input */
                    long           returnCode,  /* output */
                    varchar(1024)  returnMsg)   /* output */
```

パラメーター

表 45. dxxEnableCollection() パラメーター

パラメーター	説明	IN/OUT パラメーター
dbName	データベース名	IN
colName	XML コレクションの名前	IN
DAD	DAD ファイルを含む CLOB	IN
tablespace	デフォルトの表スペース以外のサイド表を含む表スペース。 これを指定しない場合、デフォルトの表スペースが使用されます。	IN
returnCode	ストアード・プロシージャーからの戻りコード	OUT
returnMsg	エラー発生時に戻されるメッセージ・テキスト	OUT

dxxDisableCollection()

目的

XML に使用できるコレクションを使用不可にし、表および列をコレクションの一部として識別するマーカーを除去します。

```
dxxDisableCollection(char(dbName) dbName,      /* input */
                     char(colName) colName,     /* input */
                     long          returnCode,  /* output */
                     varchar(1024)  returnMsg)   /* output */
```

パラメーター

表 46. dxxDisableCollection() パラメーター

パラメーター	説明	IN/OUT パラメーター
dbName	データベース名	IN
colName	XML コレクションの名前	IN
returnCode	ストアード・プロシージャーからの戻りコード	OUT
returnMsg	エラー発生時に戻されるメッセージ・テキスト	OUT

合成ストアード・プロシージャー

合成ストアード・プロシージャー dxxGenXML() および dxxRetrieveXML() を使用して、既存のデータベース表のデータを使って XML 文書を生成します。ストアード・プロシージャー dxxGenXML() は入力として DAD ファイルを取ります。使用可能な XML コレクションは必要ありません。ストアード・プロシージャー dxxRetrieveXML() は入力として、使用可能な XML コレクション名を取ります。

dxxGenXML()

目的

(DAD ファイルの <Xcollection> で指定された) XML コレクション表に保管されているデータを使用して XML 文書を作成し、それぞれの XML 文書を行として結果表の中に挿入します。 また、結果表でカーソルを開いて結果セットを取り出すこともできます。

dxxGenXML() では、結果表の中に生成する行の最大数をユーザーが任意に指定することができます。これによって、試行プロセス中にアプリケーションが結果を待つ時間を短縮できます。このストアード・プロシージャーは、表内の実際の行番号とエラー情報 (エラー・コードおよびエラー・メッセージ) を戻します。

動的照会をサポートするために、 dxxGenXML() は入力パラメーター *override* を取ります。入力 *overrideType* に基づいて、アプリケーションは DAD ファイル内の SQL マッピングの SQL_stmt、または RDB_node 内の RDB_node マッピング条件をオーバーライドできます。 入力パラメーター *overrideType* を使用して、 *override* のタイプを明示します。 *override* パラメーターについての詳細は、156ページの『DAD ファイル内の値を動的にオーバーライドする』を参照してください。

```
dxxGenXML(CLOB(100K)     DAD,             /* input */
          char(resultTabName) resultTabName,   /* input */
          integer        overrideType     /* input */
          varchar(1024)  override,        /* input */
          integer        maxRows,         /* input */
          integer        numRows,         /* output */
          long           returnCode,      /* output */
          varchar(1024)  returnMsg)       /* output */
```

パラメーター

表 47. dxxGenXML() パラメーター

パラメーター	説明	IN/OUT パラメーター
DAD	DAD ファイルを含む CLOB	IN
resultTabName	結果表の名前。これは呼び出しの前にすでに存在しなければなりません。この表には、XMLVARCHAR または XMLCLOB のいずれかのタイプの列を 1 つだけ含めます。	IN

表 47. *dxxGenXML()* パラメーター *(続き)*

パラメーター	説明	IN/OUT パラメーター
overrideType	下記の *override* パラメーターのタイプを示すフラグ • **NO_OVERRIDE**: オーバーライドしない • **SQL_OVERRIDE**: SQL_stmt によるオーバーライド • **XML_OVERRIDE**: XPath ベースの条件によるオーバーライド	IN
override	DAD ファイル内の条件をオーバーライドします。入力値は *overrideType* に応じて次のとおりです。 • **NO_OVERRIDE**: ヌル・ストリング。 • **SQL_OVERRIDE**: 有効な SQL ステートメント。この *overrideType* を使用するには、 DAD ファイル内で SQL マッピングを使用する必要があります。入力 SQL ステートメントは DAD ファイルの SQL_stmt をオーバーライドします。 • **XML_OVERRIDE**: 1 つまたは複数の式を含むストリング。複数の式は二重引用符で囲んで AND で分離します。この *overrideType* を使用するには、 DAD ファイル内で RDB_node マッピングを使用する必要があります。	IN
maxRows	結果表の行の最大数	IN

表 47. *dxxGenXML()* パラメーター *(続き)*

パラメーター	説明	IN/OUT パラメーター
numRows	結果表に実際に生成された行の数	OUT
returnCode	ストアード・プロシージャーからの戻りコード	OUT
returnMsg	エラー発生時に戻されるメッセージ・テキスト	OUT

例

以下の例では、XML_ORDER_TAB という名前の結果表が作成されることと、XMLVARCHAR タイプの 1 つの列が表に含まれることを想定します。

```
#include "dxx.h"
#include "dxxrc.h"

              EXEC SQL INCLUDE SQLCA;
              EXEC SQL BEGIN DECLARE SECTION;
              SQL TYPE is CLOB(100K)  dad;              /* DAD */
              SQL TYPE is CLOB_FILE   dadFile;          /* dad file */
              char                    result_tab[32];   /* name of the result table */
              char                    override[2];      /* override, will set to NULL*/
              short                   overrideType;     /* defined in dxx.h */
              short                   max_row;          /* maximum number of rows */
              short                   num_row;          /* actual number of rows */
              long                    returnCode;       /* return error code */
              char                    returnMsg[1024];  /* error message text */
              short                   dad_ind;
              short                   rtab_ind;
              short                   ovtype_ind;
              short                   ov_inde;
              short                   maxrow_ind;
              short                   numrow_ind;
              short                   returnCode_ind;
              short                   returnMsg_ind;
              EXEC SQL END DECLARE SECTION;

              /* create table */
              EXEC CREATE TABLE xml_order_tab (xmlorder XMLVarchar);

              /* read data from a file to a CLOB */
              strcpy(dadfile.name,"e:¥dxx¥dad¥litem3.dad");
              dadfile.name_length = strlen("e:¥dxx¥dad¥litem3.dad");
              dadfile.file_options = SQL_FILE_READ;
              EXEC SQL VALUES (:dadfile) INTO :dad;
              strcpy(result_tab,"xml_order_tab");
              override[0] = '¥0';
              overrideType = NO_OVERRIDE;
              max_row = 500;
              num_row = 0;
              returnCode = 0;
              msg_txt[0] = '¥0';
              collection_ind = 0;
              dad_ind = 0;
              rtab_ind = 0;
              ov_ind = -1;
              ovtype_ind = 0;
              maxrow_ind = 0;
              numrow_ind = -1;
```

```
returnCode_ind = -1;
returnMsg_ind = -1;

/* Call the store procedure */
EXEC SQL CALL dxxGenXML(:dad:dad_ind;
       :result_tab:rtab_ind,
       :overrideType:ovtype_ind,:override:ov_ind,
       :max_row:maxrow_ind,:num_row:numrow_ind,
       :returnCode:returnCode_ind,:returnMsg:returnMsg_ind);
```

dxxRetrieveXML()

目的

合成と分解の両方で同じ DAD ファイルを使用できるようにします。また、分解された XML 文書を、ストアード・プロシージャー dxxRetrieveXML() を使って取り出すこともできます。 dxxRetrieveXML() は入力として DAD ファイルを含むバッファー、作成される結果表の名前、および戻される行の最大数を取ります。 また結果セット (結果表)、結果セット内の実際の行数、およびエラー・コードとメッセージ・テキストを戻します。

動的照会をサポートするために、 dxxRetrieveXML() は入力パラメーター *override* を取ります。入力 *overrideType* に基づいて、アプリケーションは DAD ファイル内の SQL マッピングの SQL_stmt、または RDB_node 内の RDB_node マッピング条件をオーバーライドできます。 入力パラメーター *overrideType* を使用して、 *override* のタイプを明示します。 *override* パラメーターについての詳細は、156ページの『DAD ファイル内の値を動的にオーバーライドする』を参照してください。

dxxRetrieveXML() を使用するための DAD ファイルの要件は、 dxxGenXML() を使用する場合の要件と同じです。 唯一の違いとして、dxxRetrieveXML() の入力パラメーターは DAD ではなく、使用可能な XML コレクションの名前です。

```
dxxRetrieveXML(char(collectionName) collectionName,   /* input */
               char(resultTabName) resultTabName,    /* input */
               integer        overrideType,     /* input */
               varchar(1024)  override,         /* input */
               integer        maxRows,          /* input */
               integer        numRows,          /* output */
               long           returnCode,       /* output */
               varchar(1024)  returnMsg)        /* output */
```

パラメーター

表 48. dxxRetrieveXML() パラメーター

パラメーター	説明	IN/OUT パラメーター
collectionName	使用可能な XML コレクションの名前	IN

表 48. *dxxRetrieveXML()* パラメーター *(続き)*

パラメーター	説明	IN/OUT パラメーター
resultTabName	結果表の名前。これは呼び出しの前にすでに存在しなければなりません。この表には、XMLVARCHAR または XMLCLOB のいずれかのタイプの列を 1 つだけ含めます。	IN
overrideType	下記の *override* パラメーターのタイプを示すフラグ • **NO_OVERRIDE**: オーバーライドしない • **SQL_OVERRIDE**: SQL_stmt によるオーバーライド • **XML_OVERRIDE**: XPath ベースの条件によるオーバーライド	IN

表 48. *dxxRetrieveXML()* パラメーター *(続き)*

パラメーター	説明	IN/OUT パラメーター
override	DAD ファイル内の条件をオーバーライドします。入力値は *overrideType* に応じて次のとおりです。 • **NO_OVERRIDE**: ヌル・ストリング。 • **SQL_OVERRIDE**: 有効な SQL ステートメント。この *overrideType* を使用するには、 DAD ファイル内で SQL マッピングを使用する必要があります。入力 SQL ステートメントは DAD ファイルの SQL_stmt をオーバーライドします。 • **XML_OVERRIDE**: 1 つまたは複数の式を含むストリング。複数の式は二重引用符で囲んで AND で分離します。この *overrideType* を使用するには、 DAD ファイル内で RDB_node マッピングを使用する必要があります。	IN
maxRows	結果表の行の最大数	IN
numRows	結果表に実際に生成された行の数	OUT
returnCode	ストアード・プロシージャーからの戻りコード	OUT
returnMsg	エラー発生時に戻されるメッセージ・テキスト	OUT

例

以下は dxxRetrieveXML() の呼び出しの例です。この例では XML_ORDER_TAB という名前の結果表が作成され、 XMLVARCHAR タイプの 1 つの列がその表に含まれます。

```
#include "dxx.h"
#include "dxxrc.h"

            EXEC SQL INCLUDE SQLCA;
            EXEC SQL BEGIN DECLARE SECTION;
              char    collection[32];    /* dad buffer */
              char    result_tab[32];    /* name of the result table */
              char    override[2];       /* override, will set to NULL*/
              short   overrideType;      /* defined in dxx.h */
              short   max_row;           /* maximum number of rows */
              short   num_row;           /* actual number of rows */
              long    returnCode;        /* return error code */
              char    returnMsg[1024];   /* error message text */
              short   dadbuf_ind;
              short   rtab_ind;
              short   ovtype_ind;
              short   ov_inde;
              short   maxrow_ind;
              short   numrow_ind;
              short   returnCode_ind;
              short   returnMsg_ind;

              EXEC SQL END DECLARE SECTION;

              /* create table */
              EXEC SQL CREATE TABLE xml_order_tab (xmlorder XMLVarchar);

              /* initialize host variable and indicators */
              strcpy(collection,"sales_ord");
              strcpy(result_tab,"xml_order_tab");
              override[0] = '¥0';
              overrideType = NO_OVERRIDE;
              max_row = 500;
              num_row = 0;
              returnCode = 0;
              msg_txt[0] = '¥0';
              collection_ind = 0;
              rtab_ind = 0;
              ov_ind = -1;
              ovtype_ind = 0;
              maxrow_ind = 0;
              numrow_ind = -1;
              returnCode_ind = -1;
              returnMsg_ind = -1;

              /* Call the store procedure */
              EXEC SQL CALL dxxRetrieve(:collection:collection_ind;
                     :result_tab:rtab_ind,
```

```
:overrideType:ovtype_ind,:override:ov_ind,
:max_row:maxrow_ind,:num_row:numrow_ind,
:returnCode:returnCode_ind,:returnMsg:returnMsg_ind);
```

分解ストアード・プロシージャー

分解ストアード・プロシージャー dxxInsertXML() および dxxShredXML() は、受け取った XML 文書を分解または断片化して、新規または既存のデータベース表にデータを保管します。ストアード・プロシージャー dxxInsertXML() は入力として、使用可能な XML コレクション名を取ります。ストアード・プロシージャー dxxShredXML() は入力として DAD ファイルを取ります。使用可能な XML コレクションは必要ありません。

dxxShredXML()

目的

ストアード・プロシージャー dxxShredXML() は、 dxxGenXML() に対応するストアード・プロシージャーです。 dxxShredXML() を実行するには、DAD ファイルで指定されたすべての表が存在しなければならず、 DAD で指定されたすべての列およびデータ・タイプが既存の表と適合していなければなりません。ストアード・プロシージャー dxxShredXML() では、結合する表の間で基本キーと外部キーの関連は必要ありません (この関連は、コレクションの使用可能化プロセス中に XML エクステンダーが作成するものです)。ただし、root element_node の RDB_node 内で指定される結合条件の列が、すでに表に存在していなければなりません。

```
dxxShredXML(CLOB(100K)    DAD,           /* input */
            CLOB(1M)      xmlobj,        /* input */
            long          returnCode,    /* output */
            varchar(1024) returnMsg)     /* output */
```

パラメーター

表 49. dxxShredXML() パラメーター

パラメーター	説明	IN/OUT パラメーター
DAD	DAD ファイルを含む CLOB	IN
xmlobj	XMLCLOB タイプの XML 文書オブジェクト	IN
returnCode	ストアード・プロシージャーからの戻りコード	OUT
returnMsg	エラー発生時に戻されるメッセージ・テキスト	OUT

例

以下は dxxShredXML() の呼び出しの例です。

```
#include "dxx.h"
#include "dxxrc.h"

EXEC SQL INCLUDE SQLCA;
            EXEC SQL BEGIN DECLARE SECTION;
              SQL TYPE is CLOB dad;           /* DAD*/
              SQL TYPE is CLOB_FILE  dadFile; /* DAD file*/
              SQL TYPE is CLOB xmlDoc;        /* input XML document */
              SQL TYPE is CLOB_FILE xmlFile;  /* input XMLfile */
              long        returnCode;         /* error code */
              char        returnMsg[1024];    /* error message text */
              short       dad_ind;
              short       xmlDoc_ind;
              short       returnCode_ind;
              short       returnMsg_ind;
              EXEC SQL END DECLARE SECTION;
```

```
/* initialize host variable and indicators */
strcpy(dadFile.name,"c:¥dxx¥samples¥dad¥getstart_xcollection.dad");
dadFile.name_length=strlen("c:¥dxx¥samples¥dad¥getstart_xcollection.dad");
dadFile.file_option=SQL_FILE_READ;
strcpy(xmlFile.name,"c:¥dxx¥samples¥cmd¥getstart.xml");
xmlFile.name_length=strlen("c:¥dxx¥samples¥cmd¥getstart.xml");
xmlFile.file_option=SQL_FILE_READ;
SQL EXEC VALUES (:dadFile) INTO :dad;
SQL EXEC VALUES (:xmlFile) INTO :xmlDoc;
returnCode = 0;
returnMsg[0] = '¥0';
dad_ind = 0;
xmlDoc_ind = 0;
returnCode_ind = -1;
returnMsg_ind = -1;

/* Call the store procedure */
EXEC SQL CALL db2xml.dxxShredXML(:dad:dad_ind;
                :xmlDoc:xmlDoc_ind,
                :returnCode:returnCode_ind,:returnMsg:returnMsg_ind);
```

dxxInsertXML()

目的

これは 2 つの入力パラメーターとして、使用可能な XML コレクションの名前、および分解される XML 文書を取り、2 つの出力パラメーターとして、戻りコード、および戻りメッセージを戻します。

```
dxxInsertXML(char(collectionName) collectionName,   /* input */
             CLOB(1M)       xmlobj,           /* input */
             long           returnCode,       /* output */
             varchar(1024)  returnMsg)        /* output */
```

パラメーター

表 50. dxxInsertXML() パラメーター

パラメーター	説明	IN/OUT パラメーター
collectionName	使用可能な XML コレクションの名前	IN
xmlobj	CLOB タイプの XML 文書オブジェクト	IN
returnCode	ストアード・プロシージャーからの戻りコード	OUT
returnMsg	エラー発生時に戻されるメッセージ・テキスト	OUT

例

以下の例では、 dxxInsertXML() 呼び出しによって入力 XML 文書 e:¥xml¥order1.xml を分解し、この XML を使用可能にした DAD ファイル内のマッピングに従って、データを SALES_ORDER コレクション表に挿入します。

```
#include "dxx.h"
#include "dxxrc.h"

EXEC SQL INCLUDE SQLCA;
           EXEC SQL BEGIN DECLARE SECTION;
             char                 collection[64];  /* name of an XML collection */
             SQL TYPE is CLOB_FILE xmlDoc;         /* input XML document */
             long                 returnCode;      /* error code */
             char                 returnMsg[1024]; /* error message text */
             short                collection_ind;
             short                xmlDoc_ind;
             short                returnCode_ind;
             short                returnMsg_ind;
             EXEC SQL END DECLARE SECTION;

             /* initialize host variable and indicators */
             strcpy(collection,"sales_ord")
             strcpy(xmlobj.name,"c:¥dxx¥samples¥cmd¥getstart.xml");
             xmlobj.name_length=strlen("c:¥dxx¥samples¥cmd¥getstart.xml");
```

```
xmlobj.file_option=SQL_FILE_READ;
returnCode = 0;
returnMsg[0] = '¥0';
collection_ind = 0;
xmlobj_ind = 0;
returnCode_ind = -1;
returnMsg_ind = -1;

/* Call the store procedure */
EXEC SQL CALL db2xml.dxxInsertXML(:collection:collection_ind;
                :xmlobj:xmlobj_ind,
                :returnCode:returnCode_ind,:returnMsg:returnMsg_ind);
```

第11章 管理サポート表

データベースが使用可能になると、DTD 参照表 DTD_REF および XML_USAGE 表が作成されます。 DTD_REF 表は、すべての DTD に関する情報を含みます。 XML_USAGE 表は、XML を使用できる各列に関する一般情報を保管します。

サポート表にリストしているパラメーター制限は、311ページの『付録D. XML エクステンダーの制限』で説明しています。

DTD 参照表

XML エクステンダーには XML DTD リポジトリーとしての機能もあります。データベースで XML が使用可能になると、DTD 参照表 DTD_REF が作成されます。 この表の各行は、追加のメタデータ情報のある DTD を表します。ユーザーはこの表にアクセスして独自の DTD を挿入することができます。DTD_REF 表内の DTD を使用して XML 文書の妥当性検査を行い、アプリケーションにおける DAD ファイルの定義を支援します。 この表には db2xml というスキーマ名があります。 DTD_REF 表には、表51 に示すような列が含まれます。

表 51. DTD_REF 表

列名	データ・タイプ	説明
DTDID	VARCHAR(128)	(一意的で、NULL ではない) 基本キー。これは DTD の識別に使われます。基本キーが DAD ファイル内に指定されている場合、 DAD ファイルは DTD の定義したスキーマに従う必要があります。
CONTENT	XMLCLOB	DTD の内容。
USAGE_COUNT	INTEGER	データベース内において、この DTD を使って DAD ファイルを定義している XML 列および XML コレクションの数。
AUTHOR	VARCHAR(128)	DTD の作成者 (この情報の入力はオプションです)。
CREATOR	VARCHAR(128)	最初にデータ挿入を行ったユーザー ID。CREATOR 列は任意指定です。

表 51. DTD_REF 表 (続き)

列名	データ・タイプ	説明
UPDATOR	VARCHAR(128)	最後に更新を行ったユーザー ID。 UPDATOR 列は任意指定です。

制限: アプリケーションが DTD を変更できるのは、 USAGE_COUNT がゼロの場合のみです。

XML 使用状況表

この表は、XML を使用できる各列に関する一般情報を保管します。 XML_USAGE 表のスキーマ名は db2xml、基本キーは (*table_name*, *col_name*) です。 XML_USAGE 表はデータベースが使用可能になる時に作成されます。この表の列は表52 のとおりです。

表 52. XML_USAGE 表

列名	説明
table_schema	XML 列の場合、XML 列を 1 つ含むユーザー表のスキーマ名です。 XML コレクションの場合、デフォルト・スキーマ名の ″DXX_COLL″ の値です。
table_name	XML 列の場合、XML 列を 1 つ含むユーザー表の名前です。 XML コレクションの場合、そのエンティティーをコレクションとして識別する値 ″DXX_COLLECTION″ です。
col_name	XML 列または XML コレクションの名前。これは table_name とともに複合キーを構成します。
DTDID	DAD ファイルを定義する、DTD_REF 内の DTD ID。これは外部キーです。
DAD	列に関連付けられた DAD ファイルの内容。
default_view	デフォルト視点名があれば、それを保管します。
trigger_suffix	非 NULL。固有なトリガー名に使用。
Validation	可は 1、否は 0。
access_mode	XML コレクションは 1、XML 列は 0。

制限: アプリケーションが DTD を変更できるのは、 USAGE_COUNT がゼロの場合のみです。

第12章 診断情報

プログラム内のすべての組み込み SQL ステートメントの実行時、および (DB2 XML エクステンダーのユーザー定義関数 (UDF) の呼び出しを含む) すべての DB2 コマンド行インターフェース (CLI) 呼び出しの際には、その組み込み SQL ステートメントや DB2 CLI 呼び出しが正常に実行されたかどうかを示すコードが生成されます。

プログラムでは、これらのコードを補足する情報を取り出すことができます。この情報には、SQLSTATE 情報およびエラー・メッセージがあります。この診断情報を使用して、プログラムの問題を正確に把握して修正することができます。

時として、問題の原因が何であるか診断が難しいこともあります。 その場合、問題を正確に把握して修正するために、ソフトウェア・サポートに情報を提供する必要があるかもしれません。 XML エクステンダーには、XML エクステンダーの活動を記録するトレース機能があります。 このトレース情報は、IBM ソフトウェア・サポートにとって重要な情報源となります。 トレース機能は、IBM ソフトウェア・サポートの指示のもとでのみ使用してください。

この章では、この診断情報を入手する方法を説明します。内容は次のとおりです。

- XML エクステンダー UDF 戻りコードの処理方法
- トレースの制御方法

また、XML エクステンダーが戻す SQLSTATE コードとエラー・メッセージについてもリストして説明します。

UDF 戻りコードの処理

組み込み SQL ステートメントは、SQLCA 構造体の SQLCODE、SQLWARN、および SQLSTATE フィールドにコードを戻します。この構造体は SQLCA INCLUDE ファイルの中で定義されます。 (SQLCA 構造体および SQLCA INCLUDE ファイルについて、詳しくは、*DB2 アプリケーション開発の手引き* を参照してください。)

DB2 CLI 呼び出しによって SQLCODE 値および SQLSTATE 値が戻ります。これらは SQLError 関数を使って検索できます。 (SQLError 関数を使ったエラー情報の検索について、詳しくは、*CLI ガイドおよび解説書* を参照してください。)

SQLCODE 値が 0 であれば、ステートメントが正常に実行されたことを意味します (ただし、警告状態が発生している場合もあります)。正の SQLCODE 値は、ステートメントが正常に実行されたものの、警告が発生したことを意味します。 (組み込み SQL ステートメントは、 0 または正の SQLCODE 値に関連する警告情報を SQLWARN フィールドに戻します。) 負の SQLCODE 値は、エラーが生じたことを意味します。

DB2 は、それぞれの SQLCODE 値にメッセージを関連付けます。 XML エクステンダー UDF は、警告またはエラーの状態を発見すると、 SQLCODE メッセージに含める関連情報を DB2 に渡します。

SQLSTATE 値には、SQLCODE メッセージを補足するコードが含まれています。 XML エクステンダーが戻すそれぞれの SQLSTATE コードの説明は、257ページの『SQLSTATE コード』を参照してください。

DB2 XML エクステンダー UDF を呼び出す組み込み SQL ステートメントおよび DB2 CLI 呼び出しによって、これらの UDF に固有の SQLCODE メッセージや SQLSTATE 値が戻ることもありますが、 DB2 は、他の組み込み SQL ステートメントまたは DB2 CLI 呼び出しの際と同じようにしてこれらの値を戻します。したがって、これらの値にアクセスする方法は、 DB2 XML エクステンダー UDF を開始しない組み込み SQL ステートメントまたは DB2 CLI 呼び出しの場合と同じです。

XML エクステンダーが戻す SQLSTATE 値および関連メッセージのメッセージ番号については、 257ページの『SQLSTATE コード』を参照してください。それぞれのメッセージについては、262ページの『メッセージ』を参照してください。

ストアード・プロシージャー戻りコードの処理

XML エクステンダーには、ストアード・プロシージャーに関する問題の解決に役立つ戻りコードがあります。ストアード・プロシージャーから戻りコードを受け取ったら、以下のファイルを調べてください。このファイルでは、戻りコードと XML エクステンダー・エラー・メッセージ番号および記号定数が突き合わされています。

```
DXX_INSTALL/include/dxxrc.h
```

262ページの『メッセージ』のエラー・メッセージ番号を参照して、その説明の診断情報を利用することができます。

SQLSTATE コード

表53 は、XML エクステンダーの戻す SQLSTATE 値のリストおよび説明です。それぞれの SQLSTATE 値ごとに、記号表示についても説明されています。また、表には各 SQLSTATE 値に関連したメッセージ番号もリストされます。それぞれのメッセージについては、262ページの『メッセージ』を参照してください。

表 53. SQLSTATE コードおよび関連メッセージ番号

SQLSTATE	メッセージ番号	説明
00000	DXX*nnnn*I	エラーは発生していません。
01HX0	DXXD003W	パス式で指定されたエレメントまたは属性が XML 文書にありません。
38X00	DXXC000E	XML エクステンダーは、指定されたファイルを開くことができません。
38X01	DXXA072E	XML エクステンダーは、データベースを使用可能にする前に自動的にバインドを試みましたが、バインド・ファイルを検出できませんでした。
	DXXC001E	XML エクステンダーは、指定されたファイルを検出できませんでした。
38X02	DXXC002E	XML エクステンダーは、指定されたファイルからデータを読み取ることができません。
38X03	DXXC003E	XML エクステンダーは、データをファイルに書き込むことができません。
	DXXC011E	XML エクステンダーは、データをトレース制御ファイルに書き込むことができません。
38X04	DXXC004E	XML エクステンダーは、指定されたロケーターを操作できませんでした。

表 53. SQLSTATE コードおよび関連メッセージ番号 (続き)

SQLSTATE	メッセージ番号	説明
38X05	DXXC005E	ファイル・サイズが XMLVarchar サイズより大きいため、 XML エクステンダーがファイルからインポートできなかったデータがあります。
38X06	DXXC006E	ファイル・サイズが XMLCLOB のサイズより大きいため、 XML エクステンダーがファイルからインポートできなかったデータがあります。
38X07	DXXC007E	LOB ロケーターのバイト数がファイル・サイズと等しくありません。
38X08	DXXD001E	スカラー抽出関数が、複数出現するロケーション・パスを使用しました。スカラー関数は、複数オカレンスのないロケーション・パスのみを使用することができます。
38X09	DXXD002E	パス式の構文が正しくありません。
38X10	DXXG002E	XML エクステンダーは、オペレーティング・システムからメモリーを割り当てることができませんでした。
38X11	DXXA009E	このストアード・プロシージャーは XML 列専用です。
38X12	DXXA010E	XML エクステンダーが列を使用可能にしようとした時、DTD ID を検出できませんでした (DTD ID は、文書アクセス定義 (DAD) ファイルの中で DTD 用に指定された ID)。
38X14	DXXD000E	無効な文書を表の中に保管しようとしました。妥当性検査に失敗しました。

表 53. SQLSTATE コードおよび関連メッセージ番号 (続き)

SQLSTATE	メッセージ番号	説明
38X15	DXXA056E	文書アクセス定義 (DAD) ファイル内の妥当性検査エレメントが正しくないか、またはエレメントがありません。
	DXXA057E	文書アクセス定義 (DAD) ファイル内のサイド表の名前属性が正しくないか、または属性がありません。
	DXXA058E	文書アクセス定義 (DAD) ファイル内の列の名前属性が正しくないか、または属性がありません。
	DXXA059E	文書アクセス定義 (DAD) ファイル内の列のタイプ属性が正しくないか、または属性がありません。
	DXXA060E	文書アクセス定義 (DAD) ファイル内の列のパス属性が正しくないか、または属性がありません。
	DXXA061E	文書アクセス定義 (DAD) ファイル内の列の multi_occurrence 属性が正しくないか、または属性がありません。
	DXXQ000E	文書アクセス定義 (DAD) ファイル内に必須エレメントがありません。
38X16	DXXG004E	必須パラメーターのヌル値が XML ストアード・プロシージャーに渡されました。
38X17	DXXQ001E	文書アクセス定義 (DAD)、またはこれをオーバーライドするファイルの中の SQL ステートメントが無効です。 XML 文書を生成するには SELECT ステートメントが必要です。
38X18	DXXG001E	XML エクステンダーは内部エラーを検出しました。

表 53. SQLSTATE コードおよび関連メッセージ番号 (続き)

SQLSTATE	メッセージ番号	説明
	DXXG006E	CLI を使用中に XML エクステンダーで内部エラーが発生しました。
38X19	DXXQ002E	システムのメモリーまたはディスク・スペースが不足しています。生成される XML 文書を保管する容量がありません。
38X20	DXXQ003W	ユーザー定義の SQL 照会によって、指定した最大数以上の XML 文書が生成されます。指定された数の文書のみが戻されます。
38X21	DXXQ004E	指定された列は、SQL 照会の結果には含まれません。
38X22	DXXQ005E	SQL 照会の XML へのマッピングが正しくありません。
38X23	DXXQ006E	文書アクセス定義 (DAD) ファイル内の attribute_node エレメントに名前属性がありません。
38X24	DXXQ007E	文書アクセス定義 (DAD) ファイル内の attribute_node エレメントに、列エレメントまたは RDB_node がありません。
38X25	DXXQ008E	文書アクセス定義 (DAD) ファイル内の text_node エレメントに、列エレメントがありません。
38X26	DXXQ009E	指定された結果表が、システム・カタログ内にありません。
38X27	DXXQ010E	attribute_node または text_node の RDB_node には表が必要です。
	DXXQ011E	attribute_node または text_node の RDB_node には列が必要です。
	DXXQ017E	XML エクステンダーの生成した XML 文書が大きすぎて、結果表の列の中に入りません。
38X28	DXXQ012E	DAD の処理中に XML エクステンダーは予期したエレメントを検出できませんでした。

表 53. SQLSTATE コードおよび関連メッセージ番号 (続き)

SQLSTATE	メッセージ番号	説明
	DXXQ016E	すべての表は、文書アクセス定義 (DAD) ファイル内の先頭エレメントの RDB_node で定義しなければなりません。サブエレメントの表は、先頭エレメントで定義された表と一致しなければなりません。この RDB_node 内の表名は、先頭エレメントの中にはありません。
38X29	DXXQ013E	エレメントの表または列の名前が、文書アクセス定義 (DAD) ファイル内で指定されていなければなりません。
	DXXQ015E	文書アクセス定義 (DAD) の条件エレメントの条件が、無効な形式です。
38X30	DXXQ014E	文書アクセス定義 (DAD) ファイル内の element_node エレメントに名前属性がありません。
	DXXQ018E	SQL を XML にマップする文書アクセス定義 (DAD) ファイル内の SQL ステートメントに、 ORDER BY 文節がありません。
38X31	DXXQ019E	エレメント objids は、SQL を XML にマップする文書アクセス定義 (DAD) ファイルに列エレメントを持ちません。
38X36	DXXA073E	データベースがバインドされていません。使用可能にする前にデータベースをバインドしてください。
38X37	DXXG007E	サーバーのオペレーティング・システムの地域が、 DB2 コード・ページと矛盾します。
38X38	DXXG008E	サーバーのオペレーティング・システムの地域設定が、コード・ページ表にありません。

表 53. SQLSTATE コードおよび関連メッセージ番号 (続き)

SQLSTATE	メッセージ番号	説明
38x33	DXXG005E	このパラメーターはこのリリースではサポートされません。将来のリリースでサポートされます。
38x34	DXXG000E	無効なファイル名が指定されました。

メッセージ

XML エクステンダーは、問題判別に役立つエラー・メッセージを提供します。

エラー・メッセージ

XML エクステンダーは、操作の完了時またはエラー検出時に以下のようなメッセージを生成します。

DXXA000I 列 <*column_name*> の使用可能化中。しばらくお待ちください。

説明: これは情報メッセージです。

ユーザーの処置: 処置は必要ありません。

DXXA001S ビルド <*build_ID*>、ファイル <*file_name*> および行 <*line_number*> で、予期しないエラーが発生しました。

説明: 予期しないエラーが発生しました。

ユーザーの処置: このエラーが続く場合、IBM ソフトウェア・サービス提供者に連絡してください。 エラーを報告する場合、すべてのメッセージ・テキスト、トレース・ファイル、および問題の再現方法についての説明を必ず知らせてください。

DXXA002I データベース <*database*> に接続中。

説明: これは情報メッセージです。

ユーザーの処置: 処置は必要ありません。

DXXA003E データベース <*database*> に接続できません。

説明: 指定されたデータベースが存在しないか、または破損しています。

ユーザーの処置:

1. データベースが正しく指定されていることを確認してください。
2. データベースが存在し、アクセス可能であることを確認してください。
3. データベースが破損しているかどうかを判別します。データベースが破損している場合は、バックアップから回復するようデータベース管理者に要請します。

DXXA004E　データベース *<database>* を使用可能にできません。

説明: データベースはすでに使用可能であるか、または破損しています。

ユーザーの処置:

1. データベースが使用可能であるかどうかを確認します。
2. データベースが破損しているかどうかを判別します。データベースが破損している場合は、バックアップから回復するようデータベース管理者に要請します。

DXXA005I　データベース *<database>* の使用可能化中。お待ちください。

説明: これは情報メッセージです。

ユーザーの処置: 処置は必要ありません。

DXXA006I　データベース *<database>* は、正常に使用可能化されました。

説明: これは情報メッセージです。

ユーザーの処置: 処置は必要ありません。

DXXA007E　データベース *<database>* を使用不可にできません。

説明: データベースが XML 列またはコレクションを含んでいる場合、 XML エクステンダーはこれを使用不可にできません。

ユーザーの処置: 重要なデータのバックアップを取り、XML 列またはコレクションをすべて使用不可にし、表の更新または除去を行ってデータベースから XML データ・タイプをなくします。

DXXA008I　列 *<column_name>* を使用不可にしています。しばらくお待ちください。

説明: これは情報メッセージです。

ユーザーの処置: 処置は必要ありません。

DXXA009E　Xcolumn タグが DAD ファイル内に指定されていません。

説明: このストアード・プロシージャーは XML 列専用です。

ユーザーの処置: Xcolumn タグが DAD ファイル内に正しく指定されていることを確認してください。

DXXA010E　DTD ID *<dtdid>* の検索が失敗しました。

説明: XML エクステンダーが列を使用可能にしようとした時、DTD ID を検出できませんでした (DTD ID は、文書アクセス定義 (DAD) ファイルの中で DTD 用に指定された ID)。

ユーザーの処置: DTD ID の正しい値が DAD ファイルで指定されていることを確認してください。

DXXA011E　DB2XML.XML_USAGE 表への挿入が失敗しました。

説明: XML エクステンダーが列を使用可能にしようとした時、 DB2XML.XML_USAGE 表の中にレコードを挿入できませんでした。

ユーザーの処置: DB2XML.XML_USAGE 表が存在すること、および同じ名前のレコードが表にまだ存在しないことを確認します。

DXXA012E　DB2XML.DTD_REF 表の更新が失敗しました。

説明: XML エクステンダーが列を使用可能にしようとした時、DB2XML.DTD_REF 表を更新できませんでした。

ユーザーの処置: DB2XML.DTD_REF 表が存在することを確認してください。表が破壊されていないかどうか、また管理ユーザー ID が表を更新するために正しい権限を持っているかどうかを判別してください。

DXXA013E　表 <*table_name*> の更新が失敗しました。

説明: XML エクステンダーが列を使用可能にしようとした時、指定された表を更新できませんでした。

ユーザーの処置: 表の更新に必要な特権を確認してください。

DXXA014E　指定された root ID 列: <*root_id*> は、表 <*table_name*> の単一の基本キーではありません。

説明: 指定されたルート ID がキーではないか、または表 *table_name* のただ 1 つのキーではありません。

ユーザーの処置: 指定されたルート ID が、表のただ 1 つの基本キーであることを確認してください。

DXXA015E　列 DXXROOT_ID は表 <*table_name*> に既に存在しています。

説明: 列 DXXROOT_ID は存在しますが、XML エクステンダーが作成したものではありません。

ユーザーの処置: 列を使用可能にする時、異なった列名を使用することによって、ルート ID オプションに基本列を指定します。

DXXA016E　入力表 <*table_name*> が存在しません。

説明: XML エクステンダーは、システム・カタログ内に指定された表を検出できませんでした。

ユーザーの処置: データベースに表が存在し、正しく指定されていることを確認してください。

DXXA017E　入力列 <*column_name*> が、指定した表 <*table_name*> に存在しません。

説明: XML エクステンダーは、システム・カタログ内に列を検出できませんでした。

ユーザーの処置: ユーザー表内に列が存在することを確認してください。

DXXA018E　指定した列は XML データに対して使用可能化されていません。

説明: XML エクステンダーが列を使用不可にしようとした時、 DB2XML.XML_USAGE 表内に列を検出できませんでした。これは、この列が使用可能でないことを示します。 列が XML 使用可能でなければ、これを使用不可にする必要はありません。

ユーザーの処置: 処置は必要ありません。

DXXA019E　列を使用可能化するのに必要な入力パラメーターがヌルです。

説明: enable_column() ストアード・プロシージャーの必須入力パラメーターがヌルです。

ユーザーの処置: enable_column() ストアード・プロシージャーのすべての入力パラメーターを検査してください。

DXXA020E　列が表 <*table_name*> に見つかりません。

説明: XML エクステンダーがデフォルト・ビューを作成しようとした時、指定された表の中に列

を検出できませんでした。

ユーザーの処置: 列および表名が正しく指定されていることを確認してください。

DXXA021E デフォルト・ビュー <*default_view*> を作成できません。

説明: XML エクステンダーが列を使用可能化しようとした時、指定されたビューを作成できませんでした。

ユーザーの処置: デフォルト・ビュー名が固有のものであることを確認してください。その名前のビューがすでに存在する場合は、固有の名前をデフォルト・ビューに指定してください。

DXXA022I 列 <*column_name*> が使用可能化されました。

説明: これは情報メッセージです。

ユーザーの処置: 応答は必要ありません。

DXXA023E DAD ファイルが見つかりません。

説明: XML エクステンダーが列を使用不可にしようとした時、文書アクセス定義 (DAD) ファイルを検出できませんでした。

ユーザーの処置: 正しいデータベース名、表名、または列名を指定したことを確認してください。

DXXA024E システム・カタログ表にアクセス中に XML エクステンダーで内部エラーが発生しました。

説明: XML エクステンダーは、システム・カタログ表にアクセスできませんでした。

ユーザーの処置: データベースが安定状態であることを確認してください。

DXXA025E デフォルト・ビュー <*default_view*> をドロップできません。

説明: XML エクステンダーが列を使用不可にしようとした時、デフォルト・ビューを除去できませんでした。

ユーザーの処置: XML エクステンダーの管理ユーザー ID に、デフォルト・ビューの除去に必要な特権があることを確認してください。

DXXA026E サイド表 <*side_table*> をドロップできません。

説明: XML エクステンダーが列を使用不可にしようとした時、指定された表を除去できませんでした。

ユーザーの処置: XML エクステンダーの管理者ユーザー ID に、表の除去に必要な特権があることを確認してください。

DXXA027E 列を使用不可にできませんでした。

説明: XML エクステンダーは、内部トリガー障害のため、列を使用不可にできませんでした。

ユーザーの処置: トレース機能を使用してトレース・ファイルを作成し、問題の修正を試みてください。このエラーが続く場合、ソフトウェア・サービス提供者に連絡して、トレース・ファイルを提出してください。

DXXA028E 列を使用不可にできませんでした。

説明: XML エクステンダーは、内部トリガー障害のため、列を使用不可にできませんでした。

ユーザーの処置: トレース機能を使用してトレース・ファイルを作成し、問題の修正を試みてください。このエラーが続く場合、ソフトウェア・サービス提供者に連絡して、トレース・ファイルを提出してください。

DXXA029E　列を使用不可にできませんでした。

説明: XML エクステンダーは、内部トリガー障害のため、列を使用不可にできませんでした。

ユーザーの処置: トレース機能を使用してトレース・ファイルを作成し、問題の修正を試みてください。このエラーが続く場合、ソフトウェア・サービス提供者に連絡して、トレース・ファイルを提出してください。

DXXA030E　列を使用不可にできませんでした。

説明: XML エクステンダーは、内部トリガー障害のため、列を使用不可にできませんでした。

ユーザーの処置: トレース機能を使用してトレース・ファイルを作成し、問題の修正を試みてください。このエラーが続く場合、ソフトウェア・サービス提供者に連絡して、トレース・ファイルを提出してください。

DXXA031E　アプリケーション表の DXXROOT_ID 列値をヌルにリセットすることができませんでした。

説明: XML エクステンダーが列を使用不可にしようとした時、アプリケーション表の DXXROOT_ID の値をヌルに設定できませんでした。

ユーザーの処置: XML エクステンダーの管理者ユーザー ID に、アプリケーション表の更新に必要な特権があることを確認してください。

DXXA032E　DB2XML.XML_USAGE 表内の USAGE_COUNT の減分に失敗しました。

説明: XML エクステンダーが列を使用不可にしようとした時、 USAGE_COUNT 列の値を 1 つ減らすことができませんでした。

ユーザーの処置: DB2XML.XML_USAGE 表が存在することと、 XML エクステンダー管理者ユーザー ID に、表の更新に必要な特権があることを確認してください。

DXXA033E　DB2XML.XML_USAGE 表から列を削除しようとして失敗しました。

説明: XML エクステンダーが列を使用不可にしようとした時、 DB2XML.XML_USAGE 表内のこの列に関連する行を削除できませんでした。

ユーザーの処置: DB2XML.XML_USAGE 表が存在することと、 XML エクステンダー管理者ユーザー ID に、この表の更新に必要な特権があることを確認してください。

DXXA034I　XML エクステンダーは列 <*column_name*> を正常に使用不可にしました。

説明: これは情報メッセージです。

ユーザーの処置: 処置は必要ありません。

DXXA035I　XML エクステンダーはデータベース <*database*> を使用不可にしています。お待ちください。

説明: これは情報メッセージです。

ユーザーの処置: 処置は必要ありません。

DXXA036I　XML エクステンダーは、データベース <*database*> を正常に使用不可にしました。

説明: これは情報メッセージです。

ユーザーの処置: 処置は必要ありません。

DXXA037E　指定された表スペース名は 18 文字を超えています。

説明: 表スペース名を英数字で 18 文字よりも長くすることはできません。

ユーザーの処置: 18 文字以内の短い名前を指定してください。

DXXA038E　指定されたデフォルト・ビュー名は 18 文字を超えています。

説明: デフォルト・ビュー名を英数字で 18 文字よりも長くすることはできません。

ユーザーの処置: 18 文字以内の短い名前を指定してください。

DXXA039E　指定された ROOT_ID 名は 18 文字を超えています。

説明: ROOT_ID 名を英数字で 18 文字よりも長くすることはできません。

ユーザーの処置: 18 文字以内の短い名前を指定してください。

DXXA046E　サイド表 <*side_table*> を作成できません。

説明: XML エクステンダーが列を使用可能化しようとした時、指定されたサイド表を作成できませんでした。

ユーザーの処置: XML エクステンダーの管理者ユーザー ID に、サイド表の作成に必要な特権があることを確認してください。

DXXA047E　列を使用可能にできませんでした。

説明: XML エクステンダーは、内部トリガー障害のため、列を使用可能にできませんでした。

ユーザーの処置: トレース機能を使用してトレース・ファイルを作成し、問題の修正を試みてください。このエラーが続く場合、ソフトウェア・サービス提供者に連絡して、トレース・ファイルを提出してください。

DXXA048E　列を使用可能にできませんでした。

説明: XML エクステンダーは、内部トリガー障害のため、列を使用可能にできませんでした。

ユーザーの処置: トレース機能を使用してトレース・ファイルを作成し、問題の修正を試みてください。このエラーが続く場合、ソフトウェア・サービス提供者に連絡して、トレース・ファイルを提出してください。

DXXA049E　列を使用可能にできませんでした。

説明: XML エクステンダーは、内部トリガー障害のため、列を使用可能にできませんでした。

ユーザーの処置: トレース機能を使用してトレース・ファイルを作成し、問題の修正を試みてください。このエラーが続く場合、ソフトウェア・サービス提供者に連絡して、トレース・ファイルを提出してください。

DXXA050E　列を使用可能にできませんでした。

説明: XML エクステンダーは、内部トリガー障害のため、列を使用可能にできませんでした。

ユーザーの処置: トレース機能を使用してトレース・ファイルを作成し、問題の修正を試みてください。このエラーが続く場合、ソフトウェア・サービス提供者に連絡して、トレース・ファイルを提出してください。

DXXA051E　列を使用不可にできませんでした。

説明: XML エクステンダーは、内部トリガー障害のため、列を使用不可にできませんでした。

ユーザーの処置: トレース機能を使用してトレース・ファイルを作成し、問題の修正を試みてください。このエラーが続く場合、ソフトウェア・サービス提供者に連絡して、トレース・ファイルを提出してください。

DXXA052E　列を使用不可にできませんでした。

説明: XML エクステンダーは、内部トリガー障害のため、列を使用不可にできませんでした。

ユーザーの処置: トレース機能を使用してトレース・ファイルを作成し、問題の修正を試みてください。このエラーが続く場合、ソフトウェア・サ

ービス提供者に連絡して、トレース・ファイルを提出してください。

DXXA053E　列を使用可能にできませんでした。

説明: XML エクステンダーは、内部トリガー障害のため、列を使用可能にできませんでした。

ユーザーの処置: トレース機能を使用してトレース・ファイルを作成し、問題の修正を試みてください。このエラーが続く場合、ソフトウェア・サービス提供者に連絡して、トレース・ファイルを提出してください。

DXXA054E　列を使用可能にできませんでした。

説明: XML エクステンダーは、内部トリガー障害のため、列を使用可能にできませんでした。

ユーザーの処置: トレース機能を使用してトレース・ファイルを作成し、問題の修正を試みてください。このエラーが続く場合、ソフトウェア・サービス提供者に連絡して、トレース・ファイルを提出してください。

DXXA056E　DAD ファイル内の妥当性検査値 <*validation_value*> は無効です。

説明: 文書アクセス定義 (DAD) ファイル内の妥当性検査エレメントが正しくないか、またはエレメントがありません。

ユーザーの処置: 妥当性検査エレメントが DAD ファイルに正しく指定されていることを確認してください。

DXXA057E　DAD 内のサイド表名 <*side_table_name*> は無効です。

説明: 文書アクセス定義 (DAD) ファイル内のサイド表の名前属性が正しくないか、または属性がありません。

ユーザーの処置: サイド表の名前属性が DAD ファイル内に正しく指定されていることを確認してください。

DXXA058E　DAD ファイル内の列名 <*column_name*> は無効です。

説明: 文書アクセス定義 (DAD) ファイル内の列の名前属性が正しくないか、または属性がありません。

ユーザーの処置: 列の名前属性が DAD ファイル内に正しく指定されていることを確認してください。

DXXA059E　DAD ファイル内のタイプ <*column_type*> (列 <*column_name*>) が無効です。

説明: 文書アクセス定義 (DAD) ファイル内の列のタイプ属性が正しくないか、または属性がありません。

ユーザーの処置: 列のタイプ属性が DAD ファイル内に正しく指定されていることを確認してください。

DXXA060E　DAD ファイル内の <*column_name*> のパス属性 <*location_path*> が無効です。

説明: 文書アクセス定義 (DAD) ファイル内の列のパス属性が正しくないか、または属性がありません。

ユーザーの処置: 列のパス属性が DAD ファイル内に正しく指定されていることを確認してください。

DXXA061E　DAD ファイル内の multi_occurrence 属性 <*multi_occurrence*> (<*column_name*>) が無効です。

説明: 文書アクセス定義 (DAD) ファイル内の列の multi_occurrence 属性が正しくないか、または属性がありません。

ユーザーの処置: 列の multi_occurrence 属性が

DAD ファイル内に正しく指定されていることを確認してください。

DXXA062E **<*column_name*> の列番号 (表 <*table_name*>) を検索することができません。**

説明: XML エクステンダーは、 *table_name* 表の *column_name* の列番号をシステム・カタログから検索できませんでした。

ユーザーの処置: アプリケーション表が適正に定義されていることを確認してください。

DXXA063I **コレクション <*collection_name*> を使用可能にしています。しばらくお待ちください。**

説明: これは情報メッセージです。

ユーザーの処置: 処置は必要ありません。

DXXA064I **コレクション <*collection_name*> を使用不可にしています。しばらくお待ちください。**

説明: これは情報メッセージです。

ユーザーの処置: 処置は必要ありません。

DXXA065E **ストアード・プロシージャー <*procedure_name*> の呼び出しに失敗しました。**

説明: 共用ライブラリー db2xml を検査して、許可が正しいかどうかを確認してください。

ユーザーの処置: クライアントにストアード・プロシージャーを実行する許可があることを確認してください。

DXXA066I **XML エクステンダーは、コレクション <*collection_name*> を正常に使用不可にしました。**

説明: これは情報メッセージです。

ユーザーの処置: 応答は必要ありません。

DXXA067I **XML エクステンダーは、コレクション <*collection_name*> を正常に使用可能にしました。**

説明: これは情報メッセージです。

ユーザーの処置: 応答は必要ありません。

DXXA068I **XML エクステンダーは、トレースを正常にオンにしました。**

説明: これは情報メッセージです。

ユーザーの処置: 応答は必要ありません。

DXXA069I **XML エクステンダーは、トレースを正常にオフにしました。**

説明: これは情報メッセージです。

ユーザーの処置: 応答は必要ありません。

DXXA070W **データベースはすでに使用可能になっています。**

説明: データベースの使用可能化コマンドが、使用可能になっているデータベースに対して実行されました。

ユーザーの処置: 処置は必要ありません。

DXXA071W **データベースはすでに使用不可になっています。**

説明: データベースを使用不可にするコマンドが、すでに使用不可になっているデータベースに対して実行されました。

ユーザーの処置: 処置は必要ありません。

DXXA072E　XML エクステンダーはバインド・ファイルを見つけられませんでした。使用可能にする前にデータベースをバインドしてください。

説明: XML エクステンダーは、データベースを使用可能にする前に自動的にバインドを試みましたが、バインド・ファイルを検出できませんでした。

ユーザーの処置: 使用可能にする前にデータベースをバインドしてください。

DXXA073E　データベースがバインドされていません。使用可能にする前にデータベースをバインドしてください。

説明: データベースがバインドされていません。使用可能にする前にデータベースをバインドしてください。

ユーザーの処置: 使用可能にする前にデータベースをバインドしてください。

DXXA074E　パラメーター・タイプが間違っています。ストアード・プロシージャーには STRING パラメーターを使用してください。

説明: ストアード・プロシージャーには STRING パラメーターを使用してください。

ユーザーの処置: 入力パラメーターが STRING タイプになるように宣言してください。

DXXA075E　パラメーター・タイプが間違っています。入力パラメーターには、long 型を使用してください。

説明: ストアード・プロシージャーは、入力パラメーターが LONG タイプになることを予期しています。

ユーザーの処置: 入力パラメーターが LONG タイプになるように宣言してください。

DXXA076E　XML エクステンダーのトレース・インスタンス ID が無効です。

説明: 提供されたインスタンス ID のトレースを開始できません。

ユーザーの処置: インスタンス ID が正しい AS/400 ユーザー ID であるかどうか確認してください。

DXXC000E　指定されたファイルをオープンできませんでした。

説明: XML エクステンダーは、指定されたファイルを開くことができません。

ユーザーの処置: アプリケーション・ユーザー ID に、ファイルの読み取りおよび書き込み許可が与えられていることを確認してください。

DXXC001E　指定されたファイルが見つかりませんでした。

説明: XML エクステンダーは、指定されたファイルを検出できませんでした。

ユーザーの処置: ファイルが存在し、パスが正しく指定されていることを確認してください。

DXXC002E　ファイルを読み取れませんでした。

説明: XML エクステンダーは、指定されたファイルからデータを読み取ることができません。

ユーザーの処置: アプリケーション・ユーザー ID に、ファイルの読み取り許可が与えられていることを確認してください。

DXXC003E　指定されたファイルに書き込めませんでした。

説明: XML エクステンダーは、データをファイルに書き込むことができません。

ユーザーの処置: アプリケーション・ユーザー ID にファイルの書き込み許可が与えられていて、ファイル・システムに十分なスペースがある

ことを確認してください。

DXXC004E　LOB ロケーターを操作することができませんでした: rc=<*locator_rc*>

説明: XML エクステンダーは、指定されたロケーターを操作できませんでした。

ユーザーの処置: LOB ロケーターが正しく設定されていることを確認してください。

DXXC005E　入力ファイル・サイズが XMLVarchar サイズより大きいです。

説明: ファイル・サイズが XMLVarchar サイズより大きいため、 XML エクステンダーがファイルからインポートできなかったデータがあります。

ユーザーの処置: XMLCLOB 列タイプを使用してください。

DXXC006E　入力ファイルが DB2 LOB 制限を超えています。

説明: ファイル・サイズが XMLCLOB のサイズより大きいため、 XML エクステンダーがファイルからインポートできなかったデータがあります。

ユーザーの処置: ファイルをより小さいオブジェクトに分解するか、または XML コレクションを使用してください。

DXXC007E　ファイルから LOB ロケーターにデータを検索できませんでした。

説明: LOB ロケーターのバイト数がファイル・サイズと等しくありません。

ユーザーの処置: LOB ロケーターが正しく設定されていることを確認してください。

DXXC008E　ファイル <*file_name*> **を除去できませんでした。**

説明: ファイルに共用アクセスが設定されているか、またはファイルがまだ開いています。

ユーザーの処置: ファイルをクローズするか、またはファイルを保留にしているプロセスを停止してください。 DB2 を停止してから、再始動する必要があります。

DXXC009E　**<*directory*> **ディレクトリーにファイルを作成できませんでした。

説明: XML エクステンダーは、ディレクトリー *directory* 内にファイルを作成することができません。

ユーザーの処置: ディレクトリーが存在し、アプリケーション・ユーザー ID にディレクトリーに対する書き込み許可が与えられており、ファイル・システムに十分なスペースがあることを確認してください。

DXXC010E　ファイル <*file_name*> **に書き込み中にエラーが発生しました。**

説明: ファイル *file_name* に書き込み中にエラーがありました。

ユーザーの処置: ファイル・システムに十分なスペースがあることを確認してください。

DXXC011E　トレース制御ファイルに書き込めませんでした。

説明: XML エクステンダーは、データをトレース制御ファイルに書き込むことができません。

ユーザーの処置: アプリケーション・ユーザー ID にファイルの書き込み許可が与えられていて、ファイル・システムに十分なスペースがあることを確認してください。

DXXC012E　一時ファイルを作成できません。

説明: システム temp ディレクトリー内にファイルを作成できません。

ユーザーの処置: アプリケーション・ユーザー ID にディレクトリーに対する書き込み許可が与えられていて、ファイル・システムに十分なスペースがあることを確認してください。

DXXD000E　無効な XML 文書が拒否されました。

説明: 無効な文書を表の中に保管しようとしました。妥当性検査に失敗しました。

ユーザーの処置: 不可視の無効文字を表示できるエディターを使用して、この文書を DTD によって検査してください。このエラーを抑制するには、DAD ファイルの妥当性検査をオフにしてください。

DXXD001E　複数のパス <*location_path*> が存在します。

説明: スカラー抽出関数が、複数回発生するロケーション・パスを使用しました。スカラー関数は、複数オカレンスがないロケーション・パスのみを使用することができます。

ユーザーの処置: 表関数を使用してください (スカラー関数名の終わりに 's' を追加してください)。

DXXD002E　サーチ・パスの位置 <*position*> 付近で構文エラーが発生しました。

説明: パス式の構文が正しくありません。

ユーザーの処置: 照会のサーチ・パス引き数を訂正してください。パス式の構文についての資料を参照してください。

DXXD003W　パスが見つかりませんでした。ヌルが戻されました。

説明: パス式で指定されたエレメントまたは属性が XML 文書にありません。

ユーザーの処置: 指定したパスが正しいかどうか検査してください。

DXXG000E　ファイル名 <*file_name*> が無効です。

説明: 無効なファイル名が指定されました。

ユーザーの処置: 正しいファイル名を指定して、再試行してください。

DXXG001E　ビルド <*build_ID*>、ファイル <*file_name*>、行 <*line_number*> で内部エラーが発生しました。

説明: XML エクステンダーは内部エラーを検出しました。

ユーザーの処置: IBM ソフトウェア・サービス提供者に連絡してください。 エラーを報告する場合、すべてのメッセージ、トレース・ファイル、およびエラーの再現方法についての説明を必ず知らせてください。

DXXG002E　システムはメモリーが不足しています。

説明: XML エクステンダーは、オペレーティング・システムからメモリーを割り当てることができませんでした。

ユーザーの処置: いくつかのアプリケーションをクローズして再試行してください。問題が続く場合、ご使用のオペレーティング・システムの資料を参照してください。オペレーティング・システムによっては、問題を訂正するためにシステムをリブートする必要があります。

DXXG004E　無効なヌル・パラメーター。

説明: 必須パラメーターのヌル値が XML ストアード・プロシージャーに渡されました。

ユーザーの処置: ストアード・プロシージャー呼び出しの引き数リストの中で、必須パラメーターをすべて検査してください。

DXXG005E　パラメーターはサポートされていません。

説明: このパラメーターはこのリリースではサポートされません。将来のリリースでサポートされます。

ユーザーの処置: このパラメーターを NULL に設定してください。

DXXG006E　内部エラー CLISTATE=<*clistate*>、**RC=**<*cli_rc*>、**ビルド** <*build_ID*>、**ファイル** <*file_name*>、**行** <*line_number*> **CLIMSG=**<*CLI_msg*>。

説明: CLI を使用中に XML エクステンダーで内部エラーが発生しました。

ユーザーの処置: IBM ソフトウェア・サービス提供者に連絡してください。このエラーの原因は正しくないユーザー入力にあるものと考えられます。エラーを報告する場合、すべての出力メッセージ、トレース・ログ、および問題の再現方法についての説明を必ず知らせてください。可能であれば、DAD、XML 文書、および適用する表定義をすべて送付してください。

DXXG007E　ロケール <*locale*> **が DB2 コード・ページ** <*code_page*> **と矛盾しています。**

説明: サーバーのオペレーティング・システムの地域が、 DB2 コード・ページと矛盾します。

ユーザーの処置: サーバーのオペレーティング・システムの地域設定を修正して、 DB2 を再起動してください。

DXXG008E　ロケール <*locale*> **はサポートされていません。**

説明: サーバーのオペレーティング・システムの地域設定が、コード・ページ表にありません。

ユーザーの処置: サーバーのオペレーティング・システムの地域設定を修正して、 DB2 を再起動してください。

DXXQ000E　**<*Element*> **が DAD ファイルから欠落しています。

説明: 文書アクセス定義 (DAD) ファイル内に必須エレメントがありません。

ユーザーの処置: 欠落しているエレメントを DAD ファイルに追加してください。

DXXQ001E　XML 生成のための SQL ステートメントが無効です。

説明: 文書アクセス定義 (DAD)、またはこれをオーバーライドするファイルの中の SQL ステートメントが無効です。 XML 文書を生成するには SELECT ステートメントが必要です。

ユーザーの処置: SQL ステートメントを訂正してください。

DXXQ002E　XML 文書を保持するためのストレージを生成できません。

説明: システムのメモリーまたはディスク・スペースが不足しています。生成される XML 文書を保管する容量がありません。

ユーザーの処置: 生成される文書の数を制限します。文書アクセス定義 (DAD) ファイルからいずれかの不要なエレメントや属性ノードを取り除くことによって、それぞれの文書サイズを削減してください。

DXXQ003W 結果が最大を超えます。

説明: ユーザー定義の SQL 照会によって、指定した最大数以上の XML 文書が生成されます。指定された数の文書のみが戻されます。

ユーザーの処置: 処置は必要ありません。 すべての文書が必要であれば、文書の最大数としてゼロを指定してください。

DXXQ004E 列 <*column_name*> は照会の結果にありません。

説明: 指定された列は、SQL 照会の結果には含まれません。

ユーザーの処置: 文書アクセス定義 (DAD) ファイル内の指定された列名を変更して、 SQL 照会の結果に含まれる列にしてください。または、SQL 照会を変更して、指定された列が結果に含まれるようにします。

DXXQ004W DAD 内に DTD ID が見つかりません。

説明: DAD において VALIDATION が YES ですが、DTDID エレメントが指定されていません。 妥当性検査は行われません。

ユーザーの処置: 処置は必要ありません。 妥当性検査が必要であれば、DAD ファイル内で DTDID エレメントを指定してください。

DXXQ005E リレーショナル・マッピングが間違っています。エレメント <*element_name*> が、その子列 <*column_name*> より低いレベルになっています。

説明: SQL 照会の XML へのマッピングが正しくありません。

ユーザーの処置: SQL 照会の結果に含まれる列が、トップダウン順のリレーショナル階層になっていることを確認してください。また、それぞれのレベルが単一列候補キーで始まることを確認してください。そのようなキーが表にない場合、照会では表式および DB2 組み込み関数 generate_unique() を使用して、このキーを生成しなければなりません。

DXXQ006E attribute_node エレメントに名前がありません。

説明: 文書アクセス定義 (DAD) ファイル内の attribute_node エレメントに名前属性がありません。

ユーザーの処置: すべての attribute_node の名前が DAD ファイル内に指定されていることを確認してください。

DXXQ007E attribute_node <*attribute_name*> に、列エレメントおよび RDB_node がありません。

説明: 文書アクセス定義 (DAD) ファイル内の attribute_node エレメントに、列エレメントまたは RDB_node がありません。

ユーザーの処置: どの attribute_node にも、列エレメントまたは RDB_node が DAD ファイル内に指定されていることを確認してください。

DXXQ008E text_node エレメントは列エレメントがありません。

説明: 文書アクセス定義 (DAD) ファイル内の text_node エレメントに、列エレメントがありません。

ユーザーの処置: どの text_node にも、列エレメントが DAD ファイル内に指定されていることを確認してください。

DXXQ009E 結果表 <*table_name*> が存在しません。

説明: 指定された結果表が、システム・カタログ内にありません。

ユーザーの処置: ストアード・プロシージャーを

呼び出す前に、結果表を作成してください。

DXXQ010E *<node_name>* **の RDB_node が DAD ファイル内に表を持ちません。**

説明: attribute_node または text_node の RDB_node には表が必要です。

ユーザーの処置: 文書アクセス定義 (DAD) ファイル内で、 attribute_node または text_node の RDB_node の表を指定してください。

DXXQ011E *<node_name>* **の RDB_node エレメントが DAD ファイル内に列を持ちません。**

説明: attribute_node または text_node の RDB_node には列が必要です。

ユーザーの処置: 文書アクセス定義 (DAD) ファイル内で、 attribute_node または text_node の RDB_node の列を指定してください。

DXXQ012E DAD でエラーが発生しました。

説明: DAD の処理中に XML エクステンダーは予期したエレメントを検出できませんでした。

ユーザーの処置: DAD が有効な XML 文書であり、DAD DTD が必要とするすべてのエレメントを含んでいるか検査してください。 DAD DTD に関する XML エクステンダー資料を参照してください。

DXXQ013E 表または列エレメントは、DAD ファイルに名前を持ちません。

説明: エレメントの表または列の名前が、文書アクセス定義 (DAD) ファイル内で指定されていなければなりません。

ユーザーの処置: 表または列エレメントの名前を DAD 内に指定してください。

DXXQ014E element_node エレメントに名前がありません。

説明: 文書アクセス定義 (DAD) ファイル内の element_node エレメントに名前属性がありません。

ユーザーの処置: どの element_node エレメントにも、名前が DAD ファイル内に指定されていることを確認してください。

DXXQ015E 条件形式が無効です。

説明: 文書アクセス定義 (DAD) の条件エレメントの条件が、無効な形式です。

ユーザーの処置: 条件形式を有効なものにしてください。

DXXQ016E この RDB_node の表名は、DAD ファイルの先頭エレメントに定義されていません。

説明: すべての表は、文書アクセス定義 (DAD) ファイル内の先頭エレメントの RDB_node で定義しなければなりません。サブエレメントの表は、先頭エレメントで定義された表と一致しなければなりません。この RDB_node 内の表名は、先頭エレメントの中にはありません。

ユーザーの処置: RDB ノードの表が、DAD ファイルの先頭エレメントの中で定義されることを確認してください。

DXXQ017E 結果表 *<table_name>* 内の列は小さすぎます。

説明: XML エクステンダーの生成した XML 文書が大きすぎて、結果表の列の中に入りません。

ユーザーの処置: 結果表を除去します。より大きな列を使用して別の結果表を作成します。ストアード・プロシージャーを再実行します。

DXXQ018E　ORDER BY 文節が SQL ステートメントから欠落しています。

説明: SQL を XML にマップする文書アクセス定義 (DAD) ファイル内の SQL ステートメントに、 ORDER BY 文節がありません。

ユーザーの処置: DAD ファイルを編集します。エンティティーを識別する列を含む ORDER BY 文節を追加します。

DXXQ019E　エレメント objids は DAD ファイル内に列エレメントを持ちません。

説明: エレメント objids は、SQL を XML にマップする文書アクセス定義 (DAD) ファイルに列エレメントを持ちません。

ユーザーの処置: DAD ファイルを編集します。エレメント objids のサブエレメントとしてキー列を追加してください。

DXXQ020I　XML が正常に生成されました。

説明: 要求された XML 文書が、正常にデータベースから生成されました。

ユーザーの処置: 処置は必要ありません。

DXXQ021E　表 <*table_name*> に列 <*column_name*> がありません。

説明: 表には、指定された列がデータベース内にありません。

ユーザーの処置: DAD で別の列名を指定するか、または指定された列を表データベースの中に追加します。

DXXQ022E　列 <*column_name*> (<*table_name*>) はタイプ <*type_name*> を持つ必要があります。

説明: 列のタイプが正しくありません。

ユーザーの処置: 文書アクセス定義 (DAD) 内の列タイプを訂正してください。

DXXQ023E　列 <*column_name*> (<*table_name*>) は <*length*> より長くすることはできません。

説明: DAD 内の列の定義が長すぎます。

ユーザーの処置: 文書アクセス定義 (DAD) 内の列の長さを訂正してください。

DXXQ024E　表 <*table_name*> を作成できません。

説明: 指定された表を作成できません。

ユーザーの処置: 表を作成しているユーザー ID に、データベース内で表を作成するために必要な権限があることを確認してください。

DXXQ025I　XML が正常に分解されました。

説明: XML 文書が正常に分解され、コレクション内に保管されました。

ユーザーの処置: 処置は必要ありません。

DXXQ026E　XML データ <*xml_name*> は列 <*column_name*> には大きすぎます。

説明: XML 文書からの指定されたデータが大きすぎて、指定された列の中に入りません。

ユーザーの処置: ALTER TABLE ステートメントによって列の長さを長くするか、または XML 文書を編集してデータのサイズを小さくします。

DXXQ028E　コレクションを見つけられません: XML_USAGE 表からの <*collection_name*>。

説明: コレクション用のレコードが XML_USAGE 表内で検出できません。

ユーザーの処置: コレクションを使用可能にしたかどうか確認してください。

DXXQ029E 表 XML_USAGE 内に DAD を見つけられませんでした。コレクション: <*collection_name*>。

説明: コレクション用の DAD レコードが XML_USAGE 表内で検出できません。

ユーザーの処置: コレクションを正しく使用可能にしたかどうか確認してください。

DXXQ030E 不正な XML 上書きです。

説明: ストアード・プロシージャー内に正しくない XML_override 値が指定されています。

ユーザーの処置: XML_override の構文が正しいか確認してください。

DXXQ031E 表名は DB2 で許可されている最大長より長くすることはできません。

説明: DAD 内の条件エレメントによって指定されている表名が長すぎます。

ユーザーの処置: 文書アクセス定義 (DAD) 内の表名の長さを訂正してください。

DXXQ032E 列名は DB2 で許可されている最大長より長くすることはできません。

説明: DAD 内の条件エレメントによって指定されている列名が長すぎます。

ユーザーの処置: 文書アクセス定義 (DAD) 内の列名の長さを訂正してください。

DXXQ033E <*identifier*> で開始する ID が無効です。

説明: ストリングが有効な DB2 SQL ID ではありません。

ユーザーの処置: DB2 SQL ID の規則に準拠するように DAD 内のストリングを訂正してください。

DXXQ034E DAD のトップ RDB_node の状態エレメントが無効です : <*condition*>

説明: 条件エレメントは、論理積 AND によって接続された結合条件から成る有効な WHERE 文節でなければなりません。

ユーザーの処置: DAD 内の結合条件の正しい構文については、XML エクステンダー資料を参照してください。

DXXQ035E DAD のトップ RDB_node の結合状態が無効です : <*condition*>

説明: 最上位の RDB_node の条件エレメント内の列名は、 DAD が複数の表を指定する場合には表名で修飾されなければなりません。

ユーザーの処置: DAD 内の結合条件の正しい構文については、XML エクステンダー資料を参照してください。

DXXQ036E DAD 状態タグの下に指定されているスキーマ名が長過ぎます。

説明: DAD 状態タグの下にあるテキストを解析しているときにエラーが検出されました。状態テキストの ID を修飾しているスキーマ名が長すぎます。

ユーザーの処置: 文書アクセス定義 (DAD) 内の状態タグのテキストを修正してください。

DXXQ037E 複数オカレンスで <*element*> を生成できません。

説明: エレメント・ノードとその子にはデータベースへのマップがありませんが、その multi_occurrence が YES となっています。

ユーザーの処置: multi_occurrence を NO に設定するか、そのノードの子のどれかに RDB_node を作成することによって DAD を修正してください。

診断トレース

XML エクステンダーには、XML エクステンダー・サーバーの活動を記録するトレース機能があります。トレース機能は、IBM ソフトウェア・サポートの指示のもとでのみ使用してください。

トレース機能は、(XML エクステンダー・コンポーネントの開始や終了、XML エクステンダー・コンポーネントからのエラー・コード発信など) さまざまなイベントに関する情報をサーバー・ファイルの中に記録します。トレース機能は多数のイベントを記録するため、エラー状態の調査など必要な場合に限って使用してください。さらに、トレース機能を使用している間は、アクティブなアプリケーションの数を制限してください。アクティブなアプリケーションの数を制限すると、問題原因の正確な把握が容易になります。

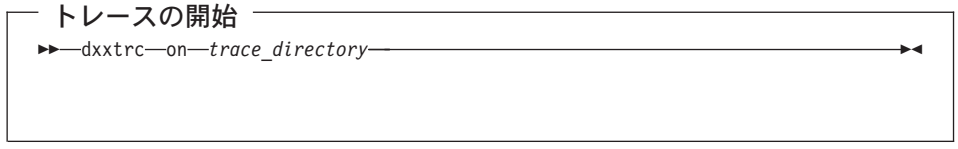
トレースを開始または停止するには、**dxxtrc** コマンドを使用します。このコマンドは、AIX、Windows NT、または Solaris サーバー上のコマンド行から発行できます。コマンドの発行には SYSADM、SYSCTRL、または SYSMINT 権限が必要です。

トレースの開始

目的

XML エクステンダー・サーバーの活動を記録します。トレースを開始するには、トレース・ファイルを入れるディレクトリーの名前を指定して、 **dxxtrc** の on オプションを適用します。トレースが開始されると、ファイル dxx*INSTANCE*.trc が、指定されたディレクトリーに保管されます。 *INSTANCE* は DB2INSTANCE の値です。それぞれの DB2 インスタンスごとに独自のログ・ファイルが作成されます。

形式



パラメーター

表 54. トレース・パラメーター

パラメーター	説明
trace_directory	dxx*INSTANCE*.trc が保管されるディレクトリーの名前。必須、デフォルトなし。

例

以下の例は、AIX 上でのインスタンス db2inst1 のトレースの開始を示しています。トレース・ファイル dxxdb2inst1.trc は、 /home/db2inst1/sqllib/log ディレクトリーに保管されます。

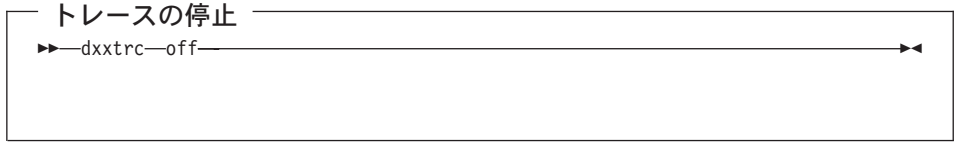
```
dxxtrc on /home/db2inst1/sqllib/log
```

トレースの停止

目的

トレースをオフにします。トレース情報はログに記録されなくなります。トレースの実行はパフォーマンスに影響するため、実稼働環境ではトレースをオフにすることを推奨します。

形式



例

この例は、トレース機能をオフにする場合を示します。

```
dxxtrc off
```

第5部 付録および後付け

付録A. DAD ファイル用の DTD

この節では、文書アクセス定義 (DAD) ファイル用の文書タイプ宣言 (DTD) について説明します。 DAD ファイル自体がツリー構造の XML 文書であり、DTD を必要とします。その DTD ファイル名は `dxxdad.dtd` です。 284ページの図13 は DAD ファイル用の DTD を示しています。このファイルのエレメントは以下の図に示されています。

```
<?xml encoding="US-ASCII"?>

  <!ELEMENT DAD (dtdid?, validation, (Xcolumn | Xcollection))>
  <!ELEMENT dtdid (#PCDATA)>
  <!ELEMENT validation (#PCDATA)>
  <!ELEMENT Xcolumn (table*)>
  <!ELEMENT table (column*)>
  <!ATTLIST table name CDATA #REQUIRED
                              key CDATA #IMPLIED
                              orderBy CDATA #IMPLIED>
  <!ELEMENT column EMPTY>
  <!ATTLIST column
                      name CDATA #REQUIRED
                      type CDATA #IMPLIED
                      path CDATA #IMPLIED
                       multi_occurrence CDATA #IMPLIED>
  <!ELEMENT Xcollection (SQL_stmt?, objids?, prolog, doctype, root_node)>
  <!ELEMENT SQL_stmt (#PCDATA)>
  <!ELEMENT objids (column+)>
  <!ELEMENT prolog (#PCDATA)>
  <!ELEMENT doctype (#PCDATA | RDB_node)*>
  <!ELEMENT root_node (element_node)>
  <!ELEMENT element_node (RDB_node*,
                                       attribute_node*,
                                       text_node?,
                                       element_node*,
                                       namespace_node*,
                                       process_instruction_node*,
                                       comment_node*)>
  <!ATTLIST element_node
                     name CDATA #REQUIRED
                     ID CDATA #IMPLIED
                     multi_occurrence CDATA "NO"
                     BASE_URI CDATA #IMPLIED>
  <!ELEMENT attribute_node (column | RDB_node)>
  <!ATTLIST attribute_node
                     name CDATA #REQUIRED>
  <!ELEMENT text_node (column | RDB_node)>
  <!ELEMENT RDB_node (table+, column?, condition?)>
  <!ELEMENT condition (#PCDATA)>
  <!ELEMENT comment_node (#PCDATA)>
  <!ELEMENT namespace_node (EMPTY)>
  <!ATTLIST namespace_node
                      name CDATA #IMPLIED
                     value CDATA #IMPLIED>
  <!ELEMENT process_instruction_node (#PCDATA)>
```

図 13. 文書アクセス定義 (DAD) 用の DTD

DAD ファイルの主なエレメントは次の 4 つです。

- DTDID
- validation

- Xcolumn
- Xcollection

Xcolumn および Xcollection には、 XML データを DB2 のリレーショナル表にマッピングする際に使われる子エレメントおよび属性があります。以下のリストは、主なエレメントおよびそれらの子エレメントや属性について説明しています。構文例は、284ページの図13 から引用しています。

DTDID エレメント

DTD_REF 表に保管された DTD の ID を指定します。 DTDID は XML 文書の妥当性検査を行う、または XML コレクション表と XML 文書間のマッピングに使われる DTD を示します。 XML コレクションには、DTDID の指定は必須です。 XML 列の場合は任意指定であり、エレメントや属性の索引付けのためにサイド表を作成したい場合、または入力 XML 文書の妥当性検査をする場合にのみ必要です。 DTDID は、 XML 文書の doctype で指定された SYSTEM ID と同じでなければなりません。

構文: `<!ELEMENT dtdid (#PCDATA)>`

validation エレメント

DAD 用の DTD を使って XML 文書を妥当性検査するかどうかを示します。 YES を指定する場合、DTDID も指定しなければなりません。

構文: `<!ELEMENT validation(#PCDATA)>`

Xcolumn エレメント

XML 列の索引付け体系を定義します。 1 つまたは複数の表を含む場合もあります。

構文: `<!ELEMENT Xcolumn (table*)>` Xcolumn には 1 つの子エレメント table があります。

table エレメント

XML 列に保管される文書のエレメントまたは属性の索引付け用に作成される、 1 つまたは複数のリレーショナル表を定義します。

構文:

```
<!ELEMENT table (column+)>
 <!ATTLIST table name CDATA #REQUIRED
key CDATA #IMPLIED
orderBy CDATA #IMPLIED>
```

table エレメントの属性は次の 1 つです。

name 属性

サイド表の名前を指定します。

table エレメントの子エレメントは次の 1 つです。

key 属性

表のただ 1 つの基本キーです。

orderBy 属性

XML 文書を生成する際、複数出現するエレメント・テキストまたは属性値の順序を決定する列の名前。

column エレメント

指定されたタイプのロケーション・パスの値を含む表の列を指定します。

構文:

```
<!ATTLIST column
                    name CDATA #REQUIRED
                    type  CDATA #IMPLIED
                    path  CDATA #IMPLIED
                    multi_occurrence CDATA #IMPLIED>
```

column エレメントには以下の属性があります。

name 属性

列の名前を指定します。これは、エレメントまたは属性を識別するロケーション・パスの別名です。

type 型属性

列のデータ・タイプを定義します。任意の SQL データ・タイプを指定できます。

path 属性

XML エレメントまたは属性のロケーション・パスを示します。これは、表 3.1.a (fix link) で指定されているような単純ロケーション・パスでなければなりません。

multi_occurrence 属性

1 つの XML 文書内でこのエレメントまたは属性が 2 度以上出現できるかどうかを指定します。値は YES または NO です。

Xcollection

XML 文書とリレーショナル表からなる XML コレクション間とのマッピングを定義します。

構文: <!ELEMENT Xcollection(SQL_stmt*, prolog, doctype, root_node)> Xcollection には次のような子エレメントがあります。

SQL_stmt

コレクションの定義のために XML エクステンダーが使用する SQL ステートメントを指定します。そのステートメントは XML コレクション表から XML データを選択し、そのデータを使用してコレクション内に XML 文書を生成します。このエレメントの値は、有効な SQL ステートメントでなければなりません。これは合成の場合のみ使用され、指定できる SQL_stmt の数は 1 つだけです。分解の場合には、必要な表の作成および挿入を行うために複数の SQL_stmt 値を指定できます。

構文: `<!ELEMENT SQL_stmt  #PCDATA >`

prolog

XML プロローグのテキスト。コレクション内のすべての文書に同じプロローグが提供されます。 prolog の値は変更されません。

構文: `<!ELEMENT prolog  #PCDATA>`

doctype

XML 文書タイプ定義のテキストを定義します。

構文: `<!ELEMENT doctype  #PCDATA | RDB_node> doctype` は以下のいずれかの方法で指定できます。

- 明示的な値を定義する。この値はコレクション内のすべての文書に提供されます。
- 分解を使用している場合、子エレメント RDB_node を指定します。これは表の列データにマップされ、表の列データとして保管されます。

doctype の子エレメントは次の 1 つです。

RDB_node

まだ実装されていません。

root_node

仮想ルート・ノードを定義します。 root_node には子エレメント element_node が必要です。これは一度だけ使用できます。 root_node の下の element_node は、実際には XML 文書の root_node です。

構文: `<!ELEMENT root_node(element_node)>`

element_node

XML エレメントを表します。これは、コレクション用に指定された DTD で定義されていなければなりません。 RDB_node マッピングの場合、ルート element_node には、 (それ自体とその子ノードの) XML データを含むすべての表を指定する RDB_node が必要です。また、ルート element_node には、任意数の attribute_nodes と子 element_nodes、および 1 つまでの text_nodes を含めることもできます。ルート・エレメントではないエレメントの場合、RDB_node は不要です。

構文:

element_node は、次の子エレメントによって定義されます。

RDB_node

(任意指定) XML データ用の表、列、および条件を指定します。エレメント用の RDB_node は、RDB_node マッピングのためにのみ定義が必要です。ここでは、1 つまたは複数の表を指定しなければなりません。エレメントの内容が text_node で指定されているため、列は不要です。条件は、DTD および照会条件に応じて任意指定です。

子ノード

(任意指定) element_node には、以下の子ノードも指定できます。

element_node

現在の XML エレメントの子エレメントを表します。

attribute_node

現在の XML エレメントの属性を表します。

text_node

現在の XML エレメントの CDATA テキストを表します。

attribute_node

XML 属性を表します。これは、XML 属性とリレーショナル表の列データの間のマッピングを定義するノードです。

構文:

attribute_node には name 属性の定義が必要であり、あわせて column または RDB_node 子エレメントのいずれかが必要です。 attribute_node には以下の属性があります。

name 属性の名前。

attribute_node には以下の子エレメントがあります。

Column
SQL マッピングに使用されます。この列は、SQL_stmt の SELECT 文節内で指定されていなければなりません。

RDB_node
RDB_node マッピングに使用されます。このノードは、この属性とリレーショナル表内の列データ間のマッピングを定義します。表および列の指定は必須です。条件は任意指定です。

text_node
XML エレメントのテキストの内容を表します。これは、XML エレメントの内容とリレーショナル表の列データ間のマッピングを定義するノードです。

構文: これは、以下の column または RDB_node 子エレメントによって定義する必要があります。

Column
SQL マッピングに必要。この場合、SQL_stmt の SELECT 文節内にこの列が指定されていなければなりません。

RDB_node
RDB_node マッピングに必要。このノードは、このテキスト内容とリレーショナル表内の列データ間のマッピングを定義します。 table および column の指定は必須です。条件は任意指定です。

付録B. サンプル

この付録では、本書の例で使われているサンプル・オブジェクトを示します。

- 『XML DTD』
- 292ページの『XML 文書: getstart.xml』
- 292ページの『文書アクセス定義ファイル』
  - 293ページの『DAD ファイル: XML 列』
  - 294ページの『DAD ファイル: XML コレクション - SQL マッピング』
  - 295ページの『DAD ファイル: XML - RDB_node マッピング』

XML DTD

以下の DTD は、本書全体を通して参照されている `getstart.xml` 文書 (292ページの図15 を参照) に使用されます。

```
<!xml encoding="US-ASCII"?>
<!ELEMENT Order (Customer, Part+)>
<!ATTLIST Order key CDATA #REQUIRED>
<!ELEMENT Customer (Name, Email)>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Email (#PCDATA)>
<!ELEMENT Part (key,Quantity,ExtendedPrice,Tax, Shipment+)>
<!ELEMENT key (#PCDATA)>
<!ELEMENT Quantity (#PCDATA)>
<!ELEMENT ExtendedPrice (#PCDATA)>
<!ELEMENT Tax (#PCDATA)>
<!ATTLIST Part color CDATA #REQUIRED>
<!ELEMENT Shipment (ShipDate, ShipMode)>
<!ELEMENT ShipDate (#PCDATA)>
<!ELEMENT ShipMode (#PCDATA)>
```

図 14. サンプル XML DTD: getstart.dtd

XML 文書: getstart.xml

以下の XML 文書 getstart.xml は、本書全体を通して例として使われているサンプル XML 文書です。この文書には購入オーダーを表す XML タグが含まれています。

```
<?xml version="1.0"?>
<!DOCTYPE Order SYSTEM "c:¥dxx¥samples¥dtd¥getstart.dtd">
<Order key="1">
  <Customer>
    <Name>American Motors</Name>
    <Email>parts@am.com</Email>
  </Customer>
  <Part color="black ">
    <key>68</key>
    <Quantity>36</Quantity>
    <ExtendedPrice>34850.16</ExtendedPrice>
    <Tax>6.000000e-02</Tax>
    <Shipment>
      <ShipDate>1998-08-19</ShipDate>
      <ShipMode>BOAT  </ShipMode>
    </Shipment>
    <Shipment>
       <ShipDate>1998-08-19</ShipDate>
       <ShipMode>AIR   </ShipMode>
    </Shipment>
  </Part>
  <Part color="red   ">
    <key>128</key>
    <Quantity>28</Quantity>
    <ExtendedPrice>38000.00</ExtendedPrice>
    <Tax>7.000000e-02</Tax>
    <Shipment>
      <ShipDate>1998-12-30</ShipDate>
      <ShipMode>TRUCK </ShipMode>
    </Shipment>
  </Part>
</Order>
```

図 15. サンプル XML 文書: getstart.xml

文書アクセス定義ファイル

以下のセクションでは、XML 列または XML コレクション・アクセス・モードを使用して XML データを DB2 リレーショナル表にマップする文書アクセス定義 (DAD) ファイルを示します。

- 293ページの『DAD ファイル: XML 列』

- 294ページの『DAD ファイル: XML コレクション - SQL マッピング』は、 SQL マッピングを使った XML コレクションの DAD ファイルを示しています。
- 295ページの『DAD ファイル: XML - RDB_node マッピング』 は、RDB_node マッピングを使った XML コレクションの DAD ファイルを示しています。

DAD ファイル: XML 列

この DAD ファイルは XML 列のマッピングを示し、XML データを保管する表、サイド表、および列を定義しています。

```
<?xml version="1.0"?>
<!DOCTYPE Order SYSTEM "c:¥dxx¥dad.dtd">
<DAD>
   <dtdid>c:¥dxx¥samples¥dtd¥getstart.dtd</dtdid>
   <validation>YES</validation>

   <Xcolumn>
     <table name="order_side_tab">
         <column name="order_key"
            type="integer"
            path="/Order/@key"
            multi_occurrence="NO"/>
         <column name="customer"
            type="varchar(50)"
            path="/Order/Customer/Name"
            multi_occurrence="NO"/>
     </table>
     <table name="part_side_tab">
         <column name="price"
            type="decimal(10,2)"
            path="/Order/Part/ExtendedPrice"
            multi_occurrence="YES"/>
     </table>
     <table name="ship_side_tab">
         <column name="date"
            type="DATE"
            path="/Order/Part/Shipment/ShipDate"
            multi_occurrence="YES"/>
     </table>

   </Xcolumn>

</DAD>
```

図 16. XML 列用のサンプル DAD ファイル

DAD ファイル: XML コレクション - SQL マッピング

この DAD ファイルには、XML データを保管する DB2 表、列、および条件を指定した SQL ステートメントが含まれています。

```
<?xml version="1.0"?>
<!DOCTYPE DAD SYSTEM "c:¥dxx¥dtd¥dad.dtd">
<DAD>
<validation>NO</validation>
<Xcollection>
<SQL_stmt>SELECT o.order_key, customer_name, customer_email, p.part_key, color, quantity,
   price, tax, ship_id, date, mode from order_tab o, part_tab p,
   table (select substr(char(timestamp(generate_unique())),16)
     as ship_id, date, mode, part_key from ship_tab) s
         WHERE o.order_key = 1 and
               p.price > 20000 and
               p.order_key = o.order_key and
               s.part_key = p.part_key
         ORDER BY order_key, part_key, ship_id</SQL_stmt>
<prolog>?xml version="1.0"?</prolog>
<doctype>!DOCTYPE Order SYSTEM "c:¥dxx¥samples¥dtd¥getstart.dtd"</doctype>
```

図 17. SQL マッピングを使用した XML コレクション用のサンプル DAD ファイル (1/2)

```
<root_node>
<element_node name="Order">
  <attribute_node name="key">
    <column name="order_key"/>
  </attribute_node>
  <element_node name="Customer">
    <element_node name="Name">
      <text_node><column name="customer_name"/></text_node>
    </element_node>
    <element_node name="Email">
      <text_node><column name="customer_email"/></text_node>
    </element_node>
  </element_node>
  <element_node name="Part">
    <attribute_node name="color">
      <column name="color"/>
    </attribute_node>
    <element_node name="key">
      <text_node><column name="part_key"/></text_node>
    </element_node>
    <element_node name="Quantity">
      <text_node><column name="quantity"/></text_node>
    </element_node>
    <element_node name="ExtendedPrice">
      <text_node><column name="price"/></text_node>
    </element_node>
    <element_node name="Tax">
      <text_node><column name="tax"/></text_node>
    </element_node>
    <element_node name="Shipment" multi_occurence="YES">
      <element_node name="ShipDate">
        <text_node><column name="date"/></text_node>
      </element_node>
      <element_node name="ShipMode">
        <text_node><column name="mode"/></text_node>
      </element_node>
    </element_node>
  </element_node>
</element_node>
</root_node>
</Xcollection>
</DAD>
```

図 17. SQL マッピングを使用した XML コレクション用のサンプル DAD ファイル (2/2)

DAD ファイル: XML - RDB_node マッピング

この DAD ファイルは、XML データを保管する DB2 表、列、および条件を定義するために <RDB_node> エレメントを使用しています。

```
<?xml version="1.0"?>
<!DOCTYPE Order SYSTEM "c:¥dxx¥dtd¥dad.dtd">
<DAD>
   <dtdid>c:¥dxx¥samples¥dtd¥getstart.dtd</dtdid>
   <validation>YES</validation>
<Xcollection>
      <prolog>?xml version="1.0"?</prolog>
      <doctype>!DOCTYPE Order SYSTEM "c:¥dxx¥samples¥dtd¥getstart.dtd"</doctype>
      <root_node>
         <element_node name="Order">
           <RDB_node>
              <table name="order_tab"/>
              <table name="part_tab"/>
              <table name="ship_tab"/>
              <condition>
                order_tab.order_key = part_tab.order_key AND
                part_tab.part_key = ship_tab.part_key
              </condition>
           </RDB_node>
           <attribute_node name="key">
              <RDB_node>
                 <table name="order_tab"/>
                 <column name="order_key"/>
              </RDB_node>
           </attribute_node>
           <element_node name="Customer">
              <text_node>
                 <RDB_node>
                    <table name="order_tab"/>
                    <column name="customer"/>
                 </RDB_node>
              </text_node>
           </element_node>
           <element_node name="Part">
              <RDB_node>
                 <table name="part_tab"/>
                 <table name="ship_tab"/>
                 <condition>
                    part_tab.part_key = ship_tab.part_key
                 </condition>
              </RDB_node>
              <attribute_node name="key">
                 <RDB_node>
                    <table name="part_tab"/>
                    <column name="part_key"/>
                 </RDB_node>
              </attribute_node>
```

図 18. RDB_node マッピングを使用した XML コレクション用のサンプル DAD ファイル (1/3)

```
<element_node name="Quantity">
   <text_node>
      <RDB_node>
         <table name="part_tab"/>
         <column name="quantity"/>
      </RDB_node>
   </text_node>
</element_node>
<element_node name="ExtendedPrice">
   <text_node>
      <RDB_node>
         <table name="part_tab"/>
            <column name="price"/>
            <condition>
               price > 2500.00
            </condition>
      </RDB_node>
   </text_node>
</element_node>
<element_node name="Tax">
   <text_node>
      <RDB_node>
         <table name="part_tab"/>
         <column name="tax"/>
      </RDB_node>
   </text_node>
</element_node>
```

図 18. RDB_node マッピングを使用した XML コレクション用のサンプル DAD ファイル (2/3)

```
            <element_node name="shipment">
               <RDB_node>
                  <table name="ship_tab"/>
                  <condition>
                     part_key = part_tab.part_key
                  </condition>
               </RDB_node>
               <element_node name="ShipDate">
                  <text_node>
                     <RDB_node>
                        <table name="ship_tab"/>
                        <column name="date"/>
                        <condition>
                           date > "1966-01-01"
                        </condition>
                     </RDB_node>
                  </text_node>
               </element_node>
               <element_node name="ShipMode">
                  <text_node>
                     <RDB_node>
                        <table name="ship_tab"/>
                        <column name="mode"/>
                     </RDB_node>
                  </text_node>
               </element_node>
               <element_node name="Comment">
                  <text_node>
                     <RDB_node>
                        <table name="ship_tab"/>
                        <column name="comment"/>
                     </RDB_node>
                  </text_node>
               </element_node>
             </element_node> <! -- end of element Shipment>
           </element_node> <! -- end of element Part --->
         </element_node> <! -- end of element Order --->
       </root_node>
  </Xcollection>
</DAD>
```

図 18. RDB_node マッピングを使用した XML コレクション用のサンプル DAD ファイル (3/3)

付録C. コード・ページに関する考慮事項

XML 文書および他の関連ファイルが、そのファイルにアクセスするそれぞれのクライアントまたはサーバーに対して、適正にエンコードされているのを確認することは重要です。したがって、ファイルの処理時の XML エクステンダー前提事項と、それがコード・ページ変換を処理する方法を理解することは重要です。主な考慮事項は以下のとおりです。

- DB2 から XML 文書を検索するクライアントの実際のコード・ページが、その文書のエンコードと一致していなければなりません。
- 文書が XML 構文解析プログラムによって処理される場合、 XML 文書のエンコード宣言も、そのエンコードと一貫性がなければなりません。

続く節では、これらの考慮事項に関係する事柄、生じる可能性がある問題への取り得る対応策、および文書をクライアントからサーバーまたはデータベースに渡すときに XML エクステンダーおよび DB2 がコード・ページをサポートする方法について説明します。

用語

この節では、以下の用語が使用されています。

文書エンコード
XML 文書の実コード・ページ。

文書エンコードの宣言
XML 宣言で指定されるコード・ページの名前。たとえば次のようにします。

```
<?xml version="1.0" encoding="ibm037"?>
```

整合文書
文書エンコードが、文書エンコードの宣言のコード・ページと一致する文書。

不整合文書
文書エンコードが、文書エンコードの宣言のコード・ページと一致しない文書。

DB2CODEPAGE レジストリー (環境) 変数
データベース・クライアント・アプリケーションから DB2 に提示されるデータのコード・ページを指定します。この変数が設定されていなけ

れば、DB2 はクライアントのコード・ページを、クライアントのオペレーティング・システムのロケールから取得します。 DB2 に対しては、この値が設定されていれば、その値はクライアントのオペレーティング・システムのロケールを変更します。

クライアントのコード・ページ
アプリケーションのコード・ページ。 DB2CODEPAGE 変数が設定された場合、クライアントのコード・ページは DB2CODEPAGE の値です。設定されない場合、クライアントのコード・ページはオペレーティング・システムのロケールです。

サーバー・コード・ページ、またはサーバーのオペレーティング・システムのロケールのコード・ページ
DB2 データベースがインストールされているオペレーティング・システムのロケール。

データベースのコード・ページ
データベース作成時に決定された、保管データのエンコード。 USING CODESET 文節で明示的に指定されていない場合、この値は、デフォルトとしてサーバーのオペレーティング・システムのロケールに設定されます。

DB2 および XML エクステンダーのコード・ページの前提事項

DB2 は XML 文書を送受信するときにエンコード宣言を検査しません。ただし、クライアントのコード・ページを調べて、それがデータベースのコード・ページに一致するかどうかを確かめます。一致しない場合 DB2 は、以下のもののコード・ページに一致するようにその XML 文書内のデータを変換します。

- 文書または文書の一部をデータベース表にインポートする場合は、データベース
- 文書を 1 つまたは複数のデータベース表に分解する場合は、データベース・サーバー
- 文書をデータベースからエクスポートする場合、および文書をクライアント用に提示する場合は、クライアント
- サーバーのファイル・システム上のファイルにデータを戻す、UDF があるファイルを処理する場合は、サーバー

XML 文書がデータベースにインポートされるとき、一般にそれは XML 列に保管される XML 文書としてまたは XML コレクションの分解用にインポートされます。この場合は、エレメントおよび属性内容は DB2 データとして保管

されます。文書がインポートされるとき、 DB2 は文書エンコードをデータベースのエンコードに変換します。 DB2 は、文書のコード・ページを、下記の表の 『ソース・コード・ページ』 列で指定されたコード・ページであると想定します。 表55 は、XML 文書のインポート時に DB2 が実行する変換を要約しています。

表 55. XML ファイルのデータベースへのインポート時の、UDF およびストアード・プロシージャーの使用

処理メソッド	変換用のソース・コード・ページ	変換用の宛先コード・ページ	コメント
DTD ファイルの DTD_REF 表への挿入	クライアントのコード・ページ	データベースのコード・ページ	
DAD ファイルをインポートする、列、コレクションのストアード・プロシージャー、または管理コマンドを使用可能にします	クライアントのコード・ページ	データベースのコード・ページ	
UDF: • XMLVarcharFromFile() • XMLCLOBFromFile() • Content(): XMLFILE から取り出して CLOB に入れます	サーバーのコード・ページ	データベースのコード・ページ	
分解ストアード・プロシージャー	クライアントのコード・ページ	データベースのコード・ページ	分解する文書は、クライアントのコード・ページで作成されていると想定します。分解されたデータは、データベースのコード・ページで表に保管されます。

XML 文書のデータベースからのエクスポートは、一般に、クライアントの文書提示要求や、XML 文書内容の照会に応じて、または DB2 データからの文書の構成時に実行されます。文書のエクスポート時に、DB2 は文書エンコードを、要求の発信元およびデータの提示先に応じて、クライアントまたはサーバーで使用しているエンコードに変換します。 302ページの表56 は、XML 文書のエクスポート時に DB2 が実行する変換を要約しています。

表 56. XML ファイルのデータベースからのエクスポート時の、UDF およびストアード・プロシージャーの使用

処理メソッド	変換	コメント
UDF • XMLFileFromVarchar() • XMLFileFromCLOB()	データがクライアントに提示される場合は、データベースのコード・ページからクライアントのコード・ページへ	
UDF • Content(): XMLVARCHAR から取り出して外部サーバー・ファイルに入れます	データベースのコード・ページからサーバー・コード・ページへ	
構成ストアード・プロシージャー: 作成された表は、結果表に保管されます。これは照会またはエクスポートが可能です。	結果セットがクライアントに提示される場合は、データベースのコード・ページからクライアントのコード・ページへ	文書の構成時に、XML エクステンダーは、 DAD 内のタグによって指定されたエンコード宣言を、新たに作成した文書にコピーします。これは提示される場合に、クライアントのコード・ページに一致していなければなりません。

エンコード宣言の考慮事項

エンコード宣言は、XML 文書のエンコードを宣言するもので、XML 宣言ステートメント上に置かれます。 XML エクステンダーの使用時には、ファイルの保管場所に応じて、文書のエンコードが、クライアントまたはサーバーのコード・ページと必ず一致するようにすることが重要です。

正しいエンコード宣言

いくつかの指針に従えば、XML 文書で任意のエンコード宣言を使用することができます。この節では、これらの指針を定義し、サポートされるエンコード宣言を説明します。

推奨される移植可能な XML データ用のエンコードは UTF-8 および UTF-16 で、XML の仕様に応じて異なります。これらのエンコードを使用すれば、ご使用のアプリケーションは、異なる企業間でも相互運用可能になります。 303 ページの表57 にリストされているエンコードを使用する場合、ご使用のアプリ

ケーションは、さまざまな IBM オペレーティング・システム間で移植できる可能性が高くなります。他のエンコードを使用する場合、ご使用のデータが移植できる可能性は低くなります。

すべてのオペレーティング・システムで、以下のエンコード宣言がサポートされます。以下のリストは、各列の意味を説明しています。

- **エンコード**は、XML 宣言で使用されるエンコード・ストリングを示しています。
- **OS** は、その上で DB2 が特定のコード・ページをサポートするオペレーティング・システムを示しています。
- **コード・ページ**は、特定のエンコードに関連した IBM 定義のコード・ページを示しています。

表 57. XML エクステンダーによってサポートされるエンコード宣言

区分	エンコード	OS	コード・ページ
Unicode	UTF-8	AIX、SUN、Linux	1208
	UTF-16	AIX、SUN、Linux	1200
ASCII	iso-8859-1	AIX、Linux、Sun	819
	ibm-1252	Windows NT	1252
	iso-8859-2	AIX、Linux、Sun	912
	iso-8859-5	AIX、Linux	915
	iso-8859-6	AIX	1089
	iso-8859-7	AIX、Linux	813
	iso-8859-8	AIX、Linux	916
	iso-8859-9	AIX、Linux	920
MBCS	gb2312	Windows NT	1386
	ibm-932、shift_jis78	AIX	932
	Shift_JIS	AIX 4.3 のみ、Windows NT	943
	IBM-eucCN	AIX、Sun	1383
	IBM-eucJP、EUC-JP	AIX、Linux、Sun	954、33722
	euc-tw、IBM-eucTW	AIX、Sun	964
	euc-kr、IBM-eucKR	AIX	970
	big5	AIX、Sun、Windows NT	950

エンコード・ストリングは、文書の宛先のコード・ページと互換性がなければなりません。文書がサーバーからクライアントに戻される場合、そのエンコード・ストリングは、クライアントのコード・ページと互換性がなければなりません。非互換のエンコードの使用の結果については、『整合性があるエンコードおよびエンコード宣言』を参照してください。 XML エクステンダーで使用する XML 構文解析プログラムがサポートするコード・ページのリストについては、以下の URL を参照してください。

http://www.ibm.com/software/data/db2/extenders/xmlext/moreinfo/encoding.html

整合性があるエンコードおよびエンコード宣言

XML 文書が他のシステムで処理または交換される場合には、エンコード宣言が、文書の実際のエンコードと対応していることが重要です。文書のエンコードを、クライアントのものと確実に整合させることが大切です。宣言内に指定されているもの以外のエンコード宣言がエンティティーに入っていると、構文解析プログラムなどの XML ツールはエラーを生成するからです。

305ページの図19 は、文書エンコードおよび宣言エンコードと整合性があるコード・ページを持つクライアントを示しています。

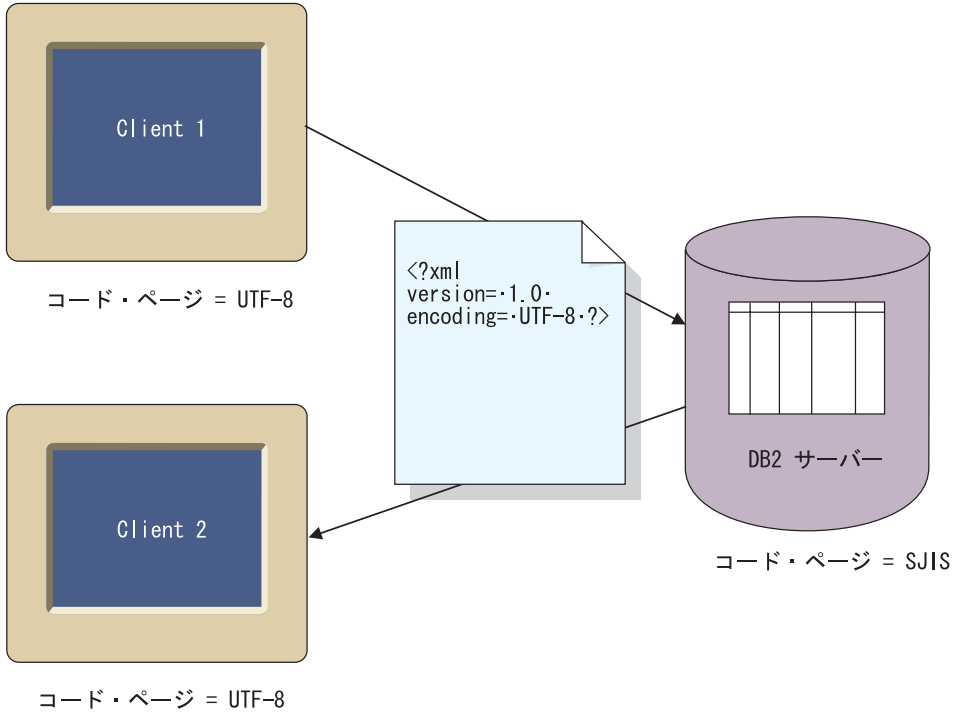


図 19. クライアントでのコード・ページの一致

別々のコード・ページを使用すると、次のような状況が生じることがあります。

- 変換時にデータが消失する可能性があります。ソース・コード・ページが Unicode で、ターゲットのコード・ページが Unicode でない場合は特にそうです。 Unicode には、文字変換の完全なセットが入っています。ファイルを、UTF-8 から、文書内の一部の文字をサポートしないコード・ページに変換する場合、変換中にデータが消失する可能性があります。
- 宣言エンコードとは異なるコード・ページを使用するクライアントがその文書を取り出すと、その XML 文書の宣言エンコードは、実際の文書エンコードと整合しなくなってしまう場合があります。

306ページの図20 は、クライアントのコード・ページが不整合である環境を示しています。

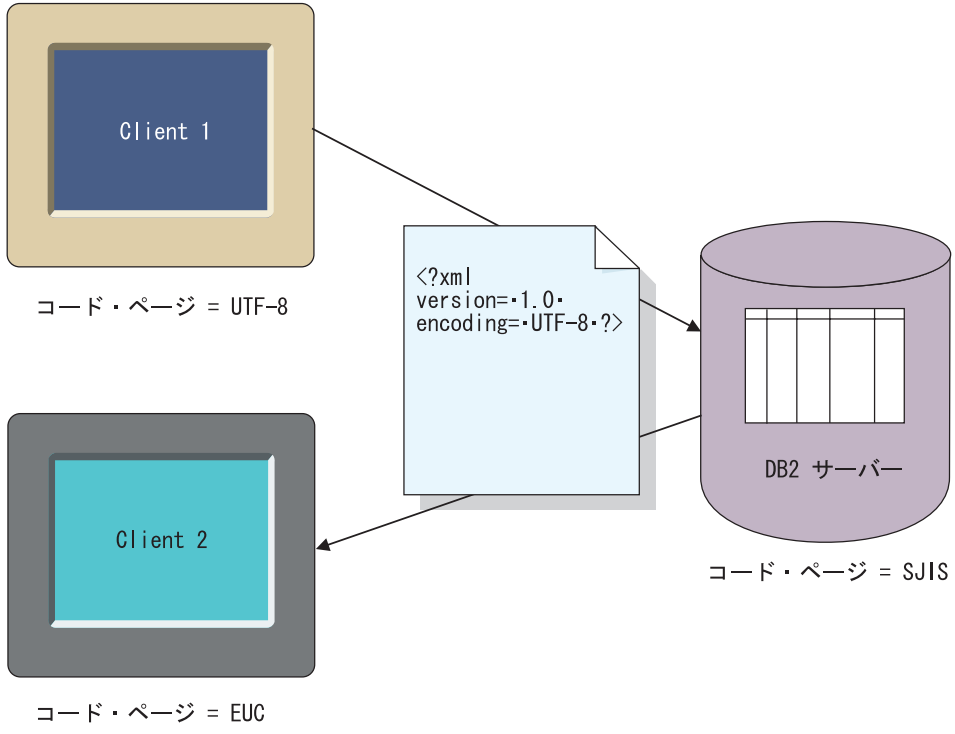


図 20. クライアント間でコード・ページが不整合

Client2 は、文書を EUC で受け取っていますが、文書のエンコード宣言は UTF-8 になります。

エンコードの宣言

エンコード宣言のデフォルト値は UTF-8 なので、エンコード宣言がない場合、文書は UTF-8 であることを意味します。

エンコード値を宣言するには、次のようにします。

XML 文書宣言内で、クライアントのコード・ページ名を付けてエンコード宣言を指定します。たとえば次のようにします。

```
<?xml version="1.0" encoding="UTF-8" ?>
```

変換のシナリオ

XML エクステンダーは、以下の場合に XML 文書を処理します。

- XML 列ストレージおよびアクセス方式を使用する、XML 列データの保管および検索
- XML 文書の構成および分解

文書は、クライアントまたはサーバーからデータベースへ渡されるときに、コード・ページ変換の処理をされます。たいていの場合、XML 文書の不整合または損傷は、クライアント、サーバー、およびデータベースのコード・ページからの変換中に発生します。文書のエンコード宣言を選択するときには、データベースから文書をインポートまたはエクスポートできるクライアントおよびサーバーの計画時と同様、前述の表で説明した変換、および下記のシナリオを考慮してください。

以下のシナリオは、起きる可能性がある一般的な変換シナリオです。

シナリオ 1: このシナリオは、エンコードに整合性があり、DB2 変換はなく、文書がサーバーからインポートされるという構成です。文書エンコード宣言は UTF-8、サーバーは UTF-8、およびデータベースは UTF-8 です。

1. 文書は XMLClobfromFile UDF を使用して DB2 にインポートされます。
2. 文書はサーバーに抽出されます。
3. サーバーのコード・ページとデータベースのコード・ページは同じであるので、DB2 は文書を変換する必要がありません。エンコードと宣言とは整合性があります。

シナリオ 2: このシナリオは、エンコードに整合性があり、DB2 変換があり、文書がサーバーからインポートされてクライアントにエクスポートされるという構成です。文書のエンコードおよび宣言は SJIS、クライアントのコード・ページは SJIS、サーバーおよびデータベースのコード・ページは UTF-8 です。

1. 文書はサーバーから XMLClobfromfile UDF を使用して DB2 にインポートされます。
2. DB2 は文書を SJIS から変換し、UTF-8 で保管します。エンコード宣言とエンコードとは、データベース内で不整合となっています。
3. SJIS を使用するクライアントが、Web ブラウザーで表示する文書を要求します。
4. DB2 は文書を SJIS (クライアントのコード・ページ) に変換します。これで、文書エンコードと宣言とは、クライアント内で整合することになります。

シナリオ 3: このシナリオは、エンコードに整合性がなく、DB2 変換があり、文書がサーバーからインポートされてクライアントにエクスポートされるとい

う構成です。文書エンコード宣言は、着信文書については SJIS です。サーバーのコード・ページは SJIS、クライアントおよびデータベースのコード・ページは UTF-8 です。

1. 文書はストレージ UDF を使用してデータベースにインポートされます。
2. DB2 は文書を SJIS から UTF-8 に変換します。エンコードと宣言とは不整合です。
3. UTF-8 コード・ページを使用するクライアントが、Web ブラウザーで表示する文書を要求します。
4. クライアントとデータベースのコード・ページとが同じなので、DB2 は変換しません。
5. 宣言が SJIS でエンコードが UTF-8 なので、文書エンコードと宣言とが不整合です。文書は XML 構文解析プログラムまたは他の XML 処理ツールによって処理できません。

シナリオ 4: このシナリオは、データが消失し、DB2 変換があり、文書が UTF-8 サーバーからインポートされるという構成です。文書エンコード宣言は UTF-8、サーバーは UTF-8、データベースは SJIS です。

1. 文書は XMLClobfromFile UDF を使用して DB2 にインポートされます。
2. DB2 はエンコードを SJIS に変換します。文書のインポート時に、UTF-8 で表示された文字は SJIS では表示されないことがあるため、データベースに保管された文書は破壊される可能性があります。

シナリオ 5: このシナリオは、Windows NT の制限がある構成です。 Windows NT 上では、オペレーティング・システムのロケールは UTF-8 に設定できませんが、 DB2 では、`db2set DB2CODEPAGE=1208` を使用すれば、クライアントはコード・ページを UTF-8 に設定できます。このシナリオでは、クライアントとサーバーとは同じマシン上にあります。クライアントは UTF-8 ですが、サーバーは UTF-8 に設定できず、そのコード・ページは 1252 です。文書は 1252 としてエンコードされ、エンコード宣言は ibm-1252 です。データベースのコード・ページは UTF-8 です。

1. 文書はストレージの UDF によってサーバーからインポートされ、1252 から 1208 に変換されます。
2. 文書はクライアントによって Content() UDF を使用して DB2 からエクスポートされます。
3. クライアントがサーバーと同じシステム上にあって 1208 に設定されているため、クライアントが 1208 を予期していたとしても、DB2 は文書を UTF-8 から 1252 に変換します。

不整合の XML 文書の防止

上記の節では、XML 文書はどのように不整合エンコードを持つ可能性があるのか、つまりどのようにエンコード宣言が文書のエンコードと矛盾するかについて解説しました。不整合エンコードが原因で、データが消失したり、XML 文書が使用できなくなる可能性があります。

XML 文書のエンコードをクライアントのコード・ページと確実に整合させるため、文書を構文解析プログラムなどの XML 処理プログラムに渡す前に、以下の推奨事項の 1 つを実行してください。

- XML エクステンダー UDF を使用してデータベースから文書をエクスポートするときには、以下の技法の 1 つを試行します (前提事項: XML エクステンダーはファイルを、サーバーのコード・ページで、サーバー上のファイル・システムにエクスポートしています)。
  - 宣言エンコードのコード・ページに文書を変換する
  - 指定変更機能がツールに備わっていれば、宣言エンコードを指定変更する
  - エクスポートされた文書のエンコード宣言を、文書の実際のエンコード (つまりサーバーのコード・ページ) に手動で変更する
- XML エクステンダーのストアード・プロシージャーを使用してデータベースから文書をエクスポートするときには、以下の技法の 1 つを試行します (前提事項: クライアントは、構成済みの文書が保管されている結果表を照会しているとします)。
  - 宣言エンコード・コード・ページに文書を変換する
  - 指定変更機能がツールに備わっていれば、宣言エンコードを指定変更する
  - 結果表を照会する前に、クライアントに環境変数 DB2CODEPAGE を設定させて、クライアントのコード・ページを、XML 文書のエンコード宣言と互換性があるコード・ページに強制的に変換する
  - エクスポートされた文書のエンコード宣言を、文書の実際のエンコード (つまりクライアントのコード・ページ) に手動で変更する
- **Unicode および Windows NT の使用時の制限:** Windows NT 上では、オペレーティング・システムのロケールを UTF-8 に設定することはできません。文書のインポートまたはエクスポート時には、以下の指針を使用します。
  - UTF-8 でエンコードされたファイルおよび DTD をインポートするには、以下を使用して、クライアントのコード・ページを UTF-8 に設定します。

    ```
    db2set DB2CODEPAGE=1208
    ```

この技法は以下の場合に使用します。

- DTD を db2xml.DTD_REF 表に挿入する
- 列またはコレクションを使用可能にする
- ストアード・プロシージャーを分解する

– Content() または XMLxxxfromFile UDF を使用して XML 文書をインポートするときには、サーバーのオペレーティング・システムのロケールのコード・ページでエンコードしなければなりませんが、それを UTF-8 にすることはできません。
– XML ファイルをデータベースからエクスポートするときには、以下のコマンドを使用して、クライアントのコード・ページを設定し、 DB2 が結果データを UTF-8 でエンコードするようにします。

```
db2set DB2CODEPAGE=1208
```

この技法は以下の場合に使用します。

- 構成後に結果表を照会する
- 抽出 UDF を使用して XML 列からデータを抽出する

– Content() または XMLxxxfromFile UDF を使用して XML 文書をサーバーのファイル・システム上のファイルにエクスポートするときには、結果文書は、サーバーのオペレーティング・システムのロケールのコード・ページでエンコードされますが、それを UTF-8 にすることはできません。

付録D. XML エクステンダーの制限

XML エクステンダーの DAD ファイル、ストアード・プロシージャー、および表には、以下の制限があります。

表 58. XML エクステンダーの制限

値	制限
分解 XML コレクション内の表	分解されたそれぞれの XML 文書から 1024 行
ストアード・プロシージャー・パラメーター:	
XML 文書 CLOB	1 MB[2]
文書アクセス定義 (DAD) CLOB	100 KB[2]
collectionName	30
colName	30
dbName	18
defaultView	128
rootID	128
resultTabName	128
tablespace	128
tbName	128[1]
db2xml.DTD_REF 表列	
AUTHOR	128
CREATOR	128
UPDATOR	128
DTDID	128
CLOB	100 KB

注:

1. *tbName* がスキーマ名によって修飾される場合、名前全体 (区切り文字も含む) は 128 文字以下でなければなりません。
2. このサイズは変更可能です。この方法については、227ページの『CLOB 制限の引き上げ』を参照してください。

名前は、DB2 がクライアントのコード・ページからデータベースのコード・ページにそれらを変換するときに、長くなる可能性があります。名前がクライアントのサイズの制限内には収まっていても、ストアード・プロシージャーが変換後の名前を取得するときに、その制限を超えている場合があります。詳細については、*DB2 アプリケーション開発の手引き* の『複合環境におけるプログラミング』という章の、『各国語サポートアプリケーションの開発』という節を参照してください。

特記事項

本書において、日本では発表されていない IBM 製品 (機械およびプログラム)、プログラミングまたはサービスについて言及または説明する場合があります。しかし、このことは、弊社がこのような IBM 製品、プログラミングまたはサービスを、日本で発表する意図があることを必ずしも示すものではありません。本書で IBM ライセンス・プログラムまたは他の IBM 製品に言及している部分があっても、このことは当該プログラムまたは製品のみが使用可能であることを意味するものではありません。 IBM 製品、プログラム、またはサービスに代えて、 IBM の有効な知的所有権またはその他の法的に保護された権利を侵害することのない、機能的に同等の製品、プログラム、またはサービスを使用することができます。ただし、IBM によって明示的に指定されたものを除き、他社の製品と組み合わせた場合の操作の評価と検証はお客様の責任で行っていただきます。

IBM は、本書で解説されている主題について特許権 (特許出願を含む)、商標権、または著作権を所有している場合があります。本書の提供は、これらの特許権、商標権、および著作権について、本書で明示されている場合を除き、実施権、使用権等を許諾することを意味するものではありません。実施権、使用権等の許諾については、下記の宛先に、書面にてご照会ください。

〒106-0032 東京都港区六本木 3 丁目 2-31
AP 事業所
IBM World Trade Asia Corporation
Intellectual Property Law & Licensing

以下の保証は、国または地域の法律に沿わない場合は、適用されません。

IBM およびその直接または間接の子会社は、本書を特定物として現存するままの状態で提供し、商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任を含むすべての明示もしくは黙示の保証責任を負わないものとします。国または地域によっては、法律の強行規定により、保証責任の制限が禁じられる場合、強行規定の制限を受けるものとします。

本書に対して、周期的に変更が行われ、これらの変更は、文書の次版に組み込まれます。 IBM は、随時、この文書に記載されている製品またはプログラムに対して、改良または変更を行うことがあります。

本プログラムのライセンス保持者で、(i) 独自に作成したプログラムとその他のプログラム (本プログラムを含む) との間での情報交換、および (ii) 交換された情報の相互利用を可能にすることを目的として、本プログラムに関する情報を必要とする方は、下記に連絡してください。

IBM Corporation
J74/G4
555 Bailey Avenue
P.O. Box 49023
San Jose, CA 95161-9023
U.S.A.

本プログラムに関する上記の情報は、適切な使用条件の下で使用することができますが、有償の場合もあります。

本書で説明されているライセンス・プログラムまたはその他のライセンス資料は、 IBM 所定のプログラム契約の契約条項、IBM プログラムのご使用条件、またはそれと同等の条項に基づいて、 IBM より提供されます。

IBM 以外の製品に関する情報は、その製品の供給者、出版物、もしくはその他の公に利用可能なソースから入手したものです。 IBM は、それらの製品のテストは行っておりません。また、IBM 以外の製品に関するパフォーマンスの正確性、互換性、またはその他の要求は確証できません。 IBM 以外の製品の性能に関する質問は、それらの製品の供給者にお願いします。

IBM の将来の方向性または意図については、予告なしに変更または中止する場合があり、 IBM の目的および目標のみを示しているものです。

著作権:

本書には、様々なオペレーティング・プラットフォームでのプログラミング手法を例示するサンプル・アプリケーション・プログラムがソース言語で掲載されています。お客様は、サンプル・プログラムが書かれているオペレーティング・プラットフォームのアプリケーション・プログラミング・インターフェースに準拠したアプリケーション・プログラムの開発、使用、販売、配布を目的として、いかなる形式においても、IBM に対価を支払うことなくこれを複製し、改変し、配布することができます。これらの例は、すべての場合について完全にテストされたものではありません。 IBM はこれらのプログラムの信頼性、可用性、および機能について法律上の瑕疵担保責任を含むいかなる明示または暗示の保証責任も負いません。

商標

以下は、IBM Corporation の商標です。

DB2	Net.Data
DB2 Extenders	OS/2
DB2 Universal Database	OS/390
IBM	OS/400
IMS	VTAM

Java およびすべての Java 関連の商標およびロゴは Sun Microsystems, Inc. の米国およびその他の国における商標または登録商標です。

Microsoft、Windows、Windows NT および Windows ロゴは Microsoft Corporation の米国およびその他の国における商標です。

UNIX は、The Open Group がライセンスしている米国およびその他の国における登録商標です。

他の会社名、製品名およびサービス名等はそれぞれ各社の商標または登録商標です。

用語集

［ア行］

アクセスおよび保管の方式 (access and storage method). XML 文書を DB2 データベースに関連付ける方法には、主にアクセスおよび保管の 2 つの方式 (XML 列と XML コレクション) がある。 *XML 列 (XML column)* および *XML コレクション (XML collection)* も参照。

アプリケーション・プログラミング・インターフェース (API) (application programming interface (API)).

(1) オペレーティング・システムから、または別途注文品のライセンス・プログラムからシステムに提供される機能インターフェース。高水準言語で作成されたアプリケーション・プログラムは、オペレーティング・システムまたはライセンス・プログラムの特定のデータまたは機能を使うことができる。

(2)DB2 では、インターフェース内の機能 (たとえば、エラー・メッセージ獲得 API)。

(3) DB2 では、インターフェースの中の機能を指す。たとえば、エラー・メッセージ取得 API がある。

エレメント (element). *XML エレメント (XML element)* を参照。

オブジェクト (object). オブジェクト指向プログラミングにおいて、あるデータとそれに関連付けられた操作から成る抽象的な実体。

［カ行］

外部キー (foreign key). 参照制約の定義に含まれるキーで、従属表の 1 つまたは複数の列から成る。

外部ファイル (external file). DB2 の外部のファイル・システムに存在するファイル。

拡張可能スタイル・シート言語 (XSL) (Extensible Stylesheet language (XSL)). スタイル・シートを表すために使用される言語。XSL は、XML 文書を変換する言語と、形式設定セマンティクスを指定するための XML ボキャブラリーの 2 つの部分から成る。

拡張可能スタイル・シート言語トランスフォーメーション (XSLT) (Extensive Stylesheet Language Transformation (XSLT)). XML 文書を別の XML 文書に変換するために使われる言語。 XSLT は、XML のスタイル・シート言語である XSL の一部として使用するように設計された。

管理サポート表 (administrative support tables). XML オブジェクトに対するユーザー要求を処理するために DB2 エクステンダーが使用する表。管理サポート表には、エクステンダーで使用可能になっているユーザー表と列を識別するものがある。他の管理サポート表には、使用可能な列内のオブジェクトに関する属性情報が含まれている。「メタデータ表 (metadata table)」と同義。

基本キー (primary key). 表の定義の一部である固有キー。基本キーは、参照制約定義のデフォルトの親キーである。

区画 (partition). ストレージの中の固定サイズの部分。

組み込み SQL (embedded SQL). アプリケーション・プログラム内にコーディングされる SQL ステートメント。*静的 SQL (static SQL)* を参照。

結果セット (result set). ストアード・プロシージャーによって戻された行の集合。

結果表 (result table). SQL 照会またはストアード・プロシージャーの実行の結果として戻された行を含む表。

結合 (join). リレーショナル操作の 1 つで、マッチングする列の値に基づいて複数の表からデータを検索できる。

結合視点 (joined view). CREATE VIEW ステートメントによって作成される DB2 視点の 1 つで、 1 つまたは複数の表を互いに結合する。

合成する (compose). XML コレクション内のリレーショナル・データから XML 文書を生成すること。

構造化テキスト索引 (structural text index). DB2 テキスト・エクステンダーを使って、 XML 文書のツリー構造に基づいてテキスト・ストリングを索引付けること。

［サ行］

サイド表 (side table). XML 列内のエレメントや属性を検索する際のパフォーマンスを改善するために XML エクステンダーが作成する追加の表。

索引 (index). キーの値によって論理順に並べられたポインターの集合。索引は、データに迅速にアクセスするのに使われ、表内の行を固有化することができる。

述部 (predicate). 比較演算を明示または暗黙指定する検索条件のエレメント。

照会 (query). 特定の条件に基づいて、データベースからの情報を要求すること。たとえば、あるカスタマー表の中で、残高が 1000 より大きいすべてのカスタマーのリストを要求する照会など。

条件 (condition). XML データの選択基準の指定、または XML コレクション表を結合する方法の指定。

スカラー関数 (scalar function). ある値から別の 1 つの値を生成する SQL 操作で、関数名の後の括弧内に引き数をリストすることによって表される。

スキーマ (schema). 表、視点、索引、またはトリガーなど、データベース・オブジェクトのコレクション。スキーマはデータベース・オブジェクトを論理的に分類する。

ストアード・プロシージャー (stored procedure). 手続き構成および組み込み SQL ステートメントのブロック。データベースに保管され、名前で呼び出される。ストアード・プロシージャーを使用して、 1 つのアプリケーション・プログラムを 2 つの部分で実行できる。 1 つの部分はクライアント上で実行され、もう 1 つの部分はサーバーで実行される。これにより、1 回の呼び出しでデータベースへの複数のアクセスが可能となる。

静的 SQL (static SQL). プログラムに組み込まれ、プログラムが実行される前のプログラム準備処理で準備される SQL ステートメント。準備された後、ステートメントで指定されたホスト変数が変更されても、静的 SQL ステートメントは変更されない。

セクション検索 (section search). 1 つのセクション内でテキスト検索を行うこと。セクションはアプリケーションで定義できる。構造化テキスト検索をサポートするために、 Xpath の短縮ロケーション・パスによってセクションを定義することができる。

絶対ロケーション・パス (absolute location path). オブジェクトの全パス名。絶対パス名は最上位つまり「ルート」エレメントから始まり、これはスラッシュ (/) または円記号 (¥) 文字によって識別される。

全テキスト検索 (full text search). DB2 テキスト・エクステンダーを使用して、文書構造を問わずすべての場所にあるテキスト・ストリングを検索すること。

先頭 element_node (top element_node). DAD 内で、XML 文書のルート・エレメントを表す。

属性 (attribute). *XML 属性 (XML attribute)* を参照。

［タ行］

多重定義関数 (overloaded function). 複数の関数インスタンスが関連付けられている関数名。

妥当性検査 (validation). DTD を使用して、XML 文書の有効性を検査したり、 XML データの構造化検索を行うプロセス。 DTD は DTD リポジトリーに保管される。

単純ロケーション・パス (simple location path). シングル・スラッシュ (/) で区切られた一連のエレメント・タイプ名。

データ交換 (data interchange). 複数のアプリケーション間でデータを共用すること。 XML のサポートするデータ交換では、最初に適切な形式からデータを変換するプロセスが必要ない。

データ・ソース (data source). ODBC API をサポートする ODBC ドライバーを介してデータにアクセスすることが可能な、ローカルまたはリモートのリレーショナル・データ・マネージャーまたは非リレーショナル・データ・マネージャー。

データ・タイプ (data type). 列やリテラルの属性。

データ・リンク (datalink). DB2 データ・タイプの 1 つで、データベースの外に保管されているファイルへのデータベースからの論理参照を可能にする。

デフォルト視点 (default view). ある XML 表とそれに関連するすべてのサイド表を結合した形でデータを表示すること。

デフォルト・キャスト関数 (default casting function). SQL 基本タイプを UDT にキャストする関数。

電子データ交換 (EDI) (Electronic Data Interchange (EDI)). 企業間 (B2B) アプリケーションの電子データ交換の規格。

特殊タイプ (distinct type). *ユーザー定義タイプ (user-defined type)* を参照。

［ナ行］

ノード (node). データベース区画化では、「データベース区画」と同義。

［ハ行］

パス式 (path expression). *ロケーション・パス (location path)* を参照。

表スペース (table space). データベース・オブジェクトが保管されているコンテナーの一まとまりを指す抽象概念。表スペースは、データベースとデータベースに保管されている表との間の間接参照レベルを提供する。表スペースは、

- 割り当てられたメディア記憶装置にスペースを持つ。
- その中に作成された表を持つ。これらの表は、表スペースに属するコンテナーのスペースを消費する。表のデータ、索引、長フィールドおよび LOB 部分は同じ表スペースに保管できる。また、それぞれ別個の表スペースに保管することもできる。

副照会 (subquery). SQL ステートメントの検索条件の中で使われる SELECT ステートメント全体。

複数出現 (multiple occurrence). 1 つの文書内で列エレメントまたは属性を 2 度以上使用できるかどうかを示す。複数出現は DAD 内で指定される。

ブラウザー (browser). *Web ブラウザー (Web browser)* も参照。

プロシージャー (procedure). *ストアード・プロシージャー (stored procedure)* を参照。

分解する (decompose). XML を分離して、複数のリレーショナル表からなる XML コレクションにすること。

文書アクセス定義 (DAD) (Document Access Definition (DAD)). XML 列の索引付け体系または XML コレクションのマッピング体系の定義に使われる。これによって、 XML エクステンダーの列を (XML 形式の) XML コレクションにすることができる。

文書タイプ定義 (DTD) (document type definition (DTD)). 複数の XML エレメントおよび属性の宣言の集合。 DTD は、XML 文書内でどのようなエレメントがどんな順序で使われるか、またどのエレメントが他のエレメントを包含しているかを定義する。 XML 文書の妥当性検査を行うために、 DTD を文書アクセス定義 (document access definition (DAD)) ファイルに関連付けることができる。

［マ行］

マッピング体系 (mapping scheme). XML データをリレーショナル・データベース内で表す方法についての定義。マッピング体系は DAD で指定される。 XML エクステンダーの提供するマッピング体系には、*SQL マッピング* とリレーショナル・データベース・ノード *(RDB_node)* マッピング の 2 つがある。

メタデータ表 (metadata table). *管理サポート表 (administrative support table)* を参照。

文字ラージ・オブジェクト (CLOB) (character large object (CLOB)). 単一バイト文字の文字ストリングで、ストリングの長さの上限は 2 GB。 CLOB には関連したコード・ページがある。単一バイト文字を含むテキスト・オブジェクトは、 DB2 データベースでは CLOB として保管される。

［ヤ行］

ユーザー定義関数 (user-defined function (UDF)). データベース管理システムに定義される関数で、定義後は SQL 照会で使用できるようになる。以下のいずれかの関数である。

- 外部関数。関数の本体はスカラー値を引き数とするプログラミング言語で書かれ、呼び出すたびにスカラー結果が生成される。
- ソース関数。すでに DBMS に認識されている別の組み込み関数またはユーザー定義関数によって実装される。この関数は、スカラー関数または列 (集合) 関数のいずれかで、値のセットから単一の値 (MAX、AVG など) を戻す。

ユーザー定義タイプ (user-defined type (UDT)). データベース・マネージャーに元々あったものではなく、ユーザーにより作成されたデータ・タイプ。*特殊タイプ (distinct type)* を参照。

ユーザー表 (user table). アプリケーション用に作成され、アプリケーションによって使用される表。

有効な文書 (valid document). 関連した DTD のある XML 文書。 XML 文書が有効であるためには、関連する DTD で指定された構文規則に従わなければならない。

［ラ行］

ラージ・オブジェクト (LOB) (large object (LOB)). 長さの上限が 2 GB のバイト順序列。LOB のタイプは、 *2 進ラージ・オブジェクト (binary large object (BLOB))*、*文字ラージ・オブ*

ジェクト *(character large object (CLOB))*、および *2* バイト文字ラージ・オブジェクト *(double-byte character large object (DBCLOB))* の 3 つ。

リレーショナル・データベース・ノード (RDB_node) (relational database node (RDB_node)). 表、任意指定の列、および任意指定の条件の定義を 1 つまたは複数含んでいるノード。表および列は、データベース内に XML データを保管する方法を定義するのに使われる。条件は、XML データの選択基準、または XML コレクション表の結合方法を指定する。

ルート ID (root ID). すべてのサイド表をアプリケーション表に関連付ける一意的な ID。

ルート・エレメント (root element). XML 文書の先頭エレメント。

列データ (column data). DB2 列の中に保管されているデータ。 DB2 のサポートするすべてのデータ・タイプが使用できる。

ローカル・ファイル・システム (local file system). DB2 の中に存在するファイル・システム。

ロケーション・パス (location path). ロケーション・パスとは、ある XML エレメントまたは属性を識別する一連の XML タグ。ロケーション・パスは XML 文書の構造を識別し、エレメントまたは属性のコンテキストを示す。単一スラッシュ (/) のパスは、コンテキストが文書全体であることを示す。 UDF を抽出する際にロケーション・パスを使用して、抽出対象のエレメントまたは属性を識別する。また、DAD では XML 列の索引付け体系の定義の際にロケーション・パスを使用して、 XML エレメント (または属性) と DB2 列とをマッピングする。さらに、テキスト・エクステンダーは構造化テキスト検索の際にロケーション・パスを使用する。

ロケーター (locator). オブジェクトの位置指定に使用されるポインター。 DB2 では、LOB の位置指定に使われるデータ・タイプはラージ・オブジェクト・ブロック (LOB) ロケーターである。

［ワ行］

［数字］

2 バイト文字ラージ・オブジェクト (DBCLOB) (double-byte character large object (DBCLOB)). 2 バイト文字からなる文字ストリング、または単一バイト文字と 2 バイト文字の組み合わせによる文字ストリングで、ストリングの長さの上限は 2 GB。各 DBCLOB には、関連した 1 つのコード・ページがある。 2 バイト文字を含むテキスト・オブジェクトは、DB2 データベースでは DBCLOB として保管される。

A

API. *アプリケーション・プログラミング・インターフェース (application programming interface)* を参照。

attribute_node. あるエレメントの属性を表したもの。

B

B-tree 索引付け (B-tree indexing). DB2 エンジンが提供する固有の索引体系。索引項目を B-tree 構造で作成する。これは DB2 基本データ・タイプをサポートしている。

C

cast 関数 (cast function). ある (ソース) データ・タイプのインスタンスを、別の (ターゲット) データ・タイプのインスタンスに変換する関数。一般に、cast 関数にはターゲット・データ・タイプの名前が付いている。この関数の引き数は 1 つで、そのタイプはソース・データ・タイプ。戻されるタイプはターゲット・データ・タイプ。

CLOB. 文字ラージ・オブジェクト (character large object)。

D

DAD. *文書アクセス定義 (document access definition)* を参照。

DBCLOB. 2 バイト文字ラージ・オブジェクト (Double-byte character large object)。

DTD. *文書タイプ定義 (document type definition)* を参照。

DTD 参照表 (DTD_REF 表) (DTD reference table (DTD_REF table)). DTD を含んでいる表。DTD は、XML 文書の妥当性検査、およびアプリケーションによる DAD ファイルの定義に使われる。ユーザーは独自の DTD を DTD_REF 表に挿入することができる。この表は、データベースが XML 使用可能になる時に作成される。

DTD リポジトリー (DTD repository). DTD_REF という DB2 表で、この表のそれぞれの行は 1 つの DTD を表し、あわせてメタデータ情報も各行に保管される。

DTD_REF 表 (DTD_REF table). DTD 参照表 (DTD reference table)。

E

EDI. 電子データ交換 (Electronic Data Interchange)。

element_node. あるエレメントを表したもの。element_node はルート・エレメントまたは子エレメントのいずれかである。

J

Java データベース・コネクティビティー (JDBC) (Java Database Connectivity (JDBC)). アプリケーション・プログラミング・インターフェース (API) の 1 つで、 Open Database Connectivity (ODBC) と同じ特性をもつが、特に Java データベース・アプリケーションによる使用のために設計された。また、JDBC ドライバーのないデータベースのために、JDBC は JDBC-ODBC 間のブリッジを提供する。これは JDBC から ODBC への変換メカニズムで、JDBC は Java データベース・アプリケーションに JDBC API を提供してこれを ODBC に変換する。 JDBC は Sun Microsystems, Inc. 社、および数多くのパートナー企業やベンダーによって開発された。

JDBC. Java データベース・コネクティビティー (Java Database Connectivity)。

L

LOB. ラージ・オブジェクト (large object)。

N

O

ODBC. Open Database Connectivity。

Open Database Connectivity. リレーショナル・データベース管理システムと非リレーショナル・データベース管理システムの両方でデータにアクセスするための、標準的なアプリケーション・プログラミング・インターフェース (API)。データベース・アプリケーションはこの API を使用して、データベース管理システムごとにデータ保管形式やプログラミング・インターフェースが違っている場合でも、さまざまなコンピューター上のデータベース管理システムに保管されているデータにアクセスできる。 ODBC は X/Open SQL Access Group のコール・レベル・インターフェース (CLI) 仕様に基づき、 Digital Equipment Corporation (DEC)、Lotus、Microsoft、および

Sybase 社によって開発された。 *Java* データベース・コネクティビティー *(Java Database Connectivity)* と対比。

Q

R

RDB_node. リレーショナル・データベース・ノード。

RDB_node マッピング (RDB_node mapping). XML エレメントの内容または XML 属性の値の位置で、RDB_node によって定義される。 XML エクステンダーはこのマッピングを使用して、XML データを保管または検索する場所を判別する。

S

SQL マッピング (SQL mapping). XML エレメントの内容または XML 属性の値をリレーショナル・データと関連付ける定義で、 1 つまたは複数の SQL ステートメントおよび XSLT データ・モデルを使用する。 XML エクステンダーはこの定義を使用して、XML データを保管または検索する場所を判別する。 SQL マッピングは、SQL_stmt エレメントとともに DAD 内で定義される。

T

text_node. あるエレメントの CDATA テキストを表したもの。

U

UDF. *ユーザー定義関数 (user-defined function)* を参照。

UDT. *ユーザー定義タイプ (user-defined type)* を参照。

uniform resource locator (URL). HTTP サーバーの名前を指定し、オプションでディレクトリーとファイル名も指定するアドレス。たとえば、http://www.ibm.com/data/db2/extenders

UNION. 2 つの SELECT ステートメントの結果を結合する SQL 操作。 UNION は、複数の表から得られた値のリストをマージするためにしばしば使用される。

URL. uniform resource locator。

W

Web ブラウザー (Web browser). 要求を Web サーバーに送信して、サーバーが戻した情報を表示するクライアント・プログラム。

well-formed 文書 (well-formed document). DTD を含んでいない XML 文書。 XML 仕様であっても、有効な DTD を含む文書もまた well-formed でなければならない。

X

XML. 拡張可能マークアップ言語 (eXtensible Markup Language)。

XML Path 言語 (XML Path Language). XML 文書の各部分をアドレッシングするための言語。XML Path 言語は、XSLT で使用する目的で設計された。すべてのロケーション・パスは、XPath 用に定義された構文を使用して表現できる。

XML UDF. XML エクステンダーの提供する DB2 ユーザー定義関数。

XML UDT. XML エクステンダーの提供する DB2 ユーザー定義タイプ。

XML エレメント (XML element). XML DTD で定義されている、すべての XML タグまたは ELEMENT。 XML エクステンダーはロケーション・パスを使用してエレメントを識別する。

XML オブジェクト (XML object). 「XML 文書 (XML document)」と同じ。

XML コレクション (XML collection). 関係表の集合であり、XML 文書の合成に使用するデータ、または XML 文書から分解されたデータを提供する。

XML 属性 (XML attribute). DTD において、XML エレメントの下の ATTLIST で指定されているすべての属性。 XML エクステンダーはロケーション・パスを使用して属性を識別する。

XML タグ (XML tag). すべての有効な XML マークアップ言語タグ (特に XML エレメント)。タグとエレメントは同義語として扱われる。

XML 表 (XML table). 1 つまたは複数の XML エクステンダー列を含んでいるアプリケーション表。

XML 列 (XML column). XML エクステンダー UDT に使用できるアプリケーション表内の列。

XPath. XML 文書の各部分をアドレッシングするための言語。

XPath データ・モデル (XPath data model). ノードを使って XML 文書をモデル化およびナビゲートするのに使用するツリー構造。

XSL. XML スタイル・シート言語 (XML Stylesheet Language)。

XSLT. XML スタイル・シート言語トランスフォーメーション (XML Stylesheet Language Transformation)。

索引

日本語, 数字, 英字, 特殊文字の順に配列されています。なお, 濁音と半濁音は清音と同等に扱われています。

[ア行]

[カ行]

［サ行］

［タ行］

[ナ行]

[ハ行]

［マ行］

［ヤ行］

［ラ行］

A

B

C

D

E

F

J

M

O

R

S

T

U

W

X

IBM と連絡をとる

技術上の問題がある場合は、時間をとって*問題判別の手引き* に定義されている処置を検討し、それらの提案を実行した後で、DB2 顧客サービスに連絡をとってください。この資料には、DB2 顧客サービスがお客様を支援するために必要とする情報が説明されています。

製品情報

以下の情報は英語で提供されます。内容は英語版製品に関する情報です。

http://www.ibm.com/software/data/

DB2 World Wide Web ページには、ニュース、製品説明、研修スケジュールなどの DB2 に関する最新情報が提供されています。ただし、提供されている情報は英語です。

http://www.ibm.com/software/data/db2/library/

「DB2 Product and Service Technical Library」では、よくされる質問 (FAQ)、修正内容、資料、および最新の DB2 技術情報などの情報へのアクセスが提供されています。

注: この情報のご提供は英語のみとなりますのでご注意ください。

http://www.elink.ibmlink.ibm.com/pbl/pbl/

「International Publications」注文用 Web サイトでは、マニュアルの注文方法についての情報を提供しています。ただし、提供されている情報は英語です。

http://www.ibm.com/education/certify/

IBM の「Professional Certification Program」Web サイトでは、 DB2 を含むさまざまな IBM 製品の認証テストの情報を提供しています。ただし、提供されている情報は英語です。

ftp.software.ibm.com

匿名でログオンしてください。ディレクトリー `/ps/products/db2` には、 DB2 および多数の他製品に関連したデモ、修正プログラム、情報、およびツールがあります。ただし、提供されている情報は英語です。

comp.databases.ibm-db2, bit.listserv.db2-l

これらのインターネット・ニュースグループは、ユーザーが DB2 製品に関する自分の経験について話し合うために利用できます。ただし、提供されている情報は英語です。

Compuserve: GO IBMDB2

このコマンドを入力すると、IBM DB2 Family forum にアクセスできます。すべての DB2 製品が、このフォーラムでサポートされています。ただし、提供されている情報は英語です。

米国以外の国で IBM に連絡する方法については、 *IBM Software Support Handbook* の Appendix A を参照してください。この資料にアクセスするには、 Web ページ: http://www.ibm.com/support/ にアクセスし、ページの最下部にある「IBM Software Support Handbook」リンク・ボタンを選択します。

注: 国によっては、 IBM が承認している販売業者が、 IBM サポート・センターの代わりにそれら販売業者のサポート・センターに連絡する場合があります。

Printed in Japan

日本アイ・ビー・エム株式会社
〒106-8711　東京都港区六本木3-2-12