IBM® DB2® Universal Database™

XML Extender 管理およびプログラミング・ガイド

バージョン 8.2

SC88-9172-01
(英文原典：SC27-1234-01)

IBM® DB2® Universal Database™

XML Extender 管理およびプログラミング・ガイド

バージョン 8.2

SC88-9172-01
(英文原典：SC27-1234-01)

ご注意！

本書および本書で紹介する製品をご使用になる前に、『*特記事項*』に記載されている情報をお読みください。

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)

お客様の環境によっては、資料中の円記号がバックスラッシュと表示されたり、バックスラッシュが円記号と表示されたりする場合があります。

原　典： SC27–1234–01
IBM® DB2® Universal Database™
XML Extender
Administration and Programming
Version 8.2

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 2004.8

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

本ガイドについて

このセクションには、以下の情報が記載されています。

- 『本書の対象読者』
- 『本書の使用方法』
- viii ページの『強調表示の規則』

本書の対象読者

本ガイド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 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.w3.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 UDB 管理の概念、ツール、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 UDB 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

本ガイドの最新バージョンは、以下の XML Extender 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

本ガイド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 Extender 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 UDB 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 UDB 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 UDB 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 Extender 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 Extender が発行するメッセージおよびコードをリストします。XML Extender の概念およびタスクは熟知しているが、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、コントロール表、またはコードについての情報が必要であるという場合には、この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。

強調表示の規則

本書では、以下の表記規則を使用します。

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
- ボタン

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

英大文字は、以下のものを示します。

- データ・タイプ
- 列名
- 表名

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

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。

  ▸▸── シンボルは、ステートメントの開始を示します。

  ──▸ シンボルは、構文図が次の行に続くことを示します。

  ▸── シンボルは、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。

  ──▸◂ シンボルは、ステートメントの終了を示します。

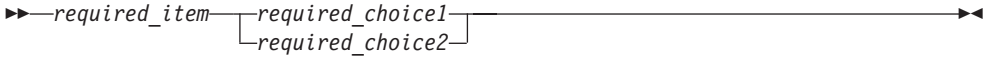
  完全なステートメント以外の、構文単位の図は、 ▸── シンボルから開始して ──▸ シンボルで終了します。
- 必須項目は、水平線 (メインパス) の線上に示されます。

- オプション項目は、メインパスの下に示されます。

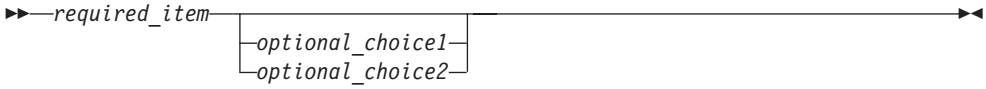
  オプション項目がメインパスの上に示されている場合、その項目は読みやすくするために示されているのであり、ステートメントの実行には影響を与えません。

- 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。

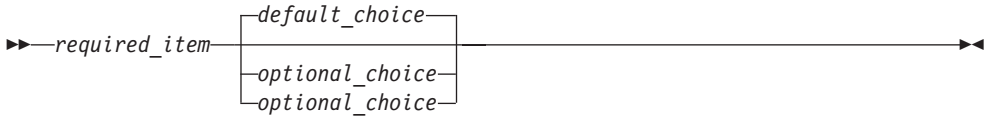
  項目から 1 つを選択しなければならない 場合、スタック内の項目の 1 つがメインパスの線上に示されます。



  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。
  - キーワードは大文字で表記されています (たとえば、FROM) 。 XML Extender では、キーワードは大文字でも小文字でもかまいません。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、または同様の他のシンボル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。

第 1 部 概要

ここでは、XML Extender 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 概要

XML Extender の概要

DB2® の XML Extender 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 文書からリレーショナル表に行を挿入したりする機能が備わっています。 XML Extender には、また、DB2 UDB で XML を管理するための新規のデータ・タイプ、関数、ストアード・プロシージャーも備わっています。

XML Extender は、次のオペレーティング・システムで使用できます。

- Windows® NT
- Windows 2000
- AIX®
- Solaris™ オペレーティング環境
- Linux
- PA-RISC プラットフォームでの HP-UX
- OS/390® および z/OS™
- iSeries™

現在、DB2 XML Extender をサポートする 64 ビット・プラットフォームは、HP-UX プラットフォームだけです。

DB2 UDB XML Extender を使用して作業を開始する前に、一時ファイルを保管する場所を指定します。 DB2DXXTEMP 環境変数は、XML Extender の一時ファイルの場所を制御します。変数が設定されない場合は、TMP 値が一時ファイルの場所を決定します。 WINDOWS 2000 で DB2DXXTEMP 値を設定するには次のようにします。

1. DB2 で使用するユーザー ID でログオンしていることを確認してください。
2. **「スタート」 -> 「設定」 -> 「コントロール パネル」**をクリックします。
3. **「システム」**アイコンをダブルクリックします。
4. **「システムのプロパティ」**ノートブックの「詳細設定」で、**「環境変数」**をクリックします。
5. **「システム環境変数」**セクションで**「新規」**をクリックします。変数名に `DB2DXXTEMP` と入力し、変数値を `C:¥temp` のように入力します。
6. すべてのウィンドウを終了し、システムを再起動します。

関連概念:

- 4 ページの『XML 文書』
- 5 ページの『XML Extender の機能』
- 9 ページの『演習: XML 列への XML 文書の保管』
- 20 ページの『演習: XML 文書の合成』
- 8 ページの『XML Extender のチュートリアル演習』

XML 文書

企業では異なるアプリケーション間でデータが共有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問題に絶えず直面しています。その場合、そのような変換処理の多くで、データの部分的な消失が起きやすかったり、またはデータの整合性を保つための面倒な処理の実行が最低限必要であったりします。これを手作業でチェックするには、時間と経費がかかります。

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、標準アプリケーション・プログラミング・インターフェース (API) を作成して、データをリレーショナルおよび非リレーショナルの両方のデータベース管理システムでアクセス可能にすることです。これらのアプリケーションは、データをデータベース管理システムに保管します。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。データベース・アプリケーションは、データをアプリケーションが必要とする形式に変換するように作成しておくべきです。アプリケーションの移り変わりは早く、すぐに時代遅れになります。データを HTML に変換するアプリケーションでは、表示用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。データをその表示から分離する方法が、アプリケーション相互でデータをやりとりする実際的な形式を得るために必要です。

XML (*eXtensible Markup Language*) は、この問題を解決します。 XML を、Extensible (拡張可能) と言えるのは、その言語がメタ言語であり、企業の必要に応じて独自の言語の作成が可能だからです。 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、データ構造をもキャプチャーすることができます。XML は、唯一のデータ交換形式ではありませんが、受け入れられる規格として定着しています。この規格に従えば、それぞれのアプリケーションは、独自の形式を使ったデータの変換を最初に行わなくても、互いにデータを共有できます。

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。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。今日の相互に接続された社会では、アプリケーション・プロバイダーには、アプリケーションの基本機能として、XML 交換形式を採用する強力な理由があります。

DB2 における XML データの処理方法

XML には、データ交換のための標準形式が用意されていて、多くの問題が解決されますが、挑戦すべき点はまだあります。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有する必要があるか。

- 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必要とする XML 結果をより効果的に取得することができます。XML Extender 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 UDB 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 XML データ (XML データ・タイプ) 用に準備されたユーザー定義タイプとして DB2 内に保管するか、または XML の内容をリレーショナル表内に基本データ・タイプとしてマップするかを選択できます。XML データ・タイプの場合、XML Extender には、 DB2 Universal Database™ が提供している構造化テキスト検索のほかに、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も加えられています。

XML Extender には、DB2 の XML データを保管し、アクセスするために 2 つの方式が用意されています。

XML 列方式

XML 文書全体を列データあるいは外部的にファイルとして保管し、必要な XML エレメントまたは属性値を抽出して、それを高速検索用の索引付き副表である*サイド表* に保管します。文書を列データとして保管することによって、次のことが行えます。

- 抽出され、SQL 基本データ・タイプとして索引を付けてサイド表に保管されている 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。
- Net Search Extender を使って構造化テキスト検索を行います。

XML コレクション方式

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。

XML Extender の機能

XML Extender 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
- データベース内の XML データを保管およびアクセスする方式
- XML データの妥当性検査に使用されるデータ・タイプ定義 (DTD) を保管するための DTD リポジトリー
- スキーマを使用して XML 文書を検証する能力

- XML 文書をリレーショナル・データにマップするために使用される文書アクセス定義 (DAD) と呼ばれるマッピング・ファイル
- ロケーション・パスは、XML 文書内のエレメントまたは属性の位置を指定します。

管理ツール: XML Extender の管理ツールを使用すると、データベースおよび表の列を XML に使用可能にしたり、XML データを DB2 リレーショナル構造にマップしたりするのに役立ちます。

次のようなツールを使用して、XML Extender の管理タスクを行うことができます。

- **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 Extender の管理ストアード・プロシージャーを使用すると、プログラムから管理コマンドを呼び出すことができます。

保管とアクセスの方式: XML Extender には、XML 文書を DB2 データ構造と統合するための保管およびアクセス方式が 2 種類準備されています。 XML 列方式と XML コレクション方式です。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 列方式

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 列方式は、文書をアーカイブするのに適した働きをします。 XML 使用可能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 UDB 表 (サイド表) にマップしてから、これに索引を付けて高速検索を行えるようにすることができます。

XML コレクション方式

この方式は、XML 文書構造を DB2 UDB 表にマップするのに役立ちます。それによって、既存の DB2 UDB データから XML 文書を合成したり、XML 文書を分解して、タグなしのデータとして DB2 UDB 表に保管したりできます。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: XML Extender では、XML エレメントの宣言および属性のセットである DTD を保管することもできます。データベースが XML 用に*使用可能にされる*と、DTD リポジトリー表 (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 は、XML 文書の構造を検査するために使用されます。

DAD ファイル: *文書アクセス定義 (DAD)* ファイルを使用して、XML Extender でどのように構造化 XML 文書を処理するかを指定します。 DAD ファイルは、XML 文書構造を DB2 UDB 表にマップする XML 文書です。 DAD ファイルは、XML 文書を列に保管するときや、XML データを合成または分解するときに使用します。DAD ファイルは、XML 列方式を使用して文書を保管するのか、または合成または分解用に XML コレクションを定義するのかを指定します。

ロケーション・パス: *ロケーション・パス* は、XML 文書内のエレメントまたは属性のロケーションを指定します。 XML Extender は、このロケーション・パスを使用して、XML 文書の構造をナビゲートし、エレメントと属性のロケーションを突き止めます。

たとえば、/Order/Part/Shipment/ShipDate というロケーション・パスは、″Shipment″、″Part″、″Order″ の子である、″ShipDate″ エレメントのロケーションを指します。この例を次に示します。

```
<Order>
  <Part>
    <Shipment>
      <ShipDate>
+...
```

図 1 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。

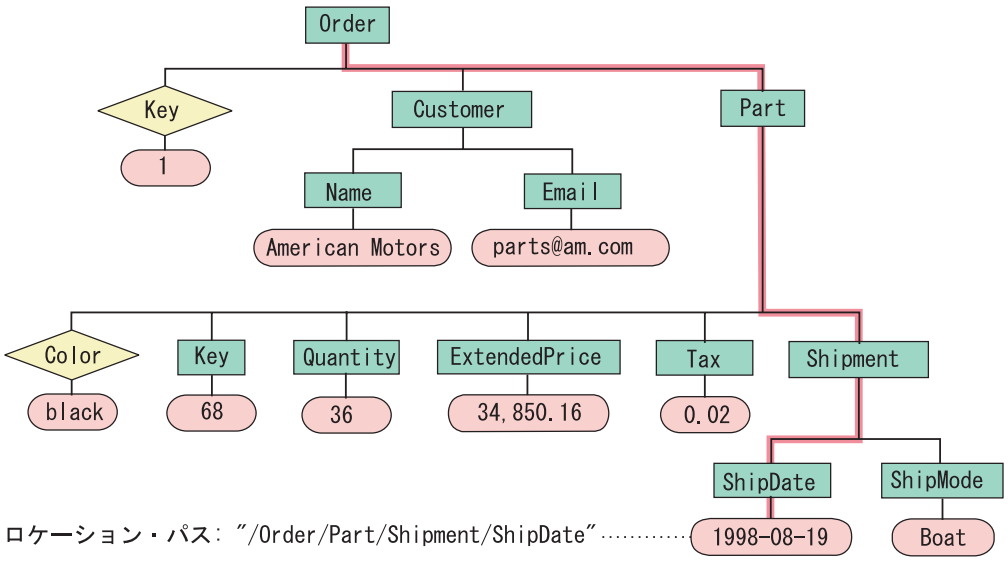


図 1. 文書を構造化 XML 文書として DB2 UDB 表の列に保管

ロケーション・パスは以下の状況で使用されます。

XML 列

- XML Extender ユーザー定義関数を使用する際に、抽出または更新すべきエレメントと属性を指定するために使用されます。
- XML エレメントまたは属性の内容をサイド表にマップするためにも使用されます。

XML コレクション

DAD ファイルの値をストアード・プロシージャーによってオーバーライドするために使用されます。

ロケーション・パスを指定するために、XML Extender は、XML 文書のアドレス部のための言語である *XML Path 言語 (XPath)* のサブセットを使用します。

XPath についての詳細は、以下の Web ページをご覧ください。

http://www.w3.org/TR/xpath

関連概念:

- 4 ページの『DB2 における XML データの処理方法』
- 9 ページの『演習: XML 列への XML 文書の保管』
- 20 ページの『演習: XML 文書の合成』
- 8 ページの『XML Extender のチュートリアル演習』

XML Extender のチュートリアル演習

このチュートリアルでは、XML Extender の使用を開始して、アプリケーション用の XML データをアクセスし、データを変更する方法を示します。演習が 3 つ用意してあります。

- XML 文書の XML 列への保管
- XML 文書の合成
- データベースのクリーンアップ

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、XML Extender 管理コマンドと一緒に、 DB2® コマンド・ウィンドウを使用します。 XML データ管理の学習では、XML Extender の 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、このセクションで使用されるサンプル・データを利用しています。

前提条件

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。 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。次に、既存の購入注文データベース SALES_DB で情報を抽出します。

関連概念:

- 39 ページの『XML Extender の管理ツール』
- 41 ページの『XML Extender 管理計画の概要』
- 9 ページの『演習: XML 列への XML 文書の保管』
- 20 ページの『演習: XML 文書の合成』

演習: XML 列への XML 文書の保管

XML Extender には、XML 文書全体をデータベース内に保管しアクセスするための方式が備わっています。 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、サイド表に列の索引を作成し、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。

この演習では XML 列の保管およびアクセス方式の使用方法を学習します。

シナリオ:

サービス部門の販売データを保存する作業が与えられています。処理したい販売データが、同じ DTD を使用する複数の XML 文書に保管されています。

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。サービス部門では、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_TAB 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ます。 この列を ORDER と名付けます。

この XML 文書を SALES_TAB 表に保管するには、次のようにします。

1. XML 文書を保管する XML Extender ユーザー定義タイプ (UDT) と、頻繁に照会される XML エレメントおよび属性を決定します。
2. XML 用に SALES_DB データベースをセットアップします。
3. 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。
4. 妥当性検査のために XML 文書用の DTD を挿入します。
5. 文書を XMLVARCHAR データ・タイプとして保管します。

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。

表 1は、SALES_TAB 表について説明しています。 XML 用に使用可能にされる XML 列、ORDER は、イタリックで示してあります。

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

スクリプトおよびサンプル:

このチュートリアルでは、1 組のスクリプトを使用して、環境をセットアップし、演習のステップを実行します。これらのスクリプトは、

dxx_install/samples/db2xml/cmd ディレクトリー (*dxx_install* は、XML Extender のファイルをインストールしたディレクトリーです) にあります。

これらのスクリプトについて、以下に説明します。

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 Extender のストアード・プロシージャーにバインドし、データベースを XML Extender 用に使用可能にします。

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 用に使用可能にされる列を追加してアプリケーション表を変更します。

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_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。

文書の保管方法の計画:

XML Extender を使用して文書を保管する前に、以下を行う必要があります。

- XML 文書の構造を理解します。
- XML 文書を保管する XML ユーザー定義タイプを決定します。
- サービス部門が頻繁に検索する XML エレメントと属性を判別します。これを決めることによって、その内容をサイド表に保管し、索引を作成して、パフォーマンスを改善することができます。

以下のセクションに、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。

この演習では XML 文書構造を理解し検証するために DTD の例を提供しています。

XML 列の XML データ・タイプの決定:

XML Extender は、XML 文書を保持するための列を定義するのに使用できる XML ユーザー定義タイプを提供します。それらのデータ・タイプは次のとおりです。

XMLVARCHAR
DB2 に保管する 3 キロバイトより小さい文書に使用します。
XMLVARCHAR 文書の最大サイズは、32672 バイトに等しい大きさに再定義することができます。

XMLCLOB
DB2 に保管する 3 キロバイトより大きい文書に使用します。最大文書サイズは 2 ギガバイトです。

XMLFILE
DB2 外に保管される文書に使用。

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。

検索するエレメントと属性の決定:

XML 文書の構造とアプリケーションのニーズを理解すれば、どのエレメントと属性が最も頻繁に検索または抽出されるか、あるいは、照会に最も経費のかかるエレメントと属性はどれかを判別することができます。サービス部門では、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、その検索では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
顧客名	/Order/Customer/Name
価格	/Order/Part/ExtendedPrice
出荷日付	/Order/Part/Shipment/ShipDate

XML 文書のサイド表へのマッピング:

XML 文書をサイド表にマップするためには、XML 列のための DAD ファイルを作成する必要があります。この DAD ファイルは、DB2 に XML 文書を保管するために使用されます。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 UDB サイド表に対して XML エレメントと属性をマップします。

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、使用する表の数、およびどの表内にどの列が入るかを判別します。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。任意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、文書構造は決まります。たとえば文書では、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回出現する可能性があります。複数回出現する可能性のあるエレメントは、それぞれが独自のサイド表に入っていなければなりません。

さらに、どの DB2 UDB 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決定する必要があります。これは、データの形式によって決まります。

- データがテキストであれば、VARCHAR を使用します。
- データが整数であれば INTEGER を使用します。
- データが日付の場合に範囲検索を行いたければ DATE を使用します。

このチュートリアルではエレメントと属性を、ORDER_SIDE_TAB、PART_SIDE_TAB または、SHIP_SIDE_TAB のいずれかにマップします。下記の表は、各エレメントまたは属性がどの表にマップされるかを示しています。

ORDER_SIDE_TAB

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

SALES_DB 表の作成:

この作業では、サンプル・データベースを作成し、XML 用のデータベースを使用可能にします。

データベースを作成するには、以下のように行います。

1. データベース・サーバーが DB2 UDB 管理者によって使用可能にされていることを確認します。
2. ディレクトリーを `dxx_install/samples/db2xml/cmd` に変更します。`dxx_install` は、XML Extender のファイルをインストールしたディレクトリー

です。 サンプル・ファイルには、絶対パス名を使用したファイルの参照が入っています。サンプル・ファイルをチェックし、ディレクトリー・パスのための値を変更します。

3. Windows® プラットフォームでは、次のように入力して DB2 UDB コマンド・ウィンドウをオープンします。

   ```
   DB2CMD
   ```

4. **getstart_db** コマンドを実行します。

サーバーの使用可能化:

XML 情報をデータベースに保管するには、それを XML Extender 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 Extender は以下のことを行います。

- ユーザー定義のタイプ (UDT)、ユーザー定義の関数 (UDF)、およびストアード・プロシージャーを作成します。
- コントロール表を作成して、そこに XML Extender 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 用データベースを使用可能にするには:

以下のいずれかの方式を使用して、データベースを使用可能にします。

次のスクリプトを実行します。

```
getstart_prep.cmd
```

このスクリプトは、データベースを XML Extender のストアード・プロシージャーおよび DB2 UDB 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_dbSALES_DB
```

XML 列を使用可能にして、文書を保管:

この演習では、XML Extender 用の、ある列を使用可能にし、その列に XML 文書を保管します。これらのタスクでは、以下のことを行います。

1. DTD リポジトリーに DTD を保管します。
2. XML 列のための DAD ファイルを作成します。
3. SALES_TAB 表を作成します。
4. XML タイプの列を追加します。
5. XML 列を使用可能にします。
6. 列およびサイド表を表示します。
7. 構造検索用のサイド表に索引を付けます。
8. XML 文書を保管します。
9. XML 文書を照会します。

DTD リポジトリーに DTD を保管する:

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 Extender は、XML 使用可能データベース内に、DTD_REF と呼ばれる表を作成しま

す。この表は DTD リポジトリーと呼ばれ、DTD を保管するために使用できます。XML 文書の妥当性検査を行う場合は、このリポジトリーに DTD を保管する必要があります。この演習では、DTD は以下の場所にあります。

```
dxx_install/samples/db2xml/dtd/getstart.dtd
```

ここで、 *dxx_install* は DB2 XML Extender をインストールしたディレクトリーです。

- 次の SQL INSERT コマンドを、すべて同じ DB2 コマンド行に入力します。

```
DB2 CONNECT TO SALES_DB
INSERT into DB2XML.DTD_REF values
('dxx_install/samples/db2xml/dtd/getstart.dtd',
 DB2XML.XMLClobFromFile
('dxx_install/samples/db2xml/dtd/getstart.dtd'),
 0, 'user1',   'user1', 'user1')
```

- 次のようなコマンド・ファイルを実行して DTD を挿入します。

```
getstart_insertDTD.cmd
```

XML 列用の DAD ファイルの作成:

ここでは、XML 列用の DAD ファイルの作成方法を説明します。 DAD ファイルに、使用するアクセスおよび保管方式が XML 列であることを指定します。 DAD ファイル内で、索引作成のための表と列を定義します。

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が、 dxx_install/samples/db2xml/dad/getstart_xcolumn.dad に入っています。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 に設定されています。

XML 列で使用する DAD ファイルを作成するには、次のようにします。

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ります。

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 "/dxx_install/samples/DB2XML/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. 文書の開始と終了のタグ (<DAD> と </DAD>) 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. 文書の妥当性検査を行う場合は、 DTD を指定するために DTD ID を付けた開始と終了のタグ (<DTDID> と </DTDID>) を挿入します。

   ```
   <dtdid>dxx_install/samples/db2xml/dtd/getstart.dtd</dtdid>
   ```

   このストリングが、DTD リポジトリー・テーブルへの 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、 DTD 参照テーブル内で挿入したストリングとは異なることがあります。

5. 開始と終了のタグ (<validation> と </validation>) およびキーワード YES または NO を使用して、XML Extender で、DTD 参照テーブルに挿入された DTD によって XML 文書構造の妥当性検査を行うことを指定します。たとえば次のようにします。

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

   <validation> の値は大文字小文字混合が可能です。
6. 開始と終了のタグ (<Xcolumn> と </Xcolumn>) を挿入して、保管方式が XML 列であると指定します。
7. サイド表を作成します。作成したい各サイド表について、次を行います。
   a. 生成する各サイド表に対して開始と終了のタグ (<table> と </table>) を挿入します。このとき、次に示すように名前属性 (name=) を使用して、サイド表の名前を二重引用符で囲んで指定します。

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

   b. 表タグの内側に、サイド表に入れたい各列に対して <column> タグを挿入します。各列には、name、type、path、および multi_occurrence の 4 つの属性があります。

      name サイド表に作成される列の名前を指定します。

      type 索引作成されるエレメントまたは属性のそれぞれに対して、サイド表内のデータ・タイプを指定します。

      path 索引作成されるエレメントまたは属性のそれぞれについて、XML 文書のロケーション・パスを指定します。

      multi_occurrence
      パス属性で参照されるエレメントまたは属性が XML 文書内で複数回出現する可能性があるかどうかを指定します。 *multi_occurrence* に指定できる値は、 *YES* または *NO* です。値が *NO* の場合は、サイド表に複数の列タグを記述することができます。値が *YES* の場合は、サイド表には 1 列だけ記述することができます。

```
<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>
```

8. 必要な終了タグがあることを確認してください。
   - 終了タグ </Xcolumn> が最後の </table> タグの後にあること
   - 終了タグ </Xcolumn> が最後の </DAD> タグの後にあること
9. ファイルを次の名前で保管します。

   ```
   getstart_xcolumn.dad
   ```

作成したばかりのファイルを、サンプル・ファイル dxx_install/samples/db2xml/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。サンプル・ファイルには、絶対パス名を使用したファイルの参照が入っています。サンプル・ファイルをチェックし、ディレクトリー・パスのための値を変更します。

SALES_TAB 表の作成:

このセクションでは、SALES_TAB 表を作成します。最初に、この表には、注文用の販売情報が入った 2 つの列を作成します。

表を作成するには、以下のように行います。

以下のいずれかの方法で、以下の CREATE TABLE ステートメントを入力します。

- 以下の DB2 UDB コマンドを入力します。

  ```
  DB2 CONNECT TO SALES_DB

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

- 次のようなコマンド・ファイルを実行して表を作成します。

  ```
  getstart_createTabCol.cmd
  ```

XML タイプの列を追加する:

SALES_TAB 表に新規の列を追加します。この列には、以前生成した XML 文書がそのまま入ります。また、この列は XML UDT でなければなりません。XML Extender は複数のデータ・タイプを備えています。この演習では、XMLVARCHAR として文書を保管します。

XML タイプの列を追加するには、以下のようにします。

以下の 3 つのうちのいずれかの方法を使用して SQL ALTER TABLE ステートメントを実行します。

- 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
  ```

- 次のコマンド・ファイルを実行して表を変更します。

  ```
  getstart_alterTabCol.cmd
  ```

XML 列を使用可能にする:

XML タイプの列を作成後、XML Extender 用にその列を使用可能にします。列が使用可能になると、XML Extender は DAD ファイルを読み取り、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。

- サイド表列と結合された XML 文書を含む、XML 列のデフォルト・ビューを作成するかどうかを決定する。 XML 文書を照会するときに、デフォルト・ビューを指定できます。この演習では、-v パラメーターを付けてビューを指定します。
- *ROOT ID* としての主キー、アプリケーション表での主キーの列名、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ユニーク ID を指定するかどうかを決定する。主キーを指定しない場合は、XML Extender 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、アプリケーション表とサイド表を結び付けるキーとして使用されるため、XML 文書が更新されると、XML Extender は自動的にサイド表を更新できます。この演習では、-r パラメーターを付けたコマンド (INVOICE_NUM) で、主キーの名前を指定します。次に、XML Extender は、指定された列を ROOT_ID として使用し、列をサイド表に追加します。
- 表スペースを指定するか、デフォルトの表スペースを使用するかを決定する。この演習では、デフォルトの表スペースを使用します。

XML 用に列を使用可能にするには:

以下の3 つのうちのいずれかの方法を使用して、 **dxxadm enable_column** コマンドを実行します。

コマンド行:

- 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
```

- 次のコマンド・ファイルを実行して列を使用可能にします。

```
getstartenableCol.cmd
```

XML Extender は、INVOICE_NUM 列をもったサイド表を作成し、デフォルト・ビューを作成します。

重要: サイド表は絶対に変更しないでください。サイド表の更新は、XML 文書自体を更新する以外の方法では行わないでください。 XML 列で XML 文書を更新すると、XML Extender が自動的にサイド表を更新します。

列およびサイド表の表示:

XML 列を使用可能にすると、XML 列とサイド表のビューが作成されます。 XML 列を処理するときは、このビューを使用できます。

XML 列とサイド表列を表示するには:

コマンド行から以下の SQL SELECT ステートメントを入力します。

```
SELECT * FROM SALES_ORDER_VIEW
```

ビューには、getstart_xcolumn.dad ファイルで指定した、サイド表内の列が表示されます。

サイド表の構造検索用索引の作成:

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。このセクションでは、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、索引を付けるそのような列を示しています。

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。

以下の 3 つのうちのいずれかの方法を使用して、次の CREATE INDEX SQL コマンドを実行します。

- 以下の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)
```

- 次のようなコマンド・ファイルを実行して索引を作成します。

```
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 Extender 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、デフォルトのキャスト関数または XML Extender 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。

XML 文書を保管するには、以下のように行います。

1. dxx_install/samples/db2xml/xml/getstart.xml の XML 文書を開きます。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 宣言のパスを変更してディレクトリー構造に一致させる必要があります。

2. 以下のいずれかの方法で SQL INSERT コマンドを実行します。
   - 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
     ('dxx_install/samples/db2xml/
     /xml/getstart.xml'))
     ```

   - 次のコマンド・ファイルを実行して文書を保管します。

     ```
     getstart_insertXML.cmd
     ```

表が更新されているか確認してください。コマンド行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。

```
SELECT * FROM SALES_TAB
SELECT * FROM PART_SIDE_TAB
SELECT * FROM ORDER_SIDE_TAB
SELECT * FROM SHIP_SIDE_TAB
```

XML 文書の照会:

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。

以下のいずれかの方法で SQL SELECT コマンドを実行します。

- 以下の SQL 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"
  ```

- 次のコマンド・ファイルを実行して文書を検索します。

  ```
  getstart_queryCol.cmd
  ```

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。

以上で、XML 文書を DB2 UDB 表に保管する入門用のチュートリアルを完了しました。以下に例を示します。

```
SALES_PERSON

--------------------

Sriram Srinivasan
```

関連概念:

- 3 ページの『XML Extender の概要』
- 20 ページの『演習: XML 文書の合成』
- 8 ページの『XML Extender のチュートリアル演習』

演習: XML 文書の合成

この演習では、既存の DB2® データから XML 文書を合成する方法を学びます。

シナリオ:

与えられた任務では、既存の購入注文データベース SALES_DB から情報を取り出し、そこから要求情報を抽出して 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。

XML 文書を作成するために、関連する表の列を、購入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。アスタリスクの付いた列名は、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)

列名	データ・タイプ
COMMENT	VARCHAR(128)
PART_KEY	INTEGER

計画:

XML Extender 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る必要があります。このセクションでは、サービス部門が指定した XML 文書の構造と、 XML 文書の構造の定義に使う DTD について概略します。このセクションでは、この文書が、文書にデータを追加するために使用されるデータを含む列にどのようにマップするかも示します。

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。サービス部門は、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ことを要求しています。(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。)

文書構造を設計した後に、XML 文書の構造を記述する DTD を作成します。この演習には、XML 文書および DTD が備わっています。 DTD の規則と XML 文書の階層構造を使用して、 22 ページの図 2 に示すように、データの階層を作成することができます。

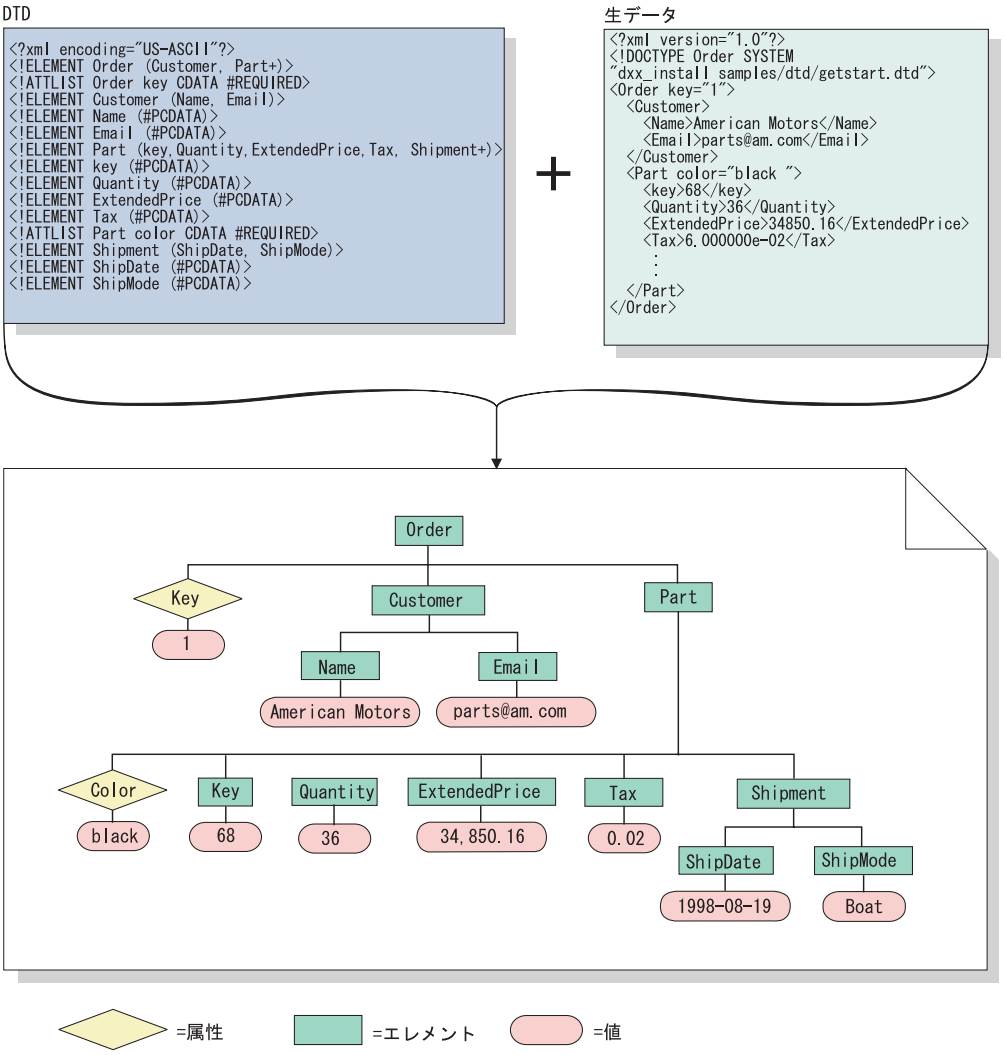


図 2. DTD および XML 文書の階層構造

XML 文書とデータベースの関係のマッピング:

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 UDB 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。 23 ページの図 3 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。

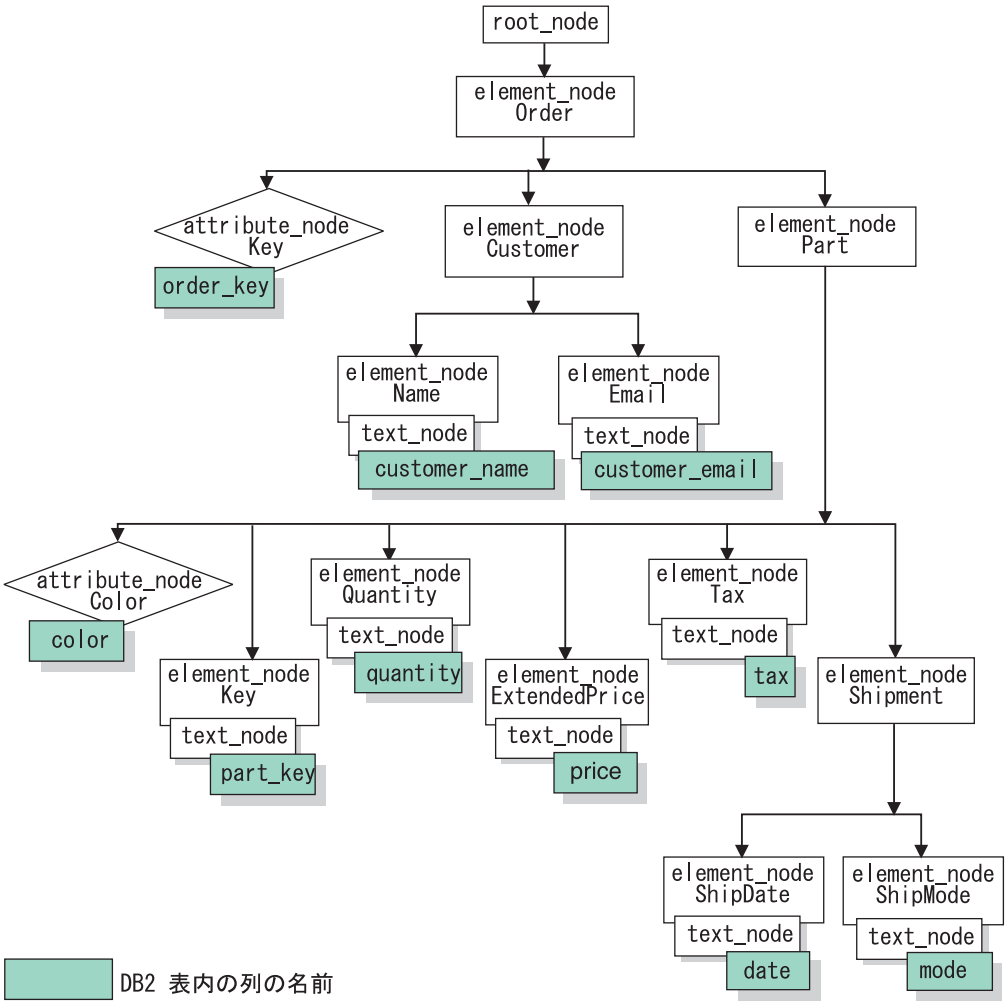


図 3. リレーショナル表の列にマップされた XML 文書

この図は、XML 文書構造内のエレメント、属性、およびテキストを示すのに、ノードを使用しています。これらのノードは、DAD ファイルで使用されているもので、後ろのステップで詳しく説明します。

このリレーションシップ記述を使用して、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 3 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。

スクリプトおよびサンプル:

この演習では、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。 これらのスクリプトは dxx_install/samples/db2xml/xml ディレクトリー (ここで、dxx_install は XML Extender ファイルをインストールしたディレクトリー) にあります。

スクリプトは、次のとおりです。

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 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。

演習環境のセットアップ:

このセクションでは、XML Extender 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:

このセクションでは、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. ディレクトリーを *dxx_install*/samples/db2xml/xml に変更します。 *dxx_install* は XML Extender ファイルをインストールしたディレクトリーです。 サンプル・ファイルには、絶対パス名を使用したファイルの参照が入っています。サンプル・ファイルをチェックし、ディレクトリー・パスのための値を変更します。
2. 以下のようにして、DB2 UDB コマンド・ウィンドウを開きます。

   ```
   DB2CMD
   ```

3. 以下のいずれかの方法を使用して、データベース・コマンド・ファイルを実行します。

   次のコマンドを入力します。

   ```
   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 Extender 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 Extender は以下のことを行います。

- ユーザー定義のタイプ (UDT)、ユーザー定義の関数 (UDF)、およびストアード・プロシージャーを作成します。
- コントロール表を作成して、そこに XML Extender 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。

XML 用データベースを使用可能にするには、以下の方式のいずれかを使用します。

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
```

これらのスクリプトは、データベースを XML Extender のストアード・プロシージャーおよび DB2 UDB 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 コレクション用の DAD ファイルの作成:

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。 DAD ファイルの作成によりコレクションを定義します。

このセクションでは、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、 *dxx_install*/samples/db2xml/dad/getstart_xcollection.dad に入っています。

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り、サンプル・ファイルを更新する必要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. dxx_install/samples/db2xml/xml 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 "dxx_install/samples/db2xml/dtd/dad.dtd">
   ```

   dxx_install を DB2 XML Extender がインストールされたディレクトリーに変更します。

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 DTD を挿入するときに、XML Extender で XML 文書構造の妥当性検査を行うかどうかを指定します。この演習では、DTD は必要ありませんので、値は**「いいえ (NO)」**です。

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

   <validation> タグの値は、大文字小文字混合が可能です。
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 UDB 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. <Xcollection> タグの後に、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。以下のステートメントを入力します。

   ```
   <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(db2xml.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>
   </Xcollection>
   ```

   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、23 ページの図 3 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・エレメントの列が第 2、出荷の列が第 3 になります。
   - データベースからのデータを必要とするテンプレートについては、反復セクションの列と反復しないセクションの列を、それぞれグループにまとめます。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がないので、generate_unique DB2 組み込み関数を使用して、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 ORDER BY でリストされます。 ORDER BY 内の列は、スキーマや表名で修飾されず、SELECT 文節内の列名に一致します。
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
"dxx_install/samples/db2xml/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 UDB 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。

23 ページの図 3 は、XML 文書および DB2 UDB 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 UDB 表の列名を示します。

ノードの各タイプを追加するには、一度に 1 タイプずつ追加します。

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 の属性があります。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、そのためユニーク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、適切な 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 UDB 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。 <column/> タグ内で定義される列は、<SQL_stmt> SELECT 文節内になければなりません。

```
<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. 必要な終了タグがあることを確認してください。
    - 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
    - 終了のための </Xcollection> タグが </root_node> タグの後にあること
    - 終了のための </Xcollection> タグが </DAD> タグの後にあること
12. ファイルを getstart_xcollection.dad として保管します。

作成したファイルを、サンプル・ファイル *dxx_install* /samples/db2xml/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ロケーション・パスおよびファイル・パス名が入っています。

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) 指定すると、パフォーマンスの改善に役立ちます。この演習では、コレクションを使用可能化しません。

XML 文書の構成:

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. 以下のいずれかの方式を使用して、dxxGenXML ストアード・プロシージャーを呼び出します。

   以下のコマンドを入力します。

   ```
   getstart_stp.cmd
   ```

   このストアード・プロシージャーは、XML 文書を合成して、それを RESULT_TAB 表に保管します。

   パーティション化された DB2 エンタープライズ・サーバー・エディション環境で XML Extender を実行している場合、修飾されたパーティション・キーを使

用する結果表を作成したか、またはシングル・ノードを使用するノード・グループ内にある表スペースの結果表を作成したかを確認します。

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。

- *dxx_install*/samples/db2xml/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 tests2x 実行可能ファイルを生成します。
- *dxx_install*/samples/db2xml/cli/sql2xml.c /dxxsamples/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. 以下のいずれかの方法を使用して、XML Extender の検索関数 Content() を呼び出し、XML 文書を表からファイルにエクスポートします。
   - 以下のコマンドを入力します。

     ```
     DB2 CONNECT TO SALES_DB

     DB2 SELECT DB2XML.Content(DB2XML.xmlVarchar(doc),
        'dxx_install/samplesdb2xml/cmd/xml/getstart.xml
          ') FROM RESULT_TAB
     ```

   - 次のコマンド・ファイルを実行して文書をエクスポートします。

     ```
     getstart_exportXML.cmd
     ```

ヒント: このステップでは、 DB2 UDB 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 文書を生成する方法を習得しました。結果セットを使用すると、複数の文書を生成するために複数の行を取り出すことができます。各文書を生成する際に、それをファイルにエクスポートできます。この方式は、結果セットの使い方を示す最も簡単な方法です。より効率的なデータのフェッチについては、 *dxx_install*/samples/db2xml/cli にある CLI の例を参照してください。

XML 文書を HTML ファイルにトランスフォームする:

ブラウザーに XML 文書からのデータを表示するには、stylesheet と XSLTransformToFile 関数を使用して XML 文書を HTML ファイルにトランスフォームする必要があります。

以下のステップに従って HTML ファイルにトランスフォームします。

1. テキスト・エディターを使用してスタイルシートを生成し、getstart.xsl と名前を付けます。

   ```
   <?xml version="1.0" encoding="UTF-8"?>
    <xsl:stylesheet version="1.0"
     xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

    <xsl:template match="/">
     <html>
     <head/>
     <body>

     ...

        </body>
      </html>
    </xsl:template>
    </xsl:stylesheet>
   ```

この完成したファイルの例が、以下のディレクトリーにあります。

%dxx_install%/samples/db2xml/xslt/getstart.xsl

2. エレメントごとに次のフォーマットを使用してタグを作成する。

```
<xsl:for-each select="xxxxxx">
```

このタグは、命令をトランスフォームするために使用されます。XML 文書の階層のエレメントごとにタグを作成します。たとえば次のようにします。

```
<?xml version="1.0" encoding="UTF-8"?>
 <xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

 <xsl:template match="/">
  <html>
  <head/>
  <body>

     <xsl:for-each select="Order">

    <xsl:for-each select="Customer">
          <xsl:for-each select="Name | Email">
          </xsl:for-each>
       </xsl:for-each>
<xsl:for-each select="Part">
        <xsl:for-each select="key | Quantity | ExtendedPrice | Tax">
      </xsl:for-each>

           <xsl:for-each select="Shipment">
        <xsl:for-each select="ShipDate | ShipMode">
        </xsl:for-each>
            </xsl:for-each>
          </xsl:for-each>
        </xsl:for-each>
    </body>
   </html>
 </xsl:template>
 </xsl:stylesheet>
```

3. HTML ファイルのフォーマットには、データを読みやすくするために XML エレメントの階層を示すリストを使用します。データの記述のためにテキスト・エレメントをいくらか作成します。たとえば、スタイルシート・ファイルは以下のようになります。

```
<?xml version="1.0" encoding="UTF-8"?>
 <xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

 <xsl:template match="/">
  <html>
  <head/>
  <body>

     <ol style="list-style:decimal outside">
      <xsl:for-each select="Order">
       <li> Orderkey : <xsl:value-of-select="@Key"/ <br/>

       <xsl:for-each select="Customer">
        <b>Customer</b><br/>
          <xsl:for-each select="Name | Email">
      <xsl:value-of select="name()"/>
      <xsl:text> : </xsl:text>
           <xsl:value-of select="."/>
           <xsl:text>, </xsl:text>
          </xsl:for-each>
       </xsl:for-each>
```

```
        <br/><br/>
    <ol type="A">
        <xsl:for-each select="Part">
         <li><b>Parts</b><br/>
      Color : <xsl:value-of select="@color"/>
           <xsl:text>, </xsl:text>

        <xsl:for-each select="key | Quantity | ExtendedPrice | Tax">
  <xsl:value-of select="name()"/>
            <xsl:text> : </xsl:text>
            <xsl:value-of select="."/>
  <xsl:text>, </xsl:text>
      </xsl:for-each>

           <br/><br/>
      <ol type="a">
           <xsl:for-each select="Shipment">
        <li><b>Shipment</b><br/>
        <xsl:for-each select="ShipDate | ShipMode">
    <xsl:value-of select="name()"/>
             <xsl:text> : </xsl:text>
             <xsl:value-of select="."/>
    <xsl:text>, </xsl:text>
        </xsl:for-each>
       </li>
           </xsl:for-each>
         <ol><br/>
         </li>
         </xsl:for-each>
        </ol>
       </li>
        </xsl:for-each>
       </ol>
    </body>
   </html>
 </xsl:template>
 </xsl:stylesheet>
```

4. XML 文書からのデータと共に <xsl:value-of select=″xxx″> タグを編集するために Xpath を使用します。

   エレメントのタグは、<xsl:value-of select″.″> です。ここで、ピリオド (″.″) は、通常のエレメントからデータを得るために使われます。

   属性のタグは、<xsl:value-of select=″@attributname″> です。属性名に付いているアットマーク (@) は属性の値を抽出します。 <xsl:value-of select=″name()″> を使用して XML タグの名前を得ることができます。

```
<?xml version="1.0" encoding="UTF-8"?>
 <xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

 <xsl:template match="/">
  <html>
  <head/>
  <body>

     <ol style="list-style:decimal outside">
      <xsl:for-each select="Order">
       <li> Orderkey : <xsl:value-of-select="@Key"/ <br/>

       <xsl:for-each select="Customer">
        <b>Customer</b><br/>
          <xsl:for-each select="Name | Email">
```

```
      <xsl:value-of select="name()"/>
      <xsl:text> : </xsl:text>
           <xsl:value-of select="."/>
           <xsl:text>, </xsl:text>
          </xsl:for-each>
       </xsl:for-each>

          <br/><br/>
    <ol type="A">
        <xsl:for-each select="Part">
         <li><b>Parts</b><br/>
      Color : <xsl:value-of select="@color"/>
           <xsl:text>, </xsl:text>

        <xsl:for-each select="key | Quantity | ExtendedPrice | Tax">
  <xsl:value-of select="name()"/>
            <xsl:text> : </xsl:text>
            <xsl:value-of select="."/>
  <xsl:text>, </xsl:text>
      </xsl:for-each>

           <br/><br/>
      <ol type="a">
           <xsl:for-each select="Shipment">
        <li><b>Shipment</b><br/>
        <xsl:for-each select="ShipDate | ShipMode">
    <xsl:value-of select="name()"/>
              <xsl:text> : </xsl:text>
              <xsl:value-of select="."/>
    <xsl:text>, </xsl:text>
        </xsl:for-each>
       </li>
           </xsl:for-each>
          </ol><br/>
          </li>
          </xsl:for-each>
       </ol>
      </li>
       </xsl:for-each>
      </ol>
    </body>
   </html>
  </xsl:template>
  </xsl:stylesheet>
```

5. スタイルシートを保管します。
6. HTML ファイルを以下のいずれかの方法で作成します。
   - XSLTransformToFile を使用して:

     ```
     SELECT XSLTransformToFile( CAST(doc AS CLOB(4k)),
         'dxx_install¥samples¥xslt¥getstart.xsl',
        'dxx_install¥samples¥html¥getstart.html')
      FROM RESULT_TAB
     ```

   - 以下のコマンドを使用します。

     ```
     Getstart_xslt.cmd
     ```

出力ファイルは、DB2 UDB サーバーにとってアクセス可能なファイル・システムにのみ書きこむことができます。

チュートリアル環境の終結処理:

演習環境を終結処理するには、提供されているスクリプトのいずれか 1 つを実行するか、またはコマンド行からコマンドを入力して、以下のようにします。

- XML 列 ORDER を使用不可にする。

- 演習で作成された表をドロップする。
- DTD リポジトリー表から DTD を削除する。

SALES_DB データベースが使用不可になったりドロップされ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 Extender で使うことができます。このセクション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。

以下のいずれかを使用して、終結処理コマンド・ファイルを実行します。

- 以下のコマンドを入力します。

  ```
  getstart_clean.cmd
  ```

- データベースを使用不可にしたい場合は、コマンド行から次の XML Extender コマンドを実行できます。

  ```
  dxxadm disable_db SALES_DB
  ```

  このコマンドは、管理コントロール表 DTD_REF および XML_USAGE をドロップするほか、XML Extender 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。

- データベースをドロップしたい場合、コマンド行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 をドロップします。

関連概念:

- 3 ページの『XML Extender の概要』
- 9 ページの『演習: XML 列への XML 文書の保管』
- 8 ページの『XML Extender のチュートリアル演習』

第 2 部 管理

ここでは、XML Extender の管理作業の実行方法について説明します。

第 2 章 管理

XML Extender の管理ツール

XML Extender の管理ツールを使うと、データベースおよび表の列を 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 Extender では、管理タスクとして使用できる以下のコマンド行ツールおよびプログラミング・インターフェースを用意しています。

XML Extender には管理タスクを行うために、次のようなツールが用意されています。

- XML Extender 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 Extender の管理ストアード・プロシージャーを使用すると、プログラムから管理コマンドを呼び出すことができます。

XML Extender を管理するための準備

XML Extender を実行するには、以下のソフトウェアをインストールしておく必要があります。

必要なソフトウェア: XML Extender には、 DB2® Universal Database のバージョン 8.2 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には、DB2 Universal Database Net Search Extender バージョン 8.2 が、DB2 Universal Database バージョン 8.2
- XML Extender 管理ウィザードには、以下のものが必要です。
  - DB2 Universal Database Java データベース・コネクティビティー (JDBC)
  - SDK 1.1.7 以降、または JRE 1.1.1。DB2 UDB コントロール・センターで利用可能です。
  - JFC 1.1 および Swing 1.1。DB2 UDB コントロール・センターで利用可能です。

XML Extender をインストールする前に、以下のタスクを行う必要があります。

- XML Extender を DB2 UDB データベース・にバインドする。

  XML Extender を各データベースにバインドする必要があります。例は以下のものを参照してください。

  dxx_install/samples/db2xml/cmd/getstart_prep.cmd
- セットアップの説明を表示する。
- XML アクセスのためのデータベースを作成する。

XML Extender を使用して管理タスクを実行するためには、DB2ADM 権限が必要です。

XML Extender データを旧バージョンからマイグレーションする

DB2® XML Extender の旧バージョンを使用している場合は、XML Extender バージョン 8.2 で既存の XML 使用可能データベースを使用する前に、XML Extender で使用可能な各データベースを移行する必要があります。

マイグレーション・プログラムは、すでにインストール済みの XML Extender のベース・レベルに従って、いろいろなステップを実行します。マイグレーション・プログラムが実行するステップには、以下のようなステップがあります。

- ユニコードと DBCS データベースで使用する XMLCLOB ユーザー定義タイプ (UDT) とユーザー定義関数 (UDF) を作成します。
- 一時表の使用をサポートするための dxxGenXML および dxxRetrieveXML のためのストアード・プロシージャーを追加作成します。
- ストアード・プロシージャーを変更して、パラメーター・スタイル SQL をパラメーター引き渡し用のリンケージ規則として使用します。
- スキーマと DTD の妥当性検査のための新しいユーザー定義関数、および XSLT 関数を作成します。
- CLOB を戻す新しいストアード・プロシージャー (dxxGenXMLCLOB および dxxRetrieveXMLCLOB) を作成します。
- スカラー UDF のパラレル機能の使用を可能にするユーザー定義関数 UDF をドロップして再作成します。

注: dxxEnableColl は dxxEnableCollection に名前変更され、 dxxDisableColl は dxxDisableCollection に名前変更されます。ストアード・プロシージャー db2xml.dxxDisableDB も追加されます。

ストアード・プロシージャーを呼び出すときは、プロシージャー名に、感嘆符 (!) の代わりに、ピリオド (.) を使用します。たとえば、*db2xml.dxxEnableColumn* を *db2xml!dxxEnableColumn* の代わりに使用します。

手順:

XML 対応のデータベースおよび XML 対応の列を移行するには次のようにします。

1. DB2 UDB XML Extender バージョン 8.1 をインストールします。
2. DB2 UDB コマンド行から次のように入力します。

```
db2 connect to database_name
db2 bind @dxxMigv.lst
dxxMigv database_name
```

前のリリースから XML Extender フィックスパックへのマイグレーション

マイグレーション・プログラムは XML Extender を前のリリースから更新済みのフィックスパックへ移行します。データベースをバックアップしてから、マイグレーション・プログラムを実行します。

データベースを移行するには、以下のように行います。

マイグレーション・ファイルは、フィックスパック CD インストール・イメージにあります。このステップはフィックスパック・ファイルを抽出したディレクトリーで実行する必要があります。

UNIX または Windows を使用している場合、以下のステップを実行してください。

1. DB2 UDB コマンド行から次のように入力します。

   ```
   db2 connect to database_name
   db2 bind @dxxMigv.lst
   ```

2. DB2 UDB コマンド行から次のように入力します。

   ```
   dxxMigv database_name
   ```

マイグレーション・ステップに失敗すると、データベースを使用不可にするのに失敗したり、新しい UDF にアクセスできないといった、予測できない結果や問題が発生する可能性があります。

関連概念:

- 40 ページの『XML Extender データを旧バージョンからマイグレーションする』

XML Extender 管理計画の概要

XML Extender には 3 つの管理方式が準備されています。 XML Extender 管理ウィザード、XML Extender 管理コマンド、および XML Extender のストアード・プロシージャーです。

- 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ています。
- 管理タスクは、プログラムから管理のためのストアード・プロシージャーを呼び出すことによって、実行できます。
- 管理タスクは、XML Extender 管理ウィザードによって進められます。これをクライアント・ワークステーションから使用することができます。

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は、以下の決定を最初に行う必要があります。

- データベースのデータから XML 文書を合成するかどうか。
- 既存の XML 文書を保管するかどうか。 XML 文書を保管する場合は、原形の XML 文書として列に保管するのか、またはリレーショナル・データに分解するのかも決定する必要があります。

この決定をした後で、以下の決定を行います。

- 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® UDB リレーショナル表にマップする方法

管理ウィザードのセットアップ

XML Extender の管理タスクは、データベース列を XML 使用可能にすること、および XML データを DB2 UDB リレーショナル構造にマップすることです。これらの管理タスクは、XML Extender ウィザードを使用して、実行することができます。このセクションでは、管理ウィザードをセットアップして、呼び出す方法を説明します。ウィザードは、Windows の「スタート」メニューまたはコマンド行プロンプトから呼び出すことができます。

前提条件:

ウィザードをセットアップする前に、ご使用のオペレーティング・システム用の README ファイルの説明に従って、管理ウィザードのインストールおよび構成を行う必要があります。 CLASSPATH 環境変数に必要とされるクラス・ファイルを含める必要があります。

改行は別として、CLASSPATH 環境変数が次の例のようになっていることを確認してください。

```
C:¥db2_install¥java¥db2java.zip;C:¥db2_install¥xml-apis.jar;
C:¥db2_install¥dxxadmin.jar;C:¥db2_install¥xerces.jar;
```

ここで、*db2_install* はインストール・ディレクトリーのことです。

手順:

XML Extender 管理ウィザードをセットアップするには、次のようにします。

1. SDK を使用してウィザードを呼び出します。 Software Development Kit または Java Runtime Environment (JRE) を使用できます。
   - JRE を使用するには、次のように入力します。

     ```
     jre -classpath classpath com.ibm.dxx.admin.Admin
     ```
   - SDK を使用するには、次のように入力します。

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

   ここで、*classpath* は、管理ウィザード・クラス・ファイルが入っている場所を示す %CLASSPATH% 環境変数を指定します。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 変数が、 dxx_install/tools および dxx_install/java ディレクトリーを指している必要があります。このディレクトリーには、dxxadmin.jar、xerces.jar、 xml-apis.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
   ```

   classpath は、dxx_install/dxxadmin ディレクトリー (ここから XML Extender 管理ウィザードを実行する) 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することもできます。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:mydb2 userid=db2xml password=db2xml
   driver=COM.ibm.db2.jdbc.app.DB2Driver
```

2. 「ログオン (Logon)」ウィンドウから、XML データの処理時に使用したいデータベースにログオンします。
3. **「アドレス (Address)」**フィールドに、接続するデータ・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 (デフォルトおよび推奨) :

   ```
   jdbc:db2:database_name
   ```

   ここで、*database_name* は、接続し、XML 文書を保管しようとするデータベースです。

   たとえば次のようにします。

   ```
   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 Extender 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、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、 ¥winnt¥system32¥driver¥etc¥services ファイル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
   ```

4. **「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 UDB 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。
5. **「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.jcc.DB2DRIVER
   ```

6. **「終了 (Finish)」**をクリックします。ウィザードを呼び出し、「ランチパッド (LaunchPad)」ウィンドウに進みます。

この手順が完了したら、「ランチパッド (LaunchPad)」ウィンドウでウィザードを起動できます。ウィザードを使用して、以下の機能を実行することができます。

- データベースを使用可能または使用不可にする。
- DTD リポジトリーに DTD を追加する。
- XML 列を処理する。
- XML コレクションを処理する。

アクセスおよび保管の方式

XML Extender 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。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 UDB 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方式、および XML データを構成する方法が決まります。

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 UDB 表に関連付けます。 45 ページの図 4 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。

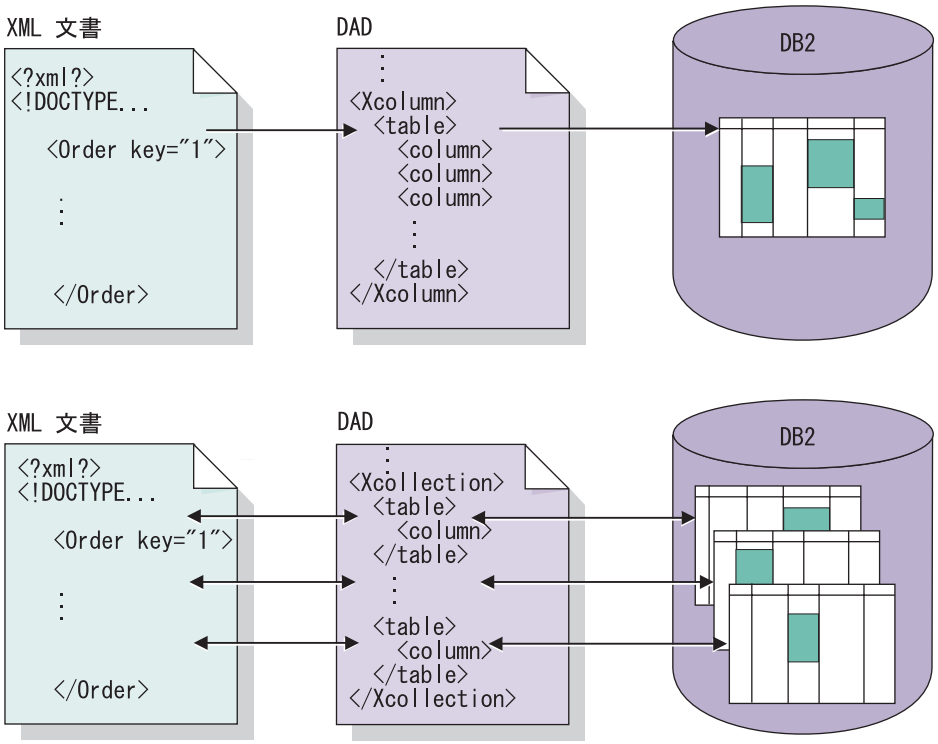


図 4. DAD ファイルは、XML 文書構造を DB2 UDB リレーショナル・データ構造にマップして、アクセスおよび保管の方式を指定します。

DAD ファイルは DTD などの主要ファイルのロケーションを定義して、XML 文書構造が DB2 UDB 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。

関連概念:

- 45 ページの『XML 列方式を使用する場合』
- 46 ページの『XML コレクション方式を使用する場合』

関連資料:

- 144 ページの『XML Extender における保管関数の概要』

XML 列方式を使用する場合

以下のいずれかの場合に XML 列を使用します。

- 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。これらを DB2® に保管するのは、整合性、アーカイブ、および監査のためです。
- XML 文書が頻繁に読み取られるが、更新はしないとき。
- ファイル名データ・タイプを使用して XML 文書 (DB2 UDB の外部にある) 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 UDB 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の値に基づく範囲検索を行う必要があり、どのエレメントまたは属性が頻繁に検索引き数に使用される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 UDB Text Extender を使用して構造テキスト検索を行いたいとき。

XML コレクション方式を使用する場合

以下のいずれかの場合に XML コレクションを使用します。

- 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・ビューを作成したいとき。
- 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。および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 列についての計画

XML Extender を使用して文書を保管する前に、XML 文書の構造を理解して、文書内のエレメントと属性に索引を作成する方法を決める必要があります。文書の索引作成方法を計画するときは、以下について決定する必要があります。

- XML 文書を保管するのに使う XML ユーザー定義タイプ
- アプリケーションで頻繁に検索する XML エレメントと属性。これらの内容をサイド表に保管し、索引を作成することによって、パフォーマンスを改善することができます。
- DTD によって、列内の XML 文書の妥当性検査を行うかどうか
- サイド表の構造およびそれら表に索引を作成する方法

XML 列用の XML データ・タイプ

XML Extender は、XML 文書を保持するための列を定義するのに使用する XML ユーザー定義タイプを提供します。これらのデータ・タイプは、表 4 で説明されています。

表 4. XML Extender 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 文書全体を VARCHAR データ・タイプとして DB2® 内に保管します。 DB2 に保管される小さな文書 (3K 未満) に使用されます。

表 4. XML Extender UDT (続き)

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
XMLCLOB	CLOB(*clob_len*)	XML 文書全体を CLOB データ・タイプとして DB2 内に保管します。 DB2 に保管される大きな文書 (3K 以上) に使用されます。
XMLFILE	VARCHAR(512)	XML 文書のファイル名を DB2 に保管し、XML 文書を DB2 サーバーのローカル側にあるファイルに保管します。DB2 外に保管される文書に使用されます。

XML 列のために索引を付けるエレメントと属性

XML 文書の構造とアプリケーションのニーズを理解すれば、どのエレメントと属性が最も頻繁に検索または抽出されるか、あるいは、照会に最も経費のかかるエレメントと属性はどれかを判別することができます。 XML 列用の DAD ファイルは、各エレメントと属性のロケーション・パスを、これらのオブジェクトを含むリレーショナル表 (サイド表) にマップすることができます。この後、サイド表に索引が作成されます。

たとえば、表 5 では、XML 列の「入門」シナリオで使用しているデータ・タイプ、およびエレメントと属性のロケーション・パスの例を示しています。データは、頻繁に使用される情報として指定されており、ロケーション・パスは、そのデータが入っているエレメントと属性を指しています。 DAD ファイルはこれらのロケーション・パスをサイド表にマップします。

表 5. 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

XML 列のための DAD ファイル

XML 列に対しては、DAD ファイルは主として、XML 列に保管される文書に索引を作成する方法を指定します。 DAD ファイルは、XML 列に挿入された文書の妥当性を検査するために使用される DTD を指定します。このファイルは最大 2 GB まで可能です。

XML 列の DAD ファイルは、索引付けのためにサイド表に保管されるすべての 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 ファイルで <Xcolumn> タグを使用します。 <Xcolumn> タグは XML データ用に使用可能な DB2 UDB 列に、XML データを XML 文書全体として保管および検索することを指定します。

XML 使用可能列は、XML Extender の UDT です。アプリケーションには、どのユーザー表内の列も組み込むことができます。 XML 列データにアクセスするには、主に、SQL ステートメントおよび XML Extender の UDF を使用します。

関連概念:

• 64 ページの『サイド表の計画』

XML コレクションの計画

XML コレクションを計画する際には、文書を DB2® データから合成したり、XML 文書をリレーショナル・データに分解したり、あるいはその両方を行うための、いろいろな考慮事項があります。以下のセクションでは、XML コレクションの計画についての問題、および合成と分解の考慮事項について説明します。

妥当性検査

アクセスおよび保管の方法を選択した後、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 またはスキーマを使用して XML データの妥当性検査を行います。 DTD またはスキーマを使用すると、XML 文書が有効であることを保証することができます。

DTD を使用して妥当性検査を行うには、XML Extender リポジトリーに DTD を入れておく必要がある場合があります。

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 Extender はすでに DB2 に挿入されたデータの妥当性検査を行いません。

考慮事項:

- 合成には、1 つの DTD しか使用できません。
- 合成には複数のスキーマを使用できます。
- 文書の妥当性検査を行う選択をしない場合、XML 文書により指定された DTD は処理されません。妥当性検査を実行できない文書フラグメントを処理する場合でも、エンティティーおよび属性デフォルトを解決するために DTD が処理されることは重要です。

XML コレクションのための DAD ファイル

XML コレクションの場合、DAD ファイルで、XML 文書の構造が DB2 UDB 表にマップされます。この表から文書を合成したり、この表へ文書を分解します。

たとえば、XML 文書内に <Tax> と呼ばれるエレメントがあるときは、 <Tax> を TAX と呼ばれる列にマップしなければならない場合があります。 DAD で、XML データとリレーショナル・データとの関係を定義します。

DAD ファイル名は、コレクションを使用可能にする際、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する必要があります。 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 Extender 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 ファイルでタグを使用します。 <Xcollection> タグは 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 データを含む一組のリレーショナル表の仮想名です。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、従来からの業務データの既存の表、または XML Extender が最近作成した表などです。

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
エレメントを識別します。これは、ルート・エレメントまたは子エレメントです。

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。

50 ページの図 5 は、DAD ファイルで使用されているマッピングの断片を示しています。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 "dxx_install/samples/db2xml/dtd/dad.dtd">
<DAD>
  ...
<Xcollection>
<SQL_stmt>
      ...
</SQL_stmt>
<prolog>?xml version="1.0"?</prolog>
<doctype>!DOCTYPE Order SYSTEM "dxx_install/samples/db2xml/dtd/
getstart.dtd"</doctype><root_node>
   <element_node name="Order">     --> エレメント <Order> を識別する。
     <attribute_node name="key">   --> 属性 "key" を識別する。
        <column name="order_key"/> --> 列の名前  "order_key" を定義する。
                                       エレメントと属性はこれにマップされる。
                                     </attribute_node>
     <element_node name="Customer">  -->  <Order> の子エレメントを
                                          <Customer> として識別する。
        <text_node>                   --> エレメント <Customer> の
                                          CDATA テキストを指定する。
          <column name="customer">    --> 列の名前 "customer" を定義する。
                                          子エレメントはこれにマップされる。
        </text_node>
     </element_node>
        ...
   </element_node>

      ...
<root_node>
</Xcollection>
</DAD>
```

図 5. XML コレクション用の DAD ファイル内のノード定義

上記の図では、SQL ステートメントの最初の 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 Extender は、<stylesheet> を使用して、スタイルシートのための処理命令もサポートします。 <stylesheet> エレメントは、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>
```

Websphere Studio Application Developer 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、XML Extender 管理ウィザードでサポートされていません。

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する場合は、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 を選択する必要があります。 XML コレクションでは、XML 文書で使用される階層構造がリレーショナル構造とマップされ

なければならないので、ユーザーは、これら 2 つの構造の対比を理解している必要があります。 図 6 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。

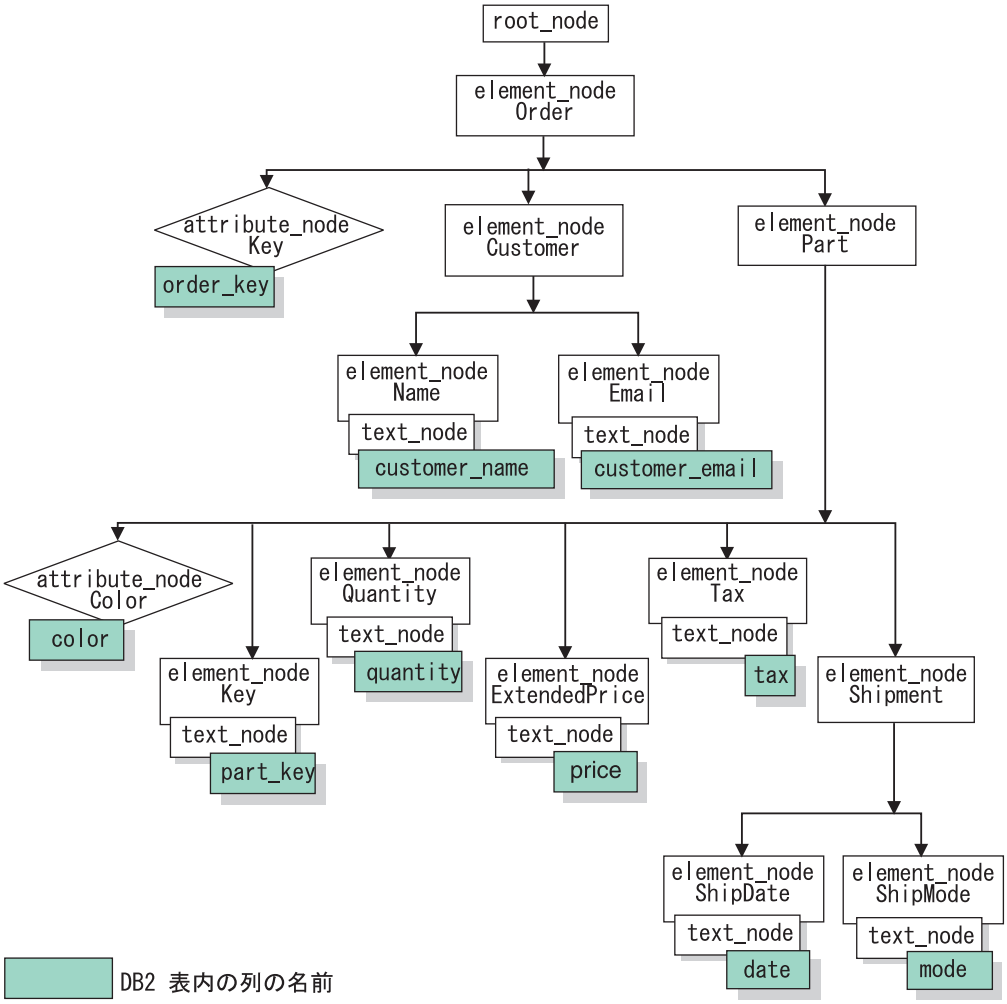


図 6. リレーショナル表の列にマップされた XML 文書構造

XML Extender は、複数のリレーショナル表内にある、リレーショナル・データを使用する XML 文書を合成または分解するとき、マッピング体系を使用します。 XML Extender 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慮し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 ファイルの <Xcollection> エレメントで指定します。 XML Extender が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 *(RDB_node) マッピング* の 2 つがあります。

SQL マッピング

単一の SQL ステートメントを使用して、リレーショナル・データから XML 文書へ直接にマッピングできます。 SQL マッピングは合成に使用されますが、分解には使用されません。 SQL マッピングは、SQL_stmt エレ

メントとともに DAD ファイル内で定義されます。 SQL_stmt エレメントの内容は有効な SQL ステートメントです。 SQL_stmt エレメントは、SELECT 文節内の列を、XML 文書で使用される XML のエレメントまたは属性にマップします。 SQL ステートメントの SELECT 文節の列名は、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 UDB 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。すべての表を結び付けるため、これらの表に主キーと外部キーの関係付けをお勧めします。

RDB_node マッピング

XML エレメントの内容または XML 属性の値のロケーションを定義して、XML Extender がどこに XML データを保管し、どこで取得するかを判別できるようにします。

この方式は、XML Extender が提供する *RDB_node* を使用します。ここには、表に関する 1 つ以上のノード定義、オプションの列、およびオプションの条件が入っています。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 を作成する必要があります。 53 ページの図 7 は、サンプル DAD ファイルの断片を、3 つのリレーショナル表にあるデータから XML 文書セットを合成する XML コレクション SQL マッピングとともに示しています。

```
<?xml version="1.0"?>
<!DOCTYPE DAD SYSTEM "dxx_install/samples/db2xml/dtd/dad.dtd">
<DAD>
  <dtdid>dxx_install/samples/db2xml/dtd/dad/
   getstart.dtd</dtdid>
  <validation>YES</validation>
  <Xcollection>
    <SQL_stmt>
       SELECT o.order_key, customer, p.part_key, quantity, price, tax, date,
              ship_id, mode, comment
         FROM order_tab o, part_tab p,
               table(select
               substr(char(timestamp(generate_unique())),16)

               as ship_id, date, node, from ship_tab)  shipid
        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 "dxx_install
     /samples/db2xml/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 7. SQL マッピング体系

XML Extender 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・プロシージャーでは、両方のタイプのマッピングがサポートされていますが、 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

以下のセクションでは、各タイプの XML コレクションのマッピング体系の要件について説明します。

SQL マッピング用マッピング・スキーマの要件

このマッピング体系では、SQL_stmt エレメントを DAD <Xcollection> エレメント内に指定する必要があります。 SQL_stmt> には、複数のリレーショナル表を照会の述部で結合する SQL ステートメントが 1 つ含まれている必要があります。さらに、以下の文節が必要です。

- **SELECT 文節**

– 列の名前がユニーク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にまとめ、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、表が XML 文書の階層構造にマップされる方法に従って、それらの列をグループ化します。 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 列と 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。
– 合成アルゴリズムは、処理を支援するためのヒントとして候補キー情報を使用します。単一列候補キーを使用して、選択した文節内の各表レベルを始めます。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ユーザー定義関数 generate_unique() を使用して、その表のためのキーを生成しなければなりません。生成されたキーは出力文書に存在する必要はありませんが、SELECT 文節内に存在する必要があります。上記の例で、o.order_key は ORDER_TAB の主キー、そして part_key は PART_TAB の主キーです。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 表は主キーをもたないため、主キーを生成する必要があります。この場合は ship_id です。この主キーは、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 文節を使用して主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、単一の主キーが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
```

この例では、単一列候補キーが関数 generate_unique() で生成され、それに別名 ship_id が与えられています。
  – 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**
  – コレクション内の表を結び合わせる結合条件を指定します。たとえば次のようにします。

```
WHERE p.price > 2500.00 AND s.date > '2003-06-01' 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 エレメント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。
  - 単一列候補キーを、対応する表の階層レベルの順番でリストします。
  - 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 エレメントには以上の要件がありますが、述部の式で表内の列を使用する場合、WHERE 文節に任意の述部を指定できるので、 SQL_stmt はたいへん強力です。

RDB_node マッピング用マッピング・スキーマの要件

このマッピング方式を使用するときは、エレメント SQL_stmt は、 DAD ファイルの <Xcollection> エレメント内で使用できません。その代わりに、先頭 element_node の子、および各 attribute_node と text_node の子として、RDB_node エレメントを使用します。

- **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。以下の要件に基づいて、先頭 element_node に対する RDB_node を指定します。

  - 行終了文字を、条件ステートメントで使用できます。
  - 条件エレメントでは、1 つの列名を制限なく何回でも参照できます。
  - XML 文書に関連したすべての表を指定します。たとえば、以下のマッピングでは、先頭の element_node である element_node <Order> の RDB_node に、 3 つの表を指定しています。

```
<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>
```

  ルート・ノード条件の述部には、順序の制約事項はありません。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。

  - DAD ファイルで指定する XML コレクションを使用可能にする場合、または分解する場合、主キーを表ごとに指定します。主キーは、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。主キーは、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。複合キーを提供する場合、キーの属性は、複数のキー列名をスペースで区切って指定します。たとえば次のようにします。

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

分解の際に指定された情報は、文書の構成の時には無視されます。

– orderBy 属性を使用して、複数回出現する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。

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

各 attribute_node と text_node に RDB_node を指定し、ストアード・プロシージャーに、どの表から、またどの列から、およびどんな照会条件の下でデータを入手するかを知らせる必要があります。表および列の値は指定しなければなりません。条件値はオプションです。

– 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。 <condition> に許容される構文を以下に示します。

  - 列名
  - 演算子
  - リテラル

  上記の例では、条件は `price > 2500.00` と指定されています。この条件に適合するデータのみが、生成される XML 文書に入ります。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 ファイルで指定されている 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します。

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

分解の際に指定された情報は、文書の構成の時には無視されます。

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。

RDB ノード・マッピング用に分解する表サイズの要件

分解では RDB_node マッピングを使って、エレメントと属性値の表の行への抽出によって XML 文書を DB2 UDB 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 10240 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 10240 行までを入れることができます。

複数回出現エレメント (XML 構造内で複数回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、各文書に挿入される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。複数回出現するエレメントを使用するときは、1 つの文書から 1 つの表への分解は、最大 10240 行であることを考慮してください。

関連概念:

• 177 ページの『XML コレクションの DAD ファイル』

関連タスク:

• 60 ページの『リポジトリー表への DTD の保管』

XML 文書の自動的妥当性検査

アクセスおよび保管の方式を、XML 列方式または XML コレクション方式のいずれにするか選択した後、XML 文書の*妥当性検査* を行うかどうかを決めることができます。また、XML コレクションから合成した XML 文書も妥当性検査をすることができます。

DAD ファイルで妥当性検査に YES を指定して XML データを自動的に妥当性検査することもできます。 DB2® に保管する際に文書を妥当性検査するには、<dtdid> エレメント内か、オリジナル文書の <!DOCTYPE> 仕様に DTD を指定する必要があります。 DB2 の XML コレクションから合成する際に文書を妥当性検査するには、<dtdid> エレメント内か DAD ファイルの <doctype> エレメント内に DTD を指定する必要があります。

文書の妥当性検査を行うかどうかを決定する際には、以下の事項を考慮に入れてください。

- この DTD ID は、XML 文書の妥当性検査を行う場合にだけ有益です。

  スキーマによって DAD を検証するには、 DAD ファイルをスキーマ・ファイルに関連付けるスキーマ・タグを挿入します。たとえば次のようにします。

  ```
  <schemabindings>
  <nonamespacelocation location="path/schema_name.xsd"/>
  </schemabindings>
  ```

- 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。

- 妥当性検査を選択するかどうかにかかわらず、エンティティーの値と属性のデフォルトを設定するために、DTD の処理が必要となる場合があります。
- DAD に妥当性検査を NO と指定すると、XML 文書によって指定される DTD は処理されません。
- XML データの妥当性検査を行うと、パフォーマンスに影響を与えます。

XML 用のデータベースの使用可能化

XML Extender を使用して DB2 UDB から XML 文書を保管または検索する前に、データベースを XML 用に使用可能にします。 XML Extender は、現行のインスタンスを使用して、接続してい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、XML Extender は以下の作業を行います。

- ユーザー定義タイプ (UDT)、ユーザー定義関数 (UDF)、および XML Extender 用のストアード・プロシージャーをすべて作成します。
- コントロール表を作成して、そこに XML Extender が必要とするメタデータを取り込みます。
- ユーザー定義の表スペースに DB2XML スキーマを作成して、必要な特権を割り当てます。

  XML 関数の完全修飾名は db2xml.*function-name* です。 db2xml は SQL オブジェクトを論理グループ化する ID です。 UDF または UDT を参照するところでは、どこでも完全修飾名を使用できます。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 UDB は関数パスを使用して、関数またはデータ・タイプを判別します。

手順:

データベースは、管理ウィザードから、またはコマンド行から使用可能にすることができます。この作業をコマンド行から行うには、コマンド行から、使用可能にしたいデータベースを指定して **dxxadm** を入力します。

次の例は、既存のSALES_DB という名前のデータベースを使用可能にします。

```
dxxadm enable_db SALES_DB
```

管理ウィザードを使用してデータベースを使用可能にするには、以下の作業を行います。

1. 管理ウィザードを開始し、「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)」**、をクリックします。

   データベースがすでに使用可能になっている場合、 **「データベースを使用不可にする (Disable database)」** というボタンが表示されます。 データベースが使用不可である場合は、**「データベースを使用可能にする (Enable database)」** ボタンが表示されます。

   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。

データベースを使用可能にした後で、XML Extender の UDT、UDF、ストアード・プロシージャーを使用することができます。

関連概念:

• 40 ページの『XML Extender データを旧バージョンからマイグレーションする』

XML 表の作成

この作業は、XML 列を定義し、使用可能にするという大きな作業の一部です。

XML 表は、原形の XML 文書を保管するために使用されます。データベース内の全文書を DB2 UDB XML Extender を使用して保管するためには、XML ユーザー定義タイプ (UDT) をもつ列が入るような表を作成する必要があります。 DB2 UDB XML Extender には、XML 文書を列データとして保管するための 3 つのユーザー定義タイプが準備されています。すなわち XMLVARCHAR、XMLCLOB および XMLFILE の 3 つの UDT です。表に XML タイプの列が 1 つあれば、その表を XML 用に使用可能にすることができます。

管理ウィザードまたはコマンド行によって、XML タイプの列を加えるための新しい表を作成することができます。

手順:

コマンド行を使用して、XML タイプの列を入れる表を作成するには、次のようにします。

DB2 UDB コマンド・プロンプトをオープンし、Create Table ステートメントを入力します。

たとえば、販売アプリケーションの場合、SALES_TAB という名前の表の ORDER という名前の列に、XML 形式の行項目注文を保管したいとします。この表には、INVOICE_NUM および SALES_PERSON という列もあります。注文は、多くないため、XMLVARCHAR タイプを使って保管するとします。主キーは、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);
```

表を作成した後、次のステップでは、列を XML データ用に使用可能にします。

関連概念:

• 64 ページの『サイド表の計画』
• 291 ページの『第 13 章 XML Extender の管理サポート表』

リポジトリー表への 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、DTD リポジトリー表 (つまり DTD_REF と呼ばれる DB2 UDB 表) に保管されます。 DTD_REF 表には 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 Extender は DTD_REF 表を作成します。 DTD を挿入するには、コマンド行または管理ウィザードのいずれも使用できます。

手順:

管理ウィザードを使用して DTD を挿入するには、次のようにします。

1. 管理ウィザードを開始し、「ランチパッド (Launchpad)」ウィンドウから**「DTD のインポート (Import a DTD)」**をクリックして、既存の DTD ファイルを現行データベースの 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)」ウィンドウが開きます。
2. **「DTD ファイル名 (DTD file name)」**フィールドに DTD ファイル名を指定します。
3. **DTD ID** フィールドに DTD ID を入力します。

   DTD ID は DTD の ID です。また、ローカル・システム上の DTD のロケーションを指定するパスでもあります。 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。
4. オプション: **「作成者 (Author)」**フィールドに DTD 作成者名を入力します。
5. **「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。

コマンド行から DTD を挿入するには、表 6 を使用して SQL INSERT ステートメントを発行します。たとえば次のようにします。

```
DB2 INSERT into DB2XML.DTD_REF values('dxx_install
/samples/db2xml/dtd/getstart.dtd',
DB2XML.XMLClobFromFile('dxx_install/dxxsamples/dtd/getstart.dtd',
0, 'user1', 'user1', 'user1');
```

表 6. DTD リポジトリー表のための列定義

列名	データ・タイプ	説明
DTDID	VARCHAR(128)	DTD の ID。
CONTENT	XMLCLOB	DTD の内容。
USAGE_COUNT	INTEGER	この DTD を使用するデータベースにおける 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。

XML 列の使用可能化

XML 文書を DB2 UDB データベースに保管するには、文書を入れる列を XML 用に使用可能にする必要があります。列を使用可能に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 Extender 管理ウィザードまたはコマンド行が使用できます。列のタイプは XML でなければなりません。

XML Extender は、XML 列を使用可能にするときに、以下の操作を行います。

- 以下の目的で DAD ファイルを読み取ります。
  - DTDID が指定されていれば、DTD_REF 表に DTD があるか、チェックする。
  - 索引用に XML 列についての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・ビューがそれらを作成します。デフォルト・ビューでは、アプリケーション表とサイド表を表示します。
- ルート ID 列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

XML 列は、管理ウィザードまたは DB2 コマンド行によって、使用可能にすることができます。

手順 (管理ウィザードを使用):

管理ウィザードを使用して XML 列を使用可能にするには、次のようにします。

1. 管理ウィザードをセットアップして開始します。
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 Extender 列に関連する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. 表と列を指定します。
   - **「表名 (Table name)」**フィールドから、XML 列を含む表を選択します。
   - **「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。
5. DAD パスおよびファイル名を**「DAD ファイル名 (DAD file name)」**フィールドに指定します。たとえば次のようにします。

   ```
   dxx_install/samples/dad/getstart.dad
   ```

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

デフォルト表スペースには、XML Extender が作成したサイド表が入ります。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。

7. オプション: デフォルト・ビューの名前を**「デフォルト・ビュー (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・ビューは、列が使用可能になるときに自動的に作成されます。デフォルト・ビューは、XML 表と関連するすべてのサイド表を結合します。

8. 推奨: 表の主キーの列名を「**ルート ID (Root ID)**」フィールドに入力します。

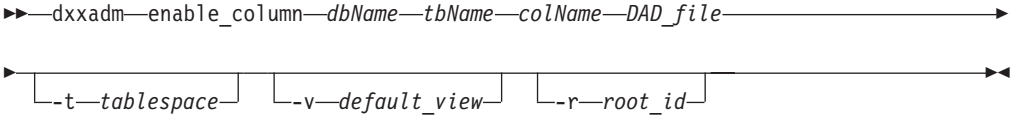
   XML Extender は、すべてのサイド表をアプリケーション表に関連付けるために、一意的な ID として**ルート ID** の値を使用します。 XML Extender がアプリケーション表に DXXROOT_ID 列を追加し、ID を生成します。

9. **「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になると、`column is enabled` 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、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するようにというプロンプトとともに、エラー・メッセージが表示されます。

手順 (コマンド行を使用):

コマンド行を使用して XML 列を使用可能にするには、DXXADM enable_column を使います。

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。

default_view
: オプション。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 Extender が作成した、デフォルト・ビューの名前。

root_id オプションですが、お勧めします。アプリケーション表内の主キーの列名、

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。DB2 Extender 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 の値を使用します。ルート ID が指定されていない場合、XML Extender 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 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:

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

この例では、ORDER 列が SALES_TAB 表で使用可能になります。 DAD ファイルは getstart.dad、デフォルト・ビューは sales_order_view、ルート ID は INVOICE_NUMBER です。

この例を使用すると、表 SALES_TAB の列は次のとおりです。

列名 (Column name)	データ・タイプ (Data type)
INVOICE_NUM	CHAR(6)
SALES_PERSON	VARCHAR(20)
ORDER	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 (Column name)	データ・タイプ (Data type)	パス式
ORDER_KEY	INTEGER	/Order/@Key
CUSTOMER	VARCHAR(50)	/Order /Customer /Name
INVOICE_NUM	CHAR(6)	N/A

PART_SIDE_TAB:

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

SHIP_SIDE_TAB:

列名 (Column name)	データ・タイプ (Data type)	パス式
DATE	DATE	/Order/Part/ Shipment/ShipDate
INVOICE_NUM	CHAR(6)	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主キー 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、行がメイン表に挿入される際に、INVOICE_NUM 列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デフォルト・ビューを指定すると、XML Extender はデフォルト・ビュー sales_order_view を作成します。このビューは、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_side_tab.order_key, order_side_tab.customer,
            part_side_tab.part_key, part_side_tab.price,
            ship_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
```

使用可能化のときに表スペースを指定した場合、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。

サイド表の計画

サイド表は、頻繁に検索される XML 文書内容を抽出するために使用される DB2® の表です。 XML 列は、XML 文書の内容を保持しているサイド表に関連付けられます。 XML 文書がアプリケーション表で更新されると、サイド表の値も自動的に更新されます。

65 ページの図 8 は、サイド表のある XML 列を示しています。

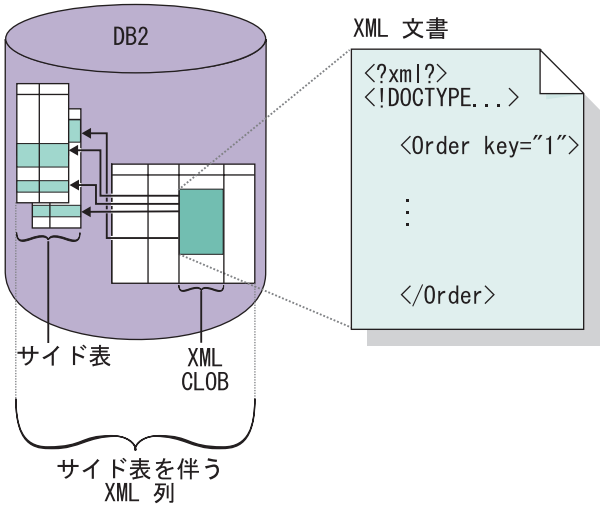


図 8. 内容がサイド表にマップされる XML 列。 XML 文書の内容を保持するサイド表に関連付けられる列には、XML ファイルが入っている。

複数回出現:

エレメントと属性がサイド表に複数回出現する場合は、計画時に以下の問題点について考慮してください。

- XML 文書内のエレメントまたは属性で複数回出現があるものについては、XML 文書構造の複雑さのために、複数回出現のある 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。すなわち、エレメントまたは属性は、そのロケーション・パスが複数回出現しても、ただ 1 つの列を持つ表にマップされる必要があるということです。この表にその他の列を入れることはできません。
- ロケーション・パスが複数回出現する文書の場合、XML Extender は、複数回出現するエレメントの順序を追跡するために、各サイド表内に INTEGER タイプの DXX_SEQNO という名前の列を追加します。 DXX_SEQNO を使うと、SQL 照会で ORDER BY DXX_SEQNO と指定することで、元の XML 文書と同じ ORDER BY でエレメントを取り出すことができます。

デフォルト・ビュー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 という名前のユニーク 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ビューを指定することができます。デフォルト・ビューを使うと、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、次のような照会が考えられます。

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

この SQL ステートメントは、SALES_TAB 内の営業担当者の内、注文が ORDER 列に保管されていて、PRICE が 2500.00 を超える営業担当者の名前を戻します。

デフォルト・ビューを照会すると、アプリケーション表とサイド表の 1 つのビュー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・ビューの作成をお勧

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ビューを作成することができます。

サイド表の索引付け

この作業は、XML 列を定義し、使用可能にするという大きな作業の一部です。

サイド表では、XML データを DAD ファイルの作成時に指定した列内に納めます。 XML 列を使用可能にし、サイド表を作成した後、サイド表に索引を作成することができます。これらの表に索引を作成すると、XML 文書に対する照会のパフォーマンスの改善に役立ちます。

手順:

DB2 UDB コマンド行からサイド表に索引を作成するには、DB2 CREATE INDEX SQL ステートメントを使用します。

これは、DB2 UDB コマンド行から行います。

以下の例では、DB2 コマンド・プロンプトを使用して 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)
```

SQL マッピングを使った XML 文書の合成

XML 文書は、コマンド行から、または管理ウィザードを使用して、 SQL マッピングを使って合成することができます。

文書を合成し、XML 文書内のデータを取り込む表および列を定義するために SQL ステートメントを使用したい場合は、SQL マッピングを使用してください。 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。 SQL マッピングを使用して XML 文書を合成するために、DAD ファイルを作成します。

前提条件:

文書を合成するときは、その前にまず、DB2 UDB 表と XML 文書との間の関係をマップする必要があります。このステップでは、XML 文書の階層をマップし、文書内のデータが DB2 UDB 表にマップされる方法を指定します。

手順:

コマンド行から XML 文書を合成するためには、以下のステップを行ってください。

1. テキスト・エディターで新しい文書を作成し、次の構文を入力します。

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

2. <DAD></DAD> タグを挿入します。

   DAD エレメントには、他のすべてのエレメントが含まれます。

3. DTD またはスキーマによって DAD を検証するために使用するタグを挿入します。

   - DTD によって合成された XML 文書を検証するには、 DAD ファイルを XML 文書 DTD に関連付ける DTDID タグを挿入します。たとえば次のようにします。

     ```
     <dtdid>path/dtd_name.dtd>
     ```

   - スキーマによって合成された XML 文書を検証するには、スキーマ・ファイルと関連する DAD ファイルをスキーマ・タグに挿入します。たとえば次のようにします。

     ```
     <schemabindings>
     <nonamespacelocation location="path/schema_name.xsd"/>
     </schemabindings>
     ```

   DTD またはスキーマは、XML 文書の妥当性検査を行う場合にだけ有益です。validation タグを使用して、DB2 UDB XML Extender が XML 文書の妥当性検査を行うかどうかを指示します。

   - XML 文書の妥当性検査を行いたい場合は、次のように入力する。

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

   - XML 文書の妥当性検査を行わない場合は、次のように入力する。

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

4. XML データのためのアクセスと保管の方式として XML コレクションを使用することを指定するには、<XCollection> </XCollection> タグを入力します。

5. <Xcollection> </Xcollection> タグの内側に、 <SQL_stmt> </SQL_stmt> タグを挿入して、リレーショナル・データを XML 文書にマップする SQL ステートメントを指定します。次のステートメントは、DB2 UDB 表からデータを照会するために使用するものです。以下は、SQL 照会の例です。

   ```
   <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>
   ```

   リレーショナル・データを XML 文書にマッピングするための例示した SQL ステートメントは、次の要件を満たします。

   - 列は、トップダウンの ORDER BY で、XML 文書構造の階層どおりに指定します。
   - 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。

- Order_tab 表には単一キー列はないので、 generate_unique DB2 UDB 組み込み関数を使って ship_id 列を生成します。
- ORDER BY 文節内では、オブジェクト ID 列は、所属する表のトップダウン階層に対応する ORDER BY でリストされます。 ORDER BY 内の列は、スキーマ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。

6. 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>
```

7. <doctype> </doctype> タグを入力します。このタグには、各合成文書に挿入する DOCTYPE 宣言が含まれています。たとえば次のようにします。

```
<doctype>! DOCTYPE Order SYSTEM "dxx_install
/samples/db2xml/dtd/getstart.dtd"</doctype>
```

8. ルート・エレメントと、XML 文書を形成するエレメントおよび属性を指定します。
   a. ルート・エレメントを定義するために、<root></root_node> タグを追加します。 XML 文書を形成するすべてのエレメントと属性は、この root_node 内に指定されます。
   b. <element_node>、<attribute_node>、 <text_node>、および <column> タグを使用して、合成文書内のエレメントと属性の名前を指定し、 SQL ステートメント内で指定される列にマップします。

      <column> タグ
      エレメントまたは属性値についてデータを検索する列を指定します。

      <element_node> タグ
      XML 文書内のエレメントを指定します。 element_node タグの名前属性を、エレメントの名前に設定します。各 element_node には子の element_node があってもかまいません。

      <attribute_node> タグ
      XML 文書内のエレメントの属性を指定します。属性は、エレメント・ノード内でネストされます。 attribute_node タグの名前属性を、属性の名前に設定します。

      <text_node> タグ
      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。最下位の element_node ごとに、<text_node> タグを定義します。これは、文書合成時に DB2 から抽出される文字データが、このエレメントに含まれることを示しています。最下位の element_node ごとに、<column> タグを使用して、XML 文書合成時にデータを抽出する列を示します。列のタグは基本的に <attribute_node> または <text_node> タグ内にあります。定義する列名はすべて、DAD ファイルの先頭の <SQL_stmt> SELECT 文節内に入れる必要があります。

9. 終了タグが適切な場所にあることを確認します。
   a. 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。

b. 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。

c. 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。

10. *file.dad* という名前を付けてファイルを保管します。 *file* は、使用するファイルの名前です。

以下の Windows の例は、完全な DAD を示しています。

```
<?xml version'"1.0"?>
<!DOCTYPE DAD SYSTEM "C:¥dxx_xml¥test¥dtd¥dad.dtd">
<DAD>
<validation>NO</validation>
<Xcollection>
<SQL_stmt> select o.order_key, customer_name, customer_email,
 p.part_key, color, qty, price, tax, ship_id, date, mode from order_tab o,
 part_tab p, (select db2xml.generate_unique() 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_install¥samples¥db2xml¥dtd/Order.dtd"
 </doctype>
 <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>
    <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="qty"/></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>
      </element_node>
     </element_node>
     </root_node>
     </Xcollection>
     </DAD>
```

RDB_node マッピングを使った XML コレクションの合成

RDB_node マッピングでは、<RDB_node> タグを使用して、エレメントまたは属性ノードに、 DB2 UDB 表、列、および条件を指定します。 XML 類似の構造を使用して XML 文書を合成したい場合は、この方式を使用してください。 <RDB_node> は、以下のエレメントを使用します。

表 (table) エレメントに対応する表を定義します。

列 (column) 対応するエレメントを含む列を定義します。

条件 (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:	以下の RDB 子エレメントが使用されます。		
	表	列	条件
ルート・エレメント	はい	いいえ	はい[1]
属性	はい	はい	オプショナル
テキスト	はい	はい	オプショナル

[1] 複数の表を使用する場合に必要

RDB_node マッピングを使用して XML 文書を合成するには、管理ウィザードまたはコマンド行を使用できます。

制約事項:

RDB_node マッピングを使用して XML コレクションを合成する場合は、あるエレメントのステートメントは、すべて同じ表内の列にマップする必要があります。

手順:

コマンド行から RDB_node マッピングを使用して XML 文書を合成するには、以下のようにします。

1. テキスト・エディターをオープンして、次の構文を入力し、DAD ヘッダーを作成します。

   ```
   <?xml version="1.0"?>
   <!DOCTYPE DAD SYSTEM "path/dad.dtd">
   ```

   ここで、*path*/dad.dtd は、DAD のための DTD のパスとファイル名です。
2. <DAD></DAD> タグを挿入します。 このエレメントには、すべての他のエレメントが含まれます。
3. DTD またはスキーマによって DAD を検証するために使用するタグを挿入します。
   - DTD によって DAD を検証するには、DAD ファイルを XML 文書 DTD に関連付ける DTDID タグを挿入します。たとえば次のようにします。

     ```
     <dtdid>path/dtd_name.dtd</dtdid>
     ```

- スキーマによって DAD を検証するには、 DAD ファイルをスキーマ・ファイルに関連付けるスキーマ・タグを挿入します。たとえば次のようにします。

```
<schemabindings>
<nonamespacelocation location="path/schema_name.xsd"/>
</schemabindings>
```

dtdid またはスキーマは、XML 文書の妥当性検査を行う場合にだけ有益です。validation タグを使用して、DB2 UDB XML Extender が XML 文書の妥当性検査を行うかどうかを指示します。

- XML 文書の妥当性検査を行いたい場合は、次のように入力します。

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

- XML 文書の妥当性検査を行わない場合は、次のように入力します。

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

4. XML データのためのアクセスと保管の方式として XML コレクションを使用することを指定するには、<XCollection> </XCollection> タグを挿入します。
5. 以下のプロローグ情報を追加します。

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

6. <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 "dxx_install
/samples/db2xml/dtd/getstart.dtd"</doctype>
```

7. <root_node></root_node> タグを挿入します。 root_node タグ内で、XML 文書を形成するエレメントおよび属性を指定します。
8. <root_node> タグ内で、XML 文書内のエレメントおよび属性を、 DB2 UDB 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。
element_node、text_node、および attribute_node に対して RDB_node エレメントを使用します。これらのノードは、XML データから DB2 UDB データへのパスを提供します。 XML 文書のエレメントと属性をマップするには、以下のようにします。
   a. 先頭 element_node に対する RDB_node を指定する。このエレメントは、XML 文書に関連するすべての表を指定します。先頭 element_node に対する RDB_node を指定するには、ステップ 9 の root_node タグの後ろに <RDB_node> タグを挿入する。
      - attribute_node のための RDB_node を指定する。
      - text_node のための RDB_node を指定する。
   b. 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表ノードを定義する。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>
```

DAD ファイルを使用して、XML 文書を分解する場合は、各表に対して主キーを指定する必要があります。主キーは、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。主キーは、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。コレクションを使用可能に

する場合は、表ごとに主キーを指定する必要があります。下記の例は、element_node に指定されている各表に対して、キー列を指定する方法を示しています。

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

関連概念:

- 109 ページの『XML コレクションのマッピング体系』
- 118 ページの『ロケーション・パス』
- 177 ページの『XML コレクションの DAD ファイル』
- 114 ページの『RDB_Node マッピングのための要件』
- 214 ページの『XML Extender 合成ストアード・プロシージャー - 概要』

関連タスク:

- 72 ページの『RDB_node マッピングを使った XML コレクションの分解』
- 98 ページの『XML コレクションのデータ管理』
- 106 ページの『XML コレクションのデータ更新および削除』

RDB_node マッピングを使った XML コレクションの分解

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、エレメントまたは属性ノードに、DB2 UDB 表、列、および条件を指定します。 <RDB_node> は、以下のエレメントを使用します。

表 (table) エレメントに対応する表を定義します。

列 (column) 対応するエレメントを含む列を定義します。

条件 (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る		
	表	列	条件
ルート・エレメント	はい	いいえ	はい[1]
属性	はい	はい	オプション
テキスト	はい	はい	オプション

(1) 複数の表を使用する場合に必要

コマンド行を使用する手順:

コマンド行を使用して XML 文書を分解するには、次のようにします。

1. 任意のテキスト・エディターを使用してファイルを作成します。次の構文を入力して、DAD ヘッダーを作成します。

```
<?xml version="1.0"?>
<!DOCTYPE DAD SYSTEM "path/dad.dtd">
```

ここで、*path*/dad.dtd は、DAD のための DTD のパスとファイル名です。

2. <DAD></DAD> タグを挿入します。
3. DTD またはスキーマによって DAD を検証するために使用するタグを挿入します。
   - DTD によって DAD を検証するには、DAD ファイルを XML 文書 DTD に関連付ける DTDID タグを挿入します。たとえば次のようにします。

     ```
     <dtdid>path/dtd_name.dtd</dtdid>
     ```

   - スキーマによって DAD を検証するには、 DAD ファイルをスキーマ・ファイルに関連付けるスキーマ・タグを挿入します。たとえば次のようにします。

     ```
     <schemabindings>
     <nonamespacelocation location="path/schema_name.xsd"/>
     </schemabindings>
     ```

   dtdid またはスキーマは、XML 文書の妥当性検査を行う場合にだけ有益です。validation タグを使用して、DB2 UDB XML Extender が XML 文書の妥当性検査を行うかどうかを指示します。

   - XML 文書の妥当性検査を行いたい場合は、次のように入力します。

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

   - XML 文書の妥当性検査を行わない場合は、次のように入力します。

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

4. XML データのためのアクセスと保管の方式として XML コレクションを使用することを指定するには、<XCollection> </XCollection> タグを挿入します。
5. 以下のプロローグ情報を追加します。

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

6. <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 "dxx_install
      /samples/db2xml/dtd/getstart.dtd"</doctype>
   ```

7. <root_node></root_node> タグを使用して root_node を定義します。
8. root_node 内で、XML 文書内のエレメントおよび属性を、 DB2 UDB データに対応するエレメント・ノード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 UDB 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) element_node エレメント内に 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。

         ```
         <element_node name="name">
         <RDB_node>
         </RDB_node>
         </element_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。

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

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) 各表ごとに主キーを指定します。主キーは、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。主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主キーを定義します。

```
<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>
```

キー属性は、分解と、コレクションの使用可能化のために必要です。

b. DB2 UDB 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 UDB 表に内容を持つことを指定します。子エレメントはありません。

attribute_node
属性を指定します。

子エレメント　element_node の子です。

text_node には、内容を DB2 UDB 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 UDB 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。

表 (table)　エレメントまたは属性をマップする表を指定します。

列 (column)　エレメントまたは属性をマップする列を指定します。

条件 (condition)
オプションで、列に挿入を行う条件を指定します。

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> タグを指定して、XML エレメントを DB2 UDB 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> タグは、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 UDB 表の 1 つの列にマップする <attribute_node> タグを 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 UDB 表および列にマップするための RDB_node が含まれています。

たとえば、Order エレメントの属性キーがあるとします。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。

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>
```

9. 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>
```

10. 終了タグが適切な場所にあることを確認します。
    a. 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。
    b. 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。
    c. 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。

関連概念:

- 226 ページの『XML Extender 分解ストアード・プロシージャー - 概要』

関連タスク:

- 102 ページの『XML 文書を分解して DB2 UDB データにする』
- 206 ページの『XML Extender ストアード・プロシージャーの呼び出し』

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手法について説明します。

第 3 章 XML 列

このセクションでは、DB2 を使用して、XML 列にあるデータを管理する方法を説明します。

XML 列のデータの管理

データを保管するために XML 列を使用するときは、XML 文書全体を、そのネイティブ・フォーマットで DB2 内に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。

データベースを XML のために使用可能にすると、XML Extender が提供する次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
この UDT は、DB2 内に文字ラージ・オブジェクト (CLOB) として保管される、XML 文書内容に使用されます。

XMLVARCHAR
この UDT は、DB2 内に VARCHAR として保管される、XML 文書内容に使用されます。

XMLFILE
この UDT は、ローカル・ファイル・システムのファイルに保管される XML 文書に使用されます。

XML UDT データ・タイプの列を入れるために、アプリケーション表を作成または変更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。

表の列を XML 用に使用可能にした後、XML 列を作成し、以下の管理作業を行うことができます。

- 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

これらすべての作業を行うために、XML Extender が提供しているユーザー定義関数 (UDF) を使用してください。 XML 文書を DB2 に保管するためには、デフォルト・キャスト関数を使用します。デフォルト・キャスト関数は、SQL 基本タイプを XML Extender ユーザー定義タイプにキャストして、あるデータ・タイプ (ソース) のインスタンスを異なるデータ・タイプ (ターゲット) のインスタンスに変換します。

関連概念:

- 80 ページの『保管およびアクセス方式としての XML 列』

• 81 ページの『XML 列データの索引の使用』

保管およびアクセス方式としての XML 列

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。 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ています。

たとえば、Web に記事を供給してい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 Extender によって XML 記事の全部または一部を、 *XML 列* である DB2® 表の列に保管できます。これを図 9 に示します。

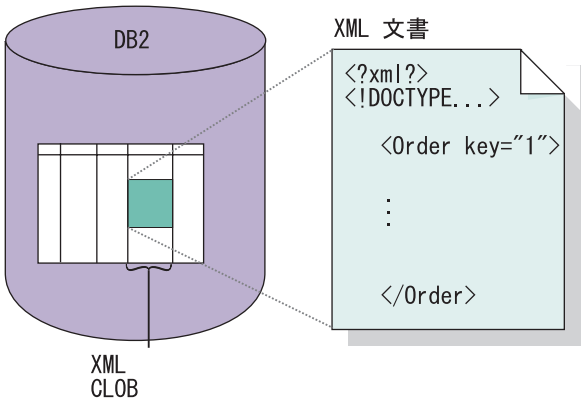


図 9. 構造化 XML 文書を DB2 UDB 表の列に保管

XML 列による保管およびアクセス方式によって、DB2 を使用して XML 文書を管理できます。 XML 文書は XML タイプの列に保管できます。また、文書の内容を照会して、特定のエレメントまたは属性を検出することができます。 1 つ以上の文書について、DTD を DB2 UDB に関連付け、保管できます。さらに、エレメントと属性の内容を、*サイド表* と呼ばれる DB2 UDB 表にマップすることもできます。これらのサイド表は、照会パフォーマンスを改善するために索引付きにすることができますが、索引は自動的には作成されません。文書を保管するために使用される列は、XML 列と呼ばれます。この列は、列が XML 列の保管およびアクセス方式に使用されることを指定します。

文書アクセス定義 (DAD) ファイルに、<Xcolumn> タグと </Xcolumn> タグを入力して、使用する保管およびアクセス方式が XML 列であることを指示します。これで、DAD は、保管すべき XML エレメントと属性の内容をサイド表にマップします。

XML Extender を使用して文書を保管する前に、XML 文書の構造を理解して、文書内のエレメントと属性に索引を作成する方法を決める必要があります。文書の索引作成方法を計画するときは、以下について決定する必要があります。

- XML 文書を保管するのに使う XML ユーザー定義タイプ
- アプリケーションで頻繁に検索する XML エレメントと属性。これらの内容をサイド表に保管し、索引を作成することによって、パフォーマンスを改善することができます。
- DTD によって、列内の XML 文書の妥当性検査を行いたいかどうか

XML 列の定義と使用可能化

XML 列は、XML 文書全体をデータベースに保管し、アクセスするために使用されます。この保管方式を使用すると、XML ファイル・タイプを使用して文書を保管すること、サイド表に列の索引を作成すること、および XML 文書を照会または検索することができます。

文書が頻繁には更新されない場合、または、原形の XML 文書を保管したい場合は、XML 文書全体を DB2 表の列に保管するときに XML 列を使用してください。

DB2 UDB 表に対して XML 文書構造をマップして既存の DB2 UDB データから XML 文書を合成したり、あるいは XML 文書を DB2 データに分解したりしたい場合は、XML 列ではなく XML コレクションを使用してください。

手順:

コマンド行から XML 列を定義して使用可能にするには、以下のようにします。

1. 文書アクセス定義 (DAD) ファイルを作成します。
2. XML 文書を保管する表を作成します。
3. XML データ用に列を使用可能にします。
4. サイド表に索引を作成します。

XML 列は、XML ユーザー・データ・タイプとして作成されます。以上の作業が完了すると、列に XML 文書を保管できるようになります。これらの文書は、更新、検索、および抽出を行うことができます。

関連概念:

- 80 ページの『保管およびアクセス方式としての XML 列』
- 81 ページの『XML 列データの索引の使用』
- 57 ページの『XML 文書の自動的妥当性検査』
- 9 ページの『演習: XML 列への XML 文書の保管』

関連タスク:

- 175 ページの『XML 列のための DAD ファイルの作成』
- 59 ページの『XML 表の作成』
- 61 ページの『XML 列の使用可能化』
- 66 ページの『サイド表の索引付け』
- 79 ページの『XML 列のデータの管理』

XML 列データの索引の使用

XML 列を使用する場合の重要な計画上の決定は、XML 列文書のためのサイド表に索引を作成するかどうかという点です。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 Extender 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。

XML Extender 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 UDB *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。 *B-tree 索引付け* に関する詳細は、DB2 UDB 資料を参照してください。

索引を作成するときは、常に次の点を考慮してください。

- XML 文書内のエレメントまたは属性で*複数回出現* のあるものについては、XML 文書の複合構造のために、複数回出現するある 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。
- 1 つの XML 列に複数の索引を作成することができます。
- 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主キーの列名、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ユニーク ID を使います。アプリケーション表の主キーを 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主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 Extender はアプリケーション表を変更して、挿入時に作成されるユニークの 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、ユニークの ID を持つ DXXROOT_ID 列があります。主キーを ROOT ID として使用すると、アプリケーション表内の主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、主キーの値が保管されます。
- XML 列を DB2 UDB Net Search Extender について使用可能にする場合、 Net Search Extender の構造化テキスト機能も使用できます。 Net Search Extender 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 Extender の索引付けと共に使用できます。

XML データの保管

XML Extender 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 Extender 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 Extender は基本タイプを XML タイプとして保管します。

前提条件:

- DAD ファイルを作成または更新したことを確認します。
- 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。
- DB2® 表内のデータを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。

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 Extender 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 7 は、それぞれの方式をいつ使用するかを示しています。

表 7. XML Extender の保管関数

DB2 UDB 基本タイプ	次のものとして DB2 UDB に保管する			
	XMLVARCHAR	**XMLCLOB**	**XMLDBCLOB**	**XMLFILE**
VARCHAR	XMLVARCHAR()	N/A	N/A	XMLFile FromVarchar()
CLOB	N/A	XMLCLOB()	XMLDB CLOB、キャスト関数	XMLFile FromCLOB()
FILE	XMLVarcha rFromFile()	XMLCLOB FromFile()	XMLDB CLOBFrom ファイル、UDF	XMLFILE

XML データを保管するためのデフォルト・キャスト関数

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 Extender 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 8 は、備わっているキャスト関数を示しています。

表 8. XML Extender のデフォルト・キャスト関数

キャスト関数	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

表 8. XML Extender のデフォルト・キャスト関数 (続き)

キャスト関数	戻りタイプ	説明
XMLFILE(VARCHAR)	XMLFILE	ファイル名だけを保管する

たとえば、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))
```

XML データを保管するための保管 UDF

XML Extender 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 UDB に XMLCLOB データ・タイプとしてインポートしたい場合、関数 XMLCLOBFromFile() を使用できます。

表 9 は、XML Extender に備わっている保管関数を示しています。

表 9. XML Extender 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 データ・タイプの値を戻します。オプション: ファイルのエンコード方式を指定します。
XMLCLOBFromFile()	XMLCLOB	XML 文書をサーバー上のファイルから読み取り、XMLCLOB データ・タイプの値を戻します。オプション: ファイルのエンコード方式を指定します。
XMLFileFromVarchar()	XMLFILE	XML 文書をメモリーから VARCHAR データとして読み取り、その文書を外部ファイルに書き込んで、XMLFILE データ・タイプの値 (ファイル名) を戻します。オプション: 外部ファイルのエンコード方式を指定します。
XMLFileFromCLOB()	XMLFILE	XML 文書をメモリーから CLOB データ、または CLOB ロケーターとして読み取り、その文書を外部ファイルに書き込んで、XMLFILE データ・タイプの値 (ファイル名) を戻します。オプション: 外部ファイルのエンコード方式を指定します。

たとえば、XMLCLOBFromFile() 関数を XMLCLOB として使用して、以下のステートメントは、レコードを XML 表に保管します。

```
EXEC SQL INSERT INTO sales_tab(ID, NAME, ORDER)
    VALUES('1234', 'MyName',
      XMLCLOBFromFile('dxx_install/samples/db2xml/xml/getstart.xml'))
```

この例では、dxx_install/samples/db2xml/xml/getstart.xml という名前のファイルから XML 文書を表 SALES_TAB の列 ORDER にインポートします。

XML 文書を検索するための方法

XML Extender 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 Extender 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、以下のセクションを参照してください。

- 『XML 文書全体を取り出す』
- 87 ページの『XML 文書からのエレメント内容と属性の取り出し』

XML Extender 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 10 は、それぞれの方式をいつ使用するかを示しています。

表 10. XML Extender の検索関数

XML タイプ	**次のものとして DB2 UDB から取り出す**			
	VARCHAR	**CLOB**	**DBCLOB**	**FILE**
XMLVARCHAR	**VARCHAR**	N/A	N/A	Content() UDF
XMLCLOB	N/A	**XMLCLOB**	N/A	Content() UDF
XMLFILE	N/A	Content() UDF	N/A	FILE

XML 文書全体を取り出す

手順:

XML 文書全体を取り出すには、次のようにします。

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 UDB 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() UDF を使用する場合は、取り出すデータのデータ・タイプ、およびエクスポートするデータ・タイプを決定します。
4. エレメントまたは属性が抽出される XML 列は XMLVARCHAR、LOCATOR としての XMLCLOB、または XMLFILE データ・タイプとして定義されていなければなりません。

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 データを取り出すためのデフォルト・キャスト関数

DB2 UDB が UDT のために提供するデフォルト・キャスト関数は、XML UDT を SQL 基本タイプに変換し、その後、それを処理します。 XML Extender が提供しているキャスト関数は、SELECT ステートメント内で使用でき、データを取り出すことができます。 表 11 は、備わっているキャスト関数を示しています。

表 11. XML Extender 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

たとえば、以下のステートメントでは、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.XMLVarchar(order) from SALES_TAB
```

XML データを取り出すための Content() UDF の使用

Content() UDF を使用すると、外部ストレージからメモリーに文書内容を取り出したり、または内部ストレージから外部ファイル (DB2 UDB サーバー上の DB2 UDB にとって外部となるファイル) に文書をエクスポートできます。

たとえば、XMLFILE データ・タイプとして保管されている XML 文書があるとします。この文書をメモリーで処理したい場合は、Content() UDF を使用できます。これは入力として XMLFILE データ・タイプを取り、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。以下のことを行うことができます。

- 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() UDF を使用して、XML 文書をメモリー・バッファーまたは CLOB *locator* (データベース・サーバーで単一の LOB 値を表す値をもつホスト変数) に取り出すことができます。

  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

  ```
  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。

  Content() UDF を使用して、DB2 UDB 内に保管されている XML 文書を XMLCLOB データ・タイプとして取り出し、それをデータベース・サーバー・ファイル・システム上のファイルにエクスポートすることができます。 Content() UDF は、そのファイルの名前を VARCHAR データ・タイプとして戻します。

  次の関数構文を使用してください。

XML タイプを外部ファイルに

```
Content(xmlobj XML type, filename varchar(512), targetencoding varchar(100))
```

ここで、

xmlobj XML 文書が取り出される XML 列の名前。*xmlobj* は、タイプ XMLVARCHAR または XMLCLOB のどちらでもかまいません。

filename
XML データを保管する外部ファイルの名前です。

targetencoding
オプション: 出力ファイルのエンコード方式を指定します。

下記の例では、SQL ステートメントが組み込まれている (SQL ステートメントがアプリケーション・プログラム内にコーディングされている) 小さな C プログラム・セグメントで、どのように XML 文書がファイルからメモリーに取り出されるかを示しています。この例では、ORDER 列のデータ・タイプは XMLFILE であるとしています。

```
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;
```

XML 文書からのエレメント内容と属性の取り出し

1 つ以上の XML 文書から、エレメントの内容または属性値の内容を取り出す (抽出する) ことができます (単一文書検索またはコレクション文書検索)。 XML Extender 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、エレメントの内容と属性値の取り出しは、アプリケーションの開発に役立ちます。たとえば、SALES_TAB 表内の ORDER 列に 1000 の XML 文書が保管されていると仮定します。 2500 ドルを超える品目を注文したすべてのカスタマーの名前を取り出すためには、SELECT 文節に抽出 UDF を指定した次の SQL ステートメントを使用します。

```
SELECT extractVarchar(Order, '/Order/Customer/Name') from sales_order_view
        WHERE price > 2500.00
```

この例では、抽出 UDF は <customer> エレメントを ORDER 列から 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。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。

表 12 は、エレメント内容と属性値を抽出するために使用できる UDF を示しています。以下の構文が、表関数またはスカラー関数のいずれかとして使用されます。

構文:

```
extract retrieved_datatype(xmlobj, path)
```

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

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)。

制限: 抽出 UDF は、属性を伴う述部を持つロケーション・パスをサポートできますが、エレメントがあるロケーション・パスに持つことはできません。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 12 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 12. XML Extender の抽出機能

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

表 12. XML Extender の抽出機能 (続き)

スカラー関数	表関数	戻される列名 (表関数)	戻りタイプ
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: 次の例では、属性キー値 1 を持つ値が 1 つ挿入されます。この値は、整数として抽出され、自動的に DECIMAL タイプに変換されます。

```
CREATE TABLE t1(key decimal(3,2));
INSERT into t1 values
SELECT * from table(DB2XML.extractInteger(DB2XML.XMLFile
   ('c:¥dxx_install¥samples¥db2xml¥xml¥getstart.xml'), '/Order/@Key="1"']'));
SELECT * from t1;
```

XML データの更新

XML Extender 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。

手順

XML データを更新するには、以下のように行います。

1. XML 文書が XML 表に保管される必要があります。
2. 取り戻すデータがわかっている必要があります。
3. DB2 UDB 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) を選択する必要があります。
4. 更新する XML 表および列を指定する SQL 照会を指定します。

XML 文書全体の更新

デフォルトのキャスト関数を使用して、または、ストレージ UDF を使用して、XML 文書を更新できます。

デフォルト・キャスト関数による更新

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 Extender 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。

たとえば、次のステートメントは、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ています。

```
UPDATE sales_tab SET=DB2XML.XMLVarchar(:xml_buff)
```

ストレージ UDF による XML 文書の更新

XML Extender UDT ごとに、基本タイプ以外のリソースから DB2 UDB 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。

次の例では、XML オブジェクトが、
dxx_install/samples/db2xml/xml/getstart.xml という名前のファイルから SALES_TAB 表の ORDER 列に更新されます。

```
UPDATE sales_tab
       set order = XMLVarcharFromFile('dxx_install/samples/db2xml
       /xml/getstart.xml) WHERE sales_person = 'MyName'
```

XML 文書の特定のエレメントおよび属性の更新

特定の変更を行うには、文書全体を更新するのではなく、Update UDF を使用してください。この UDF を使用するときは、値を置き換えたいエレメントまたは属性のロケーション・パスを指定します。 XML 文書を編集する必要はありません。XML Extender がユーザーに代わって変更を行います。

構文:

Update(*xmlobj*, *path*, *value*)

構文には、次のコンポーネントが入ります。

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。

value 更新に使用される新しい値です。

たとえば、以下のステートメントは、<Customer> エレメントの値を IBM で置き換えます。

```
UPDATE sales_tab
       set order = Update(order, '/Order/Customer/Name', 'IBM')
   WHERE sales_person = 'Sriram Srinivasan'
```

複数回出現: Update UDF にロケーション・パスを指定するときは、一致するパスをもつすべてのエレメントまたは属性の内容が、提供された値で更新されます。あるロケーション・パスが 1 文書で複数回出現すると、Update UDF は、すでにある値をすべて *value* パラメーターで提供された値で置換します。

XML 文書を検索する方法

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 文節の内容を検索基準として使用します。

XML Extender は、XML 列に保管されている XML 文書を検索するために数種類の方式を提供しています。以下のことを行うことができます。

- 文書構造を検索し、エレメント内容または属性値に基づいて結果を戻す
- XML 列のビューとそのサイド表を検索する
- パフォーマンスを上げるために、直接にサイド表を検索する

- WHERE 文節を指定して、抽出 UDF を使用し検索する
- DB2® Net Search Extender を使用して、構造内容に含まれている列データからテキスト・ストリングを検索する

XML Extender では、索引を使用してサイド表の列を高速に検索できます。これらの列には、XML 文書から抽出された XML エレメント内容または属性値が入っています。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 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、Net Search Extender 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 形式の履歴書を含んでいる、RESUME と呼ばれる列があるとします。今、すべての申込者の中から Java™ スキルを持っている人の名前を検出したいとすると、 DB2 UDB Net Search Extender を使用して、<skill> エレメントに文字ストリング 『JAVA』 を持つすべての履歴書を XML 文書から探し出すことができます。

以下のセクションには検索方法が記載されています。

- 『構造に基づいた XML 文書の検索』
- 93 ページの『DB2 UDB Net Search Extender を使用した XML 文書の構造化テキスト検索』

構造に基づいた XML 文書の検索

XML Extender の検索機能を使用して、文書構造 (文書内のエレメントと属性) に基づいて、列内の XML データを検索することができます。

プロシージャー:

データ検索では、以下を行うことができます。

- 直接にサイド表を照会する
- *結合ビュー* を使用する
- 抽出 UDF を使用する

これらの検索方式について、以降の例で解説します。次のシナリオに基づく例が使用されます。 SALES_TAB 表には 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ます。このビューは、次の CREATE VIEW ステートメントを使用して、これらの表を結合したものです。

```
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
```

例: 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。

手順:

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。

たとえば、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 表の主キーです。

例: 結合ビューからの検索

XML Extender は、ユニークの ID を使用して、アプリケーション表とサイド表を結合するデフォルト・ビューを作成します。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が多ければ多いほど、照会の実行時間は長くなります。

ヒント: root_id または DXXROOT_ID (XML Extender 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。

たとえば、次のステートメントは、SALES_ORDER_VIEW という名前のビューを検索し、注文の行項目のうち、価格が 2500.00 を超える SALES_PERSON 列からの値を戻します。

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

例: 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 Extender 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 データをスキャンすることは負荷が大きいので、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。

以下のステートメントは、抽出 XML Extender 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、サブストリング値として IBM® を含む </Order/Customer/Name> エレメントを抽出します。

例: 複数回出現するあるエレメントまたは属性の検索

複数回出現す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。

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 に複数回出現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、ユニークな値だけ戻されます。

DB2 UDB Net Search Extender を使用した XML 文書の構造化テキスト検索

DB2 UDB Net Search Extender がインストールされている場合は、構造化テキスト検索を行えます。

手順:

DB2 UDB Net Search Extender を使用するには、次のようにします。

1. 構造化テキスト検索を使用するか、または全テキスト検索を行うかを決める。
2. DB2 UDB Net Search Extender に XML 列を使用可能にする。
3. 検索を実行するために照会を作成する。

DB2 UDB Net Search Extender 検索を使用する方法を習得するには、 DB2 Universal Database Extenders: Net Search Extender 管理およびプログラミングのガイド、バージョン 7 を参照してください。

構造化テキスト検索と全テキスト検索の使用

XML 文書構造を検索するとき、XML Extender は汎用データ・タイプに変換されたエレメントを検索しますが、テキストは検索しません。 Net Search Extender を使用して、XML に使用可能になっている列について、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。 DB2 UDB Net Search Extender 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。 Net Search Extender は AIX®、Windows® オペレーティング・システム、 iSeries、および Solaris™ オペレーティング環境で使用できます。

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 において、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。ただし、その文書では、ストリング ″IBM″ が <Comment> サブエレメント内にあったり、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では、指定されたエレメント内のストリングだけが検出されます。この例では、</Order/Customer/Name> サブエレメント内に ″IBM″ がある文書だけが検出

されます。 ″IBM″ が他のエレメントにはあっても、
</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。

DB2 UDB Net Search Extender での XML 列の使用可能化

XML 対応のデータベース で、XML 対応列の内容を検索するために、 DB2 UDB Net Search Extender を使用可能にできます。この例では、データベースの名前は SALES_DB、表の名前は ORDER、そして XML 列の名前は XVARCHAR および XCLOB です。

1. Net Search Extender のインストールについては、DB2 UDB エクステンダー CD の中の `install.txt` ファイルを参照してください。
2. **txstart** コマンドを実行します。
   - 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. Net Search Extender コマンド行ウィンドウをオープンし、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
   ```

4. DB2 UDB Net Search Extender に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
   ```

5. 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て、DB2 UDB Net Search Extender に関して XML 表内の列を使用可能にします。
   - 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 1252
     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 1252
     language us_english indextype precise updateindex update
     ```

6. 索引の状況をチェックします。
   - XVARCHAR 列については、次のようにタイプします。

     ```
     get index status order handle varcharhandle
     ```

   - XCLOB 列については、次のようにタイプします。

     ```
     get index status order handle clobhandle
     ```

7. XML 文書モデルを、`desmodel.ini` と呼ばれる文書モデル初期設定ファイルで定義します。このファイルは、UNIX では `/db2tx/txins000` ディレクトリーに、

また、Windows NT では /instance/db2tx/txins000 ディレクトリーに入っています。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/Shipment/ShipMode = /Order/Part/Shipment/ShipMode
```

DB2 UDB Net Search Extender を使用したテキストの検索

DB2 UDB Net Search Extender を使用してテキストを検索するためには、検索したいエレメントまたは属性を指定する照会を作成してください。 DB2 UDB Net Search Extender は、この照会を使用して、エレメントの内容または属性値を検索します。

たとえば、以下のステートメントを DB2 UDB コマンド・ウィンドウに入力し、DB2 UDB Net Search Extender を使用して XML 文書のテキストを検索します。

```
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
```

この Net Search Extender の Contains() UDF は、XML 文書のテキストを検索します。

この例には、DB2 UDB Net Search Extender を使用して列データを検索するために必要となるすべてのステップは含まれていません。 Net Search Extender 検索の概念および機能を学習するには、「 *DB2 Universal Database Extenders: Net Search Extender 管理およびプログラミング*」を参照してください。

XML 文書の削除

XML 列から XML 文書を含む行を削除するには、SQL DELETE ステートメントを使用します。特定の文書を削除するときは、WHERE 文節を使用できます。

たとえば、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)
```

サイド表の対応する行は、自動的に削除されます。

関連概念:

- 80 ページの『保管およびアクセス方式としての XML 列』

関連タスク:

- 79 ページの『XML 列のデータの管理』

Java データベース (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。

たとえば、JDBC は以下のコードを解決できません。

```
DB2XML.XMLdefault_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.XMLdefault_casting_function(CAST(? AS cast_type(length))
```

例:

次の例では、Sales_Tab タブには 3 つの列があります。 invoice_num 列には Char(6) のデータ・タイプがあり、sales_person 列には Varchar(20) のデータ･タイプがあり、order 列には XMLVarchar のデータ･タイプがあります。

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 4 章 XML コレクションのデータ管理

保管およびアクセス方式としての XML コレクション

リレーショナル・データは、着信した XML 文書から*分解される* か、または発信する XML 文書を*合成する* ために使用されます。分解されたデータは、1 つ以上のデータベース表に保管される、XML 文書のタグのとれた内容です。あるいは、XML 文書は、1 つ以上のデータベース表の既存データから合成されます。データを他のアプリケーションと共有する場合に、 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図 10 に示してあります。

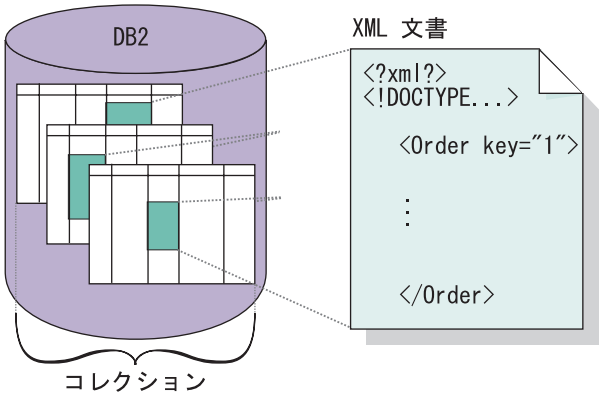


図 10. タグなしのデータとしての文書の DB2 UDB 表内での保管

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。コレクションは、DAD ファイルに関連付けられた列のセットで、特定の XML 文書や XML 文書のセットの中のデータが収められています。コレクションは、使用可能にして名前を定義することができ、その後、ストアード・プロシージャーを実行して、XML 文書を合成または分解するときに、名前でコレクションを参照できます。これは使用可能な XML コレクションと呼ばれます。コレクションに名前が与えられているため、XML 文書を合成および分解するストアード・プロシージャーは容易に実行することができます。

DAD ファイル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が使われます。マッピング体系では、表、列、および XML データを DB2 UDB 表に関連付けるために使用する条件を定義します。 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 UDB 表と条件を定義します。RDB_node マッピングでは、子エレメントを持つ XPath ベースのリレーショナル・データベース・ノード、すなわち、RDB_node が使用されます。

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・プロシージャー名は、XML Extender のスキーマ名 である DB2XML で修飾されます。

XML コレクションのデータ管理

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。

コレクションを構成するリレーショナル表は、新しい表であってもよいし、または、アプリケーション用の XML 文書を構成するために XML Extender で使用するデータが入っている既存の表であってもかまいません。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・プロシージャーを使用して、XML コレクション・データを保管、リトリーブ、更新、検索、削除することができます。

DB2 データからの XML 文書の合成の準備

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。ストアード・プロシージャーを使用するには、文書アクセス定義 (DAD) ファイルを作成してください。 DAD ファイルは、XML 文書と DB2 表構造との間のマッピングを指定します。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。

手順:

XML 文書の合成を開始する前に、以下を行います。

1. 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。
2. 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。
3. DAD ファイルを準備します。

XML Extender は、XML 文書を合成するために 4 つのストアード・プロシージャー、すなわち dxxGenXML()、dxxGenXMLCLOB()、dxxRetrieveXML()、および dxxRetrieveXMLCLOB を提供しています。どのストアード・プロシージャーを使用するかについては、XML 文書の更新の頻度が選択の主な決め手になります。

頻繁には更新されない XML 文書の合成

文書更新の頻度が高くない場合は、dxxGenXML ストアード・プロシージャーを使用して文書を合成してください。このストアード・プロシージャーを使用するには、コレクションを使用可能にする必要はありません。ストアード・プロシージャーは、コレクションではなく DAD ファイルを使用します。

dxxGenXML ストアード・プロシージャーは、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 文書を構成します。このコレクション表は、DAD ファイル内の <Xcollection> エレメントで指定されています。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、アプリケーションで作成する必要があります。また、VARCHAR、CLOB、XMLVARCHAR、または XML データを保存するために使用される XMLCLOB タイプの列が 1 つ必要です。

さらに、DAD ファイル妥当性検査エレメントの値が YES である場合に INTEGER タイプの列 DXX_VALID がまだ結果表になければ、XML Extender が DXX_VALID 列を結果表に加えます。 XML Extender は、有効な XML 文書には値 1 を、無効な文書には値 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と、戻りコードおよびメッセージを戻します。

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。

手順:

dxxGenXML ストアード・プロシージャーを使用して XML コレクションを合成するためには、次のストアード・プロシージャー宣言を使用して、アプリケーション内にストアード・プロシージャー呼び出しを組み込みます。

```
dxxGenXML(CLOB(100K)     DAD,               /* input */
          char(32 resultTabName) resultTabName,     /* input */

          integer        overrideType,      /* input */
          varchar(1024)  override,          /* input */
          integer        maxRows,           /* input */
          integer        numRows,           /* output */
          long           returnCode,        /* output */
          varchar(1024)  returnMsg)         /* output */
```

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_ind;
     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);

     /* read data from a file to a CLOB */
     strcpy(dadfile.name,"dxx_install
```

```
 /samples/dad/getstart_xcollection.dad");
dadfile.name_length = strlen("dxx_install
/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';
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 db2xml.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。

頻繁に更新される XML 文書の合成

文書が頻繁に更新されるものである場合は、dxxRetrieveXML ストアード・プロシージャーを使用して文書を合成してください。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。

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 ファイルではなく、使用可能になっている XML コレクションの名前を必要とします。 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。したがって、XML Extender は DAD ファイルを取り出し、それを使用して、dxxGenXML ストアード・プロシージャーの場合と同様に文書を合成します。

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。

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。

手順:

dxxRetrieveXML ストアード・プロシージャーを使用して XML コレクションを合成するためには、次のストアード・プロシージャー宣言を使用して、アプリケーション内にストアード・プロシージャー呼び出しを組み込みます。

```
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 */
```

例: 以下に dxxRetrieveXML() への呼び出しの例を示します。ここでは、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;
              char    collection;   /* 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   collection_ind;
              short   rtab_ind;
              short   ovtype_ind;
              short   ov_ind;
              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 db2xml.dxxRetrieveXML(: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);
```

関連概念:

- 97 ページの『保管およびアクセス方式としての XML コレクション』
- 109 ページの『XML コレクションのマッピング体系』
- 118 ページの『ロケーション・パス』
- 177 ページの『XML コレクションの DAD ファイル』
- 214 ページの『XML Extender 合成ストアード・プロシージャー - 概要』

関連タスク:

- 70 ページの『RDB_node マッピングを使った XML コレクションの合成』
- 117 ページの『XML コレクション用のスタイルシート』
- 72 ページの『RDB_node マッピングを使った XML コレクションの分解』
- 106 ページの『XML コレクションのデータ更新および削除』
- 108 ページの『XML コレクションの検索』

XML 文書を分解して DB2 UDB データにする

XML 文書の分解とは、XML 文書内のデータを解析してリレーショナル表に保管することです。 XML Extender 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 UDB 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。

分解のために XML コレクションを使用可能にする

ほとんどの場合、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する必要があります。以下のような場合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 Extender によって作成されるので、XML コレクションが使用可能である必要があります。
- 複数回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 Extender が表の複数回出現する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。

enable_collection オプションについては、dxxadm 管理コマンドに関するセクションを参照してください。

データベース内にすでに表が存在するとき 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。

XML 文書を DB2 UDB データに分解する前に以下を行います。

1. 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。
2. RDB_node マッピングを使用して DAD ファイルを準備します。
3. オプション : XML コレクションを使用可能にします。

手順:

XML 文書を分解するには、DB2 UDB XML Extender が提供する 2 つのストアード・プロシージャー dxxShredXML() または dxxInsertXML のいずれかを使用します。

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 データ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。また、XML 文書からのデータを入力 DAD ファイルの <Xcollection> 指定に従って XML コレクションに挿入します。次に、ストアード・プロシージャー dxxShredXML() は XML 文書を分解し、タグのない XML データを DAD ファイルに指定された表に挿入します。 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。

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。

dxxShredXML() による XML 文書の分解

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 */
```

例: 次は dxxShredXML() を呼び出す例です。

```
#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*/
      SQL TYPE is CLOB(1M)  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,
"dxx_install/samples/db2xml/dad/
        getstart_xcollection.dad");
        dadFile.name_length=strlen("dxx_install
       /samples/db2xml/dad/getstart_xcollection.dad");
       dadFile.file_option=SQL_FILE_READ;
       strcpy(xmlFile.name,"dxx_install
       /samples/db2xml/xml/getstart_xcollection.xml");
       xmlFile.name_length=strlen
```

```
("dxx_install/samples/db2xml/xml
/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。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 文書のデータを DAD ファイルに関連付けられている、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> での指定に応じてチェック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。

手順:

XML コレクションを分解するには: 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,   /* input */
            CLOB(1M)       xmlobj,          /* input */
            long           returnCode,      /* output */
            varchar(1024)  returnMsg)       /* output */
```

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 XML collection */
   SQL TYPE is CLOB_FILE xmlFile; /* input XML file */
```

```
    SQL TYPE is CLOB(1M)   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,"dxx_install/samples/db2xml
    /xml/getstart_xcollection.xml");
    xmlobj.name_length=strlen("dxx_install/samples/db2xml
    /xml/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);
```

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 UDB 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 UDB 表に保管されます。どの表にも、各文書から分解した最大 10240 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 10240 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 10240 行までを入れることができます。

複数回出現エレメント (XML 構造内で複数回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。複数回出現するエレメントを使用するときは、1 つの文書から 1 つの表への分解は、最大 1024 行であることを考慮してください。

関連概念:

- 226 ページの『XML Extender 分解ストアード・プロシージャー - 概要』

関連タスク:

- 72 ページの『RDB_node マッピングを使った XML コレクションの分解』
- 206 ページの『XML Extender ストアード・プロシージャーの呼び出し』

関連資料:

- 228 ページの『dxxInsertXML() ストアード・プロシージャー』
- 226 ページの『dxxShredXML() ストアード・プロシージャー』

XML コレクションのデータ更新および削除

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、タグのないデータをデータベース表に保管することです。既存のデータベース表のデータは、外部の XML 文書とは無関係です。更新、削除、および検索操作は、これらの表に対する通常の SQL アクセスで構成されます。

XML Extender には、XML コレクションのビューからデータを操作する機能が備わっています。 SQL UPDATE および DELETE ステートメントを実行して、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。コレクション表に対して SQL 操作を行うと、生成される文書に影響します。

- 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、表の主キーを含む行を削除しないでください。主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 Extender 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。更新によって、複数回出現する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もできます。

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場合、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。コレクション表に対して SQL 操作を行うと、表から生成される文書に影響します。

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。

- 既存の表に主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 マッピングの場合は、<SQL_stmt> エレメントに結合条件を含める。
  - RDB_node マッピングの場合は、ルート・エレメント・ノードの <condition> エレメントに結合条件を含める。

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。

例:

```
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

複数回出現する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場合、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ユニーク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。コレクション表に対して SQL 操作を行うと、生成される文書に影響します。

手順:

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主キー、および低レベルの表の外部キーが含まれます。この方式による文書の削除は、主キーおよび外部キーの制約が SQL に完全に指定されていて、DAD 内に示された表のリレーションシップが、その制約に厳密に合致する場合にだけ有効になります。

例:

以下の DELETE ステートメントは主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 内の主キーであり、この表は DAD に指定されている最上位の表です。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 コレクションの検索

このセクションでは、検索基準を使用した XML 文書の生成という見地から XML コレクションの検索について、また、分解された XML データの検索について説明します。

検索基準を使用した XML 文書の合成

この作業は、条件を使用した合成と同じです。

手順:

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、 *override* 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)  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 データの検索

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適用して、指定した XML 文書の取り出しまたは生成を行ってください。

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する場合は、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指定するマッピング体系 を選択する必要があります。 XML コレクションでは、XML 文書の階層構造がリレーショナル・データベースのリレーショナル構造と一致しなければならないので、ユーザーは、これら 2 つの構造の対比を理解している必要があります。 110 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。

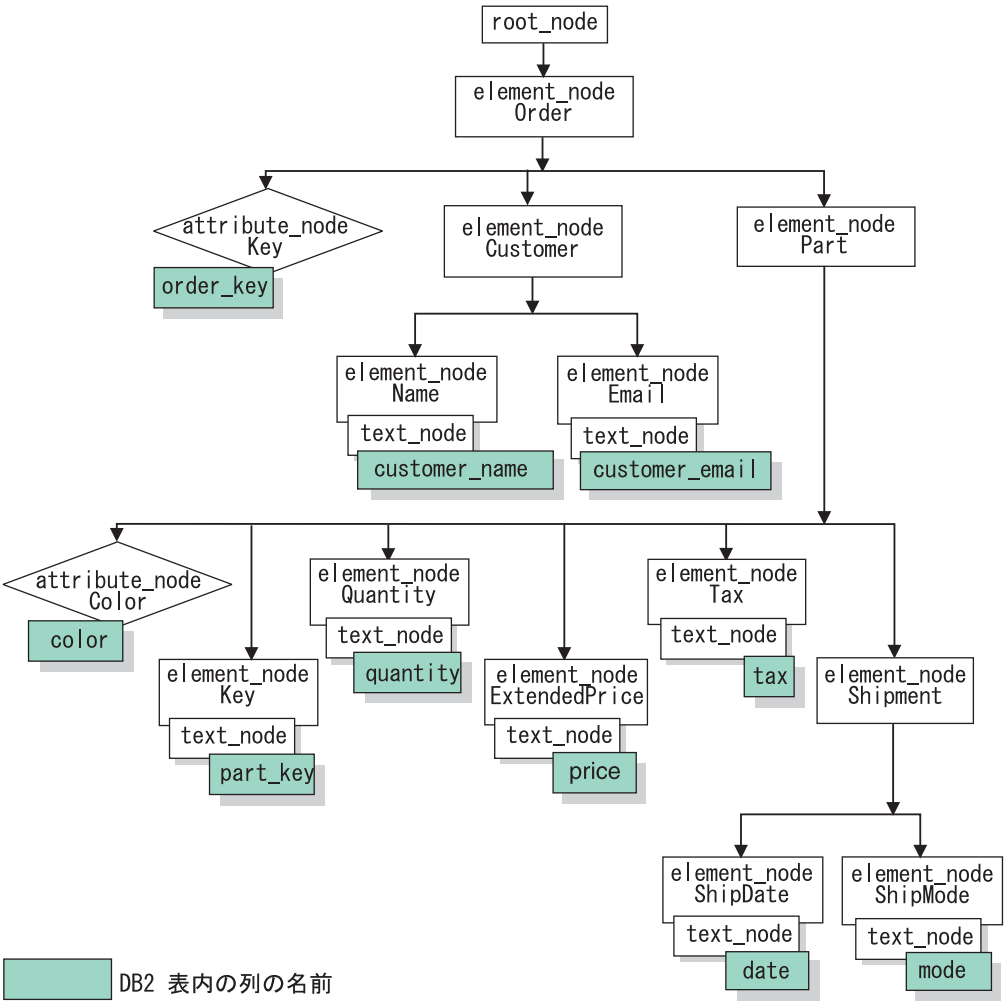


図 11. リレーショナル表の列にマップされた XML 文書構造

XML Extender 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 Extender 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:

DAD ファイルにマッピング体系を指定するには、<Xcollection> と指定します。 XML Extender が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 *(RDB_node)* マッピング の 2 つがあります。

SQL マッピング

この方式では、単一の SQL ステートメントを使用して、リレーショナル・データから XML 文書へ直接にマッピングできます。 SQL マッピングは合成のみに使用されます。 <SQL_stmt> エレメントの内容は、有効な SQL ステートメントでなければなりません。 <SQL_stmt> エレメントは、DAD の XML エレメントまたは属性に後ほどマップされる列を SELECT 文節で指定します。 XML 文書を合成するために定義する場合は、SQL ステートメントの SELECT 文節の列名は、 *attribute_node* の値または *text_node* の内

容を、同じ *name_attribute* を持つ列に関連付け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。すべての表を結び付けるため、これらの表に *主キー* と *外部キー* の関係付けをお勧めします。

RDB_node マッピング

XML エレメントの内容または XML 属性の値のロケーションを定義して、XML Extender がどこに XML データを保管し、どこから取得するかを判別できるようにします。

この方式は、XML Extender が提供する *RDB_node* を使用します。ここには、表に関する 1 つ以上のノード定義、オプションの列、およびオプションの条件が入っています。 DAD の <table> と <column> のエレメントによって、データベースに XML データを保管する方法が定義されます。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 ファイルを作成する必要があります。 112 ページの図 12 は、サンプル DAD ファイルの断片を、XML コレクションのための SQL マッピングとともに、示しています。これによって、3 つのリレーショナル表にあるデータから XML 文書セットが合成されます。

```
<?xml version="1.0"?>
<!DOCTYPE DAD SYSTEM "dxx_install/samples/db2xml/dtd/dad.dtd">
<DAD>
  <dtdid>dxx_install/samples/dad/getstart.dtd</dtdid>
  <validation>YES</validation>
  <Xcollection>
    <SQL_stmt>
       SELECT o.order_key, customer, p.part_key, quantity, price, tax, date,
              ship_id, mode, comment
         FROM order_tab o, part_tab p,
               table(select substr(char(timestamp
                (generate_unique())),16)
                as ship_id, date, mode, from ship_tab)
         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
    "dxx_install/samples/db2xml/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 Extender 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。

関連概念:

- 177 ページの『XML コレクションの DAD ファイル』
- 113 ページの『SQL マッピングを使用するための要件』
- 114 ページの『RDB_Node マッピングのための要件』

関連タスク:

- 66 ページの『SQL マッピングを使った XML 文書の合成』
- 70 ページの『RDB_node マッピングを使った XML コレクションの合成』
- 72 ページの『RDB_node マッピングを使った XML コレクションの分解』

SQL マッピングを使用するための要件

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> エレメント内に指定する必要があります。 <SQL_stmt> には、複数のリレーショナル表を照会の *述部* で結合する SQL ステートメントが 1 つ含まれている必要があります。さらに、以下の文節が必要です。

- **SELECT 文節**
  - 列の名前がユニーク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にまとめた上で、それらの表を、XML 文書の階層構造にマップするときのツリー・レベルに従って並べます。各列グループの最初の列は、オブジェクト ID 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` 列と `customer` 列は、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` の列は、最低のリレーショナル・レベルにあります。
  - それぞれのレベルは、単一列候補キーを使用して開始します。表にそのようなキーがない場合は、表の式および generate_unique() 関数を使用して、そのようなキーを照会で生成する必要があります。上記の例で、`o.order_key` は ORDER_TAB の主キー、そして `part_key` は PART_TAB の主キーです。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 表には主キーがないので、`ship_id` が主キーとして生成されます。 `ship_id` 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 文節を使用して主キー列を生成します。
- **FROM 文節**
  - 表の式および generate_unique() 関数を使用して、単一の主キーが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, part key from ship_tab) as s
    ```

    この例では、単一列候補キーが generate_unique() 関数で生成され、それに別名 `ship_id` が与えられています。
  - 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 表の列に対して `o`、PART_TAB 表の列に対して `p`、そして SHIP_TAB 表の列に対して `s` を使用できます。
- **WHERE 文節**

– 主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。列名の後ろには、ASC または DESC など、他のものを置くことはできません。

– 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。

– すべてのオブジェクト ID を、SELECT 文節に並べたのを同じ相対順序にリストします。

– ID は、表式および関数 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 文節で指定される最初の列は、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 文節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> エレメントは、述部の式が表内の列を使用する限り 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。

関連資料:

• 317 ページの『付録 A. サンプル』

RDB_Node マッピングのための要件

マッピング方式として RDB_Node を使用するときは、<SQL_stmt> エレメントは、DAD ファイルの <Xcollection> エレメント内で使用できません。その代わりに、先頭 element_node の子、および各 attribute_node と text_node の子として、RDB_node エレメントを使用します。

• **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 コレクション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、先頭のエレメント・ノードである <Order> の <RDB_node> に 3 つの表を指定しています。

```
<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。

– 条件エレメントでは、1 つの列名を制限なく何回でも参照できます。

– コレクションを使用可能にする場合は、各表に対して主キーを指定する必要があります。主キーは、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。主キーは、RDB_node の表エレメントに *attribute key* を追加することによって指定されます。複合キーを提供する場合、その *key* 属性は、複数のキー列名をスペースで区切って指定します。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、合成の場合に同じ DAD が使用されると無視されます。

– orderBy 属性を使用して、複数回出現する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。 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、複数回出現する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、 RDB_node 内のルート・エレメントを含む表の orderBy 属性を指定します。

<table> タグ内の表名と列名は、スペルを完全に書いてください。

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

XML Extender は、データベースのどこからデータを検索するべきかを知る必要があります。また、XML Extender は、XML 文書からの内容をデータベースのどこに入れるかについても知る必要があります。それぞれの属性ノードとテキスト・ノードに RDB_node を指定してください。表および列の名前は指定しなければなりません。条件の値は任意指定です。

1. 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>
```

2. 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。

3. 条件を使用して XML 文書を生成したい場合、照会条件を指定します。この条件に適合するデータのみが、生成される XML 文書に入ります。条件は有効な WHERE 文節でなければなりません。上記の例では、条件は price > 2500.00 と指定されています。したがって、価格が 2500 を超える行だけが XML 文書に組み込まれます。
4. 文書を分解する場合、または DAD ファイルで指定されている XML コレクションを使用可能にする場合、属性ノードおよびテキスト・ノードごとに列タイプを指定する必要があります。各属性ノードとテキスト・ノードに対して列タイプを指定することによって、XML コレクションの使用可能化で新規の表が作成されるときに、どの列のデータ・タイプも正しいものであることを確認できます。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します。

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

文書の分解時に指定された列タイプは、合成の場合には無視されます。

- 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。エレメント・ノードが正しくネストされていて、XML Extender が文書の分解または合成時に、エレメント間の関係を理解できるようになっている必要があります。たとえば、以下の DAD ファイルでは、Part の内側で Shipment をネストしていません。

```
<element_node name="Part">
    ...
    <element_node name="ExtendedPrice">
          ...
    </element_node>
          ...
</element_node> <!-- end of element Part -->

<element_node name="Shipment" multi_occurrence="YES">
          <element_node name="ShipDate">
              ...
          </element_node>
          <element_node name="ShipMode">
              ...
          </element_node>

 </element_node> <!-- end of element Shipment-->
```

この DAD ファイルは、Part と Shipment のエレメントを兄弟とする XML 文書を作成します。

```
<Part color="black ">
  <key>68</key>
  <Quantity>36</Quantity>
  <ExtendedPrice>34850.16</ExtendedPrice>
  <Tax>6.000000e-2</Tax>
</Part>

<Shipment>
    <ShipDate>1998-08-19</ShipDate>
    <ShipMode>BOAT  </ShipMode>
</Shipment>
```

以下のコードでは、Shipment エレメントは DAD ファイル内で Part の内側でネストされています。

```
<element_node name="Part">
    ...
    <element_node name="ExtendedPrice">
```

```
        ...
    </element_node>
    ...
    <element_node name="Shipment" multi_occurrence="YES">
        <element_node name="ShipDate">
            ...
        </element_node>
        <element_node name="ShipMode">
            ...
        </element_node>

    </element_node> <!-- end of element Shipment-->
</element_node> <!-- end of element Part -->
```

Part の内側で Shipment をネストすることにより、Shipment エレメントを Part エレメントの子エレメントとする XML ファイルが作成されます。

```
<Part color="black ">
  <key>68</key>
  <Quantity>36</Quantity>
  <ExtendedPrice>34850.16</ExtendedPrice>
  <Tax>6.000000e-2</Tax>
  <Shipment>
    <ShipDate>1998-08-19</ShipDate>
    <ShipMode>BOAT  </ShipMode>
  </Shipment>
</Part>
```

ルート・ノード条件の述部には、順序の制約事項はありません。

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。

同じ表にマップする、element_nodes および attribute_nodes を使用する DAD のサブツリーの場合、以下が当てはまります。

- 属性ノードは、同じ表にマップするエレメント・ノードの最下位の共通祖先における最初の子である必要はありません。
- 属性ノードは結合条件に関与しない限り、サブツリーのどこにあってもかまいません。

制約事項: RDB_node マッピング DAD に許可されている表の数の限度は 30 です。 1 つの表当たりに許可されている列数は 500 です。条件ステートメントの結合述部で指定可能な各表または列の回数には制限がありません。

XML コレクション用のスタイルシート

文書を構成するときには、XML Extender 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>
```

ロケーション・パス

ロケーション・パス は、XML 文書の構造内でのエレメントまたは属性のロケーションを定義します。 XML Extender は、このロケーション・パスを次の目的に使用します。

- dxxRetrieveXML などの抽出 UDF を使用する際に、エレメントまたは属性の位置を突き止めるため
- 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® 列とのマッピングを指定するため
- Net Search Extender を使用する構造化テキスト検索のため
- ストアード・プロシージャーの XML コレクション DAD ファイルの値をオーバーライドするため

図 13 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。

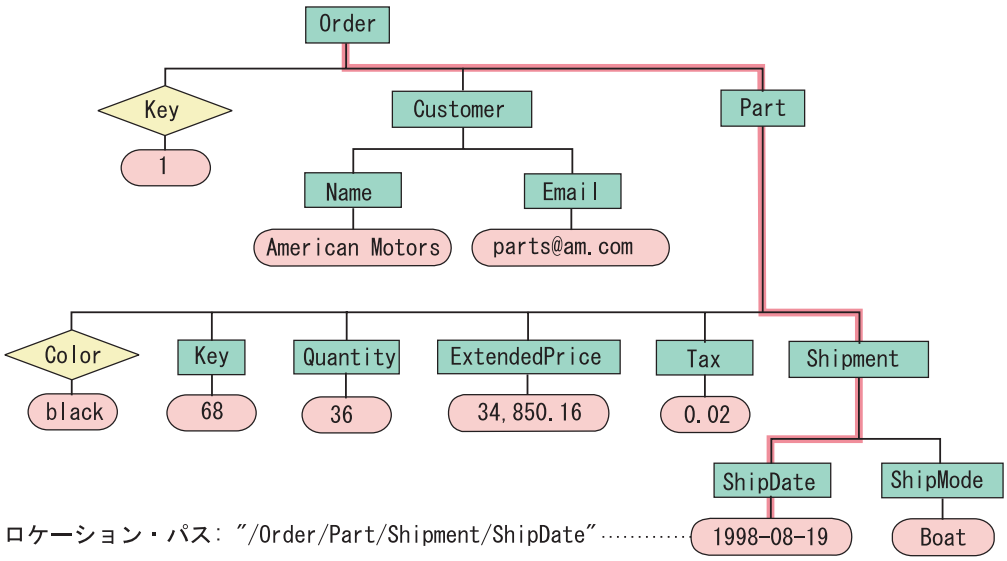


図 13. 文書を構造化 XML 文書として DB2 UDB 表の列に保管

関連資料:

- 119 ページの『ロケーション・パスの構文』

ロケーション・パスの構文

XML Extender は、XML 文書構造をナビゲートするためにロケーション・パスを使用します。以下のリストは、XML Extender 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。これは、文書内のすべての他のエレメントを含んでいるエレメントです。
2. */tag1*
   ルート・エレメン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孫であり、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、ルートの下にあるエレメント *tag1* 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。

単純ロケーション・パス

単純ロケーション・パス は、XML 列 DAD ファイルで使用されるロケーション・パスのタイプです。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。属性の値は、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 13 は、単純ロケーション・パスの構文を要約しています。

表 13. 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

ロケーション・パスの使用

ロケーション・パスの構文は、エレメントまたは属性のロケーションをアクセスするコンテキストによって変わります。 XML Extender 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される構文規則を制限します。 表 14 は、どのコンテキストで構文オプションが使用されるかを説明しています。

表 14. ロケーション・パスの使用に関する XML Extender 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 マッピングのパス属性の値	*/tag1/tag2/..../tagn* および */tag1/tag2/@attr1* (119 ページの表 13 で説明されている単純ロケーション・パス)
UDF の抽出	すべてのロケーション・パス [1]
Update UDF	すべてのロケーション・パス [1]
Net Search Extender による UDF の検索	*/tag1/tag2/..../tagn* — 例外: ルート・マークはスラッシュなしで使用。例: `tag1/tag2/..../tagn`

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

関連概念:

- 118 ページの『ロケーション・パス』

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と、DAD ファイルを解析してその XML 文書に関連した表と列を識別し、XML_USAGE 表に制御情報を記録します。次の場合、XML コレクションの使用可能化はオプションです。

- XML 文書の分解とそのデータの新しい DB2 UDB 表への保管
- 複数の DB2 UDB 表にある既存のデータから XML 文書を合成

合成と分解に同じ DAD ファイルを使用する場合は、そのコレクションを合成と分解の両方に対して使用可能にすることができます。

XML コレクションを使用可能にするには、XML Extender 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 Extender 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:

管理ウィザードを使用して XML コレクションを使用可能にするには、次のようにします。

1. 管理ウィザードをセットアップして開始します。
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)」ウィンドウが開きます。

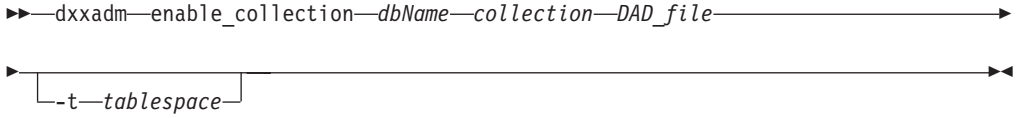
3. **「コレクションを使用可能にする (Enable a Collection)」**をクリックし、さらに**「次へ」**をクリックします。「コレクションを使用可能にする (Enable a Collection)」ウィンドウが開きます。
4. **「コレクション名 (Collection name)」**フィールドで、使用可能にするコレクションの名前を選択します。
5. DAD ファイル名を**「DAD ファイル名 (DAD file name)」**フィールドに指定します。
6. オプション: 既に作成された表スペースの名前を**「表スペース (Table space)」**フィールドに入力します。

   表スペースには、分解のために生成された新しい DB2 UDB 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になると、 `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にできなかった場合は、エラー・メッセージが表示されます。コレクションが正常に使用可能になるまで、上記のステップを繰り返してください。

dxxadm コマンドの使用による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には、**dxxadm** コマンドを DB2 UDB コマンド行から入力します。

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
: 分解のために生成された新しい DB2 UDB 表を含む既存の表スペース。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: 次の例は、コマンド行を使用して、データベース SALES_DB にある 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。

```
dxxadm enable_collection SALES_DB sales_ord getstart_collection.dad
```

XML コレクションを使用可能にした後、XML Extender のストアード・プロシージャーを使用して XML 文書を合成あるいは分解することができます。

関連概念:

- 97 ページの『保管およびアクセス方式としての XML コレクション』

関連タスク:

- 122 ページの『XML コレクションを使用不可にする』
- 98 ページの『XML コレクションのデータ管理』

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ドロップしません。 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ドロップするときに、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 Extender 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 Extender ストアード・プロシージャー dxxDisableCollection() を使用します。

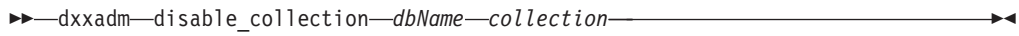
手順:

管理ウィザードを使用して XML コレクションを使用不可にするには、次のようにします。

1. 管理ウィザードを開始します。
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 Extender 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)」ウィンドウが開きます。
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)」ウィンドウが開きます。
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。

コマンド行から XML コレクションを使用不可にするには、**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。

例:

```
dxxadm disable_collection SALES_DB sales_ord
```

関連概念:

- 97 ページの『保管およびアクセス方式としての XML コレクション』
- 208 ページの『XML Extender 管理ストアード・プロシージャー - 概要』

関連タスク:

- 98 ページの『XML コレクションのデータ管理』

第 5 章 XML スキーマ

XML スキーマは、XML 文書の内容の仕様を定義するために DTD の代わりに使用することができます。 XML スキーマは、XML 文書のエレメントと属性名を定義するために XML フォーマットと XML 構文を使用し、エレメントと属性が含むことのできる内容のタイプを定義します。

DTD ではなく XML スキーマを使用する利点

DTD は、XML スキーマよりもコード化および検証が簡単です。しかし、XML スキーマを使用することには、以下に列挙するような利点があります。

- XML スキーマは、WebSphere® Studio Application Developer の XSD エディター、XML Spy、または XML Authority などのツールで処理できる、有効な XML 文書である。
- XML スキーマは、DTD よりも強力である。 DTD で定義できるすべてのものをスキーマで定義できますが、その逆は真ではありません。
- XML スキーマは、ごく普通のプログラム言語で使用されるものと同様のデータ・タイプのセットをサポートし、追加のデータ・タイプを作成する機能も備えています。 文書の内容を適切なタイプに制限することができます。たとえば、DB2® で使用されているフィールドのプロパティーを複製できます。
- XML スキーマは、文字データに制約を設定するための正規表現をサポートする。これは、DTD を使用した場合、可能ではありません。
- XML スキーマは、XML ネーム・スペースに対し、よりよいサポートを提供しており、それによって、複数のネーム・スペースを使用する文書の妥当性検査を行え、また、異なるネーム・スペースにすでに定義されているスキーマから構造を再使用することができる。
- XML スキーマは、組み込みエレメントおよびインポート・エレメントとともに、モジュール性の面でも再使用の面でも、よりよいサポートを提供する。
- XML スキーマは、エレメント、属性、およびデータ・タイプの定義について継承をサポートする。

関連タスク:

- 126 ページの『スキーマ内でのデータ・タイプ、エレメント、および属性』

関連資料:

- 128 ページの『XML スキーマの例』

XML スキーマの complexType エレメント

XML スキーマ・エレメントの complexType は、サブエレメントを含むことのできるエレメント・タイプを定義するために使用します。たとえば、以下のタグは、XML 文書での住所を示しています。

```
<billTo country="US">
    <name>Dan Jones</name>
    <street>My Street</street>
    <city>My Town</city>
    <state>CA</state>
    <zip>99999</zip>
</billTo>
```

このエレメントの構造は、XML スキーマに次のように定義できます。

```
1 <xsd:element name="billTo" type="USAddress"/>
2 < xsd:complexType name="USAddress">
 3  <xsd:sequence>
 4   < xsd:element name="name" type="xsd:string"/>
 5   < xsd:element name="street" type="xsd:string"/>
 6   < xsd:element name="city" type="xsd:string"/>
 7   < xsd:element name="state" type="xsd:string"/>
 8   < xsd:element name="zip" type="xsd:decimal"/>
 9  </xsd:sequence>
 10  < xsd:attribute name="country"
                          type="xsd:NMTOKEN" use="fixed"
                          value="US"/>
 12</xsd:complexType>
```

上記の例では、接頭部 xsd は、XML スキーマのネームスペースにバインドされていると想定しています。 2 行目から 12 行目で、complexType の USAddress が、一連のエレメント 5 つと属性 1 つとして定義されています。エレメントの順序は、sequence タグに現れる順序で決められます。

内側のエレメントは、データ・タイプが xsd:string または xsd:decimal です。これらは、両方とも単純データ・タイプと事前定義されています。

別の方法としては、<sequence> タグの代わりに、<all> タグまたは <choice> タグを使用することもできます。 all タグを指定すると、すべてのサブエレメントを表示する必要がありますが、特に決まった順序にする必要はありません。 choice タグを使用する場合は、サブエレメントの 1 つだけが、XML 文書の中に出現する必要があります。

また、その他のエレメントを定義するために、ユーザー定義のデータ・タイプを使用することもできます。

スキーマ内でのデータ・タイプ、エレメント、および属性

XML スキーマ内の単純なデータ・タイプ

XML スキーマには、単純な組み込みデータ・タイプのセットが備わっています。制約を加えることにより、それらから他のデータ・タイプを派生させることもできます。

例 1 では、基本タイプ xsd:positiveInteger の範囲は 0 から 100 までです。

例 1

```
< xsd:element name="quantity">
   < xsd:simpleType>
    < xsd:restriction base="xsd:positiveInteger">
```

```
    < xsd:maxExclusive value="100"/>
   </xsd:restriction>
  </xsd:simpleType>
 </xsd:element>
```

例 2 では、基本タイプ xsd:string は、通常の式で制限されています。

例 2

```
<xsd:simpleType name="SKU">
   < xsd:restriction base="xsd:string">
    < xsd:pattern value="¥d{3}-[A-Z]{2}"/>
   </xsd:restriction>
  </xsd:simpleType>
```

例 3 では、string 組み込みタイプに基づく列挙タイプを示しています。

例 3

```
<xsd:simpleType name="SchoolClass">
   < xsd:restriction base="xsd:string">
    < xsd:enumeration value="WI"/>
    < xsd:enumeration value="MI"/>
    < xsd:enumeration value="II"/>
    < xsd:enumeration value="DI"/>
    < xsd:enumeration value="AI"/>
   </xsd:restriction>
  </xsd:simpleType>
```

XML スキーマ内のエレメント

XML スキーマでエレメントを宣言するためには、その名前とタイプを element エレメントの属性で指示する必要があります。たとえば次のようにします。

```
<xsd:element name="street" type="xsd:string"/>
```

さらに、属性の minOccurs と maxOccurs を使用して、XML 文書内にそのエレメントが出現する必要のある最大回数と最小回数を決めることができます。 minOccurs と maxOccurs のデフォルト値は、1 です。

XML スキーマ内の属性

属性の宣言は、エレメント定義の終わりに現れます。たとえば次のようにします。

```
<xsd:complexType name="PurchaseOrderType">
< xsd:sequence>
  < xsd:element name="billTo" type="USAddress"/>
< xsd:sequence>
  < xsd:attribute name="orderDate" type="xsd:date"/>
 </xsd:complexType>
```

関連概念:

- 125 ページの『DTD ではなく XML スキーマを使用する利点』

関連タスク:

- 171 ページの『検証機能』

関連資料:

- 128 ページの『XML スキーマの例』
- 125 ページの『XML スキーマの complexType エレメント』

XML スキーマの例

UML ツールを使用して、最初に XML 文書のデータ構造を設計して XML スキーマを作成するというのは、うまい戦略です。構造を設計した後、その構造をスキーマ文書にマップすることができます。以下は、XML スキーマの例です。

```
 1 <?xml version="1.0" encoding="UTF-8"?>
 2 <xs:schema xmlns:xs='http://www.w3.org/2001/XMLSchema'>
 3
 4   <xs:element name="personnel">
 5    <xs:complexType>
 6     <xs:sequence>
 7       <xs:element ref="person" minOccurs='1' maxOccurs='unbounded'/>
 8    </xs:sequence>
 9   </xs:complexType>
10  </xs:element>
11
12   <xs:element name="person">
13    <xs:complexType>
14     <xs:sequence>
15       <xs:element ref="name"/>
16       <xs:element ref="email" minOccurs='0' maxOccurs='4'/>
17    </xs:sequence>
18     <xs:attribute name="id"  type="xs:ID" use='required'/>
19   </xs:complexType>
20  </xs:element>
21
22   <xs:element name="name">
23    <xs:complexType>
24     <xs:sequence>
25      <xs:element ref="family"/>
26      <xs:element ref="given"/>
27    </xs:sequence>
28   </xs:complexType>
29  </xs:element>
30
31   <xs:element name="family" type='xs:string'/>
32   <xs:element name="given" type='xs:string'/>
33   <xs:element name="email" type='xs:string'/>
34 </xs:schema>
```

最初の 2 行は、この XML スキーマが XML 1.0 互換で、ユニコード 8 でデコードされていると宣言し、基本 XML スキーマのデータ・タイプと構造にアクセス可能にする XML スキーマ標準ネーム・スペースを使用すると指定しています。

4 行目から 10 行目までは、1 人から n 人までの一連の人間から構成される complexType として担当者を定義しています。この complexType は、12 行目から 20 行目に定義されます。これは、complexType エレメント名とエレメント email で構成されます。 email エレメントはオプションであり (minOcccurs = '0')、最高 4 回 (maxOccurs = '4') まであっても問題ありません。エレメントの出現数が多ければ多いほど、スキーマの妥当性検査に時間がかかることになります。それに対して、DTD の場合は、エレメントの出現回数を 0、1 または無制限とすることができます。

22 行目から 29 行目は、担当者のタイプを示すために使用されています。名前 (name) のタイプは、姓 (family) と名 (given) のエレメントから成ります。

31 行名から 33 行目では、宣言されているタイプ・ストリングを含むエレメント *family*、*given* および *e-mail* を 1 つずつ定義しています。

スキーマ使用の XML 文書のインスタンス

次の例は *personalnr.xsd* スキーマのインスタンスである XML 文書です。

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <personnel xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3     xsi:noNamespaceSchemaLocation='personsnr.xsd'>
4
5   <person id="Big.Boss" >
6      <name><family>Boss</family><given>Big</given></name>
7     <email>chief@foo.com</email>
8   </person>
9
10   <person id="one.worker">
11   <name><family>Worker</family><given>One</given></name>
12      <email>one@foo.com</email>
13   </person>
14
15    <person id="two.worker">
16   <name><family>Worker</family><given>Two</given></name>
17      <email>two@foo.com</email>
18   </person>
19 </personnel>
```

DTD 使用の XML 文書

この例は、この XML スキーマが DTD としてどのように認識されるかを示しています。

```
1   <?xml version="1.0" encoding="UTF-8"?>
2   <!ELEMENT email (#PCDATA)>
3   <!ELEMENT family (#PCDATA)>
4   <!ELEMENT given (#PCDATA)>
5   <!ELEMENT name (family, given)>
6   <!ELEMENT person (name, email*)>
7
8   <!ATTLIST person
9 id ID #REQUIRED>
10 <!ELEMENT personnel (person+)>
```

DTD を使用する場合は、email の最大出現数は 1 か無制限かのどちらかに設定できます。

この DTD を使用すると、この XML 文書インスタンスは、最初の例に示したものと同じになりますが、ただし、2 行目は以下のように変わります。

```
<!DOCTYPE personnel SYSTEM "personsnr.dtd">
```

関連概念:

- 125 ページの『DTD ではなく XML スキーマを使用する利点』

関連タスク:

- 126 ページの『スキーマ内でのデータ・タイプ、エレメント、および属性』
- 171 ページの『検証機能』

関連資料:

- 125 ページの『XML スキーマの complexType エレメント』

第 4 部 参照

ここには、XML Extender 管理コマンド、ユーザー定義タイプ (UDT)、ユーザー定義関数 (UDF)、およびストアード・プロシージャーの構文が記載されています。問題判別のために、メッセージも記載されています。

第 6 章 dxxadm 管理コマンド

dxxadm コマンドの概要

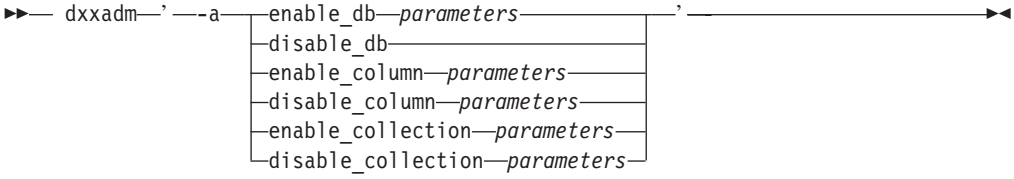
XML Extender には、以下の管理タスクを完了するための管理コマンド **dxxadm** が備わっています。

- enable_column
- enable_collection
- enable_db
- disable_column
- disable_collection
- disable_db

関連概念:

- 39 ページの『XML Extender の管理ツール』
- 41 ページの『XML Extender 管理計画の概要』

dxxadm 管理コマンドの構文



パラメーター:

表 15. dxxadm パラメーター

パラメーター	説明
enable_db	データベースのための XML Extender 機能を使用可能にします。
disable_db	データベースのための XML Extender 機能を使用不可にします。
enable_column	XML 列を使用可能にし、XML 文書が列に保管されるようにします。
disable_column	XML に使用できる列を使用不可にします。
enable_collection	指定した DAD に従って、XML コレクションを使用可能にします。
enable_collection	XML 対応のコレクションを使用不可にします。

管理コマンドのオプション

システム・プログラマーは、以下の **dxxadm** コマンド・オプション を使用することができます。

- enable_column
- enable_collection
- enable_db
- disable_column
- disable_collection
- disable_db

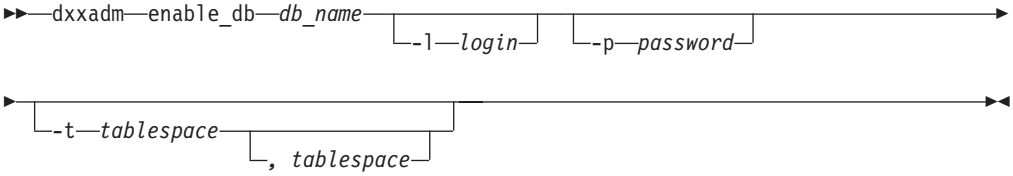
dxxadm コマンドの enable_db オプション

目的:

データベースのための XML Extender 機能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 Extender は以下のオブジェクトを作成します。

- XML Extender ユーザー定義タイプ (UDT)
- XML Extender ユーザー定義関数 (UDF)
- XML Extender DTD リポジトリー表 (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。
- XML Extender 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。

構文:



パラメーター:

表 16. 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*	データベースへの接続に使用される、オプションのユーザー 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*	データベースへの接続に使用される、オプションのパスワード。指定しない場合、現行のパスワードが使用されます。
-t *tablespace*	db2xml.XML_USAGE 表および db2xml.DTD_REF 表を保持する、既存の表スペースのオプショナル名。2 番目の表スペースも指定できます。

パーティション化された DB2 UDB Enterprise Server Edition を使用しており、データベースを使用可能化する間に表スペースを指定する場合には、表スペースを作成するときにノード・グループを指定しておく必要があります。たとえば次のようにします。

```
db2 "create database partition group mygroup on node (0,1)"
db2 "create regular tablespace mytb in database partition group mygroup
   managed by system using ('mytb')"
```

上記の例では、次に、データベースを使用可能にするときに mytb 表スペースを指定します。

データベースを使用可能にする際に表スペース・オプションが指定されない場合、XML Extender は DXXDTDRF および DXXXMLUS 表スペースが存在するかチェックします。その表スペースが存在する場合、db2xml.dtd_ref 表が DXXDTDRF 表スペースに作成され、 db2xml.xml_usage 表が DXXXMLUS 表スペースに作成されます。 DXXDTDRF または DXXXMLUS 表スペースのいずれかが存在しない場合、それぞれの表 (db2xml.dtd_ref または db2xml.xml_usage) は最適な表スペースに作成されます。

データベースを使用可能にするときに、DXXDTDRF 表スペースが 1 つだけ提供される場合、指定された表スペースに両方の表が作成されます。データベースを使用可能にするときに 2 つの表スペースが提供された場合は、 db2xml.dtd_ref 表が最初にリストされた表スペースに作成され、 db2xml.xml_usage 表が 2 番目にリストされた表スペースに作成されます。

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
```

関連資料:

- 133 ページの『dxxadm コマンドの概要』

dxxadm コマンドの disable_db オプション

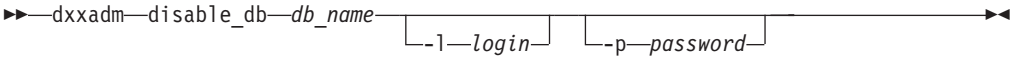
目的:

データベースのために XML Extender 機能を使用不可にします。このアクションは、「データベースを使用不可にする」と言われます。データベースが使用不可になると、XML Extender 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 Extender 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ドロップします。

- XML Extender ユーザー定義タイプ (UDT)
- XML Extender ユーザー定義関数 (UDF)
- XML Extender DTD リポジトリー表 (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。
- XML Extender 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。

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 Extender 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。 XMLCLOB などの XML Extender ユーザー定義タイプで列を定義しているすべての表をドロップする必要もあります。

構文:



パラメーター:

表 17. 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*	データベースへの接続に使用されるユーザー 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*	データベースへの接続に使用されるパスワード。 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
```

関連概念:

- 208 ページの『XML Extender 管理ストアード・プロシージャー - 概要』
- 291 ページの『第 13 章 XML Extender の管理サポート表』

関連資料:

- ix ページの『構文図の読み方』

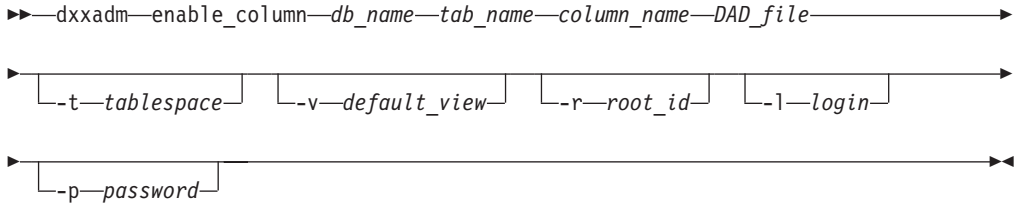
dxxadm コマンドの enable_column オプション

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 Extender UDT を含めることができるようにします。 XML Extender は、列を使用可能にする時に以下のタスクを行います。

- XML 表に主キーがあるかどうかを判別します。ない場合は、XML Extender が XML 表を変更して 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 表内の各行のユニークな 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 Extender で指定された DXXROOT_ID です。
- オプションで、XML 表およびそのサイド表のデフォルト・ビューを作成します。オプションで、ユーザーが名前を指定できます。

構文:



パラメーター:

表 18. 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。
-t *tablespace*	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、デフォルト・ビューの名前。
-r *root_id*	表表の root_id として使用される、XML 列表内の主キーの名前。 root_id は任意指定です。
-l *login*	データベースへの接続に使用される、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*	データベースへの接続に使用されるパスワード。 指定しない場合、現行のパスワードが使用されます。

パーティション化された DB2 UDB Enterprise Server Edition を使用しており、列を使用可能化する間に表スペースを指定する場合には、表スペースを作成するときにノード・グループを指定しておく必要があります。たとえば次のようにします。

```
db2 "create database partition group mygroup on node (0,1)"
db2 "create regular tablespace mytb in database partition group mygroup
   managed by system using ('mytb')"
```

上記の例では、次に列を使用可能化するときに mytb 表スペースを指定します。

```
dxxadm enable_column mydatabase mytable mycolumn "dad/mydad.dad" -t mytb
```

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 getstart.dad
   -v sales_order_view -r INVOICE_NUMBER
```

関連サンプル:

- 『dxx_xml -- s-getstart_enableCol_NT-cmd.htm』
- 『dxx_xml -- s-getstart_enableCol-cmd.htm』

dxxadm コマンドの disable_column オプション

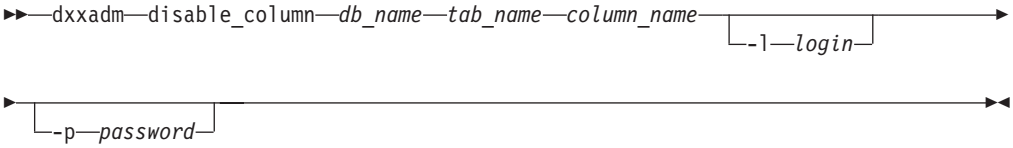
目的:

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、以下のアクションが実行されます。

- 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ドロップされます。
- この列に関連したすべてのサイド表がドロップされます。

重要: XML 表をドロップする前に、XML 列を使用不可にしなければなりません。XML 表がドロップされてもその XML 列が使用不可にされていない場合、XML Extender 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。

構文:



パラメーター:

表 19. 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*	データベースへの接続に使用されるユーザー 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*	データベースへの接続に使用されるパスワード。 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
```

関連資料:

- 139 ページの『dxxadm コマンドの enable_collection オプション』

関連サンプル:

- 『dxx_xml -- s-getstart_clean_NT-cmd.htm』
- 『dxx_xml -- s-getstart_clean-cmd.htm』

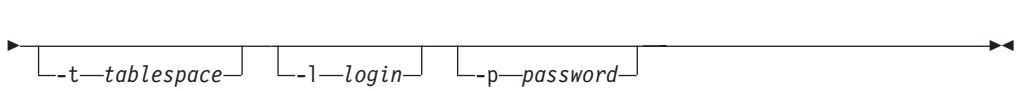
dxxadm コマンドの enable_collection オプション

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。 パーティション化された Enterprise Server Edition 環境で XML Extender を実行する場合、 DAD ファイルで指定されているすべての表に、パーティション・キーとして修飾された少なくとも 1 つの列が含まれていることを確認してください。 コレクションを使用可能にするとき、XML Extender 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。

構文:



パラメーター:

表 20. 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*	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*	データベースへの接続に使用されるユーザー 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*	データベースへの接続に使用されるパスワード。 指定しない場合、現行のパスワードが使用されます。

パーティション化された DB2 UDB Enterprise Server Edition を使用しており、コレクションを使用可能化する間に表スペースを指定する場合には、表スペースを作成するときにノード・グループを指定しておく必要があります。たとえば次のようにします。

```
db2 "create database partition group mygroup on node (0,1)"
db2 "create regular tablespace mytb in database partition group mygroup
   managed by system using ('mytb')"
```

上記の例では、次に、コレクションを使用可能化するときに mytb 表スペースを指定します。

例:

以下の例では、XML コレクションが使用可能にされます。

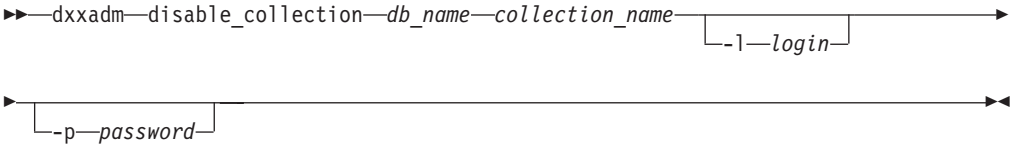
```
dxxadm enable_collection SALES_DB sales_ord
getstart_xcollection.dad -t orderspace
```

dxxadm コマンドの disable_collection オプション

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 オプションを使用して作成したコレクション表はドロップされません。

構文:



パラメーター:

表 21. 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*	データベースへの接続に使用されるユーザー 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*	データベースへの接続に使用されるパスワード。 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 7 章 XML Extender ユーザー定義タイプ

ユーザー定義タイプ (UDT) は、DB2® アプリケーションまたはツールで作成されるデータ・タイプです。 XML Extender では、XML 列で使用するために以下のユーザー定義タイプが作成されます。

- XMLVARCHAR
- XMLCLOB
- XMLFILE

データ・タイプは、XML 文書を保管するためのアプリケーション表の列を定義するために使用されます。 XML 文書は、ファイル名を指定することにより、ファイルとしてファイル・システムに保管することもできます。

XML Extender のすべてのユーザー定義タイプの修飾子は **DB2XML** です。これは DB2 UDB エクステンダーのユーザー定義タイプのスキーマ名 です。例を挙げます。

```
db2xml.XMLVarchar
```

XML Extender 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 22 は、UDT を説明しています。

表 22. XML Extender 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 Extender は XML 文書を外部サーバー・ファイルに保管します。 Text Extender 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を保持してください。

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。

Linux、Unix、および Windows® オペレーティング・システムにおける DB2 UDB の XML Extender では、*varchar_len* = 3K および *clob_len* = 2G となります。

XMLVARCHAR または XMLCLOB UDT のサイズを変更するには、XML Extender 用のデータベースを使用可能にする前に UDT を作成します。

手順:

使用可能なデータベースの XMLVARCHAR または XMLCLOB UDT のサイズを変更するには、次のようにします。

1. XML Extender 対応のデータベースのデータをすべてバックアップします。
2. すべての XML コレクション表または XML 列のサイド表をドロップします。
3. `dxxadm disable_db` コマンドで、データベースを使用不可にします。
4. XMLVARCHAR または XMLCLOB ユーザー定義タイプを作成します。
5. `dxxadm enable_db` コマンドで、データベースを使用可能にします。
6. 表を再作成し、再ロードします。

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 Extender が作成するサイド表には適用されません。

関連概念:

- 80 ページの『保管およびアクセス方式としての XML 列』
- 97 ページの『保管およびアクセス方式としての XML コレクション』
- 39 ページの『XML Extender を管理するための準備』
- 109 ページの『XML コレクションのマッピング体系』

第 8 章 XML Extender のユーザー定義関数

ユーザー定義関数 (UDF) は、データベース管理システムに定義され、SQL ステートメントで参照できる関数です。このセクションでは、DB2 UDB XML Extender が使用するユーザー定義関数について説明します。

XML Extender のユーザー定義関数のタイプ

XML Extender 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。

UDF はすべて、スキーマ名 DB2XML を持ちます。

XML Extender の関数のタイプを以下に示します。

保管関数
保管関数は、XML 対応列に原型の XML 文書を XML データ・タイプとして挿入します。

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。

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 Extender 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。

更新関数
更新関数は、ロケーション・パスによって指定された XML 文書全体、あるいは指定されたエレメント内容または属性値を変更して、値が更新された XML 文書のコピーを戻します。

XML ユーザー定義関数を使用すると、汎用の SQL データ・タイプに対して、検索を行うことができます。さらに、UDB UDB Net Search Extender を XML Extender 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。

制限: UDF でパラメーター・マーカーを使用するときには、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 Java™ データベース (JDBC) の制限上の要件です。

保管関数

XML Extender における保管関数の概要

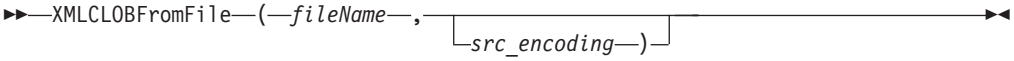
保管関数を使用して、XML 文書を DB2 UDB データベースに挿入します。 UDT ディレクトリーのデフォルト・キャスト関数を INSERT または SELECT ステートメント内に使用できます。さらに、XML Extender には XML 文書を UDT 基本データ・タイプ以外のソースから取り出して、指定した UDT に変換する UDF が備わっています。

XMLCLOBFromFile() 関数

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。
src_encoding	VARCHAR(100)	ソース・ファイルのエンコード方式

結果:

XMLCLOB を LOCATOR として

例:

以下の例では、XML 文書をサーバー上の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。サーバー・ファイルのエンコード方式は明示的に iso-8859-1 として指定されます。

```
EXEC SQL INSERT INTO sales_tab(ID, NAME, ORDER)
   VALUES('1234', 'Sriram Srinivasan',
           XMLCLOBFromFile('dxx_install/samples/db2xml
           /xml/getstart.xml
           ', 'iso-8859-1'))
```

ここで、*dxx_install* は XML Extender をインストールしたディレクトリーです。

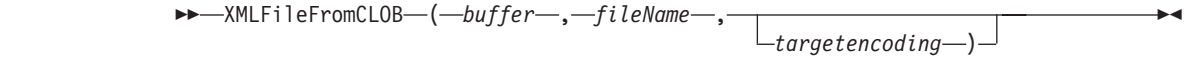
SALES_TAB 表の列 ORDER は、XMLCLOB タイプとして定義されます。

XMLFileFromCLOB() 関数

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 24. 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。
targetencoding	VARCHAR(100)	出力ファイルのエンコード方式

結果:

XMLFILE

例:

以下の例では、XML 文書を CLOB ロケーター (データベース・サーバー内で単一 LOB 値を表す値を持つホスト変数) 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。関数は ibm-808 で出力ファイルをエンコードします。

```
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, 'dxx_install/samples/db2xml
           /xml/getstart.xml', 'ibm-808'))
```

ここで、*dxx_install* は XML Extender をインストールしたディレクトリーです。

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。

XMLFileFromVarchar() 関数

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 25. XMLFileFromVarchar パラメーター

パラメーター	データ・タイプ	説明
buffer	VARCHAR(3K)	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。
targetencoding	VARCHAR(100)	出力ファイルのエンコード方式

結果:

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。関数は iso-8859-1 で出力ファイルをエンコードします。

```
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, 'dxx_install/samples/db2xml
           /xml/getstart.xml', 'iso-8859-1'))
```

ここで、*dxx_install* は XML Extender をインストールしたディレクトリーです。

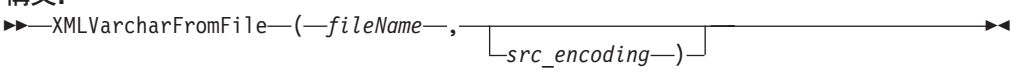
SALES_TAB 表の列 ORDER は、XMLFILE タイプとして定義されます。

XMLVarcharFromFile() 関数

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 26. *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。
src_encoding	VARCHAR(100)	ソース・ファイルのエンコード方式

結果:

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。サーバー・ファイルのエンコード方式は明示的に ibm-808 として指定されます。

```
EXEC SQL INSERT INTO sales_tab(ID, NAME, ORDER)
   VALUES('1234', 'Sriram Srinivasan',
          XMLVarcharFromFile('dxx_install/samples/db2xml
          /xml/getstart.xml', 'ibm-808'))
```

ここで、*dxx_install* は XML Extender をインストールしたディレクトリーです。

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、 ibm-808 でエンコードするように明示的に指定されたファイルから DB2 UDB にインポートして、それを XMLVARCHAR として保管します。

検索関数

XML Extender における検索関数

XML Extender には、検索に使用する多重定義関数 Content() が備わっています。この多重定義関数は、同じ名前をもつ 1 組の検索関数を参照しますが、データを検索する場所によって、その振る舞いが異なります。デフォルト・キャスト関数を使用して、XML UDT を基本データ・タイプにも変換できます。

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。この目的で、Content() を使用して、XMLFILE から CLOB に取り出すことができます。

- **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 UDB 内に保管された XML 文書を取り出し、 DB2 UDB 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。

– 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

以下のユーザー定義関数には、ソースまたは出力ファイルのエンコード方式を指定する新規パラメーターがあります。このパラメーターの値は、ユニコード標準準拠の国際コンポーネントにより認識される任意のコード・ページ名です。

```
db2xml.XMLVarcharFromFile(filename varchar(512), src_encoding  varchar(100))
returns XMLVarchar

db2xml.XMLCLOBFromFile(filename varchar(512), src_encoding varchar(100))
returns XMLCLOB AS LOCATOR

db2xml.XMLFileFromVarchar(doc varchar(3000), targetfilename varchar(512),
targetencoding varchar(100))
returns XMLFile

db2xml.XMLFileFromCLOB(doc CLOB(2G) as LOCATOR, targetfilename varchar(512),
targetencoding varchar(100))
returns XMLFile

db2xml.Content(doc XMLVarchar, targetfilename varchar(512),
targetencoding varchar(100))
returns varchar(512)

db2xml.Content(doc XMLCLOB as LOCATOR, targetfilename varchar(512),
targetencoding varchar(100))
returns varchar(512)
```

例:

ファイル /home/collins/xml/entail.xml の内容を varchar バッファーにインポートし、ソース・ファイルを iso-8859-1 でエンコードするように指定するには、以下のようにします。

```
db2xml.XMLVarcharFromFile('/home/collins/xml/entail.xml', 'iso-8859-1')
```

ファイルは varchar にインポートされ、iso-8859-1 からデータベース・コード・ページに変換されます。

varchar バッファーをファイル /home/raskolnikov/xml/confession.xml にエクスポートし、出力ファイルが ibm-808 でエンコードされるように指定するには、以下のようにします。

```
db2xml.Content('<sequence><thought>I did it!</thought></sequence>',
'/home/raskolnikov/xml/confession.xml', 'ibm-808')
```

バッファーの内容はファイルにエクスポートされ、データベース・コード・ページから ibm-808 に変換されます。次に XML ファイルのエンコード宣言が適切に更新されます。

以下のセクションに示す例では、各コマンドの先頭に『DB2』とタイプする必要のない DB2 UDB コマンド・シェルを使用していると仮定しています。

Content(): XMLFILE から CLOB への取り出し

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 27. 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。

結果:

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。

```
char    subsystem[20];
long   retcode = 0, reason = 0;
extern "OS" { int DSNALI(char * functn, ...); }

extern "OS" short DSNTIAR(struct sqlca *sqlca,
                          error_struct *error_message,
                          long *data_len);

EXEC SQL BEGIN DECLARE SECTION;
     SQL TYPE IS CLOB_LOCATOR xml_buff;
EXEC SQL END DECLARE SECTION;

/* Attach to subsystem */
rc = DSNALI("OPEN        ", subsystem, "PLANNAME",
     &retcode, &reason);
if ( retcode != 0 )
{
   /* print error message */
   goto exit;
}

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 */
     }
   }

EXEC SQL CLOSE c1;

/* Detach from sybsystem */
DSNALI("CLOSE       ", "SYNC", &retcode, &reason);
if ( retcode != 0 ) {
   /* print error message */
}
```

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。

関連タスク:

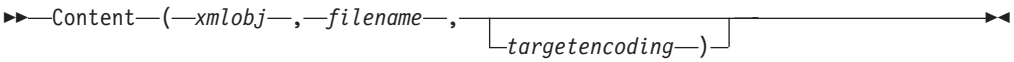
- 106 ページの『XML コレクションのデータ更新および削除』

Content(): XMLVARCHAR から外部サーバー・ファイルへの取り出し

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変更します。

パラメーター:

表 28. 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。
targetencoding	VARCHAR(100)	出力ファイルのエンコード方式

結果:

VARCHAR(512)

例:

以下の例では、XMLVARCHAR タイプとして保管されている XML の内容を探索して、それをサーバー上の外部ファイルに保管します。 UDF は 'ibm-808' でファイルをエンコードします。

```
CREATE table app1 (id int NOT NULL, order DB2XML.XMLVarchar);
INSERT into app1 values (1, '<?xml version="1.0"?>
   <!DOCTYPE SYSTEM "dxx_install/samples/db2xml/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>
          </Part>
     </Order>');

SELECT DB2XML.Content(order, 'dxx_install/samples/dad/getstart_column.dad'
, 'ibm-808')
  from app1 where ID=1;
```

関連タスク:

- 85 ページの『XML 文書を検索するための方法』

関連資料:

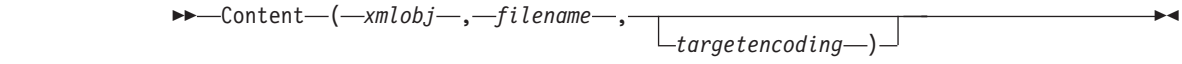
- 147 ページの『XML Extender における検索関数』

Content(): XMLCLOB から外部サーバー・ファイルへの取り出し

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変更します。

パラメーター:

表 29. 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。

表 29. *XMLCLOB* から外部サーバー・ファイルへのパラメーター *(続き)*

パラメーター	データ・タイプ	説明
targetencoding	VARCHAR(100)	出力ファイルのエンコード方式

結果:

VARCHAR(512)

例:

以下の例では、XMLCLOB タイプとして保管されている XML の内容を探索して、それをサーバー上の外部ファイルに保管します。 UDF は 'ibm-808' でファイルをエンコードします。

```
CREATE table app1 (id int NOT NULL, order DB2XML.XMLCLOB not logged);

INSERT into app1 values (1, '<?xml version="1.0"?>
  <!DOCTYPE SYSTEM "dxx_install/samples/db2xml/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>
      </Part>
    </Order>');

SELECT DB2XML.Content(order,
'dxx_install/samples/db2xml/xml/getstart.xml', 'ibm-808')
 from app1 where ID=1;
```

抽出関数

XML Extender における抽出関数

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 Extender 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 Extender 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回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。前者はスカラー関数、後者は表関数と呼ばれます。

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 UDB 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()

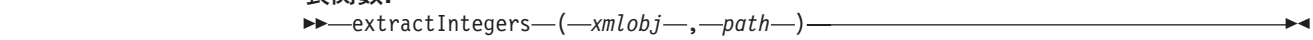
目的:

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 30. 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。

戻されるタイプ:

INTEGER

戻りコード:

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。値は INTEGER として抽出されます。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 UDB コマンド・シェルを使用していると仮定します。

```
CREATE TABLE t1(key INT);
INSERT INTO t1 values (
    DB2XML.extractInteger(DB2XML.XMLFile('/samples/db2xml
    /xml/getstart.xml
    '),
    '/Order/Part[@color="black "]/key'));
SELECT * from t1;
```

表関数の例:

以下の例では、販売注文のそれぞれの注文キー値が INTEGER として抽出されます。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 UDB コマンド・シェルを使用していると仮定します。

```
SELECT *
FROM TABLE(
  DB2XML.extractIntegers(DB2XML.XMLFile('/samples/db2xml/xml/getstart.xml'),
'/Order/Part/key')) AS X;
```

関連概念:

- 339 ページの『付録 D. XML Extender のための UDT 名および UDF 名』
- 143 ページの『XML Extender のユーザー定義関数のタイプ』

関連資料:

- 152 ページの『XML Extender における抽出関数』

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 31. 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。

戻されるタイプ:

SMALLINT

戻りコード:

returnedSmallint

例:

スカラー関数の例:

以下の例では、すべての販売注文におけるキーの値が SMALLINT として抽出されます。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 UDB コマンド・シェルを使用していると仮定します。

```
CREATE TABLE t1(key INT);
INSERT INTO t1 values (
    DB2XML.extractSmallint(db2xml.xmlfile('dxx_install
     /samples/db2xml/xml/getstart.xml'),
    '/Order/Part[@color="black "]/key'));
SELECT * from t1;
```

表関数の例:

以下の例では、すべての販売注文におけるキーの値が SMALLINT として抽出されます。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 UDB コマンド・シェルを使用していると仮定します。

```
SELECT *
  FROM TABLE(
      DB2XML.extractSmallints(DB2XML.XMLFile('dxx_install
      /samples/db2xml/xml/getstart.xml'),
      '/Order/Part/key'))   AS X;
```

関連概念:

- 81 ページの『XML 列データの索引の使用』
- 339 ページの『付録 D. XML Extender のための UDT 名および UDF 名』
- 143 ページの『XML Extender のユーザー定義関数のタイプ』

関連資料:

- 152 ページの『XML Extender における抽出関数』
- 295 ページの『XML Extender ストアード・プロシージャーの戻りコード』

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:

スカラー関数:

表関数:

パラメーター:

表 32. *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。

戻されるタイプ:

DOUBLE

戻りコード:

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。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。

```
CREATE TABLE t1(price DECIMAL(9,2));
INSERT INTO t1 values (
    DB2XML.extractDouble(DB2XML.xmlfile('dxx_install
    /samples/db2xml/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。例では、各コマンドの先頭に DB2 UDB とタイプする必要のない、 DB2 UDB コマンド・シェルを使用していると仮定します。

```
SELECT CAST(RETURNEDDOUBLE AS DOUBLE)
  FROM TABLE(
      DB2XML.extractDoubles(DB2XML.XMLFile('dxx_install
       /samples/db2xml/xml/getstart.xml'),
      '/Order/Part/ExtendedPrice'))   AS X;
```

関連概念:

- 339 ページの『付録 D. XML Extender のための UDT 名および UDF 名』

関連資料:

- 152 ページの『XML Extender における抽出関数』

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 33. 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。

戻されるタイプ:

REAL

戻りコード:

returnedReal

例:

スカラー関数の例:

以下の例では、ExtendedPrice の値が REAL として抽出されます。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 UDB コマンド・シェルを使用していると仮定します。

```
CREATE TABLE t1(price DECIMAL(9,2));
INSERT INTO t1 values (
    DB2XML.extractReal(DB2XML.xmlfile('dxx_install
    /samples/db2xml/xml/getstart.xml'),
    '/Order/Part[@color="black"]/ExtendedPrice'));
SELECT * from t1;
```

表関数の例:

以下の例では、ExtendedPrice の値が REAL として抽出されます。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 UDB コマンド・シェルを使用していると仮定します。

```
SELECT CAST(RETURNEDREAL AS REAL)
  FROM TABLE(
      DB2XML.extractReals(DB2XML.XMLFile('dxx_install
      /samples/db2xml/xml/getstart.xml'),
      '/Order/Part/ExtendedPrice'))   AS X;
```

関連概念:

- 339 ページの『付録 D. XML Extender のための UDT 名および UDF 名』
- 143 ページの『XML Extender のユーザー定義関数のタイプ』

関連資料:

- 152 ページの『XML Extender における抽出関数』
- 294 ページの『XML Extender UDF の戻りコード』

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 34. 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。

戻されるタイプ:

CHAR

戻りコード:

returnedChar

例:

スカラー関数の例:

以下の例では、Name の値が CHAR として抽出されます。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 UDB コマンド・シェルを使用していると仮定します。

```
CREATE TABLE t1(name char(30));
INSERT INTO t1 values (
    DB2XML.extractChar(DB2XML.xmlfile('dxx_install
    /samples/db2xml/xml/getstart.xml'),
    '/Order/Customer/Name'));
SELECT * from t1;
```

表関数の例:

以下の例では、Color の値が CHAR として抽出されます。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 UDB コマンド・シェルを使用していると仮定します。

```
SELECT *
  FROM TABLE(
      DB2XML.extractChars(DB2XML.XMLFile('dxx_install
       /samples/db2xml/xml/getstart.xml'),
      '/Order/Part/@color'))   AS X;
```

関連資料:

- 152 ページの『XML Extender における抽出関数』
- ix ページの『構文図の読み方』

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 35. 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。

戻されるタイプ:

VARCHAR(4K)

戻りコード:

returnedVarchar

例:

スカラー関数の例:

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
```

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 UDB コマンド・シェルを使用していると仮定します。 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。

表関数の例:

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
```

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 UDB コマンド・シェルを使用していると仮定します。 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。

スカラー関数の例:

以下の例では、Name の値が VARCHAR として抽出されます。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 UDB コマンド・シェルを使用していると仮定します。

```
CREATE TABLE t1(name varchar(30));
INSERT INTO t1 values (
    DB2XML.extractVarchar(DB2XML.xmlfile('dxx_install
     /samples/db2xml/xml/getstart.xml'),
    '/Order/Customer/Name'));
SELECT * from t1;
```

表関数の例:

以下の例では、Color の値が VARCHAR として抽出されます。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 UDB コマンド・シェルを使用していると仮定します。

```
SELECT*
  FROM TABLE(
      DB2XML.extractVarchars(DB2XML.XMLFile('dxx_install
      /samples/xml/getstart.xml'),
      '/Order/Part/@color'))   AS X;
```

関連概念:

• 339 ページの『付録 D. XML Extender のための UDT 名および UDF 名』

- 143 ページの『XML Extender のユーザー定義関数のタイプ』

関連資料:

- 152 ページの『XML Extender における抽出関数』
- 294 ページの『XML Extender UDF の戻りコード』

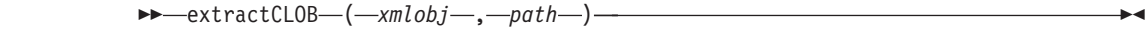
extractCLOB() および extractCLOBs()

目的:

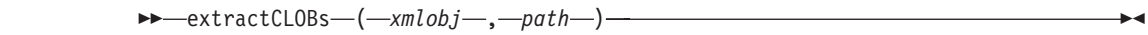
XML 文書の部分を、サブエレメントを含む、エレメントおよび属性マークアップや、エレメントおよび属性の内容とともに抽出します。この関数は、他の抽出関数とは異なります。他の抽出関数はエレメントと属性の内容だけしか戻しません。extractClob(s) 関数は文書の部分を抽出するために使用するのに対し、extractVarchar(s) および extractChar(s) は値だけを抽出するために使用します。

構文:

スカラー関数:



表関数:



パラメーター:

表 36. 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。

戻されるタイプ:

CLOB(10K)

戻りコード:

returnedCLOB

例:

スカラー関数の例:

この例では、すべての名前エレメントの内容とタグは、購入注文から抽出されます。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 UDB コマンド・シェルを使用していると仮定します。

```
CREATE TABLE t1(name DB2XML.xmlclob);
INSERT INTO t1 values (
    DB2XML.extractClob(DB2XML.xmlfile('dxx_install
     /samples/db2xml/xml/getstart.xml'),
    '/Order/Customer/Name'));
SELECT * from t1;
```

表関数の例:

この例では、カラー属性のすべてが購入注文から抽出されます。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 UDB コマンド・シェルを使用していると仮定します。

```
SELECT *
  FROM TABLE(
      DB2XML.extractCLOBs(DB2XML.XMLFile('dxx_install
     /samples/db2xml/xml/getstart.xml'),
      '/Order/Part/@color'))   AS X;
```

関連概念:

- 143 ページの『XML Extender のユーザー定義関数のタイプ』

関連資料:

- 152 ページの『XML Extender における抽出関数』

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。日付は YYYY-MM-DD の形式である必要があります。

構文:

スカラー関数:

表関数:

パラメーター:

表 37. 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。

戻されるタイプ:

DATE

戻りコード:

returnedDate

例:

スカラー関数の例:

以下の例では、ShipDate の値が DATE として抽出されます。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 UDB コマンド・シェルを使用していると仮定します。

```
CREATE TABLE t1(shipdate DATE);
INSERT INTO t1 values (
    DB2XML.extractDate(DB2XML.xmlfile('dxx_install
    /samples/db2xml/xml/getstart.xml'),
    '/Order/Part[@color="red   "]/Shipment/ShipDate'));
SELECT * from t1;
```

表関数の例:

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('dxx_install
    /samples/db2xml/xml/getstart.xml'),
      '/Order/Part[@color="black "]/Shipment/ShipDate'))   AS X;
```

関連概念:

- 143 ページの『XML Extender のユーザー定義関数のタイプ』

関連資料:

- 152 ページの『XML Extender における抽出関数』
- 294 ページの『XML Extender UDF の戻りコード』

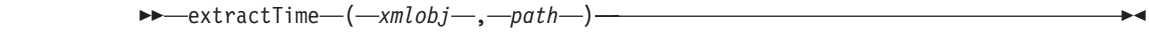
extractTime() および extractTimes()

目的:

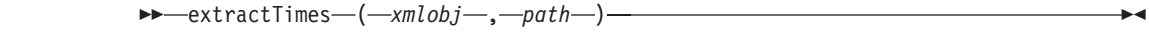
エレメント内容または属性値を XML 文書から抽出して、データを TIME タイプで戻します。

構文:

スカラー関数:



表関数:



パラメーター:

表 38. 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。

表 38. *extractTime* および *extractTimes* 関数のパラメーター (続き)

パラメーター	データ・タイプ	説明
path	VARCHAR	エレメントまたは属性のロケーション・パス。

戻されるタイプ:

TIME

戻りコード:

returnedTime

例:

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 UDB コマンド・シェルを使用していると仮定します。

スカラー関数の例:

```
CREATE TABLE t1(testtime  TIME);
INSERT INTO t1 values (
    DB2XML.extractTime(DB2XML.XMLCLOB(
         '<stuff><data>11.12.13</data></stuff>'), '//data'));
SELECT * from t1;
```

表関数の例:

```
select *
from table(
    DB2XML.extractTimes(DB2XML.XMLCLOB(
        '<stuff><data>01.02.03</data><data>11.12.13</data></stuff>'),
        '//data')) as x;
```

関連概念:

- 339 ページの『付録 D. XML Extender のための UDT 名および UDF 名』
- 143 ページの『XML Extender のユーザー定義関数のタイプ』

関連資料:

- 152 ページの『XML Extender における抽出関数』

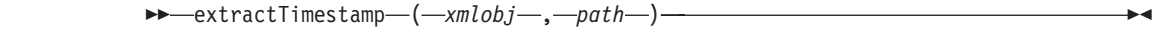
extractTimestamp() および extractTimestamps()

目的:

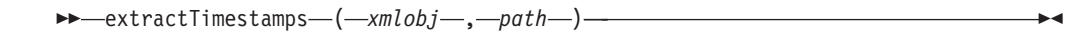
エレメント内容または属性値を XML 文書から抽出して、データを TIMESTAMP タイプで戻します。

構文:

スカラー関数:



表関数:



パラメーター:

表 39. 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。

戻されるタイプ:

TIMESTAMP

戻りコード:

returnedTimestamp

例:

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 UDB コマンド・シェルを使用していると仮定します。

スカラー関数の例:

```
CREATE TABLE t1(testtimestamp  TIMESTAMP);
INSERT INTO t1 values (
    DB2XML.extractTimestamp(DB2XML.XMLCLOB(
        '<stuff><data>2003-11-11-11.12.13.888888</data></stuff>'),
        '//data'));
SELECT * from t1;
```

表関数の例:

```
select * from
table(DB2XML.extractTimestamps(DB2XML.XMLClob(
    '<stuff><data>2003-11-11-11.12.13.888888
    </data><data>2003-12-22-11.12.13.888888</data></stuff>'),
    '//data')) as x;
```

XML Extender は自動的に XML 文書から抽出したタイム・スタンプを正規化し、必要に応じて DB2 タイム・スタンプの形式に従います。タイム・スタンプは、yyyy-mm-dd-hh.mm.ss.nnnnnn 形式、または yyyy-mm-dd-hh mm.ss.nnnnnn 形式に正規化されます。例を挙げます。

```
2003-1-11-11.12.13
```

これは次のように正規化されます。

```
2003-01-11-11.12.13.000000
```

関連概念:

- 339 ページの『付録 D. XML Extender のための UDT 名および UDF 名』
- 143 ページの『XML Extender のユーザー定義関数のタイプ』

関連資料:

- 152 ページの『XML Extender における抽出関数』
- 294 ページの『XML Extender UDF の戻りコード』

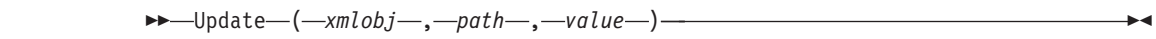
XML Extender における更新関数

Update() 関数は、XML 列に保管されている 1 つ以上の 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。

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 40. 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。 **制限:** Update 関数には、出力エスケープを使用不能にするオプションはありません。extractClob の出力 (これはタグ付きの断片) をこの関数で挿入することはできません。テキスト値だけを使用してください。

制限: Update UDF は、属性を伴う述部を持つロケーション・パスはサポートしますが、エレメントがあるパス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"]'
```

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 = db2xml.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。以降のセクションでは、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見え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成する必要があります。 表 41 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。

表 41. 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。

表 41. 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
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 サブセットは保存されません。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。
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="01/02/2003"	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失われます。 • エンティティーは置換されます。

表 41. 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
テキスト・ ノード	This section is about my dog &mydog;.	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失われます。 • エンティティーは置換されます。

複数回出現

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、文書に複数回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、属性を伴う述部を持つロケーション・パスはサポートしますが、エレメントがあるパスはサポートしないことに注意してください。

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。

表 42. 更新の前後の XML 文書

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 :

表 42. 更新の前後の XML 文書 (続き)

```
<?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="07/01/2003">
       <!-- first section -->
       <section>This is a section in Chapter
         One.</section>
  </chapter>
  <chapter id="2" date="01/02/2003">
       <section>This is a section in Chapter
         Two.</section>
        <footnote>A footnote in Chapter Two
         is here.</footnote>
  </chapter>
  <price date="12/22/2003" time="11.12.13"
       timestamp="2003-12-22-11.12.13.888888">
       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 :

表 42. 更新の前後の XML 文書 (続き)

```
<?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。

検証機能

DB2 XML Extender は、XML 文書を XML スキーマか DTD に対して妥当性検査をする 2 つのユーザー定義関数 (UDF) を提供します。

XML 文書内のエレメントは、関連するエレメント・タイプの規則を満足していれば、所定のスキーマに応じて有効と判断されます。すべてのエレメントが有効ならば、文書全体が有効です。しかし、DTD を使用すると特定のルート・エレメントを要求する方法がありません。文書が妥当であれば検証機能は、1 を戻します。文書が無効であれば、0 を戻し、トレース・ファイルにエラー・メッセージを書きこみます。その関数とは次のものです。

db2xml.svalidate:
XML 文書のインスタンスを指定されたスキーマに対して妥当性検査をします。

db2xml.dvalidate:
XML 文書のインスタンスを指定された DTD に対して妥当性検査をします。

SVALIDATE() 関数

この関数は、XML 文書を指定されたスキーマ (または XML 文書に指定されたもの) に対して妥当性検査を行い、文書が有効ならば 1、無効ならば 0 を戻します。

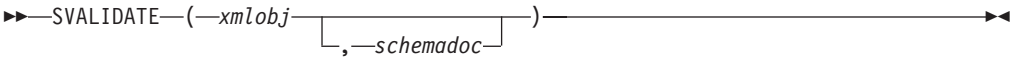
この関数は、XML 文書とスキーマがファイル・システムに存在しているか、 DB2 で CLOB として存在することを前提としています。

SVALIDATE 関数を実行する前に、XML Extender で次のコマンドを実行して、データベースを使用可能にしてください。

```
dxxadm enable_db mydbname
```

XML 文書が妥当性検査に失敗すると、エラー・メッセージが XML Extender トレース・ファイルに書き込まれます。 SVALIDATE コマンドを実行する前に、トレースを使用可能にします。

構文



パラメーター

表 43. SVALIDATE パラメーター

パラメーター	データ・タイプ	説明
xmlobj	VARCHAR(256)	検査する XML 文書のファイル・パス
	CLOB(2G)	検査する文書を含む XML 列
schemadoc	VARCHAR(256)	スキーマ文書のファイル・パス
	CLOB(2G)	スキーマを含む XML 列

例

例 1: この例は equiplog2001.xml を、文書内に指定されたスキーマに対して妥当性検査を行います。

```
db2 values db2xml.svalidate("/home/jean/xml/equiplog2001.xml")
```

例 2: この例では、XML 文書を指定されたスキーマを使用して妥当性検査します。文書とスキーマは両方とも DB2 UDB の表に保管されています。

```
db2 select db2xml.svalidate(doc,schema) from equiplogs where id=1
```

DVALIDATE() 関数

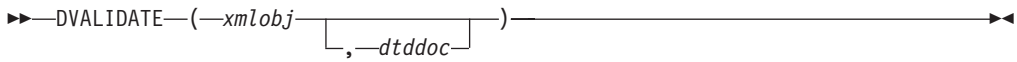
この関数は、指定された DTD (または XML 文書に指定されたもの) に対して XML 文書の妥当性検査を行い、文書が有効ならば 1、無効ならば 0 を戻します。この関数は、XML 文書と DTD がファイル・システムに存在しているか、 DB2 で CLOB として存在することを前提としています。

DVALIDATE 関数を実行する前に、XML Extender で次のコマンドを実行して、使用するデータベースを使用可能にしてください。

```
dxxadm enable_db mydbname
```

XML 文書が妥当性検査に失敗すると、エラー・メッセージが XML Extender トレース・ファイルに書き込まれます。 DVALIDATE コマンドを実行する前に、トレースを使用可能にします。

構文



パラメーター

表 44. DVALIDATE パラメーター

パラメーター	データ・タイプ	説明
xmlobj	VARCHAR(256)	検査する XML 文書のファイル・パス
	CLOB(2G)	検査する文書を含む XML 列
dtddoc	VARCHAR(256)	DTD 文書のファイル・パス
	CLOB(2G)	DTD_REF 表または通常の表からの DTD を含む XML 列。

例

例 1: この例は equiplog2001.xml を、文書内に指定された DTD に対して妥当性検査を行います。

```
db2 values db2xml.dvalidate(/home/jean/xml/equiplog2001.xml)
```

例 2: この例は、XML 文書を指定された DTD を使用して妥当性検査します。文書と DTD は、両方ともファイル・システムに存在します。

```
db2 values db2xml.dvalidate (c:/xml/equiplog.xml,c:/xml/dtds/equip.dtd)
```

例 3: この例は、指定された DTD を使用して XML 文書を妥当性検査します。文書と DTD は、両方とも DB2 UDB 表に保管されています。

```
db2 values db2xml.dvalidate (doc,dtdid) from equiplogs, db2xml.dtd_ref ¥
    where dtdid="equip.dtd"
```

関連資料:

- 293 ページの『XML Extender 用トレースの開始』

第 9 章 文書アクセス定義 (DAD) ファイル

XML 列のための DAD ファイルの作成

この作業は、XML 列を定義し、使用可能にするという大きな作業の一部です。

DAD ファイルの最新情報については、XML Extender の Web サイト www.ibm.com/software/data/db2/extenders/xmlext/downloads.html を参照してください。

XML データにアクセスするため、および XML 表内の XML データ用の列を使用可能にするためには、文書アクセス定義 (DAD) ファイルを定義する必要があります。このファイルは、列から検索する必要のあるデータの属性とキー・エレメントを定義します。 XML 列に対しては、DAD ファイルは主として、その列に保管される文書に索引を作成する方法を指定します。 DAD ファイルは、また、XML 列に挿入された文書の妥当性を検査するために使用される DTD またはスキーマも指定します。 DAD ファイルは CLOB データ・タイプとして保管され、そのサイズの上限は 100 KB です。

前提条件:

DAD ファイルを作成する前に、以下を行う必要があります。

- 検索で頻繁に使用すると思われるエレメントと属性の決定。指定したエレメントまたは属性は、サイド表に抽出され、XML Extender が高速検索に使用します。
- サイド表に索引作成される各エレメントまたは属性を表すロケーション・パスの決定。また、変換したいエレメントまたは属性のデータ・タイプも指定する必要があります。

手順:

DAD ファイルを作成するには、以下のように行います。

1. テキスト・エディターで新しい文書を作成し、次の構文を入力します。

   ```
   <?XML version="1.0"?>
   <!DOCTYPE DAD SYSTEM <"path/dtd/dad.dtd">.
   ```

   "path/dtd/dad.dtd" は、DAD ファイルのための DTD のパスとファイルの名前です。 DTD は `dxx_install¥samples¥db2xml¥dtd` にあります。

2. ステップ 1 による行の後ろに DAD タグを挿入します。

   ```
   <DAD>
   </DAD>
   ```

   このエレメントには、すべての他のエレメントが含まれます。

3. 文書と列のための妥当性検査を指定します。
   - XML 文書をデータベースに挿入する前に文書全体を DTD またはスキーマに対して妥当性検査したい場合は、次のようにします。
     - 適切なタグを挿入し、文書の妥当性検査方法を指定します。

       ```
       <dtdid>path/dtd_name.dtd</dtdid>
       ```

- 以下のタグを挿入し、スキーマを使用する文書の妥当性検査を行います。

```
<schemabindings>
<nonamespacelocation location="path/schema_name.xsd"/>
</schemabindings>
```

- 次のタグを挿入して列の妥当性検査をします。

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

- 文書の妥当性検査を望まない場合は、次のタグを使用します。

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

4. XML データのためのアクセスと保管の方式として XML 列を使用することを指定するには、<Xcolumn> </Xcolumn> タグを挿入します。
5. サイド表を指定します。作成したい各サイド表について、次を行います。
   a. <table></table> タグを指定します。たとえば次のようにします。

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

   b. 表タグの内側に、サイド表に入れたい各列に対して <column> タグを挿入します。各列には、name、type、path、および multi_occurrence の 4 つの属性があります。

例:

```
<table name="person_names">>
<column name ="fname"
     type="varchar(50)"
     path="/person/firstName"
     multi_occurrence="NO"/>
<column name ="lname"
     type="varchar(50)"
     path="/person/lastName"
     multi_occurrence="NO"/>
</table>
```

ここで、

name サイド表に作成される列の名前を指定します。

type 索引が付いたエレメントまたは属性のそれぞれに対して、サイド表内の SQL データ・タイプを指定します。

path 索引作成されるエレメントまたは属性のそれぞれについて、XML 文書のロケーション・パスを指定します。

multi_occurrence
パス属性で参照されるエレメントまたは属性が XML 文書内で複数回出現する可能性があるかどうかを指定します。 **multi_occurrence** に指定できる値は、 **YES** または **NO** です。値が**NO** の場合は、表ごとに複数の列を指定できます。値が**YES** の場合は、サイド表には 1 列だけ指定することができます。

6. DAD 拡張子を指定してファイルを保管します。

以下は、完全な DAD ファイルの例を示しています。

```
<?xml version="1.0"?>
<!DOCTYPE DAD SYSTEM "c:¥dxx_installsamples¥db2xml¥dtd¥dad.dtd">
<DAD>
<dtid>C:¥SG246130¥code¥person.dtd</dtdid>
```

```
<validation>YES</validation>
<Xcolumn>
  <table name="person_names">
   <column name="fname"
     type="varchar(50)"
     path="/person/firstName"
     multi_occurrence="NO"/>
   <column name="lname"
     type="varchar(50)"
     path="/person/lastName"
     multi_occurrence="NO"/>
</table>
<table name="person_phone_number">
   <column name="pnumber"
     type="varchar(20)"
     path="/person/phone/number"
     multi_occurrence="YES"/>
</table>
<table name="person_phone_number">
   <column name="pnumber"
     type="varchar(20)"
     path="/person/phone/number"
     multi_occurrence="YES"/>
</table>
<table name="pesron_phone_type">
   <column name="ptype"
     type="varchar(20)"
     path="/person/phone/type"
     multi_occurrence="YES"/>
</table>
<Xcolumn>
</DAD>
```

これで DAD ファイルが作成できたので、XML 列を定義し、使用可能にするための次のステップは、XML 文書を保管する表を作成することです。

関連概念:

- 97 ページの『保管およびアクセス方式としての XML コレクション』
- 177 ページの『XML コレクションの DAD ファイル』
- 193 ページの『DAD チェッカー』

関連タスク:

- 193 ページの『DAD チェック・プログラムの使用』

XML コレクションの DAD ファイル

XML コレクションの場合、DAD ファイルで、XML 文書の構造が DB2® 表にマップされ、それから文書を合成できます。 DAD ファイルにより、文書を DB2 UDB 表に分解することもできます。

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。 DAD ファイルは、XML データとリレーショナル・データとの関係 (リレーションシップ) を定義するためにも使用されます。

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、またはストアード・プロシージャーで、 DAD ファイルを XML コレクションのために使用する際

に指定する必要があります。 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 Extender 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 ファイルで <Xcollection> 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表、または XML Extender が最近作成した表などです。

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
: エレメントを識別します。これは、ルート・エレメントまたは子エレメントです。

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。

179 ページの図 14 は、DAD ファイルで使用されているマッピングの断片を示しています。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:¥dxx¥samples¥db2xml¥dtd¥dad.dtd">
<DAD>
  ...
<Xcollection>
<SQL_stmt>
       ...
</SQL_stmt>
<prolog>?xml version="1.0"?</prolog>
<doctype>!DOCTYPE Order SYSTEM
                  ""c:¥dxx¥samples¥db2xml¥dtd¥getstart.dtd""</doctype>
<root_node>
 <element_node name="Order">      --> エレメント <Order> を識別する。
    <attribute_node name="key">   --> 属性 "key" を識別する。
       <column name="order_key"/> --> 列の名前 "order_key" を定義する。
                                      エレメントと属性はこれにマップされる。
  </attribute_node>
  <element_node name="Customer">  -->  <Order> の子エレメントを
                                         <Customer> として識別する。
    <text_node>                   --> エレメント <Customer> の
                                      CDATA テキストを指定する。
      <column name="customer">    --> 列の名前 "customer" を定義する。
                                      子エレメントはこれにマップされる。
   </text_node>
  </element_node>
       ...
 </element_node>

      ...
</root_node>
</Xcollection>
</DAD>
```

図 14. XML コレクション表にマップされた XML 文書のノード定義

この例では、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 Extender は、<stylesheet> を使用して、スタイルシートのため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>
```

DAD ファイルの作成と更新には、任意のテキスト・エディターを使用できます。

関連概念:

- 109 ページの『XML コレクションのマッピング体系』

SQL 合成

同じ名前の列を使用して XML 文書を合成することができます。いろいろな表からのものであっても、選択した列で同じ名前のものは、 SQL ステートメントの select

文節でそれぞれの変数が異なるものになるように、ユニークの別名で示す必要があります。以下の例は、同じ名前をもつ列にユニークな別名を指定する方法を示しています。

```
<SQL_stmt>select o.order_key as oorder_key,
                 key customer_name, customer_email,
                 p.part_key p.order_key as porder_key,
                 color, qty, price, tax, ship_id, date, mode
           from order_tab o.part_tab p
           order by order_key, part_key</SQL_stmt>
```

また、生成されたランダムな値を持つ列を使用して XML 文書を合成することができます。 DAD ファイル内の SQL ステートメントがランダムな値をもつ場合は、ランダム値関数に、それを ORDER BY 文節で使用するための別名を指定する必要があります。この要件は、指定された表のいかなる列にも、その値が関連していないために必要です。以下の例の ORDER BY 文節の最後にある、Generate_unique の別名を参照してください。

```
<SQL_stmt>select o.order_key, customer_name,customer_email,
                 p.part_key,color,qty,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>2000 and
                o.order_key=o.order_key and s.part_key
          order by order_key, part_key,ship_id</SQL_stmt>
```

RDB ノードの合成

以下の制約事項が RDB ノード合成に適用されます。

- すべての non-root_node RDB ノード DAD ファイルに関連した条件は、リテラルと比較しなければならない。
- トップレベルの RDB_node に関連した条件内の各等式は、 2 つの表の列間の結合関係を記述したものであり、他の等式とは別に適用される。つまり、AND によって結び付けられたすべての述部は、同時に単一の結合条件には適用されないため、文書が合成されたときに外部結合をシミュレートします。表の各ペア間の親・子関係は、DAD ファイル内のそれら表の相対的なネスティングによって決まります。たとえば次のようにします。

```
<condition>order_tab.order_key=part_tab.order_key AND
part_tab.part_key=ship_tab.part_key</condition>
```

NULL 値を持つ行からの構成

XML 文書を構成するために NULL 値を持つ列を使用できます。

以下の例は、列 *Col 1* に NULL 値を含む行を持つ、 *MyTable* 表に XML 文書フォームをどのように作成できるかを示しています。この例で使用されている DAD は *nullcol.dad* です。

```
<?xml version="1.0"?>
<!DOCTYPE DAD SYSTEM "c:¥dxx¥dtd¥dad.dtd">
<DAD>
<validation>NO validation>NO>
<Xcollection>
```

```
<SQL_stmt>SELECT 1 as X, Col1 FROM MyTable order by X, Col1<¥SQL_stmt>
<prolog>?xml version="1.0"?prolog>?xml version="1.0"?>
<doctype>!DOCTYPE Order SYSTEM "e:¥t3xml¥x.dtd">
<root_node>
<element_node name="MyColumn">
<element_node name="Column1" multi_occurrence="YES">
  <text_node>
  <column name="Col1"/>
  </text_node>
 </element_node>
</element_node>
</rootт_node>
</Xcollection>
</DAD>

MyTable
_________
| Col 1 |
|_______|
| 1 |
|_______|
| 3 |
|_______|
| - |
|_______|
```

tests2x mydb nullcol.dad result_tab を実行するか、または dxxGenXML を使用して、以下の文書を生成します。3 番目の Column1 エレメントは NULL 値を表すことに注意してください。

```
<?xml version="1.0"?>
<!DOCTYPE Order SYSTEM "e:¥t3xml¥x.dtd">
<MyColumn>
 <Column1>1</Column1>
 <Column1>3</Column1>
 <Column1></Column1>
</MyColumn>
```

- すべての non-root_node RDB ノード DAD ファイルに関連した条件は、リテラルと比較しなければならない。
- すべての下位の RDB ノード DAD ファイルに関連した条件は、リテラルと比較しなければならない。
- root_node に関連した条件は、RDB ノード合成に関連する表の間の関係を記述したものである。例えば、基本外部キーの関係などです。
- トップレベルの RDB_node に関連した条件内の各等式は、 2 つの表の列間の結合関係を記述したものであり、他の等式とは別に適用される。つまり、AND によって結び付けられたすべての述部は、同時に単一の結合条件には適用されないため、文書が合成されたときに外部結合をシミュレートします。表の各ペア間の親・子関係は、DAD ファイル内のそれら表の相対的なネスティングによって決まります。たとえば次のようにします。

```
<condition>order_tab.order_key=part_tab.order_key AND
part_tab.part_key=ship_tab.part_key</condition>
```

DAD ファイル用の DTD

このトピックでは、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 dad.dtd です。次の例は DAD ファイル用の DTD を示しています。

```
<?xml encoding="US-ASCII"?>

  <!ELEMENT DAD ((schemabindings | dtdid)?, validation,
  (Xcolumn | Xcollection))>
  <!ELEMENT dtdid (#PCDATA)>
  <!ELEMENT schemabindings (nonamespacelocation)>
  <!ELEMENT nonamespacelocation (empty)>
  <!ATTLIST nonamespacelocation location CDATA #REQUIRED>
  <!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?, prolog, doctype, root_node)>
  <!ELEMENT SQL_stmt (#PCDATA)>
  <!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 process_instruction_node (#PCDATA)>
```

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、直前の例から引用しています。

DTDID エレメント

XML Extender に提供される DTD は DTD_REF 表に保管されています。各 DTD は、DAD ファイルの DTDID タグに用意されたユニークの ID で識別されます。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 コレクションでは、このエレメントは、入力および出力の XML 文書の妥当性検査を行う場合にのみ必要です。XML 列については、このエレメントは入力 XML 文書の妥当性を検査するためにだけ必要です。 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 つ以上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 つ以上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>
```

エレメント table は、1 つの必須の属性と 2 つの暗黙の属性を持っています。

name 属性

サイド表の名前を指定します。

key 属性

表のただ 1 つの主キーです。

orderBy 属性

XML 文書を生成する際、複数回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。

table エレメントの子エレメントは次の 1 つです。

column エレメント

CDATA ノードの属性を入力の XML 文書から表の列にマップします。

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 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 Extender 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。

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、結果の文書の DOCTYPE を指定するために使用されます。明示的な値を定義します。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。

doctype の子エレメントは次の 1 つです。

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)>`

RDB_node

DB2 UDB 表を定義します。XML エレメントの内容または XML 属性の値は、この表に保管され、あるいは、この表から検索されます。rdb_node は、element_node、text_node および attribute_node の子エレメントで、次のような子エレメントを持ちます。

表 (table)

エレメントまたは属性の内容を保管する表を指定します。

列 (column)

エレメントまたは属性の内容を保管する列を指定します。

条件 (condition)

列の条件を指定します。オプション。

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 DAD 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 0 または 1 つの 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 つ以上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

オプショナル : 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。

列 (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。

列 (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。

関連概念:

- 177 ページの『XML コレクションの DAD ファイル』

関連タスク:

- 187 ページの『DAD ファイルの値の動的オーバーライド』

DAD ファイルの値の動的オーバーライド

手順:

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* パラメーター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。

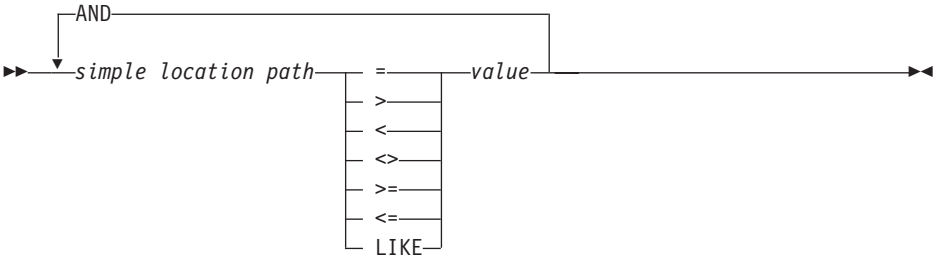
XML_OVERRIDE
DAD ファイル内の条件を XPath に基づく条件によってオーバーライドすることを指定します。

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。入力値の構文は、*overrideType* パラメーター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。

  SQL_OVERRIDE を SQL ステートメントとして使用する場合、 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 を指定する場合、入力値は 1 つ以上の式を含むストリングです。

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。



構文には、次のコンポーネントが入ります。

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・パスを指定します。

演算子
構文図に示されている SQL 演算子には、式のその他の部分から演算子を分離するためにスペースを入れることができます。

演算子の前後のスペースはオプションです。 LIKE 演算子をスペースで囲むことは必須です。

value
単一引用符で前後を囲んだ数値またはストリング。

AND
AND は、同じロケーション・パス上の論理演算子として扱われます。オーバーライド・ストリングで、単純ロケーション・パスが複数回指定されると、その単純ロケーション・パスのすべての述部は、同時に適用されます。

XML_OVERRIDE 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。

次の例は、SQL_OVERRIDE と XML_OVERRIDE を使用して、動的なオーバーライドを行うものです。

例 1: SQL_OVERRIDE を使用するストアード・プロシージャー。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 より大にして、<SQL_stmt> の値をオーバーライドします。

```
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[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   collection_ind;

  short   ovtype_ind;
  short   ov_ind;
  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");
  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)  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 db2xml.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);
```

例 2: XML_OVERRIDE を使用するストアード・プロシージャー。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 Extender は、単純ロケーション・パスをマップ先 DB2 UDB 列に変換します。

```
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[256];   /* override, XPATH condition */
  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   collection_ind;
  short   ovtype_ind;
  short   ov_ind;
  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");
  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);
```

複数のオーバーライド

XML Extender は同じパス上の複数のオーバーライドをサポートします。 RDB ノードに指定されたすべてのオーバーライドが受け入れられます。

同じロケーション・パス上の複数の XML オーバーライドを指定して、検索の設定条件を絞り込むことができます。以下の例では、XML 文書は test.dad ファイルを使用する 2 つの表から合成されます。

表 45. 部門表

部門番号	部門名
10	エンジニアリング

表 45. 部門表 (続き)

部門番号	部門名
20	操作
30	営業

表 46. 従業員表

従業員番号	部門番号	給料
123	10	$98,000.00
456	10	$87,000.00
111	20	$65,000.00
222	20	$71,000.00
333	20	$66,000.00
500	30	$55,000.00

以下で示されている DAD ファイル test.dad は、変数 *deptno* を値 10 と比較した条件を含んでいます。 10 より大きく 30 より小さい値に検索を拡張するには、この条件をオーバーライドする必要があります。 dXXGenXML を呼び出す場合には以下のように、オーバーライド・パラメーターを設定する必要があります。

```
/ABC.com/Department>10 AND /ABC.com/Department<30

<?xml version="1.0"?>
<!DOCTYPE DAD SYSTEM "C:¥dxx_xml¥test¥dtd¥dad.dtd">
<DAD>
<dtdid>E:¥dtd¥lineItem.dtd</dtdid>
<validation>NO</validation>
<Xcollection>
<porlog>?xmol version="1.0"?</prolog>
<doctype>!DOCTYPE Order SYSTEM "C:¥dxx_xml¥test¥dtd¥LineItem.dtd"</doctype>
<root_node>
<element_node name="ABC.com">
<TDB_node>
<table name="dept" key="deptno"/>
<table name="empl" key="emplno"/>
<condition>dept deptno=empl.deptno</condition>
</RDB_node>

<element_node name="Department" multi_occurrence="YES">
<text_node>
<RDB_node>
<table name="dept"/>
<column name="deptno">
<condition>deptno=10</condition><RDB_node></RDB_node><text_node></text_node>
<element_node name="Employees" multi_occurrence="YES">

<text_node>

<RDB_node>

<table name="dept"><column name="deptnot"><condition>deptno=10</condition>
</table></RDB_node></text_node>
<element_node name="Employees" multi_occurence="YES">

<element_node name="EmployeeNo">

<text_node>

<RDB_node>
```

```
<table name="empl"><column name="emplno"><condition>emplno<500</condition>
</table></RDB_node></text_node></element_node>
<element_node name="Salary">

<text_node>

<RDB_node>

<table name="empl"><column name="salary"><condition>salary>5000.00</condition>
</table></RDB_node></text_node></element_node></element_node></element_node>
```

オーバーライドせずに XML 文書を構成するには、tests2x mydb test.dad result_tab を入力するか、またはオーバーライドを設定せずに dxxGenXML を呼び出します。これにより、以下に示すような文書が生成されます。

```
<?xml version="1.0">
<!DOCTYPE Order SYSTEM "C:¥dxx_xml¥test¥dtd¥LineItem.dtd">
<ABC.com>
<Department>10
<Employees>
<EmployeeNo>123</EmployeeNO>
<Salary>98,000.00</Salary>
</Employees>
<Employees>
<EmployeeNo>456</EmployeeNo>
<Salary>87,000.00</Salary>
</Employees>
</Department>
</ABC.COM>
```

DAD ファイルをオーバーライドするには、前述の方法で dxxGenXML を呼び出すか、または指定した条件で test2x プログラムを実行します。

tests2x mydb test.dad result_tab -o 2 ″/ABC.com/Department>10 AND /ABC.com/Department<30″

```
<?xml version="1.0">
<!DOCTYPE Order SYSTEM "C:¥dxx_xml¥test¥dtd¥LineItem.dtd">
<ABC.com>
<Department>20
<Employees>
<EmployeeNo>111</EmployeeNo>
<Salary>65,000.00</Salary>
</Employees>
<EmployeeNo>222</EmployeeNo>
<Salary>71,000.00</Salary>
</Employees>
<Employees>
<EmployeeNo>333</EmployeeNo>
<Salary>66,000.00</Salary>
</Employees>
</Department>
</ABC.com>
```

関連概念:

- 177 ページの『XML コレクションの DAD ファイル』
- 193 ページの『DAD チェッカー』

関連タスク:

- 175 ページの『XML 列のための DAD ファイルの作成』
- 193 ページの『DAD チェック・プログラムの使用』

関連資料:

- 182 ページの『DAD ファイル用の DTD』

DAD チェッカー

DAD チェック・プログラムは、XML コレクション保管方式を使用する DAD ファイルの妥当性をチェックするために使用することができます。各 DAD ファイルでは、XML 文書の構造と表との間の関係を指定するマッピング体系が指定されます。

XML 文書の構文の妥当性検査に文書タイプ記述 (DTD) が使用されるのと同様に、DAD チェック・プログラムは、DAD ファイルの意味構造が正しいことを確認するために使用されます。この妥当性検査は、データベースに接続しないで行うことができます。 DAD チェック・プログラムは、処理のためにファイルを XML Extender にサブミットする際に発生するエラー数を最小にするのに役立ちます。DAD チェック・プログラムは、コマンド行から呼び出される Java™ アプリケーションです。このプログラムは、呼び出されると、エラー、警告、および成功の標識を含む 2 つの出力ファイルをセットで作成します。この 2 ファイルの内容は同じです。1 つはエラーや警告を調べるためにユーザーが使用するプレーン・テキスト・ファイル、もう 1 つは XML ファイルの `errorsOutput.xml` で、これは、DAD チェック・プログラムを適用した結果を他のアプリケーションに通知するものです。出力テキスト・ファイルの名前は、ユーザーが定義します。名前を指定しないと、標準出力が使用されます。

関連概念:

- 177 ページの『XML コレクションの DAD ファイル』

関連タスク:

- 187 ページの『DAD ファイルの値の動的オーバーライド』
- 175 ページの『XML 列のための DAD ファイルの作成』
- 193 ページの『DAD チェック・プログラムの使用』

DAD チェック・プログラムの使用

前提条件:

JRE または SDK のバージョン 1.3.1 以降がシステムにインストールされている必要があります。

手順:

DAD チェック・プログラムを使用するための手順は、次のとおりです。

1. DADChecker.zip ファイルをダウンロードし、すべてのファイルを自分で選択したディレクトリーに抽出する。
2. コマンド行から、DAD チェック・プログラムをインストールしたディレクトリー内の `/bin` サブディレクトリーに変更する。
3. `/bin` ディレクトリーにある setCP.bat ファイルを実行してクラスパスを設定する。

4. 以下のコマンドを実行します。

```
java dadchecker.Check_dad_xml [-dad | -xml] [-all][-tag tagname]
[-out outputFile] fileToCheck
```

ここで、

-dad
チェックするべきファイルが DAD ファイルであることを示します。これはデフォルト・オプションです。

-xml
チェックするべきファイルが DAD ファイルではなくて、XML 文書であることを示します。大きな XML 文書の場合は、Java 仮想マシンがメモリー不足となり、 java.lang.OutOfMemoryError 例外を生成することがあります。このような場合は、-Xmx オプションを使用して、もっと大きなメモリーを Java 仮想計算機に割り振ることができます。詳しくは SDK の資料を参照してください。

-all
エラーになったタグの出現を出力ですべて表示することを示します。

-tag
名前属性値が *tagname* である重複タグだけを表示するように指示します。XML 文書の場合は、名前が tagname である重複タグだけが表示されます。

-out
outputFile は、出力テキストのファイル名を指定します。省略すると、標準出力が使用されます。 2 番目の出力ファイルである `errorsOutput.xml` も、DAD ファイルと同じディレクトリーに作成されます。このファイルは、常に生成され、出力テキストと同じ情報を XML 形式で含んでいますが、ただし、構文解析プログラムの警告とエラーは含みません。

コマンド行オプションを表示するには、`java dadchecker.Check_dad_xml help` と入力します。

バージョン情報を表示するには、`java dadchecker.Check_dad_xml version` と入力します。

DAD チェック・プログラムのサンプル・ファイル:

`samples` ディレクトリーには、以下のサンプル・ファイルが入っています。

bad_dad.dad
起こりえるすべての意味エラーを示しているサンプル DAD ファイル

bad_dad.chk
bad_dad.dad のために DAD チェック・プログラムが生成した出力テキスト・ファイル

bad_dad.chk
bad_dad.dad のために DAD チェック・プログラムが生成した出力テキスト・ファイル

errorsOutput.xml
bad_dad.dad のために DAD チェック・プログラムが生成した出力 XML ファイル

dup.xsl

errorsOutput.xml ファイルを、重複タグだけを示す HTML ファイルにトランスフォームするために使用される XSL スタイルシート

dups.html

bad_dad.dad に含まれる重複タグだけを示す、生成された HTML ファイル

出力テキスト・ファイル内のエラーと警告:

エラーと警告はタグの出現として示されます。次の場合の 2 つのタグは、同じタグの出現であると考えられます。

- 名前属性が同じ値である
- 同じ数の上位タグがある
- 対応する上位タグの名前属性が同じ値である

同じタグの出現でも、異なる子タグを持つ可能性があります。

DAD の意味構造規則に準拠しないタグの出現は、出力テキスト・ファイルに以下のように示されます。

- すべての上位タグとその属性が順番に表示される。
- エラーのあるタグは、XML ツリーでの深さを示す番号を前に付けて表示される。タグ名の後に、このタグのすべてが出現する場所を指す、DAD ファイルにおける行番号のリストが続きます。 *-all* コマンド行オプションを使用すると、各エラーの出現を別個に表示できます。
- 最初のタグの出現の直接子タグが表示される。データ・マッピングを指定するこれらの子タグでは、データ・マッピング・タグも表示されます。 *-all* コマンド行オプションを使用すると、各エラーの出現を別個に表示できます。

DAD チェック・プログラムのエラー・レポートの例:

この例では、名前属性が値 ″Password″ を持っている element_node タグがエラーです。 DAD ファイル内でのこのタグの出現は 2 回で、1 つは 49 行目、もう 1 つは 75 行目です。エラーのタグは、タグの深さ標識 (この例では 4) を置くことによって、上位タグと子タグのリストから分離することができます。上位タグと子タグのリストは、エラーが発生したコンテキストを確認するのに役立ちます。

```
<DAD>
 <Xcollection>
  <root_node>
   <element_node name="Advertiser" multi_occurrence="YES">
4   <element_node name="Password"> line(s): 49 75
     <element_node name="Pswd1">
     <element_node name="Pswd2">
```

all のオプションを使用した場合は、出力テキスト・ファイルは次のようになります。

```
<DAD>
 <Xcollection>
  <root_node>
   <element_node name="Advertiser" multi_occurrence="YES">
4   <element_node name="Password">  line: 49
     <element_node name="Pswd1">
      <element_node name="Pswd2">
```

```
<DAD>
 <Xcollection>
  <root_node>
   <element_node name="Advertiser" multi_occurrence="YES">
4   <element_node name="Password">  line: 75
    <element_node name="Pswd1">
    <element_node name="Pswd3">
```

この例では、2 つの出現は同じ上位タグ、同じ名前属性値を持っていますが、子エレメントは異なっています。

DAD チェック・プログラムが行うチェック

DAD チェック・プログラムを呼び出すと、以下のメッセージが出ます。

```
Checking DAD document: file_path
```

ここで *file_path* は、妥当性検査を行う DAD ファイルを指すパスです。

DAD チェック・プログラムは、以下について妥当性検査を行います。

1. 形式適合検査と DTD 妥当性検査
2. 重複 <attribute_node> と、リーフ <element_node> の検出 (RDB_node マッピング)
3. タイプ属性欠落の検出
4. 表宣言欠落の検出
5. <text_node> または <attribute_node> 欠落の検出
6. <attribute_node> と <element_node> のマッピング順序チェック
7. 同一の名前属性値を持つタグに対するデータ・マッピングの整合性検査
8. マップされた子 (RDB_node マッピング) を持つ親 <element_node> に対する multi_occurrence 属性値のチェック
9. 属性とエレメントにおける、起こりえる命名競合チェック (XML 文書)

これらの妥当性検査について、以下のセクションで説明します。

形式適合検査と DTD 妥当性検査

DAD ファイルを DAD DTD に対して妥当性検査する必要があります。DAD DTD は ″c:¥*dxx_install*samples¥db2xml¥dtd¥dad.dtd″ にあります。DAD ファイルが正しく形成されていないか、DTD が見つからないと致命的エラーとなって DAD チェック・プログラムは終了し、出力テキスト・ファイルにそのことが示されます。たとえば次のようなメッセージです。

```
org.xml.sax.SAXException: Stopping after fatal error,
line 1, col 22. The XML declaration must end with "?>".
```

妥当性検査のエラーと警告も、出力テキスト・ファイルに報告されますが、エラーや警告が DAD チェック・プログラムを終了させることはありません。以下の例は、出力テキスト・ファイルの断片で、DAD ファイルの構文解析時に検出される可能性のある 2 つの妥当性検査エラーを示しています。

```
 ** The document is not valid against the DTD, line 5, col 15. Element type
     "XCollection" must be declared

** The document is not valid against the DTD, line 578, col 21. The content of
      element type "text_node" must match "(column|RDB_node)".
```

重複 <attribute_node> と、リーフ <element_node> の検出 (RDB_node マッピング)

このチェックは、RDB_node マッピングを使用する DAD ファイルにだけ関係します。

複数の <attribute_node> または <element_node> タグが同じネーム属性を持ち上位が同じ場合は、2 つのエレメントは重複と見なされます。

複数のタグは、その対応する上位タグの名前属性が同じ値であれば、同じ上位タグを持っていると考えられます。

リーフ <element_node> は、XML 文書ツリーで子タグを持たないタグをマップするために使用されます。したがって、リーフ <element_node> タグは、その直接の子の 1 つとしてテキスト・ノードを 1 つ持つ必要があります。その他の <element_node> タグは、直接の子タグとしてテキスト・ノード・タグを持つことはできません。

このような競合は、複数のリーフ <element_ node> タグ間で、または、複数の <attribute_node> タグ間で、あるいは <element_ node> タグと <attribute_ node> タグの間で発生することがあります。

例:

例 1:

リーフ <element_node> の競合:

```
<element_node name = "A1">
  <element_node name = "B">
    <element_node name = "C">
       <text_node
          ....
<element_node name = "A2">
  <element_node name = "B">
   <element_node name = "C">
      <text_node>
       ....
</element_node>
```

この例では、<element_node name = ″C″> が重複しています。これは 2 つの異なるパス ¥A1¥B¥C と ¥A2¥B¥C によってマップされるからです。 <element_node name=″B″> は、リーフ <element_node> ではないので、重複であるとはとられません。

例 2:

この例は、<attribute node> 競合の例を示しています。

```
<element_node name = "A1">
    <attribute_node name = "B">
                  ....
<element_node name = "A2">
    <attribute_node name = "B">
 /element_node>               ....
<
```

この例では、<attribute_node name = ″B″> が重複しています。これは、2 つの異なるパス ¥A1¥B と ¥A2¥B によってマップされるからです。

例 3:

次の例は、リーフ <element_node> と <attribute_node> の競合を示しています。

```
<element_node name = "A">
    <element_node name = "B">
         <text_node>
           ....
     </element_node>
</element_node>
           ....
<attribute_node name = "B">
           ....
<attribute_node name = "A">
           ....
```

この例では、<element_node name = ″B″> が <attribute_node name = ″B″> と競合しています。 <element_node name = ″A″> と <attribute_node name = ″A″> は、競合しません。 <element_node name = ″A″> は、リーフ <element_node> ではないからです。

競合がある場合は、その競合を解消するように XML 文書 DTD を訂正する必要があります。 XML 文書と DAD ファイルも、その DTD 変更を反映して訂正する必要があります。

例 4:

```
7 duplicate naming conflicts were found
A total of 16 tags are in error (cumulate occurrences of these tags: 20)

The following tags are duplicates:

<DAD>
  <Xcollection>
   <root_node>
     <element_node name="Advertiser" multi_occurrence="YES">
 4       <element_node name="Country">  line(s): 127 135
        <text_node>
        <RDB_node>
         <table name="advertiser">
         <column type="VARCHAR(63)" name="country">

------------
<DAD>
<Xcollection>
  <root_node>
    <element_node name="Advertiser" multi_occurrence="YES">
     <element_node name="Campaign" multi_occurrence="YES">
      <element_node name="Target" multi_occurrence="YES">
       <element_node name="Location" multi_occurrence="YES">
 7         <element_node name="Country">  line(s): 460
            <text_node>
             <RDB_node>
               <table name="target_location">
               <column type="VARCHAR(63)" name="country">
------------------------------------------------------------
```

エラーがあるタグは、命名の競合ごとにグループ化されます。グループは線によって分離され、タグは短い線によって分離されています。また、*all* コマンド行オプションを使用して、すべてのエラーの出現を表示することもできます。

DAD ファイルに重複がなければ、以下のメッセージが出力テキスト・ファイルに書き出されます。

```
No duplicated tags were found.
```

タイプ属性欠落の検出

コレクションを使用可能にするため、または分解するために DAD ファイルを使用するときは、タイプ属性を各 <column> タグに指定する必要があります。たとえば次のようにします。

```
<column name="email" type="varchar(20)">
```

表がない場合、enable_collection コマンドは、列のタイプ指定を使用してコレクションに表を作成します。表がある場合は、DAD に指定されたタイプは、データベースの実際の列のタイプに合致する必要があります。

例:

以下の例は、出力テキスト・ファイルの断片で、タイプ属性を持たない <column> タグを示しています。

```
If this DAD is to be used for decomposition or for enabling a collection,
the type attributes are missing for the following <column> tag(s):
<DAD>
  <Xcollection>
   <root_node>
     <element_node name="Advertiser" multi_occurrence="YES">
      <element_node name="Address">
       <text_node>
         <RDB_node>
7           <column name="address">  line: 86
```

タイプ属性の欠落がない場合は、次のメッセージが出力テキスト・ファイルに書き込まれます。

```
No type attributes are missing for <column> tags.
```

表宣言欠落の検出

DAD ファイル内の最初の <RDB_node> タグは、データ・マッピングに使用されるリレーショナル表を宣言するすべての <table> タグを含め、表宣言を囲む必要があります。このタグは、最初の <element_node> タグに囲まれる必要があります。それ以降のすべての <RDB_node> タグは、<text_node> タグで囲まれる必要があります。

最初に出会う <RDB_node> タグに <column> タグが入っている場合も、出力ファイルにエラーが加えられます。このエラーは、表宣言が欠落しているか、または表宣言に誤って <column> タグが入っていることを示しています。

<text_node> または <attribute_node> 欠落の検出

<RDB_node> タグは、表宣言に使用される最初のもの以外は、<attribute_node> または <text_node> タグで囲む必要があります。

例:

例 1:

```
<element_node name ="amount">
<text_node>
<RDB_node>
```

```
<table name="fakebank.payments"/>
<column name="amount" type="decimal(8,2)"/>
</RDB_node>
</element_node>
```

例 2:

次の例は、出力テキスト・ファイルの断片で、<text_node> または <attribute_node> タグが欠落していることを示しています。

```
<DAD>
  <Xcollection>
   <root_node>
     <element_node name="Advertiser" multi_occurrence="YES">
      <element_node name="PostalCode">
5         <RDB_node>  line: 107
           <table name="advertiser">
            <column type="VARCHAR(10)" name="postal_code">
```

<attribute_node> と <element_node> のマッピング順序に関するチェック

このチェックは、FixPak 3 およびそれ以前のものには必要です。 <attribute_node> タグは、どの <element_node> タグが表にマップされるよりも前に、表にマップされる必要があります。

例:

次の例は、表にマップする必要があるタグを示しています。

```
<element_node name="payment-request"
multi_occurrence="YES">
     <element_node name="payment-request-id">
        <text_node>
           <RDB_node>
             <table name="fakebank.payments"/>
             <column name="statement_id" type="varchar(30)"/>
                  ....
      <element_node name="bank-customer-info">
        <element_node name="account">
          <attribute_node name="type">
            <text_node>
              <RDB_node>
                 <table name="fakebank.payments"/>
                 <column name="payor_account" type="char(6)"/
```

この例では、<attribute_node name=″type″> は <element_node name = ″payment-request-id″> と同じ表 (fakebank.payments) にマップされます。 <attribute_node> のマッピングは、<element_node> のマッピングより前に行われる必要があります。

同一の名前属性値を持つタグに対するデータ・マッピングの整合性検査

DAD ファイル内では、マップされ、ユニークな名前属性値で識別されるすべての <element_node> タグとすべての <attribute_node> タグは、一度だけマップされるべきです。 <element_node> タグまたは <attribute_node> タグの複数の出現が、異なる列にマップされる場合は、その名前属性には異なる値が割り当てられる必要があります。

例:

例 1: この例では、<element_node name="type"> タグの 2 回目の出現で、最初の出現とは異なるマッピングが行われています。重複 <attribute_node> タグと重複リーフ <element_node> タグは、このチェックの結果としては表示されません。

```
<element_node name="bank-customer-info">
   <element_node name="account">
      <element_node name="type">
         <text_node>
            <RDB_node>
              <table name="fakebank.payments"/>
              <column name="payor_account" type="char(20)
            </RDB_node>
         </text_node>
      </element_node>
      <element_node>
<element_node>
<element_node name="bank-customer-info">
    <element_node name="account">
      <element_node name="type">
         <text_node>
           <RDB_node>
              <table name="fakebank.payments"/>
              <column name="payto_account" type="char(20)"/>
           </RDB_node>
         </text_node>
       </element_node>
</element_node>
<element_node>
```

このエラーは、2 回目のマッピングに使用する新しいエレメントを作成することで修正できます。また、DTD、XML 文書、および DAD ファイルも変更する必要があります。

例 2: 次は、名前と上位タグが同じだが、マッピングが異なる <element_node> タグを指示する、出力テキスト・ファイル例の断片です。

```
<DAD>
  <Xcollection>
   <root_node>
    <element_node name="Advertiser" multi_occurrence="YES">
4      <element_node name="PostalCode">  line(s): 127
       <text_node>
       <RDB_node>
       <table name="advertiser">
        <column type="VARCHAR(10)" name="postal_code">
----------------
<DAD>
  <Xcollection>
    <root_node>
      <element_node name="Advertiser" multi_occurrence="YES">
4         <element_node name="PostalCode">  line(s): 135 143
          <text_node>
           <RDB_node>
             <table name="advertiser">
              <column type="VARCHAR(10)" name="postal_code2">
```

この例では、127 行目の <element_node name="PostalCode"> は、'postal_code' 列にマップされ、135 行と 143 行にある同じタグの他の 2 つの出現は、'postal_code2' 列にマップされています。

マップされた子を持つ親 <element_node> の multi_occurrence 属性値のチェック

このチェックは、RDB_node マッピングを使用する DAD ファイルにだけ関係します。

multi_occurrence 属性のデフォルト値は NO です。multi_occurrence 属性には、各 <element_node> が直接の子を持つ場合、または、複数の <element_node> タグが以下の 1 つ以上の基準に合致する場合は、値 YES を指定してください。

- <element_node> がマップされる (直接の子として <text_node> を持つ)。
- <element_node> が少なくとも 1 つの <attribute_node> を直接の子として持つ。

例:

例 1: 次の例では、payment-request-id と amount が DB2 UDB 表にマップされます。 sender は、直接の子として <attribute_node> を持っています。 payment-request-id、amount および sender は、すべて payment-request の直接の子です。

```
<element_node name="payment-request" multi_occurrence="YES">
  <element_node name="payment-request-id">
     <text_node>
        <RDB_node>
           <table name="fakebank.payments"/>
            <column name="statement_id" type="varchar(30)"/>
        </RDB_node>
      </text_node>
  </element_node>
 <element_node name ="amount">
      <text_node>
        <RDB_node>
          <table name="fakebank.payments"/>
           <column name="amount" type="decimal(8,2)"/>
        </RDB_node>
      </text_node>
  </element_node>
  <element_node name ="sender">
     <attribute_node name ="ID">
         <RDB_node>
            <table name="fakebank.payments"/>
            <column name="sender_ID" type="decimal(8,2)"/>
          </RDB_node>
      </attribute_node>
  </element_node>
  </element_node>
```

DAD チェック・プログラムは、multi_occurrence 属性が NO に設定されているすべての <element_node> タグを指摘します。

例 2: 次の例は、出力テキスト・ファイルの断片で、multi_occurrence を YES に設定する必要のある <element_node> 属性を示しています。

```
<DAD>
 <Xcollection>
   <root_node>
     <element_node name="Advertiser" multi_occurrence="YES">
4       <element_node name="Password">  line(s): 49 75
        <element_node name="Pswd1">
        <element_node name="Pswd2">
```

属性とエレメントの命名の競合

XML 文書では、異なる上位エレメントを持つなど、コンテキストが異なれば、同じ名前をもつエレメントが現れることができます。属性とエレメントは、同じ名前を持つことができます。

DAD チェック・プログラムは、XML 文書の命名上の競合をチェックするために使用できます。複数の競合するエレメントまたは属性がマップされる必要がある場合は、文書および DTD でも名前変更を行ってください。

XML 文書のチェックは、DAD ファイルを作成する前に行うのがベストです。DAD チェック・プログラムは、DTD に基づく XML 文書の妥当性検査はしません。

例:

次の例は、命名上の競合が起きている XML 文書の断片です。

```
<A1>
  <B>
     <C>
          ....
<A2>
   <B>
      <C>
          ....
<D C="attValue">
.....

If the <C> element and the C attribute are to be mapped, then the resulting
DAD file would have the following duplicate conflicts:
<element_node name = "A1">
    <element_node name = "B">
       <element_node name = "C">
          <text_node>
            .....
<element_node name = "A2">
    <element_node name = "B">
       <element_node name = "C">
          <text_node>
            ....
      <element_node name = "D">
         <attribute_node name = "C">
          ....
</element_node>
```

2 つの <element_node name = ″C″> タグと <attribute_node name = ″C″> タグが、DAD で重複しています。

第 10 章 XML Extender ストアード・プロシージャー

XML Extender ストアード・プロシージャー - 概要

XML Extender 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 管理ガイド* 」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、スキーマ DB2XML を使用します。これは XML Extender のスキーマ名です。

XML Extender 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。

管理ストアード・プロシージャー
管理用タスクの実行を支援します。

合成ストアード・プロシージャー
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。

分解ストアード・プロシージャー
受け取った XML 文書を分解または細分化して、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・プロシージャーを呼び出すプログラムには、必ず XML Extender 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 *"$dxx_install$*¥dxx¥samples¥db2xml¥include" ディレクトリーにあります。ここで *$dxx_install$* は DB2 XML Extender をインストールしたディレクトリーです。ヘッダー・ファイルは次のとおりです。

dxx.h XML Extender の定義する定数およびデータのタイプ

dxxrc.h XML Extender 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 Extender ストアード・プロシージャーの呼び出し

ストアード・プロシージャー名を英大文字と小文字の両方で書くことにより、単一のクライアント・アプリケーションから、いろいろなオペレーティング・システムで XML Extender を使用できるようになりました。この方法でストアード・プロシージャーを呼び出すには、合成ストアード・プロシージャーの result_colname および valid_colname のバージョンを使用します。この方法により、以下の利点が得られます。

- 結果表に多くの列を組み込めるため、すべての DB2 Universal Database 環境で、これらのストアード・プロシージャーを使用できます。 result_colname および valid_colname をサポートしないストアード・プロシージャーのバージョンでは、結果表の列は 1 つだけでなければなりません。
- 宣言した一時表を結果表として使用できます。一時表は、「session」と設定されたスキーマで示されます。宣言した一時表を使用すると、マルチユーザー・クライアント環境をサポートできます。

異なるプラットフォームで確実にストアード・プロシージャーにアクセスするには、 DB2 XML Extender ストアード・プロシージャーを呼び出すときに、英大文字を使用します。

前提条件: データベースを、XML Extender のストアード・プロシージャーおよび DB2 UDB 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、″c:¥*dxx_install*¥samples¥db2xml¥cmd″ ディレクトリーにあります。バインドを行うには、以下のようにします。

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. ディレクトリーを ″c:¥*dxx_install*¥samples¥db2xml¥bnd″ に変え、XML Extender をデータベースにバインドします。

   ```
   db2 "bind @dxxbind.lst"
   ```

3. ディレクトリーを ″c:¥*dxx_install*¥samples¥db2xml¥bnd″ に変え、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"
   ```

手順:

以下の構文を使って XML Extender を呼び出します。

```
CALL DB2XML.function_entry_point
```

ここで、

function_entry_point
関数の名前を指定します。

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 NULL 標識を指定できます。

dxx_install/samples/db2xml/c と dxx_install/samples/db2xml/cli ディレクトリーに入っている、ストアード・プロシージャーを呼び出すサンプルを参照してください。 dxx_install/samples/db2xml/c ディレクトリーの中には、組み込み SQL を使用して XML コレクション・ストアード・プロシージャーを呼び出すための SQX コード・ファイルが提供されています。 dxx_install/samples/db2xml/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ています。

ストアード・プロシージャー用 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。この制限は増やすことができます。

手順:

CLOB の制限を増やすためには、次のようにします。

1. 各ストアード・プロシージャーをドロップします。たとえば次のようにします。

```
db2   "drop procedure DB2XML.dxxShredXML restrict"
```

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 CLOB 制限を 10 MB に増やすには、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_STP'
        language C
        parameter style SQL
        not deterministic
        fenced
        null call"
```

関連タスク:

- 206 ページの『XML Extender ストアード・プロシージャーの呼び出し』
- 207 ページの『CLOB を戻すストアード・プロシージャー』

CLOB を戻すストアード・プロシージャー

CLOB ファイルが 1 MB より大きい場合は、以下のコマンドを含むファイルを作成および実行することにより、ストアード・プロシージャー・パラメーターを再定義します。

```
drop procedure db2xml.dxxGenXMLClob;

create procedure db2xml.dxxGenXMLClob(
          in     dadBuf        clob(100K),
          in     overrideType  integer,
          in     override      varchar(32672),
          out    resultDoc     clob(1M),
          out    valid         integer,
          out    numDocs       integer,
          out    returnCode    integer,
          out    returnMsg     varchar(1024)
          )
```

```
          external name 'db2xml!dxxGenXMLClob'
          specific DB2XML.DXXGENXMLCLOB
          language C
          parameter style DB2DARI
          not deterministic
          fenced
          null call;

drop procedure db2xml.dxxRetrieveXMLClob;

create procedure db2xml.dxxRetrieveXMLClob(
          in   collectionName    varchar(128),
          in   overrideType      integer,
          in   override          varchar(32672),
          out  resultDoc         clob(1M),
          out  valid             integer,
          out  numDocs           integer,
          out  returnCode        integer,
          out  returnMsg         varchar(1024)
          )
          external name 'db2xml!dxxRetrieveXMLClob'
          specific DB2XML.DXXRETRIEVEXMLCLOB
          language C
          parameter style DB2DARI
          not deterministic
          fenced
```

CLOB の長さを指定するには: エディターでファイルを開き、以下の例に示す、*resultDoc* パラメーターを変更します。

```
out resultDoc clob(clob_size),
```

複数の文書が生成された場合、ストアード・プロシージャーは最初の文書を戻します。

推奨サイズ: *resultDoc* パラメーターのサイズの制限は、システム・セットアップによって異なります。文書サイズに関係なく、このパラメーターで指定したサイズが JDBC によって割り振られるサイズになることに注意してください。サイズは、最大の XML ファイルを収容できるものでなければなりませんが、1.5 ギガバイトを超えることはできません。

コマンド・ファイルを実行するには、ファイルが置かれている DB2 コマンド行およびディレクトリーから、以下のように入力します。

```
db2 -tf crtgenxc.db2
```

関連タスク:

• 206 ページの『XML Extender ストアード・プロシージャーの呼び出し』

XML Extender 管理ストアード・プロシージャー

XML Extender 管理ストアード・プロシージャー - 概要

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 Extender 管理ウィザードおよび管理コマンド **dxxadm** を使用します。

- dxxEnableDB()

- dxxDisableDB()
- dxxEnableColumn()
- dxxDisableColumn()
- dxxEnableCollection()
- dxxDisableCollection()

dxxEnableDB() ストアード・プロシージャー

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 Extender は以下のオブジェクトを作成します。

- XML Extender ユーザー定義タイプ (UDT)
- XML Extender ユーザー定義関数 (UDF)
- XML Extender DTD リポジトリー表 (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。
- XML Extender 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。

構文:

```
DB2XML.dxxEnableDB(char(dbName) dbName,       /* input */
            long           returnCode,   /* output */
            varchar(1024)  returnMsg)    /* output */
```

パラメーター:

表 47. 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

関連概念:

- 208 ページの『XML Extender 管理ストアード・プロシージャー - 概要』
- 291 ページの『第 13 章 XML Extender の管理サポート表』

関連タスク:

- 58 ページの『XML 用のデータベースの使用可能化』
- 206 ページの『XML Extender ストアード・プロシージャーの呼び出し』

関連資料:

- ix ページの『構文図の読み方』
- 335 ページの『付録 C. XML Extender の制限』

dxxDisableDB() ストアード・プロシージャー

目的:

データベースを使用不可にします。 XML Extender 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ドロップします。

- XML Extender ユーザー定義タイプ (UDT)
- XML Extender ユーザー定義関数 (UDF)
- XML Extender DTD リポジトリー表 (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。
- XML Extender 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。

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 Extender 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。

構文:

```
DB2XML.dxxDisableDB(char(dbName)     dbName,       /* input */
                    long          returnCode,   /* output */
                    varchar(1024) returnMsg)    /* output */
```

パラメーター:

表 48. 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

関連概念:

- 208 ページの『XML Extender 管理ストアード・プロシージャー - 概要』
- 291 ページの『第 13 章 XML Extender の管理サポート表』

関連タスク:

- 206 ページの『XML Extender ストアード・プロシージャーの呼び出し』

関連資料:

- ix ページの『構文図の読み方』
- 335 ページの『付録 C. XML Extender の制限』

dxxEnableColumn() ストアード・プロシージャー

目的:

XML 列を使用可能にします。 XML Extender は、列を使用可能にする時に以下のタスクを行います。

- XML 表に主キーがあるかどうかを判別します。ない場合は、XML Extender が XML 表を変更して 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 表内の各行のユニークな 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 Extender 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・ビューを作成します。オプションで、ユーザーが名前を指定できます。

構文:

```
DB2XML.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 49. 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・ビューの名前	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 つの主キーの名前	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

関連概念:

- 80 ページの『保管およびアクセス方式としての XML 列』
- 208 ページの『XML Extender 管理ストアード・プロシージャー - 概要』

関連タスク:

- 206 ページの『XML Extender ストアード・プロシージャーの呼び出し』

関連資料:

- ix ページの『構文図の読み方』

- 335 ページの『付録 C. XML Extender の制限』

dxxDisableColumn() ストアード・プロシージャー

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。

構文:

```
DB2XML.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 50. 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

関連資料:

- 335 ページの『付録 C. XML Extender の制限』

dxxEnableCollection() ストアード・プロシージャー

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。

構文:

```
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 51. 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

表 51. *dxxEnableCollection()* パラメーター *(続き)*

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
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

関連概念:

- 97 ページの『保管およびアクセス方式としての XML コレクション』
- 208 ページの『XML Extender 管理ストアード・プロシージャー - 概要』

関連タスク:

- 206 ページの『XML Extender ストアード・プロシージャーの呼び出し』

関連資料:

- ix ページの『構文図の読み方』
- 335 ページの『付録 C. XML Extender の制限』

dxxDisableCollection() ストアード・プロシージャー

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。

構文:

```
dxxDisableCollection(char(dbName) dbName,      /* input */
                     char(colName) colName,     /* input */
                     long          returnCode,  /* output */
                     varchar(1024)  returnMsg)   /* output */
```

パラメーター:

表 52. *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

関連資料:

• 335 ページの『付録 C. XML Extender の制限』

XML Extender 合成ストアード・プロシージャー

XML Extender 合成ストアード・プロシージャー - 概要

合成ストアード・プロシージャー
dxxGenXML()、dxxRetrieveXML()、dxxGenXMLCLOB()、および
dxxRetrieveXMLCLOB() 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。

合成ストアード・プロシージャーについて、以下のパフォーマンスの機能強化が行われました。

- UNIX® および Windows® オペレーティング・システムでは、オーバーライド・パラメーターの長さが、1KB から 32KB に伸ばされました。

  1KB のオーバーライド・パラメーターでは、 SQL 合成用の SQL ステートメントの長さは制限されていました。この制限によって、必要な SQL ステートメントの長さを短くするデータベース・ビューの使用が奨励されました。ただし、そのデータベース・ビューでは、ビューを実現するための追加のパス長が必要とされる場合がありました。長いオーバーライド・パラメーターを使用すれば、ビューに対する要求は大幅に減少します。
- 中間結果表が必要でなくなりました。
- このようなストアード・プロシージャーを使用することによって、以下の結果が得られます。
  - 結果表を作成する必要がないため、命令のパス長が短くなる。
  - プログラミングが単純になる。
- 複数の文書を作成したい場合は、中間結果表を必要とするストアード・プロシージャーを使用します。
- パフォーマンス向上のために、XML 列のユーザー定義関数が強化されました。
- DB2® XML Extender のユーザー定義関数は、小さな (512KB) XML 文書をメモリーに保持したままで、それらの文書を処理できるようになりました。このため、一時ファイルに使用されるディスクに関する競合、および入出力活動が減少しました。
- DB2 UDB XML Extender のスカラー (表でない) ユーザー定義関数を並列で実行できるように、それらの関数の定義が変更されました。この変更により、ユーザー定義関数を複数回参照する照会のパフォーマンスが大幅に向上します。スカラー UDF の並列機能を実現するには、マイグレーション・スクリプト・プログラムを実行する必要があります。すでにスカラー UDF を使用して列を使用可能にしている場合は、すべての列を使用不可にし、マイグレーション・スクリプトを実行してから、それらの列をもう一度使用可能にします。

dxxGenXML() ストアード・プロシージャー

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、試行処理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。

構文:

```
dxxGenXML(CLOB(100K)     DAD,       /* input */
          char(resultTabName) resultTabName, /* input */
          char resultColName, char resultValidCol /* input */

     char(30)       valid_column, /* input */
    integer        overrideType  /* input */
          varchar(1024)  override,     /* input */
      integer        maxRows,      /* input */
          integer        numRows,      /* output */
          long           returnCode,   /* output */
          varchar(1024)  returnMsg)    /* output */
```

ここで、varchar_value は Windows と UNIX では 32672、iSeries と z/OS では 16366 です。

パラメーター:

表 53. 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
result_column	合成された XML 文書が保管される結果表の列の名前	IN
valid_column	XML 文書が文書タイプ定義 (DTD) に対しての妥当性検査で有効であったかどうかを示す、列の名前	IN

表 53. *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**: NULL 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 つ以上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
resultDoc	合成 XML 文書を含む CLOB	OUT
valid	valid は、以下のように設定されます。 • VALIDATION=YES の場合、妥当性検査が正常に終了すれば valid=1、失敗すれば valid=0 • VALIDATION=NO の場合は、valid=NULL	OUT
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:

以下の断片的な例では、XML_ORDER_TAB という名前の結果表が作成されることと、表に XMLVARCHAR タイプの 1 つの列が含まれることを想定しています。完全な作業サンプルは、`DXXSAMPLES/QCSRC(GENX)` に入っています。

```
#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             verride[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 SQL CREATE TABLE xml_order_tab (xmlorder XMLVarchar);

             /* read data from a file to a CLOB */
             strcpy(dadfile.name,"dxxinstall/dad/litem3.dad");
             dadfile.name_length = strlen("dxxinstall/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 stored 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);
```

関連概念:

- 214 ページの『XML Extender 合成ストアード・プロシージャー - 概要』

関連タスク:

- 66 ページの『SQL マッピングを使った XML 文書の合成』
- 70 ページの『RDB_node マッピングを使った XML コレクションの合成』
- 206 ページの『XML Extender ストアード・プロシージャーの呼び出し』

関連資料:

- ix ページの『構文図の読み方』

dxxRetrieveXML() ストアード・プロシージャー

目的:

ストアード・プロシージャー dxxRetrieveXML() を使って、分解された XML 文書を検索できます。 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。

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。

構文:

```
dxxRetrieveXML(char(collectionName) collectionName,   /* input */
               char(resultTabName) resultTabName,    /* input */
               char resultColName, char resultValidCol   /* input */

               integer        overrideType,     /* input */
               varchar_value  override,         /* input */
          integer        maxRows,          /* input */
               integer        numRows,          /* output */
               long           returnCode,       /* output */
               varchar(1024)  returnMsg)        /* output */
```

ここで、*varchar_value* は Windows と UNIX では 32672、iSeries と z/OS では 16366 です。

パラメーター:

表 54. 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
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
result_column	合成された XML 文書が保管される結果表の列の名前	IN

表 54. 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
valid_column	XML 文書が文書タイプ定義 (DTD) に対しての妥当性検査で有効であったかどうかを示す、列の名前	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
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**: NULL 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 つ以上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:

以下の断片は 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。完全な作業サンプルは、 dxx_install¥samples¥db2xml¥qcsrc(rtrx) に入っています。

```
#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);
```

関連概念:

- 214 ページの『XML Extender 合成ストアード・プロシージャー - 概要』

関連タスク:

- 66 ページの『SQL マッピングを使った XML 文書の合成』
- 70 ページの『RDB_node マッピングを使った XML コレクションの合成』
- 206 ページの『XML Extender ストアード・プロシージャーの呼び出し』

関連資料:

- ix ページの『構文図の読み方』
- 335 ページの『付録 C. XML Extender の制限』

dxxGenXMLClob ストアード・プロシージャー

目的:

dxxGenXMLClob は、DAD を含んでいるバッファーを入力として取ります。これは、DAD の <Xcollection> が指定している 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 文書を合成し、 *resultDoc* CLOB に生成された、最初の、通常は唯一の XML 文書を戻します。

構文:

```
dxxGenXMLClob(CLOB(100k)                DAD             /*input*/
              integer                   overrideType,    /*input*/
              varchar(varchar_value)    override,       /*input*/
              CLOB(1M)                  resultDoc,       /*output*/
              integer                   valid,           /*output*/
              integer                   numDocs,         /*output*/
              long                      returnCode,      /*output*/
              varchar(1024)             returnMsg),      /*output*/
```

ここで、*varchar_value* は Windows と UNIX では 32672、iSeries と z/OS では 16366 です。

パラメーター:

表 55. dxxGenXMLClob 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
overrideType	*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 55. *dxxGenXMLClob* 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** NULL 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 つ以上の式を含むストリング。式は二重引用符で囲み、and という語で分離します。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
resultDoc	合成 XML 文書を含む CLOB	OUT
valid	valid は、以下のように設定されます。 • VALIDATION=YES の場合、妥当性検査が正常に終了すれば valid=1、失敗すれば valid=0 • VALIDATION=NO の場合は、valid=NULL	OUT
numDocs	入力データから生成される XML 文書の数。 **注:** 現在では、最初の文書だけが戻されます。	OUT

表 55. *dxxGenXMLClob* 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

CLOB パラメーターのサイズは 1 MB です。CLOB ファイルが 1 MB より大きい場合は、XML Extender は、ストアード・プロシージャー・パラメーターを再定義するコマンド・ファイルを提供します。DB2 UDB XML Extender Web サイトから `crtgenxc.zip` ファイルをダウンロードしてください。この ZIP ファイルには、以下のプログラムが含まれています。

crtgenxc.db2
XML Extender V7.2 FixPak 5 以降 (UNIX および Windows 版) で使用します。

crtgenxc.iseries
XML Extender (iSeries 版) で使用します。

crtgenxc.zox.jci および crtgenxc.zos.cmd
XML Extender (OS/390 V7 版、APAR PQ58249 以降) で使用します。

CLOB の長さを指定するには: エディターでファイルを開き、以下の例に示す、*resultDoc* パラメーターを変更します。

```
out    resultDoc     clob(clob_size),
```

推奨サイズ: *resultDoc* パラメーターのサイズの制限は、システム・セットアップによって異なりますが、文書サイズに関係なく、このパラメーターで指定したサイズが JDBC によって割り振られるサイズになることに注意してください。サイズは、最大の XML ファイルを収容できるものでなければなりませんが、1.5 ギガバイトを超えることはできません。

UNIX または Windows でコマンド・ファイルを実行するには、ファイルが置かれている DB2 UDB コマンド行およびディレクトリーから、以下のように入力します。

```
db2   -tf  crtgenxc.db2
```

関連概念:

- 214 ページの『XML Extender 合成ストアード・プロシージャー - 概要』

関連タスク:

- 66 ページの『SQL マッピングを使った XML 文書の合成』
- 70 ページの『RDB_node マッピングを使った XML コレクションの合成』
- 206 ページの『XML Extender ストアード・プロシージャーの呼び出し』

関連資料:

- ix ページの『構文図の読み方』
- 335 ページの『付録 C. XML Extender の制限』

dxxRetrieveXMLClob ストアード・プロシージャー

目的:

dxxRetrieveXMLClob ストアード・プロシージャーは、リレーショナル・データからの文書合成を使用可能にします。

dxxRetrieveXMLClob を使用するための要件は、 dxxGenXMLClob を使用する場合の要件と同じです。唯一の違いとして、dxxRetrieveXMLClob 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。

構文:

```
dxxRetrieveXMLClob(char(collectionName)          collectionName  /*input*/
                integer                  overrideType,   /*input*/
                varchar(varchar_value)   override,       /*input*/
                CLOB(1M)                 resultDoc,      /*output*/
                integer                  valid,          /*output*/
                integer                  numDocs,        /*output*/
                long                     returnCode,     /*output*/
                varchar(1024)            returnMsg),     /*output*/
```

パラメーター:

表 56. dxxRetrieveXMLClob 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
overrideType	*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 56. *dxxRetrieveXMLClob* 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** NULL 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 つ以上の式を含むストリング。式は二重引用符で囲み、and という語で分離します。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
resultDoc	結果表の行の最大数	IN
valid	valid は、以下のように設定されます。 • VALIDATION=YES の場合、妥当性検査が正常に終了すれば valid=1、失敗すれば valid=0 • VALIDATION=NO の場合は、valid=NULL	OUT
numDocs	入力データから生成される XML 文書の数。注: 現在では、最初の文書だけが戻されます。	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

CLOB パラメーターのサイズは 1 MB です。CLOB ファイルが 1 MB より大きい場合は、XML Extender は、ストアード・プロシージャー・パラメーターを再定義するコマンド・ファイルを提供します。DB2 UDB XML Extender Web サイトから `crtgenxc.zip` ファイルをダウンロードしてください。この ZIP ファイルには、以下のプログラムが含まれています。

crtgenxc.db2
: XML Extender V7.2 Fixpak 5 以降 (UNIX および Windows 版) で使用します

crtgenxc.iseries
XML Extender (iSeries 版) で使用します

crtgenxc.zox.jci および crtgenxc.zos.cmd
XML Extender (OS/390 V7 版、APAR PQ58249 以降) で使用します。

CLOB の長さを指定するには: エディターでファイルを開き、以下の例に示す、*resultDoc* パラメーターを変更します。

```
out    resultDoc     clob(clob_size),
```

推奨サイズ: *resultDoc* パラメーターのサイズの制限は、システム・セットアップによって異なりますが、文書サイズに関係なく、このパラメーターで指定したサイズが JDBC によって割り振られるサイズになることに注意してください。サイズは、最大の XML ファイルを収容できるものでなければなりませんが、1.5 ギガバイトを超えることはできません。

UNIX または Windows でコマンド・ファイルを実行するには、ファイルが置かれている DB2 UDB コマンド行およびディレクトリーから、以下のように入力します。

```
db2   -tf  crtgenxc.db2
```

関連概念:

- 214 ページの『XML Extender 合成ストアード・プロシージャー - 概要』

関連タスク:

- 66 ページの『SQL マッピングを使った XML 文書の合成』
- 70 ページの『RDB_node マッピングを使った XML コレクションの合成』
- 206 ページの『XML Extender ストアード・プロシージャーの呼び出し』

関連資料:

- ix ページの『構文図の読み方』
- 335 ページの『付録 C. XML Extender の制限』

XML Extender 分解ストアード・プロシージャー

XML Extender 分解ストアード・プロシージャー - 概要

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() ストアード・プロシージャー

目的:

これは、DAD ファイル・マッピングに基づいて、XML 文書を分解し、その XML エレメントと属性の内容を指定の DB2 UDB 表に保管します。 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。このストアード・プロシージャーでは、DAD で結合条件に指定されている列が、既存の表の主キー/外部キーの関係 (リレーションシップ) に対応している必要があります。 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。

このセクションに示すストアード・プロシージャーの断片は、説明のためのサンプルです。

構文:

```
dxxShredXML(CLOB(100K)    DAD,           /* input */
            CLOB(1M)      xmlobj,        /* input */
            long          returnCode,    /* output */
            varchar(1024) returnMsg)     /* output */
```

パラメーター:

表 57. 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:

以下の断片は 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,"dxx_install
              /samples/db2xml/dad/getstart_xcollection.dad
              ");
```

```
dadFile.name_length=strlen("dxx_install
/samples/db2xml/dad/getstart_xcollection.dad
");
dadFile.file_option=SQL_FILE_READ;
strcpy(xmlFile.name,"dxx_install
  /samples/db2xml/xml/getstart.xml");
xmlFile.name_length=strlen("dxx_install
  /samples/db2xml/xml/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);
```

関連概念:

- 226 ページの『XML Extender 分解ストアード・プロシージャー - 概要』

関連タスク:

- 72 ページの『RDB_node マッピングを使った XML コレクションの分解』
- 102 ページの『XML 文書を分解して DB2 UDB データにする』
- 206 ページの『XML Extender ストアード・プロシージャーの呼び出し』

関連資料:

- ix ページの『構文図の読み方』
- 335 ページの『付録 C. XML Extender の制限』

dxxInsertXML() ストアード・プロシージャー

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。

構文:

```
dxxInsertXML(char(collectionName) collectionName, /*input*/
             CLOB(1M)       xmlobj,          /* input */
             long           returnCode,      /* output */
             varchar(1024)  returnMsg)       /* output */
```

パラメーター:

表 58. 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

表 58. *dxxInsertXML()* パラメーター *(続き)*

パラメーター	説明	IN/OUT パラメーター
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:

以下の断片的な例では、dxxInsertXML() 呼び出しによって入力 XML 文書 dxx_install/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,"dxx_install/samples
db2xml/xml/getstart.xml");
xmlobj.name_length=strlen("dxx_install/samples
db2xml/xml/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);
```

関連概念:

- 226 ページの『XML Extender 分解ストアード・プロシージャー - 概要』

関連タスク:

- 72 ページの『RDB_node マッピングを使った XML コレクションの分解』
- 102 ページの『XML 文書を分解して DB2 UDB データにする』
- 206 ページの『XML Extender ストアード・プロシージャーの呼び出し』

関連資料:

- ix ページの『構文図の読み方』
- 335 ページの『付録 C. XML Extender の制限』

第 11 章 MQSeries 用の XML Extender ストアード・プロシージャーと関数

MQSeries 用の XML Extender ストアード・プロシージャーと関数 - 概要

XML Extender には、XML データを保管し、アクセスするために 2 つの方式が用意されています。 XML 列方式を使用すると、文書内容を照会、更新、および検索しながら、XML 文書を DB2® 表に保管できます。 MQ XML ユーザー定義関数によって、XML 文書を照会し、その結果をメッセージ・キューにパブリッシュすることができます。一方、XML コレクション方式を使用すれば、XML 文書のタグを外した内容を 1 つ以上の表に保管したり、複数の表から XML 文書を合成したりできます。 MQ XML ストアード・プロシージャーを使用して、メッセージ・キューから XML 文書を検索し、それをタグなしデータに分解し、そのデータを DB2 表に保管することが可能です。また、DB2 UDB データから XML 文書を作成し、その文書を MQSeries® メッセージ・キューに送信することもできます。

MQSeries は、XML データと文書の配布に 3 つのメッセージング・モデルをサポートします。

データグラム
: メッセージは、応答はないものとして 1 つの宛先に送信されます。

Publish/Subscribe
: 1 つ以上のパブリッシャーがメッセージをパブリッシュ・サービスに送信します。このサービスが、そのメッセージを関係するサブスクライバーに配布します。

要求/応答
: メッセージは 1 つの宛先に送信され、送信側は応答が返されることを想定しています。

MQSeries は、多様な方法で使用できます。単純なデータグラムを交換することによって、複数のアプリケーションを調整し、情報を交換し、サービスを要求し、さらに関係あるイベントの通知を提供することができます。 Publish/subscribe は、リアルタイムの情報をタイムリーに配布するために、最も頻繁に使用されます。要求/応答スタイルは、通常は、単純形式の疑似同期リモート・プロシージャー・コールとして使用されます。もっと複雑なモデルを、これらの基本的なスタイルを結合して構成することもできます。

ここに説明した基本的なメッセージング手法は、広く多様な方法で使用されています。 MQSeries は、多種類のオペレーティング・システムで使用できるため、似ている環境からの、または異なる環境からのいろいろなアプリケーションをリンクする重要な手法を提供します。

MQXML 関数とストアード・プロシージャーを使用するには、次のソフトウェアをインストールしておく必要があります。

- DB2 Universal Database™ バージョン 7.2 以降
- DB2 MQSeries Functions バージョン 7.2 (DB2 Universal Database バージョン 7.2 のオプションとしてインストール可能です。インストール情報は、DB2 Universal Database バージョン 7.2 リリース情報に入っています。)
- パブリッシュ関数を使用する場合は、MQSeries Publish/Subscribe または MQSeries Integrator

XML Extender の MQSeries 関数

XML Extender MQSeries 関数 - 概要

DB2® XML Extender には MQSeries® で使用する以下の関数が含まれています。

- MQPublishXML
- MQReadXML
- MQReadAllXML
- MQReadXMLCLOB
- MQReadAllXMLCLOB
- MQReceiveXML
- MQReceiveAllXML
- MQRcvAllXMLCLOB
- MQReceiveXMLCLOB
- MQSENDXML
- MQSENDXMLFILE
- MQSendXMLFILECLOB

関連概念:

- 231 ページの『MQSeries 用の XML Extender ストアード・プロシージャーと関数 - 概要』
- 257 ページの『XML Extender MQSeries ストアード・プロシージャー - 概要』

関連資料:

- 235 ページの『MQReadXML 関数』
- 237 ページの『MQReadAllXML 関数』
- 243 ページの『MQReceiveXML 関数』
- 239 ページの『MQReadXMLCLOB 関数』
- 240 ページの『MQReadAllXMLCLOB 関数』
- 251 ページの『MQRcvXMLCLOB 関数』
- 249 ページの『MQReceiveXMLCLOB 関数』
- 233 ページの『MQPublishXML 関数』
- 244 ページの『MQReceiveAllXML 関数』
- 247 ページの『MQRcvAllXMLCLOB 関数』
- 256 ページの『MQSendXMLFILECLOB 関数』
- 252 ページの『MQSENDXML 関数』

• 254 ページの『MQSENDXMLFILE 関数』

MQPublishXML 関数

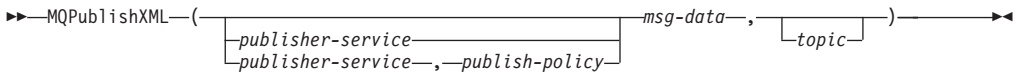
目的:

MQPublishXML 関数は、XMLVARCHAR データおよび XMLCLOB データを MQSeries にパブリッシュします。この関数を使用するには、MQSeries Publish/Subscribe または MQSeries Integrator のいずれかをインストールしておく必要があります。詳しくは、以下の Web サイトを参照してください。

```
http://www.software.ibm.com/MQSeries
```

MQPublishXML 関数は、*msg-data* に入っている XML データをパブリッシュ・ポリシーの品質 *publish-policy* を使用して *publisher-service* が指定する MQSeries パブリッシャーにパブリッシュします。オプションとして、このメッセージのトピックを *topic* で指定できます。オプションのユーザー定義メッセージ相関 ID は、*correl-id* によって指定することができます。関数は、正常に終了すれば 1 を戻します。

構文:



パラメーター:

表 59. MQPublishXML パラメーター

パラメーター	データ・タイプ	説明
publisher-service	VARCHAR(48)	論理 MQSeries 宛先を含むストリング。メッセージはここに送られます。 *publisher-service* を指定する場合は、それで AMT.XML リポジトリー・ファイルに定義されているパブリッシャー・サービス・ポイントを参照します。 *publisher-service* を指定しないと、DB2.DEFAULT.PUBLISHER が使用されます。 publisher-service の最大サイズは 48 バイトです。

表 59. *MQPublishXML* パラメーター *(続き)*

パラメーター	データ・タイプ	説明
publish-policy	VARCHAR(48)	このメッセージの処理に使用される MQSeries AMI *publish policy* を含んでいるストリング。 *publish-policy* を指定する場合は、それで AMT.XML リポジトリー・ファイルに定義されているポリシーを参照します。パブリッシュ・ポリシーは、また、メッセージング操作オプションに適用すべきパブリッシュ・オプションの品質のセットを定義します。これらのオプションには、メッセージ優先順位とメッセージ持続性が含まれます。 *service-policy* を指定しないと、デフォルトの DB2.DEFAULT.POLICY が使用されます。 service-policy の最大サイズは 48 バイトです。さらに詳しくは、「MQSeries Application Messaging Interface」を参照してください。
msg-data	XMLVARCHAR または XMLCLOB	MQSeries を介して送られるデータが入っている XMLVARCHAR または XMLCLOB 式。
topic	VARCHAR(40)	メッセージがパブリッシュされるときのトピックを含んでいるストリング。トピックを指定しないと、そのメッセージには、トピックは何も付きません。トピックの最大サイズは 40 バイトです。各トピックの間を ″:″ で離せば、トピック・ストリング内に複数のトピックをリストできます。

戻りコード:

正常に終了すると、MQPublishXML 関数は 1 を戻します。関数が正常に終了しなかった場合は、値 0 が戻されます。

関連概念:

- 231 ページの『MQSeries 用の XML Extender ストアード・プロシージャーと関数 - 概要』

関連資料:

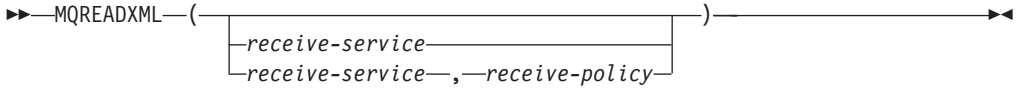
- ix ページの『構文図の読み方』

MQReadXML 関数

目的:

MQREADXML 関数は、*receive-service* で指定される MQSeries ロケーションから XMLVARCHAR データを戻します。これは、*receive-policy* の品質を使用します。MQREADXML 関数は、*receive-service* に関連付けられているキューからメッセージを除去することはありません。

構文:



パラメーター:

表 60. MQReadXML パラメーター

パラメーター	データ・タイプ	説明
receive-service	VARCHAR(48)	論理 MQSeries 宛先を含んでいるストリング。メッセージはここから受け取ります。*receive-service* を指定する場合は、それで AMT.XML リポジトリー・ファイルに定義されているサービス・ポイントを参照します。 *receive-service* を指定しないと、DB2.DEFAULT.SERVICE が使用されます。 *receive-service* の最大サイズは 48 バイトです。

表 60. *MQReadXML* パラメーター *(続き)*

パラメーター	データ・タイプ	説明
receive-policy	VARCHAR(48)	メッセージの処理に使用される、MQSeries AMI サービス・ポリシーを含んでいるストリング。 *receive-policy* を指定する場合は、それで AMT.XML リポジトリー・ファイルに定義されているポリシーを参照します。 receive-policy は、メッセージング操作に適用される受信オプションの品質のセットを定義します。これらのオプションには、メッセージ優先順位とメッセージ持続性が含まれます。 *receive-policy* を指定しないと、デフォルトの DB2.DEFAULT.POLICY が使用されます。 *receive-policy* の最大サイズは 48 バイトです。

結果:

キュー内のメッセージが正常に読めたときは、MQREADXML は db2xml.xmlvarchar を戻します。メッセージが使用可能でなければ、NULL が戻されます。

例:

例 1: この例は、デフォルト・サービス *DB2.DEFAULT.SERVICE* が指定しているキューの先頭にあるメッセージを読みます。メッセージを読むためには、デフォルト・ポリシー *DB2.DEFAULT.POLICY* が使用されます。

```
values DB2XML.MQREADXML()
```

この例は、メッセージの内容を XMLVARCHAR として戻します。メッセージが使用可能でなければ、NULL が戻されます。

例 2: この例は、デフォルト・ポリシー *DB2.DEFAULT.POLICY* を使用して、サービス *MYSERVICE* が指定するキューの先頭にあるメッセージを読みます。

```
values DB2XML.MQREADXML('MYSERVICE')
```

この例では、メッセージの内容が XMLVARCHAR として戻されます。メッセージが使用可能でなければ、NULL が戻されます。

例 3: この例は、ポリシー *MYPOLICY* を使用して、サービス *MYSERVICE* が指定するキューの先頭にあるメッセージを読みます。

```
values DB2XML.MQREADXML('MYSERVICE','MYPOLICY')
```

正常に終了すれば、メッセージの内容が XMLVARCHAR として戻されます。メッセージが使用可能でなければ、NULL が戻されます。

関連概念:

- 231 ページの『MQSeries 用の XML Extender ストアード・プロシージャーと関数 - 概要』

関連資料:

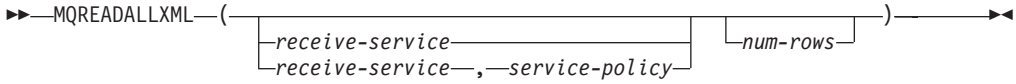
- ix ページの『構文図の読み方』

MQReadAllXML 関数

目的:

MQReadAllXML 関数は、*service-policy* の品質を使用する *receive-service* により指定される MQSeries ロケーションから、メッセージとメッセージ・メタデータを含む表を戻します。この操作を実行しても、メッセージは *receive-service* に関連付けられているキューから除去されません。 *num-rows* を指定すると、最大 num-rows 個のメッセージが戻されます。 num-rows を指定しないと、すべての有効なメッセージが戻されます。

構文:



パラメーター:

表 61. MQReadAllXML パラメーター

パラメーター	データ・タイプ	説明
receive-service	VARCHAR(48)	論理 MQSeries 宛先を含んでいるストリング。メッセージはここから読み取られます。 *receive-service* を指定する場合は、それで AMT.XML リポジトリー・ファイルに定義されているサービス・ポイントを参照する必要があります。ただし、*receive-service* を指定しないと、DB2.DEFAULT.SERVICE が使用されます。 *receive-service* の最大サイズは 48 バイトです。

表 61. *MQReadAllXML* パラメーター *(続き)*

パラメーター	データ・タイプ	説明
service-policy	VARCHAR(48)	このメッセージの処理に使用される、MQSeries AMI サービス・ポリシーを含んでいるストリング。 *service-policy* を指定する場合は、それで AMT.XML リポジトリー・ファイルに定義されているポリシーを参照します。receive-service の最大サイズは 48 バイトです。さらに詳しくは、MQSeries Application Messaging Interface の資料を参照してください。
num-rows	INTEGER	この関数によって戻されるメッセージの最大数を含む正の整数。

結果:

MQReadAllXML 関数は、下記に示すメッセージとメッセージ・メタデータを含む表を戻します。

表 62. *結果セット表*

列名	データ・タイプ	説明
MSG	XMLVARCHAR	MQSeries メッセージの内容。最大長は 4K バイトです。
CORRELID	VARCHAR(24)	メッセージに関連させるために使用できる相関 ID。
TOPIC	VARCHAR(40)	有効な場合は、メッセージが発行されたときのトピック。
QNAME	VARCHAR(48)	メッセージが入っていたキューの名前。
MSGID	VARCHAR(24)	MQSeries 割り当てによる、メッセージのためのユニーク ID。
MSGFORMAT	VARCHAR(8)	MQSeries が定義するメッセージ形式。通常のストリングは MQSTR の形式です。

例:

例 1: デフォルト・ポリシー DB2.DEFAULT.POLICY を使用して、デフォルト・サービス DB2.DEFAULT.SERVICE が指定するキューからすべてのメッセージを読みます。メッセージとすべてのメタデータが表形式で戻されます。

```
select * from table (DB2XML.MQREADALLXML()) t
```

例 2: デフォルトのポリシー DB2.DEFAULT.POLICY を使用して、サービス MYSERVICE が指定するすべてのメッセージを読みます。 *msg* と *correlid* の列だけが戻されます。メッセージ・キューは表形式になっており、そこから必要なフィールドを選択することができます。

```
select t.MSG, t.CORRELID from table (DB2XML.MQREADALLXML('MYSERVICE')) t
```

例 3: デフォルトのポリシー DB2.DEFAULT.POLICY を使用して、デフォルトのサービス DB2.DEFAULT.SERVICE が指定するキューを読み取ります。 *CORRELID* が '1234' のメッセージだけが戻されます。 10 個までのメッセージが、読み取られて戻されます。すべての列が戻されます。

```
select * from table (DB2XML.MQREADALLXML()) t where t.CORRELID = '1234'
```

例 4: デフォルトのサービス DB2.DEFAULT.SERVICE によって指定されたメッセージを、デフォルトのポリシー DB2.DEFAULT.POLICY を使用して読み取ります。すべての列が戻されます。

```
select * from table (DB2XML.MQREADALLXML(10)) t
```

関連概念:

- 231 ページの『MQSeries 用の XML Extender ストアード・プロシージャーと関数 - 概要』

関連資料:

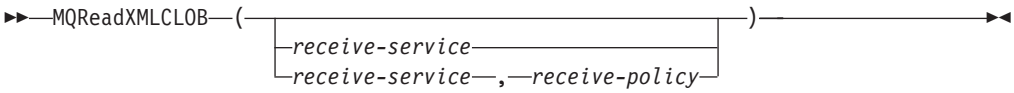
- ix ページの『構文図の読み方』

MQReadXMLCLOB 関数

目的:

MQREADXMLCLOB 関数は、*receive-policy* のサービス・ポリシーの品質を使用して、*receive-service* が指定する MQSeries ロケーションから XMLCLOB データを戻します。この操作を実行しても、*receive-service* に関連付けられているキューからメッセージは除去されません。キューの先頭にあるメッセージが戻されます。戻り値は、メッセージを含んでいる XMLCLOB です。使用可能なメッセージがなければ、NULL が戻されます。

構文:



パラメーター:

表 63. MQReadXMLCLOB パラメーター

パラメーター	データ・タイプ	説明
receive-service	VARCHAR(48)	論理 MQSeries 宛先を含んでいるストリング。メッセージはここから受け取ります。*receive-service* を指定する場合は、それで AMT.XML リポジトリー・ファイルに定義されているサービス・ポイントを参照します。 *receive-service* を指定しないと、DB2.DEFAULT.SERVICE が使用されます。*receive-service* の最大サイズは 48 バイトです。
receive-policy	VARCHAR(48)	このメッセージの処理に使用される、MQSeries AMI サービス・ポリシーを含んでいるストリング。 *receive-policy* を指定する場合は、それで AMT.XML リポジトリー・ファイルに定義されているポリシーを参照します。service-policy は、メッセージング操作に適用されるサービス・オプションの 1 組の品質を定義します。これらのオプションには、メッセージ優先順位とメッセージ持続性が含まれます。 *receive-policy* を指定しないと、デフォルトの DB2.DEFAULT.POLICY が使用されます。 receive-service の最大サイズは 48 バイトです。

結果:

キュー内のメッセージが正常に読めたときは、MQREADXMLCLOB は db2xml.xmlclob を戻します。メッセージが使用可能でなければ、NULL が戻されます。

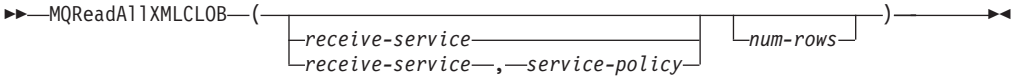
MQReadAllXMLCLOB 関数

目的:

MQReadAllXMLCLOB 関数は、receive-policy の品質を使用して、*receive-service* が指定する MQSeries ロケーションからメッセージとメッセージ・メタデータを含む

表を戻します。この操作を実行しても、receive-service に関連付けられているキューからメッセージは除去されません。 *num-rows* を指定すると、最大 num-rows 個のメッセージが戻されます。 num-rows を指定しないと、すべての使用可能なメッセージが戻されます。

構文:



パラメーター:

表 64. MQReadAllXMLCLOB パラメーター

パラメーター	データ・タイプ	説明
receive-service	VARCHAR(48)	論理 MQSeries 宛先を含んでいるストリング。メッセージはここから読み取られます。 *receive-service* を指定する場合は、それで AMT.XML リポジトリー・ファイルに定義されているサービス・ポイントを参照する必要があります。ただし、*receive-service* を指定しないと、DB2.DEFAULT.SERVICE が使用されます。 *receive-service* の最大サイズは 48 バイトです。
service-policy	VARCHAR(48)	このメッセージの処理に使用される、MQSeries AMI サービス・ポリシーを含んでいるストリング。 *service-policy* を指定する場合は、それで AMT.XML リポジトリー・ファイルに定義されているポリシーを参照します。 *service-policy* の最大サイズは 48 バイトです。
num-rows	INTEGER	この関数によって戻されるメッセージの最大数を含む正の整数。

結果:

MQReadAllXMLCLOB 関数は、下記に示すメッセージとメッセージ・メタデータを含む表を戻します。

表 65. MQReadAllXMLCLOB 結果セット表

列名	データ・タイプ	説明
MSG	XMLCLOB	MQSeries メッセージの内容。最大 1MB まで。

表 65. *MQReadAllXMLCLOB* 結果セット表 (続き)

列名	データ・タイプ	説明
CORRELID	VARCHAR(24)	メッセージに関連させるために使用できる相関 ID。
TOPIC	VARCHAR(40)	有効な場合は、メッセージが発行されたときのトピック。
QNAME	VARCHAR(48)	メッセージが入っていたキューの名前。
MSGID	VARCHAR(24)	MQSeries 割り当てによる、このメッセージのためのユニーク ID。
MSGFORMAT	VARCHAR(8)	MQSeries が定義するメッセージ形式。 通常のストリングは MQSTR の形式です。

例 1: デフォルト・ポリシー DB2.DEFAULT.POLICY を使用して、デフォルト・サービス DB2.DEFAULT.SERVICE が指定するキューからすべてのメッセージを読みます。メッセージとすべてのメタデータが表形式で戻されます。

```
select * from table (DB2XML.MQREADALLXMLCLOB()) t
```

例 2: この例は、デフォルト・ポリシー DB2.DEFAULT.POLICY を使用して、サービス MYSERVICE が指定するキューの先頭にあるメッセージを読みます。 *msg* と *correlid* の列だけが戻されます。

```
select t.MSG, t.CORRELID
from table (DB2XML.MQREADALLXMLCLOB('MYSERVICE')) t
```

例 3: デフォルト・ポリシー DB2.DEFAULT.POLICY を使用して、デフォルト・サービス DB2.DEFAULT.SERVICE が指定するキューの先頭を読みます。 *CORRELID* が '1234' のメッセージだけが戻されます。すべての列が戻されます。

```
select *
from table (DB2XML.MQREADALLXMLCLOB()) t where t.CORRELID = '1234'
```

例 4: デフォルト・ポリシー DB2.DEFAULT.POLICY を使用して、デフォルト・サービス DB2.DEFAULT.SERVICE が指定するキューの先頭から最初の 10 個のメッセージを読みます。 すべての列が戻されます。

```
select * from table (DB2XML.MQREADALLXMLCLOB(10)) t
```

関連概念:

- 231 ページの『MQSeries 用の XML Extender ストアード・プロシージャーと関数 - 概要』

関連資料:

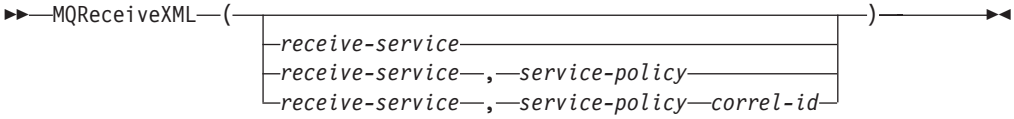
- ix ページの『構文図の読み方』

MQReceiveXML 関数

目的:

MQReceiveXML は、*receive-service* に関連付けられているメッセージをキューから除去します。この関数は、*receive-service* の品質を使用する *receive-service* 関数が指定する MQSeries ロケーションから XMLVARCHAR データを戻します。

構文:



パラメーター:

表 66. MQReceiveXML パラメーター

パラメーター	データ・タイプ	説明
receive-service	VARCHAR(48)	論理 MQSeries 宛先を含んでいるストリング。メッセージはここから受け取ります。 *receive-service* を指定する場合は、それで AMT.XML リポジトリー・ファイルに定義されているサービス・ポイントを参照します。 *receive-service* を指定しないと、DB2.DEFAULT.SERVICE が使用されます。 *receive-service* の最大サイズは 48 バイトです。
service-policy	VARCHAR(48)	メッセージの処理に使用される、MQSeries AMI サービス・ポリシーを含んでいるストリング。 *service-policy* を指定する場合は、これは AMT.XML リポジトリー・ファイルに定義されているポリシーを参照する必要があります。 *service-policy* を指定しないと、デフォルトの DB2.DEFAULT.POLICY が使用されます。 *service-policy* の最大サイズは 48 バイトです。

表 66. *MQReceiveXML* パラメーター *(続き)*

パラメーター	データ・タイプ	説明
correl-id	VARCHAR(24)	このメッセージに関連付けられるオプションの相関 ID を含んでいるストリング。 *correl-id* は、要求と応答を関連付けるために、しばしば要求/応答シナリオに指定されます。関連付けがない場合は、相関 ID は指定されません。 *correl-id* の最大サイズは 24 バイトです。

結果:

MQReceiveXML 関数は、メッセージを正常にキューから受け取ると、db2xml.XMLVARCHAR を戻します。最大長は 4000 バイトです。メッセージが使用可能でなければ、NULL が戻されます。 *correl-id* を指定すると、合致した相関 ID を持つ最初のメッセージが戻されます。 *correl-id* を指定しないと、キューの先頭にあるメッセージが戻されます。そのメッセージは、キューから除去されます。

例:

例 1: デフォルト・ポリシー DB2.DEFAULT.POLICY を使用して、デフォルト・サービス DB2.DEFAULT.SERVICE が指定しているキューの先頭にあるメッセージを読みます。

```
values db2xml.MQRECEIVEXML()
```

正常に終了すると、この例はメッセージの内容を XMLVARCHAR として戻します。メッセージが使用可能でなければ、NULL が戻されます。

関連概念:

- 231 ページの『MQSeries 用の XML Extender ストアード・プロシージャーと関数 - 概要』

関連資料:

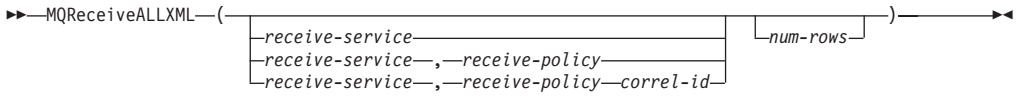
- ix ページの『構文図の読み方』

MQReceiveAllXML 関数

目的:

MQReceiveAllXML は、*receive-service* に関連付けられているメッセージをキューから除去します。 correl-id を指定すると、合致した相関 ID を持つメッセージだけが戻されます。 *correl-id* を指定しないと、キューの先頭にあるメッセージが戻されます。 *num-rows* を指定すると、最大で *num-rows* 個のメッセージが戻されます。指定しないと、すべての使用可能なメッセージが戻されます。

構文:


MQReceiveALLXML
(
receive-service
receive-service
,
receive-policy
receive-service
,
receive-policy
correl-id
num-rows
)

パラメーター:

表 67. MQReceiveAllXML パラメーター

パラメーター	データ・タイプ	説明
receive-service	VARCHAR(48)	論理 MQSeries 宛先を含むストリング。メッセージはここに送られます。*send-service* を指定する場合は、それで ATM.XML リポジトリー・ファイルに定義されているサービス・ポイントを参照します。 *send-service* を指定しないと、DB2.DEFAULT.SERVICE が使用されます。 *send-service* の最大サイズは 48 バイトです。
receive-policy	VARCHAR(48)	メッセージの処理に使用される、MQSeries AMI サービス・ポリシーを含んでいるストリング。 *receive-policy* を指定する場合は、これは AMT.XML リポジトリー・ファイルに定義されているポリシーを参照する必要があります。 *receive-policy* を指定しないと、デフォルトの DB2.DEFAULT.POLICY が使用されます。 *receive-policy* の最大サイズは 48 バイトです。
correl-id	VARCHAR(24)	このメッセージに関連付けられるオプションの相関 ID を含んでいるストリング。 *correl-id* は、要求と応答を関連付けるために、しばしば要求/応答シナリオに指定されます。関連付けがない場合は、相関 ID は指定されません。 *correl-id* の最大サイズは 24 バイトです。
num-rows	INTEGER	この関数によって戻されるメッセージの最大数を含む正の整数。

結果:

キューからメッセージの表を正常に受け取った場合は、MQRECEIVEXML は db2xml.xmlvarchar を戻します。メッセージがない場合 NULL が戻されます。 メッセージは、メタデータを持つメッセージの表として戻されます。

列名	データ・タイプ	説明
MSG	XMLVARCHAR	MQSeries メッセージの内容。
CORRELID	VARCHAR(24)	メッセージに関連させるために使用できる相関 ID。
TOPIC	VARCHAR(40)	有効な場合は、メッセージが発行されたときのトピック。
QNAME	VARCHAR(48)	メッセージが入っていたキューの名前。
MSGID	CHAR(24)	MQSeries 割り当てによる、このメッセージのためのユニーク ID。
MSGFORMAT	VARCHAR(8)	MQSeries が定義するメッセージ形式。 通常のストリングは MQSTR の形式です。

例:

例 1: キューから受け取るすべてのメッセージは、デフォルトのポリシー (DB2.DEFAULT.POLICY) を使用し、デフォルトのサービス (DB2.DEFAULT.SERVICE) で指定されます。メッセージとすべてのメタデータが表で戻されます。

```
select * from table (MQRECEIVEALLXML()) t
```

例 2: すべてのメッセージは、デフォルトのポリシー (DB2.DEFAULT.POLICY) を使用し、サービス MYSERVICE で指定されて、キューの先頭から受け取られます。MSG と CORRELID の列だけが戻されます。メッセージは表形式になっており、そこから必要なフィールドを選択できます。

```
select t.MSG, t.CORRELID from table (MQRECEIVEALLXML('MYSERVICE')) t
```

例 3: キューの先頭から受け取るすべてのメッセージは、ポリシー MYPOLICY を使用し、サービス MYSERVICE で指定され、ID '1234' に合致するものです。MSG と CORRELID の列だけが戻されます。

```
select t.MSG, t.CORRELID from table
(MQRECEIVEALLXML('MYSERVICE','MYPOLICY','1234')) t
```

例 4: 10 個のメッセージが、デフォルト・ポリシー DB2.DEFAULT.POLICY を使用して、デフォルト・サービス DB2.DEFAULT.SERVICE が指定するキューの先頭から受け取られます。すべての列が戻されます。

```
 select * from table (MQRECEIVEALLXML(10)) t
```

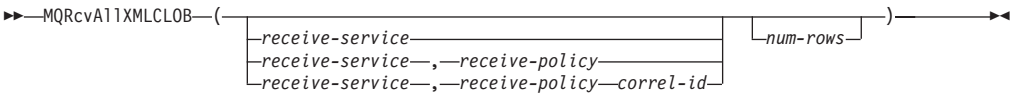
MQRcvAllXMLCLOB 関数

目的:

MQRcvAllXMLCLOB 関数は、*receive-service* に関連付けられているキューからメッセージを除去します。 correl-id を指定すると、合致した相関 ID を持つメッセージだけが戻されます。 *correl-id* が指定されていないとすべてのメッセージが戻されま

す。*num-rows* が指定されていると、最高 *num-rows* の数のメッセージが XMLCLOB として戻されます。指定しないと、すべての使用可能なメッセージが戻されます。

構文:



パラメーター:

表 68. MQRcvAllXMLCLOB パラメーター

パラメーター	データ・タイプ	説明
receive-service	VARCHAR(48)	論理 MQSeries 宛先を含んでいるストリング。メッセージはここから受け取ります。*receive-service* を指定する場合は、それで AMT.XML リポジトリー・ファイルに定義されているサービス・ポイントを参照します。 *receive-service* を指定しないと、DB2.DEFAULT.SERVICE が使用されます。receive-service の最大サイズは 48 バイトです。
receive-policy	VARCHAR(48)	メッセージの処理に使用される、MQSeries AMI サービス・ポリシーを含んでいるストリング。 *receive-policy* を指定する場合は、これは AMT.XML リポジトリー・ファイルに定義されているポリシーを参照する必要があります。 *receive-policy* を指定しないと、デフォルトの DB2.DEFAULT.POLICY が使用されます。 *receive-policy* の最大サイズは 48 バイトです。
correl-id	VARCHAR(24)	このメッセージに関連付けられるオプションの相関 ID を含んでいるストリング。*correl-id* は、要求と応答を関連付けるために、しばしば要求/応答シナリオに指定されます。関連付けがない場合は、相関 ID は指定されません。*correl-id* の最大サイズは 24 バイトです。

表 68. MQRcvAllXMLCLOB パラメーター (続き)

パラメーター	データ・タイプ	説明
num-rows	INTEGER	この関数によって戻されるメッセージの最大数を含む正の整数。

結果:

キューからメッセージを正常に受け取った場合は、MQRcvAllXMLCLOB は XMLCLOB を戻します。メッセージが使用可能でなければ、NULL が戻されます。メッセージは、下記に示す表で戻されます。

表 69. MQRcvAllXML 結果セット表

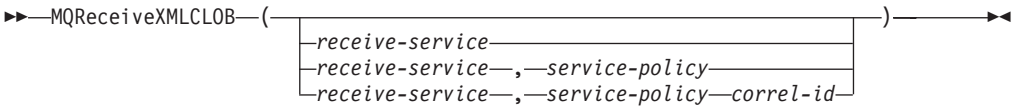
列名	データ・タイプ	説明
MSG	XMLCLOB	MQSeries メッセージの内容。
CORRELID	VARCHAR(24)	メッセージに関連させるために使用できる相関 ID。
TOPIC	VARCHAR(40)	有効な場合は、メッセージが発行されたときのトピック。
QNAME	VARCHAR(48)	メッセージが入っていたキューの名前。
MSGID	CHAR(24)	MQSeries 割り当てによる、このメッセージのためのユニーク ID。
MSGFORMAT	VARCHAR(8)	MQSeries が定義するメッセージ形式。 通常のストリングは MQSTR の形式です。

MQReceiveXMLCLOB 関数

目的:

MQReceiveXMLCLOB は、*receive-service* に関連付けられているメッセージをキューから除去します。この関数は、*receive-service* の品質を使用する *receive-service* 関数が指定する MQSeries ロケーションから XMLVARCHAR データを戻します。

構文:



パラメーター:

表 70. MQReceiveXMLCLOB パラメーター

パラメーター	データ・タイプ	説明
receive-service	VARCHAR(48)	論理 MQSeries 宛先を含んでいるストリング。メッセージはここから受け取ります。*receive-service* を指定する場合は、それで AMT.XML リポジトリー・ファイルに定義されているサービス・ポイントを参照します。ただし、*receive-service* を指定しないと、DB2.DEFAULT.SERVICE が使用されます。*receive-service* の最大サイズは 48 バイトです。
service-policy	VARCHAR(48)	メッセージの処理に使用される、MQSeries AMI サービス・ポリシーを含んでいるストリング。指定する場合は、receive-service は、AMT.XML リポジトリー・ファイルに定義されているポリシーを指していなければなりません。*service-policy* を指定しないと、デフォルトの DB2.DEFAULT.POLICY が使用されます。 *service-policy* の最大サイズは 48 バイトです。
correl-id	VARCHAR(24)	このメッセージに関連付けられるオプションの相関 ID を含んでいるストリング。*correl-id* は、要求と応答を関連付けるために、しばしば要求/応答シナリオに指定されます。関連付けがない場合は、相関 ID は指定されません。*correl-id* の最大サイズは 24 バイトです。

結果:

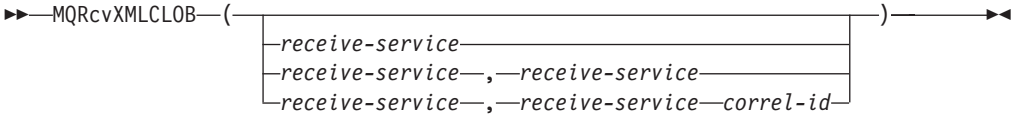
MQReceiveXMLCLOB 関数は、メッセージを正常にキューから受け取ると、db2xml.XMLCLOB を戻します。メッセージが使用可能でなければ、NULL が戻されます。 *correl-id* を指定すると、合致した相関 ID を持つ最初のメッセージが戻されます。しかし、*correl-id* を指定しないと、キューの先頭にあるメッセージが戻されます。

MQRcvXMLCLOB 関数

目的:

MQRcvXMLCLOB は、*receive-service* に関連付けられているメッセージをキューから除去します。この関数は、*receive-service* の品質を使用する *receive-service* 関数が指定する MQSeries ロケーションから XMLVARCHAR データを戻します。

構文:



パラメーター:

表 71. MQRcvXMLCLOB パラメーター

データ・タイプ	データ・タイプ	説明
receive-service	VARCHAR(48)	論理 MQSeries 宛先を含んでいるストリング。メッセージはここから受け取ります。*receive-service* を指定する場合は、それで AMT.XML リポジトリー・ファイルに定義されているサービス・ポイントを参照します。ただし、*receive-service* を指定しないと、DB2.DEFAULT.SERVICE が使用されます。*receive-service* の最大サイズは 48 バイトです。
receive-service	VARCHAR(48)	メッセージの処理に使用される、MQSeries AMI サービス・ポリシーを含んでいるストリング。指定する場合は、receive-service は、AMT.XML リポジトリー・ファイルに定義されているポリシーを指していなければなりません。*receive-service* を指定しないと、デフォルトの DB2.DEFAULT.POLICY が使用されます。 *receive-service* の最大サイズは 48 バイトです。

表 71. *MQRcvXMLCLOB* パラメーター *(続き)*

データ・タイプ	データ・タイプ	説明
correl-id	VARCHAR(24)	このメッセージに関連付けられるオプションの相関 ID を含んでいるストリング。*correl-id* は、要求と応答を関連付けるために、しばしば要求/応答シナリオに指定されます。関連付けがない場合は、相関 ID は指定されません。*correl-id* の最大サイズは 24 バイトです。

結果:

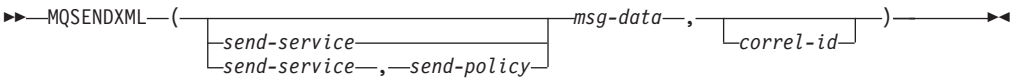
MQRcvMLCLOB 関数は、メッセージを正常にキューから受け取ると、db2xml.XMLCLOB を戻します。最大メッセージ長は 1M バイトです。メッセージが使用可能でなければ、NULL が戻されます。 *correl-id* を指定すると、合致した相関 ID を持つ最初のメッセージが戻されます。しかし、*correl-id* を指定しないと、キューの先頭にあるメッセージが戻されます。

MQSENDXML 関数

目的:

MQSENDXML 関数は、send-policy を使用して、*msg_data* に入っているデータを、send-service が指定する MQSeries ロケーションに送ります。オプションのユーザー定義メッセージ相関 ID も、correl-id で指定することができます。関数は、正常に終了すれば 1 を戻します。

構文:



パラメーター:

表 72. *MQSendXML* パラメーター

パラメーター	データ・タイプ	説明
msg-data	XMLVARCHAR または XMLCLOB	MQSeries を介して送られるデータが入っている式。

表 72. *MQSendXML* パラメーター (続き)

パラメーター	データ・タイプ	説明
send-service	VARCHAR(48)	論理 MQSeries 宛先を含むストリング。メッセージはここに送られます。*send-service* を指定する場合は、それで AMT.XML リポジトリー・ファイルに定義されているサービス・ポイントを参照します。 *send-service* を指定しないと、DB2.DEFAULT.SERVICE が使用されます。 *send-service* の最大サイズは 48 バイトです。
send-policy	VARCHAR(48)	このメッセージの処理に使用される、MQSeries AMI サービス・ポリシーを含んでいるストリング。 *send-policy* を指定する場合は、それで AMT.XML リポジトリー・ファイルに定義されているポリシーを参照します。*send-policy* を指定しないと、デフォルトの DB2.DEFAULT.POLICY が使用されます。 send-policy の最大サイズは 48 バイトです。
correl-id	VARCHAR(24)	このメッセージに関連付けられるオプションの相関 ID を含んでいるストリング。correl-id は、要求と応答を関連付けるために、しばしば要求/応答シナリオに指定されます。指定しないと、相関 ID は表示されません。 correl-id の最大サイズは 24 バイトです。

結果:

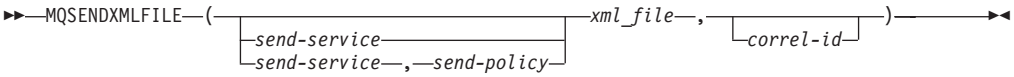
正常なメッセージの結果は、値 1 となります。*msg-data* に入っているメッセージが、*send-policy* で定義されたポリシーを使用して、send-service の指定するロケーションに送られます。

MQSENDXMLFILE 関数

目的:

MQSENDXMLFILE 関数は、service-policy の品質を使用して、*xml_file* に入っているデータを、send-service が指定する MQSeries ロケーションに送ります。オプションのユーザー定義メッセージ相関 ID は、correl-id によって指定することができます。関数は、正常に終了すれば '1' を戻します。

構文:



パラメーター:

表 73. MQSENDXMLFILE パラメーター

パラメーター	データ・タイプ	説明
xml_file	XMLCLOB	最大サイズ 80 バイトまでの XML ファイル名。このファイルには、MQSeries を介して送られるデータが入っています。
send-service	VARCHAR(48)	論理 MQSeries 宛先を含むストリング。メッセージはここに送られます。*send-service* を指定する場合は、それで AMT.XML リポジトリー・ファイルに定義されているサービス・ポイントを参照します。 *send-service* を指定しないと、DB2.DEFAULT.SERVICE が使用されます。 *send-service* の最大サイズは 48 バイトです。
send-policy	VARCHAR(48)	メッセージの処理に使用される、MQSeries AMI サービスを含んでいるストリング。*send-policy* を指定する場合は、それで AMT.XML リポジトリー・ファイルに定義されているポリシーを参照します。 *send-policy* を指定しないと、デフォルトの DB2.DEFAULT.POLICY が使用されます。 *send-policy* の最大サイズは 48 バイトです。

表 73. *MQSENDXMLFILE* パラメーター *(続き)*

パラメーター	データ・タイプ	説明
correl-id	VARCHAR(24)	このメッセージに関連付けられるオプションの相関 ID を含んでいるストリング。*correl-id* は、要求と応答を関連付けるために、しばしば要求/応答シナリオに指定されます。 指定しないと、相関 ID はリストに載りません。*correl-id* の最大サイズは 24 バイトです。

結果:

この関数は正常に終了すると、結果が '1' になります。この関数が正常に実行された場合のもう 1 つの結果として、*msg-data* に入っているメッセージが、*send-policy* で定義されたポリシーを使用して、*send-service* の指定するロケーションに送られます。

例:

例 1: ファイル "c:¥xml¥test1.xml" に入っている XML 文書が、相関 ID の指定なしに、デフォルト・ポリシー (DB2.DEFAULT.POLICY) を使用して、デフォルト・サービス (DB2.DEFAULT.SERVICE) に送られます。

```
Values MQSENDXMLFILE('c:¥xml¥test1.xml');
```

この例は、正常に終了すれば、値 '1' を戻します。

例 2: ファイル c:¥xml¥test2.xml に入っている XML 文書が、相関 ID の指定なしに、ポリシー MYPOLICY を使用し、サービス MYSERVICE に送られます。

```
Values MQSENDXMLFILE('MYSERVICE', 'MYPOLICY', 'c:¥xml¥test2.xml');
```

この例は、正常に終了すれば、値 '1' を戻します。

例 3: ファイル "c:¥xml¥test3.xml" に入っている XML 文書が、相関 ID を "Test3" として、ポリシー MYPOLICY を使用し、サービス MYSERVICE に送られます。

```
Values MQSENDXML('MYSERVICE','MYPOLICY', 'c:¥xml¥test3.xml', 'Test3');
```

この例は、正常に終了すれば、値 '1' を戻します。

例 4: ファイル "c:¥xml¥test4.xml" に入っている XML 文書が、相関 ID を指定せず、デフォルト・ポリシー (DB2.DEFAULT.POLICY) を使用して、サービス MYSERVICE に送られます。

```
Values MQSENDXMLFILE('MYSERVICE', 'c:¥xml¥test4.xml');
```

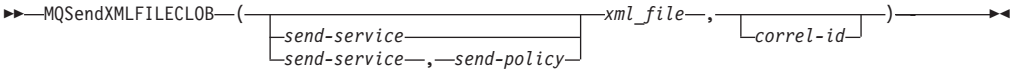
この例は、正常に終了すれば、値 '1' を戻します。

MQSendXMLFILECLOB 関数

目的:

MQSendXMLFILECLOB 関数は、*send-policy* の品質を使用して、*xml_file* に入っているデータを、send-service が指定する MQSeries ロケーションに送ります。送信されたデータ・タイプは XMLCLOB です。オプションのユーザー定義メッセージ相関 ID は、correl-id によって指定することができます。関数は、正常に終了すれば 1 を戻します。

構文:



パラメーター:

表 74. MQSENDXMLFILE パラメーター

パラメーター	データ・タイプ	説明
xml_file	XMLCLOB	最大サイズ 80 バイトまでの XML ファイル名。このファイルには、MQSeries を介して送られるデータが入っています。
send-service	VARCHAR(48)	論理 MQSeries 宛先を含むストリング。メッセージはここに送られます。send-service を指定する場合は、それで AMT.XML リポジトリー・ファイルに定義されているサービス・ポイントを参照します。 *send-service* を指定しないと、DB2.DEFAULT.SERVICE が使用されます。 send-service の最大サイズは 48 バイトです。
send-policy	VARCHAR(48)	メッセージの処理に使用される、MQSeries AMI サービスを含んでいるストリング。 *send-policy* を指定する場合は、それで AMT.XML リポジトリー・ファイルに定義されているポリシーを参照します。 *send-policy* を指定しないと、デフォルトの DB2.DEFAULT.POLICY が使用されます。 *send-policy* の最大サイズは 48 バイトです。

表 74. *MQSENDXMLFILE* パラメーター *(続き)*

パラメーター	データ・タイプ	説明
correl-id	VARCHAR(24)	このメッセージに関連付けられるオプションの相関 ID を含んでいるストリング。*correl-id* は、要求と応答を関連付けるために、しばしば要求/応答シナリオに指定されます。指定しないと、相関 ID はリストに載りません。*correl-id* の最大サイズは 24 バイトです。

結果:

この関数は正常に終了すると、結果が '1' になります。この関数が正常に実行された場合のもう 1 つの結果として、*msg-data* に入っているメッセージが、*send-policy* で定義されたポリシーを使用して、*send-service* の指定するロケーションに送られます。

XML Extender の MQSeries ストアード・プロシージャー

XML Extender MQSeries ストアード・プロシージャー - 概要

合成ストアード・プロシージャー

合成のためのストアード・プロシージャー dxxmqGen()、dxxmqGenCLOB()、 dxxmqRetrieve()、および dxxmqRetrieveCLOB() は、既存のデータベース表内のデータを使用して XML 文書を生成し、生成した XML 文書をメッセージ・キューに送るために使用されます。dxxmqGen() と dxxmqGenCLOB() のストアード・プロシージャーは、入力として DAD ファイルを使用します。これらのプロシージャーでは、使用可能になった XML コレクションを必要としません。 dxxmqRetrieve と dxxmqRetrieveCLOB のストアード・プロシージャーは、入力としてコレクション名を使用します。

分解ストアード・プロシージャー

分解のためのストアード・プロシージャー dxxmqInsert()、dxxmqInsertAll()、 dxxInsertCLOB()、 dxxmqShred()、 dxxmqShredCLOB、および dxxmqShredAll() は、メッセージ・キューからの入力 XML 文書を分解、すなわち細分化するためと、データを新規のまたは既存のデータベース表に保管するために使用されます。

dxxmqInsert()、dxxmqInsertAll()、dxxmqInsertAllCLOB()、および dxxInsertCLOB() のストアード・プロシージャーは、使用可能になった XML コレクション名を入力として使用します。

また、dxxmqShred()、dxxmqShredAll()、dxxmqShredCLOB、および dxxmqShredAllCLOB のストアード・プロシージャーは、DAD ファイルを入力として使用します。これらのプロシージャーでは、使用可能になった XML コレクションを必要としません。

下記の表は、これら各種のストアード・プロシージャーを要約し、その機能を説明しています。

表 75. MQSeries® XML ストアード・プロシージャー

関数	目的
dxxmqGen	入力パラメーターとして DAD ファイルを使用し、XML 文書を合成するために dxxmqGen ストアード・プロシージャーを呼び出します。結果として得られる文書タイプは XMLVARCHAR(4000) です。
dxxmqGenCLOB	これは、DAD ファイルに指定された XML コレクション表に保管されているデータから XML 文書を構成し、その XML 文書を MQ メッセージ・キューに送ります。結果として得られる文書タイプは XMLCLOB(1M) です。
dxxmqRetrieve	入力パラメーターとしてコレクション名を使用し、XML 文書を合成するために dxxmqRetrieve ストアード・プロシージャーを呼び出します。結果として得られる文書タイプは XMLVARCHAR(4000) です。
dxxmqRetrieveCLOB	入力パラメーターとしてコレクション名を使用し、XML 文書を合成するために dxxmqRetrieve ストアード・プロシージャーを呼び出します。結果として得られる文書タイプは XMLCLOB(1M) です。
dxxmqShred	入力パラメーターとして DAD ファイルを使用し、XML 文書を分解するために dxxmqShred ストアード・プロシージャーを呼び出します。結果として得られる文書タイプは XMLVARCHAR(4000) です。
dxxmqShredAll	入力パラメーターとして DAD ファイルを使用し、複数の XML 文書を分解するために dxxmqShredAll ストアード・プロシージャーを呼び出します。結果として得られる文書タイプは XMLVARCHAR(4000) です。
dxxmqShredCLOB	これは、DAD ファイル・マッピングに基づいて、メッセージ・キューからの入力 XML 文書を分解し、その XML エレメントと属性の内容を指定の DB2 ®表に保管します。結果として得られる文書タイプは XMLCLOB(1M) です。

表 75. MQSeries® XML ストアード・プロシージャー (続き)

関数	目的
dxxmqShredAllCLOB	これは、DAD ファイル・マッピングに基づいて、メッセージ・キューからの入力 XML 文書を分解し、その XML エレメントと属性の内容を指定の DB2 UDB 表に保管します。結果として得られる文書タイプは XMLCLOB(1M) です。
dxxmqInsert	入力パラメーターとしてコレクション名を使用し、XML 文書を分解するために dxxmqInsert ストアード・プロシージャーを呼び出します。結果として得られる文書タイプは XMLVARCHAR(4000) です。
dxxmqInsertAll	入力パラメーターとしてコレクション名を使用し、複数の XML 文書を分解するために dxxmqInsertAll ストアード・プロシージャーを呼び出します。結果として得られる文書タイプは XMLVARCHAR(4000) です。
dxxmqInsertCLOB	これは、メッセージ・キューからの入力 XML 文書を分解、つまり細分化して、データを新規あるいは既存のデータベース表に保管します。結果として得られる文書タイプは XMLCLOB(1M) です。
dxxmqInsertAllCLOB	これは、メッセージ・キューからのすべての入力 XML 文書を分解、つまり細分化して、データを新規あるいは既存のデータベース表に保管します。 dxxmqInsertAllCLOB は、DAD ファイル名ではなくコレクション名を使用して、データの保管方法を判別します。結果として得られる文書タイプは XMLCLOB(1M) です。

関連資料:

- 262 ページの『dxxmqGenCLOB ストアード・プロシージャー』
- 264 ページの『dxxmqRetrieve ストアード・プロシージャー』
- 267 ページの『dxxmqRetrieveCLOB ストアード・プロシージャー』
- 270 ページの『dxxmqShred ストアード・プロシージャー』
- 271 ページの『dxxmqShredAll ストアード・プロシージャー』
- 273 ページの『dxxmqShredCLOB ストアード・プロシージャー』
- 276 ページの『dxxmqInsert ストアード・プロシージャー』
- 279 ページの『dxxmqInsertAll ストアード・プロシージャー』
- 277 ページの『dxxmqInsertCLOB ストアード・プロシージャー』
- 260 ページの『dxxmqGen() ストアード・プロシージャー』
- 274 ページの『dxxmqShredAllCLOB ストアード・プロシージャー』
- 281 ページの『dxxmqInsertAllCLOB ストアード・プロシージャー』

dxxmqGen() ストアード・プロシージャー

目的:

これは、DAD ファイルに指定された XML コレクション表に保管されているデータから XML 文書を構成し、その XML 文書を MQ メッセージ・キューに送ります。このストアード・プロシージャーは、ストアード・プロシージャーの状況を示すストリングを戻します。

動的照会をサポートするために、 dxxmqGen() 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。

構文:

```
dxxmqGen(varchar(48)    serviceName,     /*input*/
         varchar(48)    policyName,      /*input*/
         varchar(80)    dadFileName,     /*input*/
         integer        overrideType,    /*input*/
         varchar(1024)  override,        /*input*/
         integer        maxRows,         /*input*/
         integer        numRows,         /*output*/
         char(20)       status)          /*output*/
```

パラメーター:

表 76. dxxmqGen() パラメーター

パラメーター	説明	IN/OUT パラメーター
serviceName	論理 MQSeries 宛先を含むストリング。メッセージはここに送られます。*serviceName* を指定する場合は、それで AMT.XML リポジトリー・ファイルに定義されているサービス・ポイントを参照します。*serviceName* を指定しないと、DB2.DEFAULT.SERIVCE が使用されます。 *serviceName* の最大サイズは 48 バイトです。	IN
policyName	メッセージの処理に使用される、MQSeries AMI サービス・ポリシーを含んでいるストリング。 *policyName* を指定する場合は、それで AMT.XML リポジトリー・ファイルに定義されているポリシーを参照します。 *policyName* を指定しないと、デフォルトの DB2.DEFAULT.POLICY が使用されます。 *policyName* の最大サイズは 48 バイトです。	IN
dadFileName	DAD ファイルの名前。	IN

表 76. *dxxmqGen()* 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**: NULL 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 つ以上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	メッセージ・キューに実際に生成されたメッセージの最大数	IN
numRows	メッセージ・キューに実際に生成された行の数	OUT
status	ストアード・プロシージャーが正常に実行されたかどうかを示し、エラー・コードが生成された場合はそのコード、および、メッセージ・キューに送られたかまたはそこから受け取られた XML 文書数を示す、戻されたテキストとコード。	OUT

例:

次の例の断片は、XML 文書を生成し、それをキューに送ります。この例では、MQ/AMI サービスの *myService*、およびポリシーの *myPolicy* がリポジトリー・ファイルに定義されていると想定しています。このファイルは、XML 形式でリポジトリー定義を保管しています。

```
#include "dxx.h"
#include "dxxrc.h"
               EXEC SQL INCLUDE SQLCA;
               EXEC SQL BEGIN DECLARE SECTION;
               char           serviceName[48];  /* name of the MQ/AMI service*/
               char           policyName[48];   /* name of the MQ/AMI policy*/
```

```
char           dadFileName[80];  /* name of the DAD file */
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 */
char           status[20]        /* status code or message */
short          ovtype_ind;
short          ov_ind;
short          maxrow_ind;
short          numrow_ind;
short          dadFileName_ind;
short          serviceName_ind;
short          policyName_ind;
short          status_ind;

EXEC SQL END DECLARE SECTION;
strcpy(dadFileName,"c:¥dxx¥dad¥litem3.dad");
strcpy(serviceName,"myService");
strcpy(policyName,"myPolicy");
override[0] = '¥0';
overrideType = NO_OVERRIDE;
max_row = 500;
num_row = 0;
status[0] = '¥0';
dadFileName_ind = 0;
serviceName_ind = 0;
policyName_ind = 0;
maxrow_ind = 0;
numrow_ind = -1;
ovtype_ind=0;
ov_ind=-1;
status_ind = -1;

/* Call the store procedure */
EXEC SQL CALL dxxmqGen(:serviceName:serviceName_ind,
       :policyName:policyName_ind,
       :dadFileName:dadFileName_ind,
       :overrideType:ovtype_ind,
       :override:ov_ind,
       :max_row:maxrow_ind,
       :num_row:numrow_ind,
       :status:status_ind);
```

関連概念:

- 231 ページの『MQSeries 用の XML Extender ストアード・プロシージャーと関数 - 概要』

関連タスク:

- 206 ページの『XML Extender ストアード・プロシージャーの呼び出し』

関連資料:

- ix ページの『構文図の読み方』
- 335 ページの『付録 C. XML Extender の制限』

dxxmqGenCLOB ストアード・プロシージャー

目的:

これは、DAD ファイルに指定された XML コレクション表に保管されているデータから XML 文書を構成し、その XML 文書を MQ メッセージ・キューに送ります。文書タイプは XMLCLOB です。このストアード・プロシージャーは、ストア

ード・プロシージャーの状況を示すストリングを戻します。このストアード・プロシージャーは、エンタープライズ・サーバー・エディション (ESE) ではサポートされていません。

動的照会をサポートするために、 dxxmqGenCLOB 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。

構文:

```
dxxmqGenCLOB(varchar(48)    serviceName,     /*input*/
             varchar(48)    policyName,      /*input*/
             varchar(80)    dadFileName,     /*input*/
             integer        overrideType,    /*input*/
             varchar(1024)  override,        /*input*/
             integer        maxRows,         /*input*/
             integer        numRows,         /*output*/
              char(20)       status)         /*output*/
```

パラメーター:

表 77. dxxmqGenCLOB パラメーター

パラメーター	説明	IN/OUT パラメーター
serviceName	論理 MQSeries 宛先を含むストリング。メッセージはここに送られます。*serviceName* を指定する場合は、それで AMT.XML リポジトリー・ファイルに定義されているサービス・ポイントを参照します。 *serviceName* を指定しないと、DB2.DEFAULT.SERIVCE が使用されます。 *serviceName* の最大サイズは 48 バイトです。	IN
policyName	メッセージの処理に使用される、MQSeries AMI サービス・ポリシーを含んでいるストリング。 *policyName* を指定する場合は、それで AMT.XML リポジトリー・ファイルに定義されているポリシーを参照します。 *policyName* を指定しないと、デフォルトの DB2.DEFAULT.POLICY が使用されます。 *policyName* の最大サイズは 48 バイトです。	IN
dadFileName	DAD ファイルの名前。	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 77. *dxxmqGenCLOB* 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**: NULL 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 つ以上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	メッセージ・キューに実際に生成されたメッセージの最大数	IN
numRows	メッセージ・キューに実際に生成された行の数	OUT
status	ストアード・プロシージャーが正常に実行されたかどうかを示し、エラー・コードが生成された場合はそのコード、および、メッセージ・キューに送られたかまたはそこから受け取られた XML 文書数を示す、戻されたテキストとコード。	OUT

関連概念:

- 231 ページの『MQSeries 用の XML Extender ストアード・プロシージャーと関数 - 概要』

関連資料:

- ix ページの『構文図の読み方』
- 335 ページの『付録 C. XML Extender の制限』

dxxmqRetrieve ストアード・プロシージャー

目的:

分解された XML 文書を、ストアード・プロシージャー dxxmqRetrieve() を使って取り出すこともできます。 dxxmqRetrieve() は、入力として、使用可能にされたコレクション名を含んでいるバッファー、MQ/AMI サービス名、およびポリシー名を取ります。これは、合成された XML 文書を MQ キューに送ります。また、キュ

ーに送られた行数と状況メッセージを戻します。ストアード・プロシージャー dxxmqRetrieve により、同じ DAD ファイルを合成と分解の両方に使用することができます。

動的照会をサポートするために、 dxxmqRetrieve() 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。

dxxmqRetrieve() を使用するための DAD ファイルの要件は、 dxxmqGen() を使用する場合の要件と同じです。唯一の違いは、dxxmqRetrieve() 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を必須入力パラメーターとすることです。

構文:

```
dxxmqRetrieve(varchar(48)    serviceName,        /*input*/
              varchar(48)    policyName,         /*input*/
              varchar(80)    collectionName,     /*input*/
              integer        overrideType,       /*input*/
              varchar(1024)  override,           /*input*/
              integer        maxrows,            /*input*/
              integer        numrows,            /*output*/
              char(20)       status)             /*output*/
```

パラメーター:

表 78. dxxmqRetrieve() パラメーター

パラメーター	説明	IN/OUT パラメーター
serviceName	論理 MQSeries 宛先を含むストリング。メッセージはここに送られます。 *serviceName* を指定する場合は、それで AMT.XML リポジトリー・ファイルに定義されているサービス・ポイントを参照します。 *serviceName* を指定しないと、DB2.DEFAULT.SERVICE が使用されます。 *serviceName* の最大サイズは 48 バイトです。	IN
policyName	メッセージの処理に使用される、MQSeries AMI サービス・ポリシーを含んでいるストリング。 *policyName* を指定する場合は、それで AMT.XML リポジトリー・ファイルに定義されているポリシーを参照します。 *policyName* を指定しないと、デフォルトの DB2.DEFAULT.POLICY が使用されます。 *policyName* の最大サイズは 48 バイトです。	IN
collectionName	使用可能なコレクションの名前	IN

表 78. *dxxmqRetrieve()* 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**: NULL 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 つ以上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。最大長は 1024 バイトです。 *overrideType* ストリングを使用するには、DAD ファイルで 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
status	ストアード・プロシージャーが正常に実行されたかどうかを示し、エラー・コードが生成された場合はそのコード、および、メッセージ・キューに送られたかまたはそこから受け取られた XML 文書数を示す、戻されたテキストとコード。	OUT

例:

以下の断片は dxxmqRetrieve() の呼び出しの例です。

```
#include "dxx.h"
#include "dxxrc.h"
              EXEC SQL INCLUDE SQLCA;
              EXEC SQL BEGIN DECLARE SECTION;
              char      serviceName[48];   /* name of the MQ/AMI service*/
              char      policyName[48];    /* name of the MQ/AMI policy*/
              char      collection[32];    /* name of the XML collection */
              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 */
char      status[20];        /* status code or message */
short     ovtype_ind;
short     ov_ind;
short     maxrow_ind;
short     numrow_ind;
short     collection_ind;
short     serviceName_ind;
short     policyName_ind;
short     status_ind;

EXEC SQL END DECLARE SECTION;

/* initialize host variable and indicators */
strcpy(collection,"sales_ord");
strcpy(serviceName,"myService");
strcpy(policyName,"myPolicy");
override[0] = '¥0';
overrideType = NO_OVERRIDE;
max_row = 500;
num_row = 0;
status[0] = '¥0';
serviceName_ind = 0;
policyName_ind = 0;
collection_ind = 0;
maxrow_ind = 0;
numrow_ind = -1;
ovtype_ind=0;
ov_ind=-1;
status_ind = -1;

/* Call the store procedure */
EXEC SQL CALL dxxmqRetrieve(:serviceName:serviceName_ind,
       :policyName:policyName_ind,
       :collection:collection_ind,
       :overrideType:ovtype_ind,
       :override:ov_ind,
       :max_row:maxrow_ind,
       :num_row:numrow_ind,
       :status:status_ind);
```

関連概念:

- 231 ページの『MQSeries 用の XML Extender ストアード・プロシージャーと関数 - 概要』

関連資料:

- ix ページの『構文図の読み方』
- 335 ページの『付録 C. XML Extender の制限』

dxxmqRetrieveCLOB ストアード・プロシージャー

目的:

分解された XML 文書を、ストアード・プロシージャー dxxmqRetrieveCLOB を使って取り出すこともできます。 dxxmqRetrieveCLOB は、入力として、使用可能化されたコレクション名を含んでいるバッファー、MQ/AMI サービス名、およびポリシー名を取ります。これは、合成された XML 文書を MQ キューに送ります。また、キューに送られた行数と状況メッセージを戻します。ストアード・プロシージャー dxxmqRetrieveCLOB により、同じ DAD ファイルを合成と分解の両方に使用

することができます。このストアード・プロシージャーは、エンタープライズ・サーバー・エディション (ESE) ではサポートされていません。

動的照会をサポートするために、 dxxmqRetrieveCLOB 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。

dxxmqRetrieveCLOB を使用するための DAD ファイルの要件は、 dxxmqGenCLOB を使用する場合の要件と同じです。唯一の違いは、dxxmqRetrieveCLOB 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を必須入力パラメーターとすることです。

構文:

```
dxxmqRetrieveCLOB(varchar(48)    serviceName,        /*input*/
                  varchar(48)    policyName,         /*input*/
                  varchar(80)    collectionName,     /*input*/
                  integer        overrideType,       /*input*/
                  varchar(1024)  override,           /*input*/
                  integer        maxrows,            /*input*/
                  integer        numrows,            /*output*/
                  char(20)       status)             /*output*/
```

パラメーター:

表 79. dxxmqRetrieveCLOB パラメーター

パラメーター	説明	IN/OUT パラメーター
serviceName	論理 MQSeries 宛先を含むストリング。メッセージはここに送られます。 *serviceName* を指定する場合は、それで AMT.XML リポジトリー・ファイルに定義されているサービス・ポイントを参照します。 *serviceName* を指定しないと、DB2.DEFAULT.SERVICE が使用されます。 *serviceName* の最大サイズは 48 バイトです。	IN
policyName	メッセージの処理に使用される、MQSeries AMI サービス・ポリシーを含んでいるストリング。 *policyName* を指定する場合は、それで AMT.XML リポジトリー・ファイルに定義されているポリシーを参照します。 *policyName* を指定しないと、デフォルトの DB2.DEFAULT.POLICY が使用されます。 *policyName* の最大サイズは 48 バイトです。	IN
collectionName	使用可能なコレクションの名前	IN

表 79. *dxxmqRetrieveCLOB* 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**: NULL 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 つ以上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。最大サイズは 1024 バイトです。 *overrideType* ストリングを使用するには、DAD ファイルで 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
status	ストアード・プロシージャーが正常に実行されたかどうかを示し、エラー・コードが生成された場合はそのコード、および、メッセージ・キューに送られたかまたはそこから受け取られた XML 文書数を示す、戻されたテキストとコード。	OUT

関連概念:

- 231 ページの『MQSeries 用の XML Extender ストアード・プロシージャーと関数 - 概要』

関連資料:

- ix ページの『構文図の読み方』
- 335 ページの『付録 C. XML Extender の制限』

dxxmqShred ストアード・プロシージャー

目的:

これは、DAD ファイル・マッピングに基づいて、メッセージ・キューからの入力 XML 文書を分解し、その XML エレメントと属性の内容を指定の DB2 UDB 表に保管します。

dxxmqShred() 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。このストアード・プロシージャーでは、DAD で結合条件に指定されている列が、既存の表の主キー/外部キーの関係 (リレーションシップ) に対応している必要があります。 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。

構文:

```
dxxmqShred(varchar(48)   serviceName,     /* input */
           varchar(48)   policyName,     /* input */
           varchar(80)   dadFileName,    /* input */
           varchar(10)   status)         /* output */
```

パラメーター:

表 80. dxxmqShred() パラメーター

パラメーター	説明	IN/OUT パラメーター
serviceName	論理 MQSeries 宛先を含むストリング。メッセージはここに送られます。*serviceName* を指定する場合は、それで AMT.XML リポジトリー・ファイルに定義されているサービス・ポイントを参照します。*serviceName* を指定しないと、DB2.DEFAULT.SERVICE が使用されます。 *serviceName* の最大サイズは 48 バイトです。	IN
policyName	メッセージの処理に使用される、MQSeries AMI サービス・ポリシーを含んでいるストリング。 *policyName* を指定する場合は、それで AMT.XML リポジトリー・ファイルに定義されているポリシーを参照します。 *policyName* を指定しないと、デフォルトの DB2.DEFAULT.POLICY が使用されます。 *policyName* の最大サイズは 48 バイトです。	IN
dadFileName	DAD ファイルの名前。最大サイズは 80 バイトです。	IN

表 80. *dxxmqShred()* パラメーター *(続き)*

パラメーター	説明	IN/OUT パラメーター
status	ストアード・プロシージャーが正常に実行されたかどうかを示し、エラー・コードが生成された場合はそのコード、および、メッセージ・キューに送られたかまたはそこから受け取られた XML 文書数を示す、戻されたテキストとコード。	OUT

例:

以下の断片は dxxmqShred() の呼び出しの例です。

```
#include "dxx.h"
#include "dxxrc.h"

EXEC SQL INCLUDE SQLCA;
        EXEC SQL BEGIN DECLARE SECTION;
         char        serviceName[48];    /* name of the MQ/AMI service */
         char        policyName[48];     /* name of the MQ/AMI policy */
         char        dadFileName[80];    /* name of the DAD file */
         char        status[20];         /* status code or message */
         short       serviceName_ind;
         short       policyName_ind;
         short       dadFileName_ind;
         short       status_ind;
         EXEC SQL END DECLARE SECTION;

              /* initialize host variable and indicators */
         strcpy(dadFileName,"e:/dxx/samples/dad/getstart_xcollection.dad");
         strcpy(serviceName, "myService");
         strcpy(policyName, "myPolicy");
         status[0]='¥0';
         serviceName_ind=0;
         policyName_ind=0;
         dadFileName_ind=0;
         status_ind=-1;

              /* Call the store procedure */
         EXEC SQL CALL dxxmqShred(:serviceName:serviceName_ind,
                         :policyName:policyName_ind,
                         :dadFileName:dadFileName_ind,
                         :status:status_ind);
```

関連資料:

- 335 ページの『付録 C. XML Extender の制限』

dxxmqShredAll ストアード・プロシージャー

目的:

これは、メッセージ・キューからのすべての入力 XML 文書を、DAD ファイル・マッピングに基づいて分解します。 XML エレメントと属性の内容は、指定の DB2 UDB 表に保管されます。

dxxmqShredAll() 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。このストアード・プロシージャーでは、DAD で、結合条件に指定されている列が、既存の表の主キー/外部キーの関係 (リレーションシップ) に対応している必要があります。 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。

構文:

```
dxxmqShredAll(varchar(48)   serviceName,     /* input */
              varchar(48)   policyName,     /* input */
              varchar(80)   dadFileName,    /* input */
              varchar(20)   status)         /* output */
```

パラメーター:

表 81. dxxmqShredAll() パラメーター

パラメーター	説明	IN/OUT パラメーター
serviceName	論理 MQSeries 宛先を含むストリング。メッセージはここに送られます。*serviceName* を指定する場合は、それで AMT.XML リポジトリー・ファイルに定義されているサービス・ポイントを参照します。 *serviceName* を指定しないと、DB2.DEFAULT.SERVICE が使用されます。 *serviceName* の最大サイズは 48 バイトです。	IN
policyName	メッセージの処理に使用される、MQSeries AMI サービス・ポリシーを含んでいるストリング。 *policyName* を指定する場合は、それで AMT.XML リポジトリー・ファイルに定義されているポリシーを参照します。 *policyName* を指定しないと、デフォルトの DB2.DEFAULT.POLICY が使用されます。 *policyName* の最大サイズは 48 バイトです。	IN
dadFileName	DAD ファイルの名前。最大サイズは 80 バイトです。	IN
status	ストアード・プロシージャーが正常に実行されたかどうかを示し、エラー・コードが生成された場合はそのコード、および、メッセージ・キューに送られたかまたはそこから受け取られた XML 文書数を示す、戻されたテキストとコード。	OUT

例:

以下の断片は dxxmqShredAll() の呼び出しの例です。

```
#include "dxx.h"
#include "dxxrc.h"

EXEC SQL INCLUDE SQLCA;
      EXEC SQL BEGIN DECLARE SECTION;
        char        serviceName[48];   /* name of the MQ/AMI service */
```

```
char        policyName[48];    /* name of the MQ/AMI policy */
char        dadFileName[80];   /* name of the DAD file */
char        status[20];        /* status code or message */
short       serviceName_ind;
short       policyName_ind;
short       dadFileName_ind;
short       status_ind;
EXEC SQL END DECLARE SECTION;

      /* initialize host variable and indicators */
strcpy(dadFileName,"e:/dxx/samples/dad/getstart_xcollection.dad");
strcpy(serviceName, "myService");
strcpy(policyName, "myPolicy");
status[0]=¥0;
serviceName_ind=0;
policyName_ind=0;
dadFileName_ind=0;
status_ind=-1;

  /* Call the store procedure */
EXEC SQL CALL dxxmqShredAll(:serviceName:serviceName_ind,
                   :policyName:policyName_ind,
                   :dadFileName:dadFileName_ind,
                   :status:status_ind);
```

関連概念:

- 231 ページの『MQSeries 用の XML Extender ストアード・プロシージャーと関数 - 概要』

関連資料:

- ix ページの『構文図の読み方』
- 335 ページの『付録 C. XML Extender の制限』

dxxmqShredCLOB ストアード・プロシージャー

目的:

これは、DAD ファイル・マッピングに基づいて、メッセージ・キューからの入力 XML 文書を分解し、その XML エレメントと属性の内容を指定の DB2 UDB 表に保管します。入力文書タイプは XMLCLOB です。

dxxmqShredCLOB では、DAD ファイルで指定されたすべての表が存在し、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る必要があります。このストアード・プロシージャーでは、DAD の結合条件に指定されている列が、既存の表の主キー/外部キーの関係 (リレーションシップ) に対応していることが必要です。 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。

構文:

```
dxxmqShredCLOB(varchar(48)   serviceName,    /* input */
               varchar(48)   policyName,     /* input */
               varchar(80)   dadFileName,    /* input */
               varchar(10)   status)         /* output */
```

パラメーター:

表 82. *dxxmqShredCLOB* パラメーター

パラメーター	説明	IN/OUT パラメーター
serviceName	論理 MQSeries 宛先を含むストリング。メッセージはここに送られます。*serviceName* を指定する場合は、それで AMT.XML リポジトリー・ファイルに定義されているサービス・ポイントを参照します。*serviceName* を指定しないと、DB2.DEFAULT.SERVICE が使用されます。 *serviceName* の最大サイズは 48 バイトです。	IN
policyName	メッセージの処理に使用される、MQSeries AMI サービス・ポリシーを含んでいるストリング。 *policyName* を指定する場合は、それで AMT.XML リポジトリー・ファイルに定義されているポリシーを参照します。 *policyName* を指定しないと、デフォルトの DB2.DEFAULT.POLICY が使用されます。 *policyName* の最大サイズは 48 バイトです。	IN
dadFileName	DAD ファイルの名前。最大サイズは 80 バイト。	IN
status	ストアード・プロシージャーが正常に実行されたかどうかを示し、エラー・コードが生成された場合はそのコード、および、メッセージ・キューに送られたかまたはそこから受け取られた XML 文書数を示す、戻されたテキストとコード。	OUT

関連概念:

- 231 ページの『MQSeries 用の XML Extender ストアード・プロシージャーと関数 - 概要』

関連資料:

- ix ページの『構文図の読み方』
- 335 ページの『付録 C. XML Extender の制限』

dxxmqShredAllCLOB ストアード・プロシージャー

目的:

これは、DAD ファイル・マッピングに基づいて、メッセージ・キューからの入力 XML 文書を分解し、その XML エレメントと属性の内容を指定の DB2 UDB 表に保管します。

dxxmqShredAllCLOB では、DAD ファイルで指定されたすべての表が存在し、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る必要があります。このストアード・プロシージャーでは、DAD の結合条件に指定されている

列が、既存の表の主キー/外部キーの関係 (リレーションシップ) に対応していることが必要です。 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。

構文:

```
dxxmqShredCLOB(varchar(48)   serviceName,    /* input */
               varchar(48)   policyName,     /* input */
               varchar(80)   dadFileName,    /* input */
               varchar(10)   status)         /* output */
```

パラメーター:

表 83. dxxmqShredAllCLOB パラメーター

パラメーター	説明	IN/OUT パラメーター
serviceName	論理 MQSeries 宛先を含むストリング。メッセージはここに送られます。*serviceName* を指定する場合は、それで AMT.XML リポジトリー・ファイルに定義されているサービス・ポイントを参照します。 *serviceName* を指定しないと、DB2.DEFAULT.SERVICE が使用されます。 *serviceName* の最大サイズは 48 バイトです。	IN
policyName	メッセージの処理に使用される、MQSeries AMI サービス・ポリシーを含んでいるストリング。 *policyName* を指定する場合は、それで AMT.XML リポジトリー・ファイルに定義されているポリシーを参照します。 *policyName* を指定しないと、デフォルトの DB2.DEFAULT.POLICY が使用されます。 *policyName* の最大サイズは 48 バイトです。	IN
dadFileName	DAD ファイルの名前。最大サイズは 80 バイトです。	IN
status	ストアード・プロシージャーが正常に実行されたかどうかを示し、エラー・コードが生成された場合はそのコード、および、メッセージ・キューに送られたかまたはそこから受け取られた XML 文書数を示す、戻されたテキストとコード。	OUT

関連資料:

- 335 ページの『付録 C. XML Extender の制限』

dxxmqInsert ストアード・プロシージャー

目的:

これは、メッセージ・キューからの入力 XML 文書を分解、つまり細分化して、データを新規あるいは既存のデータベース表に保管します。 dxxmqInsert は、DAD ファイル名ではなくコレクション名を使用して、データの保管方法を判別します。

構文:

```
dxxmqInsert(varchar(48)  serviceName,      /* input */
            varchar(48)   policyName,      /* input */
            varchar(80)   collectionName,  /* input */
            varchar(20)   status)          /* output */
```

パラメーター:

表 84. dxxmqInsert() パラメーター

パラメーター	説明	IN/OUT パラメーター
serviceName	論理 MQSeries 宛先を含むストリング。メッセージはここに送られます。*serviceName* を指定する場合は、それで AMT.XML リポジトリー・ファイルに定義されているサービス・ポイントを参照します。*serviceName* を指定しないと、DB2.DEFAULT.SERVICE が使用されます。 *serviceName* の最大サイズは 48 バイトです。	IN
policyName	メッセージの処理に使用される、MQSeries AMI サービス・ポリシーを含んでいるストリング。*policyName* を指定する場合は、それで AMT.XML リポジトリー・ファイルに定義されているポリシーを参照します。 *policyName* を指定しないと、デフォルトの DB2.DEFAULT.POLICY が使用されます。 *policyName* の最大サイズは 48 バイトです。	IN
collectionName	使用可能な XML コレクションの名前最大サイズは 80 バイトです。	IN
status	ストアード・プロシージャーが正常に実行されたかどうかを示し、エラー・コードが生成された場合はそのコード、および、メッセージ・キューに送られたかまたはそこから受け取られた XML 文書数を示す、戻されたテキストとコード。	OUT

例:

以下の断片的な例では、dxxmqInsert() 呼び出しは、*serviceName* の指定するメッセージ・キューから入力 XML 文書 order1.xml を取り出し、その文書を分解し、その文書を使用可能にした DAD ファイルに指定されている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          serviceName[48];
              char          policyName[48];
              char          collection[80];     /* name of an XML collection */
              char          status[10];

              short                 serviceName_ind;
              short                 policyName_ind;
              short                 collection_ind;
              short                 status_ind;
              EXEC SQL END DECLARE SECTION;

              /* initialize host variable and indicators */
              strcpy(serviceName, "myService");
              strcpy(policyName, "myPolicy");
              strcpy(collection,"sales_ord")
              status[0]=¥0;
              serviceName_ind = 0;
              policyName_ind = 0;
              collection_ind = 0;
              status_ind = -1;

              /* Call the store procedure */
              EXEC SQL CALL dxxmqInsert(:serviceName:serviceName_ind,
                              :policyName:policyName_ind,
                              :collection:collection_ind,
                              :status:status_ind);
```

関連概念:

- 231 ページの『MQSeries 用の XML Extender ストアード・プロシージャーと関数 - 概要』

関連資料:

- ix ページの『構文図の読み方』
- 335 ページの『付録 C. XML Extender の制限』

dxxmqInsertCLOB ストアード・プロシージャー

目的:

これは、メッセージ・キューからの入力 XML 文書を分解、つまり細分化して、データを新規あるいは既存のデータベース表に保管します。 dxxmqInsertCLOB は、DAD ファイル名ではなくコレクション名を使用して、データの保管方法を判別します。入力文書タイプは XMLCLOB です。

構文:

```
dxxmqInsertCLOB(varchar(48)  serviceName,  /* input */
            varchar(48)   policyName,      /* input */
            varchar(80)   collectionName,  /* input */
            varchar(20)   status)          /* output */
```

パラメーター:

表 85. dxxmqInsertCLOB() パラメーター

パラメーター	説明	IN/OUT パラメーター
serviceName	論理 MQSeries 宛先を含むストリング。メッセージはここに送られます。*serviceName* を指定する場合は、それで AMT.XML リポジトリー・ファイルに定義されているサービス・ポイントを参照します。*serviceName* を指定しないと、DB2.DEFAULT.SERVICE が使用されます。 *serviceName* の最大サイズは 48 バイトです。	IN
policyName	メッセージの処理に使用される、MQSeries AMI サービス・ポリシーを含んでいるストリング。*policyName* を指定する場合は、それで AMT.XML リポジトリー・ファイルに定義されているポリシーを参照します。 *policyName* を指定しないと、デフォルトの DB2.DEFAULT.POLICY が使用されます。 *policyName* の最大サイズは 48 バイトです。	IN
collectionName	使用可能な XML コレクションの名前	IN
status	ストアード・プロシージャーが正常に実行されたかどうかを示し、エラー・コードが生成された場合はそのコード、および、メッセージ・キューに送られたかまたはそこから受け取られた XML 文書数を示す、戻されたテキストとコード。	OUT

例:

以下の断片的な例では、dxxmqInsertCLOB() 呼び出しは、*serviceName* の指定するメッセージ・キューから入力 XML 文書 `order1.xml` を取り出し、その文書を分解し、その文書を使用可能にした DAD ファイルに指定されている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          serviceName[48];
              char          policyName[48];
              char          collection[48];     /* name of an XML collection */
              char          status[10];

              short                 serviceName_ind;
```

```
short                policyName_ind;
short                collection_ind;
short                status_ind;
EXEC SQL END DECLARE SECTION;

/* initialize host variable and indicators */
strcpy(serviceName, "myService");
strcpy(policyName, "myPolicy");
strcpy(collection,"sales_ord")
status[0] = ¥0;
serviceName_ind = 0;
policyName_ind = 0;
collection_ind = 0;
status_ind = -1;

/* Call the store procedure */
EXEC SQL CALL dxxmqInsertCLOB(:serviceName:serviceName_ind;
                :policyName:policyName_ind,
                :collection:collection_ind,
                :status:status_ind);
```

関連資料:

• 335 ページの『付録 C. XML Extender の制限』

dxxmqInsertAll ストアード・プロシージャー

目的:

これは、メッセージ・キューからのすべての入力 XML 文書を分解、つまり細分化して、データを新規あるいは既存のデータベース表に保管します。 dxxmqInsertAll は、DAD ファイル名ではなくコレクション名を使用して、データの保管方法を判別します。

構文:

```
dxxmqInsertAll(varchar(48)  serviceName,      /* input */
               varchar(48)   policyName,      /* input */
               varchar(48)   collectionName,  /* input */
               varchar(20)   status)          /* output */
```

パラメーター:

表 86. dxxmqInsertAll() パラメーター

パラメーター	説明	IN/OUT パラメーター
serviceName	論理 MQSeries 宛先を含むストリング。メッセージはここに送られます。*serviceName* を指定する場合は、それで AMT.XML リポジトリー・ファイルに定義されているサービス・ポイントを参照します。*serviceName* を指定しないと、DB2.DEFAULT.SERVICE が使用されます。 *serviceName* の最大サイズは 48 バイトです。	IN

表 86. *dxxmqInsertAll()* パラメーター *(続き)*

パラメーター	説明	IN/OUT パラメーター
policyName	メッセージの処理に使用される、MQSeries AMI サービス・ポリシーを含んでいるストリング。 *policyName* を指定する場合は、それで AMT.XML リポジトリー・ファイルに定義されているポリシーを参照します。 *policyName* を指定しないと、デフォルトの DB2.DEFAULT.POLICY が使用されます。 *policyName* の最大サイズは 48 バイトです。	IN
collectionName	使用可能な XML コレクションの名前最大サイズは 80 バイトです。	IN
status	ストアード・プロシージャーが正常に実行されたかどうかを示し、エラー・コードが生成された場合はそのコード、および、メッセージ・キューに送られたかまたはそこから受け取られた XML 文書数を示す、戻されたテキストとコード。	OUT

例:

以下の断片的な例では、dxxmqInsertAll 呼び出しは、*serviceName* で定義しているメッセージ・キューからすべての入力 XML 文書を取り出し、それらの文書を分解し、その文書を使用可能にした DAD ファイルに指定されている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          serviceName[48];
              char          policyName[48];
              char          collection[80];     /* name of an XML collection */
              char          status[10];

              short                 serviceName_ind;
              short                 policyName_ind;
              short                 collection_ind;
              short                 status_ind;
              EXEC SQL END DECLARE SECTION;

              /* initialize host variable and indicators */
              strcpy(serviceName, "myService");
              strcpy(policyName, "myPolicy");
              strcpy(collection,"sales_ord");
              status[0]='¥0';
              serviceName_ind = 0;
              policyName_ind = 0;
              collection_ind = 0;
              status_ind = -1;

              /* Call the store procedure */
```

```
EXEC SQL CALL dxxmqInsertAll(:serviceName:serviceName_ind,
                             :policyName:policyName_ind,
                             :collection:collection_ind,
                             :status:status_ind);
```

関連概念:

- 231 ページの『MQSeries 用の XML Extender ストアード・プロシージャーと関数 - 概要』

関連資料:

- ix ページの『構文図の読み方』
- 335 ページの『付録 C. XML Extender の制限』

dxxmqInsertAllCLOB ストアード・プロシージャー

目的:

これは、メッセージ・キューからのすべての入力 XML 文書を分解、つまり細分化して、データを新規あるいは既存のデータベース表に保管します。
dxxmqInsertAllCLOB は、DAD ファイル名ではなくコレクション名を使用して、データの保管方法を判別します。

構文:

```
dxxmqInsertAllCLOB(varchar(48)  serviceName,  /* input */
                   varchar(48)  policyName,   /* input */
                   varchar(48)  collectionName,  /* input */
                   varchar(20)  status)       /* output */
```

パラメーター:

表 87. dxxmqInsertAllCLOB() パラメーター

パラメーター	説明	IN/OUT パラメーター
serviceName	論理 MQSeries 宛先を含むストリング。メッセージはここに送られます。*serviceName* を指定する場合は、それで AMT.XML リポジトリー・ファイルに定義されているサービス・ポイントを参照します。*serviceName* を指定しないと、DB2.DEFAULT.SERVICE が使用されます。 *serviceName* の最大サイズは 48 バイトです。	IN

表 87. *dxxmqInsertAllCLOB()* パラメーター *(続き)*

パラメーター	説明	IN/OUT パラメーター
policyName	メッセージの処理に使用される、MQSeries AMI サービス・ポリシーを含んでいるストリング。 *policyName* を指定する場合は、それで AMT.XML リポジトリー・ファイルに定義されているポリシーを参照します。 *policyName* を指定しないと、デフォルトの DB2.DEFAULT.POLICY が使用されます。 *policyName* の最大サイズは 48 バイトです。	IN
collectionName	使用可能な XML コレクションの名前	IN
status	ストアード・プロシージャーが正常に実行されたかどうかを示し、エラー・コードが生成された場合はそのコード、および、メッセージ・キューに送られたかまたはそこから受け取られた XML 文書数を示す、戻されたテキストとコード。	OUT

例:

以下の断片的な例では、dxxmqInsertAllCLOB 呼び出しは、*serviceName* で定義しているメッセージ・キューからすべての入力 XML 文書を取り出し、それらの文書を分解し、その文書を使用可能にした DAD ファイルに指定されている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          serviceName[48];
              char          policyName[48];
              char          collection[48];     /* name of an XML collection */
              char          status[10];

              short                 serviceName_ind;
              short                 policyName_ind;
              short                 collection_ind;
              short                 status_ind;
              EXEC SQL END DECLARE SECTION;

              /* initialize host variable and indicators */
              strcpy(serviceName, "myService");
              strcpy(policyName, "myPolicy");
              strcpy(collection,"sales_ord")
              status[0] = '¥0';
              serviceName_ind = 0;
              policyName_ind = 0;
              collection_ind = 0;
              status_ind = -1;

              /* Call the store procedure */
```

```
EXEC SQL CALL dxxmqInsertAllCLOB(:serviceName:serviceName_ind;
                :policyName:policyName_ind,
                :collection:collection_ind,
                :status:status_ind);
```

関連資料:

- 335 ページの『付録 C. XML Extender の制限』

第 12 章 Extensible stylesheet language transformation (XSLT)

XSLT スタイルシートを使用した HTML 文書の作成

XSLT (Extensible stylesheet language transformation) は、XML 文書内の各エレメントに対してフォーマット設定規則を適用するために使用できる一連のマークアップから構成されます。 XSL は、検出したエレメントに基づいて、XML 文書の内容に各種のスタイル規則を適用することによって機能します。設計上は、XSLT スタイルシートは通常の XML 文書です。

XSLT は、もともとはページ・レイアウトのために作成されたものですが、現在では、多様な方面で使用されています。たとえば、汎用の変換ツールとして、文書内容を再編成するシステムとして、または、単一のソースから HTML、WAP および SVG などの複数の結果を生成する方法として使用できます。

XSLT は、XML 処理と、HTML などもっと広く使用されている言語との重要な掛け橋になっています。 XSLT を使用すると、XML タグを除去したり、置換することによって、XML 構造をその他のデータ・タイプに変換することができます。また、情報の順序を変更したり、特定の情報を抽出したり、あるいは、情報をソートしたりする手段として使用することもできます。

前提条件:

スタイルシートを使用して HTML 文書を作成するには、以下の作業を行う必要があります。

1. 結果表に XML ファイルを作成します。
2. スタイルシートを作成します。

XSLTransformToFile または XSLTransformToClob を使用して HTML ファイルを作成できます。この出力ファイルは、DB2 UDB サーバーに書き込むか、テキスト・エディターのコマンド行から実行できます。

手順:

DB2 UDB サーバーに HTML ファイルを作成するには、以下の構文を入力してください。

```
SELECT XSLTransformToFile( CAST(doc AS CLOB(4k)),
    '$dxx_install$¥samples¥db2xml¥xslt¥getstart.xsl',
         0,
   '$dxx_install$¥samples¥db2xml¥html¥getstart.html')
 FROM RESULT_TAB
```

ここで、*$dxx_install$* は DB2 XML Extender をインストールしたディレクトリーです。

コマンド行から HTML を作成するには、任意のテキスト・エディターをオープンし、次のコマンドを入力してください。

getstart_xslt.cmd

関連資料:

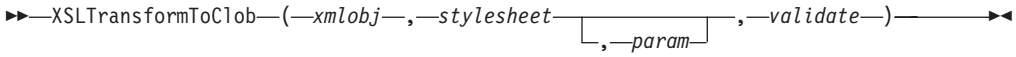
- 286 ページの『XSLTransformToClob() ストアード・プロシージャー』
- 287 ページの『XSLTransformToFile() ストアード・プロシージャー』

XSLTransformToClob() ストアード・プロシージャー

目的:

XSLTransformToClob() は、XML 文書を CLOB ロケーターとして、スタイルシートを CLOB として読み取るか、あるいはファイルから読み取り、その文書を CLOB として戻します。

構文:



パラメーター:

パラメーター	データ・タイプ	説明
xmlobj	CLOB	XML 文書
stylesheet	CLOB、 VARCHAR	スタイルシート スタイルシート入力ファイルのロケーションと名前
param	CLOB VARCHAR	XML パラメーター文書 XML パラメーター・ファイルのロケーションと名前
validate	INTEGER	xmlobj の妥当性検査の使用可能 (1) または使用不可 (0)

結果:

XSLTransformToClob() は、正常に終了すると CLOB タイプのデータを戻します。

例:

次の例は、サンプル表を作成して、2 つの入力ファイル getstart.xml および getstart.xsl をデータベースに保管します。データベースは、XML Extender 用に使用可能になっている必要があります。

```
CREATE TABLE xslt_tab(xmlobj CLOB(4k), stylesheet CLOB(4k))
INSERT INTO xslt_tab(xmlobj, stylesheet) VALUES(
 DB2XML.XMLCLOBFromFile('c:¥dxx_installsamples¥db2xml¥xml¥getstart.xml
'),
DB2XML.XMLCLOBFromFile('c:¥dxx_installsamples¥db2xml¥xslt¥getstart.xsl
'))
```

例 1: 次の例は、作成した表を使用して、XML 文書を HTML 文書にトランスフォームします。

```
SELECT XSLTransformToClob(xmlobj, stylesheet)
FROM xslt_tab
```

例 2: この例は、スタイルシート・ファイルを使用して、XML 文書を HTML 文書にトランスフォームします。

```
SELECT XSLTransformToClob( xmlobj,
  c:¥dxx_installsamples¥db2xml¥xslt¥getstart.xsl
')
  FROM xslt_tab
```

例 3: この例では、出力はさらに別のパラメーターで変えられています。XML パラメーター文書では、ネームスペースを定義しておかなければなりません。パラメーターは、<*param*> エレメントに入れておく必要があります。また、対応する値を *value* 属性か <*param*> エレメントの内容として指定することもできます。

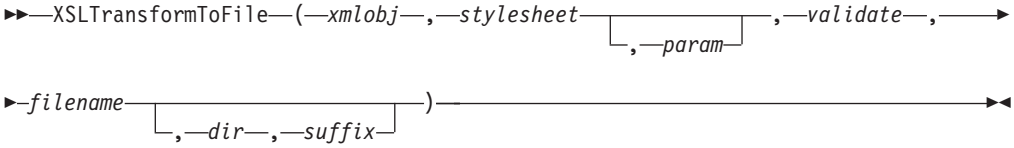
```
c:¥dxx_install¥samples¥db2xml¥xml¥getstart_xslt_param.xml:
<?xml version="1.0"?>
<params xmlns="http://www.ibm.com.XSLtransformParameters">
   <param name="noShipments" value="true"/>
   <param name="headline">The customers...</param>
</params>

SELECT XSLTranfsormToClob( xmlobj, stylesheet, param, 1)
FROM xslt_tab
```

XSLTransformToFile() ストアード・プロシージャー

目的:

これは、XML 文書を CLOB として、スタイルシートを CLOB として、またはファイルから読み取ります。 XSLTransformToFile() ユーザー定義関数 (UDF) は、スタイルシートおよび XML 文書からの結果をファイルに書き込みます。パラメーターとしてディレクトリーとファイル拡張子が指定されると、UDF は、このディレクトリー内にユニークのファイル名を付けたファイルを作成します。

構文:



パラメーター:

表 88. XSLTransformDir() パラメーター記述

パラメーター	データ・タイプ	説明
xmlobj	CLOB	XML 文書
stylesheet	CLOB	スタイルシート
	VARCHAR	スタイルシート入力ファイルのロケーションと名前
param	VARCHAR	XML パラメーター文書
	VARCH	XML パラメーター・ファイルのロケーションと名前

表 88. XSLTransformDir() パラメーター記述 (続き)

パラメーター	データ・タイプ	説明
validate	INTEGER	xmlobj の妥当性検査の使用可能 (1) または使用不可 (0)
filename	VARCHAR	出力ファイルの名前
dir	VARCHAR	出力ファイルのディレクトリー
suffix	VARCHAR	出力ファイルのサフィックス

結果:

XSLTransformToFile() は、作成したファイル名の VARCHAR を戻します。

例:

次の例は、サンプル表を作成し、2 つのファイルを getstart.xml および getstart.xsl 表に保管します。 サンプル表を作成するには、DB2 UDB データベースが XML Extender 用に使用可能になっている必要があります。

```
CREATE TABLE xslt_tab(xmlobj CLOB(4k), stylesheet CLOB(4k))

INSERT INTO xslt_tab(xmlobj, stylesheet) VALUES(
DB2XML.XMLCLOBFromFile('$dxx_install$¥samples¥db2xml¥xml¥getstart.xml
'),
DB2XML.XMLCLOBFromFile('$dxx_install$¥samples¥db2xml¥xslt¥getstart.xsl
'))
```

ここで、*$dxx_install$* は DB2 XML Extender をインストールしたディレクトリーです。

例 1: この例は、XML 文書を HTML 文書にトランスフォームし、作成した文書を指定されたファイルに書き込みます。

```
SELECT XSLTransformFile( xmlobj, stylesheet,
   '$dxx_install$samples¥db2xml¥html¥getstart.html

  FROM xslt_tab
```

ここで、*$dxx_install$* は DB2 XML Extender をインストールしたディレクトリーです。

例 2: この例は、スタイルシート・ファイルを使用して HTML 文書を書きます。妥当性検査は使用可能になっていますが、結果は同じです。この機能は、トランスフォーメーション処理で XML スキーマからのデフォルト値を含めるために必要です。パラメーターは指定しません。ファイル名は、UDF が生成します。

```
SELECT XSLTransformToFile( xmlobj,
   '$dxx_install$¥samples¥db2xml¥xslt¥getstart.xsl',
   '$dxx_install$¥samples¥db2xml¥html¥getstart.html')
     FROM xslt_tab
```

ここで、*$dxx_install$* は DB2 XML Extender をインストールしたディレクトリーです。

例 3: この例では、出力はさらに別のパラメーターで変えられています。XML パラメーター文書では、ネームスペースを定義しておかなければなりません。パラメーターは、*<param>* エレメントに入れておく必要があります。また、対応する値を *value* 属性か *<param>* エレメントの内容として指定することもできます。

```
$dxx_install$¥samples¥db2xml¥xml¥getstart_xslt_param.xml:', 'html')
<?xml version="1.0"?>
<params xmlns="http://www.ibm.com.XSLtransformParameters">
   <param name="noShipments" value="true"/>
   <param name="headline">The customers...</param>
</params>
```

ここで、*$dxx_install$* は DB2 XML Extender をインストールしたディレクトリーです。

例 4: この例は、スタイルシート・ファイルを使用して HTML 文書をファイルに書き込み、表内の追加列に各行のためのファイル名を保管します。

```
UPDATE TABLE xslt_tab ADD COLUMN filename VARCHAR(512)
UPDATE TABLE xslt_tab SET filename =
  XSLTransformToFile(xmlobj,stylesheet, param, 1,
    '$dxx_install$samples¥db2xml¥html
          ', 'html')
  FROM xslt_tab
```

ここで、*$dxx_install$* は DB2 XML Extender をインストールしたディレクトリーです。

第 13 章 XML Extender の管理サポート表

データベースが使用可能になると、 DTD リポジトリー表 (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。

DTD 参照表

XML Extender 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 リポジトリー表 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。 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 89 に示すような列が含まれます。

表 89. 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) 主キー。これは DTD の識別に使われます。 DTD 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。この情報はオプショナルです。
CREATOR	VARCHAR(128)	最初にデータ挿入を行ったユーザー ID。この列はオプショナルです。
UPDATOR	VARCHAR(128)	最後に更新を行ったユーザー ID。この列はオプショナルです。

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 (XML_USAGE)

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。 XML_USAGE 表のスキーマ名は DB2XML、主キーは (*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。 XML_USAGE 表内の列が 292 ページの表 90 に示されています。

表 90. 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	DTD_REF に挿入された DTD を、DAD ファイルに指定されている DTD と関連付けるストリング。この値は、DAD 内の DTDID エレメントの値と一致する必要があります。この列は外部キーです。
DAD	XML 列または XML コレクションに関連付けられた DAD ファイルの内容。
access_mode	どのアクセス・モードを使用するかを指定します。XML コレクションならば 1、XML 列ならば 0。
default_view	デフォルト・ビュー名があれば、それを保管します。
trigger_suffix	非 NULL。ユニークなトリガー名に使用。
validation	値が 1 の場合は妥当性検査を行い、0 の場合は妥当性検査をスキップします。

XML_USAGE 表の項目に追加、変更、または削除を行ってはなりません。この表は XML Extender だけが内部的に使用します。

第 14 章 トラブルシューティング

XML Extender のトラブルシューティング

プログラム内のすべての組み込み SQL ステートメント、および (DB2 UDB XML Extender のユーザー定義関数 (UDF) を呼び出すものを含む) すべての DB2 UDB 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 UDB 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、SQLSTATE 情報およびエラー・メッセージを含め、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、 IBM 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 Extender には、XML Extender 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。

このセクションでは、トレース機能、エラー・コードおよびメッセージについて説明します。

関連資料:

- 295 ページの『XML Extender のための SQLSTATE コードと関連メッセージ番号』
- 301 ページの『XML Extender のメッセージ』
- 294 ページの『トレースの停止』
- 293 ページの『XML Extender 用トレースの開始』

XML Extender 用トレースの開始

目的:

XML Extender 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・ファイルを入れる既存のディレクトリーの名前を指定して、**dxxtrc** に `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 UDB 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。 トレース・ファイルにはサイズ制限がありません。

構文:

トレースの開始:

パラメーター:

表 91. トレース・パラメーター

パラメーター	説明
trace_directory	dxx*INSTANCE*.trc が入っている既存のパスと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、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/dxx_install/log ディレクトリーに保管されます。

```
dxxtrc on /home/db2inst1/dxx_install/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。

勧告: トレース・ログ・ファイルのサイズは制限がなく、トレースの実行はパフォーマンスに影響する可能性があるため、実稼働環境ではトレースをオフにしてください。

構文:

トレースの停止:

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
```

XML Extender UDF の戻りコード

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 Extender 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 UDB に渡します。

DB2 XML Extender UDF を呼び出す組み込み SQL ステートメントおよび DB2 UDB CLI 呼び出しによって、これらの UDF にユニークな SQLCODE メッセージや SQLSTATE 値が戻ることもありますが、DB2 UDB は、他の組み込み SQL ステートメントまたは DB2 UDB 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 UDB XML Extender UDF を開始しない、組み込み SQL ステートメントまたは DB2 UDB CLI 呼び出しの場合と同じです。

XML Extender ストアード・プロシージャーの戻りコード

XML Extender 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。戻りコードと XML Extender エラー・メッセージ番号およびシンボリック定数が突き合わされています。

```
dxx_install/include/dxxrc.h
```

関連資料:

- 295 ページの『XML Extender のための SQLSTATE コードと関連メッセージ番号』

XML Extender のための SQLSTATE コードと関連メッセージ番号

表 92. 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 Extender は、指定されたファイルを開くことができません。
38X01	DXXA072E	XML Extender 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 Extender は、指定されたファイルを検出できませんでした。

表 92. 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
38X02	DXXC002E	XML Extender 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 Extender は、データをファイルに書き込むことができません。
	DXXC011E	XML Extender 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 Extender は、指定されたロケーターを操作できませんでした。
38X05	DXXC005E	ファイル・サイズが XMLVarchar サイズより大きいため、XML Extender 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 Extender 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、複数回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、複数回出現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。
38X09	DXXD002E	パス式の構文が正しくありません。
38X10	DXXG002E	XML Extender は、オペレーティング・システムからメモリーを割り振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 Extender 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)。
	DXXQ060E	XML Extender は、列を使用可能にするよう試みている間にスキーマ ID を検出できませんでした。スキーマ ID は DAD ファイルの schemabindings タグ内にある、nonamespacelocation タグのロケーション属性の値に対応します。
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 92. 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。
	DXXQ056E	指定されたエレメントまたは属性は、外部キーの一部として指定された列にマップすることはできません。外部キーのためのデータ値は主キーのデータ値により決定されます。 XML 文書内の指定されたエレメントまたは属性を表および列にマッピングする必要はありません。
	DXXQ057E	schemabindings と DTD ID タグが DAD ファイルに共に存在することはできません。
	DXXQ058E	schemabindings タグ内の nonamespacelocation タグが DAD ファイル内で欠落しています。
	DXXQ059E	doctype タグは、スキーマの妥当性検査のための DAD ファイルの XCollection タグ内に配置できません。
	DXXQ062E	このエラー状態は普通、指定されたエレメントまたは属性の親 element_node 上の multi_occurrence = YES 指定が欠落していることが原因で発生します。

表 92. 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
	DXXQ063E	文書アクセス定義 (DAD) ファイル内の指定された element_node の multi_occurrence 属性の値が正しくないか、または属性がありません。値は、大文字小文字を区別しない、'yes' または 'no' である必要があります。
	DXXQ064E	結合条件で指定されたキー列はどのエレメントまたは属性ノードにもマップされませんでした。
38X16	DXXG004E	必要パラメーターの NULL 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) ファイル、またはこれをオーバーライドするステートメントの中の 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 Extender は内部エラーを検出しました。
	DXXG006E	CLI を使用中に XML Extender 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。

表 92. 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
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 DXXQ040E	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 Extender 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。
	DXXQ040E	文書アクセス定義 (DAD) ファイル内の指定されたエレメント名が正しくありません。
38X28	DXXQ012E	DAD の処理中に XML Extender は予期したエレメントを検出できませんでした。
	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) ファイルの条件エレメント内の条件が、無効な形式です。
	DXXQ061E	ストリング表記の形式が無効です。ストリングが、日付、時刻、またはタイム・スタンプ値の場合、構文はそのデータ・タイプに準拠していません。
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。

表 92. 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。
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・システムのロケールが、 DB2 UDB 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・システムのロケール設定が、コード・ページ表にありません。
38X41	DXXQ048E	スタイルシートのプロセッサーが内部エラーを戻しました。 XML 文書またはスタイルシートが有効ではない可能性があります。
38X42	DXXQ049E	指定された出力ファイルは、このディレクトリーにすでに存在しています。
38X43	DXXQ050E	UDF にアクセス権がないため、UDF は指定のディレクトリーに出力文書のためのユニーク・ファイル名を作成できませんでした。生成可能なすべてのファイル名が使用されているか、またはディレクトリーが存在しない可能性があります。
38X44	DXXQ051E	入力または出力パラメーターの中に有効な値を持っていないものがあります。
38X45	DXXQ055E	変換操作中に ICU エラーが見つかりました。

XML Extender のメッセージ

DXXA000I　列 <*column_name*> の使用可能化中。お待ちください。

説明: これは通知メッセージです。

ユーザーの処置: アクションは不要です。

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は、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含めてください。

DXXA002I　データベース <*database*> に接続中。

説明: これは通知メッセージです。

ユーザーの処置: アクションは不要です。

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、バックアップからリカバリー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、バックアップからリカバリー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: これは通知メッセージです。

ユーザーの処置: アクションは不要です。

DXXA006I　データベース <*database*> は、正常に使用可能化されました。

説明: これは通知メッセージです。

ユーザーの処置: アクションは不要です。

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 Extender 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、表の更新またはドロップ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。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: アクションは不要です。

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 Extender 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 表へのレコードの挿入が失敗しました。

説明: XML Extender 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 Extender 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*> の変更が失敗しました。

説明: XML Extender が列を使用可能にしようとした時、指定された表を変更できませんでした。

ユーザーの処置: 表の変更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*> の単一の主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 つの主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 Extender 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 Extender 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 Extender 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 Extender 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: アクションは不要です。

DXXA019E　列を使用可能化するのに必要な入力パラメーターが NULL 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 NULL 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チェックしてください。

DXXA020E　列が表 <*table_name*> に見つかりません。

説明: XML Extender 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 Extender 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・ビュー名がユニーク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、ユニークな名前をデフォルト・

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: これは通知メッセージです。

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 Extender 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 Extender で内部エラーが発生しました。

説明: XML Extender 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 Extender が列を使用不可にしようとした時、デフォルト・ビューをドロップできませんでした。

ユーザーの処置: XML Extender の管理ユーザー ID に、デフォルト・ビューのドロップ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 Extender が列を使用不可にしようとした時、指定された表をドロップできませんでした。

ユーザーの処置: XML Extender の管理者ユーザー ID に、表のドロップ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 Extender 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。

- システムはメモリーが不足しています。
- この名前のトリガーがない

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。問題が解決しない場合には、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 Extender 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。

- システムはメモリーが不足しています。
- この名前のトリガーがない

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。問題が解決しない場合には、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 Extender 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。

- システムはメモリーが不足しています。
- この名前のトリガーがない

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。問題が解決しない場合には、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 Extender 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。

- システムはメモリーが不足しています。
- この名前のトリガーがない

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。問題が解決しない場合には、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 NULL にリセットすることができませんでした。

説明: XML Extender 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 NULL に設定できませんでした。

ユーザーの処置: XML Extender の管理者ユーザー ID に、アプリケーション表の変更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 Extender 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 Extender 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 Extender 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 Extender 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 Extender は列 <*column_name*> を正常に使用不可にしました。

説明: これは通知メッセージです。

ユーザーの処置: アクションは不要です。

DXXA035I　XML Extender 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: これは通知メッセージです。

ユーザーの処置: アクションは不要です。

DXXA036I　XML Extender は、データベース <*database*> を正常に使用不可にしました。

説明: これは通知メッセージです。

ユーザーの処置: アクションは不要です。

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 文字未満の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 文字未満の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 文字未満の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 Extender 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 Extender 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 Extender 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。

- DAD ファイルの構文に誤りがある
- システムはメモリーが不足しています。
- 同じ名前のトリガーが別にある

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。問題が解決しない場合には、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 Extender 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。

- DAD ファイルの構文に誤りがある
- システムはメモリーが不足しています。
- 同じ名前のトリガーが別にある

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。問題が解決しない場合には、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 Extender 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。

- DAD ファイルの構文に誤りがある
- システムはメモリーが不足しています。
- 同じ名前のトリガーが別にある

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。問題が解決しない場合には、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 Extender 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。

- DAD ファイルの構文に誤りがある
- システムはメモリーが不足しています。
- 同じ名前のトリガーが別にある

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。問題が解決しない場合には、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 Extender 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。

- システムはメモリーが不足しています。
- この名前のトリガーがない

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。問題が解決しない場合には、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 Extender 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。

- DAD ファイルの構文に誤りがある
- システムはメモリーが不足しています。
- 同じ名前のトリガーが別にある

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。問題が解決しない場合には、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 Extender 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。

- DAD ファイルの構文に誤りがある
- システムはメモリーが不足しています。
- 同じ名前のトリガーが別にある

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。問題が解決しない場合には、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 Extender 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。

- DAD ファイルの構文に誤りがある
- システムはメモリーが不足しています。
- 同じ名前のトリガーが別にある

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。問題が解決しない場合には、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 Extender 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。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: アクションは不要です。

DXXA064I コレクション <*collection_name*> 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: アクションは不要です。

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: 共有ライブラリー db2xml をチェックして、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 Extender は、コレクション <*collection_name*> を正常に使用不可にしました。

説明: これは通知メッセージです。

ユーザーの処置: 応答は必要ありません。

DXXA067I XML Extender は、コレクション <*collection_name*> を正常に使用可能にしました。

説明: これは通知メッセージです。

ユーザーの処置: 応答は必要ありません。

DXXA068I XML Extender は、トレースを正常にオンにしました。

説明: これは通知メッセージです。

ユーザーの処置: 応答は必要ありません。

DXXA069I XML Extender は、トレースを正常にオフにしました。

説明: これは通知メッセージです。

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: アクションは不要です。

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: アクションは不要です。

DXXA072E XML Extender 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 Extender 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 Extender 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 iSeries ユーザー ID であるかどうか確認してください。

DXXA077E ライセンス・キーが無効です。詳しくはサーバーのエラー・ログを参照してください。

説明: ソフトウェア・ライセンスの有効期限が切れているか、ライセンスを入手していません。

ユーザーの処置: サービス提供者に連絡して、新しいソフトウェア・ライセンスを入手してください。

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 Extender 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 Extender 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 Extender 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 Extender 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 Extender 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 Extender 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 UDB 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 Extender 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: ファイルが共用アクセスに違反しているか、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 Extender 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 Extender 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。

DXXC013E　抽出 UDF の結果が UDF 戻りタイプのサイズ制限を超えています。

説明: 抽出 UDF が戻すデータは、その UDF の戻りタイプのサイズ制限内である必要があります。サイズ制限については、本書「DB2 UDB XML Extender 管理およびプログラミング・ガイド」に定義が記載されています。たとえば、extractVarchar の結果は 4000 バイト (終了 NULL を含めて) 未満でなければなりません。

ユーザーの処置: 戻りタイプのサイズ制限がもっと大きい抽出 UDF を使用してください。制限は、extractChar() では 254 バイト、extractVarchar() では 4 KB、extractClob() では 2 GB です。

DXXD000E　無効な XML 文書がリジェクトされました。

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 によってチェックしてください。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、複数回出現

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。 NULL 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*>、および行 <*line_number*> で内部エラーが発生しました。

説明: XML Extender 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 Extender は、オペレーティング・システムからメモリーを割り振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 NULL パラメーター。

説明: 必要パラメーターの NULL 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、必要パラメーターをすべてチェックしてください。

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 Extender 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 UDB 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・システムのロケールが、 DB2 UDB 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・システムのロケール設定を修正して、 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・システムのロケール設定が、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・システムのロケール設定を修正して、 DB2 を再起動してください。

DXXG017E *XML_Extender_constant* **の制限が、ビルド** *build_ID*、**ファイル** *file_name*、**および行** *line_number* **で超過しました。**

説明: アプリケーションで制限表の値を超えているかどうかを、本書「XML Extender 管理とプログラミングのガイド」を参照して調べてください。制限を超えていない場合は、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は、すべての出力メッセージ、トレース・ファイル、および問題の再現方法 (入力 DAD、XML 文書、表定義など) についての説明を必ず含めてください。

ユーザーの処置: サーバーのオペレーティング・システムのロケール設定を修正して、DB2 を再起動してください。

DXXM001W DB2 UDB エラーが発生しました。

説明: DB2 が指定されたエラーを検出しました。

ユーザーの処置: 付随するメッセージでさらに詳細な説明が提供されていないか調べ、使用しているオペレーティング・システムの「DB2 UDB メッセージおよびコード」の資料を参照してください。

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 文書を保持するストレージ・スペースを生成できません。

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: アクションは不要です。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。

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 UDB 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 Extender 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チェックしてください。 DAD DTD に関する XML Extender 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 Extender 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: 結果表をドロップします。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: アクションは不要です。

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　<*table_name*> の列 <*column_name*> は、タイプが <*type_name*> であることが必要です。

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: アクションは不要です。

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 構文変更です。

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 UDB SQL ID ではありません。

ユーザーの処置: DB2 UDB 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 Extender 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 Extender 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 複数回出現のある <*element*> を生成できません。

説明: エレメント・ノードとその子孫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、そのノードの子のいずれかに RDB_node を作成することによって DAD を修正してください。

DXXQ038E SQL ステートメントが長すぎます: SQL_statement

説明: DAD の <*SQL_stmt*> エレメントに指定されている SQL ステートメントが、許可されているバイト数を超えています。

ユーザーの処置: SQL ステートメントの長さを、Windows および UNIX では 32765 バイト以下、OS/390 および iSeries では 16380 バイト以下にしてください。

DXXQ039E DAD ファイル内の表に指定された列が多すぎます。

説明: 分解または RDB 合成のために使用される DAD ファイルには、同じ表内のユニークの列を指定する text_node エレメントと attribute_node エレメントを 100 個までしか入れられません。

ユーザーの処置: 同じ表内のユニークの列を参照する text_node と attribute_node のエレメント合計数を 100 以下に減らしてください。

DXXQ040E DAD ファイル内のエレメント名 <*element_name*> が無効です。

説明: 文書アクセス定義 (DAD) ファイル内の指定されたエレメント名が正しくありません。

ユーザーの処置: エレメント名が DAD ファイルに正確に入力されているか、確認してください。DAD ファイルのための DTD を調べてください。

DXXQ041W XML 文書が正常に生成されました。指定されている 1 つ以上のオーバーライド・パスは無効であり、無視されます。

説明: オーバーライド・パスはただ 1 つだけ指定してください。

ユーザーの処置: エレメント名が DAD ファイルに正確に入力されているか、確認してください。DAD ファイルのための DTD を調べてください。

DXXQ043E 属性 <attr_name> がエレメント <elem_name> の下で見つかりません。

説明: 属性 <attr_name> が、エレメント <elem_name> またはその子エレメントの 1 つにありません。

ユーザーの処置: XML 文書で DAD が必要とするところには、属性が入っているようにしてください。

DXXQ044E エレメント <elem_name> に上位エレメント <ancestor> がありません。

説明: DAD によれば、<ancestor は <elem_name> の上位エレメントです。この XML 文書には、1 つ以上の <elem_name> で、このような上位エレメントがありません。

ユーザーの処置: XML 文書内のエレメントのネストが、対応する DAD における指定に準拠しているか、確認してください。

DXXQ045E エレメント <elem_name> の下のサブツリーは、 <attrib_name> と命名された複数の属性を持っています。

説明: XML 文書内の <elem_name> 下のサブツリーには、属性 <attrib_name> の複数のインスタンスが入っています。この属性は、DAD に従って、同じ行に分解されることになります。分解されるエレメントまたは属性は、ユニークな名前をもっている必要があります。

ユーザーの処置: サブツリー内のエレメントまたは属性にユニークな名前が付いているか、確認してください。

DXXQ046W 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: アクションは不要です。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。

DXXQ047E 行 <mv>linenumber</mv> 列 *colnumber*: *msg* でパーサー・エラーが発生しました

説明: パーサーは、報告されたエラーのために、文書を解析できませんでした。

ユーザーの処置: 必要ならば、XML の指定を調べて、文書内のエラーを訂正してください。

DXXQ048E 内部エラー - トレース・ファイルを参照してください。

説明: スタイルシートのプロセッサーが内部エラーを戻しました。 XML 文書またはスタイルシートが有効ではない可能性があります。

ユーザーの処置: XML 文書とスタイルシートが有効であるか、確認してください。

DXXQ049E 出力ファイルはすでに存在します。

説明: 指定された出力ファイルは、このディレクトリーにすでに存在しています。

ユーザーの処置: 出力文書の出力パスまたはファイル名をユニークな名前に変更するか、または既存のファイルを削除してください。

DXXQ050E ユニーク・ファイル名を作成できません。

説明: UDF にアクセス権がないか、生成可能なすべてのファイル名が使用されているか、またはディレクトリーが存在しないため、UDF は指定のディレクトリーに出力文書のためのユニークなファイル名を作成できませんでした。

ユーザーの処置: UDF に指定のディレクトリーに対するアクセス権があることを確認し、使用できるファイル名をもったディレクトリーに変更してください。

DXXQ051E 入力データまたは出力データがありません。

説明: 入力または出力パラメーターの中に有効な値を持っていないものがあります。

ユーザーの処置: 必要パラメーターが欠落していないか、ステートメントをチェックしてください。

DXXQ052E DB2XML.XML_USAGE 表にアクセス中にエラーが発生しました。

説明: データベースが使用可能になっていないか、または、表 DB2XML.XML_USAGE がドロップされています。

ユーザーの処置: データベースが使用可能になっていて、表 DB2XML.XML_USAGE がアクセス可能であるか、確認してください。

DXXQ053E SQL ステートメントが失敗しました : *msg*

説明: XML Extender 処理時に生成された SQL ステートメントは、実行に失敗しました。

ユーザーの処置: 詳細をトレースで調べてください。エラー状態を訂正できない場合は、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。

DXXQ054E 無効な入力パラメーター: *param*

説明: ストアード・プロシージャーまたは UDF に対して指定された入力パラメーターが無効です。

ユーザーの処置: 関係のあるストアード・プロシージャーまたは UDF の署名をチェックし、実際の入力パラメーターが正しいか、確認してください。

DXXQ055E ICU エラー: *uerror*

説明: 変換操作中に ICU エラーが見つかりました。

ユーザーの処置: エラーをソフトウェア・サービス・プロバイダーに報告してください。トレース・ファイル、エラー・メッセージ、エラーを再現するための説明を含めてください。

DXXQ056E エレメントまたは属性 *xmlname* は、外部キー (表 *table* 内の列 *column*) の一部として指定された列にマップすることはできません。

説明: 指定されたエレメントまたは属性は、外部キーの一部として指定された列にマップすることはできません。外部キーのためのデータ値は主キーのデータ値により決定されます。 xml 文書内の指定されたエレメントまたは属性を表および列にマッピングする必要はありません。

ユーザーの処置: DAD 内の指定された列および表にマップされた RDB_node を除去してください。

DXXQ057E schemabindings と DTDID タグが DAD ファイルに一緒に存在することはできません。

説明: schemabindings と DTDID タグが DAD ファイルに一緒に存在することはできません。

ユーザーの処置: schemabindings タグと DTDID タグの両方ではなく、どちらか一方が DAD ファイルに存在していることを確認してください。

DXXQ058E schemabindings タグ内の nonamespacelocation タグが DAD ファイル内で欠落しています。

説明: schemabindings タグ内の nonamespacelocation タグが DAD ファイル内で欠落しています。

ユーザーの処置: nonamespacelocation タグを schemabindings タグに追加してください。

DXXQ059E doctype タグは、スキーマの妥当性検査のための DAD ファイルの XCollection タグ内に配置できません。

説明: doctype タグは、スキーマの妥当性検査のための DAD ファイルの XCollection タグ内に配置できません。

ユーザーの処置: スキーマ妥当性検査のための Xcollection タグ内の doctype タグを除去します。

DXXQ060E スキーマ ID *schemaid* の検索が失敗しました。

説明: XML Extender は、列を使用可能にするよう試みている間にスキーマ ID を検出できませんでした。スキーマ ID は DAD ファイルの schemabindings タグ内にある、 nonamespacelocation タグのロケーション属性の値に対応します。

ユーザーの処置: スキーマ 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。

DXXQ061E ストリングの形式が無効です。

説明: ストリング表記の形式が無効です。ストリングが、日付、時刻、またはタイム・スタンプ値の場合、構文はそのデータ・タイプに準拠していません。

ユーザーの処置: 日付、時刻、またはタイム・スタンプ値の形式がそのデータ・タイプの形式に準拠するか確認してください。

DXXQ062E *element* 用の XML 値を生成するための、*table* の結果セットの行がもうありません。

説明: このエラー状態は普通、指定されたエレメントまたは属性の親 element_node 上の multi_occurrence = YES 指定が欠落していることが原因で発生します。

ユーザーの処置: 親 element_nodes 上の multi_occurrence 値が、子 element_nodes の数指定を正確に反映しているかどうか DAD を確認してください。

DXXQ063E DAD ファイル内の *elementname* 上の multi_occurrence 属性値が無効です。

説明: 文書アクセス定義 (DAD) ファイル内の指定された element_node の multi_occurrence 属性の値が正しくないか、または属性がありません。値は、大文字小文字を区別しない、'yes' または 'no' である必要があります。

ユーザーの処置: 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。

DXXQ064E 外部表 *table* 内の列 *column* が見つかりません。

説明: 結合条件で指定されたキー列はどのエレメントまたは属性ノードにもマップされませんでした。

ユーザーの処置: DAD ファイル内で指定された結合条件が適切か、またすべてのキー列がエレメントまたは属性ノードにマップされているかを確認してください。

DXXQ065I XML 使用可能列に関係するすべてのトリガーが正常に再生成されました。

説明: これは単なる通知メッセージです。

ユーザーの処置: アクションは不要です。

DXXQ066E 表 *tablename* の主キーが存在しません。

説明: XML Extender は、表 *tablename* の主キーを判別できませんでした。表の主キーが XML 用に列を使用可能にした後にドロップされていないか確認してください。

ユーザーの処置: XML 用に列を使用可能にした場合、ルート ID として指定された主キーを追加するために表を変更します。

DXXQ067E *action* の試行が失敗しました。

説明: *action* の試行中に、SQL 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。エラーを報告する場合、XML Extender トレース・ファイルを必ず含めてください。

DXXQ068E 現行の SQLID を [userid] に設定できません。 SQLCODE = [sqlcode]。

説明: 現行の SQLID を 2 次許可 ID に設定する間に、SQL エラーが発生しました。

ユーザーの処置: 有効な 2 次許可 ID を指定しているかどうか、 ID に対する権限を持っているかどうかを確認してください。

付録 A. 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 の例』
- 『XML 文書の例: getstart.xml』
- 318 ページの『文書アクセス定義ファイル』
  - 318 ページの『DAD ファイルの例: XML 列』
  - 319 ページの『DAD ファイルの例: XML コレクション: SQL マッピング』
  - 321 ページの『DAD ファイルの例: XML: RDB_node マッピング』

XML DTD の例

以下の DTD は、本書全体を通して参照されている getstart.xml 文書に使用されます。

```
<!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 15. サンプル XML DTD: getstart.dtd

XML 文書の例: getstart.xml

以下の XML 文書 getstart.xml は、このガイド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 "dxx_install/samples/db2xml/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 16. 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 UDB 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。

- 『DAD ファイルの例: XML 列』
- 319 ページの『DAD ファイルの例: 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。
- 321 ページの『DAD ファイルの例: 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 ファイルの例: 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  "dxx_install/samples/db2xml/dtd/dad.dtd">
<DAD>
   <dtdid> "dxx_install/samples/db2xml/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 17. XML 列用のサンプル DAD ファイル : getstart_xcolumn.dad

DAD ファイルの例: 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 UDB 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 "dxx_install/samples/db2xml/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
               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 "dxx_install/samples/db2xml/dtd/getstart.dtd
"</doctype>
```

図 18. 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: order_sql.dad (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_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>
  </element_node>
</element_node>
</root_node>
</Xcollection>
</DAD>
```

図 18. 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: order_sql.dad (2/2)

DAD ファイルの例: 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 UDB 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 DAD SYSTEM "SQLLIB/samples/db2xml/dtd/dad.dtd>
<DAD>
 <dtdid>E:¥dtd¥lineItem.dtd</dtdid>
 <validation>YES</validation>
 <Xcollection>
 <prolog>?xml version="1.0"?</prolog>
 <doctype>!DOCTYPE Order SYSTEM
                  "SQLLIB/samples/db2xml/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">

       <element_node name="Name">
           <text_node>
              <RDB_node>
                 <table name="order_tab"/>
                 <column name="customer_name"/>
               </RDB_node>
          </text_node>
       </element_node>
       <element_node name="Email">
          <text_node>
              <RDB_node>
                  <table name="order_tab"/>
                  <column name="customer_email"/>
              </RDB_node>
          </text_node>
       </element_node>
 </element_node>
     <element_node name="Part">
        <attribute_node name="Key">
              <RDB_node>
                  <table name="part_tab"/>
                  <column name="part_key"/>
              </RDB_node>
        </attribute_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>

         <element_node name="Quantity">
              <text_node>
                 <RDB_node>
                     <table name="part_tab"/>
                     <column name="qty"/>
                 </RDB_node>
              </text_node>
        </element_node>
        <element_node name="Shipment" multi_occurrence="YES">
             <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>
```

付録 B. 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なければなりません。XML Extender では、ファイルの処理時に前提とされる事項があります。そこで、コード・ページ変換がどのように処理されるかを理解しておく必要があります。主な考慮事項は以下のとおりです。

- DB2 UDB 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ときに、XML 文書のエンコード宣言も、文書の実際のエンコードと整合している必要があります。

続くトピックでは、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 Extender および DB2 UDB 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。

XML コード・ページに関する用語

XML コード・ページに関するこのトピックでは、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、次のエンコード宣言は ibm-1047 を指定しています。

```
<?xml version="1.0" encoding="ibm-1047"?>
```

整合文書
コード・ページが、エンコードの宣言と一致する文書。

不整合文書
コード・ページが、エンコードの宣言と一致しない文書。

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 UDB 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 UDB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 UDB 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 Extender のコード・ページの前提事項

DB2 UDB は、XML 文書の送受信時にエンコード宣言をチェックしません。 その代わり、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。 これらが一致しない場合、DB2 UDB 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 つ以上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 文書をインポートするための前提事項

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 UDB 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 UDB 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 UDB 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 93 は、XML 文書のインポート時に DB2 UDB が実行する変換を要約しています。

表 93. 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

タスク	変換用ソース・コード・ページ	変換用ターゲット・コード・ページ	コメント
DTD ファイルを DTD_REF 表へ挿入する場合	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	

表 93. 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 (続き)

タスク	変換用ソース・コード・ページ	変換用ターゲット・コード・ページ	コメント
ストアード・プロシージャーを使用して、または DAD ファイルをインポートする管理コマンドを使用して、列またはコレクションを使用可能にする場合	クライアント・コード・ページ (インストール中に DXXADMIN をバインドするために使用されるコード・ページ)	データベースのコード・ページ	
ユーザー定義関数を使用する場合 • XMLVarchar FromFile() • XMLCLOB FromFile() • Content(): XMLFILE から取り出して CLOB に入れます	サーバー・コード・ページ	データベースのコード・ページ	データベース・コード・ページは、データがクライアントに表示されるときに、クライアント・コード・ページに変換されます。
分解のためにストアード・プロシージャーを使用する場合	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 文書をエクスポートするための前提事項

XML 文書は、データベースからエクスポートされるとき、次のオブジェクトのいずれか 1 つを提示するためのクライアントの要求に基づいてエクスポートされます。

- XML 列からの XML 文書
- XML 列の XML 文書の照会結果
- XML コレクションからの合成 XML 文書

文書のエクスポート時に、DB2 UDB 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。 328 ページの表 94 は、XML 文書のエクスポート時に DB2 UDB が実行する変換を要約しています。

表 94. 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

タスク	DB2 による変換	コメント
ユーザー定義関数を使用する場合 • XMLFileFromVarchar() • XMLFileFromCLOB() • 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	
照会またはエクスポートできる結果表に保管される XML 文書をストアード・プロシージャーで合成する場合。	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 Extender 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。

XML Extender のためのエンコード宣言の考慮事項

エンコード宣言 は、XML 文書のエンコードのコード・ページを指定するもので、XML 宣言ステートメントに現れます。 XML Extender を使用するときは、文書のエンコード方式が ファイルの保管場所に応じて、クライアントまたはサーバーのコード・ページと一致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。このセクションでは、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。 329 ページの表 95 にリストされているエンコードを使用する場合は、使用する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。
- **カテゴリー**は、その上で DB2 UDB 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 95. *XML Extender* によってサポートされるエンコード宣言

カテゴリー	エンコード	コード・ページ
Unicode	UTF-8	1208
	UTF-16	1200
ASCII	iso-8859-1	819
	ibm-1252	1252
	iso-8859-2	912
	iso-8859-5	915
	iso-8859-6	1089
	iso-8859-7	813
	iso-8859-8	916
	iso-8859-9	920
MBCS	gb2312	1386
	ibm-932、shift_jis78	932
	Shift_JIS	943
	IBM-eucCN	1383
	ibm-1388	1388
	IBM-eucJP、EUC-JP	954、33722
	ibm-930	930
	ibm-939	939
	ibm-1390	1390
	ibm-1399	1399
	ibm-5026	5026
	ibm-5035	5035
	euc-tw、IBM-eucTW	964
	ibm-937	937
	euc-kr、IBM-eucKR	970
	big5	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 Extender 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 Web アドレス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。

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。

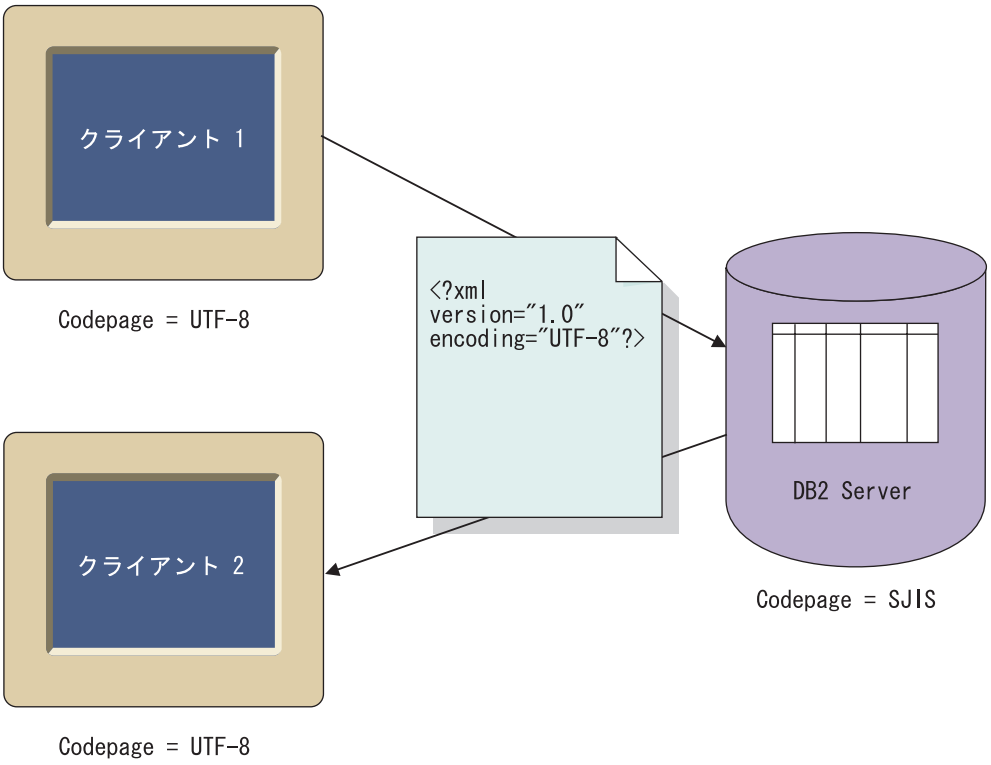


図 19. クライアントでのコード・ページの一致

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。

- 変換時にデータが脱落する場合があります。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のおそれがあります。 Unicode には、文字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。

331 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。

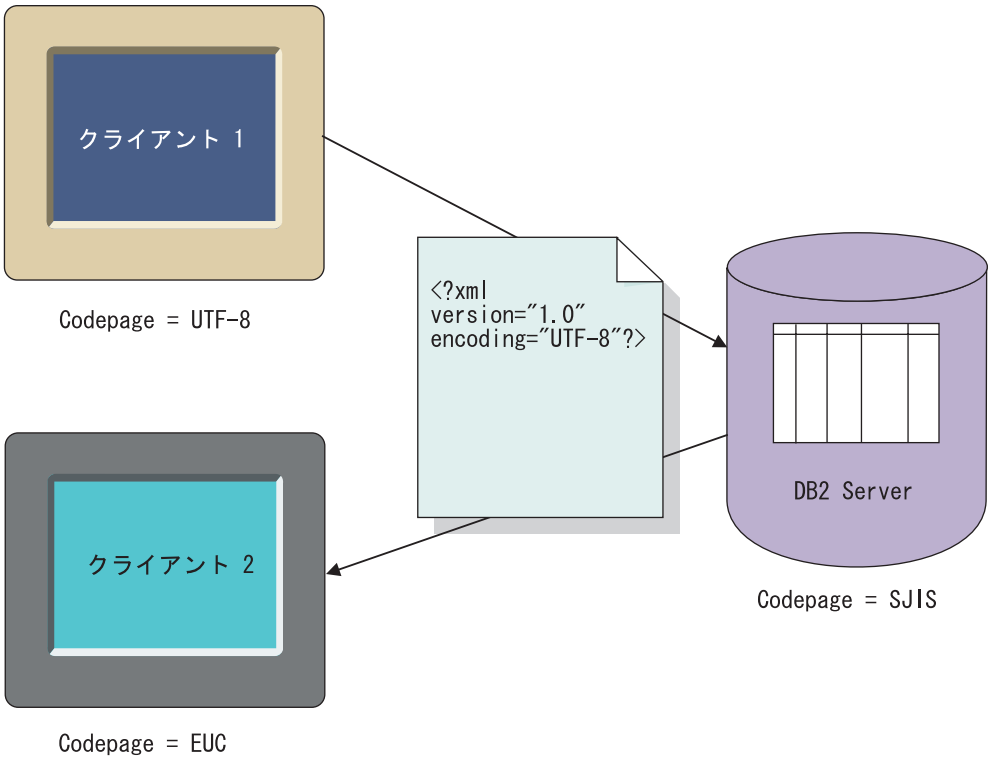


図 20. クライアント間でコード・ページが不整合

Client 2 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 Extender 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 UDB 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。 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 UDB 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。

1. 文書は XMLClobfromFile UDF を使用して DB2 UDB 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。

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 UDB 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 、クライアントとサーバーのコード・ページは SJIS 、そして、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 UDB にインポートされます。 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。
2. 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。 DB2 UDB 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 UDB 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 SJISibm-1047、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。 DB2 UDB 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。
2. 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。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。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、データ損失、DB2 UDB 変換、UTF-8 サーバーからの文書のインポートという構成です。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。

文書は XMLClobfromFile UDF を使用して DB2 UDB にインポートされます。 DB2 UDB 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 UDB 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、XML ファイルを戻す Content() UDF を使用して DB2 UDB からエクスポートされます。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 UDB は文書を UTF-8 から 1252 に変換します。

不整合な XML 文書を防止のための勧告

上記のセクションでは、XML 文書が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 プロセッサーに渡す前に、以下の推奨事項の 1 つを実行してください。

- XML Extender 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 Extender 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
  - オーバーライド機能がツールに備わっていれば、宣言エンコードをオーバーライドする
  - 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 Extender 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)。
  - 宣言エンコードのコード・ページに文書を変換する
  - オーバーライド機能がツールに備わっていれば、宣言エンコードをオーバーライドする
  - 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 クライアントの使用時の制限: 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 XMLFromFile 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 UDB 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 C. XML Extender の制限

このトピックでは、次のものについての制限を説明します。

- XML Extender オブジェクト
- ユーザー定義関数が戻す値
- ストアード・プロシージャー・パラメーター
- 管理サポート表の列
- 合成と分解

次の表は、XML Extender オブジェクトに対する制限です。

表 96. XML Extender オブジェクトの制限

オブジェクト	制限
分解 XML コレクション内の表における最大行数	分解されたそれぞれの XML 文書から 10240 行
パラメーター値として指定される XML ファイル・パス名の最大バイト	512 バイト
SQL 合成用 DAD ファイル内の sql_stmt エレメントの長さ	Windows および UNIX オペレーティング・システム: 32,765 バイト。OS/390 および iSeries オペレーティング・システム: 16,380 バイト。
RDB_node 分解用の DAD ファイルで 1 つの表に対して指定される、 1 つの表の列の最大数	500 列 (1 つの表の列) は、DAD ファイル内の text_node と attribute_node エレメントで指定される。

次の表は、XML Extender ユーザー定義関数が戻す値の制限です。

表 97. ユーザー定義関数値の制限

ユーザー定義関数から戻される値	制限
extractCHAR UDF が戻す最大バイト	254 バイト
extractCLOB UDF が戻す最大バイト	2 ギガバイト
extractVARCHAR UDF が戻す最大バイト	4 キロバイト

次の表は、XML Extender ストアード・プロシージャーのパラメーターに対する制限です。

表 98. ストアード・プロシージャー・パラメーターの制限

ストアード・プロシージャー・パラメーター	制限
XML 文書 CLOB の最大サイズ[1]	1 MB
文書アクセス定義 (DAD) CLOB の最大サイズ[1]	100 KB
collectionName の最大サイズ	30 バイト

表 98. ストアード・プロシージャー・パラメーターの制限 (続き)

ストアード・プロシージャー・パラメーター	制限
colName の最大サイズ	30 バイト
dbName の最大サイズ	8 バイト
defaultView の最大サイズ	128 バイト
rootID の最大サイズ	30 バイト
resultTabName の最大サイズ	18 バイト
tablespace の最大サイズ	8 バイト
tbName の最大サイズ [2]	18 バイト
resultColumn の最大サイズ	30 バイト
validColumn の最大サイズ	30 バイト
varchar_value の最大サイズ	32672 バイト

注:

1. dxxGenXMLClob および dxxRetrieveXMLCLOB では、このサイズは変更可能です。
2. *tbName* パラメーターの値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 バイト以下でなければなりません。

次の表は、DB2XML.DTD_REF 表に対する制限です。

表 99. XML Extender の制限

DB2XML.DTD_REF 表列	制限
AUTHOR 列のサイズ	128 バイト
CREATOR 列のサイズ	128 バイト
UPDATOR 列のサイズ	128 バイト
DTDID 列のサイズ	128 バイト
CLOB 列のサイズ	100 KB

名前は、DB2 UDB 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。

次の表は、合成および分解に対する制限です。

表 100. XML Extender 合成および分解に対する制限

オブジェクト	制限
分解 XML コレクション内の表における最大行数	分解されたそれぞれの XML 文書から 10240 行
DAD 内の elements_node または attribute_node での名前属性の最大長	63 バイト
パラメーター値として指定される XMLFile パス名の最大バイト	512 バイト

DB2DXX_MIN_TMPFILE_SIZE 環境変数:

XML Extender は一時ファイルに文書を置くことで、処理中のメモリーの大量使用を回避することができます。大量の物理メモリーを持つシステムでは、入出力アクティビティーの量を減らして、一時ファイルへの文書移動を回避することが可能です。環境変数 DB2DXX_MIN_TMPFILE_SIZE は、指定の値よりも小さい文書を処理する場合、一時ファイルではなくメモリー・バッファーを使用するように XML Extender に指示します。この変数はサーバーでのみ適用可能で、クライアントでは適用できません。複数の物理ノードが複数ノードのパーティションに参加する場合、この変数は、各マシンにインストールされているメモリー量を正確に反映するように、各ノードごとに設定することができます。環境変数が設定されていない場合、 128 キロバイトより大きい文書は処理中に自動的に一時ファイルに置かれ、128 キロバイトより小さい文書はメモリーで処理されます。

付録 D. XML Extender のための 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 オブジェクトのセットの論理グループ化を行うための ID です。 XML Extender の UDF と UDT のためのスキーマ名は、DB2XML です。本書では、関数名だけを参照します。

UDT と UDF は、スキーマ名を関数パスに加えておけば、スキーマ名なしに指定できます。関数パスは、スキーマ名の順番付きリストです。 DB2 UDB 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。このステートメントにより、関数パスが CURRENT FUNCTION PATH 特殊レジスター内に設定されます。

推奨: DB2XML スキーマ名を関数パスに加えます。このスキーマ名を加えておくと、XML Extender の UDF と UDT 名を DB2XML で修飾せ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
```

制約事項: ユーザー ID DB2XML としてログオンしている場合は、最初のスキーマとして DB2XML を関数パスに加えないでください。 DB2XML としてログオンすると、自動的に DB2XML が最初のスキーマとして設定されるからです。 DB2XML を最初のスキーマとして関数パスに加えると、関数パスが 2 つの DB2XML スキーマで開始するため、エラー状態が戻されます。

XML Extender 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 UDB 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。

アクセス関数 (access function). 列に保管されるテキストのデータ・タイプを、 Net Search Extender で処理できるタイプに変換する、ユーザー提供の関数。

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 UDB エクステンダーに備わった API の用途は、ユーザー定義関数の要求、管理操作、表示操作、およびビデオ画面の変更の検出である。 DB2 Net Search エクステンダーに備わった API の用途は、ユーザー定義関数の要求、管理操作、および情報検索サービスである。 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。

イメージ (image). 絵や写真を電子的に表したもの。

エスケープ文字 (escape character). それ以降の文字が、解釈されない マスク文字 *(masking character)* であることを示す文字。

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 つ以上の列から成る。

外部ファイル (external file). DB2 のコントロールする表の中の 1 つのセルに保管されたファイルではなく、オペレーティング・システムのファイル・システムに保管されたファイルとしてのテキスト文書。DB2 の外部のファイル・システムに存在するファイル。

拡張する (expand). シソーラスから導出された用語を検索項目に追加すること。

カタログ・ビュー (catalog view). 管理目的のために Net Search Extender が作成するシステム表のビュー。カタログ・ビューには、 Net Search Extender により使用可能になっている表および列についての情報が含まれる。

関数 (function). *アクセス関数 (access function)* を参照。

管理 (administration). 検索、索引の保守、および状況情報の取得などのためにテキスト文書を用意する作業。

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 UDB 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。

管理サポート表 (administrative support table). イメージ、オーディオ、およびビデオ・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 UDB エクステンダーが使用する 1 つの表。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)* とも呼ばれる。

ギガバイト (gigabyte, GB). 10 億 (10^9) バイト。 メモリー容量の場合は、1,073,741,824 バイト。

キャスト関数 (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、キャスト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。

境界検索 (bound search). 韓国語文書における、ワード境界を区別した検索。

キロバイト (kilobyte, KB). 千 (10^3) バイト。 メモリー容量の場合は 1024 バイト。

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コレクション。

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。

結合ビュー (joined view). CREATE VIEW ステートメントによって作成される DB2 UDB ビューの 1 つで、 1 つ以上の表を互いに結合する。

言語索引 (linguistic index). テキスト*索引 (text index)* の 1 つで、言語処理によって基本形に変換された用語を含む。 たとえば “Mice” (ねずみの複数形) は、“mouse” (単数形) として索引付けされる。*厳密索引 (precise index)*、*Ngram 索引 (Ngram index)*、および*二重索引 (dual index)* も参照。

検索引き数 (search argument). 検索の際に指定される条件で、 1 つ以上の検索項目と検索パラメーターから成る。

コード・ページ (code page). すべてのコード・ポイントに GRAPHIC 文字および制御機能の意味を割り当てたもの。たとえば、8 ビット・コードの場合、文字および意味を 256 のコード・ポイントへ割り当てたもの。

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 UDB Net Search Extender を使って、XML 文書のツリー構造に基づいてテキスト・キーを索引付けること。

コマンド行プロセッサー (command line processor). DB2TX というプログラムで、次のような機能がある:

ユーザーが Net Search Extender のコマンドを入力する

コマンドを処理する

結果を表示する

[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 Extender が作成する追加の表。

索引 (index). テキスト内から重要な用語を抽出してテキスト*索引 (text index)* に保管すること。キーの値によって論理順に並べられたポインターのコレクション。 索引は、データに迅速にアクセスするのに使われ、表内の行でのユニーク性を強化できる。

主キー (primary key). 表の定義の一部であるユニーク・キー。主キーは、参照制約定義のデフォルトの親キーである。

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。

照会オブジェクト (query object). QBIC 照会用の機能、機能の値、および機能の重みを指定するオブジェクト。オブジェクトに名前を付けて保管し、後で QBIC 照会で使用することができる。照会ストリング (query string) と対比。

使用可能にする (enable). データベース、テキスト表、またはテキスト列を XML Extender で使用できるようにすること。

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。

使用不可にする (disable). 使用可能処理中に作成された項目を除去することにより、データベース、テキスト表、またはテキスト列を、XML Extender で使用可能になる前の状態にリストアすること。

ショット・カタログ (shot catalog). ビデオ・クリップ内のショットに関するデータ (たとえば、ショットの開始フレーム番号と終了フレーム番号) の保管に使われるデータベース表またはファイル。ユーザーは SQL 照会を介して表のビューにアクセスしたり、またはファイル内のデータにアクセスすることができる。

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、ビュー、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). オブジェクトの絶対パス名。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。

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処理。 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、最初に独自の形式からデータを変換する処理が必要ない。

データベース・パーティション (database partition). 固有のユーザー・データ、索引、構成ファイル、およびトランザクション・ログから成るデータベースの部分。ノードまたはデータベース・ノードということもある。

データベース・パーティション・サーバー (database partition server). データベース・パーティション *(database partition)* を管理するサーバー。 データベース・パーティション・サーバーは、データベース・マネージャーと、データベース・マネージャーの管理するデータやシステム・リソースから構成される。通常は、1 つのデータベース・パーティション・サーバーが 1 台のマシンに割り当てられる。

データ・ストリーム (data stream). API 機能から戻される情報で、(少なくとも 1 つの段落からなる) テキストで構成される。テキストには検索対象の用語、および見付かった用語をそのテキスト内で強調表示するための情報が入っている。

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。

テキスト表 (text table). テキスト列 *(text columns)* を含んでいる DB2 UDB 表。

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。

デフォルト・ビュー (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。

テラバイト (terabyte). 1 兆 (10^{12}) バイト。 10 の 12 乗バイト。メモリー容量の場合は、 1,099,511,627,776 バイト。

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。

トリガー (trigger). テキスト列内で文書が追加、変更、または削除されるたびに、索引付けの必要な文書についての情報をログ表 *(log table)* に自動的に追加するメカニズム。

トリガー (trigger). 表が変更された時に実行される一連のアクションの定義。トリガーを使って実行できるアクションには、入力データの妥当性検査、新たに挿入された行の値の自動生成、相互参照のための他の表の読み取り、または監査のための他の表への書き込みがある。トリガーは、保全性検査やビジネス・ルール施行のためにしばしば利用される。

トレース (tracing). あとでエラー原因の検出に使用できる情報をファイルに保管するアクション。

[ハ行]

バイナリー・ラージ・オブジェクト (BLOB) (binary large object (BLOB)). 長さの上限が 2 GB のバイナリー・ストリング。イメージ、オーディオ、およびビデオ・オブジェクトは、 DB2 データベースでは BLOB として保管される。

パス式 (path expression). ロケーション・パス *(location path)* を参照。

ビデオ (video). 再生して見ることのできる、録画された情報を指す。

ビデオ索引 (video index). ビデオ・クリップ内の特定のショット *(shot)* またはフレームを見つけるためにビデオ・エクステンダーが使用するファイル。

ビデオ・クリップ (video clip). フィルム撮影またはビデオ録画されたデータの 1 つのセクション。

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。

ブール検索 (Boolean search). 1 つ以上の言葉をブール演算子を使って結合する検索。

ファイル参照変数 (file reference variable). プログラミング変数の 1 つで、クライアント・ワークステーション上のファイルに LOB を移動したり戻したりするのに使用できる。

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。

複数回出現 (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。複数回出現は DAD 内で指定される。

ブラウザー (browser). コンピューター・モニター上にテキストを表示する、Net Search Extender の機能の 1 つ。*Web ブラウザー (Web browser)* も参照。

ブラウズする (browse). コンピューター・モニター上にテキストを表示すること。

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 文書を XML コレクション内の複数のリレーショナル表に分離すること。

文書 (document). *テキスト文書 (text document)* を参照。

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 Extender 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に対する宣言のコレクション。 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。

分析する (analyze). あるイメージ (画像) の特徴を表す数値を計算して、その値を QBIC カタログに追加すること。

ホスト変数 (host variable). 組み込み SQL ステートメントが参照できる、アプリケーション・プログラム内の変数。ホスト変数は、データベースとアプリケーション・プログラム作業域の間でデータを伝送するための主要なメカニズムである。

［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 Extender 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。

メガバイト (megabyte, MB). 百万 (10⁶) バイト。 メモリー容量の場合は 1,048,576 バイト。

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 UDB 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。

ユーザー定義関数 (user-defined function (UDF)). DB2 ユーザーによって作成される SQL 関数で、DB2 の提供する SQL 関数と対比される。 Net Search Extender は、CONTAINS のような検索関数を UDF の形で提供する。

ユーザー定義関数 (user-defined function (UDF)). ユーザーが DB2 に定義する関数。関数が定義されると、SQL 照会やビデオ・オブジェクトで使用できる。たとえば、ビデオの圧縮形式を得る UDF や、オーディオのサンプリング・レートを戻す UDF を作成できる。これによって、特定タイプのオブジェクトの振る舞いを定義することができる。

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。

ユーザー定義タイプ (user-defined type (UDT)). ユーザーが DB2 に定義するデータ・タイプ。 UDT は 1 つの LOB を別の LOB と区別するために使用される。たとえば、イメージ・オブジェクト用とオーディオ・オブジェクト用とに、別の UDT を作成することができる。こうすることによって、イメージ・オブジェクトとオーディオ・オブジェクトは BLOB として保管されるものの、 BLOB とは違うタイプとして、また互いに違うタイプとして扱われる。

ユーザー定義特殊タイプ (user-defined distinct type (UDT)). DB2 ユーザーによって作成されるデータ・タイプで、 DB2 UDB の提供する LONG VARCHAR のようなデータ・タイプと対比される。

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、*バイナリー・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 UDB 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 UDB 列とをマッピングする。さらに、Net Search Extender 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。

論理ノード (logical node). あるプロセッサーに複数のノードが割り当てられる場合の、プロセッサー上の 1 つのノード。

［ワ行］

ワイルドカード文字 (wildcard character). マスク文字 *(masking character)* を参照。

［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 UDB 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 UDB エンジンが提供する各国語の索引体系。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

CCSID. コード化文字セット ID (Coded Character Set Identifier)。

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)。

DBCS. 2 バイト文字サポート (Double-byte character support)。

DTD. (1) . (2) 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 UDB 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。

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。

Extensible 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。

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、フォーマット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。

J

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)。

LOB ロケーター (LOB locator). ホスト変数の中に保管される短精度 (4 バイトの) 値で、プログラムはこれを使用して DB2 UDB データベース内のより大きな LOB を参照する。 LOB ロケーターを使用すると、ユーザーは LOB が通常のホスト変数の中に保管されているかのように操作でき、クライアント・マシン上のアプリケーションとデータベース・サーバーとの間で LOB を移送する必要がない。

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)。各データベース管理システムが異なるデータ・ストレージ形式およびプログラミング・インターフェースを採用している場合でも、データベース・アプリケーションは、この API を使用することにより、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ます。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

QBIC カタログ (QBIC catalog). イメージの視覚的な特徴についてのデータを保持しているリポジトリー。

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 属性の値のロケーションで、RDB_node によって定義される。 XML Extender 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

SBCS. 1 バイト文字サポート。

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 つ以上の SQL ステートメントおよび XSLT データ・モデルを使用する。 XML Extender 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/software /data/db2/extender。

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. 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 Extender の提供する DB2 UDB ユーザー定義関数。

XML UDT. XML Extender の提供する DB2 UDB 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 Extender 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). 関係表のコレクションであり、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 Extender 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 つ以上の XML Extender 列を含んでいるアプリケーション表。

XML 列 (XML column). XML Extender 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)。

特記事項

本書に記載の製品、サービス、または機能が日本においては提供されていない場合があります。日本で利用可能な製品、サービス、および機能については、日本 IBM の営業担当員にお尋ねください。本書で IBM 製品、プログラム、またはサービスに言及していても、その IBM 製品、プログラム、またはサービス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。これらに代えて、IBM の知的所有権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 以外の製品とプログラムの操作またはサービスの評価および検証は、お客様の責任で行っていただきます。

IBM は、本書に記載されている内容に関して特許権 (特許出願中のものを含む) を保有している場合があります。本書の提供は、お客様にこれらの特許権について実施権を許諾することを意味するものではありません。実施権についてのお問い合わせは、書面にて下記宛先にお送りください。

〒106-0032
東京都港区六本木 3-2-31
IBM World Trade Asia Corporation
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。

この情報には、技術的に不適切な記述や誤植を含む場合があります。本書は定期的に見直され、必要な変更は本書の次版に組み込まれます。 IBM は予告なしに、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。

本書において IBM 以外の Web サイトに言及している場合がありますが、便宜のため記載しただけであり、決してそれらの Web サイトを推奨するものではありません。それらの Web サイトにある資料は、この IBM 製品の資料の一部ではありません。それらの Web サイトは、お客様の責任でご使用ください。

IBM は、お客様が提供するいかなる情報も、お客様に対してなんら義務も負うことのない、自ら適切と信ずる方法で、使用もしくは配布することができるものとします。

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 Canada Limited
Office of the Lab Director
8200 Warden Avenue
Markham, Ontario
L6G 1C7
CANADA

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。

この文書に含まれるいかなるパフォーマンス・データも、管理環境下で決定されたものです。そのため、他の操作環境で得られた結果は、異なる可能性があります。一部の測定が、開発レベルのシステムで行われた可能性がありますが、その測定値が、一般に利用可能なシステムのものと同じである保証はありません。さらに、一部の測定値が、推定値である可能性があります。実際の結果は、異なる可能性があります。お客様は、お客様の特定の環境に適したデータを確かめる必要があります。

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。したがって、他社製品に関する実行性、互換性、またはその他の要求については確証できません。 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 の将来の方向または意向に関する記述については、予告なしに変更または撤回される場合があり、単に目標を示しているものです。

本書には、日常の業務処理で用いられるデータや報告書の例が含まれています。より具体性を与えるために、それらの例には、個人、企業、ブランド、あるいは製品などの名前が含まれている場合があります。これらの名称はすべて架空のものであり、名称や住所が類似する企業が実在しているとしても、それは偶然にすぎません。

著作権使用許諾:

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。このサンプル・プログラムは、あらゆる条件下における完全なテストを経ていません。従って IBM は、これらのサンプル・プログラムについて信頼性、利便性もしくは機能性があることをほのめかしたり、保証することはできません。

それぞれの複製物、サンプル・プログラムのいかなる部分、またはすべての派生した創作物には、次のように、著作権表示を入れていただく必要があります。

商標

以下は、IBM Corporation の商標です。

ACF/VTAM
AISPO
AIX
AIXwindows
AnyNet
APPN
AS/400
BookManager
C Set++
C/370
CICS
Database 2
DataHub
DataJoiner
DataPropagator
DataRefresher
DB2
DB2 Connect
DB2 Extenders
DB2 OLAP Server
DB2 Information Integrator
DB2 Query Patroller
DB2 Universal Database
Distributed Relational Database Architecture
DRDA
eServer
Extended Services
FFST
First Failure Support Technology
IBM
IMS
IMS/ESA
iSeries
LAN Distance
MVS
MVS/ESA
MVS/XA
Net.Data
NetView
OS/390
OS/400
PowerPC
pSeries
QBIC
QMF
RACF
RISC System/6000
RS/6000
S/370
SP
SQL/400
SQL/DS
System/370
System/390
SystemView
Tivoli
VisualAge
VM/ESA
VSE/ESA
VTAM
WebExplorer
WebSphere
WIN-OS/2
z/OS
zSeries

以下は、それぞれ各社の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。

Pentium は、Intel Corporation の米国およびその他の国における商標です。

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。

UNIX は、The Open Group の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です。

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]

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、お客様サポートにご連絡ください。

製品情報

DB2 Universal Database 製品に関する情報は、
http://www.ibm.com/software/data/db2/udb から入手できます。

このサイトには、技術ライブラリー、資料の注文方法、製品のダウンロード、ニュースグループ、フィックスパック、ニュース、および Web リソースへのリンクに関する最新情報が掲載されています。

米国以外の国で IBM に連絡する方法については、 IBM Worldwide ページ (www.ibm.com/planetwide) にアクセスしてください。

Printed in Japan

SC88-9172-01

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