IBM DB2 UDB for iSeries

XML エクステンダー 管理およびプログラミング

バージョン 8

SC88-4030-00
(英文原典：SC18-9179-00)

IBM DB2 UDB for iSeries

XML エクステンダー 管理およびプログラミング

バージョン 8

SC88-4030-00
(英文原典：SC18-9179-00)

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、 291 ページの『特記事項』に記載されている情報をお読みください。

本書は、IBM DB2 Database (プログラム番号 5722-DE1) のバージョン 5、リリース 3、モディフィケーション 0 に適用されます。また、改訂版で断りがない限り、それ以降のすべてのリリースおよびモディフィケーションに適用されます。この版は、縮小命令セット・コンピューター (RISC) システムにだけ適用されます。

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：	SC18-9179-00 IBM DB2 UDB for iSeries XML Extender Administration and Programming Version 8
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 2004.4

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

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

目次

表

本書について

このセクションでは、以下の情報が含まれています。

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

本書の対象読者

本書は以下の読者を対象としています。

- DB2® アプリケーション内で XML データを扱う方で、XML の概念を理解している方。本書の読者には、XML および DB2 についての一般的な知識が必要です。 XML についての詳細は、次の Web サイトを参照してください。

  http://www.w3.org/XML

  DB2 についての詳細は、次の 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

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

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

本書の使用方法

本書は以下のように構成されています。

第 1 部 概要

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

第 5 部 付録

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

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
- プッシュボタン

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。

  ▶▶── 記号は、ステートメントの開始を示します。

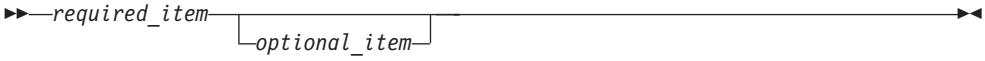
  ──▶ 記号は、構文図が次の行に続くことを示します。

  ▶── 記号は、ステートメントが前の行から続いていることを示します。

  ──▶◀ 記号は、ステートメントの終了を示します。

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

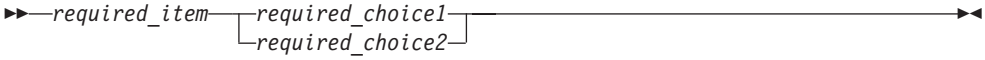
- オプション項目は、メインパスの下に示されます。



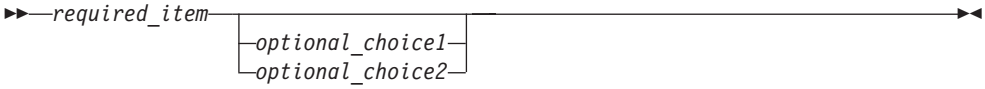
  オプション項目がメインパスの上に示されている場合、その項目は読みやすくするためだけに示されているのであり、ステートメントの実行には影響を与えません。

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

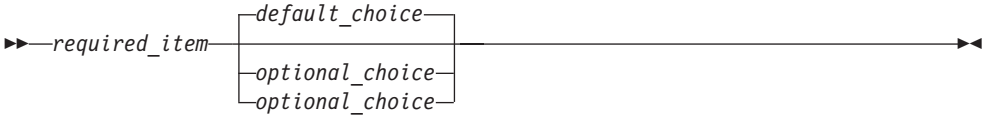
  項目から 1 つを選択しなければならない 場合、スタック内の項目の 1 つがメインパスの線上に示されます。



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

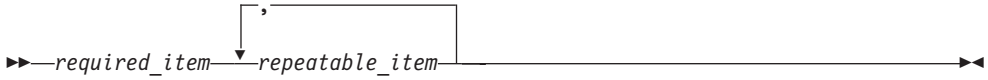


  項目の 1 つがデフォルト値である場合、その項目はメインパスの上に示されて、残りの項目は下に示されます。



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

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



- スタックの上にある反復の矢印は、そのスタック内の項目を反復できることを示します。
  - キーワードは、英大文字 (FROM など) で示されます。 XML エクステンダーでは、キーワードを大文字でも小文字でも指定できます。キーワードではない用語は、英小文字 (*column-name* など) で示されます。それらは、ユーザーが指定する名前または値を示します。
  - 句読記号、括弧、算術演算子、または同様の他の記号が示されている場合、それらを構文の一部として入力しなければなりません。

第 1 部 概要

ここでは、XML エクステンダー の概要とビジネス・アプリケーションにおけるその使用法について示します。

第 1 章 概要

XML エクステンダーの概要

DB2 の XML エクステンダーが提供する機能により、XML 文書の保管およびアクセス、既存のリレーショナル・データからの XML 文書の生成、また、XML 文書からリレーショナル表に行を挿入することができます。 XML エクステンダーは、DB2 UDB リレーショナル・データベース (本書では「RDB データベース」または単に「データベース」と呼びます) で XML データを管理するための新しいデータ・タイプ、関数、およびストアード・プロシージャーを提供します。

XML エクステンダーは、以下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
- OS/390 および z/OS
- iSeries

関連概念:

- 3 ページの『XML 文書』
- 5 ページの『XML エクステンダーの機能』
- 9 ページの『学習: XML 列への XML 文書の保管』
- 19 ページの『学習: XML 文書の合成』
- 8 ページの『XML エクステンダーのチュートリアル』

XML 文書

異なるアプリケーション間でのデータの共用がしばしば行われるため、データを他のアプリケーションにインポートできる形式で、複製、変換、エクスポート、または保管するという課題が絶えず発生します。これらの変換プロセスでは、多くの場合、データが部分的に喪失したり、データの整合性を保証するための面倒なプロセスの実行が最低限必要であったりします。この手作業による検査は時間と経費の浪費です。

この問題に対処する方法の 1 つは、アプリケーション開発者が、 *Open Database Connectivity (ODBC)* アプリケーションを作成することです。 ODBC は、リレーショナル・データベース管理システムおよび非リレーショナル・データベース管理システムの両方において、データにアクセスするための標準のアプリケーション・プログラミング・インターフェース (API) です。これらのアプリケーションでは、データはデータベース管理システムに保管されます。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、アプリケーションが必要とする形式にデータが変換されるように作成しなければなりません。しかし、アプリケーションはすぐに変更され、すぐに陳腐化します。データを HTML に変換するアプリケーションでは表示ソリューションが用意されていますが、その表示用のデータを、他の用途に使えることはまずありません。アプリケーション間で相互にデータをやりとりするための実際的な手段を提供するために、データと表示を分離する手段が必要です。

XML —*eXtensible Markup Language*— は、この問題に対処するために開発されました。 XML が拡張可能と呼ばれているのは、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を作成できるからです。 XML を使用すると、特定のアプリケーション用のデータだけでなく、データ構造をも取り込むことができます。 XML は、データ交換のための唯一の形式ではありませんが、標準規格として受け入れられるようになってきました。この標準規格に従えば、最初に専用の形式を使用してデータ変換を行わなくても、アプリケーション間でデータを共用することができます。

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、タスクを XML 形式でエクスポートし、その XML 形式のタスクをそのままの形で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。今日のような相互にネットワーク接続された世界において、アプリケーション・プロバイダーの間では、XML による相互交換形式を、アプリケーションの基本機能にしようとする傾向が強まっています。

DB2 における XML データの処理方法

XML は、データ交換のための標準形式を提供することにより多くの問題を解決しますが、課題もいくつか残ります。企業のデータ・アプリケーションを構築する場合には、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。
- アプリケーション間で共用すべき情報にはどのようなものがあるか。
- どのようにして必要な情報をす早く検索できるか。
- 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、より効率的に取得することができます。XML エクステンダーを使用すると、多くの XML アプリケーションで DB2® の能力を生かすことができます。

DB2 UDB データベース内の構造化された XML 文書の内容を使用して、構造化された XML 情報と従来のリレーショナル・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基づき、XML 文書全体を、XML データに与えられているユーザー定義のタイプ (XML データ・タイプ) として DB2 に保管するか、 XML の内容

をリレーショナル表内の基本データ・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 エクステンダーには、 Text Extender が備えている構造化テキスト検索機能のほかに、 XML エレメント値または属性値の豊富なデータ・タイプを検索する機能が追加されています。

XML エクステンダーでは、 DB2 での XML データの保管およびアクセスのための方式が 2 とおり提供されています。

XML 列方式

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 エクステンダーの機能

XML エクステンダーには、 DB2 を使用して 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
- データベース内での XML データの保管とアクセスの方式
- XML データの妥当性検査に使用される DTD を保管するためのデータ・タイプ定義 (DTD) リポジトリー
- 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する、文書アクセス定義 (DAD) と呼ばれるマッピング・ファイル
- XML 文書内でのエレメントまたは属性の位置を指定するためのロケーション・パス

管理ツール: XML エクステンダーの管理ツールは、データベースおよび表の列を XML 対応にしたり、 XML データを DB2® リレーショナル構造にマップしたりする際に役立ちます。

以下のツールを使用すると、 XML エクステンダーのための管理タスクを完了することができます。

- **dxxadm** コマンドは、OS のコマンド行から実行できます。
- ストアード・プロシージャーは、iSeries™ ナビゲーターから実行できます。
- XML エクステンダー管理ストアード・プロシージャーにより、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。

保管およびアクセスの方式: XML エクステンダーには、 DB2 データ構造に XML 文書を組み込むために、 XML 列および XML コレクションの 2 つの保管およびアクセスの方式を提供しています。これらの方式は、それぞれかなり異なる用途に使用されますが、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、XML 文書をそのまま DB2 に保管するうえで役立ちます。 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 対応の列に文書を挿入した後、これを更新、取り出し、および検索することが

できます。エレメントおよび属性データを DB2 UDB 表 (サイド表) にマップすることができます。サイド表には高速検索を行えるよう索引を付けることができます。

XML コレクション方式
この方式は、DB2 UDB 表に XML 文書構造をマップする上で役立ちます。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向きであり、XML 文書の内容を頻繁に更新する場合は特に適しています。

DTD: XML エクステンダーを使用すると、 XML エレメントおよび属性の宣言の集合である 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*になっている場合は、 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 文書の構造の妥当性検査に使用されます。

DAD ファイル: 構造化された XML 文書を XML エクステンダーで処理するための方法を指定するには、*文書アクセス定義 (DAD)* ファイルを使用します。 DAD ファイルは、XML 文書の構造を 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場合、または XML データを合成あるいは分解する場合に使用します。 DAD ファイルは、XML 列方式を使用して文書を保管するのか、合成または分解のために 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 文書内でのエレメントまたは属性の位置を指定します。 XML エクステンダーはロケーション・パスを使用して、XML 文書の構造内を移動し、エレメントと属性を見つけます。

例えば、`/Order/Part/Shipment/ShipDate` というロケーション・パスは、以下の例に示すように、shipDate エレメント、つまり Shipment、Part、および Order エレメントの子を指しています。

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

7 ページの図 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。

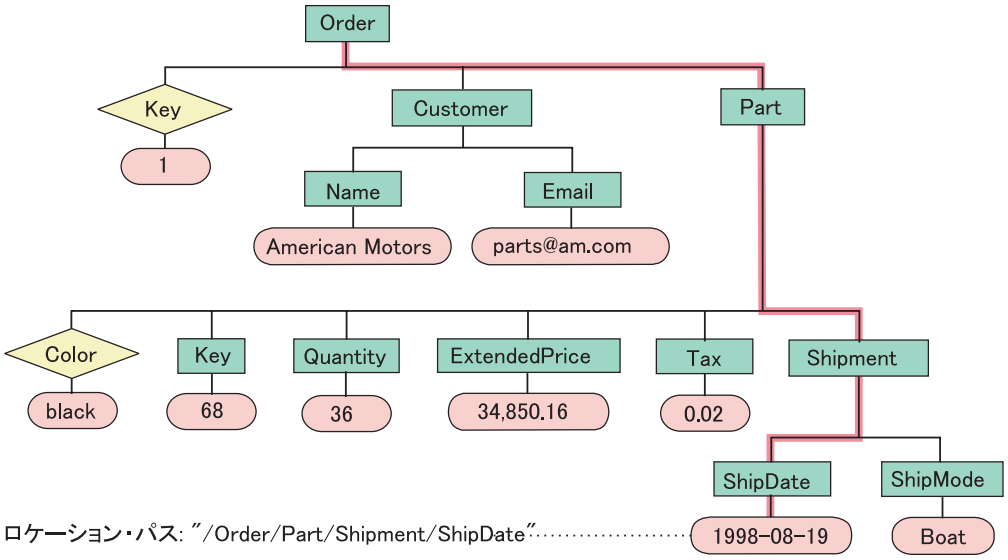


図 1. 文書を構造化 XML 文書として DB2 UDB 表の列に保管する

ロケーション・パスは以下の状況で使用されます。

XML 列

- XML エクステンダーのユーザー定義関数を使用する場合に、抽出または更新するエレメントおよび属性を指定するために使用されます。
- XML の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

ストアード・プロシージャーの DAD ファイルの値を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 エクステンダーは、 *XML Path 言語 (XPath)* (XML 文書のパーツの位置を指定するための言語) のサブセットを使用します。

Xpath についての詳細は、次の Web ページを参照してください。

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『学習: XML 列への XML 文書の保管』
- 19 ページの『学習: XML 文書の合成』
- 8 ページの『XML エクステンダーのチュートリアル』

XML エクステンダーのチュートリアル

このチュートリアルでは、XML エクステンダーを使用してアプリケーション用の XML データにアクセスし、それを変更する方法を示します。次の 3 つのレッスンが提供されています。

- XML 列への XML 文書の保管
- 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。

管理についての学習では、 XML エクステンダー管理コマンドと共に を使用します。 XML データ管理の学習では、 XML エクステンダーに備わっている UDF およびストアード・プロシージャーを使用します。本書の残りの部分にある例のほとんどは、この章で使用されるサンプル・データを利用しています。

前提条件

このチュートリアルの学習を完了するには、次のものがインストールされている必要があります。

- DB2 Universal Database™ バージョン 5 リリース 3
- オプション: 学習サンプルを実行するための iSeries™ ナビゲーター

また、管理環境をセットアップする必要もあります。

『iSeries 用チュートリアル環境のセットアップ』を参照してください。

学習のシナリオ

この学習では、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。

チュートリアル・レッスンの実行方法:

スクリプトおよびコマンドの実行には、いくつかの方法が用意されています。
iSeries ナビゲーターまたは OS コマンド行を使用することができます。

- Windows® 環境でストアード・プロシージャーとしてチュートリアルを実行するには、ナビゲーターを使用します。
- スクリプトおよび SQL ステートメントを実行するには、OS コマンド行を使用します。

関連概念:

- 33 ページの『XML エクステンダーの管理ツール』
- 42 ページの『XML エクステンダーの管理計画』
- 9 ページの『学習: XML 列への XML 文書の保管』
- 19 ページの『学習: XML 文書の合成』

学習: XML 列への XML 文書の保管

XML エクステンダーには、データベースに XML 文書全体を保管および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たうえで、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、頻繁には更新されない文書を保存するアプリケーションで特に役立ちます。

このレッスンは、XML 列の保管とアクセスの方法を示します。

シナリオ:

サービス部門用の営業データを保存する作業を任務として与えられたとします。処理する必要のある販売データは、同じ DTD を使用する複数の XML 文書内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かが通知されています。サービス部門としては、XML 文書を SALES_DB データベース内の SALES_TAB 表に保管する必要があり、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ることになります。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、次の手順を実行します。

1. XML 文書を保管する XML エクステンダーのユーザー定義タイプ (UDT) と、どの XML エレメントおよび属性が頻繁に照会されるかを判別します。
2. XML 用の 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・タイプで保管します。

列を使用可能にするときに、文書アクセス定義 (DAD) ファイル内の文書 (サイド表の構造を指定する XML 文書) の構造検索用に索引を付けるサイド表を定義します。

SALES_TAB 表に関して、表 1 に説明します。 XML 用に使用可能にする XML 列の ORDER は、イタリックで示されています。

表 1. SALES_TAB 表

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

スクリプトとサンプル:

このチュートリアルでは、環境を設定し、学習を進めるために、スクリプトを使用します。 オペレーティング・システム・コマンド行スクリプトは、 dxxsamples ラ

イブラリーの中にあります。ナビゲーター SQL スクリプト・ファイルは、/dxxsamples ディレクトリーの中にあります。

表 2 は、入門学習での作業を行うためのサンプルをリストしています。

表 2. XML コレクションの学習のサンプル

学習の説明	OS コマンド行スクリプト	ナビゲーター SQL スクリプト・ファイル
SALES_DB 表を作成し、埋める	C_SALESDB	C_SalesDb.sql
DTD getstart.dtd を DTD_REF 表に挿入する	INSERTDTD	InsertDTD.sql
XML 列用の SALES_TAB を作成する	C_SALESTAB	C_SalesTab.sql
ORDER 列を SALES_TAB に追加する	ADDORDER	AddOrder.sql
ORDER 列を XML 列として使用可能にする	テキスト中で記述される手動コマンド	EnableCol.sql
サイド表に索引を作成する	C_INDEX	C_Index.sql
XML 文書を SALES_TAB XML 列に挿入する	INSERTXML	InsertXML.sql
sales_tab XML 列にある XML 文書をサイド表を介して照会する	手動コマンド	QueryCol.sql
サンプル表を除去して列を使用不可にする	D_SALESDB および CLEANUPCLL	CleanupCol.sql

文書を保管する方法の計画:

XML エクステンダーを使用して文書を保管する前に、次のことを実行する必要があります。

- XML 文書の構造を理解する。
- XML 文書を入れる XML ユーザー定義タイプを決定する。
- サービス部門が頻繁に検索する XML エレメントと属性を決定する。これは、パフォーマンスを向上させるために、それらの内容をサイド表に保管し、索引を付けられるようにするためです。

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:

この章の学習の XML 文書の構造では、注文キーを使用して構造化されている個々の注文に関する情報を最上位レベルとし、次に顧客、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。

このレッスンには、XML 文書の構造を理解するため、また妥当性検査を実行するために使用できるサンプル DTD が用意されています。

XML 列の XML データ・タイプの決定:

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

XMLVARCHAR
小さい文書を DB2 に保管するのに使用

XMLCLOB
大きい文書を DB2 に保管するのに使用

XMLFILE
DB2 以外に文書を保管するのに使用

この学習では小さい文書を DB2 に保管するので、XMLVARCHAR データ・タイプを使用します。

検索対象のエレメントと属性の判別:

XML 文書の構造とアプリケーションの要件を理解すると、最も頻繁に検索または抽出されるエレメントおよび属性、あるいは照会に最もコストのかかるエレメントおよび属性を判別することができます。サービス部門は、注文キー、顧客名、価格、および注文品の出荷日付を頻繁に照会するため、その検索での迅速なパフォーマンスが必要です。この情報は、XML 文書構造のエレメントと属性に入ります。 表 3 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 3. 検索対象のエレメントと属性

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

サイド表に対する XML 文書のマッピング:

XML 文書をサイド表にマップするには、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、XML 文書を DB2 に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、索引付けに使用される 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、およびどの表にどの列が入るかを決定します。似通った情報を同じ表に入れてサイド表を編成します。いずれかのエレメントのロケーション・パスを文書内で複数回反復できるかどうかによっても文書構造は決まります。例えばこの文書では、パーツ (part) エレメントが複数回繰り返されることがあるので、価格 (price) エレメントと日付 (date) エレメントも複数回出現する可能性があります。複数回出現可能なエレメントは、それぞれのサイド表に含まれ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。

- データがテキストの場合には、VARCHAR を使用します。

- データが整数の場合には、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いときには、DATE を使用します。

このチュートリアルでは、エレメントおよび属性は、
ORDER_SIDE_TAB、PART_SIDE_TAB、または SHIP_SIDE_TAB にマップされます。下記の表は、それぞれのエレメントまたは属性がどの表にマップされるのかを示しています。

ORDER_SIDE_TAB

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

PART_SIDE_TAB

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

SHIP_SIDE_TAB

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

データベースの使用可能化:

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

- ユーザー定義タイプ (UDT)、ユーザー定義関数 (UDF)、およびストアード・プロシージャーを作成します。
- 制御表を作成して、そこに XML エクステンダーが必要とするメタデータを取り込みます。
- DB2XML スキーマを作成して、必要な特権を割り当てます。

XML に対してデータベースを使用可能にする:

以下のいずれかの方法で、データベースを使用可能にします。

- **ナビゲーター:** 次のコマンドを入力します。

  ```
  Run EnableDB.sql
  ```

- **OS コマンド行:** 次のように入力します。

  ```
  CALL PGM(QDBXM/QZXMADM) PARM(enable_db &RDBDatabase)
  ```

SALES_DB 表を作成し、埋める:

学習用の環境をセットアップするには、SALES_DB 表を作成して配置します。これらの表には、計画のセクションで説明した表が含まれています。

以下のいずれかの方法により、表を作成します。

- **ナビゲーター:** 次のファイルを実行します。

  C_SalesDb.sql

- **OS コマンド行:** 次のコマンドを入力します。

```
RUNSQLSTM SRCFILE(DXXSAMPLES/SQLSTMT)
          SRCMBR(C_SALESDB) NAMING(*SQL)
```

XML 列を使用可能にして、文書を保管する:

この学習では、XML エクステンダー用の列を使用可能にし、その列に XML 文書を保管します。それらの作業のために、以下のことを行います。

1. DTD を DTD リポジトリーに保管します。
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。

DTD リポジトリーへの DTD の保管:

DTD を使用して XML 列内の XML データを妥当性検査できます。 XML エクステンダーは、XML 対応のデータベース (DTD_REF) 内に表を作成します。この表は DTD リポジトリーと呼ばれ、DTD を保管するのに使用することができます。
XML 文書を妥当性検査する場合は、DTD をこのリポジトリーに保管する必要があります。このレッスンの DTD は、 dxxsamples/dtd/getstart.dtd にあります。

- **ナビゲーター:** 次のファイルを実行します。

  InsertDTD.sql

- **OS コマンド行:** 次のように入力します。

```
RUNSQLSTM SRCFILE(DXXSAMPLES/SQLSTMT)
          SRCMBR(INSERTDTD) NAMING(*SQL)
```

XML 列のための DAD ファイルの作成:

ここでは、XML 列のための DAD ファイルを作成する方法について説明します。DAD ファイルでは、使用するアクセス方式および保管方式が XML 列であることを指定します。索引付けのための表および列も、この DAD ファイルで定義します。

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の DAD ファイルが、 dxxsamples/dad/getstart_xcolumn.dad にあります。このサンプルは、以下のステップで生成するファイルとは若干異なります。サンプルを学習に使用する場合、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ります。また、サンプルでは、<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、以下の手順を実行します。

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 ファイル内で使用するどのタグでも、大文字小文字の区別があります。

2. XML および DOCTYPE 宣言を使用して DAD ヘッダーを作成します。

   ```
   <?xml version="1.0"?>
   <!DOCTYPE DAD SYSTEM " /dxxsamples/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. 文書の開始タグと終了タグ (<DAD> および </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、開始と終了の (<DTDID> および </DTDID>) タグの付いた DTD ID を挿入して、DTD を指定します。

   ```
   <dtdid> dxxsamples/dtd/getstart.dtd</dtdid>
   ```

   このストリングが、 DTD リポジトリー表への DTD の挿入時の最初のパラメーター値として使用した値に一致することを確認します。例えば、別のマシン・ドライブで作業している場合、 DTDID に使用するパスは、DTD 参照表に挿入したストリングとは異なることがあります。

5. 開始および終了 (<validation> および </validation>) のタグとキーワード YES または NO を挿入します。これにより、XML エクステンダーに、 DTD 参照表に挿入された DTD を使用して XML 文書構造の妥当性検査を行わせるかどうかが指示されます。例えば、

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

   <validation> の値は英大文字で指定しなければなりません。

6. 保管方式が XML 列であることを指定するために、開始および終了 (<Xcolumn> および </Xcolumn>) タグを挿入します。

7. サイド表を作成します。作成するサイド表ごとに、

   a. 生成するサイド表ごとに開始および終了 (<table> および </table>) タグを挿入します。ここで示すように、″name=″ 属性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てください。

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

   b. 表タグの中で、サイド表に含めたいそれぞれの列ごとに <column> タグを挿入します。それぞれの列には、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 サイド表に作られる列の名前を指定します。

type それぞれの索引付きエレメントまたは属性ごとに、サイド表でのデータ・タイプを示します。

path 索引付けするそれぞれのエレメントまたは属性ごとに、 XML 文書内のロケーション・パスを指定します

multi_occurrence

path 属性で参照された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示します。 *multi_occurrence* として可能な値は、 *YES* または *NO* です。値が *NO* である場合、サイド表で複数の列タグを指定することができます。値が *YES* である場合、サイド表に指定できるのは 1 つの列だけです。

```
<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. 終了タグを正しく指定してください。
   - 最後の </table> タグの後に </Xcolumn> 終了タグが必要です。
   - </Xcolumn> タグの後に </DAD> 終了タグが必要です。
9. 次の名前でファイルを保管します。

   ```
   getstart_xcolumn.dad
   ```

作成したファイルを、サンプル・ファイル dxxsamples/dad/getstart_xcolumn.dad と比較できます。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ために必要な DAD ファイルの作業用コピーです。サンプル・ファイルには、絶対パス名を使用するファイルに対する参照が入っています。サンプル・ファイルをチェックして、それらの値を適切なディレクトリー・パスに変更してください。

SALES_TAB 表を作成する:

このセクションでは、SALES_TAB 表を作成します。初期状態では、注文についての販売情報を含む 2 つの列があります。

その表を作成するには、以下のようにします。

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

- **ナビゲーター: C_SalesTab.sql** を実行します。
- **OS コマンド行:** 次のように入力します。

```
RUNSQLSTM SRCFILE(DXXSAMPLES/SQLSTMT)
          SRCMBR(C_SALESTAB) NAMING(*SQL)
```

XML タイプの列を追加する:

新しい列を SALES_TAB 表に追加します。この列には以前に生成した原形の XML 文書が含まれることになり、それは XML UDT のものでなければなりません。XML エクステンダーは複数のデータ・タイプを提供しています。このレッスンでは、文書を XMLVARCHAR として保管します。

XML タイプの列を追加するには、次のようにします。

以下のいずれかの方法で、SQL ALTER TABLE ステートメントを実行します。

- **ナビゲーター: AddOrder.sql** を実行します。
- **OS コマンド行:** 次のように入力します。

```
RUNSQLSTM SRCFILE(DXXSAMPLES/SQLSTMT)
          SRCMBR(ADDORDER) NAMING(*SQL)
```

XML 列を使用可能にする:

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

- サイド表列と結合された XML 文書を入れる XML 列のデフォルトのビューを作成するかどうかを決めます。 XML 文書を照会するときにデフォルトのビューを指定できます。この学習では、ビューを指定するには、`-v` パラメーターを使用します。
- 基本キーを *ROOT ID* と指定し、基本キーの列名をアプリケーション表内に指定し、そしてすべてのサイド表をアプリケーション表に関連付ける固有 ID を指定するかどうかを決めます。基本キーを指定しないと、 XML エクステンダーはアプリケーション表とサイド表に DXXROOT_ID 列を追加します。

  ROOT_ID 列は、アプリケーションとサイド表とを結び付けるキーとして使用され、それによって、XML エクステンダーは XML 文書の更新時にサイド表を自動的に更新することができます。この学習では基本キーの名前は、コマンド (INVOICE_NUM) の `-r` パラメーターで指定します。すると XML エクステンダーは、指定された列を ROOT_ID として使用して、その列をサイド表に追加します。
- 表スペースの指定またはデフォルトの表スペースの使用のどちらを行うかを決めます。この学習では、デフォルトの表スペースを使用します。

XML に対して列を使用可能にするには、次のようにします。

以下のいずれかの方法で、**dxxadm enable_column** コマンドを実行します。

- **ナビゲーター:**

  EnableCol.sql を実行します。

- **OS コマンド行:** 次のように入力します。

```
CALL QDBXM/QZXMADM PARM(enable_column dbname
 Sales_Tab Order
'/dxxsamples/dad/getstart_xcolumn.dad' '-v'
sales_order_view   '-r'  invoice_num)
```

  dbname は、RDB データベースの名前です。

XML エクステンダーは、INVOICE_NUM 列の入ったサイド表を作成し、デフォルトのビューを作成します。

重要: サイド表は決して変更しないでください。サイド表に対する更新は、XML 文書そのものを更新することによってのみ、行う必要があります。 XML 列内の XML 文書を更新すると、XML エクステンダーはサイド表を自動的に更新します。

列およびサイド表を表示する:

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

XML 列およびサイド表の列を表示するには、以下のように行います。

DB2 UDB コマンド行から次の SQL SELECT ステートメントを実行依頼します。

```
SELECT * FROM SALES_ORDER_VIEW
```

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

構造検索のためのサイド表の索引付け:

サイド表に対する索引を作成すると、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、サービス部門から通知されています。 表 4 はそのような列を示しています。これらに索引を付けます。

表 4. 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。

以下のいずれかの方法で、次の CREATE INDEX SQL コマンドを実行します。

- **ナビゲーター: C_Index.sql** を実行します。

- **OS コマンド行:** 次のように入力します。

```
RUNSQLSTM SRCFILE(DXXSAMPLES/SQLSTMT)
          SRCMBR(C_INDEX) NAMING(*SQL)
```

コマンド行: 次のいずれか実行します。

XML 文書の保管:

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

XML 文書を保管するには、次のようにします。

1. XML 文書 dxxsamples/xml/getstart.xml dxxsample/xml/getstart.xmldxxsamples/xml/getstart.xml を開きます。DOCTYPE にあるファイル・パスが、 DTD リポジトリーに DTD を挿入するときに DAD に指定する DTD ID と一致するかどうかを確認してください。この確認は、DB2XML.DTD_REF 表を照会し、 DAD ファイルの DTDID エレメントを調べることによって実行できます。デフォルトとは異なるドライブおよびディレクトリー構造を使用している場合、 DOCTYPE 宣言のパスを変更する必要があります。
2. 以下のいずれかの方法で、SQL INSERT コマンドを実行します。
   - **ナビゲーター: InsertXML.sql** を実行します。
   - **OS コマンド行:** 次のように入力します。

     ```
     RUNSQLSTM SRCFILE(DXXSAMPLES/SQLSTMT)
               SRCMBR(INSERTXML)NAMING(*SQL)
     ```

表が更新されていることを確認してください。コマンド行からその表に対して次の 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、次のようにします。

以下のいずれかの方法で、SQL SELECT コマンドを実行します。

- **ナビゲーター: QueryCol.sql** を実行します。
- **DB2 コマンド行:**

  以下を入力します。

```
select distinct sales_person from schema.sales_tab S,
     part_side_tab P where price > 2500.00
     and S.invoice_num = P.invoice_num;
```

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

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。例えば、

```
SALES_PERSON

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

Sriram Srinivasan
```

関連概念:

- 3 ページの『XML エクステンダーの概要』
- 19 ページの『学習: XML 文書の合成』
- 8 ページの『XML エクステンダーのチュートリアル』

学習: XML 文書の合成

この学習では、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:

与えられた作業では、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、顧客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、サービス部門から通知されています。

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 文書を作成するには、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 ファイルを計画し、作成します。この文書は複数の表から合成されるため、 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 を使用して、XML 文書の構造を定義します。また、これを使用して、合成した XML 文書をアプリケーションで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)
COMMENT	VARCHAR(128)
PART_KEY	INTEGER

計画:

XML エクステンダーを使用して文書を合成する前に、XML 文書の構造と、それをデータベース・データに対応させる方法について決めなければなりません。このセクションでは、サービス部門が指定した XML 文書の構造、 XML 文書の構造の定義に使用する DTD の概略を示します。さらに、文書に取り込むデータの入った列にこの文書をマップする方法についても示します。

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。注文番号を最上位レベルとし、顧客、パーツ、および出荷情報をその下に置く構造の文書をサービス部門は必要としています。文書構造は直感的かつ柔軟で、文書の構造ではなくエレメントがデータを説明することが求められています。 (例えば、顧客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。)

文書構造を設計し終わったら、XML 文書の構造を記述する DTD を作成しなければなりません。このレッスンには、XML 文書および DTD が用意されています。DTD の規則と XML 文書の階層構造を使用して、 21 ページの図 2 に示されているようにデータの階層マップを作成できます。

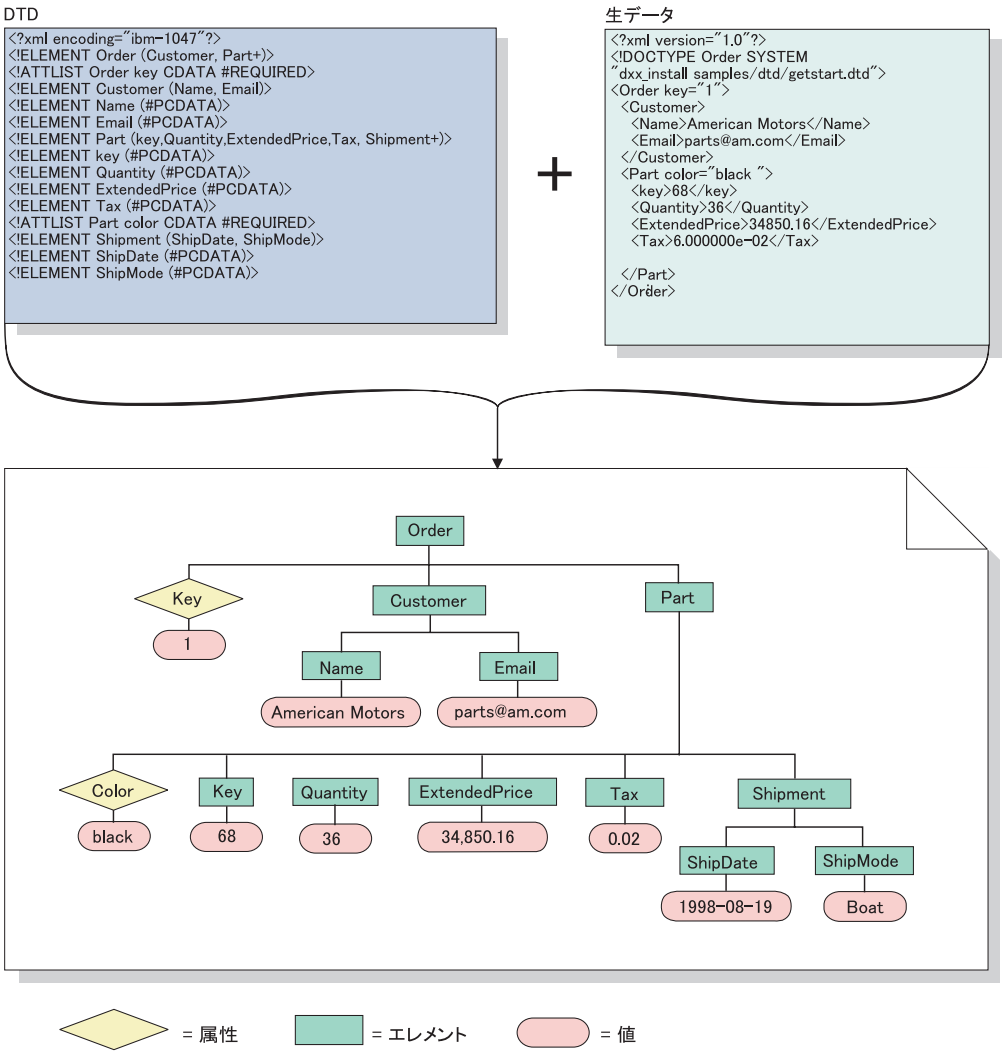


図 2. DTD および XML 文書の階層構造

XML 文書とデータベース関係とのマッピング:

構造を設計して DTD を作成した後、文書の構造とエレメントおよび属性にデータを取り込むために使用する DB2 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。 22 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。

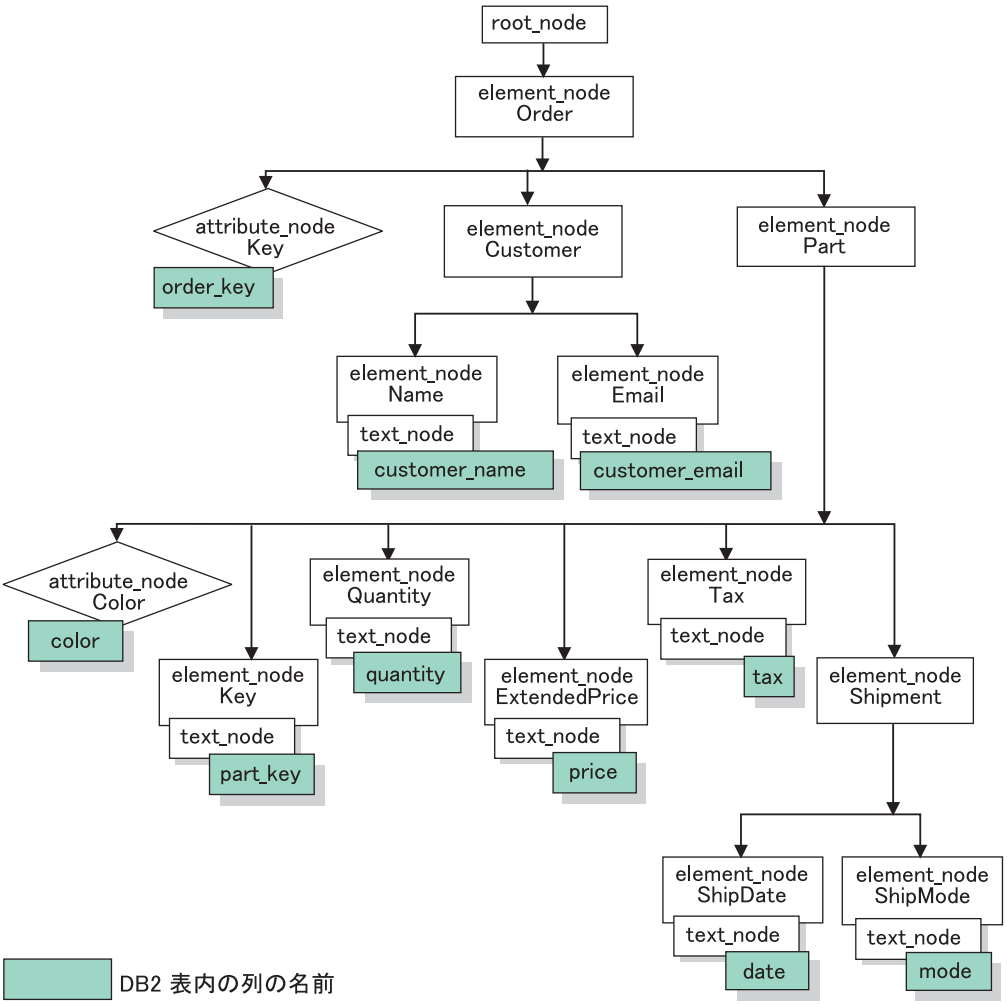


図 3. リレーショナル表の列にマップされた XML 文書

この図では、ノードを使用して XML 文書構造内のエレメント、属性、およびテキストを示しています。これらのノードは、DAD ファイルで使用されますが、詳細は後ほど説明します。

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

このチュートリアルでは、文書の合成に使用する XML コレクションのための DAD ファイルを作成します。 XML コレクションの DAD ファイルは、既存のデータを持つ表を XML 文書構造にマップします。

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。

スクリプトとサンプル:

このレッスン用に、環境を設定するためのスクリプトのセットが用意されています。 OS コマンド行スクリプトは、dxxsamples ライブラリーの中にあります。ナビゲーター SQL スクリプト・ファイルは、/dxxsamples ディレクトリーの中にあります。

表 5 は、入門学習での作業を行うためのサンプルをリストしています。

表 5. XML コレクションの学習のサンプル

学習の説明	OS コマンド行スクリプト	ナビゲーター SQL スクリプト・ファイル
SALES_DB 表を作成し、埋める	C_SALESDB	C_SalesDb.sql
XML 文書を合成し、それを結果表に戻す	手動コマンド	Genxml_sql.sql
サンプル表を除去して列を使用不可にする	D_SALESDB および CLEANUPCOL	CleanupCllec.sql

学習環境のセットアップ:

すでに最初の学習『XML 列への XML 文書の保管』が完了している場合には、このセッションをスキップしてください。このセクションでは、以下の内容を行います。

- データベースを使用可能にする
- 学習で使用する表を作成し配置する

データベースの使用可能化:

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

- ユーザー定義タイプ (UDT)、ユーザー定義関数 (UDF)、およびストアード・プロシージャーを作成します。
- 制御表を作成して、そこに XML エクステンダー が必要とするメタデータを取り込みます。
- DB2XML スキーマを作成して、必要な特権を割り当てます。

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

データベースを XML 処理可能にするには、次のいずれかの方法を使用します。

- **ナビゲーター:** 次のコマンドを入力します。

```
CALL &Schema.QZXMADM('enable_db','
  &DBNAME');
```

- **OS コマンド行:** 次のように入力します。

```
CALL PGM(QDBXM.QZXMADM) PARM(enable_db
  &RDBDatabase)
```

SALES_DB 表を作成し、埋める:

学習用の環境をセットアップするには、SALES_DB 表を作成して配置します。これらの表には、計画のセクションで説明した表が含まれています。

表を作成するには、以下の手順を実行します。

- **ナビゲーター: C_SalesDb.sql** を実行します。
- **(iSeries) OS コマンド行:** 次のコマンドを入力します。

```
RUNSQLSTM SRCFILE(DXXSAMPLES/SQLSTMT)
          SRCMBR(C_SALESDB) NAMING(*SQL)
```

XML コレクションのための 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。そのコレクションを定義するには、DAD ファイルを作成します。

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*dxxsamples/dad/getstart_xcollection.dad にあります。

このサンプルは、以下のステップで生成するファイルとは若干異なります。サンプルを学習で使用する場合、環境によってはファイル・パスが異なり、サンプル・ファイルを変更する必要がある場合がある点に注意してください。

XML 文書を合成するための DAD ファイルを作成するには、次のようにします。

1. dxxsamples/xml ディレクトリーでテキスト・エディターを起動し、getstart_xcollection.dad という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 "/dxxsamples/dtd/dad.dtd">
   ```

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 を DTD リポジトリー表に挿入する際に、 XML エクステンダーが XML 文書構造の妥当性検査を実行するかどうかを示します。この学習では DTD は必要ないため、値は **NO** です。

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

   <validation> タグの値は英大文字で指定しなければなりません。
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> タグの後ろに 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。次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,
    (select db2xml.generate_unique()
as ship_id, date, mode, part_key from ship_tab) as 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、 22 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。例えば、注文と顧客のエレメントの列が第 1、パーツ・エレメントの列が第 2、出荷が第 3 になります。
- データベースからのデータが一緒にまとめられている必要があるテンプレートの、繰り返しセクション、または非繰り返しセクションのための列。各グループに、オブジェクト ID の列である ORDER_KEY、PART_KEY、および SHIP_ID があります。
- 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。例えば、キー属性に関連した列よりも O.ORDER_KEY を前に置き、パーツ・エレメントの列よりも p.PART_KEY を前に置きます。
- SHIP_TAB 表には単一キー条件列はないため、 generate_unique ユーザー定義関数を使用して SHIP_ID 列が生成されます。
- こうすれば、ORDER BY ステートメント内でオブジェクト ID 列はトップダウンの順序でリストされます。 ORDER BY 内の列はスキーマや表名で修飾されず、 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 文書で使用する <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
   "/dxxsamples/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。

22 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 文書を合成するための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″ の属性が指定されています。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、そのため固有のものとなっているため、 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、該当する 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。これは、文書合成時に 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 文書の合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/> タグの中で定義される列は、 <SQL_stmt> SELECT 文節の中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. 終了タグを正しく指定してください。
    - 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 </DAD> タグが </Xcollection> 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。

作成したファイルは、サンプル・ファイル *dxx_install* dxxsamples/dad/getstart_xcollection.dad と比較できます。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、ロケーション・パスとファイル・パス名が入っていますが、正常に実行できるようにするには、個々の環境に合わせて変更しなければならない場合があります。

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

XML 文書を合成する:

このステップでは、dxxGenXML() ストアード・プロシージャーを使用して、 DAD ファイルによって指定された XML 文書を合成します。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 文書を合成するには、次のようにします。

以下の方式のいずれかを使用します。

- **ナビゲーター: Genxml_sql.sql** を実行します。
- **OS コマンド行:** 次のように入力します。

```
CALL DXXSAMPLES/GENX PARM(dbName'/dxxsamples
/dad/getstart_xcollection.dad' result_tab doc ' ')
```

ヒント: このレッスンでは、DB2 UDB ストアード・プロシージャーの結果セット機能を使用して 1 つ以上の合成 XML 文書の生成方法を学びます。結果セットを使用することによって、複数の文書を生成するために複数の行をフェッチすることができます。各文書を生成するたびに、それをファイルにエクスポートできます。結果セットの使用についてデモするには、これが最も簡単な方法です。より効率的なデータの取り出し方法については、 DXXSAMPLES/QCSRC ソース・ファイルにある CLI の例を参照してください。

チュートリアル環境の終結処理:

レッスン環境をクリーンアップするには、提供されているスクリプトのいずれかを実行するか、以下のことを行うためのコマンドをコマンド行から入力します。

- XML 列の ORDER を使用不可にする。
- レッスンで作成した表を除去する。
- DTD リポジトリー表から DTD を削除する。

この章の両方の学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。

以下のいずれかの方法で、クリーンアップ・コマンド・ファイルを実行します。

ナビゲーター:

- XML 列環境を終結処理するには、**CleanupCol.sql** を実行します。
- XML コレクション環境を終結処理するには、**CleanupCllec.sql** を実行します。

OS コマンド行:

- XML 列環境を終結処理するには、次のようにします。
  1. 以下を入力します。

     ```
     RUNSQLSTM SRCFILE(DXXSAMPLES/SQLSTMT)
               SRCMBR(D_SALESDB) NAMING(*SQL)
     ```

  2. 以下を入力します。

     ```
     CALL PGM(QDBXM/QZXMADM)
     PARM(disable_column &DBNAME Sales_Tab Order)
     ```

  3. 以下を入力します。

     ```
     RUNSQLSTM SRCFILE(DXXSAMPLES/SQLSTMT)
               SRCMBR(CLEANUPCOL) NAMING(*SQL)
     ```

- XML コレクション環境を終結処理するには、次のように入力します。

  ```
  RUNSQLSTM SRCFILE(DXXSAMPLES/SQLSTMT)
            SRCMBR(D_SALESDB) NAMING(*SQL)
  ```

関連概念:

- 3 ページの『XML エクステンダーの概要』
- 9 ページの『学習: XML 列への XML 文書の保管』
- 8 ページの『XML エクステンダーのチュートリアル』

第 2 部 管理

ここでは、XML エクステンダー の管理タスクを実行する方法を示します。

第 2 章 管理

XML エクステンダーの管理ツール

XML エクステンダー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 対応にしたり、 XML データを DB2® リレーショナル構造にマップする際に役立ちます。 XML エクステンダーには、管理作業のために使用できる次のコマンド行ツールおよびプログラミング・インターフェースが用意されています。

管理—詳細

iSeries における XML 操作環境

以下のセクションでは、iSeries の XML 操作環境を解説します。

アプリケーション・プログラミング

アプリケーション・プログラム用に提供されているすべての XML エクステンダー・ ファシリティーは、ストアード・プロシージャーまたはユーザー定義関数 (UDF) として iSeries 環境で稼働します。 XML データ・タイプを参照する UDF によっては、IFS システムにアクセスする必要があるものがあります。 DB2 UDB XML トレース・ファイルも、IFS ファイルに書かれます。

XML エクステンダー・ アプリケーションの開発用に 2 つの C ヘッダー・ファイルがあります。これらのファイルには、ストアード・プロシージャー呼び出しやエラー・コード定義の役に立つ定数が含まれています。ヘッダー・ファイルは、製品をインストールすると、以下のディレクトリーから使用することができます。

- `/qibm/proddata/db2extenders/xml/include/dxx.h`
- `/qibm/proddata/db2extenders/xml/include/dxxrc.h`

これらのヘッダーを使用して C++ アプリケーションを開発するには、**CRTCPPMOD** iSeries コマンドで `INCDIR('/qibm/proddata/db2extenders/xml/include')` オプションを使用してください。

管理環境

iSeries 環境で管理タスクを実行するには、XML エクステンダーの管理ウィザード、Qshell、ナビゲーター、またはオペレーティング・システム (OS) のネイティブのコマンド行を使用します。

管理ウィザード

Windows または UNIX クライアント、あるいは iSeries 環境から管理ウィザードを使用して、管理作業を行うことができます。

Qshell

Qshell では、管理コマンド **dxxadm** とそのオプションを実行することができます。管理コマンドについては、 131 ページの『第 6 章 dxxadm 管理コ

マンド』を参照してください。 XML エクステンダーの XML 列、XML コレクションおよびデータベースの管理オプションが用意されています。

ナビゲーター

ナビゲーターで管理ストアード・プロシージャーを呼び出すことができます。管理ストアード・プロシージャーについては、 204 ページの『XML エクステンダーの管理ストアード・プロシージャー』を参照してください。 XML エクステンダーの XML 列、XML コレクションおよびデータベースの管理オプションが用意されています。

OS コマンド行

OS のコマンド行から管理プログラム QZXMADM を実行することができます。このプログラムは、 131 ページの『第 6 章 dxxadm 管理コマンド』に記載されている管理コマンド・パラメーターを使用します。このプログラムには、XML エクステンダーの XML 列、XML コレクションおよびデータベースの管理オプションが用意されています。

次の表に、XML エクステンダーの管理環境の要約を示します。

表 6. XML エクステンダーのストアード・プロシージャーおよびコマンド

環境	Qshell	ナビゲーター	OS コマンド行
サンプル・ファイル	DAD、DTD、および XML ファイルは *dxxsamples* ディレクトリーの下に保管される。 • *dxxsamples*/dtd/dad.dtd • *dxxsamples*/dtd/*.* • *dxxsamples*/dad/*.* • *dxxsamples*/xml/*.*	DAD、DTD、および XML ファイルは *dxxsamples* ディレクトリーの下に保管される。 • *dxxsamples*/getstart.exe • *dxxsamples*/dtd/dad.dtd • *dxxsamples*/dtd/*.* • *dxxsamples*/dad/*.* • *dxxsamples*/xml/*.*	DAD、DTD、および XML ファイルは *dxxsamples* ディレクトリーの下に保管される。 • *dxxsamples*/dtd/dad.dtd • *dxxsamples*/dtd/*.* • *dxxsamples*/dad/*.* • *dxxsamples*/xml/*.*
プログラム実行可能ファイル	DXXSAMPLES ライブラリー。 *dxxsamples*/* 中のシンボリック・リンクによりポイントされる。	iSeries の DXXSAMPLES ライブラリー。 Windows または UNIX では実行可能ファイルなし。	DXXSAMPLES ライブラリー • SQLSTMT ファイル内のサンプル SQL スクリプト • QCLSRC ファイル内のサンプル CL ソース • QCSRC ファイル内のサンプル C++ ソース
コマンド・スクリプト	なし	*path*/DXXSAMPLES にある *.SQL ファイル	DXXSAMPLES/SQLSTMT

XML エクステンダーの管理作業の準備

XML エクステンダーを実行するには、以下のソフトウェアをインストールする必要があります。

iSeries のために必要なソフトウェア:

- DB2 Universal Database™ バージョン 5 リリース 3
- iSeries™ International コンポーネント、ユニコード用 (5722SS1 のオプション 39) QICU

- iSeries System Openness Includes (5722SS1 のオプション 13) QSYSINC
- iSeries Portable App Solutions Environment (5722SS1 のオプション 33) QPASE (コンパイルのために必要)
- WebSphere® Development ToolSet (5722WDS のオプション 51) 51-54 ILE-C ファミリー
- XML Toolkit (iSeries 版) (5733XT1)
- Java™ または Web ベースのアプリケーションを使用するアプリケーションを開発する予定の場合には、 5722JV1 も必要です。

オプションのソフトウェア:

- 構造化テキスト検索のためには、 DB2 Universal Database XML エクステンダー バージョン 7.2。これは、DB2 Universal Database (5722DE1 のオプション 1 および 3) で使用できます。
- XML エクステンダー管理ウィザード用
  - DB2 Universal Database Java Database Connectivity (JDBC)

XML エクステンダーのバージョン 7 からバージョン 8 へのマイグレーション

これまで XML エクステンダーのバージョン 7.2 を使用していた場合は、既存の XML 対応データベースを XML エクステンダーのバージョン 8 で使用する前に、XML エクステンダー対応のデータベースをマイグレーションする必要があります。また、iSeries V5R2 では XML 保管方式が拡張されたため、 iSeries V5R1 の XML 対応列を含む各スキーマをそれぞれマイグレーションする必要があります。

注: DB2 UDB XML エクステンダーのバージョン 8 をインストールする前に、バージョン 7.2 のすべての PTF を適用し、 PTF のカバー・レターに記載されているマイグレーション手順に従う必要があります。

手順:

XML 対応のデータベース、および XML 対応の列を自動的にマイグレーションするには、

1. DB2 UDB XML エクステンダー バージョン 8 をインストールします。
2. オペレーティング・システムのコマンド行で、次のように入力します。

```
CALL QDBXM/QZXMMIGV
```

IASP に関する考慮事項

iSeries™ システムの場合、独立補助記憶域プール (IASP) 装置を外部データベースとして使用することができます。その IASP は、XML エクステンダー対応にすることができます。 IASP データベースを XML エクステンダー対応にする際には、次のことを考慮してください。

- *SYSBAS データベースまたは IASP 上のデータベースを XML エクステンダー対応にすることができます。
- 複数の IASP データベースを XML エクステンダー対応にすることができますが、一度に活動化できる IASP データベースは 1 つだけです。

- *SYSBAS データベースを XML エクステンダー対応にした場合、 IASP データベースを使用可能にすることはできません。

SYSBAS から IASP への XML エクステンダーのデータのマイグレーション

XML データを SYSBAS から IASP に移動するには、詳細な計画が必要です。

次のものについて、さまざまな保管/復元ステップが必要になります。

- XML コレクション・データベース・ファイル
- XML ユーザー定義タイプ (UDT) を持つデータベース・ファイル
- XML 列データベース・ファイル

IASP にマイグレーションするには、次の手順に従います。

1. 『XML 列および XML コレクションの保管および復元』に記載されたステップに従って XML データを保管します。
2. すべての XML 列を使用不可にします。
3. すべての XML コレクションを使用不可にします。
4. XML データ (ステップ 1 で保管したデータ) を含むすべてのスキーマを除去します。
5. XML エクステンダー UDT で定義されたすべての列を除去します。
6. XML エクステンダーで SYSBAS データベースを使用できないようにします。
7. サインオフします。
8. サインオンします。
9. IASP グループに対して SETASPGRP を実行します。
10. XML エクステンダーで IASP データベースを使用できるようにします。
11. 『XML 列および XML コレクションの保管および復元』の指示に従って XML データを復元します。

XML 列および XML コレクションの保管および復元

iSeries では、スキーマの保管および復元手順には次の制約があります。

- DB2XML スキーマ (ライブラリー) の保管、復元、および削除は行わないでください。
- XML エクステンダーによって使用されるデータベース・ファイルを含むユーザー作成のスキーマを復元する際には、以下の条件が適用されます。
  - XML コレクションを含んでおり XML 対応列を含んでいないスキーマは、新規システム上のデータベースが XML エクステンダーで使用可能となっている場合には、 SAVLIB コマンドと RSTLIB コマンドを使用してライブラリー・レベルで復元することができます。旧システムで XML コレクションが使用可能になっていた場合には、再度、新規システムで XML コレクションを使用可能にする必要があります。
  - XML ユーザー定義タイプ (XMLCLOB、XMLVarchar など) の列を含むスキーマは、その列が XML で使用可能となっておらず、かつ、新規システム上のデータベースが XML エクステンダーで使用可能となっている場合には、ライブラリー・レベルで復元することができます。

– XML で使用可能になっていた列を含むスキーマは、ライブラリー・レベルで復元することはできません。基本表およびサイド表 (データベース・ファイル) は、RSTOBJ コマンドを使用してオブジェクト・レベルで復元することができます。

以下の手順は、XML コレクションおよび XML 列とともに使用されるデータベース・ファイルを含むスキーマを復元する方法を示しています。

XML コレクションのデータベース・ファイルを復元するには、次の手順を実行してください。

1. ターゲット・システム上のデータベースを XML エクステンダーで使用できるようにします。
2. RSTLIB を使用して XML コレクションのデータベース・ファイルを復元します。
3. XML コレクションが元のシステムで使用可能になっていた場合には、enable_collection コマンドを実行して、 XML コレクションをターゲット・システムで使用可能にします。

XML ユーザー定義タイプを含むデータベース・ファイルを復元するには、次の手順を実行してください。

1. ターゲット・システム上のデータベースを XML エクステンダーで使用できるようにします。
2. RSTLIB コマンドを使用してデータベース・ファイルを復元します。

XML 列のデータベース・ファイルを復元するには、次の手順を実行してください。

1. ターゲット・システム上のデータベースを XML エクステンダーで使用できるようにします。
2. RSTOBJ コマンドを使用して基本表を復元します。
3. 基本表で定義されていた古いトリガーを、RMVPFTRG コマンドを使用してすべて除去します。
4. ターゲット・システムで XML 列を使用可能にします。前のシステムで基本表を使用可能にするために「-r」パラメーターが使用されていた場合には、基本表の基本キーを識別するために「-r」パラメーターを使用しなければなりません。
5. ADDPFTRG コマンドを使用してユーザー定義のトリガーを基本表に追加し、それらのプログラムをターゲット・システムに復元します。
6. RSTOBJ コマンドを使用してデータをサイド表に復元します。

制約事項: 以下の理由により、XML 対応の列を含むデータベース・ファイルは、RSTLIB コマンドで復元することはできません。

- XML エクステンダーに保管されている重要なメタデータは、ライブラリーおよびデータベース・ファイルを復元する際には、新規システムに復元されません。このメタデータは、enable_column コマンドを実行することのよってのみ、ターゲット・システムに作成することができます。
- RSTLIB を使用してライブラリーを復元すると、前提条件となるメタデータが XML エクステンダーから欠落するため、ライブラリー内の SQL トリガーが使用できなくなります。このようなトリガーが存在すると、enable_column コマンドを実行できなくなります。

iSeries のための XML エクステンダー・ サンプルおよび開発環境のセットアップ

以下のセクションでは、アプリケーション使用の計画に応じて管理環境をセットアップする方法を説明します。

- すべての環境用 - 『サンプル・ファイルおよび入門用ファイルのアンパックと復元』
- すべての環境用 - 39 ページの『サンプル用の SQL コレクション (スキーマ) の作成』
- 選択した管理環境用
  - ウィザードを使用 - 40 ページの『ウィザードのセットアップ』
  - Qshell コマンド行を使用 - 40 ページの『Qshell のセットアップ』
  - ナビゲーターを使用 - 40 ページの『ナビゲーター・インターフェースの設定』
  - OS コマンド行を使用 - 41 ページの『iSeries コマンド行用のサンプル・プログラムの作成』
- 入門用のレッスンを実行 - 41 ページの『iSeries のチュートリアル環境のセットアップ』

サンプル・ファイルおよび入門用ファイルのアンパックと復元

サンプルは、2 つの iSeries 保管ファイル・オブジェクトとして、プロダクト・ディレクトリーに入れて出荷されます。これらのファイルは、以下のとおりです。

QDBXM/QZXMSAMP1

DXXSAMPLES ライブラリーのための SAVLIB 保管ファイルが入っています。このライブラリーには、アプリケーション開発用のサンプルの C と CL ソース・コード、C ヘッダー・ファイルおよび SQL ステートメントがあります。

QDBXM/QZXMSAMP2

IFS ディレクトリー・ツリーの SAV 保管ファイルが入っています。このファイルには、ナビゲーターで使用するサンプル XML、DTD とデータ・アクセス定義 (DAD) ファイル、および自己解凍の `GetStart.exe` が含まれます。

管理環境を使用するための最初の準備ステップとして、 iSeries 管理者は、これらの保管ファイルをアンパックしてシステムに復元します。

管理者は次のことを行う必要があります。

- QDBXM/QZXMSAMP1 保管ファイルをアンパックして、サンプル・ソース・コードと SETUP プログラムをシステムにリストアする。

  OS のコマンド行から、次のように入力します。

```
RSTLIB SAVLIB(DXXSAMPLES)
   DEV(*SAVF)
   SAVF(QDBXM/QZXMSAMP1)
```

  RSTLIB コマンドは保管ファイルを DXXSAMPLES ライブラリーにアンパックします。これには、 39 ページの表 7 にリストされているオブジェクトが含まれます。

表 7. *DXXSAMPLES* ライブラリー・オブジェクト

オブジェクト	種類	属性	説明
SETUP	PGM	CLP	サンプル・プログラムをコンパイルし、IFS を追加
H	FILE	PF-SRC	C ヘッダー・ファイル
QCLSRC	FILE	PF-SRC	ナビゲーターのためのインターフェース
QCSRC	FILE	PF-SRC	サンプル・プログラム
SQLSTMT	FILE	PF-SRC	サンプルの SQL ステートメント

- QDBXM/QZXMSAMP2 保管ファイルをアンパックして、 XML ファイルと GetStart.exe をユーザーのシステムにリストアする。

  OS のコマンド行から、次のように入力します。

  ```
  RST DEV('/qsys.lib/qdbxm.lib/qzxmsamp2.file')
     OBJ(('/QIBM/UserData/DB2Extenders/XML/Samples'))
  ```

  RST DEV コマンドは、保管ファイルを IFS ディレクトリー `/QIBM/UserData/DB2Extenders/XML/Samples` の XML ファイルに復元します。

サンプルのセットアップ中にシンボリック・リンク `/dxxsamples` が作成され、 IFS ディレクトリー `/QIBM/UserData/DB2Extenders/XML/Samples` をポイントします。本書中の用語 *dxxsamples* は、これらのいずれかの値を表します。

サンプル用の SQL コレクション (スキーマ) の作成

サンプル・データの一連のストアード・プロシージャーとテーブルがスキーマで作成されるため、サンプルを実行するには、スキーマが必要です。

SQL スキーマを作成する際には、SQL におけるデフォルトのスキーマ規則に従い、サンプルの実行時に使用するユーザー ID にスキーマ名を一致させることをお勧めします。

このユーザー ID に一致する SQL スキーマがある場合は、新規にスキーマを作成する必要はありません。

スキーマの作成方法

1. OS のコマンド行から、次のように入力して SQL セッションをオープンします。

   ```
   STRSQL
   ```

2. SQL セッションから、次のように入力します。

   ```
   CREATE SCHEMA UserId
   ```

   UserId は、このサンプルの実行する際に使用するユーザー ID です。

iSeries のための管理ツールのセットアップ

管理用に使用可能なツールが 33 ページの『管理環境』に記載されています。これらのいずれかの環境を選択して管理タスクを行うことができます。これらの環境には、XML エクステンダー管理コマンド、ストアード・プロシージャー、およびサンプルを使用してセットアップを行う必要のあるものがあります。以下のセクションでは、セットアップのための要件について説明します。

ウィザードのセットアップ

次の XML エクステンダー Web サイトを参照してください。

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

Qshell のセットアップ

Qshell オプションのインストールを除き、セットアップは必要ありません。

ナビゲーター・インターフェースの設定

iSeries ナビゲーターを使用して、管理コマンド、SQL ステートメント、ストアード・プロシージャーの実行、および入門学習を行うことができます。 iSeries ナビゲーターを使用する場合は、以下の手順によりサンプル・プログラムをダウンロードして作成します。サンプル・プログラムを作成することにより、管理ストアード・プロシージャーの実行環境の準備ができます。この作業は必須です。

1. `GetStart.exe` ファイルをダウンロードし、アンパックします。
   a. Windows オペレーティング・システムに *path*`/dxxsamples` ディレクトリーを作成します。 *path* は、`dxxsamples` ディレクトリーを置くドライブとディレクトリーです。
   b. Windows のコマンド行から、次のように入力します。

      `FTP` *SystemId*

      SystemId は、*dxxsamples* ディレクトリーに保管ファイルを復元した iSeries システムのホスト名です。

      要求されたユーザー ID とパスワードを入力します。
   c. FTP コマンド `Binary` を入力して、バイナリー・モードに変更します。
   d. 次の FTP コマンドを入力して、getstart.exe ファイルを `dxxsamples` ディレクトリーに移動します。

      `get` *dxxsamples*`/getstart.exe` *path*`/dxxsamples/getstart.exe`
   e. 次のコマンドにより、FTP セッションをクローズします。

      `exit`
   f. *path*`/dxxsamples` ディレクトリーから、次のように入力します。

      `GetStart.exe`
2. iSeries ナビゲーターを開始します。
3. iSeries システムのツリーを展開して、「**データベース (Database)**」を右クリックします。メニューが表示されます。
4. メニューで、「**Run SQL Scripts**」をクリックします。
5. *path*`/dxxsamples/setup.sql` スクリプト・ファイルをオープンします。

6. すべての &SCHEMA を、 39 ページの『サンプル用の SQL コレクション (スキーマ) の作成』で作成したスキーマ名に変更します。
   a. メニューにある「**編集 (Edit)**」→「**置換 (Replace)**」をクリックします。「検索と置換 (Search and Replace)」ウィンドウが表示されます。
   b. 「検索と置換 (Search and Replace)」ウィンドウから、すべての &SCHEMA を実際のスキーマ名で置き換えます。
7. すべての &DBNAME を、サンプル・プログラムを実行するシステムの RDB データベース名に置き換えます。この名前を判別するには、 OS のコマンド行から **WRKRDBDIRE** コマンドを実行します。登録済みデータベースのリストから、リモート・アドレス *LOCAL の付いた名前を選択します。
   a. メニューにある「**編集 (Edit)**」→「**置換 (Replace)**」をクリックします。「検索と置換 (Search and Replace)」ウィンドウが表示されます。
   b. 「検索と置換 (Search and Replace)」ウィンドウから、すべての &DBNAME をローカル・データベース名で置き換えます。
8. setup.sql ファイルを保管します。
9. それぞれの SQL ファイルについて 5 から 8 までのステップを繰り返します。
10. *path*/dxxsamples/Setup.sql スクリプト・ファイルをオープンして、「**すべて実行 (Run all)**」をクリックします。

これで、iSeries ナビゲーターを使用して入門学習を始めることができます。

作成したサンプル・プログラムは、管理コマンドおよび DB2 UDB コマンドの入力に使用することができます。

iSeries コマンド行用のサンプル・プログラムの作成

OS コマンド行を使用して、管理コマンドを実行し、入門学習を完了することができます。

管理コマンドを実行するためのセットアップは不要です。

サンプルおよび入門学習用に OS コマンド行を使用する場合は、 SETUP を実行してすべてのサンプル・プログラムを作成してください。 OS のコマンド行から、次のように入力します。

```
CALL DXXSAMPLES/SETUP
```

iSeries のチュートリアル環境のセットアップ

以下のセクションでは、「入門」学習を行うための環境のセットアップに必要な手順を説明します。この環境は、提供されているサンプルを使用して独自のアプリケーションを開発するうえで役立ちます。

以下の環境を使用して、入門学習を完了することができます。

- iSeries ナビゲーター
- OS コマンド行

これらの環境を使用するには、以下のことを行う必要があります。

1. サンプル・ソース・コードをサンプル・ライブラリーにリストアする。 38 ページの『サンプル・ファイルおよび入門用ファイルのアンパックと復元』のステップに従ってください。
2. XML サンプル・ファイルと入門のところで使用する実行可能ファイルを IFS ディレクトリーにリストアする。 38 ページの『サンプル・ファイルおよび入門用ファイルのアンパックと復元』のステップに従ってください。
3. SQL スキーマ (コレクション) を作成する。 39 ページの『サンプル用の SQL コレクション (スキーマ) の作成』のステップに従ってください。
4. 管理作業を行う環境をセットアップする。
   - iSeries ナビゲーターを使用 - 40 ページの『ナビゲーター・インターフェースの設定』
   - OS コマンド行を使用 - 41 ページの『iSeries コマンド行用のサンプル・プログラムの作成』

XML エクステンダーの管理計画

XML 文書を使用するアプリケーションについて計画する際には、最初に以下のことを決定しておく必要があります。

- XML 文書をデータベースのデータから合成するかどうか。
- 既存の XML 文書を保管するかどうか。また、XML 文書を保管する場合には、それらをそのまま XML 文書として列に保管するのか、あるいは通常の DB2® データに分解して保管するのかも決定する必要があります。

これらの点が決まると、以下のことを決定できます。

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

XML エクステンダーをどのように使用するかは、アプリケーションが何を必要とするかに依存します。 XML 文書を既存の DB2 UDB データから合成して、 XML 文書を原形の文書または DB2 UDB データとして DB2 内に保管することができます。これらの保管方式とアクセス方式に応じて計画上の要件は異なります。

アクセスと保管の方法

XML エクステンダーには、DB2® を XML リポジトリーとして使用するための 2 つのアクセスおよび保管の方式として、 XML 列と XML コレクションが備わっています。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合成します。

ご使用のアプリケーションの性質によって、最適なアクセスおよび保管方式、および XML データを構成する方式が異なります。

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。 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。

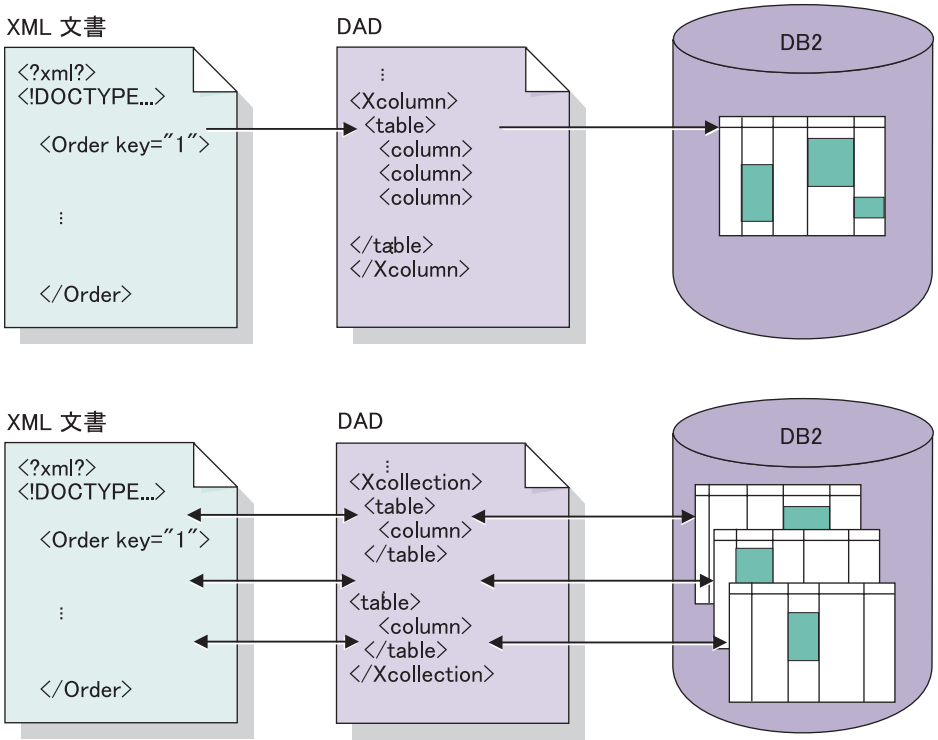


図 4. DAD ファイルは XML 文書構造を DB2 UDB リレーショナル・データ構造にマップして、アクセスおよび保管の方式を指定します。

DAD ファイルは DTD などの主要ファイルの位置を定義して、 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:

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

関連参照:

- 146 ページの『XML エクステンダーの保管関数の概要』

XML 列方式を使用する場合

XML 列は以下のいずれかの状況で使用します。

- XML 文書がすでに存在するか、外部ソースから送られてきて、その文書をネイティブの XML 形式で保管したいとき。整合性のため、および保存と監査のために、それらを DB2® に保管することをお勧めします。
- XML 文書が頻繁に読み取られるものの、更新されないとき。

- ファイル名データ・タイプを使用して XML 文書 (DB2 UDB 外のもの) 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場合。
- XML エレメントまたは属性に基づく範囲検索を行う必要があり、検索引き数として頻繁に使用されるエレメントまたは属性がわかっている場合。
- 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 を使用して構造化テキスト検索を実行したい場合。

XML コレクション方式を使用する場合

XML コレクションは以下のいずれかの状況で使用します。

- 既存のリレーショナル表にデータがあり、特定の DTD に基づいて XML 文書を合成したいとき。
- 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・データの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。データは重要であっても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場合で、データを既存の表に保管するのか、新しい表に保管するのかを柔軟に決めたい場合。
- 着信 XML 文書の全体のデータを保管する必要があるものの、取り出すのはそのサブセットのみであることが多いとき。

XML 列について計画する

文書を保管するために XML エクステンダーで作業を始める前に、文書内のエレメントと属性にどのように索引を付けるかを決めるために、 XML 文書の構造を理解する必要があります。文書に索引を付けるには、以下のことを判断する必要があります。

- XML 文書を保管するのに使用する 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。これは、パフォーマンスを向上させるために、それらの内容をサイド表に保管し、索引を付けられるようにするためです。
- DTD により、列で XML 文書の妥当性検査をするかどうか。
- サイド表の構造と、どのように索引を付けるか。

XML 列の XML データ・タイプの決定

XML エクステンダーには、 XML 文書を入れる列を定義するために使用する XML ユーザー定義タイプが提供されています。これらのデータ・タイプについては、 45 ページの表 8 で説明しています。

表 8. XML エクステンダーの 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。小さい文書を DB2 に保管するために使用されます。
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。大きい文書を DB2 に保管するために使用されます。
XMLFILE	VARCHAR(512)	XML 文書のファイル名を DB2 内に保管して、 XML 文書を DB2 UDB サーバーにとってローカルのファイル内に保管します。文書を DB2 の外に保管するために使用されます。

索引を付け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要件を理解すると、どのエレメントと属性を検索すればよいかを判別できます。多くの場合、それらは、最も頻繁に検索または抽出されるエレメントと属性、あるいは照会に最もコストのかかるエレメントと属性です。各エレメントと属性のロケーション・パスは、これらのオブジェクトが入っているリレーショナル表 (サイド表) に XML 列のための DAD ファイルでマップできます。そして、サイド表に索引が付けられます。

例えば 表 9 は、「入門」のシナリオでの XML 列のためのエレメントと属性のロケーション・パスおよびデータ・タイプの例を示しています。このデータは、頻繁に検索される情報として指定され、ロケーション・パスはデータが入っているエレメントと属性を指しています。したがって、これらのロケーション・パスを DAD ファイルでサイド表にマップできます。

表 9. 検索対象のエレメントと属性

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

DAD ファイル

XML 列に対して、DAD ファイルはクライアント側にある XML 形式の文書であり、主に XML 列に保管された文書にどのように索引を付けるかを指定します。DAD ファイルは、XML 列に挿入される文書の妥当性検査を行うために使用する DTD を指定します。 DAD ファイルのデータ・タイプは CLOB です。このファイルは最大 100 KB まで可能です。

XML 列の DAD ファイルは、索引付けのためにサイド表に保管する XML データのマップを提供します。

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

<Xcolumn>

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

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

XML コレクションについて計画する

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

妥当性検査

アクセスおよび保管の方式を選択した後、データの妥当性検査を行うかどうかを決めることができます。 XML データの妥当性検査は、DTD またはスキーマを使用して実行します。 DTD またはスキーマを使用することにより、 XML 文書が有効であることを確認できます。

DTD を使用して妥当性検査を実行するには、

XML エクステンダー・ リポジトリーの中に DTD が必要です。 DTD をリポジトリーに挿入する方法については、 59 ページの『リポジトリー表への DTD の保管』を参照してください。

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

考慮事項:

- 合成に使用できる DTD は 1 つだけです。
- 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実行しない場合、その XML 文書で指定された DTD は処理されません。妥当性検査を実行できない文書の断片を処理する場合でも、エンティティーや属性のデフォルト値を解決するために、DTD を処理することは重要です。

DAD ファイル

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

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

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

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

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

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

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

root_node
: 文書のルート・エレメントを指定します。

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

text_node
: エレメントの CDATA テキストを表します。

attribute_node
: エレメントの属性を表します。

48 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一部を示しています。このノードは、XML 文書の内容をリレーショナル表内の表列にマップします。

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

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

図 5. ノード定義

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

XML エクステンダー管理ウィザードまたはエディターを使用して、 DAD ファイルの作成および更新を実行できます。

XML コレクションのマッピング体系

XML コレクションを使用している場合、 XML データをリレーショナル・データベース内で表す方法を定義するマッピング体系 を選択しなければなりません。 XML コレクションはリレーショナル構造の XML 文書内で使用される階層構造と一致しなければならないため、それら 2 つの構造の相違点を理解している必要があります。 49 ページの図 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。

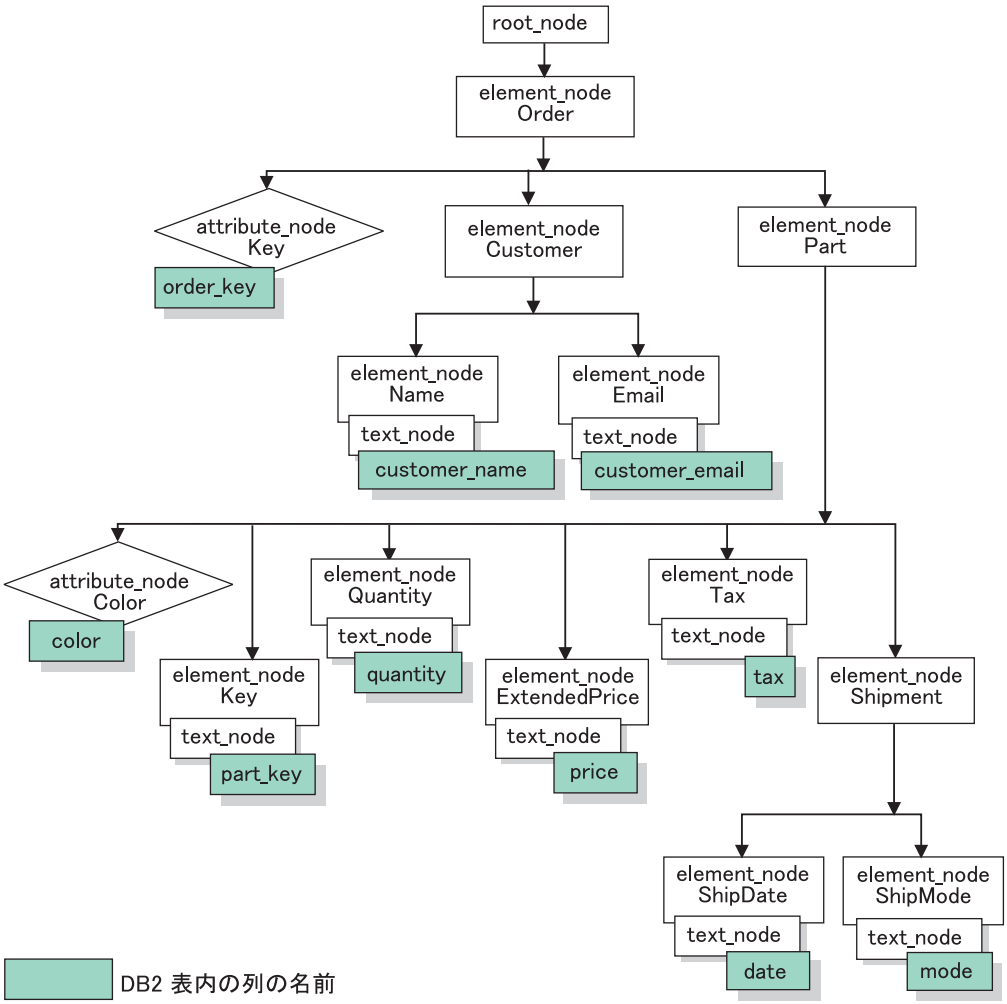


図 6. リレーショナル表の列にマップされた XML 文書構造

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

マッピング体系の種類: マッピング体系は、DAD ファイル内の <Xcollection> エレメントに指定されます。 XML エクステンダーには、*SQL* マッピング とリレーショナル・データベース *(RDB_node)* マッピング という、 2 種類のマッピング体系が備わっています。

SQL マッピング

リレーショナル・データから XML 文書への直接的なマッピングを可能にします。これは、単一の SQL ステートメントと *XPath* データ・モデル を介して行います。SQL マッピングは合成に使用され、分解には使用されません。 SQL マッピングは DAD ファイル内の SQL_stmt エレメントによって定義されます。 SQL_stmt の内容は、有効な SQL ステートメントです。SQL_stmt は、SELECT 文節の列を、XML 文書で使用される XML エレメントまたは属性にマップします。 XML 文書の合成のために定義されたと

き、SQL ステートメントの SELECT 文節にある列名が、 *attribute_node* の値または *text_node* の内容を定義するために使用されます。 FROM 文節は、データを含む表を定義します。 WHERE 文節は、*結合* と検索*条件* を指定します。

SQL マッピングにより、DB2 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。すべての表を結び合わせるため、これらの表に*基本キー* と*外部キー* の関係があることが推奨されます。

RDB_node マッピング

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

この方式では、表、オプションの列、およびオプションの条件についてのノード定義が 1 つ以上入っている、 XML エクステンダーが提供する *RDB_node* を使用します。表および列は、XML データをデータベースに保管する方法を定義するために使用されます。条件は、XML データの選択基準、または XML コレクション表を結合する方法を指定します。

マッピング体系を定義するには、<Xcollection> エレメントのある DAD を作成します。 51 ページの図 7 は、 3 つのリレーショナル表内のデータから一組の XML 文書を合成する、 XML コレクション SQL マッピングのあるサンプルの DAD ファイルの一部を示しています。

```
<?xml version="1.0"?>
<!DOCTYPE DAD SYSTEM "dxx_install/dtd/dad.dtd">
<DAD>
  <dtdid>dxx_install/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,
               (select db2xml.generate_unique()
               as ship_id, date, mode, from ship_tab) as
 s
        WHERE p.price > 2500.00 and s.date > "1996-06-01" AND
              p.order_key = o.order_key and s.part_key = p.part_key
    </SQL_stmt>
    <prolog>?xml version="1.0"?</prolog>
    <doctype>!DOCTYPE DAD SYSTEM "dxx_install/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 エクステンダーには、 XML コレクション内のデータを管理するいくつかのストアード・プロシージャーが備わっています。これらのストアード・プロシージャーは両方のタイプのマッピングをサポートしますが、 DAD ファイルは 『マッピング体系の要件』で説明されている規則に従う必要があります。

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

SQL マッピングのための要件

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

- **SELECT 文節**
  - 列の名前が固有のものであることを確認してください。 2 つの表に同じ列名がある場合、AS キーワードを使用していずれか一方に別名を作成します。
  - 同じ表の列をグループ化して、リレーショナル表の論理階層レベルを使用します。つまり、XML 文書の階層構造にマップしたときの重要度

のレベルに応じて、表をグループ化するということです。 SELECT 文節では、より高いレベルの表の列はより低いレベルの表の列より前に指定します。以下の例は、複数の表の間での階層関係を例示しています。

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

この例で、表 ORDER_TAB の order_key および customer は、XML 文書の階層ツリーで高いレベルにあるために、最高のリレーショナル・レベルを持ちます。表 SHIP_TAB の ship_id、date、および mode は、最低のリレーショナル・レベルにあります。

– 単一列候補キーを使用して、各レベルを開始します。そのようなキーが表にない場合、照会では表式およびユーザー定義関数 generate_unique() を使用してそのキーを生成しなければなりません。上記の例で、o.order_key は ORDER_TAB の基本キー、そして part_key は PART_TAB の基本キーです。それらは、選択されるそれぞれの列グループの先頭にあります。 SHIP_TAB には基本キーがないため、生成しなければなりません。この場合、ship_id となります。それは SHIP_TAB 表グループの最初の列としてリストされています。以下の例に示されているように、FROM 文節を使用して基本キー列を生成します。

- **FROM 文節**
  – 表式およびユーザー定義関数 generate_unique() を使用して、基本単一キーのない表に単一キーを生成します。例えば、

```
FROM order_tab as o, part_tab as p,
       (select db2xml.generate_unique() as
       ship_id, date, mode from ship_tab) as s
```

この例では、単一の列候補キーが generate_unique() 関数で生成され、別名 ship_id が付けられます。

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

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

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

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

- **ORDER BY 文節**
  – ORDER BY 文節を SQL_stmt の最後に定義します。
  – 列名が SELECT 文節内の列名に一致していることを確認します。
  – データベースのエンティティー関連 (ER) 設計でエンティティーを固有に識別する列名または ID を指定します。 ID は、表式および関数 generate_unique またはユーザー定義関数 (UDF) を使用して生成することができます。

– エンティティーの階層のトップダウン順序を保持します。 ORDER BY 文節で指定される列は、各エンティティーについてリストされる最初の列でなければなりません。順序を維持すれば、生成される XML 文書に誤った重複が入らないようにすることができます。
– ORDER BY 内の列をスキーマ名または表名で修飾しないでください。

SQL_stmt には前述の要件がありますが、述部の式がその表の列を使用する限り WHERE 文節に任意の述部を指定できることから、非常に強力です。

RDB_node マッピングを使用する際の要件

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

ルート・ノード条件の述部の順序に関する制限はありません。

- **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。

  – 条件ステートメントでは、行終了文字を使用することができます。
  – 条件エレメントでは、1 つの列名を何回でも参照することができます。
  – XML 文書に関連したすべての表を指定します。例えば、以下のマッピングは、先頭 element_node である element_node <Order> の RDB_node 内に 3 つの表を指定します。

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

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

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

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

  分解のために指定された情報は、文書を合成するときには無視されます。

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

表名および列名は、明示的に略さずに指定しなければなりません。

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

  このマッピング体系では、データは各 element_node に対する attribute_node および text_node 内に存在します。そのため、XML エクステンダーはデータベースのどこからデータを見つけるかを知る必要があります。 attribute_node および text_node ごとに RDB_node を指定して、ストアード・プロシージャーにどの表、どの列、そしてどの照会条件からデータを取得するかを指定します。表および列の値は指定しなければなりません。条件の値はオプションです。

  – 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。指定できる条件には、次のものがあります。

    - columname (列名)
    - operator (演算子)
    - literal (リテラル)

    上記の例では、条件は `price > 2500.00` として指定されています。条件に適合するデータのみが、生成される XML 文書に含まれます。条件は有効な WHERE 文節でなければなりません。

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

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

分解のために指定された情報は、文書を合成するときには無視されます。

- 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。これは、合成または分解において、 XML エクステンダーがエレメント間の関係を理解できるようにエレメント・ノードが適切にネストされるようにすることを意味します。例えば、次のように Shipment を Part 内にネストしていない DAD ファイルを使用すると、

```
<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-->
```

これは Part および Shipment が兄弟エレメントである XML ファイルを生成する場合があります。

```
<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>
```

Shipment を Part の内部でネストする DAD にしたい場合、

```
<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 -->
```

Which produces an XML file with Shipment as a child element of Part:

```
<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>
```

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。

分解表サイズの要件

分解では RDB_node マッピングを使用して、エレメントと属性値を抽出し表行に入れて 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。

例えば、XML 文書を 5 つの表に分解する場合、その 5 つの表のそれぞれに、該当する文書中の 10240 行までを入れることができます。複数の文書のための行を含む表でも、各文書につき 10240 行までを入れることができます。

複数出現エレメント (XML 構造内で複数出現する可能性のあるロケーション・パスをもつエレメント) を使用すると、各文書に挿入される行数に影響します。例えば、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場合、単一の文書から 1 つの表に分解できる最大の行数は 10240 である点に注意してください。

XML 文書の自動妥当性検査

アクセスおよび保管の方式 (XML 列または XML コレクション) を選択した後で、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。 XML 文書を保存目的で保管するのでなければ、 DB2 に保管する前にまず妥当性検査を実行することをお勧めします。 XML コレクションから構成される XML 文書も妥当性検査可能です。

DAD ファイル内で妥当性検査について YES を指定すると、 XML データの妥当性検査が自動的に実行されるようになります。文書を DB2 に保管する際にその妥当性検査が実行されるようにするには、元の文書の <dtdid> エレメントの中か、または <!DOCTYPE> 指定の中で DTD を指定する必要があります。 DB2 において XML コレクションから文書を合成する際にその妥当性検査が実行されるようにするには、 DAD ファイルの <dtdid> エレメントの中か、または <doctype> エレメントの中で 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める際には、以下の事項を考慮する必要があります。

- スキーマを使用して DAD の妥当性検査を実行するには、その 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。例えば、

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

DTD ID またはスキーマは、XML 文書の妥当性検査を行う場合にのみ有効です。

- XML 文書の保管またはアーカイブには、DTD は必要ありません。
- XML データを DB2 に挿入する前に、妥当性検査を行うかどうかを決める必要があります。 XML エクステンダーは、すでに DB2 に挿入されているデータの妥当性検査は実行しません。
- 妥当性検査を実行するかどうかに関係なく、エンティティー値および属性のデフォルトを設定するために DTD を処理する必要があります。
- DAD で妥当性検査に NO を指定した場合、その XML 文書によって指定された DTD は処理されません。
- XML データの妥当性検査を行うと、パフォーマンスが影響を受けます。

XML 用のデータベースの使用可能化

XML エクステンダーを使用して XML 文書を DB2 UDB から保管または取り出すには、その前にデータベースを XML 対応可能にします。 XML エクステンダーを使用すると、接続先のデータベースを使用可能にできます。

データベースを XML 用に使用可能にすると、XML エクステンダーは以下のことを実行します。

- 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・プロシージャーのすべてを作成します。
- 制御表を作成して、そこに XML エクステンダーが必要とするメタデータを取り込みます。
- ユーザー定義表スペース内に 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。この db2xml は、SQL オブジェクトを論理グループ化するための ID です。 UDF または UDT を参照するときには、いかなる場所でも完全修飾名を使用できます。また、UDF や UDT を参照する際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、必要な関数またはデータ・タイプを判別します。

iSeries では、*SYSBAS データベースまたは独立補助記憶域プール (IASP) のどちらを使用しても、データベースを XML エクステンダーで使用可能にすることができます。複数の IASP データベースを XML エクステンダーで使用可能にすることができます。一度に活動化できる IASP データベースは 1 つのみです。 *SYSBAS データベースが使用可能になっている場合には、 IASP データベースを XML エクステンダーで使用可能にすることができません。

手順:

データベースを使用可能にする作業は、管理ウィザードを使用して実行するか、コマンド行から実行できます。

以下の例では、SALES_DB という既存のデータベースが使用可能にされます。

```
CALL QDBXM/QZXMADM PARM(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、以下の作業を行う必要があります。

1. 管理ウィザードを開始して、「ランチパッド (LaunchPad)」ウィンドウで**「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は、ボタンには**「データベースを使用不可にする (Disable database)」**と表示されます。データベースが使用不可になっている場合は、ボタンには**「データベースを使用可能にする (Enable database)」**と表示されます。

   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すると、XML エクステンダーを使用して XML 文書を DB2 UDB に保管したり、 DB2 UDB から検索したりできるようになります。

関連概念:

- 35 ページの『XML エクステンダーのバージョン 7 からバージョン 8 へのマイグレーション』

XML 表の作成

この作業は、XML 列の定義および使用可能化という、より大きな作業の一部です。

XML 表は、XML 文書をそのまま保管するために使用されます。 DB2 UDB XML エクステンダーを使用して文書全体をデータベースに保管するには、 XML ユーザー定義タイプ (UDT) の列を含むように表を作成しなければなりません。 DB2 UDB XML エクステンダーには、 XML 文書を列データとして保管するためのユーザー定義タイプ (UDT) として、 XMLVARCHAR、XMLCLOB、および XMLFILE の 3 つが備わっています。表に XML タイプの列が含まれている場合には、その表を XML 対応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を使用して、表を新規に作成し、XML タイプの列を追加することができます。

手順:

XML タイプの列を含む表を、コマンド行を使用して作成するには、次の手順を実行してください。

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。

例えば、販売アプリケーションで、 XML 形式の行項目である注文を、SALES_TAB という表の ORDER という列に保管したいものと仮定します。この表には、INVOICE_NUM および SALES_PERSON という列もあります。小規模な販売注文であるため、XMLVARCHAR タイプを使用して販売注文を保管することにします。基本キーは INVOICE_NUM です。次の CREATE TABLE ステートメントによって、XML タイプの列を 1 つ含む表を作成します。

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

表を作成した後で、次のステップとして、この列に XML データを入れることができるようにします。

関連概念:

- 63 ページの『サイド表の計画』
- 227 ページの『第 11 章 XML エクステンダーの管理サポート表』

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 という DB2 UDB 表) に保管されます。 DTD_REF 表のスキーマ名は DB2XML です。 DTD_REF 表内のそれぞれの DTD には、固有の ID があります。あるデータベースを XML に関して使用可能にすると、XML エクステンダーは DTD_REF 表を作成します。 DTD 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)」ウィンドウで「**DTD のインポート (Import a DTD)**」を選択して、既存の DTD ファイルを現行データベースの 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)」ウィンドウが表示されます。
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 を識別するための ID です。また、ローカル・システム上の DTD の場所を指定するパスです。 DTD ID は、DAD ファイル内で <DTDID> エレメントに指定されている値と同じでなければなりません。
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. **「完了」**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 10 の SQL INSERT ステートメントを発行します。例えば、

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

表 10. DTD リポジトリー表の列定義

列名	データ・タイプ	説明
DTDID	VARCHAR(128)	DTD の ID。
CONTENT	XMLCLOB	DTD の内容。

表 10. DTD リポジトリー表の列定義 (続き)

列名	データ・タイプ	説明
USAGE_COUNT	INTEGER	データベース内において、この DTD を使用して DAD を定義している XML 列および XML コレクションの数。
AUTHOR	VARCHAR(128)	DTD の作成者 (この情報の入力はオプションです)。
CREATOR	VARCHAR(128)	最初にデータ挿入を行ったユーザー ID。
UPDATOR	VARCHAR(128)	最後に更新を行ったユーザー ID。
ROW_ID	ROWID	行の 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、その文書が入る列で XML を使用可能にしなければなりません。列を使用可能にして索引付けができるようにすると、その列を素早く検索することができます。列を使用可能にするには、XML エクステンダー管理ウィザードまたはコマンド行を使用します。列のタイプは XML でなければなりません。

XML エクステンダーは、XML 列を使用可能にする際に次の操作を実行します。

- 以下の目的で DAD ファイルを読み取ります。
  - DTDID が指定された場合には、DTD が DTD_REF 表の中にあるかどうかを検査する。
  - 索引付けのために、サイド表を XML 列内に作成する。
  - XML データを入れる列を用意する。
- オプションで、XML 表およびサイド表のデフォルト・ビュー を作成します。デフォルト・ビューには、アプリケーション表とサイド表が表示されます。

  列名の制限: iSeries の場合、ビューでの列のサイズは 10 文字に制限されています。
- まだルート 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後、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 列を使用可能にする作業は、管理ウィザードを使用するか、または DB2 コマンド行から実行できます。

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)」ウィンドウで**「XML 列を処理する (Work with XML Columns)」**をクリックし、 XML エクステンダー列に関連したタスクを表示します。「タスクの選択 (Select a Task)」ウィンドウが表示されます。

3. 既存の表列を使用可能にするために、**「列を使用可能にする (Enable a Column)」**をクリックしてから、**「次へ」**をクリックします。
4. **「表名 (Table name)」**フィールドから、 XML 列を含む表を選択します。
5. **「列名 (Column name)」**フィールドから、使用可能にする列を選択します。列のタイプは XML で存在しなければなりません。
6. DAD パスおよびファイル名を**「DAD ファイル名 (DAD file name)」**フィールドに入力するか、または**「...」**をクリックして既存の DAD ファイルをブラウズします。例えば、

   dxx_install/dad/getstart.dad
7. (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。

   デフォルトでは、表スペースには XML エクステンダーによって作成されたサイド表が含まれます。表スペースが指定されている場合、サイド表は指定された表スペースの中に作成されます。表スペースが指定されていない場合、サイド表はデフォルト表スペースの中に作成されます。
8. (オプション) 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。

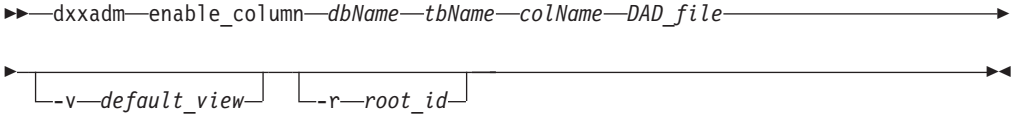
   デフォルトのビューが指定されている場合、デフォルトのビューは、列が使用可能になる際に自動的に作成され、XML 表と関連するすべてのサイド表を結合します。
9. (オプション) 表内の基本キーの列名を**「ルート ID (Root ID)」**フィールドに入力します。これを入力することをお勧めします。

   XML エクステンダーは、すべてのサイド表をアプリケーション表に関連付けるために、固有 ID として**ルート ID** の値を使用します。 XML エクステンダーはアプリケーション表とサイド表に DXXROOT_ID 列を追加します。
10. **「完了」**をクリックして、 XML 列を使用可能にし、サイド表を作成し、「ランチパッド (LaunchPad)」ウィンドウに戻ります。
    - 列が正常に使用可能化されると、`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、エラー・メッセージが表示されます。 DAD を訂正してから、使用可能化処理を再び開始することが必要です。

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 を使用してください。このコマンドの構文およびパラメーターについては、このセクションで説明します。

構文:



パラメーター:

dbName
RDB 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 エクステンダーが作成した、デフォルト・ビューの名前。

root_id オプションですが、指定することをお勧めします。アプリケーション表内の基本キーの列名、およびすべてのサイド表をアプリケーション表に関連付ける固有 ID。 ROOT_ID と呼ばれます。 XML エクステンダーはすべてのサイド表をアプリケーション表に関連付けるために、固有 ID として ROOT_ID (ルート 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 エクステンダーは DXXROOT_ID 列をアプリケーション表に追加して ID を生成します。

制限: アプリケーション表の列名の 1 つが DXXROOT_ID の場合は、ユーザーが *root_id* パラメーターを指定しなければなりません。これを行わない場合、エラーが発生します。

例:

Qshell から、

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

OS のコマンド行から、

```
CALL QDBXM/QZXMADM PARM(enable_column SALES_DB 'MYSCHEMA.SALES_TAB'
   ORDER 'getstart.dad' '-v' sales_order_view '-r' INVOICE_NUMBER)
```

iSeries ナビゲーターから、

```
CALL MYSCHEMA.QZXMADM('enable_column', 'SALES_DB', 'MYSCHEMA.SALES_TAB',
   'ORDER', 'getstart.dad', '-v sales_order_view', '-r INVOICE_NUMBER');
```

この例では、ORDER 列が SALES_TAB 表内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 の列は次のようになります。

列名	データ・タイプ
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	データ・タイプ	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	データ・タイプ	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	データ・タイプ	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基本キー INVOICE_NUM によってルート ID が指定されているためです。列が使用可能になると、行がメイン表に挿入される際に INVOICE_NUM の値がサイド表に挿入されます。 XML 列 ORDER を使用可能にする際に *default_view* パラメーターを指定したため、デフォルト・ビュー sales_order_view が作成されます。このビューは、次のステートメントを使用して上記の表をすべて結合します。

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

サイド表の計画

表は、頻繁に検索される XML 文書の内容を取り出すために使用する DB2® の副表です。 XML 列は、XML 文書の内容が入るサイド表に関連付けられます。 アプリケーションの表内で XML 文書が更新されると、サイド表内の値が自動的に更新されます。

64 ページの図 8 は、サイド表のある XML 列を示しています。

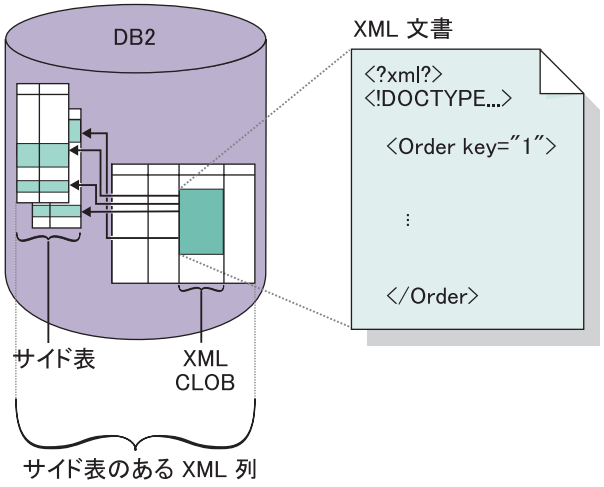


図 8. 内容がサイド表にマップされる XML 列。この列には、XML 文書の内容が入るサイド表に関連付けられた XML ファイルがあります。

サイド表の計画を立てる際、この表の編成方法、いくつの表を作成するか、およびサイド表のデフォルトのビューを作成するかどうかを考慮しなければなりません。これらの決定は、エレメントおよび属性が複数回出現する可能性があるのか、また、どの程度の照会パフォーマンスを必要とするのかに基づいて行ってください。いかなる方法であれ、サイド表は更新しないでください。サイド表は、XML 列の文書が更新されると自動的に更新されます。

複数出現:

エレメントおよび属性がサイド表で複数回出現する場合は、計画にあたって以下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出現するものについては、XML 文書の複雑な構造のため、複数ある 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。つまり、ロケーション・パスが複数回出現するエレメントおよび属性は、列が 1 つしかない表にマップしなければなりません。他の列を表内に持つことはできません。
- ロケーション・パスが複数ある文書の場合、XML エクステンダーは、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 という列を追加します。 DXX_SEQNO を使用すると、SQL 照会内に ORDER BY DXX_SEQNO と指定することで、元の XML 文書と同じ順序でエレメントを取り出すことができます。

デフォルトの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 という名前の固有 ID を使用し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。デフォルトのビューを使用すると、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。例えば、アプリケーション表が SALES_TAB で、サイド表が ORDER_TAB、PART_TAB、および SHIP_TAB であるとすると、照会は次のように行います。

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

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

デフォルトの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、デフォルトの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た独自のビューを作成することができます。

列名の制限: iSeries の場合、ビューでの列のサイズは 10 文字に制限されています。より長い名前を使用するには、手作業でビューを作成するか、別名を使用する必要があります。

サイド表の索引付け

この作業は、XML 列の定義および使用可能化という、より大きな作業の一部です。

サイド表には、DAD ファイルの作成時に指定した列内に XML データが含まれます。 XML 列を使用可能にし、サイド表を作成した後、サイド表を索引付けすることができます。これらの表の索引付けを行うと、XML 文書に対する照会のパフォーマンスを向上させることができます。

手順:

DB2 UDB コマンド行からサイド表の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。

DB2 UDB コマンド行から。

次の例では、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。

```
CREATE INDEX KEY_IDX
      ON ORDER_SIDE_TAB(ORDER_KEY)

CREATE INDEX CUSTOMER_IDX
      ON ORDER_SIDE_TAB(CUSTOMER)

CREATE INDEX PRICE_IDX
      ON PART_SIDE_TAB(PRICE)

CREATE INDEX DATE_IDX
      ON SHIP_SIDE_TAB(DATE)
```

SQL マッピングを使用した XML 文書の合成

XML 文書の合成は、コマンド行から SQL マッピングを使用するか、管理ウィザードを使用することによって実行できます。

XML 文書を合成し、 SQL ステートメントを使用して XML 文書内のデータの派生元となる表および列を定義する場合は、 SQL マッピングを使用してください。 SQL マッピングは、XML 文書の合成にのみ使用できます。 SQL マッピングによって XML 文書を合成するには、DAD ファイルを作成します。

前提条件:

文書を合成する前に、DB2 UDB 表と XML 文書との関連をマップしなければなりません。このステップには、XML 文書の階層をマップして、文書内のデータが DB2 UDB 表にマップされる方法を指定することが含まれます。

手順:

コマンド行から XML 文書を合成するには、次のようにします。

1. テキスト・エディターで新規文書を作成し、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る DAD の妥当性検査で使用するタグを挿入します。

   - 合成された XML 文書に関して、DTD を使用して妥当性検査を実行するには、 DAD ファイルと XML 文書 DTD を関連付ける DTDID タグを挿入します。例えば、

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

   - 合成された XML 文書に関して、スキーマを使用して妥当性検査を実行するには、 DAD ファイルとスキーマ・ファイルを関連付けるスキーマ・タグを挿入します。例えば、

     ```
     <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実施して初めて、有効なものとして利用できます。 DB2 UDB XML エクステンダーが XML 文書の妥当性検査を実行するかどうかを指定するには、以下のようにして妥当性検査タグを使用します。

   - XML 文書を妥当性検査する場合には、次のように入力してください。

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

   - XML 文書を妥当性検査しない場合には、次のように入力してください。

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

4. XML データのアクセスおよび保管方式として XML コレクションを使用することを指定するために、<XCollection> </XCollection> タグを入力します。

5. <Xcollection> </Xcollection> タグ内に、 XML 文書へのリレーショナル・データをマップする SQL ステートメントを指定するための <SQL_stmt> </SQL_stmt> タグを挿入します。こ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されます。次の例は、SQL 照会のサンプルです。

   ```
   <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,
      (select db2xml.generate_unique()
       as ship_id, date, mode, part_key from ship_tab) as 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 文書にマップするためのサンプル SQL ステートメントの構文は、次のとおりです。

- 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。
- 1 つのエンティティーに関する列は同じグループにまとめられます。
- 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 表には単一キー列がないため、 ship_id 列の作成には generate_unique DB2 UDB 組み込み関数が使用されます。
- これにより、ORDER BY ステートメント内でオブジェクト ID 列はトップダウンの順序でリストされます。 ORDER BY 内の列は、スキーマや列名で修飾してはならず、 SELECT 文節内の列名に一致しなければなりません。

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、合成された文書の妥当性検査に使用する DTD へのパスが入ります。例えば、

   ```
   <doctype>! DOCTYPE Order SYSTEM "dxxsamples/dtd/getstart.dtd"</doctype>
   ```

8. ルート・エレメントと、XML 文書を構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および属性は、root_node 内で指定されます。

   b. <element_node>、<attribute_node>、および <text_node> の各タグを使用することにより、 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> タグ
      XML 文書内のエレメントを指定します。 element_node タグの name 属性は、このエレメントの名前に設定します。各 element_node に子 element_node が存在しても問題ありません。

      <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、そのエレメント・ノード内でネストされます。 attribute_node タグの name 属性は、この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。最下位の各エレメントごとに、<text_node> タグを定義します。これは、文書合成時に DB2 から取り出される文字データがそのエレメントに含まれることを示すものです。最下位の各 element_node ごとに、 <column> タグを使用することによって、 XML 文書合成時にデータを取り出す元になる列を指定します。列 (column) タグは、一般に、<attribute_node> または <text_node> タグの内側に指定します。定義されるすべての列は、DAD ファイルの最初にある <SQL_stmt> SELECT 文節内になければなりません。

9. 適切な位置に終了タグが必要です。

a. </root_node> 終了タグが最後の </element_node> タグの後にあることを確認します。

b. </Xcollection> 終了タグが </root_node> タグの後にあることを確認します。

c. </DAD> 終了タグが </Xcollection> タグの後にあることを確認します。

10. ファイルを *file.dad* として保管します。ここで *file* は、実際のファイルの名前です。

次の例は、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 "dxxsamples¥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 マッピングを使用した XML コレクションの合成

RDB_node マッピングは、<RDB_node> タグを使用して、エレメントまたは属性ノードの DB2 UDB 表、列、および条件を指定します。 XML に類似した構造を使用して XML 文書を合成する場合には、この方法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。

table エレメントに対応する表を定義します。

column 対応するエレメントを含む列を定義します。

condition オプションで、列に対する条件を指定します。

RDB_node エレメント内に使用さ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かどうか		
	表	列	条件 [1]
ルート・エレメント	○	×	○
属性	○	○	オプション
テキスト	○	○	オプション

[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の XML 文書の合成には、管理ウィザードを使用することも、またコマンド行を使用することもできます。

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には、あるエレメントのすべてのステートメントを同じ表の列にマップしなければなりません。

手順:

RDB_node マッピングを使用してコマンド行から XML 文書を合成するには、

1. テキスト・エディターをオープンし、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 DTD のパスおよびファイル名です。
2. <DAD></DAD> タグを挿入します。このエレメントに、他のすべてのエレメントが入れられることになります。
3. DTD またはスキーマによる DAD の妥当性検査で使用するタグを挿入します。
   - DTD を使用して DAD の妥当性検査を実行するには、 DAD ファイルと XML 文書 DTD を関連付ける DTDID タグを挿入します。例えば、

     ```
     <dtdid>path/dtd_name.dtdid>
     ```
   - スキーマを使用して DAD の妥当性検査を実行するには、その 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。例えば、

```
<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実施して初めて、有効なものとして利用できます。 DB2 UDB XML エクステンダーが XML 文書の妥当性検査を実行するかどうかを指定するには、以下のようにして妥当性検査タグを使用します。

- XML 文書を妥当性検査する場合には、次のように入力してください。

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

- XML 文書を妥当性検査しない場合には、次のように入力してください。

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

4. XML データのアクセスおよび保管方式として XML コレクションを使用することを指定するために、 <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。例えば、

   ```
   <doctype>! DOCTYPE Order SYSTEM "dxxsamples/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> タグの内側に、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 には、RDB_node エレメントを使用してください。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 文書内のエレメントおよび属性をマップするには、

   a. 先頭 element_node に対する RDB_node を指定します。このエレメントは、XML 文書に関連付けられているすべての表を指定します。先頭 element_node の RDB_node を指定するには、 root_node タグの後に <RDB_node> タグを挿入します。

      - attribute_node の RDB_node を指定します。
      - text_node の RDB_node を指定します。

   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します。例えば、文書に入れる列データを含む 3 つの表 (ORDER_TAB、PART_TAB、および SHIP_TAB) がある場合には、各表ごとに表ノードを作成します。例えば、

      ```
      <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、表ごとに基本キーを指定しなければなりません。基本キーは、単一の列または複数の列 (複合キーと呼ばれる) から構成されます。基本キーは、RDB_node の表エレメントに属性キーを追加することによって指定されます。コレクションを使用可能にする場合には、それぞれの表ごとに基本キーの指定も行わなければなりません。以下の例は、element_node で指定されたそれぞれの表ごとに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』
- 175 ページの『XML コレクションのための DAD ファイル』
- 114 ページの『RDB_node マッピングの要件』

関連タスク:

- 71 ページの『RDB_node マッピングを使用した XML 文書の分解』
- 96 ページの『XML コレクション内のデータの管理』
- 106 ページの『XML コレクションのデータの更新、削除、および取り出し』

関連参照:

- 209 ページの『XML エクステンダーの合成ストアード・プロシージャー』

RDB_node マッピングを使用した XML 文書の分解

XML 文書を分解するには、RDB_node マッピングを使用します。この方法では、<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。

table エレメントに対応する表を定義します。

column 対応するエレメントを含む列を定義します。

condition オプションで、列に対する条件を指定します。

<RDB_node> 内に使用さ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 子エレメントの使用:		
	表	列	条件 [1]
ルート・エレメント	○	×	○
属性	○	○	オプション
テキスト	○	○	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場合の手順:

コマンド行を使用して XML 文書を分解するには、

1. テキスト・エディターでファイルを作成します。次の構文を入力することにより、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 DTD のパスおよびファイル名です。

DAD ファイルは、統合ファイル・システム (IFS) ディレクトリーに保管するか、物理ファイル・メンバーとして、その物理ファイル・メンバーを指すリンクと共に IFS ディレクトリー内に作成しなければなりません。

2. <DAD></DAD> タグを挿入します。
3. DTD またはスキーマによる DAD の妥当性検査で使用するタグを挿入します。
   - DTD を使用して DAD の妥当性検査を実行するには、 DAD ファイルと XML 文書 DTD を関連付ける DTDID タグを挿入します。例えば、

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

   - スキーマを使用して DAD の妥当性検査を実行するには、その 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。例えば、

     ```
     <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実施して初めて、有効なものとして利用できます。 DB2 UDB XML エクステンダーが XML 文書の妥当性検査を実行するかどうかを指定するには、以下のようにして妥当性検査タグを使用します。

   - XML 文書を妥当性検査する場合には、次のように入力してください。

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

   - XML 文書を妥当性検査しない場合には、次のように入力してください。

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

4. XML データのアクセスおよび保管方式として XML コレクションを使用することを指定するために、 <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。例えば、

   ```
   <doctype>! DOCTYPE Order SYSTEM "dxxsample/dtd/getstart.dtd"</doctype>
   ```

   国際化対応用のエンコード値を指定する必要がある場合には、 ENCODING 属性および値を追加してください。

7. <root_node></root_node> タグを使用して root_node を定義します。
8. root_node タグの内側に、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) RDB_node エレメントを作成します。例えば、

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

2) XML 文書内に組み込まれるデータを含むそれぞれの表ごとに table_node (表ノード) を定義します。例えば、文書に入れる列データを含む 3 つの表 (ORDER_TAB、PART_TAB、および SHIP_TAB) がある場合には、各表ごとに表ノードを作成します。例えば、

```
<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、次のとおりです。

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

例えば、

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

4) 各表ごとに基本キーを指定します。基本キーは、単一の列または複合キーという複数の列から構成されます。基本キーを指定するには、属性キーを RDB_node の表エレメントに追加します。以下の例では、 Order というルート element_node の RDB_node 内のそれぞれの表ごとに基本キーを定義します。

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

分解機能を使用してコレクションを有効にする場合、使用される DAD ファイルは合成と分解の両方をサポートする必要があるため、キー属性が必要です。

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。例えば、

```
<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 エレメントの内容が DB2 UDB 表にあり、子エレメントがないことを指定する場合。

attribute_node
属性を指定する場合。

child elements

element_node の子。

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。

table エレメントに対応する表を定義します。

column 対応するエレメントを含む列を定義します。

condition オプションで、列に対する条件を指定します。

例えば、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す。

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。例えば、

```
<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。 RDB_node には次のような子エレメントがあります。

table エレメントに対応する表を定義します。

column 対応するエレメントを含む列を定義します。

condition オプションで、列に対する条件を指定します。

例えば、Order エレメントに属性キーを指定できます。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. 適切な位置に終了タグが必要です。
    a. </root_node> 終了タグが最後の </element_node> タグの後にあることを確認します。
    b. </Xcollection> 終了タグが </root_node> タグの後にあることを確認します。
    c. </DAD> 終了タグが </Xcollection> タグの後にあることを確認します。

関連タスク:

- 101 ページの『XML 文書を分解して DB2 UDB データにする』
- 210 ページの『XML エクステンダーの合成ストアード・プロシージャーの呼び出し』

関連参照:

- 222 ページの『XML エクステンダーの分解ストアード・プロシージャー』

第 3 部 プログラミング

ここでは、XML データを管理するためのプログラミング技法を示します。

第 3 章 XML 列

この章では、DB2 を使用して XML 列内のデータを管理する方法について説明します。

XML 列内のデータの管理

XML 列を使用してデータを保管する場合には、 XML 文書全体をその固有の形式で DB2 の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 対応にすると、 XML エクステンダーによって提供される以下のユーザー定義タイプ (UDT) が使用できるようになります。

XMLCLOB
DB2 に文字ラージ・オブジェクト (CLOB) として保管される XML 文書内容には、この UDT を使用してください。

XMLVARCHAR
DB2 に VARCHAR として保管される XML 文書内容には、この UDT を使用してください。

XMLFile
ローカル・ファイル・システムにファイルとして保管される XML 文書には、この UDT を使用してください。

アプリケーション表を作成または変更して、XML UDT データ・タイプの列を含めることができます。これらの表を、XML 表と呼びます。

表の中の列を XML 対応にすると、 XML 列を作成して以下の管理タスクを実行できるようになります。

- XML 文書を DB2 内に保管する
- XML データまたは文書を DB2 から取り出す
- XML 文書を更新する
- XML データまたは文書を削除する

これらのすべてのタスクを実行するには、 XML エクステンダーの提供するユーザー定義関数 (UDF) を使用してください。 XML 文書を DB2 に保管するには、デフォルト・キャスト関数を使用してください。デフォルト・キャスト関数は、 SQL 基本タイプを XML エクステンダーのユーザー定義タイプにキャストし、データ・タイプのインスタンス (オリジン) を異なるデータ・タイプのインスタンス (ターゲット) 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ります。 XML に、一連の文書を作成するために必要な情報がすべて含まれるという場合です。

例えば、Web に記事を供給するニュース発行会社であれば、発行済みの記事のアーカイブを保守したいと思うことがあります。そのようなシナリオでは、図 9 に示すように、 XML エクステンダーを使用して XML 記事の全部または一部を DB2® 表の列 (*XML 列*) に保管することができます。

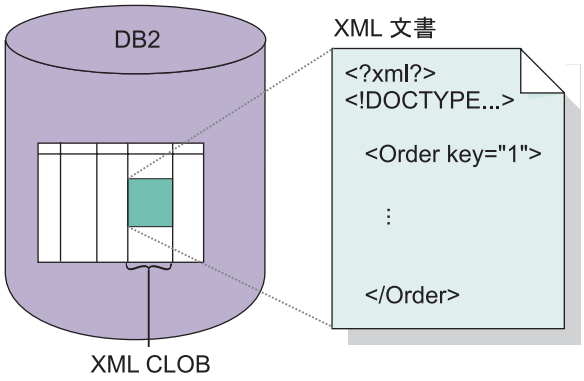


図 9. 構造化 XML 文書を DB2 UDB 表の列に保管する

XML 列による保管およびアクセス方式を使用することにより、DB2 を使用して XML 文書を管理できます。 XML 文書を XML タイプの列に保管し、その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したりすることができます。DB2 UDB 内に、1 つ以上の文書に対して DTD を関連づけ、保管できます。さらに、エレメントと属性内容をサイド表 と呼ばれる DB2 UDB 表にマップできます。これらのサイド表に索引を付けて照会のパフォーマンスを向上させることができますが、索引付けは自動的には行われません。文書を保管するために使用する列は XML 列と呼ばれます。その列が XML 列のアクセスと保管のために使用されることを示します。

保管およびアクセス方式として XML 列を使用することを示すために、文書アクセス定義 (DAD) ファイルに <Xcolumn> タグと </Xcolumn> タグを入れます。その後で、DAD が、保管する必要のある XML エレメントおよび属性内容をサイド表にマップします。

文書を保管するために XML エクステンダーで作業を始める前に、文書内のエレメントと属性にどのように索引を付けるかを決めるために、 XML 文書の構造を理解する必要があります。文書に索引を付けるには、以下のことを判断する必要があります。

- XML 文書を保管するのに使用する 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。これは、パフォーマンスを向上させるために、それらの内容をサイド表に保管し、索引を付けられるようにするためです。
- DTD により、列内の XML 文書の妥当性検査をするかどうか。

XML 列の定義および使用可能化

XML 列を使用して、データベース内で XML 文書全体を保管したり、アクセスしたりすることができます。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し、サイド表に列の索引を作成したうえで、XML 文書を照会または検索することができます。

XML 文書を頻繁に更新する予定がない場合、または XML 文書をそのまま保管したい場合は、 DB2 表の列に XML 文書全体を保管する際に XML 列を使用してください。

XML 文書構造を DB2 UDB 表にマップして、既存の DB2 UDB データから XML 文書を合成したり XML 文書を DB2 データに分解したりできるようにしたい場合には、 XML 列ではなく XML コレクションを使用する必要があります。

手順:

コマンド行から XML 列を定義し、使用可能にするには、次の手順を実行してください。

1. 文書アクセス定義 (DAD) ファイルを作成します。
2. XML 文書を保管するための表を作成します。
3. XML データ用の列を使用可能にする。 DAD で妥当性検査を指定した場合には、dtd_ref 表に列を挿入してください。
4. サイド表を索引付けします。

XML 列は XML ユーザー・データ・タイプとして作成されます。これらのタスクが終了すると、XML 文書をその列に保管できるようになります。これらの文書は、その後で更新、検索、および抽出することができます。

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』
- 56 ページの『XML 文書の自動妥当性検査』
- 9 ページの『学習: XML 列への XML 文書の保管』

関連タスク:

- 173 ページの『XML 列のための DAD ファイルの作成』
- 58 ページの『XML 表の作成』
- 60 ページの『XML 列の使用可能化』
- 65 ページの『サイド表の索引付け』
- 79 ページの『XML 列内のデータの管理』

XML 列データの索引の使用

XML 列を使用するためには、 XML 列文書のサイド表に索引を付けるかどうかを計画時に決定する必要があります。この決定は、データへのアクセスが必要になる頻度、および構造検索の際のパフォーマンスの重要性に基づいて行います。

XML 文書の全体を含む XML 列を使用する際、 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。索引を作成する対象となるエレメントおよび属性を決定する必要があります。

XML 列を索引付けすることによって、頻繁に照会される整数、10 進数、または日付などの汎用データ・タイプのデータを、データベース・エンジンのネイティブ DB2® 索引サポートを使用して索引付けすることができます。 XML エクステンダーは XML エレメントの値または XML 文書から属性を抽出して、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 エクステンダーは、XML 文書を XML 列に保管するときに自動的にサイド表にデータを取り入れます。

高速検索のために、DB2 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。

索引を作成する際には、以下の点を考慮する必要があります。

- XML 文書内のエレメントまたは属性で*複数出現* するものについては、 XML 文書の複合構造のために、複数出現する 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基本キーの列名、そしてすべてのサイド表をアプリケーション表に関連付ける固有 ID を使用します。アプリケーション表の基本キーを ROOT ID にするかどうかを決めることができます。ただし、それを複合キーにすることはできません。この方法をお勧めします。

  アプリケーション表に単一の基本キーが存在しない場合、または何かの理由でそれを使用したくない場合、XML エクステンダーはアプリケーション表を変更して、挿入時に作成される固有の ID を保管する列 DXXROOT_ID を追加します。すべてのサイド表には、固有の ID を持つ DXXROOT_ID 列があります。基本キーを ROOT ID として使用すると、アプリケーション表内の基本キー列と同じ名前とタイプの列が、すべてのサイド表に入り、基本キーの値が保管されます。
- XML 列を DB2 UDB Text Extender について使用可能にする場合、 Text Extender の構造化テキスト機能も使用できます。 Text 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 エクステンダーの索引付けと共に使用できます。

XML データの保管

XML エクステンダーを使用して、原形の XML 文書を XML 列に挿入することができます。サイド表を定義している場合、XML エクステンダーは自動的にこのサイド表を更新します。 XML 文書を直接保管する場合、XML エクステンダーは基本タイプを XML タイプとして保管します。

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。
- データを DB2® 表に保管するための方式 (キャスト関数または UDF) を選択します。

XML 表および列に XML 文書を含むように指定する SQL INSERT ステートメントを指定します。

XML エクステンダーには、XML 文書を保管するための、デフォルト・キャスト関数と保管 UDF の 2 つの方式が備わっています。

表 11 は、それぞれの方式をいつ使用するかを示しています。

表 11. XML エクステンダーの保管関数

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 File, 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 エクステンダーが提供するキャスト関数を 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 12 は、提供されているキャスト関数を示しています。

表 12. XML エクステンダーのデフォルト・キャスト関数

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
XMLFILE(VARCHAR)	XMLFILE	ファイル名のみを保管する

例えば、次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 エクステンダー UDT ごとに、基本タイプ以外のリソースから DB2 にデータをインポートするために保管 UDF が存在します。例えば、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 13 は、XML エクステンダーが提供している保管関数を示しています。

表 13. XML エクステンダーの保管 UDF

保管ユーザー定義関数	戻りタイプ	説明
XMLVarcharFromFile()	XMLVARCHAR	XML 文書をサーバー上のファイルから読み取り、XMLVARCHAR 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を指定します。

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

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

この例では、ファイル dxxsamples/xml/getstart.xml から SALES_TAB 表の ORDER 列に XML 文書がインポートされます。

XML 文書を取り出す方法

XML エクステンダーを使用して、文書全体、またはエレメントおよび属性の内容を取り出すことができます。 XML 列を直接取り出すと、XML エクステンダーは UDT を列タイプとして戻します。データの取り出しの詳細については、以下のセクションを参照してください。

- 『XML 文書全体の取り出し』
- 87 ページの『XML 文書からのエレメントの内容および属性値の取り出し』

XML エクステンダーには、データを取り出すための 2 つの方式が備わっています。それらは、デフォルト・キャスト関数と Content() 多重定義 UDF です。 表 14 は、それぞれの方式をいつ使用するかを示しています。

表 14. XML エクステンダーの取り出し関数

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 文書全体の取り出し

手順:

XML 文書全体を取り出すには、次の手順を実行してください。

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、取り出されるデータのデータ・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、XMLCLOB を LOCATOR として、または XMLFILE データ・タイプとして定義します。

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 が提供するデフォルト・キャスト関数は、UDT が XML UDT を SQL 基本タイプに変換して、その後それを処理します。 SELECT ステートメント内で、XML エクステンダーが提供するキャスト関数を使用して、データを取り出すことができます。 表 15 は、提供されているキャスト関数を示しています。

表 15. XML エクステンダーのデフォルト・キャスト関数

SELECT 文節内で使用されるキャスト	戻りタイプ	説明
varchar(XMLVARCHAR)	VARCHAR	VARCHAR による XML 文書

表 15. XML エクステンダーのデフォルト・キャスト関数 (続き)

SELECT 文節内で使用されるキャスト	戻りタイプ	説明
clob(XMLCLOB)	CLOB	CLOB による XML 文書
varchar(XMLFile)	VARCHAR	VARCHAR による XML ファイル名

例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

外部記憶装置からメモリーへの文書内容の取り出し、および内部ストレージから外部ファイル (DB2 UDB サーバー上にある DB2 UDB の外側のファイル) への文書のエクスポートには、 Content() UDF を使用します。

例えば、XML 文書を XMLFILE データ・タイプとして保管することができます。それをメモリー内で処理したい場合、Content() UDF を使用することができます。Content() UDF は XMLFILE データ・タイプを入力として受け入れ、CLOB を戻します。

Content() UDF は、指定されたデータ・タイプに応じて次の 2 つの異なる取り出し機能を実行します。

- 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 文書を取り出し、それをメモリー・バッファーまたは CLOB ロケーター (データベース・サーバー内の単一の LOB 値を表す値を持つホスト変数) に入れることができます。

  次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)
  ```

- 内部ストレージ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、 XMLCLOB データ・タイプとして DB2 UDB 内に保管されている XML 文書を取り出し、データベース・サーバーのファイル・システム上のファイルにエクスポートすることができます。 Content() UDF は、ファイルの名前を VARCHAR データ・タイプで戻します。

  次の関数構文を使用してください。

  XML タイプから外部ファイルへ:

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

  ここで、

  xmlobj XML の内容が取り出される XML 列の名前。*xmlobj* のタイプは XMLVARCHAR または XMLCLOB です。

filename
XML データを保管する外部ファイルの名前。

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

下記の例では、組み込み SQL ステートメント (アプリケーション・プログラム内にコーディングされた SQL ステートメント) のある小さな C プログラムの一部によって、どのようにして XML 文書がファイルから取り出されてメモリーに入れられるかを示しています。この例では、ORDER 列のデータ・タイプが XMLFILE であると想定しています。

```
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の内容および属性値の取り出し

1 つ以上の XML 文書から、エレメントの内容または属性の値を取り出す (抽出する) ことができます (単一文書または集合文書検索)。 XML エクステンダーは、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を提供しています。

XML データにはリレーショナル・データとしてアクセスできるため、エレメントの内容および属性の値の取り出しは、アプリケーションの開発に役立ちます。例えば、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顧客すべての名前を取り出すには、SELECT 文節に抽出 UDF を指定して、次の SQL ステートメントを使用してください。

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

この例では、抽出 UDF は <customer> エレメントの内容を ORDER 列から取り出し、VARCHAR データ・タイプとして保管します。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> エレメントの内容のみを指定する WHERE 文節を使用することにより、戻り値の数が減少します。

88 ページの表 16 は、スカラー関数として次の構文を使用して、エレメントの内容と属性の値を抽出するために使用できる UDF を示しています。

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。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 as LOCATOR
- 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)。

制約事項: UDF の抽出は、述部で属性が指定されているロケーション・パスはサポートできますが、述部でエレメントが指定されているロケーション・パスはサポートできません。例えば、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 16 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 16. XML エクステンダーの抽出関数

スカラー関数	戻される列名 (表関数)	戻りタイプ
extractInteger()	returnedInteger	INTEGER
extractSmallint()	returnedSmallint	SMALLINT
extractDouble()	returnedDouble	DOUBLE
extractReal()	returnedReal	REAL
extractChar()	returnedChar	CHAR
extractVarchar()	returnedVarchar	VARCHAR
extractCLOB()	returnedCLOB	CLOB
extractDate()	returnedDate	DATE
extractTime()	returnedTime	TIME
extractTimestamp()	returnedTimestamp	TIMESTAMP

スカラー関数の例: 次の例では、1 つの値が属性キー値 1 で挿入されます。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
   (SELECT db2xml.extractInteger(db2xml.xmlfile('c:¥dxx¥samples¥xml¥getstart.xml'),
 '/Order[@key="1"]/@key')
    FROM db2xml.onerow) ;
SELECT * from t1;
```

表関数の例: 次の例では、販売注文のそれぞれのキー値 (@key) が INTEGER として抽出されます。

```
SELECT * from table(DB2XML.extractIntegers(DB2XML.XMLFile
   ('/dxxsamples/xml/getstart.xml'), '/Order/@key')) as x;
```

XML データの更新

XML エクステンダーを使用して、XML 列データを置換して XML 文書全体を更新したり、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、以下の手順を実行します。

1. XML 文書は、XML 表の中に保管されていなければなりません。
2. どのデータを取り出すかがわかっていなければなりません。
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

XML 文書の更新は、デフォルト・キャスト関数を使用するか、または保管 UDF を使用して実行できます。

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 エクステンダーが提供す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。

例えば、xml_buf が VARCHAR タイプとして定義されたホスト変数である場合、次のステートメントにより、キャスト VARCHAR タイプから XMLVARCHAR タイプが更新されます。

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

保管 UDF による XML 文書の更新

XML エクステンダー UDT ごとに、基本タイプ以外のリソースから DB2 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、 dxxsample/xml/getstart.xml という名前のファイルから SALES_TAB 表の ORDER 列に XML オブジェクトが更新されます。

```
UPDATE sales_tab
       set order = XMLVarcharFromFile('dxxsample
       /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

Update UDF は、文書全体の変更ではなく、特定の変更を行うために使用します。この UDF を使用する際には、値が置き換えられるエレメントまたは属性のロケーション・パスを指定してください。 XML 文書を編集する必要はありません。XML エクステンダーがユーザーに代わって変更を行います。

構文:

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

この構文には以下の構成要素が含まれます。

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 更新される新しい値です。

例えば、次のステートメントを使用すると <Customer> エレメントの値が IBM で置き換えられます。

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

複数出現: Update UDF にロケーション・パスを指定した場合、パスが一致するすべてのエレメントまたは属性の内容は、指定した値で更新されます。あるロケーション・パスが文書内で複数回出現する場合、Update UDF は、既存のすべての値を *value* パラメーターで指定された値で置き換えます。

XML 文書の検索の方法

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操作できるようにしますが、検索では WHERE 文節の内容が検索基準として使用されます。

XML エクステンダーには、XML 列に保管される XML 文書を検索するための数種類の方式が用意されています。それらの方式を使用して、以下のことを実行できます。

- エレメントの内容または属性値に基づいて文書構造を検索し、結果を戻す。
- XML 列およびそのサイド表のビューを検索する。
- パフォーマンスを向上させるためにサイド表を直接検索する。
- WHERE 文節を含む抽出 UDF を使用して検索する。
- DB2 ® Text 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 エクステンダーでは、索引を使用してサイド表内の列を素早く検索することができます。これらの列には、XML 文書から抽出された XML エレメントの内容または属性値が含まれます。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、範囲検索を実行したりすることができます。例えば、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、Text 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。例えば、XML 形式の履歴書を含む、RESUME という列があるとします。 Java™ のスキルを保有している志望者全員の名前を検索したい場合には、DB2 UDB Text Extender を使用して XML 文書を検索し、 <skill> エレメントに「JAVA」という文字ストリングが含まれるすべての履歴書を見つけることができます。

以下のセクションでは、次の検索方式について解説します。

- 『構造に基づいた XML 文書の検索』

構造に基づいた XML 文書の検索

XML エクステンダーの検索機能を使用して、文書構造 (文書内のエレメントおよび属性) に基づいて列内の XML データを検索することができます。

手順:

データを検索するために、次のことを行うことができます。

- サイド表を直接照会する。
- *結合ビュー* を使用する。
- 抽出 UDF を使用する。

以下の例では、次のシナリオに基づく例を使用して、これらの検索方式について説明します。 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。 ORDER 列が使用可能にされた際に、デフォルトのビュー sales_order_view が指定されています。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ます。

```
CREATE VIEW sales_order_view(invoice_num, sales_person, order,
                             order_key, customer, part_key, price, date)
     AS
     SELECT sales_tab.invoice_num, sales_tab.sales_person, sales_tab.order,
            order_side_tab.order_key, order_side_tab.customer,
            part_side_tab.part_key, ship_side_tab.date
     FROM sales_tab, order_side_tab, part_side_tab, ship_side_tab
     WHERE sales_tab.invoice_num = order_side_tab.invoice_num
                AND sales_tab.invoice_num = part_side_tab.invoice_num
                AND sales_tab.invoice_num = ship_side_tab.invoice_num
```

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。

例えば次のステートメントは照会または副照会を使用して、サイド表を直接検索します。

```
SELECT sales_person from sales_tab
   WHERE invoice_num in
     (SELECT invoice_num from part_side_tab
      WHERE price > 2500.00)
```

この例で、invoice_num は SALES_TAB 表の基本キーです。

例: 結合ビューからの検索

XML エクステンダーでは、固有の ID を使用してアプリケーション表とサイド表とを結合する、デフォルトのビューを作成することができます。このデフォルトのビューか、アプリケーション表とサイド表とを結合する任意のビューを使用して、列データの検索とサイド表の照会を行うことができます。この方式は、アプリケーション表とそのサイド表のための、単一の仮想ビューを提供します。しかし、作成されるサイド表の数が多くなると、それにつれて照会の実行時間が長くなります。

ヒント: ルート ID または DXXROOT_ID (XML エクステンダーによって作成される) を使用して、独自のビューを作成する際に複数の表を結合することができます。

例えば、次のステートメントは、SALES_ORDER_VIEW というビューを検索し、注文の行項目の価格が 2500.00 を超えている SALES_PERSON 列から値を戻します。

```
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場合、 XML エクステンダーの抽出 UDF を使用してエレメントおよび属性を検索することもできます。抽出 UDF を使用して XML データを走査することは負荷が大きいので、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。

次のステートメントは、抽出 XML エクステンダー UDF を使用して検索を実行します。

```
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 IBM というサブストリングを含む </Order/Customer/Name> エレメントを抽出します。

例: 複数出現するエレメントまたは属性の検索

複数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。

次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 が複数出現することが指定され、そのためのサイド表 PART_SIDE_TAB が作成されます。 PART_SIDE_TAB

表には、同一の invoice_num を持つ行が複数存在する可能性があります。DISTINCT を使用すると、固有の値のみが戻されます。

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。 WHERE 文節を指定して、特定の文書を削除することができます。

例えば次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)
```

サイド表内の対応する行が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 列内のデータの管理』

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ため、参照を解決することができません。

例えば、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 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 コレクション内のデータの管理

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、または送信される 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ている XML 文書の内容からタグを取ったものです。または、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用する場合に、 DB2 のリレーショナル機能の利点を生かすには、やりとりする XML 文書を必要に応じて合成および分解してデータを管理できるようにします。このようなタイプの XML 文書ストレージを、*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。

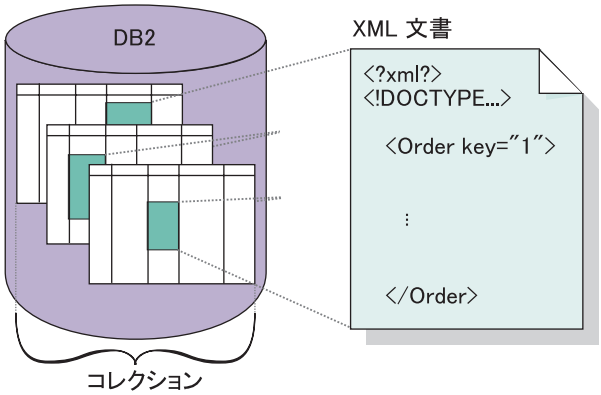


図 10. タグの付いていないデータとしての文書の DB2 UDB 表内での保管

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。このコレクションは、特定の XML 文書内のデータ、または一組の XML 文書が入っている DAD ファイルに関連した、列セットです。コレクション名を使用可能にすることでそれを定義して、次に XML 文書を合成または分解するためのストアード・プロシージャーを発行する際に、そのコレクションを名前で参照することができます。これは、使用可能 XML コレクションと呼ばれます。このコレクションには、 XML 文書を合成および分解するストアード・プロシージャーで容易に実行できるように、名前が付けられます。

DAD ファイル内にコレクションを定義する場合、 *SQL マッピング* または *RDB_node マッピング* の 2 つのマッピング体系のうちのいずれかを使用します。これらのマッピングは、XML データを DB2 UDB 表に関連付けるために使用される、表、列、および条件を定義します。 SQL マッピングでは、SQL SELECT ステートメントを使用して、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 マッピングは、XPath ベースのリレーショナル・データベース・ノード、つまり RDB_node (子エレメントを持つ) を使用します。

XML 文書の合成または分解用のストアード・プロシージャーが用意されています。ストアード・プロシージャー名は、 XML エクステンダーのスキーマ名 である DB2XML によって修飾されます。

XML コレクション内のデータの管理

XML コレクションは、XML 文書にマップされるデータを含むリレーショナル表のセットです。アクセスおよび保管のためのこの方式によって、既存のデータから XML 文書を合成したり、 XML 文書を分解したり、XML を交換の方式として使用することができます。

コレクションを構成するリレーショナル表としては、新しい表か、またはユーザーのアプリケーションの XML 文書を構成するために XML エクステンダーとともに使用されるデータを含む既存の表を使用できます。これらの表にある列データには、XML タグが含まれません。 XML タグ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。 XML コレクションのデータの格納や取り出し、更新、検索、および削除を実行するには、ストアード・プロシージャーを使用します。

ストアード・プロシージャーの結果の CLOB のサイズは、増やすことができます。

XML 文書を DB2 UDB データから合成するための準備

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。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、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、

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 (SQL マッピングまたは RDB_node マッピング) を選択します。
3. DAD ファイルを準備します。

XML エクステンダーには XML 文書を合成するための 4 つのストアード・プロシージャー、 dxxGenXML()、dxxGenXMLCLOB()、dxxRetrieveXML() および dxxRetrieveXMLCLOB が含まれています。ユーザーが計画する、XML 文書の更新の頻度は、使用するストアード・プロシージャーを選択する上で非常に重要な要素です。

更新頻度の低い XML 文書の合成

更新頻度の低い文書は、dxxGenXML ストアード・プロシージャーを使用して合成してください。このストアード・プロシージャーを使用するためにコレクションを

使用可能にする必要はありません。このストアード・プロシージャーでは、コレクションではなく DAD ファイルが使用されます。

dxxGenXML ストアード・プロシージャーは、 DAD ファイル内の <Xcollection> エレメントによって指定された XML コレクション表に保管されているデータを使用して、 XML 文書を生成します。このストアード・プロシージャーは、各 XML 文書を結果表内に行として挿入します。また、結果表でカーソルを開いて結果セットを取り出すこともできます。結果表はアプリケーションで作成する必要があり、VARCHAR、CLOB、XMLVARCHAR、または XMLCLOB タイプの列が 1 つ必要です。

また、DAD ファイル内の妥当性検査エレメントの値が YES になっており、INTEGER タイプの列 DXX_VALID がまだ結果表に含まれていない場合、 XML エクステンダーは DXX_VALID 列を結果表に追加します。 XML エクステンダーは、XML 文書が有効である場合には、値 1 を挿入し、文書が無効である場合には、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。このストアード・プロシージャーは、表にある実際の行数、および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。これはまた、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必要はありません。

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には、次のストアード・プロシージャー宣言を使用して、ストアード・プロシージャー呼び出しをアプリケーションに組み込みます。

```
dxxGenXML(CLOB(100K)     DAD,               /* input */
          char(32 ) resultTabName,     /* input */
          char(30)       result_column, /* 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 */
```

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 */

char  result_tab[32];     /* name of the result table */
char  result_colname[32]; /* name of the result column */
char  valid_colname[32];  /* name of the valid column, will set to NULL */
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 rcol_ind;
short vcol_ind;
short ovtype_ind;
short ov_ind;
short maxrow_ind;
short numrow_ind;
short returnCode_ind;
short returnMsg_ind;
EXEC SQL END DECLARE SECTION;

FILE  *file_handle;
long  file_length=0;

/* create table */
EXEC SQL CREATE TABLE xml_order_tab (xmlorder XMLVarchar);

/* initialize the DAD CLOB object. */
file_handle = fopen("/dxx/dad
    /getstart_xcollection.dad", "r");
if ( file_handle != NULL ) {
  file_length = fread ((void *) &dad.data,
                       1, FILE_SIZE, file_handle);
  if (file_length == 0) {
    printf ("Error reading dad file
             /dxx/dad
              /getstart_xcollection.dad¥n");
    rc = -1;
    goto exit;
 } else
    dad.length = file_length;
}
else {
  printf("Error opening dad file  ¥n", );
  rc = -1;
  goto exit;
}
/* initialize host variable and indicators */
strcpy(result_tab,"xml_order_tab");
strcpy(result_colname, "xmlorder")
valid_colname = '¥0';
override[0] = '¥0';
overrideType = NO_OVERRIDE;
max_row = 500;
num_row = 0;
returnCode = 0;
msg_txt[0] = '¥0';
dad_ind = 0;
rtab_ind = 0;
rcol_ind = 0;
vcol_ind = -1;
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,
        :result_colname:rcol_ind,
```

```
        :valid_colname:vcol_ind,
        :overrideType:ovtype_ind,:override:ov_ind,
        :max_row:maxrow_ind,:num_row:numrow_ind,
        :returnCode:returnCode_ind,:returnMsg:returnMsg_ind);

if (SQLCODE < 0) {
    EXEC SQL ROLLBACK;
  else
    EXEC SQL COMMIT;
}

exit:
  return rc;
```

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。

更新頻度の高い XML 文書の合成

頻繁に更新される文書は、dxxRetrieveXML ストアード・プロシージャーを使用して合成してください。同じタスクが繰り返されるため、パフォーマンスの向上が重要になります。

ストアード・プロシージャー dxxRetrieveXML は、 dxxGenXML ストアード・プロシージャー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受け入れます。 XML コレクションが使用可能にされている場合、DAD ファイルは XML_USAGE 表に保管されます。したがって、XML エクステンダーは 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の方法で文書を合成します。

dxxRetrieveXML() ストアード・プロシージャーによって、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 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には、次のストアード・プロシージャー宣言を使用して、ストアード・プロシージャー呼び出しをアプリケーションに組み込みます。

```
dxxRetrieveXML(char() collectionName,  /* input */
               char() resultTabName,   /* input */
               char(30)           result_column, /* 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 */
```

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  collectionName[32]; /* name of an XML collection */
char  result_tab[32];     /* name of the result table */
char  result_colname[32]; /* name of the result column */
char  valid_colname[32];  /* name of the valid column, will set to NULL*/
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 collectionName_ind;
short rtab_ind;
short rcol_ind;
short vcol_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);

/* initial host variable and indicators */
strcpy(collection, "sales_ord");
strcpy(result_tab,"xml_order_tab");
strcpy(result_col,"xmlorder");
valid_colname[0] = '¥0';
override[0] = '¥0';
overrideType = NO_OVERRIDE;
max_row = 500;
num_row = 0;
returnCode = 0;
msg_txt[0] = '¥0';
collectionName_ind = 0;
rtab_ind = 0;
rcol_ind = 0;
vcol_ind = -1;
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.DXXRETRIEVE" (:collectionName:collectionName_ind,
                                    :result_tab:rtab_ind,
                                    :result_colname:rcol_ind,
                                    :valid_colname:vcol_ind,
                                    :overrideType:ovtype_ind,:override:ov_ind,
                                    :max_row:maxrow_ind,:num_row:numrow_ind,
                                    :returnCode:returnCode_ind,
            :returnMsg:returnMsg_ind);

if (SQLCODE < 0) {
    EXEC SQL ROLLBACK;
  else
    EXEC SQL COMMIT;
}
```

関連概念:

- 95 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』
- 175 ページの『XML コレクションのための DAD ファイル』

関連タスク:

- 69 ページの『RDB_node マッピングを使用した 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』
- 71 ページの『RDB_node マッピングを使用した XML 文書の分解』
- 106 ページの『XML コレクションのデータの更新、削除、および取り出し』
- 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 文書内のデータを細かくしてリレーショナル表に保管することです。 XML エクステンダーには、 XML データをソースの XML 文書から分解してリレーショナル表に入れるストアード・プロシージャーが提供されています。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しなければなりません。コレクションを使用可能にしなければならないのは、次の場合です。

- 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際に XML エクステンダーによって作成されるので、 XML コレクションを使用可能にしなければなりません。
- 複数出現するエレメントおよび属性のシーケンスを保持することが重要である場合。 XML エクステンダーが表の複数出現するエレメントまたは属性のシーケンスを保持するのは、それらが、コレクションが使用可能となっている際に作成された場合のみです。 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、シーケンス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際、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、

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 文書を分解するためには、DB2 UDB XML エクステンダーによって提供されている、 dxxShredXML() または dxxInsertXML の 2 つのストアード・プロシージャーのうち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、使用可能コレクションは必要ありません。代わりに 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。また、このプロシージャー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。これは、DAD ファイル内の <Xcollection> の指定に応じて、XML 文書を 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 文書を分解して、タグのない XML データを DAD ファイル内で指定された表に挿入します。 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ます。 XML コレクションが使用可能になっている必要はありません。

dxxShredXML() によって XML コレクションを分解する

次のストアード・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: 以下に dxxShredXML() への呼び出しの例を示します。

```
#include "dxx.h"
#include "dxxrc.h"

EXEC SQL INCLUDE SQLCA;
EXEC SQL BEGIN DECLARE SECTION;
SQL TYPE is CLOB(100K) dad;    /* DAD */

SQL TYPE is CLOB(100K) xmlDoc; /* input xml document */

long    returnCode;            /* return 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;

FILE   *file_handle;
long    file_length=0;
```

```
/* initialize the DAD CLOB object. */
file_handle = fopen( "/dxx
/dad/getstart_xcollection.dad", "r" );
if ( file_handle != NULL ) {
  file_length = fread ((void *) &dad.data, 1, FILE_SIZE,
  file_handle);
  if (file_length == 0) {
    printf ("Error reading dad file getstart_xcollection.dad¥n");
    rc = -1;
    goto exit;
 } else
    dad.length = file_length;
}
else {
  printf("Error opening dad file  ¥n");
  rc = -1;
  goto exit;
}

/* Initialize the XML CLOB object. */
  file_handle = fopen( "/dxx
/xml/getstart_xcollection.xml", "r" );
  if ( file_handle != NULL ) {
    file_length = fread ((void *) &xmlDoc.data, 1,
        FILE_SIZE, file_handle);
    if (file_length == 0) {
      printf ("Error reading xml file
         getstart_xcollection.xml ¥n");
      rc = -1;
      goto exit;
    } else
        xmlDoc.length = file_length;
  }
  else {
    printf("Error opening xml file ¥n");
    rc = -1;
    goto exit;
  }

/* initialize host variable and indicators */
returnCode = 0;
msg_txt[0] = '¥0';
dad_ind = 0;
xmlDoc_ind = 0;
returnCode_ind = -1;
returnMsg_ind = -1;

/* Call the store procedure */
EXEC SQL CALL "DB2XML.DXXSHRED"  (:dad:dad_ind;
                                  :xmlDoc:xmlDoc_ind,
                                  :returnCode:returnCode_ind,
                                  :returnMsg:returnMsg_ind);

if (SQLCODE < 0) {
    EXEC SQL ROLLBACK;
  else
    EXEC SQL COMMIT;
}

exit:
  return rc;
```

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 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 ファイルに関連した、使用可能 XML コレクションに挿入します。DAD ファイルには、コレクション表およびマッピングの指定が含まれています。コレクション表は、<Xcollection> での指定に応じて検査または作成されます。ストアード・プロシージャー dxxInsertXML() はその後、マッピングに従って XML 文書を分解し、タグのない XML データを、名前の指定された XML コレクションに挿入します。

合成のための対応するストアード・プロシージャーは dxxRetrieveXML() です。これはまた、使用可能 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: dxxInsertXML():

次のストアード・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    collectionName[32];   /* name of an XML collection */
SQL TYPE is CLOB(100K) xmlDoc; /* input xml document */
long    returnCode;            /* return error code */
char    returnMsg[1024];       /* error message text */
short   collectionName_ind;
short   xmlDoc_ind;
short   returnCode_ind;
short   returnMsg_ind;
EXEC SQL END DECLARE SECTION;

FILE   *file_handle;
long    file_length=0;

/* initialize the DAD CLOB object. */

file_handle = fopen( "dxxsamples/dad
/getstart_xcollection.dad", "r" );
 if ( file_handle != NULL ) {
  file_length = fread ((void *) &dad.data, 1, FILE_SIZE,
      file_handle);
 if (file_length == 0) {
    printf ("Error reading dad file getstart_xcollection.dad¥n");
    rc = -1;
    goto exit;
 } else
    dad.length = file_length;
}
else {
  printf("Error opening dad file  ¥n");
  rc = -1;
```

```
  goto exit;
}

/* initialize host variable and indicators */
strcpy(collectionName, "sales_ord");
returnCode = 0;
msg_txt[0] = '¥0';
collectionName_ind = 0;
xmlDoc_ind = 0;
returnCode_ind = -1;
returnMsg_ind = -1;

/* Call the store procedure */
EXEC SQL CALL "db2xml.DXXINSERTXML"
                    (:collection_name:collection_name_ind,
                     :xmlDoc:xmlDoc_ind,
                     :returnCode:returnCode_ind,
                     :returnMsg:returnMsg_ind);

if (SQLCODE < 0) {
    EXEC SQL ROLLBACK;
  else
    EXEC SQL COMMIT;
}

exit:
  return rc;
```

分解する表サイズの制限

分解では RDB_node マッピングを使用して、エレメントと属性値を抽出し表行に入れて 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 1024 行までを入れることができます。

例えば、XML 文書を 5 つの表に分解する場合、その 5 つの表のそれぞれに、該当する文書中の 1024 行までを入れることができます。複数の文書のための行を含む表の場合、各文書ごとに 1024 行までを入れることができます。

複数出現エレメント (XML 構造内で複数出現する可能性のあるロケーション・パスのエレメント) を使用すると、行数が影響を受けます。例えば、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場合、単一の文書から 1 つの表に分解できる最大の行数は 1024 である点に注意してください。

関連タスク:

- 71 ページの『RDB_node マッピングを使用した XML 文書の分解』
- 210 ページの『XML エクステンダーの合成ストアード・プロシージャーの呼び出し』

関連参照:

- 222 ページの『XML エクステンダーの分解ストアード・プロシージャー』
- 224 ページの『dxxInsertXML() ストアード・プロシージャー』
- 222 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の更新、削除、および取り出し

XML コレクションに対して、更新、削除、検索、および取り出しを行うことができます。しかし、XML コレクションを使用する目的は、タグのない純粋なデータをデータベース表に保管することです。既存のデータベース表内のデータは、着信する XML 文書とは無関係です。更新、削除、および検索操作は、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 エクステンダーは、XML コレクションのビューからデータを操作する機能を提供しています。 UPDATE および DELETE SQL ステートメントを使用することにより、 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できます。コレクション表で SQL 操作を行うと、生成された文書が影響を受けます。

制約事項:

- 文書を更新するとき、他のコレクション表にとっては外部キーである、表の基本キーを含む行を削除しないでください。基本キーおよび外部キーの行が削除されると、その文書は削除されます。
- エレメントおよび属性値を置き換えるためには、文書を削除することなく、下位レベルの表の行を削除および挿入することができます。
- 文書を削除するには、DAD に指定された先頭の element_node を合成する行を削除します。

XML コレクション内のデータを更新する

XML エクステンダーによって、 XML コレクション表に保管されたタグのないデータを更新することができます。 XML コレクション表の値を更新することにより、XML エレメントのテキスト、または XML 属性の値を更新することになります。さらに、更新によって、複数出現のエレメントまたは属性からデータのインスタンスを削除することができます。

SQL の観点からは、エレメントまたは属性の値を変更することは更新操作であり、エレメントまたは属性のインスタンスを削除することは削除操作です。 XML の観点からは、ルート element_node のエレメント・テキストまたは属性値が存在する場合、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。コレクション表で行った SQL 操作は、コレクション表から生成される文書に影響を与えます。

要件: XML コレクション内でデータを更新する際には、以下の規則に従ってください。

- 既存の表に基本キーと外部キーとの関係が存在する場合、コレクション表の間にもその関係を指定します。その関係が存在しない場合、結合される列が存在することを確認してください。
- DAD ファイルに指定された結合条件を、以下の場所に指定します。
  - SQL マッピングの場合、<SQL_stmt> エレメントに結合条件を含めます。
  - RDB_node マッピングの場合、ルート・エレメント・ノードの先頭の <condition> エレメントに結合条件を含めます。

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)
```

SHIP_TAB 表の <ShipMode> エレメントの値が AIR から BOAT に更新されます。キーは 68 です。

エレメントおよび属性のインスタンスを削除する

複数出現のエレメントまたは属性を削除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。

例えば、次の DELETE ステートメントでは、そのサブエレメントのいずれかの固有値を指定することにより <shipment> エレメントが削除されます。

```
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 つのみです。

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。コレクション表で SQL 操作を行うと、生成された文書が影響を受けます。

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場合は基本キー、下位レベルの表の場合は外部キーが含まれます。この方法によって文書を削除できるのは、基本キーおよび外部キーの制約が SQL で完全に指定されており、かつ DAD で示されている表の関係がその制約と正確に一致している場合に限られます。

例:

次の DELETE ステートメントは基本キー列の値を指定します。

```
DELETE from order_tab
   WHERE order_key=1
```

ORDER_KEY は ORDER_TAB 表 (DAD 文書で指定される最上位の表) 内の基本キーです。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 文書を XML コレクションから取り出す

XML 文書を XML コレクションから取り出すことは、文書をコレクションから合成することと似ています。

DAD ファイルの考慮事項: XML 文書を分解して XML コレクションにする場合、DAD ファイル内で順序を指定していない限り、複数出現のエレメントおよび属性値の順序が保持されないことがあります。この順序を保持するためには、RDB_node マッピング体系を使用してください。このマッピング体系により、RDB_node 内のルート・エレメントを含む表の orderBy 属性を指定することができます。

XML コレクションの検索

このセクションでは、検索基準を使用して XML 文書を生成したり、分解された XML データを検索したりするために、XML コレクションを検索する方法を解説します。

検索基準を使用した XML 文書の合成

このタスク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場合は、 *overwrite* パラメーターを指定します。

  例えば、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,

(select db2xml.generate_unique()
            as ship_id, date, mode from ship_tab) as s
       WHERE p.price > %d and s.date >'1996-06-01' AND
            p.order_key = o.order_key and s.part_key = p.part_key",
            price_value);

      /* Call the store procedure */
      EXEC SQL CALL db2xml.dxxRetrieve(:collection:collection_ind,
              :result_tab:rtab_ind,
              :overrideType:overrideType_ind,:overwrite:overwrite_ind,
              :max_row:maxrow_ind,:num_row:numrow_ind,
              :returnCode:returnCode_ind,:returnMsg:returnMsg_ind);
```

order.dad 内の price > 2500.00 の条件は、 price > ? によってオーバーライドされます。ここで、? は、入力変数 *price_value* に基づいています。

分解された XML データを検索する

コレクション表の検索には通常の SQL 照会操作を使用できます。コレクション表を結合するか、または副照会を使用してから、テキスト列に対して構造化テキスト検索を実行することができます。構造化検索の結果を適用して、指定した XML 文書を検索または生成してください。

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している場合、 XML データをリレーショナル・データベース内で表す方法を指定するマッピング体系 を選択する必要があります。 XML コレクションは、 XML 文書の階層構造をリレーショナル・データベースのリレーショナル構造と突き合わせなければならないため、ユーザーはそれら 2 つの構造の相違点を理解している必要があります。 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。

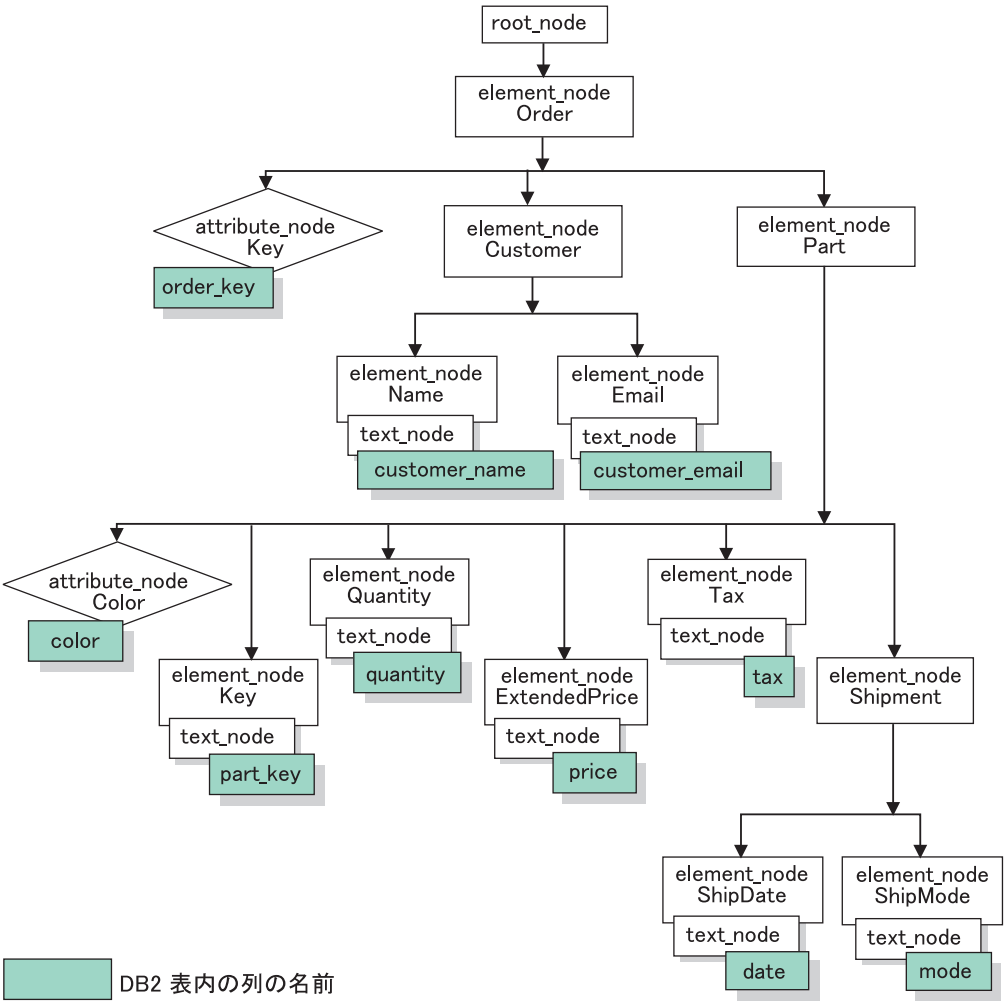


図 11. リレーショナル表の列にマップされた XML 文書構造

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

マッピング体系の種類:

DAD ファイルでマッピング・スキームを指定するには、<Xcollection> を使用してください。 XML エクステンダーには、*SQL マッピング* とリレーショナル・データベース *(RDB_node)* マッピング という、 2 種類のマッピング体系が備わっています。

SQL マッピング

この方法を使用すると、 1 つの SQL ステートメントでリレーショナル・データから XML 文書への直接的なマッピングを行うことができます。 SQL マッピングは合成にのみ使用されます。 <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、後で DAD 内の XML エレメントまたは属性にマップされる、 SELECT 文節内の列を指定します。 XML 文書の合成のために定義さ

れた場合、SQL ステートメントの SELECT 文節にある列名が、*attribute_node* の値または *text_node* の内容を、同じ *name_attribute* が指定されている列との関連付けに使用されます。 FROM 文節は、データを含む表を定義します。 WHERE 文節は、*結合* と*検索条件* を指定します。

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

RDB_node マッピング

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

この方式では、表、オプションの列、およびオプションの条件についてのノード定義が 1 つ以上入っている、 XML エクステンダーが提供する *RDB_node* を使用します。 DAD 内の <table> および <column> エレメントは、 XML データをデータベースに保管する方法を定義します。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 ファイルを作成しなければなりません。 112 ページの図 12 は、 3 つのリレーショナル表内のデータから一組の XML 文書を合成する、 XML コレクションの SQL マッピングを行うサンプルの DAD ファイルの一部を示しています。

```
<?xml version="1.0"?>
<!DOCTYPE DAD SYSTEM "dxx_install/samples/db2xml/dtd/dad.dtd">
<DAD>
  <dtdid>dxxsamples/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,
               (select db2xml.generate_unique()
                as ship_id, date, mode, from ship_tab) as
         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
    "
     dxxsamples/dtd/getstart.dtd"</doctype>
    <root_node>
    <element_node name="Order">
    <attribute_node name="key">
   <column name="order_key"/>
    </attribute_node>
    <element_node name="Customer">
     <text_node>
      <column name="customer"/>
     </text_node>
    <element_node>

  ...

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

図 12. SQL マッピング体系

XML エクステンダーには、 XML コレクション内のデータを管理するいくつかのストアード・プロシージャーが備わっています。これらのストアード・プロシージャーは両方のタイプのマッピングをサポートします。

関連概念:

- 175 ページの『XML コレクションのための DAD ファイル』
- 112 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 マッピングの要件』

関連タスク:

- 65 ページの『SQL マッピングを使用した XML 文書の合成』
- 69 ページの『RDB_node マッピングを使用した XML コレクションの合成』
- 71 ページの『RDB_node マッピングを使用した XML 文書の分解』

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> エレメント内で指定しなければなりません。 <SQL_stmt> には、照会*述部*で複数のリレーショナル表を結合することのできる、単一の SQL ステートメントが含まれていなければなりません。さらに、以下の文節が必要です。

- **SELECT 文節**
  - 列の名前が固有のものであることを確認してください。 2 つの表に同じ列名がある場合、AS キーワードを使用していずれか一方に別名を作成します。
  - 表を XML 文書の階層構造にマップする際に、同じ表の列同士をグループ化し、ツリー・レベルに基づいて表を配列してください。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例示しています。

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

    この例で、ORDER_TAB 表の order_key および 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・レベルになります。 SHIP_TAB 表の ship_id、date、および mode 列は、最低のリレーショナル・レベルになります。
  - 単一列候補キーを使用して、各レベルを開始します。そのようなキーが表にない場合、照会では表式および generate_unique()user-defined 関数を使用して、その表にこのキーを生成しなければなりません。上記の例で、o.order_key は ORDER_TAB の基本キー、そして part_key は PART_TAB の基本キーです。それらは、選択されるそれぞれの列グループの先頭にあります。 SHIP_TAB 表には基本キーがないため、ship_id は基本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 文節を使用して基本キー列を生成します。
- **FROM 文節**
  - 表式および generate_unique()user-defined 関数を使用して、基本単一キーのない表に単一キーを生成します。例えば、

    ```
    FROM order_tab as o, part_tab as p,
           (select
            db2xml.generate_unique() 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 ship_id という別名が付けられます。
  - 列を明瞭にする必要がある場合、別名を使用します。例えば、ORDER_TAB 表の列には o を、PART_TAB 表の列には p を、また、SHIP_TAB 表の列には s を使用することができます。
- **WHERE 文節**
  - 基本キーと外部キーとの関係を、表とコレクションを結び合わせる結合条件として指定します。例えば、

    ```
    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。列名の後には ASC または DESC などを付けないでください。
  - 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 は、SELECT 文節内の順序と同じ相対順序でリストしてください。
  - ID は、表式および generate_unique 関数またはユーザー定義関数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 文節の列は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、述部の式がその表の列を使用する限り WHERE 文節で任意の述部を指定できるため、非常に強力です。

関連参照:

• 257 ページの『付録 A. サンプル』

RDB_node マッピングの要件

マッピング方式として RDB_Node を使用する際、 <SQL_stmt> エレメントを DAD ファイルの <Xcollection> エレメント内で使用しないでください。その代わりに、<RDB_node> エレメントをエレメント・ノードの各トップ・ノード内で、また各属性ノードおよびテキスト・ノード内で使用します。

- **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付けられているすべての表を指定します。例えば、以下のマッピングは、先頭エレメント・ノードである <Order> の <RDB_node> 内に 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、省略することができます。

– 条件エレメントでは、1 つの列名を何回でも参照することができます。
– 文書を分解する場合、または DAD ファイルに指定されている XML コレクションを使用可能にする場合、表ごとに基本キーを指定しなければなりません。基本キーは、単一の列または複数の列 (複合キーと呼ばれる) から構成されます。基本キーは、RDB_node の表エレメントに属性キー を追加することによって指定します。複合キーを提供した場合、キー 属性はスペースで区切られた複数のキー列名で指定されます。例えば、

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

分解用に指定された情報は、同じ DAD が合成に使用される場合には無視されます。
– orderBy 属性を使用して、複数出現するエレメントまたは属性を含む XML 文書を再合成して元の構造に戻します。この属性により、文書の順序の保持に使用されるキーとなる列名を指定できます。 orderBy 属性は DAD ファイル内の表エレメントの一部であり、オプションの属性です。

表名および列名は、<table> タグ内で指定してください。

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

XML エクステンダーはデータベースのどこからデータを見つけるかを知る必要があります。 XML エクステンダーは、 XML 文書から得られた内容をデータベース内のどこに書き込むのかを知っていなければなりません。それぞれの属性ノードおよびテキスト・ノードごとに RDB_node を指定する必要があります。表および列名も指定しなければなりません。条件の値はオプションです。

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。条件に適合するデータのみが、生成される XML 文書に含まれます。条件は有効な WHERE 文節でなければなりません。上記の例では、price > 2500.00 という条件が指定されているため、 XML 文書には、価格が 2500 を超えている行のみが含まれます。
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、それぞれの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しなければなりません。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と、 XML コレクションを使用可能にする際

に新規の表が作成されたとき、各列に正しいデータ・タイプが割り当てられるようになります。列タイプは、属性タイプを列エレメントに追加することによって指定されます。例えば、

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

文書を分解する際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。文書の合成または分解時に XML エクステンダーがエレメント間の関係を理解できるように、エレメント・ノードを適切にネストするようにしてください。例えば、次の DAD ファイルは Shipment を Part 内に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 ファイルでは、 Part エレメントと Shipment エレメントが兄弟関係になった XML 文書が作成されます。

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

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

次のコードでは、 DAD ファイル内で Shipment エレメントが Part エレメントの内側に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 -->
```

Shipment エレメントを Part エレメントの内側にネストすると、 Shipment が Part エレメントの子エレメントとなる 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・ノード条件の述部の順序に関する制限はありません。

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。

element_nodes と attribute_nodes が同じ表にマップされる DAD のサブツリーでは、次のことが成立します。

- 属性ノードは、同じ表にマップされる要素ノードの共通の祖先のうち最低位のものの最初の子とは限りません。
- 属性ノードは、結合条件に関係していない限り、サブツリーの中のどこにでも出現する可能性があります。

制約事項: RDB_node マッピング DAD の中で可能な表の数は、 30 以下です。 1 つの表について可能な列の数は 500 です。さらに、条件ステートメントの結合述部の中で各表または各列を指定できる回数に制限はありません。

XML コレクション用のスタイルシート

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

```
<?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 文書の構造内における XML エレメントや属性の位置を定義します。 XML エクステンダーは、以下の目的でロケーション・パスを使用します。

- 抽出 UDF (dxxRetrieveXML のような) の使用時に抽出対象のエレメントおよび属性を識別するため
- DAD での XML 列の索引付け体系の定義の際に、 XML エレメント (または属性) と DB2® 列とのマッピングを指定するため
- Text Extender を使用した、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。

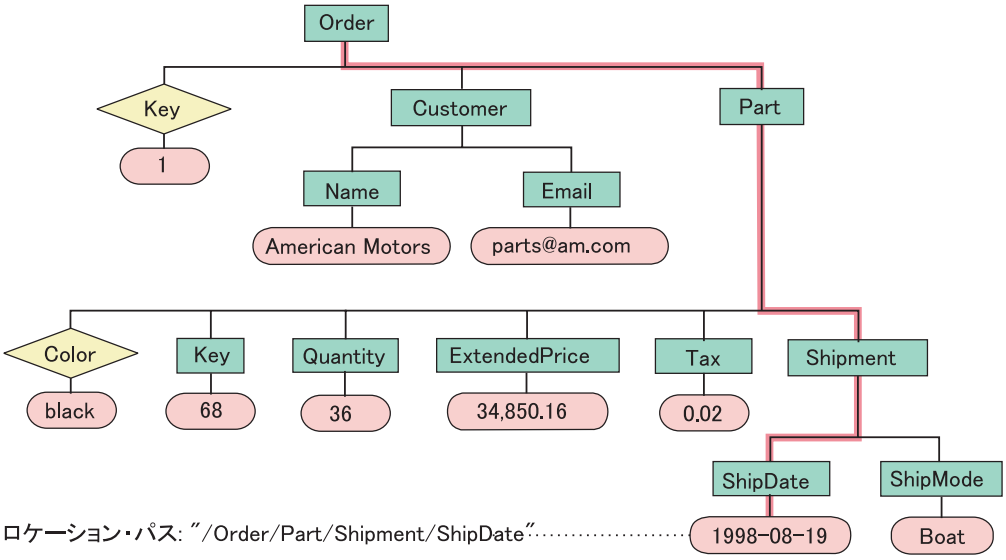


図 13. 文書を構造化 XML 文書として DB2 UDB 表の列に保管する

関連参照:

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

1. / XML ルート・エレメントを表します。これは、文書内にある他のすべてのエレメントを含むエレメントです。

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。 tag2 はルートの下にあるエレメント *tag1* の子です。

7. */tag1/tag2[@attr1="5"]*
   名前が *tag2* で属性 *attr1* の値が 5 であるエレメントを表します。 *tag2* はルートの下にある *tag1* エレメントの子です。

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

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・パス構文のタイプの 1 つです。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。それぞれの属性の値は、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 17 は、単純ロケーション・パスの構文を要約しています。

表 17. 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・パスの使用法

ロケーション・パスの構文は、どのようなコンテキストでエレメントあるいは属性のロケーションにアクセスするのかによって異なります。 XML エクステンダーは、エレメントまたは属性と DB2 列との間で 1 対 1 マッピングを使用するため、 DAD ファイルおよび関数の構文規則がそれによって制約を受けます。 120 ページの表 18 は、どのようなコンテキストでこの構文が使用されるかを説明しています。

表 18. ロケーション・パスの使用に関する XML エクステンダーの制限

ロケーション・パスの使用	サポートされるロケーション・パス
サイド表の XML 列 DAD マッピングのパス属性の値	3、6 (119 ページの表 17 で説明されている単純ロケーション・パス)
UDF の抽出	1 から 8 まで[1]
UDF の更新	1 から 8 まで[1]
Text Extender による UDF の検索	3 - 例外: スラッシュなしでルート・マークが指定された。例えば、`tag1/tag2/..../tagn`

[1] UDF の抽出および更新は、属性を指定した述部を持つロケーション・パスはサポートしますが、エレメントの指定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化すると、 DAD ファイルが解析されてその XML 文書に関連する表と列が識別され、制御情報が XML_USAGE 表に記録されます。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 文書を分解し、データを新しい DB2 UDB 表に保管する場合
- 複数の DB2 UDB 表に含まれる既存のデータに基づいて XML 文書を合成する場合

合成と分解で同じ DAD ファイルが使用される場合には、合成と分解の両方についてコレクションを使用可能にできます。

XML コレクションを使用可能にするには、XML エクステンダー管理ウィザード、**dxxadm** コマンド (enable_collection オプションを指定)、または XML エクステンダー・ ストアード・プロシージャー dxxEnableCollection() を使用します。

管理ウィザードを使用する場合:

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、

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)」ウィンドウが表示されます。
3. **「コレクションの使用可能化 (Enable a Collection)」**をクリックしてから、**「次へ」**をクリックします。「コレクションの使用可能化 (Enable a Collection)」ウィンドウが表示されます。
4. **「コレクション名 (Collection name)」**フィールドで、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 (DAD file name)」**フィールドに DAD ファイル名を指定します。
6. オプション: 以前に作成し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。

その表スペースに、分解用に生成される新しい DB2 UDB 表が入れられることになります。

7. **「完了」**をクリックします。コレクションが使用可能になり、「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化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 コマンドを使用した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、 DB2 UDB コマンド行に **dxxadm** コマンドを入力します。

構文:

パラメーター:

dbName
RDB 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
分解用に生成された新しい DB2 UDB 表を入れる既存の表スペース。これを指定しない場合、デフォルトの表スペースが使用されます。

例: コマンド行を使用することによって、データベース SALES_DB の中の sales_ord というコレクションを使用可能にする例を以下に示します。 DAD ファイルで SQL マッピングが使用されています。

Qshell から、

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

OS のコマンド行から、

```
CALL QDBXM/QZXMADM PARM(enable_collection SALES_DB sales_ord
  'getstart_collection.dad')
```

オペレーション・ナビゲーターから、

```
CALL MYSCHEMA.QZXMADM('enable_collection', 'SALES_DB', 'sales_ord',
  'getstart_collection.dad');
```

XML コレクションを使用可能にしたなら、 XML エクステンダーのストアード・プロシージャーを使用して XML 文書を合成したり分解したりできるようになります。

関連概念:

- 95 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』
- 96 ページの『XML コレクション内のデータの管理』

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除去されません。 DAD を更新してからコレクションを再び使用可能にする必要のある場合、またはコレクションを除去する場合には、コレクションを使用不可にします。

XML コレクションを使用不可にするには、XML エクステンダー管理ウィザード、**dxxadm** コマンド (disable_collection オプションを指定)、または XML エクステンダー・ ストアード・プロシージャー dxxDisableCollection() を使用します。

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、

1. 管理ウィザードを起動します。
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 エクステンダーのコレクション関連タスクを表示します。「タスクの選択 (Select a Task)」ウィンドウが表示されます。
3. XML コレクションを使用不可にするには、**「XML コレクションを使用不可にする (Disable an XML Collection)」**をクリックしてから、**「次へ」**をクリックします。「コレクションを使用不可にする (Disable a Collection)」ウィンドウが表示されます。
4. **「コレクション名 (Collection name)」**フィールドに、使用不可にするコレクションの名前を入力します。
5. **「完了」**をクリックします。コレクションが使用不可になり、「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** コマンドを入力してください。

構文:

パラメーター:

dbName
RDB 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:

Qshell から、

```
dxxadm disable_collection SALES_DB sales_ord
```

OS のコマンド行から、

```
CALL QDBXM/QZXMADM PARM(disable_collection SALES_DB sales_ord)
```

オペレーション・ナビゲーターから、

```
CALL MYSCHEMA.QZXMADM('disable_collection', 'SALES_DB', 'sales_ord');
```

関連概念:

- 95 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:

- 96 ページの『XML コレクション内のデータの管理』

関連参照:

- 204 ページの『XML エクステンダーの管理ストアード・プロシージャー』

第 5 章 XML スキーマ

XML 文書の内容データの仕様を定義するには、 DTD の代わりに XML スキーマも使用できます。 XML スキーマは、 XML 形式または SML 構文を使用することにより XML 文書のエレメントと属性名を定義し、それらのエレメントと属性の値として可能な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 スキーマと比較してコーディングおよび妥当性検査が容易です。しかし、XML スキーマを使用すると、以下の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することのできる、有効な XML 文書です。
- XML スキーマは DTD よりも強力です。 DTD によって定義できるものは、すべてスキーマによって定義することができますが、その逆は成り立ちません。
- XML スキーマでは、一連のデータ・タイプがサポートされています。それらは、多くのプログラミング言語のデータ・タイプによく似たものであり、独自のタイプを作成する機能も用意されています。文書の内容を特定のタイプのみに制限することができます。例えば、DB2 内のフィールドの特性を複製することができます。
- XML スキーマは、文字データに対する制約を設定するための正規表現をサポートします。 DTD を使用する場合、そのような指定は不可能です。
- XML スキーマのほうが、XML ネーム・スペースのサポート能力が高く、複数のネーム・スペースを使用する文書を妥当性検査したり、すでに定義されているスキーマの構成を別のネーム・スペースで再利用したりするために使用することができます。
- XML スキーマは、モジュール性のサポート能力が高く、 include エレメントおよび import エレメントを再利用することができます。
- XML スキーマはエレメント、属性、およびデータ・タイプの定義の継承をサポートします。

関連タスク:

- 127 ページの『スキーマでのデータ・タイプ、エレメント、および属性』

関連参照:

- 128 ページの『XML スキーマの例』

XML エクステンダーの UDT と UDF 名

DB2® 関数のフルネームは *schema-name.function-name* です。ここで *schema-name* は、SQL オブジェクトのセットを論理グループ化するための ID です。 XML エクステンダーの UDF および UDT のスキーマ名は DB2XML です。本書では、関数名のみを参照します。

関数パスにスキーマ名を追加すると、スキーマ名なしで UDT および UDF を指定することができます。関数パスは、スキーマ名の番号付きリストです。 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は、関数パスが CURRENT FUNCTION PATH 特殊レジスター内に設定されます。

XML スキーマの complexType エレメント

XML スキーマの complexType エレメントは、複数のサブエレメントからなるエレメント・タイプを定義するために使用されます。例えば、次のタグは、XML 文書における住所の射影を表しています。

```
<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 スキーマにより次のように定義されます。

```
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、xsd 接頭部が XML スキーマ・ネームスペースにバインドされていることが前提になっています。 2 行目から 5 行目までででは、 5 つのエレメントと 1 つの属性のシーケンスとして complexType USAddress が定義されています。これらのエレメントの順序は、 sequence タグの中での出現順序によって決定されます。

内側にあるエレメントは、データ・タイプ xsd:string または xsd:decimal から得られます。いずれのタイプも事前定義された単純データ・タイプです。

あるいは、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 タグを使用した場合、そのすべてのサブエレメントが必ず出現しなければなりませんが、特定の順序で指定する必要はありません。 choice タグを使用する場合、XML 文書に出現するのは、そのサブエレメントのうちちょうど 1 つだけでなければなりません。

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。

スキーマでのデータ・タイプ、エレメント、および属性

XML スキーマの基本データ・タイプ

XML スキーマには、組み込みの基本データ・タイプがいくつか用意されています。制約を適用することにより、それらのデータ・タイプから別のデータ・タイプを派生させることができます。

例 1 では、基本タイプ xsd:positiveInteger の範囲が 0 より大きく 100 未満の整数に制限されています。

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 は正規表現に限定されています。

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 に示すのは、組み込みストリング・タイプに基づく列挙タイプです。

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 スキーマのエレメント

XML スキーマ内でエレメントを宣言するには、名前とタイプを element エレメントの属性として指定しなければなりません。例えば、

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

また、属性 minOccurs および maxOccurs を使用して、そのエレメントが XML 文書内に現れる最小回数または最大回数を決定することができます。 minOccurs および maxOccurs のデフォルト値は 1 です。

XML スキーマの属性

属性宣言はエレメント定義の終わりに指定します。例えば、

```
<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:

- 169 ページの『妥当性検査関数』

関連参照:

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

XML スキーマの例

XML スキーマを作成する際には、最初に UML ツールを使用して XML 文書のデータ構造を設計するという方針で作業することをお勧めします。構造を設計した後で、その構造をスキーマ文書にマップします。以下に、XML スキーマの例を示します。

```
 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 行では、この XML スキーマが XML 1.0 と互換性を持ち、 Unicode 8 でデコードされることが宣言され、 XML スキーマの標準ネームスペースを使用することが指定されています。標準ネームスペースを使用することにより、基本的な XML スキーマ・データ・タイプおよび構造にアクセスできるようになります。

4 行目から 10 行目まででは、 1 人から n 人までの person (人物) のシーケンスからなる complexType として personnel (個人情報) を定義しています。complexType である person は 12 行目から 20 行目までで定義されています。これは、complexType のエレメント name とエレメント email からなります。 email エレメントはオプション (minOcccurs = '0') であり、 4 回まで出現可能です (maxOccurs = '4')。エレメントの出現回数が多ければ多いほど、スキーマの妥当性検査にかかる時間が長くなります。これに対し、DTD では、エレメントの出現回数として 0、1、または無制限しか選択することができません。

22 行目から 29 行目までは person のタイプとして使用される name タイプを定義しています。名前 (name) タイプは、family (姓) と given (名) のエレメントのシーケンスからなります。

31 行目から 33 行目までで、単純エレメントとしての *family*、*given*、および *email* が定義されています。それらの内容は string タイプであり、それは宣言済みです。

スキーマを使用した XML 文書インスタンス

次の例で示す XML 文書は、*personalnr.xsd* スキーマのインスタンスです。

```
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 を使用した XML 文書インスタンス

次の例は、この XML スキーマがどのようにして DTD として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 を使用した場合、 email の最大出現回数として 1 または無制限の出現以外の値を設定することはできません。

この DTD を使用した場合の XML 文書インスタンスは、2 行目が次のように変わることを除き、最初の例で示した XML 文書インスタンスと同じです。

```
<!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:

- 127 ページの『スキーマでのデータ・タイプ、エレメント、および属性』
- 169 ページの『妥当性検査関数』

関連参照:

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

第 6 章 dxxadm 管理コマンド

dxxadm コマンドの概要

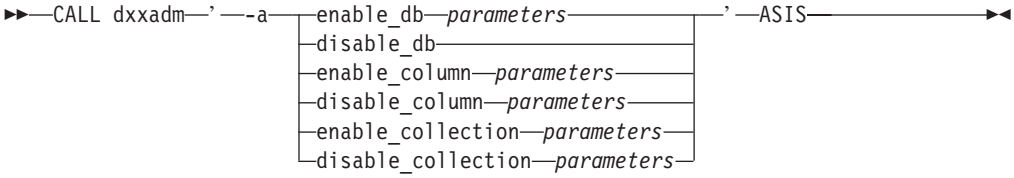
XML エクステンダーには、 Q-shell または OS のコマンド行から以下の管理作業を実行するための管理コマンド **dxxadm** が含まれています。

- データベースを XML エクステンダーに対して使用可能または使用不可にする
- XML 列を使用可能または使用不可にする
- XML コレクションを使用可能または使用不可にする

関連概念:

- 33 ページの『XML エクステンダーの管理ツール』
- 42 ページの『XML エクステンダーの管理計画』

dxxadm 管理コマンドの構文



パラメーター:

表 19. dxxadm パラメーター

パラメーター	説明
enable_db	データベースに対して XML エクステンダー機能を使用可能にします。
disable_db	データベースに対して XML エクステンダー機能を使用不可にします。
enable_column	XML 列を使用可能にして、そこに XML 文書を保管できるようにします。
disable_column	XML に使用できる列を使用不可にします。
enable_collection	指定された DAD に従って XML コレクションを使用可能にします。
enable_collection	XML 使用可能化されているコレクションを使用不可にします。

呼び出しは、 XML エクステンダーのロード・モジュール・ライブラリーが活動化されていることを前提とします。活動化していない場合には、**dxxadm** の完全修飾名を使用してください。

管理コマンド

システム・プログラマーは、以下の **dxxadm** を使用することができます。

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

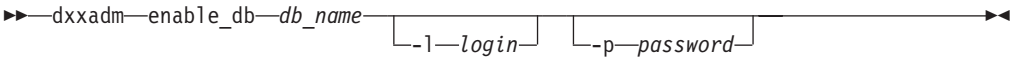
dxxadm コマンドの enable_db オプション

目的:

データベースに対して XML エクステンダー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 エクステンダーは以下のオブジェクトを作成します。

- XML エクステンダー・ ユーザー定義タイプ (UDT)
- XML エクステンダー・ ユーザー定義関数 (UDF)
- XML エクステンダー DTD 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 エクステンダー使用状況表 (XML_USAGE)。これは XML に使用できるそれぞれの列およびコレクションに関する一般情報を保管します。

構文:



パラメーター:

表 20. 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 RDB 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 (オプション)。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 (オプション)。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 を使用していて、データベースを使用可能化する際に表スペースを指定する場合は、その表スペースの作成時にノード・グループを指定していることが必要です。例えば、

```
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、この後、データベース使用可能化において mytb 表スペースを指定することになります。

データベース使用可能化において表スペース・オプションを指定しない場合、DXXDTDRF および DXXXMLUS の表スペースが存在するかどうかが XML エクステンダーによって確認されます。 DXXDTDRF 表スペースが存在する場合、db2xml.dtd_ref 表がその中に作成されます。また、DXXXMLUS 表スペースには db2xml.xml_usage 表が作成されます。 DXXDTDRF 表か DXXXMLUS 表のいずれかが存在しない場合、対応する表 (db2xml.dtd_ref または db2xml.xml_usage) は、それに最もふさわしい表スペースの中に作成されます。

データベース使用可能化時に DXXDTDRF 表スペースが 1 つしか指定されていない場合、指定された表スペースの中に 2 つの表が作成されます。データベース使用可能化時に 2 つの表スペースが指定された場合、最初に指定された表スペースの中に db2xml.dtd_ref 表が作成され、 2 番目に指定された表スペースの中に db2xml.xml_usage 表が作成されます。

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。

Qshell から、

```
dxxadm enable_db SALES_DB
```

OS のコマンド行から、

```
CALL QDBXM/QZXMADM PARM(enable_db SALES_DB)
```

iSeries ナビゲーターから、

```
CALL MYSCHEMA.QZXMADM('enable_db', 'SALES_DB');
```

関連参照:

- 131 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

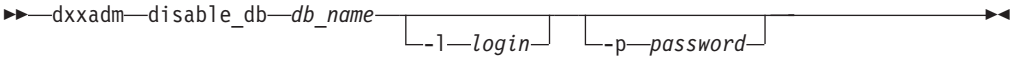
目的:

データベースで XML エクステンダー機能を使用できないようにします。このアクションは、「データベースの使用不可化」と呼ばれます。データベースが使用不可になると、 XML エクステンダーがそのデータベースを使用することはできなくなります。 XML エクステンダーがデータベースを使用不可にすると、以下のオブジェクトが除去されます。

- XML エクステンダー・ ユーザー定義タイプ (UDT)
- XML エクステンダー・ ユーザー定義関数 (UDF)
- XML エクステンダー DTD 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 エクステンダー使用状況表 (XML_USAGE)。これは XML に使用できるそれぞれの列およびコレクションに関する一般情報を保管します。

重要: データベースを使用不可にする前に、すべての XML 列を使用不可にする必要があります。 XML エクステンダーは、 XML に使用できる列またはコレクションを含んでいるデータベースを使用不可にできません。 XMLCLOB など XML エクステンダー・ ユーザー定義タイプにより定義された列を持つすべての表も除去する必要があります。

構文:



パラメーター:

表 21. 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 RDB 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。

Qshell から、

```
dxxadm disable_db SALES_DB
```

OS のコマンド行から、

```
CALL QDBXM/QZXMADM PARM(disable_db SALES_DB)
```

iSeries ナビゲーターから、

```
CALL MYSCHEMA.QZXMADM('disable_db', 'SALES_DB');
```

関連概念:

- 227 ページの『第 11 章 XML エクステンダーの管理サポート表』

関連参照:

- 204 ページの『XML エクステンダーの管理ストアード・プロシージャー』
- xi ページの『構文図の読み方』

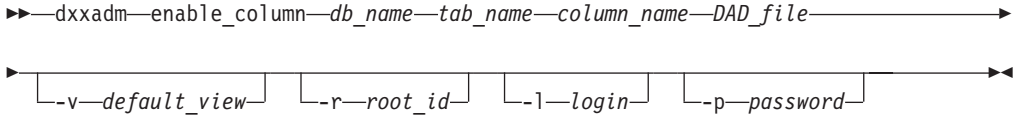
dxxadm コマンドの enable_column オプション

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 エクステンダー UDT を含めることができるようにします。 XML エクステンダーは、列を使用可能にする際に以下のタスクを行います。

- XML 表に基本キーがあるかどうかを判別します。基本キーがない場合は、XML エクステンダーが 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 表内の各行の固有な ID を含む列があります。この列は、ユーザーが指定したルート ID か、または XML エクステンダーが命名する DXXROOT_ID です。
- ユーザーが指定した名前をオプションで使用して 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)。

構文:



パラメーター:

表 22. 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 RDB 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。
-v *default_view*	XML 列とサイド表を結合するデフォルトのビューの名前。
-r *root_id*	サイド表の root_id として使用される、XML 列表内の基本キーの名前。 root_id はオプションです。
-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 を使用していて、列を使用可能化する際に表スペースを指定する場合は、その表スペースの作成時にノード・グループを指定していることが必要です。例えば、

```
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、この後、列の使用可能化において mytb 表スペースを指定することになります。

```
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。

Qshell から、

```
dxxadm enable_column SALES_DB MYSCHEMA.SALES_TAB ORDER getstart.dad
   -v sales_order_view -r INVOICE_NUMBER
```

OS のコマンド行から、

```
CALL QDBXM/QZXMADM PARM(enable_column SALES_DB 'MYSCHEMA.SALES_TAB'
   ORDER 'getstart.dad' '-v' sales_order_view '-r' INVOICE_NUMBER)
```

iSeries ナビゲーターから、

```
CALL MYSCHEMA.QZXMADM('enable_column', 'SALES_DB', 'MYSCHEMA.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

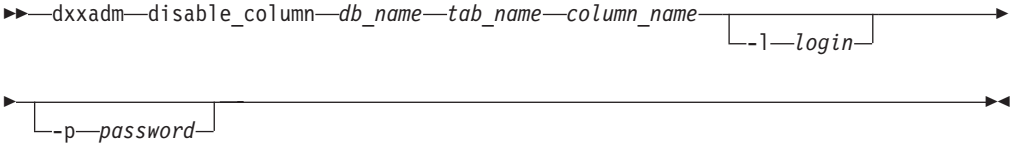
目的:

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、以下の処置が実行されます。

- XML 列の使用項目が XML_USAGE 表から削除されます。
- USAGE_COUNT が DTD_REF 表内で減分されます。
- この列に関連したすべてのトリガーが除去されます。
- この列に関連したすべてのサイド表が除去されます。

重要: XML 表を除去する前に、XML 列を使用不可にしなければなりません。 XML 表が除去されてもその XML 列が使用不可にされていない場合、 XML エクステンダーは作成したサイド表および XML 列項目の両方を XML_USAGE 表内に保ちます。

構文:



パラメーター:

表 23. 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 RDB 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。

表 23. *disable_column* のパラメーター *(続き)*

パラメーター	説明
-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。

Qshell から、

```
dxxadm disable_column SALES_DB MYSCHEMA.SALES_TAB ORDER
```

OS のコマンド行から、

```
CALL QDBXM/QZXMADM PARM(disable_column SALES_DB 'MYSCHEMA.SALES_TAB' ORDER)
```

iSeries ナビゲーターから、

```
CALL MYSCHEMA.QZXMADM('disable_column', 'SALES_DB',
'MYSCHEMA.SALES_TAB', 'ORDER');
```

関連参照:

- 137 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

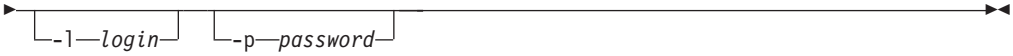
目的:

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 エクステンダーを実行している場合は、 DAD ファイルの中で指定したすべての表に、パーティション・キーとして指定可能な列が 1 つ以上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 エクステンダーは以下のタスクを実行します。

- XML コレクションの使用項目を XML_USAGE 表内に作成します。
- RDB_node マッピングでは、表がデータベースに存在しない場合、 DAD で指定されたコレクション表を作成します。

構文:

```
►►—dxxadm—enable_collection—db_name—collection_name—DAD_file————————————►
```



パラメーター:

表 24. 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 RDB 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。
-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 を使用していて、コレクションを使用可能化する際に表スペースを指定する場合は、その表スペースの作成時にノード・グループを指定していることが必要です。例えば、

```
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、この後、コレクションの使用可能化において mytb 表スペースを指定することになります。

例:

以下の例では、XML コレクションが使用可能にされます。

Qshell から、

```
dxxadm enable_collection SALES_DB sales_ord
getstart_xcollection.dad
```

OS のコマンド行から、

```
CALL QDBXM/QZXMADM PARM(enable_collection SALES_DB sales_ord
  'getstart_collection.dad')
```

iSeries ナビゲーターから、

```
CALL MYSCHEMA.QZXMADM('enable_collection', 'SALES_DB', 'sales_ord',
  'getstart_collection.dad');
```

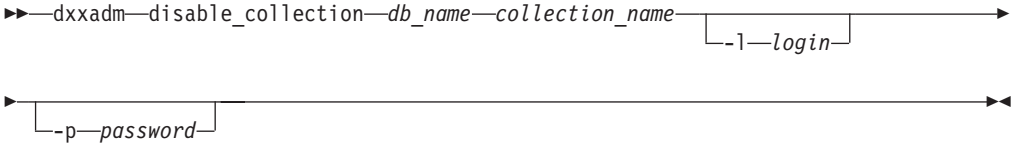
dxxadm コマンドの disable_collection オプション

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た際に作成されたコレクション表は除去されません。

構文:



パラメーター:

表 25. 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 RDB 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。

Qshell から、

```
dxxadm disable_collection SALES_DB sales_ord
```

OS のコマンド行から、

```
CALL QDBXM/QZXMADM PARM(disable_collection SALES_DB sales_ord)
```

iSeries ナビゲーターから、

```
CALL MYSCHEMA.QZXMADM('disable_collection', 'SALES_DB', 'sales_ord');
```

第 4 部 参照情報

ここでは、XML エクステンダー の管理コマンド、ユーザー定義データ・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に関する情報を提供します。また、問題判別のためのメッセージ・テキストも示します。

第 7 章 XML エクステンダーのユーザー定義タイプ

これらのデータ・タイプは、 XML 文書を保管するために使用される列をアプリケーション表で定義するために使用されます。また、ファイル名を指定して、XML 文書を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。

すべての XML エクステンダーのユーザー定義タイプは、修飾子 **DB2XML** を持っています。これは、DB2 UDB XML 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。例えば、

```
db2xml.XMLVarchar
```

XML エクステンダー は、XML 文書を保管および検索するための UDT を作成します。 表 26 に UDT の説明を示します。

表 26. XML エクステンダー UDT

ユーザー定義タイプ列	ソース・データ・タイプ	使用法の説明
XMLVARCHAR	VARCHAR(*varchar_len*)	XML 文書の全体を VARCHAR として DB2 内に保管します。
XMLCLOB	CLOB(*clob_len*)	XML 文書の全体を文字ラージ・オブジェクト (CLOB) として DB2 内に保管します。
XMLFILE	VARCHAR(512)	ローカル・ファイル・サーバーのファイル名を指定します。 XMLFILE を XML 列に指定した場合、XML エクステンダー は XML 文書を外部サーバー・ファイルに保管します。 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の間で、保全性がなければなりません。

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。

iSeries の XML エクステンダーでは、 *varchar_len* = 3K および *clob_len* = 10M となります。

XMLVARCHAR または XMLCLOB UDT のサイズを変更するには、データベースを XML エクステンダー対応にする前に UDT を作成してください。

手順:

使用可能化されたデータベースの XMLVARCHAR または XMLCLOB UDT のサイズを変更するには、

1. XML エクステンダー対応のデータベースに含まれるすべてのデータを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 列サイド表をドロップします。
3. コマンドを使用して、データベースを使用不可にします。
4. ユーザー定義タイプ XMLVARCHAR または XMLCLOB を作成します。
5. コマンドを使用して、データベースを使用可能にします。
6. 表を再作成および再ロードします。

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 エクステンダー が作成するサイド表には適用されません。

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』
- 95 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』
- 34 ページの『XML エクステンダーの管理作業の準備』
- 109 ページの『XML コレクションのマッピング体系』

第 8 章 XML エクステンダー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して、その後 SQL ステートメントで参照できるようになる機能です。この章では、 DB2 UDB XML エクステンダーで使用されるユーザー定義関数について説明します。

XML エクステンダーのユーザー定義関数のタイプ

XML エクステンダーには、XML 文書を保管、取り出し、検索、および更新する関数、そして XML エレメントまたは属性を抽出する関数が備わっています。 XML ユーザー定義関数 (UDF) を XML 列に使用することはできますが、 XML コレクションには使用できません。

すべての UDF について、そのスキーマ名は DB2XML です。

XML エクステンダー関数のタイプは、次のとおりです。

保管関数
保管関数は、XML 文書をそのまま XML データ・タイプとして XML 対応列に保管します。

取り出し関数
取り出し関数は、XML 文書を DB2® データベース内の XML 列から取り出します。

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 エクステンダーには、種々の SQL データ・タイプ用の抽出関数のセットが備わっています。

更新関数
更新関数は、XML 文書全体または指定したエレメントの内容または属性値を変更して、ロケーション・パスによって指定された更新値とともに XML 文書のコピーを戻します。

generate_unique 関数
generate_unique 関数は固有のキーを戻します。

妥当性検査関数
妥当性検査関数は、XML スキーマまたは DTD に基づいて XML 文書の妥当性検査を実行します。

XML のユーザー定義関数を使用すると、一般の SQL データ・タイプに対して検索を実行できます。さらに、DB2 UDB Text Extender を XML エクステンダーで使用すると、 XML 文書内のテキストに対して、構造化検索と全テキスト検索 を実行できます。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 サイトを使用しやすくすることができます。

制約事項: UDF でパラメーター・マーカーを使用するときには、 Java™ データベース (JDBC) 制限の要件として、 UDF 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。

保管関数

XML エクステンダーの保管関数の概要

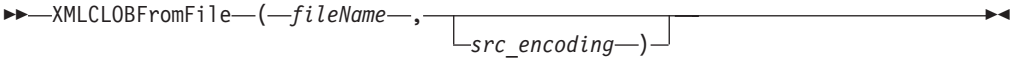
保管関数を使用して、XML 文書を DB2 UDB データベースに挿入します。 UDT のデフォルトのキャスト関数を INSERT または SELECT ステートメントで直接、使用できます。さらに、XML エクステンダーには XML 文書を UDT 基本データ・タイプ以外のソースから取り出して、指定した UDT に変換する UDF が備わっています。

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 27. 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 as LOCATOR

例:

以下の例では、サーバー上のファイルから XML 文書を読み取り、それを 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が iso-8859-1 として明示的に指定されています。

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

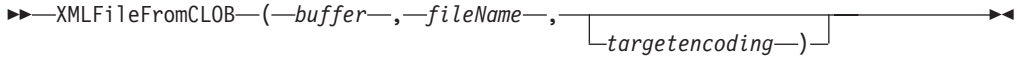
SALES_TAB 表の列 ORDER は、XMLCLOB タイプとして定義されています。

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 28. 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・サーバーで単一の 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。この関数により出力ファイルは ibm-808 でエンコードされます。

```
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, 'dxxsamples/xml/getstart.xml', 'ibm-808'))
```

SALES_TAB 表の列 ORDER は、XMLFILE タイプとして定義されています。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 29. 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。この関数により出力ファイルは iso-8859-1 でエンコードされます。

```
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, 'dxxsample/xml/getstart.xml', 'iso-8859-1'))
```

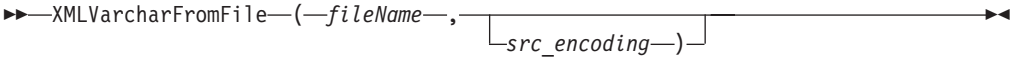
SALES_TAB 表の列 ORDER は、XMLFILE タイプとして定義されています。

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 30. 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。

表 30. XMLVarcharFromFile パラメーター (続き)

パラメーター	データ・タイプ	説明
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が ibm-808 として明示的に指定されています。

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

この例では、レコードが SALES_TAB 表に挿入されます。関数 XMLVarcharFromFile() は、エンコードが ibm-808 であるとして明示的に指定されているファイルから、 DB2 UDB に XML 文書をインポートし、それを XMLVARCHAR として保管します。

取り出し関数

XML エクステンダーの検索関数

XML エクステンダーには、取り出し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、名前が同じであっても、データの取り出し場所に基づいて振る舞いが異なる、取り出し関数のセットを参照します。また、デフォルト・キャスト関数を使用して、XML UDT を基本データ・タイプに変換することもできます。

Content() 関数は、次のタイプの取り出しを実行します。

- **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取り出し**

  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。この場合には、Content() を使用し、XMLFILE から取り出して CLOB に入れることができます。
- **内部ストレージ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 から取り出し外部サーバー・ファイルに入れる
  - Content(): XMLCLOB から取り出し外部サーバー・ファイルに入れる

下記のユーザー定義関数には、ソース・ファイルまたは出力ファイルのエンコード方式を指定するためのパラメーターが新たに設けられています。そのパラメーターの値は、 ICU によって認識されるコード・ページの名前です。

```
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 の内容をインポートして varchar バッファーに入れ、ソース・ファイルのエンコードが iso-8859-1 であることを指定するには、

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

ファイルが varchar にインポートされ、 iso-8859-1 からデータベースのコード・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、出力ファイルのエンコードが ibm-808 であることを指定するには、

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

バッファーの内容がファイルにエクスポートされ、データベースのコード・ページから ibm-808 に変換されます。それに応じて XML ファイルのエンコード宣言が更新されます。

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ます。

Content(): XMLFILE から取り出し CLOB に入れる

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 31. 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。

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

EXEC SQL CONNECT TO SALES_DB

EXEC SQL DECLARE c1 CURSOR FOR

     SELECT Content(order) from sales_tab
     WHERE sales_person = 'Sriram Srinivasan'

EXEC SQL OPEN c1;

do {
   EXEC SQL FETCH c1 INTO :xml_buff;
   if (SQLCODE != 0) {
      break;
      }
   else {
     /* do with the XML doc in buffer */
     }
   }

EXEC SQL CLOSE c1;

EXEC SQL CONNECT RESET;
```

SALES_TAB 内の列 ORDER は XMLFILE タイプなので、 Content() UDF はサーバー・ファイルからデータを検索してそれを CLOB ロケーターに保管します。

関連タスク:

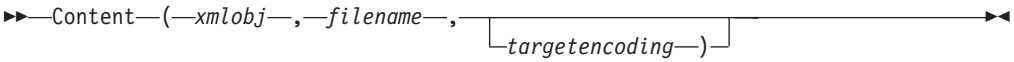
- 106 ページの『XML コレクションのデータの更新、削除、および取り出し』

Content(): XMLVARCHAR から取り出し外部サーバー・ファイルに入れる

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、 Content() 関数はそのファイルの内容をオーバーライドします。

パラメーター:

表 32. 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 の内容を検索して、それをサーバー上の外部ファイルに保管します。 UDF によりファイルは ibm-808 でエンコードされます。

```
CREATE table app1 (id int NOT NULL, order DB2XML.XMLVarchar);
INSERT into app1 values (1, '<?xml version="1.0"?>
   <!DOCTYPE SYSTEM
"dxxsample/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, '
dxxsamples/dad/getstart_column.dad', 'ibm-808')
  from app1 where ID=1;
```

関連タスク:

- 85 ページの『XML 文書を取り出す方法』

関連参照:

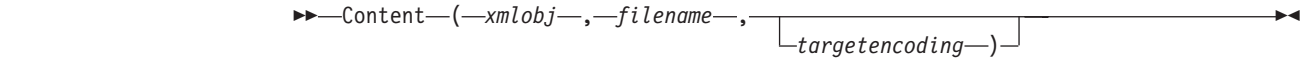
- 149 ページの『XML エクステンダーの検索関数』

Content(): XMLCLOB から取り出し外部サーバー・ファイルに入れる

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、 Content() 関数はそのファイルの内容をオーバーライドします。

パラメーター:

表 33. 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 as LOCATOR	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、XMLCLOB タイプとして保管されている XML の内容を検索して、それをサーバー上の外部ファイルに保管します。 UDF によりファイルは ibm-808 でエンコードされます。

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

INSERT into app1 values (1, '<?xml version="1.0"?>
  <!DOCTYPE SYSTEM
"dxxsamples/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,
'dxxsamples/xml/getstart.xml', 'ibm-808')
 from app1 where ID=1;
```

抽出関数

XML エクステンダーの抽出関数

抽出関数は、XML 文書からエレメント内容または属性値を抽出して、要求された SQL データ・タイプを戻します。 XML エクステンダーには、種々の SQL データ・タイプ用の抽出関数のセットが備わっています。抽出関数には、2 つの入力パラメーターがあります。 1 番目のパラメーターは XML エクステンダー UDT で、それには XML UDT の 1 つを指定できます。 2 番目のパラメーターは、XML エレメントまたは属性を指定するロケーション・パスです。それぞれの抽出関数は、ロケーション・パスによって指定される値を戻します。

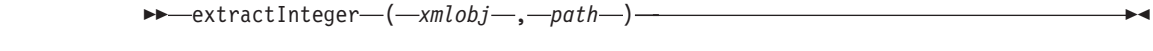
例では、各コマンドの先頭に「DB2」とタイプする必要のない、 DB2 UDB コマンド・シェルを使用していると仮定します。

extractInteger() および extractIntegers()

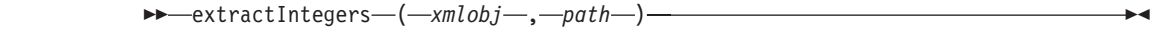
目的:

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:



表関数:



パラメーター:

表 34. extractInteger 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 34. extractInteger 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:

INTEGER

例:

以下の例では、キーの属性値が ″1″ のときに 1 つの値が戻されます。この値は 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('
    dxxsamples/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('dxxsamples/xml/getstart.xml'),
'/Order/Part/key')) AS X;
```

関連概念:

- 125 ページの『XML エクステンダーの UDT と UDF 名』
- 145 ページの『XML エクステンダーのユーザー定義関数のタイプ』

関連参照:

- 154 ページの『XML エクステンダーの抽出関数』

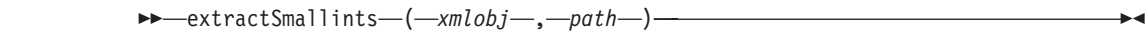
extractSmallint() および extractSmallints()

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:

表関数:



パラメーター:

表 35. extractSmallint 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

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('dxxsamples/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('dxxsamples/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』
- 125 ページの『XML エクステンダーの UDT と UDF 名』
- 145 ページの『XML エクステンダーのユーザー定義関数のタイプ』

関連参照:

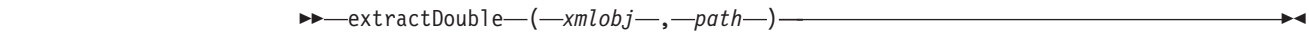
- 154 ページの『XML エクステンダーの抽出関数』
- 232 ページの『XML エクステンダーのストアード・プロシージャーの戻りコード』

extractDouble() および extractDoubles()

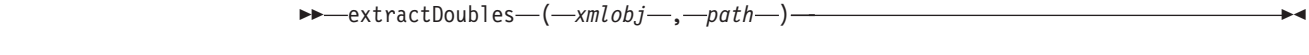
目的:

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:



表関数:



パラメーター:

表 36. extractDouble 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

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('dxxsamples/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('dxxsamples/xml/getstart.xml'),
      '/Order/Part/ExtendedPrice'))   AS X;
```

関連概念:

- 125 ページの『XML エクステンダーの UDT と UDF 名』

関連参照:

- 154 ページの『XML エクステンダーの抽出関数』

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:

表関数:

パラメーター:

表 37. extractReal 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

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('dxxsamples/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('dxxsamples/xml/getstart.xml'),
      '/Order/Part/ExtendedPrice'))   AS X;
```

関連概念:

- 125 ページの『XML エクステンダーの UDT と UDF 名』
- 145 ページの『XML エクステンダーのユーザー定義関数のタイプ』

関連参照:

- 154 ページの『XML エクステンダーの抽出関数』
- 231 ページの『XML エクステンダー 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:

表関数:

パラメーター:

表 38. extractChar 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

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('dxxsamples/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('dxxsamples/xml/getstart.xml'),
      '/Order/Part/@color'))   AS X;
```

関連参照:

- 154 ページの『XML エクステンダーの抽出関数』
- xi 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:

表関数:

パラメーター:

表 39. extractVarchar 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)

例:

SALES_TAB 表の ORDER 列に保管されている XML 文書の数が 1000 を超えているデータベースで、 ExtendedPrice の値が 2500.00 を超える商品を注文したすべての顧客を探し出したい場合があります。次の SQL ステートメントは、SELECT 文節で抽出 UDF を使用しています。

```
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 ORDER 列を入力として使用し、ロケーション・パス /Order/Customer/Name を選択 ID として使用します。この UDF は顧客の名前を戻します。この WHERE 文節があるため、抽出関数は ExtendedPrice が 2500.00 を超える注文のみを評価します。

表関数の例:

SALES_TAB 表の ORDER 列に保管されている XML 文書の数が 1000 を超えているデータベースで、 ExtendedPrice の値が 2500.00 を超える商品を注文したすべての顧客を探し出したい場合があります。次の SQL ステートメントは、SELECT 文節で抽出 UDF を使用しています。

```
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 ORDER 列を入力として使用し、ロケーション・パス /Order/Customer/Name を選択 ID として使

用します。この UDF は顧客の名前を戻します。この WHERE 文節があるため、抽出関数は ExtendedPrice が 2500.00 を超える注文のみを評価します。

次の例では、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('dxxsamples/xml/getstart.xml'),
    '/Order/Customer/Name'));
SELECT * from t1;
```

表関数の例:

次の例では、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('dxxsamples/xml/getstart.xml'),
      '/Order/Part/@color'))   AS X;
```

関連概念:

- 125 ページの『XML エクステンダーの UDT と UDF 名』
- 145 ページの『XML エクステンダーのユーザー定義関数のタイプ』

関連参照:

- 154 ページの『XML エクステンダーの抽出関数』
- 231 ページの『XML エクステンダー 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のみしか戻しません。extractClob(s) 関数は文書の一部を抽出するために使用するのに対し、extractVarchar(s) および extractChar(s) は値のみを抽出するために使用します。

構文:

表関数:

パラメーター:

表 40. *extractCLOB* 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)

例:

この例では、すべての name エレメントの内容とタグが注文書から抽出されます。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('dxxsamples/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('dxxsamples/xml/getstart.xml'),
      '/Order/Part/@color'))   AS X;
```

関連概念:

- 145 ページの『XML エクステンダーのユーザー定義関数のタイプ』

関連参照:

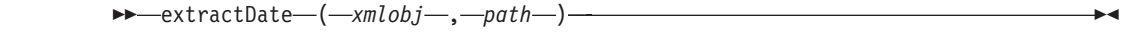
- 154 ページの『XML エクステンダーの抽出関数』

extractDate() および extractDates()

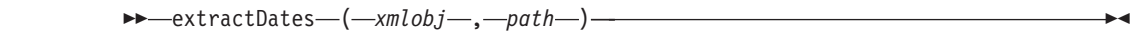
目的:

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 形式で指定する必要があります。

構文:



表関数:



パラメーター:

表 41. extractDate 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

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('dxxsamples/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('dxxsamples/xml/getstart.xml'),
      '/Order/Part[@color="black "]/Shipment/ShipDate'))   AS X;
```

関連概念:

- 145 ページの『XML エクステンダーのユーザー定義関数のタイプ』

関連参照:

- 154 ページの『XML エクステンダーの抽出関数』
- 231 ページの『XML エクステンダー 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:

表関数:

パラメーター:

表 42. extractTime 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:

TIME

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。

```
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:

- 125 ページの『XML エクステンダーの UDT と UDF 名』
- 145 ページの『XML エクステンダーのユーザー定義関数のタイプ』

関連参照:

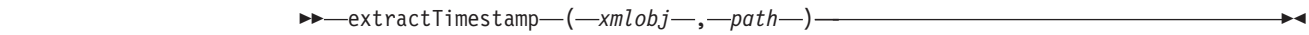
- 154 ページの『XML エクステンダーの抽出関数』

extractTimestamp() および extractTimestamps()

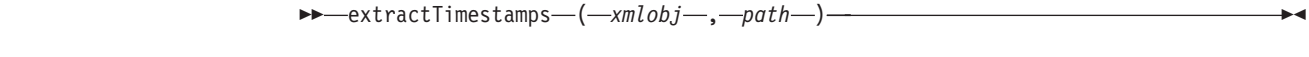
目的:

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:



表関数:



パラメーター:

表 43. extractTimestamp 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

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。

```
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 エクステンダーは、XML 文書から抽出したタイム・スタンプを、必要に応じて DB2 のタイム・スタンプ形式に適するように自動的に正規化します。タイム・スタンプは、 `yyyy-mm-dd-hh.mm.ss.nnnnnn` という形式か、または `yyyy-mm-dd-hh mm.ss.nnnnnn` という形式に正規化します。例えば、

```
2003-1-11-11.12.13
```

は、次のように正規化されます。

```
2003-01-11-11.12.13.000000
```

関連概念:

- 125 ページの『XML エクステンダーの UDT と UDF 名』
- 145 ページの『XML エクステンダーのユーザー定義関数のタイプ』

関連参照:

- 154 ページの『XML エクステンダーの抽出関数』

• 231 ページの『XML エクステンダー UDF の戻りコード』

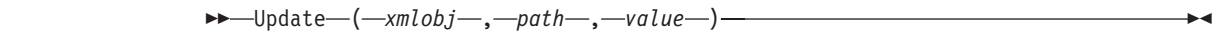
XML エクステンダーの更新関数

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() 関数を使用すると、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 44. 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。 **制約事項:** 更新関数には、出力エスケープを使用不可にするオプションはありません。したがって、この関数を使用して (タグ付きフラグメントである) extractClob の出力を挿入することはできません。記述したとおりの値のみが使用できます。

制限: Update UDF は、述部に属性が含まれているロケーション・パスはサポートしますが、述部にエレメントを含むものはサポートしない点に注意してください。例えば、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 as LOCATOR	XMLCLOB

例

以下の例では、販売員 Sriram Srinivasan によって扱われた注文を更新します。

```
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 つ以上の XML 文書内の値を変更すると、 XML 列内の XML 文書が置換されます。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変更されるかの例を示します。

Update() 関数が XML 文書を処理する方法

Update() 関数が XML 文書を置換する場合、 XML 構文解析プログラムの出力に基づいて文書を再構成する必要があります。 表 45 では、文書の各部分を処理する方法を例とともに示しています。

表 45. 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 宣言は保存されます。
DOCTYPE 宣言		文書タイプ宣言は以下のものを保存します。
	<!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] >	

表 45. 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
コメント	`<!-- comment -->`	ルート・エレメントの外部のコメントは保存されます。 ルート・エレメントの内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。
テキスト・ノード	`This chapter 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。

複数出現

Update() UDF にロケーション・パスを指定した場合、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、指定した値で更新されます。つまり、文書に複数出現するロケーション・パスがある場合、 Update() 関数は、既存の値を *value* パラメーターで指定した値で置換します。

path パラメーターに述部を指定して、明確なロケーション・パスを指定することによって、意図しない更新を防止することができます。 Update() UDF は、属性がある述部を持つロケーション・パスはサポートしますが、エレメントがあるものはサポートしません。

固有文字生成関数

目的

固有文字生成関数は、実行するたびに固有の文字ストリングを戻します。この関数には引き数はありません (空の括弧は指定する必要があります)。この関数の結果は、固有の値です。結果がヌルになることはありません。

構文

戻り値

VARCHAR(13)

例

次の例では、db2xml.generate_unique() を使用して、索引を付ける列に対して固有キーを生成しています。

```
<SQL_stmt>
SELECT o.order_key, customer_name, customer_email, p.part_key, color,
quantity, price, tax, ship_id, date, mode from order_tab o, part_tab p,
 (select db2xml.generate_unique()
  as ship_id, date, mode, part_key from ship_tab) as 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>
```

妥当性検査関数

DB2 XML エクステンダーには、 XML スキーマまたは DTD に基づいて XML 文書の妥当性検査を実行するユーザー定義関数 (UDF) が 2 つ提供されています。

XML 文書内のエレメントは、関連するエレメント・タイプ・ルールが満たされる場合には、決められたスキーマに従って有効になります。すべてのエレメントが有効な場合には、文書全体が有効です。しかし DTD の場合、特定のルート・エレメントを必須として指定する手段がありません。妥当性検査は、文書が有効なら 1 を戻します。また、文書が無効なら 0 を戻し、トレース・ファイルにエラー・メッセージを書き込みます。それらの関数は、以下のとおりです。

db2xml.svalidate:
指定されたスキーマに基づいて、XML 文書インスタンスの妥当性検査を実行します。

db2xml.dvalidate:
指定された DTD に基づいて、XML 文書インスタンスの妥当性検査を実行します。

SVALIDATE() 関数

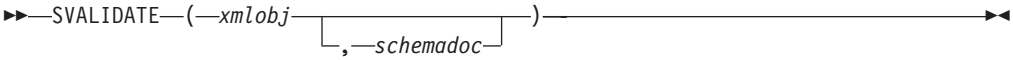
この関数は、指定されたスキーマ (または XML 文書の中で指定されているスキーマ) に基づいて XML 文書の妥当性検査を実行し、文書が有効な場合には 1 を、文書が無効な場合には 0 を戻します。この関数では、 XML 文書とスキーマがファイル・システムに存在しているか、または DB2 の CLOB として存在していることが前提になっています。

SVALIDATE 関数を実行する前に、次のコマンドを実行することによって、 XML エクステンダーをデータベースで使用できるようにしてください。

```
CALL QDBXM/QZXMADM PARM(enable_db mydbname)
```

XML 文書が妥当性検査に失敗した場合、 XML エクステンダーのトレース・ファイルにエラー・メッセージが書き込まれます。 SVALIDATE コマンドを実行する前に、トレースを使用可能にしてください。トレースを使用可能にする方法については、 229 ページの『XML エクステンダーのトレースの開始』を参照してください。

構文



パラメーター

表 46. SVALIDATE のパラメーター

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: この例では、 equiplog2001.xml について、その文書内で指定されているスキーマに基づいて妥当性検査を実行します。

```
select db2xml.svalidate('/dxxsamples/xml/equiplog2001..xml') from db2xml.onerow
```

例 2: この例では、指定したスキーマを使用して XML 文書の妥当性検査を実行します。文書もスキーマも、DB2 UDB の表に保管されています。

```
select db2xml.svalidate(doc,schema) from db2xml.onerow
```

DVALIDATE() 関数

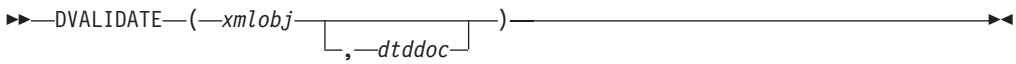
この関数は、指定された DTD (または XML 文書の中で指定されている DTD) に基づいて XML 文書の妥当性検査を実行し、文書が有効な場合には 1 を、文書が無効な場合には 0 を戻します。この関数では、 XML 文書と DTD がファイル・システムに存在しているか、または DB2 の CLOB として存在していることが前提になっています。

DVALIDATE 関数を実行する前に、次のコマンドを実行することによって、データベースで XML エクステンダーを使用できるようにしてください。

```
CALL QDBXM/QZXMADM PARM(enable_db mydbname)
```

XML 文書が妥当性検査に失敗した場合、 XML エクステンダーのトレース・ファイルにエラー・メッセージが書き込まれます。 SVALIDATE コマンドを実行する前に、トレースを使用可能にしてください。トレースを使用可能にする方法については、 229 ページの『XML エクステンダーのトレースの開始』を参照してください。

構文



パラメーター

表 47. DVALIDATE のパラメーター

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 が入っている XML 列 (DTD_REF 表のものか、正規の表のもの)。

例

例 1: この例では、 equiplog2001.xml について、その文書内で指定されている DTD に基づいて妥当性検査を実行します。

```
select db2xml.dvalidate('/dxxsamples/xml/equiplog2001.xml') from db2xml.onerow
```

例 2: この例では、指定した DTD を使用して XML 文書の妥当性検査を実行します。文書も DTD も、ファイル・システム内にあります。

```
select db2xml.dvalidate('/dxxsamples/xml/equiplog2001.xml',
'/dxxsamples/dtd/equip.dtd') from db2xml.onerow
```

例 3: この例では、指定した DTD を使用して XML 文書の妥当性検査を実行します。文書も DTD も、DB2 UDB の表の中に保管されています。

```
select db2xml.dvalidate (doc,dtdid) from equiplogs, db2xml.dtd_ref ¥
    where dtdid='equip.dtd'
```

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 列の定義および使用可能化という、より大きな作業の一部です。

DAD ファイルに関する最新情報については、 XML エクステンダーの Web サイト (www.ibm.com/software/data/db2/extenders/xmlext/downloads.html) を参照してください。

XML データにアクセスして、XML 表内の XML データを列で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、列内で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 列については、DAD ファイルはまず、その列に保管された文書にどのように索引を付けるかを指定します。 DAD ファイルは、 XML 列に挿入される文書の妥当性検査を実行するために使用する 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、そのサイズは 100 KB 以下です。

前提条件:

DAD ファイルを作成する前に、以下のことを行う必要があります。

- 検索で頻繁に使用することが予想されるエレメントまたは属性を判別する。指定するエレメントまたは属性は、 XML エクステンダーによって迅速に検索できるように、指定したサイド表に抽出されます。
- サイド表で索引付けされるそれぞれのエレメントまたは属性を表すロケーション・パスを定義する。エレメントまたは属性をどのようなデータ・タイプに変換するのかについて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、

1. テキスト・エディターで新規文書を作成し、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、その DAD ファイルの DTD のパスおよびファイル名です。 DTD は、 `dxx_install¥samples¥db2xml¥dtd` に入っています。
2. ステップ 1 の行の後に DAD タグを挿入します。

   ```
   <DAD>
   </DAD>
   ```

   このエレメントに、他のすべてのエレメントが入れられることになります。
3. 文書および列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、 DTD またはスキーマに基づいてその文書全体の妥当性検査を実行するには、
     – 文書を妥当性検査する方法を指定するためのタグを挿入します。

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

– スキーマを使用して文書の妥当性検査を実行するために、次のタグを挿入します。

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

– 次のタグを挿入することにより、列の妥当性検査を実行します。

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

- 文書の妥当性検査を実行しない場合には、次のタグを使用します。

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

4. XML データのアクセスおよび保管方式として XML 列を使用することを指定するために、 <Xcolumn> </Xcolumn> タグを挿入します。
5. サイド表を指定します。作成するサイド表ごとに、
   a. <table></table> タグを指定します。例えば、

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

   b. 表タグの中で、サイド表に含めたいそれぞれの列ごとに <column> タグを挿入します。それぞれの列には、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 サイド表に作られる列の名前を指定します。

type それぞれの索引付きエレメントまたは属性ごとに、サイド表での SQL データ・タイプを示します。

path 索引付けするそれぞれのエレメントまたは属性ごとに、 XML 文書内のロケーション・パスを指定します。

multi_occurrence
path 属性で参照された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示します。 **multi_occurrence** として可能な値は、 **YES** または **NO** です。値が **NO** である場合、表ごとに複数の列を指定できます。値が **YES** である場合、サイド表で 1 つの列しか指定できません。

6. DAD 拡張子を付けてファイルを保管します。

次の例は、完全な DAD ファイルを示しています。

```
<?xml version="1.0"?>
<!DOCTYPE DAD SYSTEM "dxxsamples¥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>
```

以上で DAD ファイルが作成されました。次に、XML 列を定義して使用可能にするための次のステップとして、XML 文書を保管する表を作成します。

関連概念:

- 95 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』
- 175 ページの『XML コレクションのための DAD ファイル』
- 191 ページの『DAD チェッカー』

関連タスク:

- 191 ページの『DAD チェッカーの使用』

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 ファイルは XML 文書の構造を、その文書の合成元である DB2® 表にマップします。 DAD ファイルを使用して文書を DB2 UDB 表に分解することもできます。

例えば、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ことです。 XML データとリレーショナル・データとの関係を定義するために DAD ファイルを使用します。

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

XML コレクションのアクセスおよび保管の方法を指定するには、DAD ファイル内で <Xcollection> タグを使用してください。

<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合成して 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としては、既存の業務データ用の表、または XML エクステンダーが最近作成した表などがあります

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

root_node
文書のルート・エレメントを指定します。

element_node
エレメントを識別します。ルート・エレメントまたは子エレメントのいずれかです。

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。

177 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一部を示しています。このノードは、XML 文書の内容をリレーショナル表内の表列にマップします。

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

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

図 14. XML コレクション表にマップされる 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 エクステンダー管理ウィザードまたはエディターを使用して、 DAD ファイルの作成および更新を実行できます。

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

同じ名前の複数の列を使用して XML 文書を合成することができます。選択された複数の列が同じ名前になっているときには、それらが異なる表から得られた場合であっても、変数を固有の別名で識別して、 SQL ステートメントの select 文節内の各変数を異なる名前にしなければなりません。次の例は、同じ名前の列に固有の別名を割り当て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>
```

ランダムに生成した値による列を使用して XML 文書を合成することもできます。DAD ファイル内の SQL ステートメントでランダム値が指定されている場合には、ランダム値関数に別名を割り当てて、それを ORDER BY 文節で使用しなければなりません。このようにする必要があるのは、値が表内のどの列にも関連付けられていないためです。次の例の ORDER BY 文節の終わりにある 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 db2xml.generate_unique()
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

RDB ノードの合成には、以下の制約事項が適用されます。

- root_node 以外の RDB ノードの DAD ファイルに関連付けられた条件は、リテラルと比較しなければなりません。
- 最上位 RDB_node に関連付けられた条件内のそれぞれの等式は、 2 つの表の列の結合関係を指定するもので、他の等式とは別に適用されます。つまり、AND で結合されたすべての述部は、1 つの結合条件で同時に適用されることはなく、文書合成時に外側の結合がシミュレートされます。それぞれの表のペアの親子関係は、DAD ファイル内における相対的なネスティングによって決定されます。例えば、

```
<condition>order_tab.order_key=part_tab.order_key AND
part_tab.part_key=ship_tab.part_key</condition>
```

NULL 値を含む行による合成

NULL 値を含む列を使用して XML 文書を合成することができます。

次の例は、列 *Col 1* の値が NULL である行が含まれている表 *MyTable* から、XML 文書を生成する方法を示すものです。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>
```

```
</roott_node>
</Xcollection>
</DAD>

MyTable

┌──────────┐
│ Col 1 │
├──────────┤
│ 1 │
├──────────┤
│ 3 │
├──────────┤
│ - │
└──────────┘
```

tests2x mydb nullcol.dad result_tab を実行するか、dxxGenXML を使用することによって、次の文書を生成します。 3 番目の Column1 エレメントは NULL 値を表している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>
```

- root_node 以外の RDB ノードの DAD ファイルに関連付けられた条件は、リテラルと比較しなければなりません。
- DAD の中でさらに低レベルの RDB ノードに関連した条件は、リテラルと比較しなければなりません。
- root_node と関連付けられた条件では、RDB ノードの合成に関わる表同士の関係を記述します。例えば、基本外部キーの関係を記述します。
- 最上位 RDB_node に関連付けられた条件内のそれぞれの等式は、 2 つの表の列の結合関係を指定するもので、他の等式とは別に適用されます。つまり、AND で結合されたすべての述部は、1 つの結合条件で同時に適用されることはなく、文書合成時に外側の結合がシミュレートされます。それぞれの表のペアの親子関係は、DAD ファイル内における相対的なネスティングによって決定されます。例えば、

```
<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

ここでは、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。その DAD ファイルの DTD は、次の例に示すとおりです。

```
<?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用さ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、前の例のものです。

DTDID エレメント

XML エクステンダーに用意されている DTD は、DTD_REF 表の中に入っています。各 DTD は、DAD ファイルの DTDID タグの中に指定されている固有の ID によってそれぞれ識別されます。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用される DTD を示します。 XML コレクションの場合、このエレメントが必要とされるのは、入出力 XML 文書の妥当性検査を実行する場合だけです。 XML 列の場合、このエレメントが必要とされるのは、入力 XML 文書の妥当性検査を実行する場合だけです。 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 を使用して 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。 0 個以上の表で構成されます。

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>
```

table エレメントには、必須の属性が 1 個と、暗黙の属性が 2 個あります。

name 属性

サイド表の名前を指定します。

key 属性

表のただ 1 つの基本キーです。

orderBy 属性

XML 文書を生成する際、複数出現するエレメント・テキストまたは属性値のシーケンス順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

入力 XML 文書の CDATA ノードの属性を、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 エクステンダーが使用する SQL ステートメントを指定します。そのステートメントは XML コレクション表から XML データを選択し、そのデータを使用してコレクション内に XML 文書を生成します。このエレメントの値は、有効な SQL ステートメントでなければなりません。これは合成の場合のみ使用され、指定できる SQL_stmt の数は 1 つのみです。

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 の値は固定です。

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、結果として出力される文書の DOCTYPE を指定するために使用されます。明示的な値を定義してください。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。これは一度のみ使用できます。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

XML エレメントの内容または XML 属性の値を保管する、あるいはそこから取り出す DB2 UDB 表を定義します。 rdb_node は element_node、text_node、および attribute_node の子エレメントであり、その子エレメントは次のとおりです。

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。

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。

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、 0 個以上の attribute_nodes と子 element_nodes、および 0 個または 1 個の text_nodes を含めることができます。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

(オプション) XML データに対して表、列、および条件を指定します。エレメントの RDB_node を定義することが必要なのは、 RDB_node マッピングの場合だけです。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オプションになります。

child nodes

オプション: element_node には、以下の子ノードも指定できます。

element_node

現在の XML エレメントの子エレメントを表します。

attribute_node

現在の XML エレメントの属性を表します。

text_node

現在の XML エレメントの CDATA テキストを表します。

attribute_node

XML 属性を表します。これは、XML 属性とリレーショナル表の列データの間のマッピングを定義するノードです。

構文:

attribute_node には name 属性の定義が必要であり、あわせて column または RDB_node 子エレメントのいずれかが必要です。attribute_node には以下の属性があります。

name 属性の名前。

attribute_node には以下の子エレメントがあります。

column
SQL マッピングに使用されます。この列は、SQL_stmt の SELECT 文節内で指定されていなければなりません。

RDB_node
RDB_node マッピングに使用されます。このノードは、この属性とリレーショナル表内の列データ間のマッピングを定義します。表および列の指定は必須です。この条件はオプションです。

text_node
XML エレメントのテキストの内容を表します。これは、XML エレメントの内容とリレーショナル表の列データ間のマッピングを定義するノードです。

構文: これは、以下の column または RDB_node 子エレメントによって定義する必要があります。

column
SQL マッピングに必要。この場合、SQL_stmt の SELECT 文節内にこの列が指定されていなければなりません。

RDB_node
RDB_node マッピングに必要。このノードは、このテキスト内容とリレーショナル表内の列データ間のマッピングを定義します。 table および column の指定は必須です。この条件はオプションです。

関連概念:

• 175 ページの『XML コレクションのための DAD ファイル』

関連タスク:

• 184 ページの『DAD ファイル内の値を動的にオーバーライドする』

DAD ファイル内の値を動的にオーバーライドする

手順:

動的な照会のためには、 2 つのオプション・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* パラメーターの値は次のとおりです。

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。

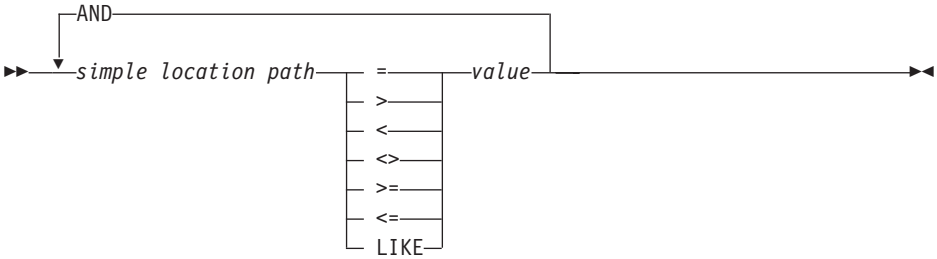
XML_OVERRIDE
DAD ファイル内の条件を XPath に基づく条件によってオーバーライドすることを指定します。

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 ステートメントとして SQL_OVERRIDE を使用する場合、 SQL マッピング体系を DAD ファイルで使用しなければなりません。入力 SQL ステートメントは、DAD ファイル内の <SQL_stmt> エレメントで指定された SQL ステートメントをオーバーライドします。
- XML_OVERRIDE を指定した場合、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、 DAD ファイル内で RDB_node マッピング体系を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。式は、次の構文を使用します。



この構文には以下の構成要素が含まれます。

simple location path
XPath によって定義された構文を使用して、単純ロケーション・パスを指定します。

演算子
構文図で示されている SQL 演算子には、式のその他の部分から演算子を分け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 演算子の前後には、必ずスペースを入れる必要があります。

value
単一引用符で囲まれた数値またはストリングです。

AND
AND は同じロケーション・パスにある論理演算子として扱われます。override ストリングで単純ロケーション・パスが 2 回以上指定されている場合、その単純ロケーション・パスのすべての述部が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。

次の例では、動的なオーバーライドを表すために SQL_OVERRIDE と XML_OVERRIDE が使用されています。

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 より高く変更し、日付を 1998-12-01 より以降にして、<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    collectionName[32]; /* name of an XML collection */
char    result_tab[32];     /* name of the result table */
char    result_colname[32]; /* name of the result column */
char    valid_colname[32];  /* name of the valid column, will set to NULL*/
char    override[512];      /* override */
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   collectionName_ind;
short   rtab_ind;
short   rcol_ind;
short   vcol_ind;
short   ovtype_ind;
short   ov_ind;
short   maxrow_ind;+
short   numrow_ind;
short   returnCode_ind;
short   returnMsg_ind;
EXEC SQL END DECLARE SECTION;
float price_value;

/* create table */
EXEC SQL CREATE TABLE xml_order_tab (xmlorder XMLVarchar);

/* initial host variable and indicators */
```

```
strcpy(collection, "sales_ord");
strcpy(result_tab,"xml_order_tab");
strcpy(result_col,"xmlorder");
valid_colname[0] = '¥0';

/* get the price_value from some place, such as from data */
price_value = 1000.00        /* for example */

/* specify the override */
sprintf(override,
  " SELECT o.order_key, customer, p.part_key,
    quatity, price, tax, ship_id, date, mode
  FROM order_tab o, part_tab p,
     table(select db2xml.generate_unique()
     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);

overrideType = SQL_OVERRIDE;
max_row = 0;
num_row = 0;
returnCode = 0;
msg_txt[0] = '¥0';
collectionName_ind = 0;
rtab_ind = 0;
rcol_ind = 0;
vcol_ind = -1;
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.DXXRETRIEVEXML" (:collectionName:collectionName_ind,
                                       :result_tab:rtab_ind,
                                       :result_colname:rcol_ind,
                                       :valid_colname:vcol_ind,
                                       :overrideType:ovtype_ind,:override:ov_ind,
                                       :max_row:maxrow_ind,:num_row:numrow_ind,
                                       :returnCode:returnCode_ind,
            :returnMsg:returnMsg_ind);

if (SQLCODE < 0) {
    EXEC SQL ROLLBACK;
  else
    EXEC SQL COMMIT;
}
```

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 エクステンダーは単純ロケーション・パスをマップされた 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    collectionName[32]; /* name of an XML collection */
char    result_tab[32];     /* name of the result table */
char    result_colname[32]; /* name of the result column */
char    valid_colname[32];  /* name of the valid column, will set to NULL*/
```

```
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   collectionName_ind;
short   rtab_ind;
short   rcol_ind;
short   vcol_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);

/* initial host variable and indicators */
strcpy(collection, "sales_ord");
strcpy(result_tab,"xml_order_tab");
strcpy(result_col,"xmlorder");
valid_colname[0] = '¥0';
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';
collectionName_ind = 0;
rtab_ind = 0;
rcol_ind = 0;
vcol_ind = -1;
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"
          (:collectionName:collectionName_ind,
           :result_tab:rtab_ind,
           :result_colname:rcol_ind,
           :valid_colname:vcol_ind,
            :overrideType:ovtype_ind,:override:ov_ind,
           :max_row:maxrow_ind,:num_row:numrow_ind,
           :returnCode:returnCode_ind,:returnMsg:returnMsg_ind);

if (SQLCODE < 0) {
    EXEC SQL ROLLBACK;
  else
    EXEC SQL COMMIT;
}
```

複数オーバーライド

XML エクステンダーでは、同じパスに対する複数のオーバーライドがサポートされています。 RDB ノードに対して指定されているすべてのオーバーライドが受け入れられます。

検索の集合条件をさらに詳細に指定するため、同じ場所に対して複数の XML オーバーライドを指定できます。次の例の XML 文書は、test.dad ファイルを使用することにより、 2 つの表から構成されます。

表 48. Department (部門) 表

部門番号	部門名
10	Engineering
20	Operations
30	Marketing

表 49. Employee (従業員) 表

従業員番号	部門番号	給与
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

下記の DAD ファイル test.dad には、変数 *deptno* を値 10 と比較する条件が含まれています。 10 より大きく 30 より小さいという検索条件に拡張するには、ここに示されている条件をオーバーライドする必要があります。 dXXGenXML を呼び出す際に、次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>
```

オーバーライドなしで XML 文書を合成するには、 tests2x mydb test.dad result_tab と入力するか、またはオーバーライドを設定せずに dxxGenXML を起動します。その場合に生成される文書は、次のようなものになります。

```
<?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>
<Salray>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、前述のように dxxGenXML を起動するか、または次のように条件を指定して 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:

- 175 ページの『XML コレクションのための DAD ファイル』
- 191 ページの『DAD チェッカー』

関連タスク:

- 173 ページの『XML 列のための DAD ファイルの作成』
- 191 ページの『DAD チェッカーの使用』

関連参照:

- 179 ページの『DAD ファイル用の DTD』

DAD チェッカー

DAD チェッカーは、 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検査するために使用することができます。各 DAD ファイルの中で、表と XML 文書の構造との間の関係を指定するマッピング体系が指定されています。

文書タイプ記述 (DTD) が XML 文書の構文の妥当性検査に使用されるように、DAD チェッカーは、 DAD ファイルが意味的に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 チェッカーの使用は、ファイルを XML エクステンダーに送って処理する際に発生するエラーの数を最小限に抑えるうえで役立ちます。 DAD チェッカーは、コマンド行から呼び出される Java™ アプリケーションです。起動された DAD チェッカーは、エラー、警告、および成功の標識を含む 2 つの出力ファイルのセットを作成します。これらの 2 つのファイルは同等です。1 つは、ユーザーがエラーまたは警告の有無を検査するために使用する非暗号化テキスト・ファイルであり、もう 1 つは、 DAD チェッカー・アプリケーションの結果を別のアプリケーションに伝える `errorsOutput.xml` という XML ファイルです。出力テキスト・ファイルの名前はユーザーが定義します。名前を指定しない場合には、標準出力が使用されます。

関連概念:

- 175 ページの『XML コレクションのための DAD ファイル』

関連タスク:

- 184 ページの『DAD ファイル内の値を動的にオーバーライドする』
- 173 ページの『XML 列のための DAD ファイルの作成』
- 191 ページの『DAD チェッカーの使用』

DAD チェッカーの使用

前提条件:

システムに JRE または JDK バージョン 1.3.1 以降がインストールされている必要があります。

手順:

DAD チェッカーを使用するには、次の手順を実行してください。

1. DADChecker.zip ファイルをダウンロードし、すべてのファイルを任意のディレクトリーに解凍します。
2. コマンド行で、DAD チェッカーがインストールされているディレクトリー内の /bin サブディレクトリーに移動します。
3. /bin ディレクトリーにある setCP.bat ファイルを実行することにより、classpath を設定します。
4. 次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
   検査対象のファイルが DAD ファイルであることを表します。これはデフォルト・オプションです。

   -xml
   検査対象のファイルが DAD ファイルではなく XML 文書であることを表します。大きい XML 文書の場合、Java 仮想マシンがメモリー不足になり、java.lang.OutOfMemoryError 例外が生成されることがあります。このような場合、 -Xmx オプションを使用すると Java 仮想マシンに割り振るメモリー量を増やすことができます。詳細については、JDK の資料を参照してください。

   -all
   誤りのあるタグの出現をすべて出力することを表しています。

   -tag
   name 属性値が *tagname* になっている重複タグだけが表示されることを表しています。 XML 文書の場合、name 属性値が tagname になっている重複タグのみが表示されます。

   -out
   outputFile は出力テキストのファイル名を指定しています。これを指定しない場合には、標準出力が使用されます。 2 番目の出力ファイル errorsOutput.xml も、同じディレクトリーに DAD ファイルとして作成されます。このファイルは常に生成され、パーサーの警告およびエラーを除き、出力テキスト・ファイルと同じ情報を XML 形式で含んでいます。

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 チェッカーのためのサンプル・ファイル:

以下のサンプル・ファイルが samples ディレクトリーに入っています。

bad_dad.dad
起こりうるすべての意味エラーを示すサンプル DAD ファイル。

bad_dad.chk
bad_dad.dad に関して DAD チェッカーで生成された出力テキスト・ファイル。

bad_dad.chk
bad_dad.dad に関して DAD チェッカーで生成された出力テキスト・ファイル。

errorsOutput.xml
bad_dad.dad に関して DAD チェッカーで生成された出力 XML ファイル。

dup.xsl
errorsOutput.xml ファイルを HTML ファイルに変換して、重複タグのみを表示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のみを表示するように生成された HTML ファイル。

出力テキスト・ファイル内のエラーおよび警告:

エラーおよび警告はタグの出現によって示されます。次の場合、2 つのタグは、同じタグの出現と見なされます。

- name 属性の値が同じである。
- 上位タグの数が同じである。
- 対応する上位タグの name 属性の値が同じである。

出現したタグが同じであっても、子タグが異なっている可能性があります。

DAD の意味規則に従わないタグの出現は、出力テキスト・ファイル内で次のように示されます。

- すべての上位タグとそれらの属性は、連続して表示されます。
- エラーのあるタグは、XML ツリーにおけるその深さを示す数値の後に表示されます。タグ名の後には、その DAD ファイルでこのタグが出現しているすべての行の番号のリストが示されます。 *-all* コマンド行オプションを使用すると、それぞれのエラーの出現を別個に表示することができます。
- 最初に出現したタグの直接の子タグが表示されます。データ・マッピングを指定する子タグの場合、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、それぞれのエラーの出現を別個に表示することができます。

DAD チェッカーのエラー報告書のサンプル:

この例では、name 属性の値が "Password" になっている element_node タグにエラーがあります。 DAD ファイルでは、このタグが 2 回出現しています (行 49 と行 75)。エラーのあるタグは、タグの深さ標識 (この例では 4) を見つけることにより、上位タグおよび子タグのリストから特定することができます。上位タグおよび子タグのリストは、エラーが発生したコンテキストを確定するうえで役立ちます。

```
<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 オプションを使用した場合、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 つの出現で上位タグと name 属性値が同じになっていますが、子エレメントが異なっています。

DAD チェッカーによって行われる検査

DAD チェッカーを起動すると、次のメッセージが出されます。

```
Checking DAD document: file_path
```

ここで、*file_path* は、妥当性検査の対象となる DAD ファイルへのパスです。

DAD チェッカーは以下の妥当性検査を行います。

1. 整形式かどうかの検査および DTD 妥当性検査。
2. 重複した <attribute_node> およびリーフ <element_node> の検出 (RDB_node マッピング)。
3. 欠落した type 属性の検出。
4. 欠落した表宣言の検出。
5. 欠落した <text_node> または <attribute_node> の検出。
6. <attribute_node> および <element_node> のマッピング順序の検査。
7. name 属性値が同じになっているタグのデータ・マッピング整合性検査。
8. 子がマップされている親 <element_node> に関する multi_occurrence 属性値検査 (RDB_node マッピング)。
9. 属性とエレメントの潜在的な命名の競合の検査 (XML 文書)。

これらの妥当性検査については、以下のいくつかのセクションで説明します。

整形式かどうかの検査および DTD 妥当性検査

DAD ファイルは、 dxxsamples¥dtd¥dad.dtd にある DAD DTD と突き合わせて妥当性検査する必要があります。 DAD ファイルが整形式でなかったり、DTD が見つからなかったりした場合には、致命的エラーが発生して DAD チェッカーが終了し、出力テキスト・ファイルにエラーの発生が表示されます。例えば、

```
org.xml.sax.SAXException: Stopping after fatal error,
line 1, col 22. The XML declaration must end with "?>".
```

妥当性検査に関するエラーおよび警告が発生した場合にも出力テキスト・ファイルで報告されますが、これによって DAD チェッカーが終了することはありません。次の例は、 DAD ファイルの構文解析中に発生する可能性のある 2 つの妥当性検査エラーを示す出力テキスト・ファイルの一部です。

```
 ** 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)".
```

重複した <attribute_node> およびリーフ <element_node> の検出 (RDB_node マッピング)

この検査は、RDB_node マッピングを使用する DAD ファイルにのみ関係します。

2 つのエレメントが重複しているとみなされるのは、 2 つ以上の <attribute_node> または <element_node> タグの name 属性の値が同じであり、かつその祖先が同じ場合です。

2 つ以上のタグに対応する祖先タグの name 属性の値が同じである場合、それらのタグは同じ祖先であるとみなされます。

リーフ <element_node> は、 XML 文書ツリー内に子を持たないタグをマップするために使用される element_node です。したがって、リーフ <element_node> タグには、直接の子のうちの 1 つとして、テキスト・ノード・タグ 1 つが含まれていなければなりません。それ以外の <element_node> タグには、直接の子としてテキスト・ノード・タグを含めることはできません。

この競合は、複数のリーフ <element_ node> タグ間、複数の <attribute_node> タグ間、またはリーフ <element_ node> タグと <attribute_ node> タグの間で発生する可能性があります。

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、<element_node name = ″C″> は、 2 つの異なるパス ¥A1¥B¥C および ¥A2¥B¥C を介してマップされるため、重複しています。 <element_node name=″B″> は、非リーフ <element_node> であるため、重複しているとは見なされません。

例 2

この例は <attribute node> の競合を示しています。

```
<element_node name = "A1">
    <attribute_node name = "B">
                  ....
<element_node name = "A2">
    <attribute_node name = "B">
 /element_node>               ....
<
```

この例で、<attribute_node name = ″B″> は、 2 つの異なるパス ¥A1¥B および ¥A2¥B を介してマップされるため、重複しています。

例 3

この例は、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″> はリーフ <element_node> ではないため、 <element_node name = ″A″> と <attribute_node name = ″A″> は競合していません。

競合が発生した場合、XML 文書の DTD を修正して、競合を解消しなければなりません。 XML 文書および DAD ファイルも修正して、DTD の変更を反映させる必要があります。

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">
-----------------------------------------------------------
```

エラーのあるタグは命名競合ごとにグループ化されます。これらのグループは線によって区切られ、タグは短い線によって区切られます。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 ファイルに重複タグがない場合には、出力テキスト・ファイルに次のメッセージが書き込まれます。

```
No duplicated tags were found.
```

欠落した type 属性の検出

コレクションを使用可能にしたり分解を行ったりするために DAD ファイルを使用する場合には、それぞれの <column> タグごとに type 属性を指定しなければなりません。例えば、

```
<column name="email" type="varchar(20)">
```

enable_collection コマンドは、表が存在しない場合には、列タイプの指定値を使用してコレクション内に表を作成します。表が存在する場合には、DAD 内で指定されるタイプは、データベース内の実際の列タイプと一致していなければなりません。

例:

次の例は、type 属性が指定されていない <column> タグを示す出力テキスト・ファイルの一部です。

```
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
```

type 属性が欠落していない場合には、出力テキスト・ファイルに次のメッセージが書き込まれます。

```
No type attributes are missing for <column> tags.
```

欠落した表宣言の検出

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> タグを含む、表宣言を囲んでいなければなりません。このタグは、最初の <element_node> タグによって囲まれていなければなりません。それ以降のすべての <RDB_node> タグは、<text_node> タグによって囲まれていなければなりません。

最初に検出された <RDB_node> タグに <column> タグが含まれている場合にも、出力ファイルにエラーが追加されます。このエラーは、表宣言が欠落しているか、あるいは表宣言に誤って <column> タグが含まれていることのいずれかを示しています。

欠落した <text_node> または <attribute_node> の検出

それぞれの <RDB_node> タグは、表宣言に使用される最初のものを除き、<attribute_node> または <text_node> タグに囲まれていなければなりません。

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、 <text_node> または <attribute_node> タグが欠落していることを示す出力テキスト・ファイルの一部です。

```
<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> および <element_node> のマッピング順序の検査

この検査は、フィックスパック 3 およびそれ以前で必要となります。なんらかの <element_node> タグが表にマップされる前に、 <attribute_node> タグをその表にマップする必要があります。

例:

次の例は、表にマップする必要のあるタグを表しています。

```
<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、 <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> のマッピングよりも先に行われなければなりません。

name 属性値が同じになっているタグのデータ・マッピング整合性検査

DAD ファイル内にある、マップされ、個別の name 属性値で識別される、すべての <element_node> タグおよびすべての <attribute_node> タグは、 1 回のみマップするようにしてください。 <element_node> タグまたは <attribute_node> タグの複数の出現が別の列にマップされる場合には、それらの name 属性に異なる値を割り当てる必要があります。

例:

例 1: この例で、 2 番目に出現している <element_node name=″type″> タグのマッピングは、最初に出現したものと異なっています。この検査の結果では、重複した <attribute_node> タグと重複したリーフ <element_node> タグは表示されません。

```
<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 番目のマッピングで使用する新規のエレメントを作成することにより、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: この例は、名前と上位タグは同じだがマッピングが異なっている <element_node> タグを示す、出力テキスト・ファイルの一部です。

```
<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、行 127 における <element_node name=″PostalCode″> タグの出現は「postal_code」列にマップされ、行 135 および 143 における同じタグの他の 2 つの出現は、「postal_code2」列にマップされます。

子がマップされている親 <element_node> に関する multi_occurrence 属性値検査

この検査は、RDB_node マッピングを使用する DAD ファイルにのみ関係します。

multi_occurrence 属性のデフォルト値は NO です。直接の子として <attribute_node> タグを持つ各 <element_node> タグ、または次のいずれかまたは両方の基準に該当する複数の <element_node> タグについては、 multi_occurrence 属性の値として YES を割り当てる必要があります。

- <element_node> がマップされている (<text_node> がその直接の子となっている)。
- <element_node> の直接の子として、少なくとも 1 つ <attribute_node> がある。

例:

例 1: 次の例で、payment-request-id と amount は DB2 UDB 表にマップされます。sender には、直接の子として <attribute_node> があります。
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 チェッカーは、 multi_occurrence 属性が NO に設定されているすべての <element_node> タグを表示します。

例 2: 次の例は、 multi_occurrence 属性を YES に設定する必要のある <element_node> タグを示す出力テキスト・ファイルの一部です。

```
<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競合

XML 文書では、同じ名前のエレメントが、異なるコンテキスト (例えば、上位エレメントが異なっているなど) で現れることがあります。属性とエレメントは、同じ名前になっていることがあります。これらの命名競合は、DAD ファイル内で重複タグを生成するため、現在のところ XML エクステンダーでは解決できません。したがって、マップされる属性およびエレメントのうち、上位の属性やエレメントが同じになっているものは、それぞれ固有の名前にしなければなりません。

DAD チェッカーを使用すると、XML 文書で命名競合が生じているかどうかを検査することができます。複数の競合したエレメントまたは属性をマップする必要がある場合、文書および DTD の名前を変更する必要があります。

DAD ファイルを作成する前に XML 文書を検査することをお勧めします。 DAD チェッカーは、XML 文書を DTD と突き合わせての妥当性検査は行いません。

例:

次の例は、命名競合が起こっている XML 文書の一部です。

```
<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>
```

DAD 内で 2 つの <element_node name = ″C″> タグと <attribute_node name = ″C″> タグが重複しています。

第 10 章 XML エクステンダーのストアード・プロシージャー

XML エクステンダーのストアード・プロシージャー

XML エクステンダーには、 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 (*プロシージャー* とも呼ばれる) が提供されています。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の詳細は、「*DB2 UDB for iSeries SQL Programming*」のストアード・プロシージャーに関するセクションを参照してください。

ストアード・プロシージャーは、DB2XML を使用します。これは XML エクステンダーのスキーマ名です。

XML エクステンダーのストアード・プロシージャーには、次の 3 つのタイプがあります。

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・プロシージャーを呼び出す前に、すべての環境をセットアップする必要があります。ストアード・プロシージャーを呼び出すサンプル・プログラムは、`DXXSAMPLES/QCSRC` にあります。

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 エクステンダーの外部ヘッダー・ファイルを組み込んでください。ヘッダー・ファイルは、dxxsamples¥include ディレクトリーにあります。ヘッダー・ファイルは次のとおりです。

dxx.h XML エクステンダーの定義する定数およびデータのタイプ

dxxrc.h XML エクステンダーの戻りコード

これらのヘッダー・ファイルを組み込む構文は次のとおりです。

```
#include "dxx.h"
#include "dxxrc.h"
```

組み込みファイルのパスが、 MAKE ファイル内でコンパイル・オプションとともに指定されていることを確認してください。

XML エクステンダーの管理ストアード・プロシージャー

これらのストアード・プロシージャーは、 XML 列やコレクションを使用可能または使用不可にする場合などの管理作業に使用します。これらを呼び出すには、 XML エクステンダー管理ウィザードおよび管理コマンド **dxxadm** を使用します。

- dxxEnableDB()
- 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 エクステンダーは以下のオブジェクトを作成します。

- XML エクステンダー・ ユーザー定義タイプ (UDT)
- XML エクステンダー・ ユーザー定義関数 (UDF)
- XML エクステンダー DTD 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 エクステンダー使用状況表 (XML_USAGE)。これは XML に使用できるそれぞれの列およびコレクションに関する一般情報を保管します。

構文:

```
QDB2XML.dxxEnableDB(char(dbName) dbName,       /* input */
            long           returnCode,   /* output */
            varchar(1024)  returnMsg)    /* output */
```

パラメーター:

表 50. 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:

- 227 ページの『第 11 章 XML エクステンダーの管理サポート表』

関連タスク:

- 57 ページの『XML 用のデータベースの使用可能化』

- 210 ページの『XML エクステンダーの合成ストアード・プロシージャーの呼び出し』

関連参照:

- 204 ページの『XML エクステンダーの管理ストアード・プロシージャー』
- xi ページの『構文図の読み方』
- 269 ページの『付録 C. XML エクステンダーの制限』

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 エクステンダー がデータベースを使用不可にすると、以下のオブジェクトが除去されます。

- XML エクステンダー・ ユーザー定義タイプ (UDT)
- XML エクステンダー・ ユーザー定義関数 (UDF)
- XML エクステンダー DTD 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 エクステンダー使用状況表 (XML_USAGE)。これは XML に使用できるそれぞれの列およびコレクションに関する一般情報を保管します。

重要: データベースを使用不可にする前に、すべての XML 列を使用不可にする必要があります。 XML エクステンダーは、 XML に使用できる列またはコレクションを含んでいるデータベースを使用不可にできません。

構文:

```
QDB2XML.dxxDisableDB(char(dbName)     dbName,       /* input */
                     long          returnCode,   /* output */
                     varchar(1024) returnMsg)    /* output */
```

パラメーター:

表 51. 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:

- 227 ページの『第 11 章 XML エクステンダーの管理サポート表』

関連タスク:

- 210 ページの『XML エクステンダーの合成ストアード・プロシージャーの呼び出し』

関連参照:

- 204 ページの『XML エクステンダーの管理ストアード・プロシージャー』

- xi ページの『構文図の読み方』
- 269 ページの『付録 C. XML エクステンダーの制限』

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 エクステンダーは、列を使用可能にする際に以下のタスクを行います。

- XML 表に基本キーがあるかどうかを判別します。基本キーがない場合は、XML エクステンダーが 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 表内の各行の固有な ID を含む列があります。この列はユーザーが指定する root_id であるか、または XML エクステンダーで指定された DXXROOT_ID のいずれかです。
- 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(defaultView) defaultView, /* input */
                char(rootID) rootID,      /* input */
                long           returnCode,  /* output */
                varchar(1024)  returnMsg)   /* output */
```

パラメーター:

表 52. 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
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のビューの名前。	IN
rootID	アプリケーション表における、サイド表のルート ID として使用されるただ 1 つの基本キーの名前。	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』

関連タスク:

- 210 ページの『XML エクステンダーの合成ストアード・プロシージャーの呼び出し』

関連参照:

- 204 ページの『XML エクステンダーの管理ストアード・プロシージャー』
- xi ページの『構文図の読み方』
- 269 ページの『付録 C. XML エクステンダーの制限』

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 53. 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

関連参照:

- 269 ページの『付録 C. XML エクステンダーの制限』

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 */
                      long           returnCode,  /* output */
                    varchar(1024)  returnMsg)   /* output */
```

パラメーター:

表 54. dxxEnableCollection() パラメーター

パラメーター	説明	IN/OUT パラメーター
dbName	データベース名。	IN
colName	XML コレクションの名前。	IN
DAD	DAD ファイルを含む CLOB。	IN
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:

- 95 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:

- 210 ページの『XML エクステンダーの合成ストアード・プロシージャーの呼び出し』

関連参照:

- 204 ページの『XML エクステンダーの管理ストアード・プロシージャー』
- xi ページの『構文図の読み方』
- 269 ページの『付録 C. XML エクステンダーの制限』

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 55. 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

表 55. *dxxDisableCollection()* 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

関連参照:

- 269 ページの『付録 C. XML エクステンダーの制限』

XML エクステンダーの合成ストアード・プロシージャー

合成ストアード・プロシージャー dxxGenXML()、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用して 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は、以下のようにパフォーマンスが向上しています。

- iSeries および zSeries オペレーティング・システムでは、 override パラメーターの長さが 16KB に増えました。iSeries および zSeries オペレーティング・システムでは、 override パラメーターの長さが 16KB に増えました。
- 中間結果表のための要件が除去されました。
- これらのストアード・プロシージャーを使用することには、以下の利点があります。
  - 結果表を作成する必要がないため、命令パスの長さが短縮されます。
  - プログラミングが単純化されます。
- 複数の文書を作成したい場合には、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てください。
- XML 列用のユーザー定義関数のパフォーマンスが向上しました。
- DB2 UDB XML エクステンダーのユーザー定義関数が、処理中の小さな (512KB) XML 文書をメモリーに保持するようになりました。これにより、入出力活動が削減され、また、一時ファイルに使用されるディスクの競合が減少します。
- DB2 UDB XML エクステンダーの (表以外の) スカラー・ユーザー定義関数の定義が変更され、並列実行できるようになりました。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実行パフォーマンスが大幅に向上しました。スカラー UDF を並列実行できるようにするには、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ある場合には、すべての列を使用不可にし、移行スクリプトを実行したうえで、それらの列を再び使用可能にする必要があります。

XML エクステンダーの合成ストアード・プロシージャーの呼び出し

XML エクステンダーは、ストアード・プロシージャーの名前を大文字と小文字のどちらで指定しても、単一クライアント・アプリケーションからさまざまなオペレーティング・システムで使用できます。この方法でストアード・プロシージャーを呼び出すには、 result_colname および valid_colname バージョンの合成ストアード・プロシージャーを使用してください。この方法を使用した場合、以下の利点があります。

- 結果表に多くの列を含めることができるため、これらのストアード・プロシージャーはすべての DB2 Universal Database 環境で使用することができます。result_colname および valid_colname をサポートしないバージョンのストアード・プロシージャーでは、結果表には列が 1 つのみ含まれている必要があります。
- 宣言済みの一時表を結果表として使用することができます。一時表は、「session」に設定されたスキーマによって識別されます。宣言済みの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るようになります。

DB2 XML エクステンダーのストアード・プロシージャーを呼び出して、各種のプラットフォームにあるストアード・プロシージャーに一貫してアクセスできるようにするために、英大文字を使用してください。

手順:

次の構文を使用して XML エクステンダーを呼び出します。

```
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。ホスト変数にはヌル標識を指定できます。

DXXSAMPLES/QCSRC ソース・ファイル内の、ストアード・プロシージャーの呼び出し例を参照してください。 DXXSAMPLES/QCSRC ソース・ディレクトリーには、組み込み SQL を使用して XML コレクションのストアード・プロシージャーを呼び出すための、 SQX コード・ファイルが用意されています。

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、試行プロセス中にアプリケーションが結果を待つ時間を短縮できます。このストアード・プロシージャーは、表内の実際の行番号とエラー情報 (エラー・コードおよびエラー・メッセージ) を戻します。

動的照会をサポートするために、 dxxGenXML() は入力パラメーター *override* を取ります。入力 *overrideType* に基づいて、アプリケーションは DAD ファイル内の SQL マッピングの SQL_stmt、または RDB_node 内の RDB_node マッピング条件をオーバーライドできます。入力パラメーター *overrideType* を使用して、 *override* のタイプを明示します。

構文:

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

          char(resultColumn) result_column,
         char(validColumn) valid_column,
    integer        overrideType  /* input */
   varchar(varchar_value) override,

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

この varchar_value は、Windows および UNIX の場合には 32672 であり、 iSeries および z/OS の場合には 16366 です。

パラメーター:

表 56. 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 つのみ含めます。	IN
result_column	合成された XML 文書が保管される結果表内の列の名前。	IN
valid_column	XML 文書を文書タイプ定義 (DTD) と突き合わせて妥当性検査する際に、その文書が有効であることを示す列の名前。	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

表 56. *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
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	合成された XML 文書を含む CLOB。	OUT
valid	valid は次のように設定されます。 • VALIDATION=YES の場合、妥当性検査が正常に行われたときには valid=1、妥当性検査が正常に行われなかったときには valid=0。 • VALIDATION=NO の場合には 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:

以下の例では、XML_ORDER_TAB という名前の結果表が作成されることと、XMLVARCHAR タイプの 1 つの列が表に含まれることを想定します。完全な作業用のサンプルが DXXSAMPLES/QCSRC(GENX) にあります。

```
#include "dxx.h"
#include "dxxrc.h"

EXEC SQL INCLUDE SQLCA;
EXEC SQL BEGIN DECLARE SECTION;
SQL TYPE is CLOB(100K) dad;    /* DAD */

char    result_tab[32];        /* name of the result table */
char    result_colname[32];    /* name of the result column */
char    valid_colname[32];     /* name of the valid column, will set to NULL */
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   rcol_ind;
short   vcol_ind;
short   ovtype_ind;
short   ov_ind;
short   maxrow_ind;
short   numrow_ind;
short   returnCode_ind;
short   returnMsg_ind;
EXEC SQL END DECLARE SECTION;

FILE   *file_handle;
long    file_length=0;

/* create table */
EXEC SQL CREATE TABLE xml_order_tab (xmlorder XMLVarchar);

/* initialize the DAD CLOB object. */
file_handle = fopen( "/dxx/dad/getstart_xcollection.dad", "r" );
if ( file_handle != NULL ) {  file_length = fread ((void *) &dad.data
, 1, FILE_SIZE, file_handle);
  if (file_length == 0) {
    printf ("Error reading dad file
             /dxx/dad/getstart_xcollection.dad¥n");
    rc = -1;
    goto exit;
 } else
    dad.length = file_length;
}
else {
  printf("Error opening dad file  ¥n", );
  rc = -1;
  goto exit;
}
/* initialize host variable and indicators */
strcpy(result_tab,"xml_order_tab");
strcpy(result_colname, "xmlorder")
valid_colname = '¥0';
override[0] = '¥0';
overrideType = NO_OVERRIDE;
max_row = 500;
num_row = 0;
returnCode = 0;
msg_txt[0] = '¥0';
dad_ind = 0;
rtab_ind = 0;
rcol_ind = 0;
vcol_ind = -1;
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,
                          :result_colname:rcol_ind,
                          :valid_colname:vcol_ind,
                          :overrideType:ovtype_ind,:override:ov_ind,
```

```
                          :max_row:maxrow_ind,:num_row:numrow_ind,
                          :returnCode:returnCode_ind,
                          :returnMsg:returnMsg_ind);

if (SQLCODE < 0) {
    EXEC SQL ROLLBACK;
  else
    EXEC SQL COMMIT;
}

exit:
  return rc;
```

関連タスク:

- 65 ページの『SQL マッピングを使用した XML 文書の合成』
- 69 ページの『RDB_node マッピングを使用した XML コレクションの合成』
- 210 ページの『XML エクステンダーの合成ストアード・プロシージャーの呼び出し』

関連参照:

- 209 ページの『XML エクステンダーの合成ストアード・プロシージャー』
- xi 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:

分解された XML 文書は、ストアード・プロシージャー dxxRetrieveXML() を使用して取り出すことができます。 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(resultColumn) result_column,
       char(validColumn) valid_column,
       integer        overrideType,     /* input */
      varchar(varchar_value) override,

                          integer        maxRows,          /* input */
```

```
integer        numRows,          /* output */
long           returnCode,       /* output */
varchar(1024)  returnMsg)        /* output */
```

この *varchar_value* は、Windows および UNIX の場合には 32672 であり、 iSeries および z/OS の場合には 16366 です。

パラメーター:

表 57. 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 つのみ含めます。	IN
result_column	合成された XML 文書が保管される結果表内の列の名前。	IN
valid_column	XML 文書を文書タイプ定義 (DTD) と突き合わせて妥当性検査する際に、その文書が有効であることを示す列の名前。	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

表 57. *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
numRows	結果表に実際に生成された行の数。	OUT
returnCode	ストアード・プロシージャーからの戻りコード。	OUT
returnMsg	エラー発生時に戻されるメッセージ・テキスト。	OUT

例:

以下は dxxRetrieveXML() の呼び出しの例です。この例では XML_ORDER_TAB という名前の結果表が作成され、 XMLVARCHAR タイプの 1 つの列がその表に含まれます。完全な作業用サンプルが DXXSAMPLES/QCSRC(RTRX) にあります。

```
#include "dxx.h"
#include "dxxrc.h"

EXEC SQL INCLUDE SQLCA;
EXEC SQL BEGIN DECLARE SECTION;
char    collectionName[32];    /* name of an XML collection */
char    result_tab[32];        /* name of the result table */
char    result_colname[32];    /* name of the result column */
char    valid_colname[32];     /* name of the valid column, will set to NULL*/
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   collectionName_ind;
short   rtab_ind;
short   rcol_ind;
short   vcol_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);

/* initial host variable and indicators */
strcpy(collection, "sales_ord");
strcpy(result_tab,"xml_order_tab");
strcpy(result_col,"xmlorder");
valid_colname[0] = '¥0';
override[0] = '¥0';
overrideType = NO_OVERRIDE;
max_row = 500;
num_row = 0;
returnCode = 0;
msg_txt[0] = '¥0';
collectionName_ind = 0;
rtab_ind = 0;
rcol_ind = 0;
vcol_ind = -1;
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.DXXRETRIEVE"
                 (:collectionName:collectionName_ind,
                 :result_tab:rtab_ind,
                 :result_colname:rcol_ind,
                 :valid_colname:vcol_ind,
                 :overrideType:ovtype_ind,:override:ov_ind,
                 :max_row:maxrow_ind,:num_row:numrow_ind,
                 :returnCode:returnCode_ind,:returnMsg:returnMsg_ind);

if (SQLCODE < 0) {
    EXEC SQL ROLLBACK;
  else
    EXEC SQL COMMIT;
}
```

関連タスク:

- 65 ページの『SQL マッピングを使用した XML 文書の合成』
- 69 ページの『RDB_node マッピングを使用した XML コレクションの合成』
- 210 ページの『XML エクステンダーの合成ストアード・プロシージャーの呼び出し』

関連参照:

- 209 ページの『XML エクステンダーの合成ストアード・プロシージャー』
- xi ページの『構文図の読み方』
- 269 ページの『付録 C. XML エクステンダーの制限』

dxxGenXMLClob ストアード・プロシージャー

目的:

dxxGenXMLClob は、DAD が入っているバッファーを入力として使用します。これは、DAD 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 文書を構成し、生成された最初の (そして通常は唯一の) XML 文書を *resultDoc* CLOB に戻します。

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*/
```

この *varchar_value* は、Windows および UNIX の場合には 32672 であり、 iSeries および z/OS の場合には 16366 です。

パラメーター:

表 58. dxxGenXMLClob のパラメーター

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
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。式が複数ある場合は二重引用符で囲み、and という語で区切ります。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	合成された XML 文書を含む CLOB。	OUT
valid	valid は次のように設定されます。 • VALIDATION=YES の場合、妥当性検査が正常に行われたときには valid=1、妥当性検査が正常に行われなかったときには valid=0。 • VALIDATION=NO の場合には valid=NULL。	OUT
numDocs	入力データから生成される XML 文書の数。 **注:** 現在は、最初の文書のみが戻されます。	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。 1 MB よりも大きな CLOB ファイルがある場合には、XML エクステンダーにより、ストアード・プロシージャーのパラメーターを再定義するためのコマンド・ファイルが提供されます。 `crtgenxc.zip`

ファイルを、 DB2 UDB XML エクステンダーの Web サイトから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
UNIX および Windows 用の XML エクステンダー V7.2 修正パッケージ 5 以降で使用。

crtgenxc.iseries
iSeries 用の XML エクステンダーで使用。

iSeries の場合には、このファイルをメンバーとしてファイルに入れてください。(例えば、このファイルを DXXSAMPLES/SQLSTMT に入れてください。)

CLOB の長さを指定するには: エディターでファイルを開き、次の例で示すように *resultDoc* パラメーターを変更してください。

```
out    resultDoc     clob(clob_size),
```

推奨サイズ: *resultDoc* パラメーターの上限サイズは、実際のシステム・セットアップに応じて異なりますが、このパラメーターで指定する大きさは、文書のサイズにかかわりなく JDBC によって割り振られる大きさになることに注意してください。このサイズは、使用する最大の XML ファイルが収まる大きさにする必要がありますが、 1.5 ギガバイトを超えないようにしてください。

iSeries でこのコマンド・ファイルを実行するには、コマンド行から次のように入力してください。

```
RUNSQLSTM SRCFILE(DXXSAMPLES/SQLSTMT) SRCMBR(CRTGENXC) NAMING(*SQL)
```

ここで、*DXXSAMPLES/SQLSTMT* は、ファイルをダウンロードしたライブラリーおよびファイルの名前と一致する名前です。

関連タスク:

- 65 ページの『SQL マッピングを使用した XML 文書の合成』
- 69 ページの『RDB_node マッピングを使用した XML コレクションの合成』
- 210 ページの『XML エクステンダーの合成ストアード・プロシージャーの呼び出し』

関連参照:

- 209 ページの『XML エクステンダーの合成ストアード・プロシージャー』
- xi ページの『構文図の読み方』
- 269 ページの『付録 C. XML エクステンダーの制限』

dxxRetrieveXMLClob ストアード・プロシージャー

目的:

dxxRetrieveXMLClob ストアード・プロシージャーを使用すると、リレーショナル・データから文書を合成することができます。

dxxRetrieveXMLClob を使用するための要件は、dxxGenXMLClob 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(varchar(collectionName)          collelctionName  /*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 59. dxxRetrieveXMLClob のパラメーター

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
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。式が複数ある場合は二重引用符で囲み、and という語で区切ります。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

表 59. *dxxRetrieveXMLClob* のパラメーター *(続き)*

パラメーター	説明	IN/OUT パラメーター
valid	valid は次のように設定されます。 • VALIDATION=YES の場合、妥当性検査が正常に行われたときには valid=1、妥当性検査が正常に行われなかったときには valid=0。 • VALIDATION=NO の場合には valid=NULL。	OUT
numDocs	入力データから生成される XML 文書の数。注: 現在は、最初の文書のみが戻されます。	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。 1 MB よりも大きな CLOB ファイルがある場合には、XML エクステンダーにより、ストアード・プロシージャーのパラメーターを再定義するためのコマンド・ファイルが提供されます。 `crtgenxc.zip` ファイルを、 DB2 UDB XML エクステンダーの Web サイトから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
UNIX および Windows 用の XML エクステンダー V7.2 修正パッケージ 5 以降で使用。

crtgenxc.iseries
iSeries 用の XML エクステンダーで使用。

iSeries の場合には、このファイルをメンバーとしてファイルに入れてください。 (例えば、このファイルを DXXSAMPLES/SQLSTMT に入れてください。)

CLOB の長さを指定するには: エディターでファイルを開き、次の例で示すように *resultDoc* パラメーターを変更してください。

```
out    resultDoc     clob(clob_size),
```

推奨サイズ: *resultDoc* パラメーターの上限サイズは、実際のシステム・セットアップに応じて異なりますが、このパラメーターで指定する大きさは、文書のサイズにかかわりなく JDBC によって割り振られる大きさになることに注意してください。このサイズは、使用する最大の XML ファイルが収まる大きさにする必要がありますが、 1.5 ギガバイトを超えないようにしてください。

iSeries でこのコマンド・ファイルを実行するには、コマンド行から次のように入力してください。

```
RUNSQLSTM SRCFILE(DXXSAMPLES/SQLSTMT) SRCMBR(CRTGENXC) NAMING(*SQL)
```

ここで、*DXXSAMPLES/SQLSTMT* は、ファイルをダウンロードしたライブラリーおよびファイルの名前と一致する名前です。

関連タスク:

- 65 ページの『SQL マッピングを使用した XML 文書の合成』
- 69 ページの『RDB_node マッピングを使用した XML コレクションの合成』
- 210 ページの『XML エクステンダーの合成ストアード・プロシージャーの呼び出し』

関連参照:

- 209 ページの『XML エクステンダーの合成ストアード・プロシージャー』
- xi ページの『構文図の読み方』
- 269 ページの『付録 C. XML エクステンダーの制限』

XML エクステンダーの分解ストアード・プロシージャー

分解ストアード・プロシージャー dxxInsertXML() および dxxShredXML() は、受け取った XML 文書を分解または断片化して、新規または既存のデータベース表にデータを保管します。ストアード・プロシージャー dxxInsertXML() は入力として、使用可能な XML コレクション名を取ります。ストアード・プロシージャー dxxShredXML() は入力として DAD ファイルを取ります。使用可能な XML コレクションは必要ありません。

dxxShredXML() ストアード・プロシージャー

目的:

DAD ファイル・マッピングに基づき XML 文書を分解し、 XML エレメントの内容と属性を指定の 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。ストアード・プロシージャーでは、DAD の中で結合条件に指定された列が、既存の表での基本キーと外部キーの関係に対応している必要があります。ルートの element_node の RDB_node に指定された結合条件の列が、すでに表になければなりません。

このセクションのストアード・プロシージャーは、説明用のサンプルです。完全な実行可能サンプルが DXXSAMPLES/QCSRC(SHDX) にあります。

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 60. 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

表 60. *dxxShredXML()* 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	XMLCLOB タイプの XML 文書オブジェクト。	IN
returnCode	ストアード・プロシージャーからの戻りコード。	OUT
returnMsg	エラー発生時に戻されるメッセージ・テキスト。	OUT

例:

以下は dxxShredXML() 呼び出しの例です。完全な実行可能サンプルが DXXSAMPLES/QCSRC(SHDX) にあります。

```
#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(100K) xmlDoc; /* input xml document */

long    returnCode;            /* return 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;

FILE   *file_handle;
long    file_length=0;

/* initialize the DAD CLOB object. */
file_handle = fopen( "/dxxsamples/dad/getstart_xcollection.dad", "r" );
if ( file_handle != NULL ) {
  file_length = fread ((void *) &dad.data
, 1, FILE_SIZE, file_handle);
  if (file_length == 0) {
    printf ("Error reading dad file getstart_xcollection.dad¥n");
    rc = -1;
    goto exit;
 } else
    dad.length = file_length;
}
else {
  printf("Error opening dad file  ¥n");
  rc = -1;
  goto exit;
}

/* Initialize the XML CLOB object. */
  file_handle = fopen( "/dxxsamples/xml/getstart_xcollection.xml", "r" );
  if ( file_handle != NULL ) {
    file_length = fread ((void *) &xmlDoc.data
, 1, FILE_SIZE,
                                 file_handle);
    if (file_length == 0) {
      printf ("Error reading xml file getstart_xcollection.xml ¥n");
```

```
      rc = -1;
      goto exit;
    } else
        xmlDoc.length = file_length;
  }
  else {
    printf("Error opening xml file ¥n");
    rc = -1;
    goto exit;
  }

/* initialize host variable and indicators */
returnCode = 0;
msg_txt[0] = '¥0';
dad_ind = 0;
xmlDoc_ind = 0;
returnCode_ind = -1;
returnMsg_ind = -1;

/* Call the store procedure */
EXEC SQL CALL "DB2XML.DXXSHRED" (:dad:dad_ind;
                                 :xmlDoc:xmlDoc_ind,
                                 :returnCode:returnCode_ind,
                                 :returnMsg:returnMsg_ind);

if (SQLCODE < 0) {
    EXEC SQL ROLLBACK;
  else
    EXEC SQL COMMIT;
}

exit:
  return rc;
```

関連タスク:

- 71 ページの『RDB_node マッピングを使用した XML 文書の分解』
- 101 ページの『XML 文書を分解して DB2 UDB データにする』
- 210 ページの『XML エクステンダーの合成ストアード・プロシージャーの呼び出し』

関連参照:

- 222 ページの『XML エクステンダーの分解ストアード・プロシージャー』
- xi ページの『構文図の読み方』
- 269 ページの『付録 C. XML エクステンダーの制限』

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 61. dxxInsertXML() パラメーター

パラメーター	説明	IN/OUT パラメーター
collectionName	使用可能な XML コレクションの名前。	IN
xmlobj	CLOB タイプの XML 文書オブジェクト。	IN
returnCode	ストアード・プロシージャーからの戻りコード。	OUT
returnMsg	エラー発生時に戻されるメッセージ・テキスト。	OUT

例:

以下の例では、 dxxInsertXML() 呼び出しによって入力 XML 文書 dxx_install/xml/order1.xml を分解し、 XML 使用可能化に使用された DAD ファイルで指定された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。完全な作業用のサンプルが DXXSAMPLES/QCSRC(INSX) にあります。

```
#include "dxx.h"
#include "dxxrc.h"

EXEC SQL INCLUDE SQLCA;
EXEC SQL BEGIN DECLARE SECTION;
char    collectionName[32];   /* name of an XML collection */
SQL TYPE is CLOB(100K) xmlDoc; /* input xml document */
long    returnCode;            /* return error code */
char    returnMsg[1024];       /* error message text */
short   collectionName_ind;
short   xmlDoc_ind;
short   returnCode_ind;
short   returnMsg_ind;
EXEC SQL END DECLARE SECTION;

FILE   *file_handle;
long    file_length=0;

/* initialize the DAD CLOB object. */
file_handle = fopen( "/dxxsamples/dad/getstart_xcollection.dad", "r" );
if ( file_handle != NULL ) {
  file_length = fread ((void *) , &dad.data;
1, FILE_SIZE, file_handle);
  if (file_length == 0) {
    printf ("Error reading dad file getstart_xcollection.dad¥n");
    rc = -1;
    goto exit;
 } else
    dad.length = file_length;
}
else {
  printf("Error opening dad file  ¥n");
  rc = -1;
  goto exit;
}

/* initialize host variable and indicators */
strcpy(collectionName, "sales_ord");
returnCode = 0;
```

```
msg_txt[0] = '¥0';
collectionName_ind = 0;
xmlDoc_ind = 0;
returnCode_ind = -1;
returnMsg_ind = -1;

/* Call the store procedure */
EXEC SQL CALL "DB2XML.DXXINSERTXML" (:collection_name:collection_name_ind,
                                     :xmlDoc:xmlDoc_ind,
                                     :returnCode:returnCode_ind,
                                     :returnMsg:returnMsg_ind);

if (SQLCODE < 0) {
    EXEC SQL ROLLBACK;
  else
    EXEC SQL COMMIT;
}

exit:
  return rc;
```

関連タスク:

- 71 ページの『RDB_node マッピングを使用した XML 文書の分解』
- 101 ページの『XML 文書を分解して DB2 UDB データにする』
- 210 ページの『XML エクステンダーの合成ストアード・プロシージャーの呼び出し』

関連参照:

- 222 ページの『XML エクステンダーの分解ストアード・プロシージャー』
- xi ページの『構文図の読み方』
- 269 ページの『付録 C. XML エクステンダーの制限』

第 11 章 XML エクステンダーの管理サポート表

データベースが使用可能になっている場合、 DTD リポジトリー表 (DTD_REF) と 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。それぞれは、特定の PUBLIC 特権を使用して作成されます。

DTD 参照表

XML エクステンダーには XML DTD リポジトリーとしての機能もあります。データベースで XML が使用可能になると、DTD 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。これには DB2XML というスキーマ名があります。 DTD_REF 表には、表 62 に示すような列が含まれます。

表 62. DTD_REF 表

列名	データ・タイプ	説明
DTDID	VARCHAR(128)	(固有で、NULL ではない) 基本キー。これは DTD の識別に使用されます。 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準拠している必要があります。
CONTENT	XMLCLOB	DTD の内容。
USAGE_COUNT	INTEGER	データベース内において、この DTD を使用して 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ンです。
CREATOR	VARCHAR(128)	最初にデータ挿入を行ったユーザー ID。この列はオプションです。
UPDATOR	VARCHAR(128)	最後に更新を行ったユーザー ID。この列はオプションです。

アプリケーションが DTD を変更できるのは、 USAGE_COUNT がゼロの場合だけです。

PUBLIC に認可される特権

INSERT、UPDATE、DELETE、および SELECT の特権が PUBLIC に付与されます。

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、基本キーは (*table_name*, *col_name*) です。この表の読み取り特権だけが PUBLIC に付与されます。XML_USAGE 表はデータベースが使用可能になった時点で作成されます。 XML_USAGE 表に含まれる列は、表 63 に示すとおりです。

表 63. XML_USAGE 表

列名	説明
table_schema	XML 列の場合、XML 列を 1 つ含むユーザー表のスキーマ名です。 XML コレクションの場合、デフォルト・スキーマ名の ″DXX_COLL″ の値です。
table_name	XML 列の場合、XML 列を 1 つ含むユーザー表の名前です。 XML コレクションの場合、そのエンティティーをコレクションとして識別する値 DXX_COLLECTION です。
col_name	XML 列または XML コレクションの名前。これは table_name とともに複合キーを構成します。
DTDID	DAD ファイルに指定された DTD と DTD_REF に挿入された DTD とを関連付けるストリング。この値は DAD にある DTDID エレメントの値と一致している必要があります。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示します。 XML コレクションは 1、XML 列は 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。固有なトリガー名に使用。
validation	はいは 1、いいえは 0

XML_USAGE 表への項目の追加、変更、削除はしないでください。この表は XML エクステンダー専用です。

PUBLIC に認可される特権

XML_USAGE の場合、SELECT 特権が PUBLIC に与えられます。 INSERT、DELETE、および UPDATE が DB2XML に認可されます。

第 12 章 トラブルシューティング

XML エクステンダー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 ステートメントの実行時、および (DB2 UDB XML エクステンダーのユーザー定義関数 (UDF) の呼び出しを含む) すべての DB2 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 情報やエラー・メッセージなどの、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ことが必要な場合があります。 XML エクステンダーには、XML エクステンダーの活動を記録するトレース機能があります。このトレース情報は、IBM ソフトウェア・サポートにとって重要な情報源となります。トレース機能は、IBM ソフトウェア・サポートの指示のもとでのみ使用してください。

この章ではトレース機能、エラー・コード、およびメッセージについて説明します。

関連参照:

- 232 ページの『XML エクステンダーの SQLSTATE コードおよび関連するメッセージ番号』
- 237 ページの『XML エクステンダーのメッセージ』
- 230 ページの『トレースの停止』
- 229 ページの『XML エクステンダーのトレースの開始』

XML エクステンダーのトレースの開始

目的:

XML エクステンダー サーバーの活動を記録します。トレースを開始するには、ユーザー・プロファイルとトレース・ファイルを保管する既存のディレクトリーの名前を指定して、 `on` オプションを **dxxtrc** に適用します。トレースがオンラインになったとき、ファイル `dxx`*`INSTANCE`*`.trc` は、指定されたディレクトリーに保管されます。 *INSTANCE* は、トレース対象のユーザー・プロファイルに割り当てられている、数値の UID です。トレース・ファイルのサイズには制限はありません。

構文:

Qshell からのトレースの開始:

iSeries ナビゲーターからのトレースの開始:

```
call schema.QZXMTRC('on', 'user_profile', 'trace_directory');
```

OS のコマンド行からのトレースの開始:

```
call QDBXM/QZXMTRC PARM(on user_profile 'trace_directory')
```

パラメーター:

表 64. トレース・パラメーター

パラメーター	説明
user_profile	XML エクステンダーが実行されているジョブに関連付けられているユーザー・プロファイルの名前。
trace_directory	dxx*INSTANCE*.trc が保管される既存の パスおよび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、/u/user1/dxx/trace ディレクトリーにある dxxdb2inst1.trc ファイルを使用したトレースの開始を示しています。

Qshell から、

```
dxxtrc on user1 /u/user1/trace
```

iSeries ナビゲーターから、

```
call myschema.QZXMTRC('on', 'user1', '/u/user1/trace');
```

OS のコマンド行から、

```
call QDBXM/QZXMTRC PARM(on user1 '/u/user1/trace')
```

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。

推奨: トレースを実行中のログ・ファイルのサイズには制限がなく、パフォーマンスに影響を与えかねないため、実稼働環境ではトレースをオフにしてください。

構文:

Qshell からのトレースの停止:

iSeries ナビゲーターからのトレースの停止:

```
call schema.QZXMTRC('off', 'user_profile');
```

OS のコマンド行からのトレースの停止:

```
call QDBXM/QZXMTRC PARM(off user_profile)
```

パラメーター:

表 65. トレース・パラメーター

パラメーター	説明
user_profile	XML エクステンダーが実行されているジョブに関連付けられているユーザー・プロファイルの名前。

例:

以下の例では、トレースの停止を示します。

Qshell から、

```
dxxtrc off user1
```

iSeries ナビゲーターから、

```
call myschema.QZXMTRC('off', 'user1');
```

OS のコマンド行から、

```
call QDBXM/QZXMTRC PARM(off user1)
```

XML エクステンダー 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 ファイルについての詳細は、「*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 関数を使用して検索できます。 (SQLError 関数を使用したエラー情報の取得についての詳細は、「*コール・レベル・インターフェースの手引きおよび解説書*」を参照してください。)

SQLCODE 値が 0 であれば、ステートメントが正常に実行されたことを意味します (ただし、警告状態が発生している場合もあります)。正の SQLCODE 値は、ステートメントが正常に実行されたものの、警告が発生したことを意味します。 (組み込み SQL ステートメントは、 0 または正の SQLCODE 値に関連する警告情報を SQLWARN フィールドに戻します。) 負の SQLCODE 値は、エラーが生じたことを意味します。

DB2 は、それぞれの SQLCODE 値にメッセージを関連付けます。 XML エクステンダー UDF は、警告またはエラーの状態を発見すると、 SQLCODE メッセージに含める関連情報を DB2 UDB に渡します。

DB2 XML エクステンダー UDF を呼び出す組み込み SQL ステートメントおよび DB2 UDB CLI 呼び出しによって、これらの UDF に固有の SQLCODE メッセージや SQLSTATE 値が戻ることもありますが、 DB2 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 エクステンダー UDF を開始しない組み込み SQL ステートメントまたは DB2 UDB CLI 呼び出しの場合と同じです。

XML エクステンダーのストアード・プロシージャーの戻りコード

XML エクステンダーには、ストアード・プロシージャーに関する問題の解決に役立つ戻りコードがあります。ストアード・プロシージャーから戻りコードを受け取ったら、以下のファイルを調べてください。このファイルでは、戻りコードと XML エクステンダー・ エラー・メッセージ番号および記号定数が突き合わされています。

```
dxx_install/include/dxxrc.h
```

関連参照:

- 232 ページの『XML エクステンダーの SQLSTATE コードおよび関連するメッセージ番号』

XML エクステンダーの SQLSTATE コードおよび関連するメッセージ番号

表 66. SQLSTATE コードおよび関連メッセージ番号

SQLSTATE	メッセージ番号	説明
00000	DXX*nnnn*I	エラーは発生していません。
01HX0	DXXD003W	パス式で指定されたエレメントまたは属性が XML 文書にありません。
38X00	DXXC000E	XML エクステンダーは、指定されたファイルを開くことができません。
38X01	DXXA072E	XML エクステンダーは、データベースを使用可能にする前に自動的にバインドを試みましたが、バインド・ファイルを検出できませんでした。
	DXXC001E	XML エクステンダーは、指定されたファイルを検出できませんでした。
38X02	DXXC002E	XML エクステンダーは、指定されたファイルからデータを読み取ることができません。
38X03	DXXC003E	XML エクステンダーは、データをファイルに書き込むことができません。
	DXXC011E	XML エクステンダーは、データをトレース制御ファイルに書き込むことができません。
38X04	DXXC004E	XML エクステンダーは、指定されたロケーターを操作できませんでした。
38X05	DXXC005E	ファイル・サイズが XMLVarchar サイズより大きいため、 XML エクステンダーがファイルからインポートできなかったデータがあります。

表 66. 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
38X06	DXXC006E	ファイル・サイズが XMLCLOB のサイズより大きいため、 XML エクステンダーがファイルからインポートできなかったデータがあります。
38X07	DXXC007E	LOB ロケーターのバイト数がファイル・サイズと等しくありません。
38X08	DXXD001E	スカラー抽出関数が、複数出現するロケーション・パスを使用しました。スカラー関数は、複数出現することのないロケーション・パスのみを使用することができます。
38X09	DXXD002E	パス式の構文が正しくありません。
38X10	DXXG002E	XML エクステンダーは、オペレーティング・システムからメモリーを割り当てることができませんでした。
38X11	DXXA009E	このストアード・プロシージャーは XML 列専用です。
38X12	DXXA010E	XML エクステンダーが列を使用可能にしようとしたとき、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	列を使用可能にしようとして、XML エクステンダーが SCHEMA ID を見つけることができませんでした。SCHEMA ID は、 DAD ファイルの schemabindings タグの内側の nonamespacelocation タグの location 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。
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。

表 66. 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
	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/属性を、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。外部キーのデータ値は、基本キーのデータ値によって決定されます。この XML 文書の指定されたエレメント/属性を表および列にマップする必要はありません。
	DXXQ057E	DAD ファイルの中で schemabindings タグと DTD ID タグを混在させることはできません。
	DXXQ058E	DAD ファイルの中で、schemabindings タグの内側に nonamespacelocation タグが欠落しています。
	DXXQ059E	スキーマ妥当性検査のための DAD の中で、 XCollection タグの内側に doctype タグを指定することはできません。
	DXXQ062E	通常、このエラー条件は、特定のエレメントまたは属性の親 element_node に multi_occurrence = YES が指定されていない場合に発生します。
	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、または欠落しています。その値は yes か no でなければなりません (大文字小文字を区別しません)。
	DXXQ064E	結合条件に指定されたキー列が、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。

表 66. 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
38X16	DXXG004E	必須パラメーターのヌル値が XML ストアード・プロシージャーに渡されました。
38X17	DXXQ001E	文書アクセス定義 (DAD) ファイル内の SQL ステートメント、または DAD をオーバーライドする SQL ステートメントが無効です。 XML 文書を生成するには SELECT ステートメントが必要です。
38X18	DXXG001E	XML エクステンダーは内部エラーを検出しました。
	DXXG006E	CLI を使用中に XML エクステンダーで内部エラーが発生しました。
38X19	DXXQ002E	システムのメモリーまたはディスク・スペースが不足しています。生成される XML 文書を保管する容量がありません。
38X20	DXXQ003W	ユーザー定義の SQL 照会によって、指定した最大数以上の XML 文書が生成されます。指定された数の文書のみが戻されます。
38X21	DXXQ004E	指定された列は、SQL 照会の結果には含まれません。
38X22	DXXQ005E	SQL 照会の XML へのマッピングが正しくありません。
38X23	DXXQ006E	文書アクセス定義 (DAD) ファイル内の attribute_node エレメントに名前属性がありません。
38X24	DXXQ007E	文書アクセス定義 (DAD) ファイル内の attribute_node エレメントに、列エレメントまたは RDB_node がありません。
38X25	DXXQ008E	文書アクセス定義 (DAD) ファイル内の text_node エレメントに、列エレメントがありません。
38X26	DXXQ009E	指定された結果表が、システム・カタログ内にありません。
38X27	DXXQ010E 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 エクステンダーの生成した XML 文書が大きすぎて、結果表の列の中に入りません。

表 66. 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
	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 エクステンダーは予期したエレメントを検出できませんでした。
	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 condition エレメントの条件の形式が無効です。
	DXXQ061E	ストリング表記の形式が無効です。ストリングの値が日付、時刻、またはタイム・スタンプの場合、構文が対応するデータ・タイプに適合していません。
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。
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。

表 66. 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
38X37	DXXG007E	サーバーのオペレーティング・システムの地域が、 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・システムの地域設定が、コード・ページ表にありません。
38X41	DXXQ048E	スタイルシート・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無効である可能性があります。
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 は、アクセス権がないため、指定されたディレクトリー内で出力文書の固有の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中であるか、あるいはディレクトリーが存在していない可能性があります。
38X44	DXXQ051E	1 つ以上の入出力パラメーターで有効な値が指定されていません。
38X45	DXXQ055E	変換操作中に ICU エラーが発生しました。

XML エクステンダーのメッセージ

DXXA000I　列 <*column_name*> の使用可能化中。しばらくお待ちください。

説明: これは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、ソフトウェア・サービス提供者に連絡してください。エラーを報告する場合、すべてのメッセージ・テキスト、トレース・ファイル、および問題の再現方法についての説明を必ず知らせてください。

DXXA002I　データベース <*database*> に接続中。

説明: これは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、または壊れています。

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. データベースが壊れているかどうかを判別します。データベースが壊れている場合は、バックアップから回復するようデータベース管理者に要請します。

DXXA004E　データベース <*database*> を使用可能にできません。

説明: データベースはすでに使用可能であるか、または壊れています。

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. データベースが壊れているかどうかを判別します。データベースが壊れている場合は、バックアップから回復するようデータベース管理者に要請します。

DXXA005I　データベース <*database*> の使用可能化中。しばらくお待ちください。

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 エクステンダーはこれを使用不可にできません。

ユーザーの処置: 重要なデータのバックアップを取り、XML 列またはコレクションをすべて使用不可にし、表の更新または除去を行ってデータベースから XML データ・タイプをなくします。

DXXA008I　列 <*column_name*> を使用不可にしています。しばらくお待ちください。

説明: これは情報メッセージです。

ユーザーの処置: 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 エクステンダーが列を使用可能にしようとしたとき、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 表へのレコード挿入が失敗しました。

説明: XML エクステンダーが列を使用可能にしようとしたとき、 DB2XML.XML_USAGE 表の中にレコードを挿入できませんでした。

ユーザーの処置: DB2XML.XML_USAGE 表が存在すること、および同じ名前のレコードが表にまだ存在しないことを確認します。

DXXA012E　DB2XML.DTD_REF 表の更新が失敗しました。

説明: XML エクステンダーが列を使用可能にしようとしたとき、DB2XML.DTD_REF 表を更新できませんでした。

ユーザーの処置: DB2XML.DTD_REF 表が存在することを確認してください。表が破壊されていないかどうか、また管理ユーザー ID が表を更新するために正しい権限を持っているかどうかを判別してください。

DXXA013E　表 <*table_name*> の更新が失敗しました。

説明: XML エクステンダーが列を使用可能にしようとしたとき、指定された表を更新できませんでした。

ユーザーの処置: 表の更新に必要な特権を確認してください。

DXXA014E　指定された root ID 列: <*root_id*> は、表 <*table_name*> の単一の基本キーではありません。

説明: 指定されたルート ID がキーではないか、または表 *table_name* のただ 1 つのキーではありません。

ユーザーの処置: 指定されたルート ID が、表のただ 1 つの基本キーであることを確認してください。

DXXA015E　列 DXXROOT_ID は表 <*table_name*> に既に存在しています。

説明: 列 DXXROOT_ID は存在しますが、XML エクステンダーが作成したものではありません。

ユーザーの処置: 列を使用可能にするとき、異なった列名を使用することによって、ルート ID オプションに基本列を指定します。

DXXA016E　入力表 <*table_name*> が存在しません。

説明: XML エクステンダーは、システム・カタログ内に指定された表を検出できませんでした。

ユーザーの処置: データベースに表が存在し、正しく指定されていることを確認してください。

DXXA017E　入力列 <*column_name*> が、指定した表 <*table_name*> に存在しません。

説明: XML エクステンダーは、システム・カタログ内に列を検出できませんでした。

ユーザーの処置: ユーザー表内に列が存在することを確認してください。

DXXA018E　指定した列は XML データに対して使用可能化されていません。

説明: XML エクステンダーが列を使用不可にしようとしたとき、 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ヌルです。

説明: enable_column() ストアード・プロシージャーの必須入力パラメーターがヌルです。

ユーザーの処置: enable_column() ストアード・プロシージャーのすべての入力パラメーターを検査してください。

DXXA020E　列が表 <*table_name*> に見つかりません。

説明: XML エクステンダーがデフォルトのビューを作成しようとしたとき、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　デフォルトのビュー <*default_view*> を作成できません。

説明: XML エクステンダーが列を使用可能化しようとしたとき、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: デフォルトのビュー名が固有のものであることを確認してください。その名前のビューがすでに存在する場合は、固有の名前をデフォルトの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 エクステンダーが列を使用不可にしようとしたとき、文書アクセス定義 (DAD) ファイルを検出できませんでした。

ユーザーの処置: 正しいデータベース名、表名、または列名を指定したことを確認してください。

DXXA024E　システム・カタログ表にアクセス中に XML エクステンダーで内部エラーが発生しました。

説明: XML エクステンダーは、システム・カタログ表にアクセスできませんでした。

ユーザーの処置: データベースが安定状態であることを確認してください。

DXXA025E　デフォルトのビュー <*default_view*> をドロップできません。

説明: XML エクステンダーが列を使用不可にしようとしたとき、デフォルトのビューを除去できませんでした。

ユーザーの処置: XML エクステンダーの管理ユーザー ID に、デフォルトのビューの除去に必要な特権があることを確認してください。

DXXA026E　サイド表 <*side_table*> をドロップできません。

説明: XML エクステンダーが列を使用不可にしようとしたとき、指定された表を除去できませんでした。

ユーザーの処置: XML エクステンダーの管理者ユーザー ID に、表の除去に必要な特権があることを確認してください。

DXXA027E　列を使用不可にできませんでした。

説明: XML エクステンダーは、内部トリガー障害のため、列を使用不可にできませんでした。原因としては次のことが考えられます。

- システムはメモリーが不足しています。
- この名前を持つトリガーは存在しません。

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 エクステンダーは、内部トリガー障害のため、列を使用不可にできませんでした。原因としては次のことが考えられます。

- システムはメモリーが不足しています。
- この名前を持つトリガーは存在しません。

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 エクステンダーは、内部トリガー障害のため、列を使用不可にできませんでした。原因としては次のことが考えられます。

- システムはメモリーが不足しています。
- この名前を持つトリガーは存在しません。

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 エクステンダーは、内部トリガー障害のため、列を使用不可にできませんでした。原因としては次のことが考えられます。

- システムはメモリーが不足しています。
- この名前を持つトリガーは存在しません。

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 エクステンダーが列を使用不可にしようとしたとき、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 エクステンダーの管理者ユーザー ID に、アプリケーション表の更新に必要な特権があることを確認してください。

DXXA032E　DB2XML.XML_USAGE 表内の USAGE_COUNT の減分に失敗しました。

説明: XML エクステンダーが列を使用不可にしようとしたとき、 USAGE_COUNT 列の値を 1 つ減らすことができませんでした。

ユーザーの処置: DB2XML.XML_USAGE 表が存在することと、 XML エクステンダー管理者ユーザー ID に、表の更新に必要な特権があることを確認してください。

DXXA033E　DB2XML.XML_USAGE 表から行を削除しようとして失敗しました。

説明: XML エクステンダーが列を使用不可にしようとしたとき、 DB2XML.XML_USAGE 表内のこの列に関連する行を削除できませんでした。

ユーザーの処置: DB2XML.XML_USAGE 表が存在することと、 XML エクステンダー管理者ユーザー ID に、この表の更新に必要な特権があることを確認してください。

DXXA034I　XML エクステンダーは列 <*column_name*> を正常に使用不可にしました。

説明: これは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 エクステンダーはデータベース <*database*> を使用不可にしています。しばらくお待ちください。

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 エクステンダーは、データベース <*database*> を正常に使用不可にしました。

説明: これは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 文字以内の短い名前を指定してください。

DXXA038E　指定されたデフォルトのビュー名は 18 文字を超えています。

説明: デフォルトのビュー名を英数字で 18 文字よりも長くすることはできません。

ユーザーの処置: 18 文字以内の短い名前を指定してください。

DXXA039E　指定された ROOT_ID 名は 18 文字を超えています。

説明: ROOT_ID 名を英数字で 18 文字よりも長くすることはできません。

ユーザーの処置: 18 文字以内の短い名前を指定してください。

DXXA046E　サイド表 <*side_table*> を作成できません。

説明: XML エクステンダーが列を使用可能化しようとしたとき、指定されたサイド表を作成できませんでした。

ユーザーの処置: XML エクステンダーの管理者ユーザー ID に、サイド表の作成に必要な特権があることを確認してください。

DXXA047E　列を使用可能にできませんでした。

説明: XML エクステンダーは、内部トリガー障害のため、列を使用可能にできませんでした。原因としては次のことが考えられます。

- DAD ファイルには、不正な構文があります。
- システムはメモリーが不足しています。
- 同じ名前を持つ別のトリガーが存在しています。

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 エクステンダーは、内部トリガー障害のため、列を使用可能にできませんでした。原因としては次のことが考えられます。

- DAD ファイルには、不正な構文があります。
- システムはメモリーが不足しています。
- 同じ名前を持つ別のトリガーが存在しています。

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 エクステンダーは、内部トリガー障害のため、列を使用可能にできませんでした。原因としては次のことが考えられます。

- DAD ファイルには、不正な構文があります。
- システムはメモリーが不足しています。
- 同じ名前を持つ別のトリガーが存在しています。

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 エクステンダーは、内部トリガー障害のため、列を使用可能にできませんでした。原因としては次のことが考えられます。

- DAD ファイルには、不正な構文があります。
- システムはメモリーが不足しています。
- 同じ名前を持つ別のトリガーが存在しています。

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 エクステンダーは、内部トリガー障害のため、列を使用不可にできませんでした。原因としては次のことが考えられます。

- システムはメモリーが不足しています。

- この名前を持つトリガーは存在しません。

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 エクステンダーは、内部トリガー障害のため、列を使用不可にできませんでした。原因としては次のことが考えられます。

- DAD ファイルには、不正な構文があります。
- システムはメモリーが不足しています。
- 同じ名前を持つ別のトリガーが存在しています。

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 エクステンダーは、内部トリガー障害のため、列を使用可能にできませんでした。原因としては次のことが考えられます。

- DAD ファイルには、不正な構文があります。
- システムはメモリーが不足しています。
- 同じ名前を持つ別のトリガーが存在しています。

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 エクステンダーは、内部トリガー障害のため、列を使用可能にできませんでした。原因としては次のことが考えられます。

- DAD ファイルには、不正な構文があります。
- システムはメモリーが不足しています。
- 同じ名前を持つ別のトリガーが存在しています。

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 ファイル内の列 <*column_name*> のタイプ <*column_type*> 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 <*column_name*> の multi_occurrence 属性 <*multi_occurrence*> が無効です。

説明: 文書アクセス定義 (DAD) ファイル内の列の multi_occurrence 属性が正しくないか、または属性がありません。

ユーザーの処置: 列の multi_occurrence 属性が DAD ファイル内に正しく指定されていることを確認してください。

DXXA062E *<column_name>* **の列番号 (表** *<table_name>***) を検索することができません。**

説明: XML エクステンダーは、*table_name* 表の *column_name* の列番号をシステム・カタログから検索できませんでした。

ユーザーの処置: アプリケーション表が適正に定義されていることを確認してください。

DXXA063I **コレクション** *<collection_name>* **を使用可能にしています。しばらくお待ちください。**

説明: これは情報メッセージです。

ユーザーの処置: 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。しばらくお待ちください。**

説明: これは情報メッセージです。

ユーザーの処置: 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: 共用ライブラリー db2xml を検査して、許可が正しいかどうかを確認してください。

ユーザーの処置: クライアントにストアード・プロシージャーを実行する許可があることを確認してください。

DXXA066I **XML エクステンダーは、コレクション** *<collection_name>* **を正常に使用不可にしました。**

説明: これは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 エクステンダーは、コレクション** *<collection_name>* **を正常に使用可能にしました。**

説明: これは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 エクステンダーは、トレースを正常にオンにしました。**

説明: これは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 エクステンダーは、トレースを正常にオフにしました。**

説明: これは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 エクステンダーはバインド・ファイルを見つけられませんでした。使用可能にする前にデータベースをバインドしてください。**

説明: XML エクステンダーは、データベースを使用可能にする前に自動的にバインドを試みましたが、バインド・ファイルを検出できませんでした。

ユーザーの処置: 使用可能にする前にデータベースをバインドしてください。

DXXA073E **データベースがバインドされていません。使用可能にする前にデータベースをバインドしてください。**

説明: データベースがバインドされていません。使用可能にする前にデータベースをバインドしてください。

ユーザーの処置: 使用可能にする前にデータベースをバインドしてください。

DXXA074E　パラメーター・タイプが間違っています。ストアード・プロシージャーには STRING パラメーターを使用してください。

説明: ストアード・プロシージャーには STRING パラメーターを使用してください。

ユーザーの処置: 入力パラメーターが STRING タイプになるように宣言してください。

DXXA075E　パラメーター・タイプが間違っています。入力パラメーターには、long 型を使用してください。

説明: ストアード・プロシージャーは、入力パラメーターが LONG タイプになることを予期しています。

ユーザーの処置: 入力パラメーターが LONG タイプになるように宣言してください。

DXXA076E　XML エクステンダーのトレース・インスタンス ID が無効です。

説明: 提供されたインスタンス ID のトレースを開始できません。

ユーザーの処置: インスタンス ID が正しい 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・ライセンスが有効期限切れになっているか、あるいは存在しません。

ユーザーの処置: サービス提供者に連絡して、新規ソフトウェア・ライセンスを取得してください。

DXXC000E　指定されたファイルをオープンできませんでした。

説明: XML エクステンダーは、指定されたファイルを開くことができません。

ユーザーの処置: アプリケーション・ユーザー ID に、ファイルの読み取りおよび書き込み許可が与えられていることを確認してください。

DXXC001E　指定されたファイルが見つかりませんでした。

説明: XML エクステンダーは、指定されたファイルを検出できませんでした。

ユーザーの処置: ファイルが存在し、パスが正しく指定されていることを確認してください。

DXXC002E　ファイルを読み取れませんでした。

説明: XML エクステンダーは、指定されたファイルからデータを読み取ることができません。

ユーザーの処置: アプリケーション・ユーザー ID に、ファイルの読み取り許可が与えられていることを確認してください。

DXXC003E　指定されたファイルに書き込めませんでした。

説明: XML エクステンダーは、データをファイルに書き込むことができません。

ユーザーの処置: アプリケーション・ユーザー ID にファイルの書き込み許可が与えられていて、ファイル・システムに十分なスペースがあることを確認してください。

DXXC004E　LOB ロケーターを操作することができませんでした: rc=<*locator_rc*>

説明: XML エクステンダーは、指定されたロケーターを操作できませんでした。

ユーザーの処置: LOB ロケーターが正しく設定されていることを確認してください。

DXXC005E　入力ファイル・サイズが XMLVarchar サイズより大きいです。

説明: ファイル・サイズが XMLVarchar サイズより大きいため、 XML エクステンダーがファイルからインポートできなかったデータがあります。

ユーザーの処置: XMLCLOB 列タイプを使用してください。

DXXC006E　入力ファイルが DB2 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 エクステンダーがファイルからインポートできなかったデータがあります。

ユーザーの処置: ファイルをより小さいオブジェクトに分解するか、または XML コレクションを使用してください。

DXXC007E　ファイルから LOB ロケーターにデータを検索できませんでした。

説明: LOB ロケーターのバイト数がファイル・サイズと等しくありません。

ユーザーの処置: LOB ロケーターが正しく設定されていることを確認してください。

DXXC008E　ファイル *<file_name>* を除去できませんでした。

説明: ファイルに共用アクセスが設定されているか、またはファイルがまだオープンしています。

ユーザーの処置: ファイルをクローズするか、またはファイルを保留にしているプロセスを停止してください。DB2 を停止してから、再始動する必要があります。

DXXC009E　*<directory>* ディレクトリーにファイルを作成できませんでした。

説明: XML エクステンダーは、ディレクトリー *directory* 内にファイルを作成することができません。

ユーザーの処置: ディレクトリーが存在し、アプリケーション・ユーザー ID にディレクトリーに対する書き込み許可が与えられており、ファイル・システムに十分なスペースがあることを確認してください。

DXXC010E　ファイル *<file_name>* に書き込み中にエラーが発生しました。

説明: ファイル *file_name* に書き込み中にエラーがありました。

ユーザーの処置: ファイル・システムに十分なスペースがあることを確認してください。

DXXC011E　トレース制御ファイルに書き込めませんでした。

説明: XML エクステンダーは、データをトレース制御ファイルに書き込むことができません。

ユーザーの処置: アプリケーション・ユーザー ID にファイルの書き込み許可が与えられていて、ファイル・システムに十分なスペースがあることを確認してください。

DXXC012E　一時ファイルを作成できません。

説明: システム temp ディレクトリー内にファイルを作成できません。

ユーザーの処置: アプリケーション・ユーザー ID にディレクトリーに対する書き込み許可が与えられていて、ファイル・システムに十分なスペースがあることを確認してください。

DXXC013E　抽出 UDF の結果が、UDF 戻りタイプのサイズ限界を超えています。

説明: 抽出 UDF によって戻されるデータは、「DB2 UDB XML エクステンダー 管理およびプログラミングの手引き」で説明されている UDF の戻りタイプのサイズ限界に収まっていなければなりません。例えば、extractVarchar の結果は (末尾の NULL を含めて) 4000 バイトを超えてはなりません。

ユーザーの処置: 戻りタイプのサイズ限界値を大きくした抽出 UDF を使用してください。 extractChar() の場合には 254 バイト、extractVarchar() の場合には 4 KB、また extractClob() の場合には 2 GB にしてください。

DXXD000E　無効な XML 文書が拒否されました。

説明: 無効な文書を表の中に保管しようとしました。妥当性検査に失敗しました。

ユーザーの処置: 不可視の無効文字を表示できるエディターを使用して、この文書を DTD によって検査してください。このエラーを抑制するには、DAD ファイルの妥当性検査をオフにしてください。

DXXD001E　*<location_path>* は複数回発生します。

説明: スカラー抽出関数が、複数出現するロケーション・パスを使用しました。スカラー関数は、複数回出現することのないロケーション・パスのみを使用することができます。

ユーザーの処置: 表関数を使用してください (スカラー関数名の終わりに 's' を追加してください)。

DXXD002E　サーチ・パスの位置 *<position>* 付近で構文エラーが発生しました。

説明: パス式の構文が正しくありません。

ユーザーの処置: 照会のサーチ・パス引き数を訂正してください。パス式の構文についての資料を参照してください。

DXXD003W　パスが見つかりませんでした。ヌルが戻されました。

説明: パス式で指定されたエレメントまたは属性が XML 文書にありません。

ユーザーの処置: 指定したパスが正しいかどうか検査してください。

DXXG000E　ファイル名 <*file_name*> が無効です。

説明: 無効なファイル名が指定されました。

ユーザーの処置: 正しいファイル名を指定して、再試行してください。

DXXG001E　ビルド <*build_ID*>、ファイル <*file_name*>、行 <*line_number*> で内部エラーが発生しました。

説明: XML エクステンダーは内部エラーを検出しました。

ユーザーの処置: 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情報を必ず伝えてください。

DXXG002E　システムはメモリーが不足しています。

説明: XML エクステンダーは、オペレーティング・システムからメモリーを割り当てることができませんでした。

ユーザーの処置: いくつかのアプリケーションをクローズして再試行してください。問題が続く場合、ご使用のオペレーティング・システムの資料を参照してください。オペレーティング・システムによっては、問題を訂正するためにシステムをリブートする必要があります。

DXXG004E　無効なヌル・パラメーター。

説明: 必須パラメーターのヌル値が XML ストアード・プロシージャーに渡されました。

ユーザーの処置: ストアード・プロシージャー呼び出しの引き数リストの中で、必須パラメーターをすべて検査してください。

DXXG005E　パラメーターはサポートされていません。

説明: このパラメーターはこのリリースではサポートされません。将来のリリースでサポートされます。

ユーザーの処置: このパラメーターを NULL に設定してください。

DXXG006E　内部エラー CLISTATE=<*clistate*>、RC=<*cli_rc*>、ビルド <*build_ID*>、ファイル <*file_name*>、行 <*line_number*> CLIMSG=<*CLI_msg*>。

説明: CLI を使用中に XML エクステンダーで内部エラーが発生しました。

ユーザーの処置: 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*> と一致しません。

説明: サーバーのオペレーティング・システムの地域が、 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・システムの地域設定を修正して、 DB2 を再起動してください。

DXXG008E　ロケール <*locale*> はサポートされていません。

説明: サーバーのオペレーティング・システムの地域設定が、コード・ページ表にありません。

ユーザーの処置: サーバーのオペレーティング・システムの地域設定を修正して、 DB2 を再起動してください。

DXXG017E　ビルド *build_ID*、ファイル *file_name*、および行 *line_number* で、*XML_Extender_constant* の限界を超過しました。

説明: 「XML エクステンダー 管理およびプログラミングの手引き」を参照して、アプリケーションが制限表の値を超えていないか確認してください。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、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についての情報 (入力 DAD、XML 文書、および表定義など) を伝えてください。

ユーザーの処置: サーバーのオペレーティング・システムの地域設定を修正して、 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: このメッセージと一緒に表示されるメッセージにさらに説明がないか調べ、使用しているオペレーティング・システムの DB2 UDB メッセージとコードの文書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) 内の SQL ステートメント、または DAD をオーバーライドする SQL ステートメントが無効です。 XML 文書を生成するには SELECT ステートメントが必要です。

ユーザーの処置: SQL ステートメントを訂正してください。

DXXQ002E XML 文書を保持するための記憶域を生成できません。

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 エクステンダーは予期したエレメントを検出できませんでした。

ユーザーの処置: DAD が有効な XML 文書であり、DAD DTD が必要とするすべてのエレメントを含んでいるか検査してください。 DAD DTD については、XML エクステンダーの資料を参照してください。

DXXQ013E 表または列エレメントは、**DAD** ファイルに名前を持ちません。

説明: エレメントの表または列の名前が、文書アクセス定義 (DAD) ファイル内で指定されていなければなりません。

ユーザーの処置: 表または列エレメントの名前を DAD 内に指定してください。

DXXQ014E element_node エレメントに名前がありません。

説明: 文書アクセス定義 (DAD) ファイル内の element_node エレメントに名前属性がありません。

ユーザーの処置: どの element_node エレメントにも、名前が DAD ファイル内に指定されていることを確認してください。

DXXQ015E 条件形式が無効です。

説明: 文書アクセス定義 (DAD) の条件エレメントの条件が、無効な形式です。

ユーザーの処置: 条件形式を有効なものにしてください。

DXXQ016E この **RDB_node** の表名は、**DAD** ファイルの先頭エレメントに定義されていません。

説明: すべての表は、文書アクセス定義 (DAD) ファイル内の先頭エレメントの RDB_node で定義しなければなりません。サブエレメントの表は、先頭エレメントで定義された表と一致しなければなりません。この RDB_node 内の表名は、先頭エレメントの中にはありません。

ユーザーの処置: RDB ノードの表が、DAD ファイルの先頭エレメントの中で定義されることを確認してください。

DXXQ017E 結果表 <*table_name*> 内の列は小さすぎます。

説明: XML エクステンダーの生成した XML 文書が大きすぎて、結果表の列の中に入りません。

ユーザーの処置: 結果表を除去します。より大きな列を使用して別の結果表を作成します。ストアード・プロシージャーを再実行します。

DXXQ018E ORDER BY 文節が **SQL** ステートメントから欠落しています。

説明: SQL を XML にマップする文書アクセス定義 (DAD) ファイル内の SQL ステートメントに、 ORDER BY 文節がありません。

ユーザーの処置: DAD ファイルを編集します。エンティティーを識別する列を含む ORDER BY 文節を追加します。

DXXQ019E エレメント **objids** は **DAD** ファイル内に列エレメントを持ちません。

説明: エレメント objids は、 SQL を XML にマップする文書アクセス定義 (DAD) ファイルに列エレメントを持ちません。

ユーザーの処置: DAD ファイルを編集します。エレメント objids のサブエレメントとしてキー列を追加してください。

DXXQ020I XML が正常に生成されました。

説明: 要求された XML 文書が、正常にデータベースから生成されました。

ユーザーの処置: 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*> のタイプは <*type_name*> でなければなりません。

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 <*table_name*> の列 <*column_name*> を <*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 XML_USAGE 表内にコレクション <*collection_name*> が見つかりません。

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 コレクション <*collection_name*> の XML_USAGE 内に DAD が見つかりませんでした。

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 override 構文です。

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 エクステンダーの資料を参照してください。

DXXQ035E DAD のトップ RDB_node の結合状態が無効です: <*condition*>

説明: 最上位の RDB_node の条件エレメント内の列名は、 DAD が複数の表を指定する場合には表名で修飾されなければなりません。

ユーザーの処置: DAD 内の結合条件の正しい構文については、XML エクステンダーの資料を参照してください。

DXXQ036E DAD 状態タグの下に指定されているスキーマ名が長過ぎます。

説明: DAD 状態タグの下にあるテキストを解析しているときにエラーが検出されました。状態テキストの ID を修飾しているスキーマ名が長すぎます。

ユーザーの処置: 文書アクセス定義 (DAD) 内の状態タグのテキストを修正してください。

DXXQ037E 複数オカレンスで <*element*> を生成できません。

説明: エレメント・ノードとその子孫にはデータベースへのマッピングがありませんが、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、そのノードの子孫のいずれかに RDB_node を作成して 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*> エレメントで指定された SQL ステートメントが、許容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 の場合には 32765 バイト以下に、 OS/390 および iSeries の場合には 16380 バイト以下に短縮してください。

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 合成に使用される DAD ファイルでは、同じ表内の固有の列を指定する text_node エレメントおよび attribute_node エレメントの数は、最大 100 までに制限されています。

ユーザーの処置: 同じ表内の固有の列を参照する text_node および attribute_node エレメントの合計数を 100 以下に削減してください。

DXXQ040E DAD ファイル内のエレメント名 <*element_name*> は無効です。

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 ファイルで正しく指定されていることを確認してください。 DAD ファイルの 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・パスを 1 つのみ指定してください。

ユーザーの処置: エレメント名が DAD ファイルで正しく指定されていることを確認してください。 DAD ファイルの 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うちの 1 つに存在しませんでした。

ユーザーの処置: XML 文書内の、DAD によって必要とされる個所に、その属性を指定してください。

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 によると、<*ancestor*> は <*elem_name*> の上位エレメントです。 XML 文書内の 1 つまたは複数のエレメント <*elem_name*> に、このような上位エレメントはありません。

ユーザーの処置: XML 文書内のエレメントのネスティングが、対応する DAD で指定されたとおりに行われていることを確認してください。

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含みます。

説明: XML 文書内の <*elem_name*> の下のサブツリーに属性 <*attrib_name*> のインスタンスが複数含まれています。 DAD によると、この属性は分解して同じ行に入れることになっています。分解されるエレメントまたは属性には、固有の名前を指定してください。

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固有の名前が指定されていることを確認してください。

DXXQ046W DTD ID が DAD 内に見つかりませんでした。

説明: DAD において VALIDATION は YES に設定されていますが、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* でパーサー・エラーが発生しました: *msg*

説明: 報告されたエラーのため、パーサーは文書を構文解析することができませんでした。

ユーザーの処置: 文書内のエラーを訂正してください。必要であれば、XML 仕様書を参照してください。

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・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無効である可能性があります。

ユーザーの処置: XML 文書とスタイルシートが有効であることを確認してください。

DXXQ049E 出力ファイルはすでに存在しています。

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固有の名前に変更するか、あるいは既存のファイルを削除してください。

DXXQ050E 固有ファイル名を作成できません。

説明: UDF は、指定されたディレクトリー内で出力文書の固有のファイル名を作成することができませんでした。原因としては、UDF がアクセス権を持っていないか、生成可能なすべてのファイル名が使用中であるか、あるいはディレクトリーが存在していないことが考えられます。

ユーザーの処置: UDF が指定されたディレクトリーにアクセスできることを確認するか、あるいは使用可能なファイル名を含む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: 1 つ以上の入出力パラメーターで有効な値が指定されていません。

ユーザーの処置: ステートメントを調べて、必須パラメーターが欠落していないかどうか検査してください。

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、あるいは DB2XML.XML_USAGE 表が除去されています。

ユーザーの処置: データベースが使用可能になっていること、および DB2XML.XML_USAGE 表にアクセスできることを確認してください。

DXXQ053E SQL ステートメントが失敗しました: *msg*

説明: XML エクステンダーの処理中に生成された SQL ステートメントの実行が失敗しました。 DB2XML.XML_USAGE が除去されています。

ユーザーの処置: 詳細についてはトレースを調べてください。エラー状態を訂正できない場合には、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情報を必ず伝えてください。

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: 関係するストアード・プロシージャーまたは UDF のシグニチャーを検査し、実際の入力パラメーターが正しいことを確認してください。

DXXQ055E ICU エラー: *uerror*

説明: 変換操作中に ICU エラーが発生しました。

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、およびエラーを再現するための手順を含めてください。

DXXQ056E エレメント/属性 *xmlname* を、外部キーの一部として指定された列 (表 *table* の列 *column*) 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/属性を、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

部キーのデータ値は、基本キーのデータ値によって決定されます。この xml 文書の指定されたエレメント/属性を表および列にマップする必要はありません。

ユーザーの処置: DAD の中で、指定された列および表への RDB_node のマッピングを除去してください。

DXXQ057E DAD ファイルの中で schemabindings タグと dtdid タグを混在させることはできません。

説明: DAD ファイルの中で schemabindings タグと dtdid タグを混在させることはできません。

ユーザーの処置: DAD ファイルの中には、schemabindings タグか dtdid タグのどちらか一方だけが存在するようにしてください。

DXXQ058E DAD ファイルの中で、schemabindings タグの内側に nonamespacelocation タグが欠落しています。

説明: DAD ファイルの中で、 schemabindings タグの内側に nonamespacelocation タグが欠落しています。

ユーザーの処置: schemabindings タグに nonamespacelocation タグを追加してください。

DXXQ059E スキーマ妥当性検査のための DAD の中で、 XCollection タグの内側に doctype タグを指定することはできません。

説明: スキーマ妥当性検査のための DAD の中で、XCollection タグの内側に doctype タグを指定することはできません。

ユーザーの処置: スキーマ妥当性検査のための Xcollection タグの内側にある doctype タグを除去してください。

DXXQ060E SCHEMA ID *schemaid* を検索しようとして失敗しました。

説明: 列を使用可能にしようとして、XML エクステンダーが SCHEMA ID を見つけることができませんでした。 SCHEMA ID は、 DAD ファイルの schemabindings タグの内側の nonamespacelocation タグの location 属性の値に対応します。

ユーザーの処置: SCHEMA ID の正しい値が DAD ファイルの中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。ストリングの値が日付、時刻、またはタイム・スタンプの場合、構文が対応するデータ・タイプに適合していません。

ユーザーの処置: 日付、時刻、またはタイム・スタンプの値の形式を、対応するデータ・タイプの形式に適合したものにしてください。

DXXQ062E *table* の結果セットの中に、 *element* の XML 値を生成するための行がありません。

説明: 通常、このエラー条件は、特定のエレメントまたは属性の親 element_node に multi_occurrence = YES が指定されていない場合に発生します。

ユーザーの処置: DAD を調べて、親 element_node の multi_occurrence の値に、子 element_node の重複度が正しく反映されていることを確認してください。

DXXQ063E DAD ファイルの中で *elementname* の multi_occurrence 属性の値が無効です。

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、または欠落しています。その値は yes か no でなければなりません (大文字小文字を区別しません)。

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 列 *column* が外部表 *table* の中にありません。

説明: 結合条件に指定されたキー列が、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 ファイルの中で指定されている結合条件が正しいこと、またすべてのキー列がエレメント・ノードまたは属性ノードにマップされていることを確認してください。

DXXQ065I XML 対応の列に関連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 エクステンダーが表 *tablename* の主キーを判別できませんでした。列が XML 対応にされた後で、この表の主キーがドロップされた可能性があります。

ユーザーの処置: その列が XML 対応にされた時点で ROOT ID として指定されていた主キーを表に追加してください。

DXXQ067E *action* **を実行しようとして失敗しました。**

説明: *action* を実行しようとして、SQL エラーが発生しました。

ユーザーの処置: 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 エクステンダー・ トレース・ファイルを必ず含めてください。

第 5 部 付録

付録 A. サンプル

この付録では、本書の例で使用されているサンプル・オブジェクトを示します。

XML DTD のサンプル

本書全体を通して参照されている `getstart.xml` 文書で使用されている DTD は、以下のとおりです。

```
<!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 文書のサンプル: getstart.xml

以下の XML 文書 `getstart.xml` は、本書全体を通して例として使用されているサンプル XML 文書です。この文書には注文を表す XML タグが含まれています。

```
<?xml version="1.0"?>
<!DOCTYPE Order SYSTEM "dxxsamples/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。

- 『サンプル DAD ファイル: XML 列』
- 259 ページの『サンプル DAD ファイル: XML コレクション: SQL マッピング』は、 SQL マッピングを使用した XML コレクションの DAD ファイルを示しています。
- 261 ページの『サンプル DAD ファイル: XML: RDB_node マッピング』は、 RDB_node マッピングを使用した XML コレクションの DAD ファイルを示しています。

サンプル DAD ファイル: XML 列

この DAD ファイルには XML 列のマッピングが含まれており、 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  "dxxsamples/dtd/dad.dtd">
<DAD>
   <dtdid>
    "dxxsamples/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

サンプル DAD ファイル: XML コレクション: SQL マッピング

この DAD ファイルには、XML データを保管する DB2 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 "dxxsamples/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,
   (select db2xml.generate_unique()
     as ship_id, date, mode, part_key from ship_tab) ass
               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 "
dxxsamples/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)

サンプル DAD ファイル: XML: RDB_node マッピング

この DAD ファイルは、XML データを保管する DB2 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>
```

図 19. RDB_node マッピングを使用した XML コレクション用のサンプル DAD ファイル: order_rdb.dad (1/2)

```
          <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>
```

図 19. RDB_node マッピングを使用した XML コレクション用のサンプル DAD ファイル: order_rdb.dad (2/2)

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は、そのファイルにアクセスするクライアントまたはサーバーごとに、適切にエンコードされていなければなりません。 XML エクステンダーは、ファイルを処理する際の前提事項をいくつか設けているため、コード・ページがどのように変換されるかを理解している必要があります。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 パーサーによって処理される場合には、XML 文書のエンコード宣言も、文書の実際のエンコードとの間で整合がとれていなければなりません。
- ロケールが正しく構成されているかを確認します。

iSeries の場合、ジョブ、 DB2 UDB および XML 文書はすべて同じ CCSID を持っていなければなりません。次のセクションでは、CCSID に整合性を持たせるための方法について説明します。

ロケールの設定値の構成

XML エクステンダーは、ロケールの設定値に基づき、メッセージ・カタログから完了メッセージとエラー・メッセージを選択します。使用している言語でメッセージを受け取るには、 XML エクステンダーのメッセージ・カタログをインストールし、ロケールを正しくセットアップしなければなりません。 XML エクステンダーは、使用している言語のメッセージ・カタログを IFS ディレクトリー `/QIBM/ProdData/DB2Extenders/XML/MRI`*xxxx* (*xxxx* は言語コード) にインストールします。

例えば、英語 2924 のメッセージ・カタログは、ディレクトリー `/QIBM/ProdData/DB2Extenders/XML/MRI2924/dxx.cat` にインストールされます。 XML エクステンダーが英語 2924 のメッセージ・カタログを選択するようにするには、次のように **WRKUSRPRF** コマンドを使用して、ユーザー・プロファイルをセットアップします。

```
Language ID  . . . . . .  LANGID    ENU
Country/Region ID . . . . . .  CNTRYID   US
Coded character set ID . . . . .  CCSID    037
```

このユーザー・プロファイルで実行される XML エクステンダーのすべてのインスタンスは、 MRI2924 メッセージ・カタログを使用します。

XML エクステンダーで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・ページを指定し、XML 宣言ステートメントに記述されます。 XML エクステンダーを使用する場合は、文書のエンコードが、ジョブと DB2 に一致していることが重要です。

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、クライアントのエンコードと確実に整合させることが大切です。宣言で指定されているエンコード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、パーサーなどの XML ツールはエラーを生成するためです。

別々のコード・ページを使用すると、次のような状況が生じることがあります。

- 変換時にデータが消失する可能性があります。
- 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。

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。例えば、

```
<?xml version="1.0" encoding="UTF-8" ?>
```

不整合な XML 文書を防止するための推奨事項

XML 文書のエンコードをクライアントのコード・ページと確実に整合させるため、文書を構文解析プログラムなどの XML 処理プログラムに渡す前に、以下の推奨事項の 1 つを実行してください。

- XML エクステンダー UDF を使用してデータベースから文書をエクスポートするときには、以下の手法の 1 つを試行します (XML エクステンダーは、ファイルをサーバーのコード・ページで、サーバー上のファイル・システムにエクスポートすることを前提としています)。
  - 宣言されているエンコード・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、宣言された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 エクステンダーのストアード・プロシージャーを使用してデータベースから文書をエクスポートするときには、以下の手法の 1 つを試行します (クライアントは、合成文書が保管されている結果表を照会していることを前提とします)。
  - 宣言されているエンコード・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、宣言されたエンコードをオーバーライドする
  - ストアード・プロシージャーを実行する前に、クライアントに CCSID 変数を設定させて、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

付録 C. XML エクステンダーの制限

ここでは、以下に対する制限について説明します。

- XML エクステンダー・ オブジェクト
- ユーザー定義関数から戻される値
- ストアード・プロシージャーのパラメーター
- 管理サポート表の列
- 合成と分解

次の表は、XML エクステンダー・ オブジェクトの制限を示しています。

表 67. XML エクステンダー・ オブジェクトの制限

オブジェクト	制限
表内の分解 XML コレクション内の最大行数	分解されたそれぞれの XML 文書から 10240 行
デフォルトのビューに指定される列名での最大文字数	10 文字
パラメーター値として指定される XML ファイル・パス名での最大バイト数。	512 バイト
SQL 合成の場合の 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 分解の場合に DAD ファイルの 1 つの表について指定される、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 および attribute_node エレメントで指定されます。)

次の表は、XML エクステンダーのユーザー定義関数から戻される制限値について説明したものです。

表 68. ユーザー定義関数の制限

ユーザー定義関数の戻り値	制限
extractCHAR UDF が戻す最大バイト数	254 バイト
extractCLOB UDF が戻す最大バイト数	2 G バイト
extractVARCHAR UDF が戻す最大バイト数	4 K バイト

次の表は、XML エクステンダーのストアード・プロシージャーのパラメーターの制限を説明したものです。

表 69. ストアード・プロシージャーのパラメーターの制限

ストアード・プロシージャーのパラメーター	制限
XML 文書 CLOB[1] の最大サイズ	1 MB
文書アクセス定義 (DAD) CLOB[1] の最大サイズ	100 KB

表 69. ストアード・プロシージャーのパラメーターの制限 (続き)

ストアード・プロシージャーのパラメーター	制限
collectionName の最大サイズ	30 バイト
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	16366 バイト

注:

1. このサイズは、dxxGenXMLClob および dxxRetrieveXMLCLOB については変更することができます。
2. *tbName* パラメーターの値をスキーマ名で修飾する場合、名前全体の長さ (区切り文字も含む) を 128 バイト以下にする必要があります。

次の表は、DB2XML.DTD_REF 表の制限について説明したものです。

表 70. XML エクステンダーの制限

DB2XML.DTD_REF 表の列	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側のサイズの制限内に収まっていても、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、合成および分解の制限を示しています。

表 71. XML エクステンダーでの合成および分解の制限

オブジェクト	制限
分解 XML コレクションで表内に挿入される最大行数	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 内の name 属性の最大長	63 バイト
デフォルトのビューに指定される列名での最大文字数	10 文字

表 71. XML エクステンダーでの合成および分解の制限 (続き)

オブジェクト	制限
パラメーター値として指定される XMLFile パス名での最大バイト数	512 バイト

DB2DXX_MIN_TMPFILE_SIZE 環境変数:

XML エクステンダーは、処理中のメモリー使用量が大きくなり過ぎないようにするため、大きな文書を一時ファイルの中に入れることがあります。大容量の物理メモリーを搭載しているシステムの場合、文書が一時ファイルに入れられないようにすることにより、入出力の処理量を少なくすることができます。環境変数 DB2DXX_MIN_TMPFILE_SIZE を使用すれば、指定した値より小さい文書を処理する場合に、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 XML エクステンダーに対して指示できます。この変数が適用されるのはサーバーだけであり、クライアントでは適用されません。 1 つのマルチノード・パーティションの中に複数の物理ノードが関係している場合、この変数には、各マシンにインストールされているメモリー量に応じてノードごとに異なる値を設定できます。この環境変数を設定しない場合には、処理中に 128 KB を超える文書は自動的に一時ファイルに入れられ、 128 KB より小さい文書はメモリー内で処理されます。

XML エクステンダー用語集

絶対ロケーション・パス (absolute location path). オブジェクトの全パス名。絶対パス名は最上位つまり「ルート」エレメントから始まり、これはスラッシュ (/) または円記号 (¥) 文字によって識別される。

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、 Text Extender で処理できるタイプに変換する、ユーザー提供の関数。

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 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表の 1 つ。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。

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。

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。

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。

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、 API によりオペレーティング・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用することができる。

(2)DB2 では、インターフェース内の機能 (例えば、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 Text Extender 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。例えば、エラー・メッセージ取得 API がある。

属性 (attribute). *XML 属性 (XML attribute)* を参照。

attribute_node. あるエレメントの属性を表したもの。

2 進ラージ・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。

ブール検索 (Boolean search). 1 つ以上の言葉をブール演算子を使用して結合する検索。

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。

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。

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、 Text Extender の機能の 1 つ。*Web ブラウザー (Web browser)* も参照。

B-tree 索引付け (B-tree indexing). DB2 UDB エンジンが提供する固有の索引体系。索引項目を B-tree 構造で作成する。これは DB2 基本データ・タイプをサポートしている。

cast 関数 (cast function). ある (ソース) データ・タイプのインスタンスを、別の (ターゲット) データ・タイプのインスタンスに変換する関数。一般に、cast 関数にはターゲット・データ・タイプの名前が付いている。この関数の引き数は 1 つで、そのタイプはソース・データ・タイプ。戻されるタイプはターゲット・データ・タイプ。

カタログ・ビュー (catalog view). 管理目的のために Text Extender が作成するシステム表のビュー。カタログ・ビューには、 Text 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。

CCSID. コード化文字セット ID (Coded Character Set Identifier)。

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。

CLOB. 文字ラージ・オブジェクト (character large object)。

コード・ページ (code page). すべてのコード・ポイントに図形文字および制御機能の意味を割り当てたもの。例えば、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。

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。

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 Text 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

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。

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。

DAD. *文書アクセス定義 (document access definition)* を参照。

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プロセスが必要ない。

データ・ソース (data source). ODBC API をサポートする ODBC ドライバーを介してデータにアクセスすることが可能な、ローカルまたはリモートのリレーショナル・データ・マネージャーまたは非リレーショナル・データ・マネージャー。

データ・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、および見つ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・タイプ (data type). 列やリテラルの属性。

データベース区画 (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。

データベース区画サーバー (database partition server). *データベース区画 (database partition)* を管理するサーバー。データベース区画サーバーは、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 つのデータベース区画サーバーが 1 台のマシンに割り当てられる。

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)。

分解する (decompose). XML を分離して、複数のリレーショナル表からなる XML コレクションにすること。

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。

使用不可にする (disable). 使用可能プロセス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 エクステンダーで使用可能になる前の状態に復元すること。

特殊タイプ (distinct type). *ユーザー定義タイプ (user-defined type)* を参照。

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用される。これによって、 XML エクステンダーの列を (XML 形式の) XML コレクションにすることができる。

文書タイプ定義 (DTD) (document type definition (DTD)). 複数の XML エレメントおよび属性の宣言の集合。DTD は、XML 文書内でどのようなエレメントがどんな順序で使用されるか、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。

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。

DTD. *文書タイプ定義 (document type definition)* を参照。

DTD 参照表 (DTD_REF 表) (DTD reference table (DTD_REF table)). DTD を含んでいる表。DTD は、XML 文書の妥当性検査、およびアプリケーションによる DAD ファイルの定義に使用される。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 使用可能になるときに作成される。

DTD_REF 表 (DTD_REF table). DTD 参照表 (DTD reference table)。

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。

EDI. 電子データ交換 (Electronic Data Interchange)。

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。

エレメント (element). *XML エレメント (XML element)* を参照。

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。

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。

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 エクステンダーで使用できるようにすること。

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。

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。

拡張可能スタイル・シート言語 (XSL) (Extensible Stylesheet language (XSL)). スタイルシートを表すために使用される言語。 XSL は、XML 文書を変換する言語と、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。

Extensible Stylesheet Language Transformation (XSLT). XML 文書を別の XML 文書に変換するために使用される言語。 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ている。

外部ファイル (external file). DB2 の制御する表の中の 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。

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。

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。

関数 (function). *アクセス関数 (access function)* を参照。

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。

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。

イメージ (image). 絵や写真を電子的に表したもの。

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集合。索引は、データに迅速にアクセスするのに使用され、表内の行を固有化することができる。

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)。

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。

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。

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・シーケンス。 LOB のタイプは、 *2 進ラージ・オブジェクト (binary large object (BLOB))*、*文字ラージ・オブジェクト (character large object (CLOB))*、および *2 バイト文字ラージ・オブジェクト (double-byte character large object (DBCLOB))* の 3 つ。

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。例えば “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。

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。

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 ファイルでは 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、Text 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 の位置指定に使用されるデータ・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。

マッピング体系 (mapping scheme). XML データをリレーショナル・データベース内で表す方法についての定義。マッピング体系は DAD で指定される。 XML エクステンダーの提供するマッピング体系には、*SQL マッピング* とリレーショナル・データベース・ノード *(RDB_node)* マッピング の 2 つがある。

メガバイト (megabyte, MB). 百万 (10^6) 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。

複数出現 (multiple occurrence). 1 つの文書内で列エレメントまたは属性を 2 度以上使用できるかどうかを示す。複数出現は DAD 内で指定される。

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。

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。

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。

パス式 (path expression). *ロケーション・パス (location path)* を参照。

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。

基本キー (primary key). 表の定義の一部である固有キー。基本キーは、参照制約定義のデフォルトの親キーである。

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。

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。

照会オブジェクト (query object). QBIC 照会用のフィーチャー、フィーチャーの値、およびフィーチャー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。

RDB_node. リレーショナル・データベース・ノード。

RDB_node マッピング (RDB_node mapping). XML エレメントの内容または XML 属性の値の位置で、RDB_node によって定義される。 XML エクステンダーはこのマッピングを使用して、 XML データを保管または検索する場所を判別する。

リレーショナル・データベース・ノード (RDB_node) (relational database node (RDB_node)). 表、オプションの列、およびオプションの条件の定義を 1 つ以上含んでいるノード。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用される。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。

結果セット (result set). ストアード・プロシージャーによって戻された行の集合。

結果表 (result table). SQL 照会またはストアード・プロシージャーの実行の結果として戻された行を含む表。

ルート・エレメント (root element). XML 文書の先頭エレメント。

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固有 ID。

SBCS. 1 バイト文字サポート。

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。

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。

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。

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 (例えば、ショットの開始フレーム番号と終了フレーム番号) の保管に使用さ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。

サイド表 (side table). XML 列内のエレメントや属性を検索する際のパフォーマンスを改善するために XML エクステンダーが作成する追加の表。

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。

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 エクステンダーはこの定義を使用して、XML データを保管または検索する場所を判別する。 SQL マッピングは、SQL_stmt エレメントとともに DAD 内で定義される。

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。

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。

構造化テキスト索引 (structural text index). DB2 UDB Text Extender を使用して、 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。

副照会 (subquery). SQL ステートメントの検索条件の中で使用される SELECT ステートメント全体。

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。

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。

text_node. あるエレメントの CDATA テキストを表したもの。

テキスト表 (text table). テキスト*列 (text columns)* を含んでいる DB2 UDB 表。

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。

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。

トリガー (trigger). 表が変更されたときに実行される一連のアクションの定義。トリガーを使用し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。

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。例えば、http://www.ibm.com/software /data/db2/extenders.

UNION. 2 つの SELECT ステートメントの結果を結合する SQL 操作。 UNION は、複数の表から得られた値のリストをマージするためにしばしば使用される。

URL. uniform resource locator。

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。例えば、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。 Text Extender は、CONTAINS のような取り出し関数を UDF の形で提供する。

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。例えば、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。

ユーザー定義関数 (user-defined function (UDF)). データベース管理システムに定義される関数で、定義後は SQL 照会で使用できるようになる。以下のいずれかの関数である。

- 外部関数。関数の本体はスカラー値を引き数とするプログラミング言語で書かれ、呼び出すたびにスカラー結果が生成される。
- ソース関数。すでに DBMS に認識されている別の組み込み関数またはユーザー定義関数によって実装される。この関数は、スカラー関数または列 (集合) 関数のいずれかで、値のセットから単一の値 (MAX、AVG など) を戻す。

ユーザー定義タイプ (user-defined type (UDT)). データベース・マネージャーに元々あったものではなく、ユーザーにより作成されたデータ・タイプ。*特殊タイプ (distinct type)* を参照。

ユーザー表 (user table). アプリケーション用に作成され、アプリケーションによって使用される表。

妥当性検査 (validation). DTD を使用して、XML 文書の有効性を検査したり、 XML データの構造化検索を行うプロセス。 DTD は DTD リポジトリーに保管される。

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。

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・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。

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 Video Extender が使用するファイル。

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。

ワイルドカード文字 (wildcard character). マスク文字 *(masking character)* を参照。

XML. eXtensible Markup Language。

XML 属性 (XML attribute). DTD において、XML エレメントの下の ATTLIST で指定されているすべての属性。 XML エクステンダーはロケーション・パスを使用して属性を識別する。

XML コレクション (XML collection). 関係表の集合であり、XML 文書の合成に使用するデータ、または XML 文書から分解されたデータを提供する。

XML 列 (XML column). XML エクステンダー UDT に使用できるアプリケーション表内の列。

XML エレメント (XML element). XML DTD で定義されている、すべての XML タグまたは ELEMENT。 XML エクステンダーはロケーション・パスを使用してエレメントを識別する。

XML オブジェクト (XML object). 「XML 文書 (XML document)」と同じ。

XML 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 表 (XML table). 1 つ以上の XML エクステンダー列を含んでいるアプリケーション表。

XML 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 UDF. XML エクステンダーの提供する DB2 UDB ユーザー定義関数。

XML UDT. XML エクステンダーの提供する DB2 UDB ユーザー定義タイプ。

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). ノードを使用して XML 文書をモデル化およびナビゲートするのに使用するツリー構造。

XSL. XML スタイル・シート言語 (XML Stylesheet Language)。

XSLT. XML スタイル・シート言語トランスフォーメーション (XML Stylesheet Language Transformation)。

索引

日本語, 数字, 英字, 特殊文字の順に配列されています。なお, 濁音と半濁音は清音と同等に扱われています。

[ア行]

[カ行]

[サ行]

[タ行]

[ナ行]

[ハ行]

[マ行]

[ヤ行]

[ラ行]

A

B

C

D

E

F

G

I

J

M

O

R

S

T

U

W

X

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 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。

プログラム番号: 5722-DE1

Printed in Japan

SC88-4030-00

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