IBM DB2 10.1
for Linux, UNIX, and Windows

pureXML ガイド

SA88-4686-00
(英文原典：SC27-3892-00)

IBM DB2 10.1
for Linux, UNIX, and Windows

pureXML ガイド

SA88-4686-00
(英文原典：SC27-3892-00)

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、 539 ページの『付録 E. 特記事項』に記載されている情報をお読みください。

IBM 資料は、オンラインでご注文いただくことも、ご自分の国または地域の IBM 担当員を通してお求めいただくこともできます。

- オンラインで資料を注文するには、IBM Publications Center (http://www.ibm.com/shop/publications/order) をご利用ください。
- ご自分の国または地域の IBM 担当員を見つけるには、IBM Directory of Worldwide Contacts (http://www.ibm.com/planetwide/) をお調べください。

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。

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

原典： SC27-3892-00
IBM DB2 10.1
for Linux, UNIX, and Windows
pureXML Guide

発行： 日本アイ・ビー・エム株式会社

担当： トランスレーション・サービス・センター

第1刷 2012.4

目次

本書について

「pureXML® ガイド」では、DB2® データベースで XML データを操作する方法について説明します。 XML データ・タイプと XML ストレージについて、SQL 言語と XQuery 言語を使って XML データを操作する方法について、およびパフォーマンスのために XML データの索引を作成する方法について説明しています。さらに、pureXML アプリケーション開発、データ移動、および XML データからリレーショナル形式への分解について扱ったトピックも含まれています。

第 1 章 pureXML の概要 -- XML データベースとしての DB2

pureXML を使用すると、整形式 XML 文書を XML データ・タイプのデータベース表列に保管できます。XML データを XML 列に保管すると、データがテキストとして保管されたり、異なるデータ・モデルにマップされたりすることはなく、ネイティブの階層形式のままで保持されます。

pureXML データ・ストレージは完全に統合されているので、保管された XML データは既存の DB2 データベース・サーバー機能を活用してアクセスおよび管理が可能です。

ネイティブの階層形式の XML データのストレージによって、効率的な XML の検索、取り出し、および更新ができるようになります。 XQuery、SQL、または両方の組み合わせを、XML データを照会および更新するのに使用することができます。さらに、XML データを戻したり、XML 引数を取る SQL 関数 (SQL/XML 関数と呼ばれる) によって、XML データを構成したり、データベースから検索された値から XML データを発行したりすることも可能になります。

照会および更新

XML 列に保管された XML 文書は、以下のメソッドを使用して照会および更新できます。

XQuery

XQuery は XML データの解釈、取り出し、および変更を行うための汎用言語です。 DB2 データベース・サーバーを使用すると、XQuery を直接、または SQL 内から呼び出すことができます。XML データは DB2 表またはビューに保管されているので、表またはビューの名前を直接指定したり、SQL 照会を指定することによって、XML データを指定された表またはビューから抽出する関数が提供されています。 XQuery は、XML データの処理や、要素および属性などの既存の XML オブジェクトの更新、および新規 XML オブジェクトの構成のために、さまざまな式をサポートします。XQuery のプログラミング・インターフェースは、照会を実行し、結果を取得するための SQL の機能に似た機能を提供します。

SQL ステートメントおよび SQL/XML 関数

多くの SQL ステートメントが、XML データ・タイプをサポートしています。そのため、XML 列を使って表を作成する、既存の表に XML 列を追加する、XML 列の索引を作成する、XML 列を使って表にトリガーを作成する、および XML 文書を挿入、更新、または削除するなどの、XML データを使った多くの一般的なデータベース操作を実行できるようになります。DB2 データベース・サーバーによってサポートされる SQL/XML 関数、式、および仕様のセットは、XML データ・タイプを十分に活用するために拡張されました。

XQuery は SQL 照会内から呼び出すことができます。この場合、SQL 照会はデータを、バインドされた変数の形で XQuery に渡します。

アプリケーション開発

アプリケーション開発のサポートは複数のプログラミング言語で提供され、SQL および外部プロシージャーを介しても提供されます。

プログラミング言語のサポート

新規の pureXML のアプリケーション開発サポートによって、アプリケーションは XML とリレーショナル・データのアクセスおよびストレージを組み合わせることができます。以下のプログラミング言語は、XML データ・タイプをサポートします。

- C または C++ (組み込み SQL または CLI)
- COBOL
- Java (JDBC または SQLJ)
- C# および Visual Basic (IBM® Data Server Provider for .NET)
- PHP
- Perl

SQL および外部プロシージャー

CREATE PROCEDURE パラメーターのシグニチャーにデータ・タイプ XML のパラメーターを含めることによって、XML データを SQL プロシージャーおよび外部プロシージャーに渡すことができます。既存のプロシージャー機能は、XML データ値の変数への一時格納に加えて、XML 値を生成または利用する SQL ステートメントのプロシージャー型ロジックの構成をサポートします。

管理

pureXML により、XML 文書の URI 従属関係の管理に関するリポジトリーが提供され、データベース管理のために XML データ移動が可能になります。

XML スキーマ・リポジトリー (XSR)

XML スキーマ・リポジトリー (XSR) は、XML 列に保管された XML インスタンス文書を処理するために必要とされるすべての XML 作成物のリポジトリーです。これは、XML 文書で参照される XML スキーマ、DTD、および外部エンティティーを保管します。

インポート、エクスポート、およびロード・ユーティリティー

インポート、エクスポート、およびロード・ユーティリティーは、ネイティブ XML データ・タイプをサポートするように更新されました。これらのユーティリティーは、XML データを LOB データのように扱います (どちらのタイプのデータも、元となる実表とは別の場所に保管されています)。XML データのインポート、エクスポート、およびロードのためのアプリケーション開発サポートが、更新された db2Import、db2Export、および db2Load API でも提供されています。これらの更新されたユーティリティーは、XML 列に保管された XML 文書のデータ移動を可能にします。これは、リレーショナル・データのデータ移動サポートに類似しています。

パフォーマンス

XML 列に保管されている XML 文書を処理する際に利用できる、パフォーマンスを向上させる機能がいくつかあります。

XML データの索引

XML 列に保管されているデータへの索引作成をサポートします。 XML データの索引を使用すると、XML 文書に対して発行される照会の効率を向上できます。リレーショナル索引 と同様に、XML データに対する索引 は列を索引付けします。ただし、リレーショナル索引が列全体に索引を付けるのに対して XML データに対する索引は列の一部に索引を付ける点が異なります。 XML 列のどの部分に索引を付けるかを、XML パターン (限定された XPath 式) を指定することによって指示してください。

オプティマイザー

オプティマイザーは、XML およびリレーショナル・データに対して、SQL、XQuery、および XQuery が組み込まれた SQL/XML 関数の評価をサポートするように更新されました。オプティマイザーは、効率的な照会の実行プランを作成するために、XML データについて集められた統計、さらには XML データの索引からのデータについて集められた統計を活用します。

EXPLAIN および Visual Explain

EXPLAIN 機能は、XML データを照会する SQL 機能拡張と XQuery 式をサポートするように更新されました。この EXPLAIN 機能の更新により、XML データに対する照会ステートメントを DB2 データベース・サーバーがどのように評価するか、すぐに調べることができます。

ツール

XML データ・タイプのサポートは、コントロール・センター、コマンド行プロセッサー、IBM Data Studio、および IBM Database Add-Ins for Microsoft Visual Studio などのツールで使用できます。

アノテーション付き XML スキーマ分解

pureXML では、XML データを XML として (階層形式で) 保管およびアクセスできるようになりますが、XML データにリレーショナル・データとしてアクセスすることが必要な場合もあります。アノテーション付き XML スキーマ分解は、XML スキーマに指定されたアノテーションに基づいて文書をリレーショナル・データに分解します。

XML データ・タイプ

XML データ・タイプは、XML 値を保管する表の列を定義するために使用され、すべての保管された XML 値は整形式 XML 文書になる必要があります。このネイティブ XML データ・タイプが導入されることにより、整形式 XML 文書を他のリレーショナル・データとともにデータベースにネイティブの階層形式で保管する機能が提供されます。

XML 値はストリングではない内部表記で処理され、ストリング値とは直接比較できません。XML 値は、XMLSERIALIZE 関数を使用するか、または値を XML、ストリング、またはバイナリー形式のアプリケーション変数にバインドすることにより、XML 文書を表すシリアライズされたストリング値に変換できます。同様に、XML 文書を表すストリング値は、XMLPARSE 関数を使用するか、またはアプリケーション・ストリング、バイナリー、または XML アプリケーション・タイプを

XML 値にバインドすることにより、XML 値に変換できます。XML 列が関係する SQL データ変更ステートメント (INSERT など) では、XML 文書を表すストリングまたはバイナリー値は、挿入された XMLPARSE 関数を使用して XML 値に変換されます。XML 値は、アプリケーションのストリングおよびバイナリーのデータ・タイプとの交換時に、暗黙で解析またはシリアライズ化することができます。

データベース内の XML 値のサイズには、設計上の限度はありません。ただし、DB2 データベース・サーバーと交換されるシリアライズされた XML データは事実上 2 GB に制限されます。

XML 文書は、SQL データ操作ステートメントを使用して挿入、更新、および削除できます。通常は挿入または更新中に実行される、XML スキーマに照らした XML 文書の妥当性検査は、XML スキーマ・リポジトリー (XSR) によりサポートされます。DB2 データベース・システムは、XML 値を構成して照会するためのメカニズムに加えて、XML データをエクスポートおよびインポートするためのメカニズムも提供します。XML 列を元に XML データに対する索引を定義することができ、これにより XML データの検索パフォーマンスが向上します。表またはビューの列の XML データは、さまざまなアプリケーション・インターフェースを介して、シリアライズされたストリング・データとして取得できます。

XML 入出力の概要

リレーショナル・データと XML データの両方を管理する DB2 データベース・サーバーは、XML 文書の入出力のためのさまざまな方法を提供します。

XML 文書は、XML データ・タイプで定義される列内に保管されます。 XML 列の各行は、単一の整形式 XML 文書を保管します。保管された文書は、階層形式で保持され、XML データ・モデルを保ちます。文書は、テキストとして保管されたり、異なるデータ・モデルにマップされることはありません。

XML 列は、リレーショナル・データを保持する、他のタイプの列を含む表で定義することができます。複数の XML 列を単一の表に対して定義することもできます。

入力

5 ページの図 1 は、XML データをデータベース・システムに書き込むさまざまな方法を示しています。

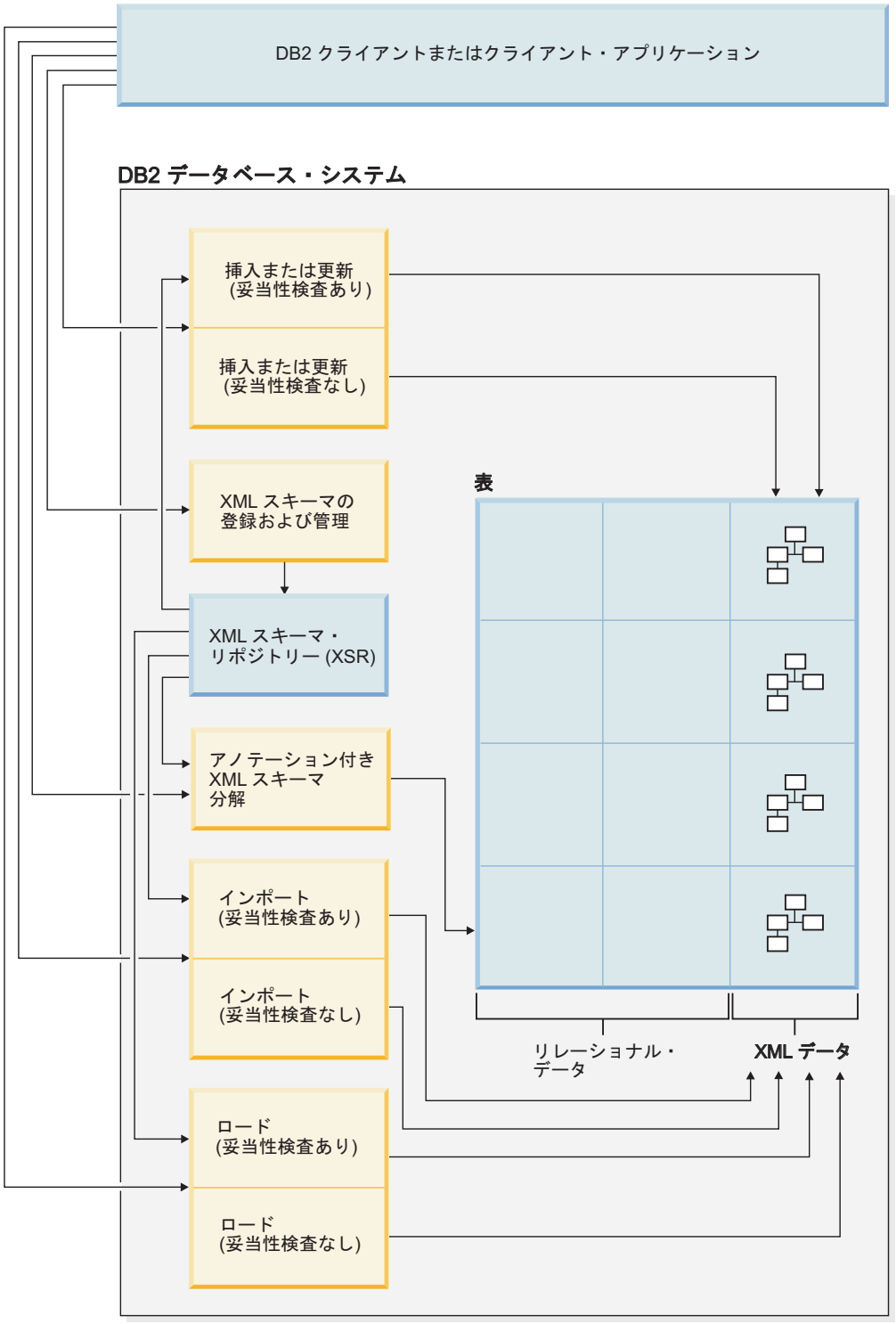


図 1. XML データの入力方法

使用する入力方法は、実行するタスクによって異なります。

挿入または更新

整形式文書は、INSERT SQL ステートメントを使用して、XML 列に挿入されます。構文解析が正常に行われると、文書は整形式になります。挿入または更新操作時の XML 文書の妥当性検査はオプションです。妥当性検査を

実行する場合、まず XML スキーマを XML スキーマ・リポジトリー (XSR) に登録する必要があります。UPDATE SQL ステートメント、または XQuery 更新式を使用して、文書を更新します。

アノテーション付き XML スキーマ分解

XML 文書からのデータは、アノテーション付き XML スキーマ分解を使用して、分解したり、リレーショナル列および XML 列に保管したりすることができます。分解によって、XML スキーマ文書に追加されたアノテーションに従って、データを列に保管します。これらのアノテーションは、XML 文書内のデータを表の列にマップします。

分解フィーチャーにより参照される XML スキーマ文書は、XML スキーマ・リポジトリー (XSR) に保管されます。

インポート

XML 文書は、インポート・ユーティリティーを使用して XML 列にインポートできます。インポートする XML 文書の妥当性検査はオプションです。妥当性検査を実行する場合、文書の妥当性検査に使用する XML スキーマをまず XML スキーマ・リポジトリー (XSR) に登録する必要があります。

XML スキーマ・リポジトリー (XSR) 登録

XML スキーマ・リポジトリー (XSR) は、XML 文書の妥当性検査または分解に使用する XML スキーマを保管します。通常 XML スキーマの登録は、こうしたスキーマと従属関係を持つ XML 文書で実行される他のタスクの前提条件となります。XML スキーマは、ストアード・プロシージャーまたはコマンドを使用して、XSR に登録されます。

出力

7 ページの図 2 は、XML データをデータベース・システムから取得するさまざまな方法を示しています。

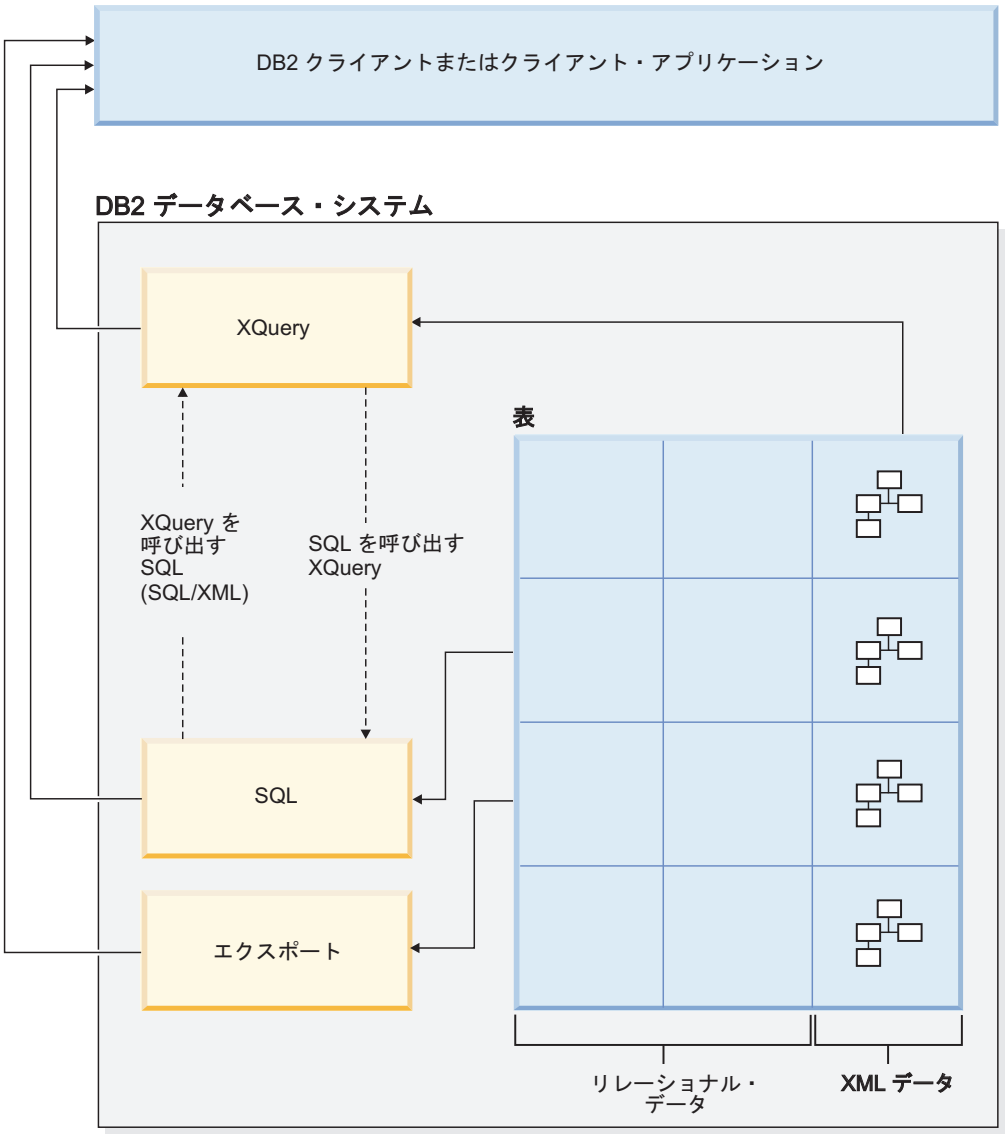


図 2. XML データの出力方法

使用する出力方法は、実行するタスクによって異なります。

XQuery

XQuery は、XML 文書内での照会を可能にする言語です。これは予測可能な構造を期待するリレーショナル・データに対する照会とは異なり、構造が非常に変わりやすい XML データを照会するための固有の要件に対応します。

DB2 データベースに保管されている XML を照会するため、XQuery を単体で呼び出すことも、SQL を呼び出すこともできます。これは db2-fn:xmlcolumn および db2-fn:sqlquery XQuery 関数によって行えます。db2-fn:xmlcolumn は、XML 列全体を検索します。一方 db2-fn:sqlquery は、SQL 全選択に基づく XML 値を検索します。

SQL SQL 全選択を使用して XML データを照会する場合、照会は列レベルで実行されます。このため、照会によって戻すことができるのは XML 文書の全体だけとなります。SQL だけを使用して XML 文書の断片を戻すことは

できません。 XML 文書内で照会を行うには、XQuery を使用する必要があります。 XQuery は、XMLQUERY または XMLTABLE SQL/XML 関数、あるいは XMLEXISTS 述部を使用して、SQL から呼び出すことができます。 XMLQUERY 関数は、XQuery 式の結果を XML シーケンスとして戻します。 XMLTABLE 関数は、XQuery 式の結果を表として戻します。 XMLEXISTS SQL 述部は、XQuery 式が空でないシーケンスを戻すかどうかを判別します。

また DB2 データベース・サーバーに保管されている XML データから XML 値を構成するために使用できる発行関数が多数あります。こうした発行関数によって構成される XML 値を整形式 XML 文書にする必要はありません。

エクスポート

XML 文書は、エクスポート・ユーティリティーを使用して XML 列からエクスポートできます。エクスポートされた XML データは、メイン・データ・ファイル内のエクスポート・リレーショナル・データとは別個に保管されます。各エクスポート XML 文書に関する詳細は、メイン・エクスポート・データ・ファイルに直接保管されることはありません。代わりに詳細は、メイン・データ・ファイル内で XML データ指定子 (XDS) によって表されます。

XML モデルとリレーショナル・モデルとの比較

データベースを設計するとき、扱うデータが XML モデルまたはリレーショナル・モデルのどちらにより適しているかを判別する必要があります。 DB2 データベースのハイブリッド性、つまり単一のデータベースでリレーショナル・データと XML データの両方をサポートするという利点を設計に取り入れることができます。

この解説では、これらのモデル間のいくつかの主な相違点およびそれぞれに適用される要素について説明しますが、ご使用の実装環境に最適な選択を決めるための要素は多数あります。この解説を指針として使用しながら、特定の実装環境に影響を与える可能性のある要素を評価してください。

XML データとリレーショナル・データとの主な相違点

XML データは階層構造で、リレーショナル・データは論理関係のモデルで表される。 XML 文書にはデータ項目の相互関係に関する情報が、階層の形式で含まれています。リレーショナル・モデルでは、定義可能な関係のタイプは親表と従属表との関係だけです。

XML データは自己記述型であり、リレーショナル・データはそうではない。 XML 文書にはデータだけではなく、そのデータの内容を説明するタグも含まれています。単一の文書にさまざまなタイプのデータを入れることができます。リレーショナル・モデルでは、データの内容は列定義によって定義されます。 1 つの列内のすべてのデータは同じタイプのデータでなければなりません。

XML データには特有の順序付けがあるが、リレーショナル・データにはそれがない。 XML 文書では、データ項目が指定される順序は文書内のデータの順序と同じであると想定されます。多くの場合、文書内での順序を指定する別の方法

はありません。リレーショナル・データでは、1 つ以上の列に対して ORDER BY 節を指定しなければ、行の順序は保証されません。

データ・モデルの選択に影響する要因

保管するデータの種類が決まると、データの保管方法が判別できます。例えば、データが元々階層的であり自己記述型であれば、それを XML データとして保管できます。ただし、どのモデルを使用するかを決める際には他の要因も影響することがあります。

柔軟性が最も必要とされる場合

リレーショナル表はかなり固定的なモデルになります。例えば、1 つの表を正規化して多数の表にしたり、多数の表を非正規化して 1 つの表にすることは非常に困難です。データ設計が頻繁に変更される場合は、それを XML データで表現する方が優れた選択となります。例えば、XML スキーマは時間とともに発展させることが可能です。

データ取得のために最高のパフォーマンスが必要となる場合

XML データのシリアライズおよび解釈には、いくらかの費用が余分にかかります。柔軟性よりもパフォーマンスが重要である場合、リレーショナル・データの方が優れた選択肢となることがあります。

データが後にリレーショナル・データとして処理される場合

後続のデータ処理が、データがリレーショナル・データベースに保管されているということに依存する場合、分解を使用して、データの一部をリレーショナルとして保管することが適切な場合があります。この状態の一例は、オンライン分析処理 (OLAP) がデータウェアハウス内のデータに適用される場合です。また、XML 文書の全体に対して他の処理が必要な場合は、XML 文書の全体を保管することに加えて、データの一部をリレーショナルとして保管することが適切な方法であることがあります。

データ・コンポーネントが階層の外部で意味を持つ場合

データは元々階層的な性質を持っていることがありますが、子コンポーネントは親から値を提供される必要がありません。例えば、購入注文にはパーツ・ナンバーが含まれることがあります。パーツ・ナンバーのある購入注文は、XML 文書として表現するのが最適かもしれません。ところが、各パーツ・ナンバーにはパーツ記述が関連付けられています。パーツ記述はリレーショナル表に含めた方が良いかもしれません。なぜなら、パーツ・ナンバーとパーツ記述との間の関係は、パーツ・ナンバーが使用される購入注文とは論理的に独立しているからです。

データ属性がすべてのデータに適用されるか、またはデータの小さなサブセットだけに適用される場合

考えられる多数の属性を持つデータ・セットもありますが、それらの属性のうち、少数のみが特定のデータ値に適用されます。例えば、小売カタログでは、サイズ、色、重さ、素材、スタイル、織り方、消費電力、燃料の所要量など、考えられるデータ属性が多数あります。カタログ内のどのアイテムにせよ、関係があるのはそれらの属性のサブセットに過ぎません。消費電力はテーブル型電動鋸には対しては意味がありますが、外套に対しては意味がありません。このタイプのデータはリレーショナル・モデルでは表現および検索が困難ですが、XML モデルでは表現および検索が比較的容易になります。

ボリュームに対するデータの複雑さの比率が高い場合

高度に構造化された、非常に少量の情報が関係する状況が多くあります。そのようなデータをリレーショナル・モデルで表現すると、複雑なスター・スキーマが関係することになり、その中の各ディメンション表が更に多くのディメンション表に結合し、ほとんどの表には少数の行しかないという事態になる可能性があります。このデータを表現するためのより優れた方法は、XML 列のある単一の表を使用して、その表に複数のビューを作成し、各ビューが 1 つのディメンションを表すようにすることです。

参照整合性が必要な場合

XML 列を参照制約の一部として定義することはできません。そのため、XML 文書内の値を参照制約に含める必要がある場合は、データをリレーショナル・データとして保管してください。

データを頻繁に更新する必要がある場合

XML 列内の XML データを更新するには、文書全体を置き換えます。非常に大きい文書の小さな断片を多数の行に関して頻繁に更新する必要がある場合、データを XML 以外の列に保管する方が効率的です。ただし、更新するのが小さな文書であり、一度に少数の文書だけを更新する場合には、XML として保管しても同様に効率的なものになり得ます。

XQuery および XPath のデータ・モデル

XQuery 式は、XQuery および XPath のデータ・モデル (XDM) のインスタンスに対して作動し、データ・モデルのインスタンスを戻します。XDM では、1 つ以上の XML 文書またはフラグメントの要約表記を使用します。データ・モデルは、中間計算時に使用される値を含め、XQuery における式のすべての暗黙的値を定義します。

XML データの XDM への構文解析、およびスキーマに対するデータの妥当性検査は、データが XQuery で処理される前に行われます。データ・モデルの生成時に、入力 XML 文書は解析され、XDM のインスタンスに変換されます。文書は、妥当性検査の有無にかかわらず構文解析できます。

XDM は、アトミック値およびノードのシーケンスで説明されます。

シーケンスおよび項目

XQuery および XPath データ・モデル (XDM) のインスタンスはシーケンスです。*シーケンス* は、 0 個以上の項目の順序付けられたコレクションです。*項目* は、アトミック値またはノードです。

シーケンスには、ノード、アトミック値、またはノードとアトミック値を混合させたものを含めることができます。例えば、以下のリストの各項目はシーケンスです。

- `36`
- `<dog/>`
- `(2, 3, 4)`
- `(36, <dog/>, "cat")`
- `()`

リスト内の項目に加えて、DB2 データベース内の XML 列に保管されている XML 文書もシーケンスです。

この例では、XQuery でシーケンスを構成するために使用する構文と整合した、シーケンスを表すための記法を使用しています。

- シーケンス内の各項目は、コンマで区切ります。
- シーケンス全体は括弧で囲みます。
- 一対の空の括弧は、空のシーケンスを表します。
- 単独で表示される単一の項目は、1 つの項目を含むシーケンスと等価です。

  例えば、シーケンス (36) とアトミック値 36 との間に区別はありません。

シーケンスはネストできません。2 つのシーケンスが結合されると、結果は常にノードとアトミック値のフラット化されたシーケンスです。例えば、シーケンス (2, 3) をシーケンス (3, 5, 6) に付加した結果は、単一のシーケンス (3, 5, 6, 2, 3) です。これらのシーケンスを結合しても、ネストされたシーケンスは存在し得ないため、シーケンス (3, 5, 6, (2, 3)) は作成されません。

0 個の項目を含むシーケンスを空のシーケンス と呼びます。空のシーケンスは、欠落した、または不明の情報を表すために使用できます。

アトミック値

アトミック値 とは、XML スキーマで定義されている組み込みアトミック・データ・タイプの 1 つのインスタンスです。これらのデータ・タイプには、ストリング、整数、10 進数、日付、その他のアトミック・タイプが含まれています。これらのタイプは、それ以上分割できないため、アトミックとして記述されます。

ノードとは異なり、アトミック値には ID がありません。アトミック値の各インスタンス (整数 7 など) は、その値の他のすべてのインスタンスと同一です。

以下は、アトミック値の作成方法の例です。

- 原子化と呼ばれるプロセスを使用して、ノードから抽出する。原子化は、アトミック値のシーケンスが必要な式で使用されます。
- 数値リテラルまたはストリング・リテラルとして指定する。リテラルは、XQuery によってアトミック値として解釈されます。例えば、以下のリテラルは、アトミック値として解釈されます。
  - "this is a string" (xs:string タイプです)
  - 45 (xs:integer タイプです)
  - 1.44 (xs:decimal タイプです)
- コンストラクター関数で計算する。例えば、以下のコンストラクター関数は、ストリング "2005-01-01" から xs:date タイプの値を作成します。

  ```
  xs:date("2005-01-01")
  ```

- 組み込み関数 fn:true() および fn:false() によって戻す。これらの関数は、ブール値 (`true` および `false`) を戻します。これらの値は、リテラルで表すことはできません。
- 算術式や論理式など、さまざまな種類の式を使用して戻す。

ノード階層

シーケンスのノードは、1 つのルート・ノードおよびルート・ノードから直接的または間接的に到達可能なすべてのノードで構成される、 1 つ以上の*階層* (またはツリー) を形成します。

すべてのノードは必ず 1 つの階層に属し、すべての階層には必ず 1 つのルート・ノードがあります。DB2 は、文書、要素、属性、テキスト、処理命令、およびコメントの 6 種類のノードをサポートします。

以下の XML 文書 products.xml には、 products というルート要素が含まれ、この要素に product 要素が含まれます。各 product 要素は、pid (製品 ID) という属性と、 description という子要素を持ちます。description 要素には、 name および price という子要素が含まれます。

```
<products>
  <product pid="10">
    <description>
      <name>Fleece jacket</name>
      <price>19.99</price>
    </description>
  </product>
  <product pid="11">
    <description>
      <name>Nylon pants</name>
      <price>9.99</price>
    </description>
  </product>
</products>
```

13 ページの図 3 は、 products.xml の単純化されたデータ・モデルを示しています。この図には、文書ノード (D)、要素ノード (E)、属性ノード (A)、およびテキスト・ノード (T) が含まれています。

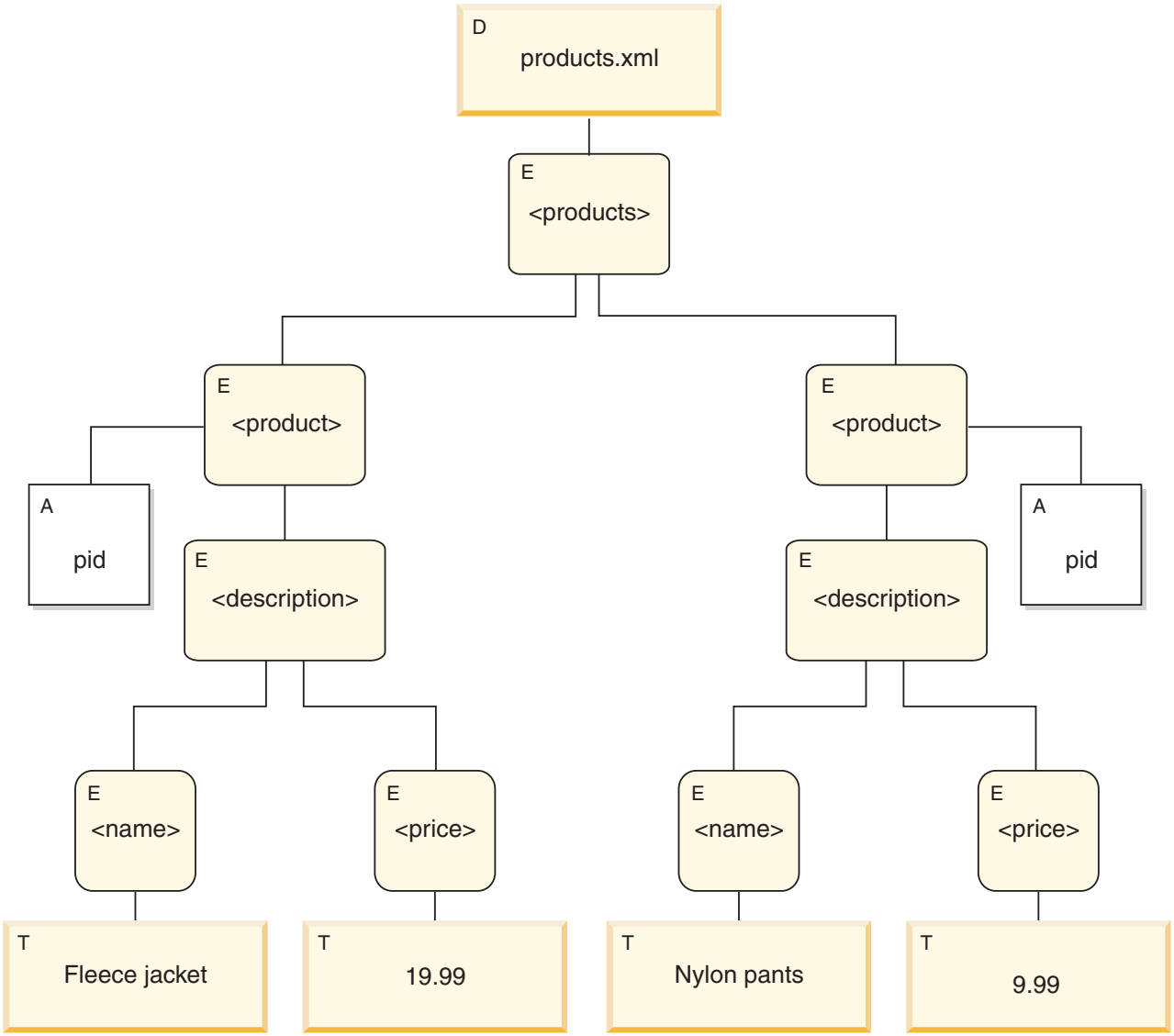


図 3. 文書 products.xml のデータ・モデル図

この例が示しているように、ノードは他のノードを子として持つことができ、このようにして 1 つ以上の*ノード階層* が形成されます。この例では、要素 `product` は `products` の子です。要素 `description` は `product` の子です。要素 `name` および `price` は、要素 `description` の子です。値 `Fleece Jacket` を持つテキスト・ノードは、要素 `name` の子であり、テキスト・ノード `19.99` は、要素 `price` の子です。

ノードのプロパティー

各ノードは、そのノードの特性を説明する*プロパティー* を持ちます。例えば、ノードのプロパティーには、ノードの名前、子、親、属性、およびそのノードを説明するその他の情報が含まれます。ノードの種類により、特定のノードがどのプロパティーを持つかが決まります。

ノードは、以下の 1 つ以上のプロパティーを持つことができます。

node-name
QName として表現されるノードの名前。

parent 現行ノードの親であるノード。

type-name
ノードの動的 (ランタイム) タイプ (タイプ・アノテーション とも言う)。

children
現行ノードの子であるノードのシーケンス。

attributes
現行ノードに属する属性ノードのセット。

string-value
ノードから抽出可能なストリング値。

typed-value
ノードから抽出可能な 0 個以上のアトミック値のシーケンス。

in-scope namespaces
ノードに関連付けられた範囲内の名前空間。

content
ノードの内容。

ノードの種類

DB2 は、文書、要素、属性、テキスト、処理命令、およびコメントの 6 種類のノードをサポートします。

文書ノード

文書ノードは、XML 文書をカプセル化します。

文書ノードは、0 個以上の子を持つことができます。子は、要素ノード、処理命令ノード、コメント・ノード、およびテキスト・ノードを含むことができます。

文書ノードのストリング値は、その派生したすべてのテキスト・ノードを文書順序で連結したコンテンツと同じです。ストリング値のタイプは xs:string です。文書ノードの型付き値は、型付き値が xdt:untypedAtomic タイプであることを除き、ストリング値と同じです。

文書ノードには、以下のノード・プロパティーがあります。

- children (空の場合もある)
- string-value
- typed-value

文書ノードは、計算コンストラクターを使用することで、XQuery 式で構成できます。文書ノードのシーケンスも、db2-fn:xmlcolumn 関数により戻すことができます。

要素ノード

要素ノードは、XML 要素をカプセル化します。

要素は、 0 または 1 個の親、および 0 個以上の子を持つことができます。子は、要素ノード、処理命令ノード、コメント・ノード、およびテキスト・ノードを含むことができます。文書ノードおよび属性ノードは、要素ノードの子になることはありません。ただし、要素ノードは、その属性の親であると認識されます。要素ノードの属性は、固有の QName を持つ必要があります。

要素ノードには、以下のノード・プロパティーがあります。

- node-name
- parent (空の場合もある)
- type-name
- children (空の場合もある)
- attributes (空の場合もある)
- string-value
- typed-value
- in-scope-namespaces

要素ノードは、直接コンストラクターまたは計算コンストラクターを使用することで、XQuery 式で構成できます。

要素ノードの type-name プロパティーは、要素ノードの型付き値とストリング値との関係を示します。例えば、要素ノードが type-name プロパティー xs:decimal とストリング値「47.5」を持つ場合、型付き値は 10 進数値 47.5 になります。要素ノードの type-name プロパティーが xdt:untyped の場合、要素の型付き値は、そのストリング値と等しく、xdt:untypedAtomic タイプになります。

属性ノード

属性ノードは、XML 属性を意味します。

属性ノードは、0 または 1 個の親を持つことができます。属性を所有する要素ノードは、属性ノードが親要素の子でない場合でも、その親であると考えられます。

属性ノードには、以下のノード・プロパティーがあります。

- node-name
- parent (空の場合もある)
- type-name
- string-value
- typed-value

属性ノードは、直接コンストラクターまたは計算コンストラクターを使用することで XQuery 式で構成できます。

属性ノードの type-name プロパティーは、属性ノードの型付き値とストリング値との関係を示します。例えば、属性ノードが type-name プロパティー xs:decimal とストリング値「47.5」を持つ場合、その型付き値は 10 進数値 47.5 になります。

テキスト・ノード

テキスト・ノードは、XML の文字内容をカプセル化します。

テキスト・ノードは、0 または 1 個の親を持つことができます。1 つの文書の子であるテキスト・ノード、または要素ノードが、隣接する兄弟として出現することはありません。文書ノードまたは要素ノードが構成されると、隣接するテキスト・ノードの兄弟が結合されて単一のテキスト・ノードになります。結果のテキスト・ノードが空の場合は廃棄されます。

テキスト・ノードには、以下のノード・プロパティーがあります。

- content (空の場合もある)
- parent (空の場合もある)

テキスト・ノードは、XQuery 式において、計算コンストラクターによって、または直接要素コンストラクターのアクションによって構成することができます。

処理命令ノード

処理命令ノードは、XML 処理命令をカプセル化します。

処理命令ノードは、0 または 1 個の親を持つことができます。処理命令の内容に、ストリング ?> を含めることはできません。処理命令のターゲットは、NCName にする必要があります。ターゲットは、命令の送信先であるアプリケーションを識別するために使用されます。

処理命令ノードには、以下のノード・プロパティーがあります。

- target
- content
- parent (空の場合もある)

処理命令ノードは、XQuery 式で、直接コンストラクターまたは計算コンストラクターを使用して構成することができます。

コメント・ノード

コメント・ノードは、XML コメントをカプセル化します。

コメント・ノードは、0 または 1 個の親を持つことができます。コメント・ノードのコンテンツには、ストリング「--」(2 つのハイフン) を含めることも、最後の文字としてハイフン文字 (-) を含めることもできません。

コメント・ノードは、以下のプロパティーを持ちます。

- content
- parent (空の場合もある)

コメント・ノードは、直接コンストラクターまたは計算コンストラクターを使用することで、XQuery 式内に構成できます。

ノードの文書順序

階層のすべてのノードは、*文書順序* と呼ばれる順序に従います。この順序では、各ノードはその子の前に表示されます。文書順序は、ノードの階層がシリアライズされた XML で示される場合にノードが表示される順序に対応します。

階層内のノードは、以下の順序で表示されます。

- ルート・ノードが 1 番目のノードです。
- 要素ノードは、その子の前に表示されます。
- 属性ノードは関連付けられている要素ノードの直後に表示されます。属性ノードの相対順序は任意ですが、この順序は照会処理時も変わりません。
- 兄弟の相対順序は、ノード階層内の順序で決まります。
- ノードの子と子孫は、ノードの後にある兄弟の前に表示されます。

ノード ID

各ノードにはユニークな ID があります。2 つのノードは、その名前や値が同じでも区別できます。一方、アトミック値には ID はありません。

ノード ID は、ID-type 属性とは異なります。XML 文書内の要素には、文書の作成者が ID-type 属性を指定することができます。一方ノード ID は、システムによってすべてのノードに自動的に割り当てられますが、ユーザーにとって直接的には不可視です。

ノード ID は、以下の種類の式を処理するために使用されます。

- ノード比較。**is** 演算子は、2 つのノードが同一 ID を持っているかどうか判別するためにノード ID を使用します。
- パス式。パス式は、重複ノードを除去するためにノード ID を使用します。
- シーケンス式。**union** 演算子、**intersect** 演算子、または **except** 演算子は、重複ノードを除去するためにノード ID を使用します。

ノードの型付き値およびストリング値

各ノードは、*型付き値* および*ストリング値* の両方を持っています。これらの 2 つのノード・プロパティーは、特定の XQuery 操作 (原子化など) および関数 (fn:data、fn:string、および fn:deep-equal など) の定義で使用されます。

表 1. ノードのストリング値および型付き値

ノードの種類	ストリング値	型付き値
文書	文書の順序におけるそのすべての子孫テキスト・ノードを連結した内容である xs:string データ・タイプのインスタンス。	文書の順序におけるそのすべての子孫テキスト・ノードを連結した内容である、xdt:untypedAtomic データ・タイプのインスタンス。
XML 文書内の要素	文書の順序におけるそのすべての子孫テキスト・ノードを連結した内容である、xs:string データ・タイプのインスタンス。	文書の順序におけるそのすべての子孫テキスト・ノードを連結した内容である、xdt:untypedAtomic データ・タイプのインスタンス。
XML 文書内の属性	元の XML 文書内の属性値を表す xs:string データ・タイプのインスタンス。	元の XML 文書内の属性値を表す xdt:untypedAtomic データ・タイプのインスタンス。
テキスト	xs:string データ・タイプのインスタンスとしての内容。	xdt:untypedAtomic データ・タイプのインスタンスとしての内容。
コメント	xs:string データ・タイプのインスタンスとしての内容。	xs:string データ・タイプのインスタンスとしての内容。
処理命令	xs:string データ・タイプのインスタンスとしての内容。	xs:string データ・タイプのインスタンスとしての内容。

XML をサポートするツール

IBM ツールとサード・パーティー・ツールの両方で、pureXML の処理がサポートされています。以下のツールが IBM によって提供されており、DB2 データベース・サーバーに同梱されているか、別個にダウンロードできます。

IBM Data Studio: XML のサポートには以下のものが含まれます。

- **ストアード・プロシージャー:** 入力または出力パラメーターとして XML データ・タイプを含むストアード・プロシージャーを作成および実行できます。
- **データ出力:** XML 列に含まれる文書をツリーまたはテキストとして表示できます。
- **SQL エディター:** リレーショナル・データと XML データの両方を処理する SQL ステートメントと XQuery 式を作成できます。
- **XML スキーマ:** XML スキーマ・リポジトリー (XSR) 内のスキーマ文書を管理できます。これにはスキーマの登録およびドロップ、またスキーマ文書の編集が含まれます。
- **XML 文書の妥当性検査:** XSR に登録されたスキーマに照らして XML 文書の妥当性検査を実行できます。
- **SQL ユーザー定義関数:** XML パラメーターを使用する SQL ユーザー定義関数を作成および実行できます。

コマンド行プロセッサー: いくつかの DB2 コマンドは、XML データのネイティブ・ストレージをサポートします。 DB2 コマンド行プロセッサー (CLP) から、XML データをリレーショナル・データと共に扱うことができます。 CLP から実行できるタスクの例は、次のとおりです。

- 接頭部として XQUERY キーワードを付けて、XQuery ステートメントを発行する。
- XML データをインポートおよびエクスポートする。
- XML 列の統計を収集する。
- XML データ・タイプの IN、OUT、または INOUT パラメーターを指定してストアード・プロシージャーを呼び出す。
- XML 文書の処理に必要な XML スキーマ、DTD、および外部エンティティーを処理する。
- XML データの索引、および XML 列を含む表を再編成する。
- XML 文書を分解する。

IBM Database Add-Ins for Microsoft Visual Studio: IBM Database Add-Ins for Microsoft Visual Studio を使用して、XML 列および XML データに対する索引を持つ表を作成できます。このツールでは、XML 列を他の列と同様の方法で作成できます。単にデータ・タイプを XML に指定するだけです。このツールでは、XML Index Designer を使用して索引を作成できます。CREATE INDEX 構文を使用する場合とは異なり、XML データに対する索引のために XML パターン式を手動で指定する必要はありません。代わりに、登録済み XML スキーマ、XML 列からの文書、またはローカル・ファイル内の XML スキーマのツリー表現から、索引作成する XML ノードをグラフィカルに選択できます。後はツールが XML パターン式を

自動的に生成します。またはその代わりに、XML パターン式を手動で指定することもできます。他のすべての索引属性を指定すれば、後はツールが索引を自動的に生成します。

EXPLAIN: XQuery ステートメントおよび SQL/XML ステートメントに対する EXPLAIN ステートメントを発行して、これらのステートメントのアクセス・プラン、および DB2 データベース・サーバーが索引を使用するかどうかを素早く調べることができます。XQuery ステートメントに対する EXPLAIN ステートメントを発行するには、以下の例のように XQuery キーワードを使用し、その後に単一または二重引用符で囲んだ XQuery ステートメントを指定します。

```
EXPLAIN PLAN SELECTION FOR XQUERY 'for $c in
db2-fn:xmlcolumn("XISCANTABLE.XMLCOL" )/a[@x="1"]/b[@y="2"] return $c'
```

DB2 は、アクセス・プラン情報を収集して EXPLAIN 表に入れます。XML 列のシーケンス・サイズの予期値は、EXPLAIN_STREAM 表の SEQUENCE_SIZES 列に保管されます。また、EXPLAIN_PREDICATE 表に、ユーザーが指定していない述部に関するデータが書かれることがあります。これらの述部は索引スキャンで使用される XPath 式を評価するために、DB2 データベース・サーバーによって EXPLAIN 操作の際に生成されます。この述部情報を評価する必要はありません。これらの述部はオプティマイザー・プランの一部ではないので、PREDICATE_ID 列および FILTER_FACTOR 列では値が -1 になります。

またはその代わりに、Visual Explain ツールを使用してこれらのアクセス・プランをグラフィカルに表示することにより、Explain 表を手動で解釈しないで済ませることもできます。以下のノードは、XML 操作を示すためにグラフ内に表示されます。

IXAND
DB2 データベース・サーバーが、複数の索引スキャン結果に対して AND 述部を適用したことを示します。

XISCAN
DB2 データベース・サーバーが、XML データに対する索引を使用してデータにアクセスしたことを示します。

XSCAN
DB2 データベース・サーバーが XPath 式を評価し、XML 文書から XML 文書フラグメントを取り出したことを示します。

XANDOR
DB2 データベース・サーバーが、複数の索引スキャン結果に対して AND および OR 述部を適用したことを示します。

XTQ
DB2 データベース・サーバーが、特別な表キュー (TQ) を使用して、グローバル XML シーケンスの各項目をその元のデータベース・パーティションにルーティングし、項目の評価に基づいて XML 文書から XML データを取り出し、XML データを出力シーケンスに集約したことを示します。

pureXML のフェデレーション・サポート

フェデレーテッド環境では、XML 列に保管された XML 文書を含むリモート・データ・ソースを使用して作業できます。リモート XML データの照会と操作の両方が可能で、それには XML 文書をリモート表に分解することが含まれます。

リモート XML データで作業を始める前に、作業対象の文書が保管される XML 列を含むリモート表に対してニックネームを作成する必要があります。

XML データ・ソースを含むフェデレーテッド・システムのセットアップ方法について詳しくは、 Federation Server の資料で『リモート XML データの操作』を参照してください。

pureXML のレプリケーションおよびイベント・パブリッシング・サポート

WebSphere® Replication Server および XML データ・タイプの WebSphere® Data Event Publisher サポートによって、 XML 列に保管された XML 文書を複製および公開することができます。

Q レプリケーションを使用してデータベース間で XML 文書を複製すること、またはイベント・パブリッシングを使用して文書をアプリケーションに公開することができます。

XML 列に保管された XML 文書を含むデータベース用に Q レプリケーションおよびイベント・パブリッシングをセットアップする方法について詳しくは、WebSphere Replication Server および WebSphere Data Event Publisher の資料で、「XML データ・タイプ」および親トピックを参照してください。

XML サポートの記事

XML サポートの活用に関する付加的な記事が、developerWorks® Information Management から入手できます。これらの記事は幅広いトピックを取り上げており、それには移行およびデータ移動、一般概説、段階的チュートリアル、および XML データ処理のベスト・プラクティスなどが含まれています。

これらの記事は、www.ibm.com/developerworks/data/zones/xml/ からアクセスできます。

注: developerWorks は DB2 インフォメーション・センターの一部ではありません。このリンクは、DB2 インフォメーション・センターの外部へのリンクです。

第 2 章 pureXML チュートリアル

pureXML XML データ・タイプを使用して、単一の整形式 XML 文書を各行に格納できるように表の列を定義できます。このチュートリアルでは、DB2 データベースをセットアップし、XML データを格納して、基本的な pureXML の操作を実行する方法について、実例を使用して説明します。

このチュートリアルを完了すると、次の操作を実行できるようになります。

- XML データを格納する DB2 データベースおよび表を作成する
- XML データに索引を作成する
- XML タイプの列に XML 文書を挿入する
- XML 列に保存された XML 文書を更新する
- XML 文書の内容に基づいて行を削除する
- XML データを照会する
- XML スキーマに対して XML 文書を妥当性検査する
- XSLT スタイルシートを使用して XML 文書を変換する

C++、Java、および PHP などのアプリケーション・プログラミング言語は、XML データ・タイプをサポートします。XML データを DB2 データベース表に保管したり、表からデータを検索したり、XML パラメーターのあるストアード・プロシージャーまたはユーザー定義関数を呼び出したりするアプリケーションを作成することができます。

このチュートリアルは単一パーティション・データベース環境用に作成されていますが、パーティション・データベース環境でも pureXML を使用できます。

前提条件

DB2 コマンド・ウィンドウで、`db2 -td~` コマンド (**-td~** オプションを指定した **db2** コマンド) を発行して DB2 コマンド行プロセッサーを開始します。[1]

-td オプションは、ティルド (~) をステートメント終了文字として設定します。名前空間宣言の終了文字もセミコロンであるため、デフォルトのセミコロン (**-t** オプション) 以外の終了文字を指定することによって、名前空間宣言を使用するステートメントまたは照会が誤って解釈されないようにします。このチュートリアル内の例では、~ 終了文字を使用します。

対話モードで DB2 コマンド行プロセッサーにこの演習内の例を入力するか、コピー・アンド・ペーストできます。

名前空間: このチュートリアルで使用される XML 文書には、名前空間が含まれています。名前空間を含む XML 文書を使用する場合、CREATE INDEX ステートメントを使用した XML データの索引の作成、または XQuery 式を使用した XML デ

1. Windows オペレーティング・システムでは、**db2 -td~** コマンドを発行する前に **db2cmd** コマンドを使用して DB2 コマンド・ウィンドウを開始します。

ータの照会など、名前空間を指定するすべての照会および関連操作では、期待通りの結果を得るために同じ名前空間を宣言する必要があります。名前空間を含む XML 文書を使用する際には名前空間を宣言するのが、標準の名前空間の動作です。

演習 1: XML データを格納する DB2 データベースおよび表を作成する

この演習では、データベースを XML 列を含む表と共に作成する方法を示します。

チュートリアルでは、タイプ XML の列を持つ表が含まれる表に XML データを保管します。以下のステップを実行して、チュートリアルで使用するデータベースおよび表を作成します。

1. 以下のコマンドを実行して、XMLTUT というデータベースを作成します。

   ```
   CREATE DATABASE xmltut~
   ```

   デフォルトではデータベースは UTF-8 (Unicode) コード・セットを使用します。XML データを UTF-8 以外のコード・セットでデータベースに保管することを選択した場合、 BIT DATA、BLOB、または XML など、このデータをコード・ページ変換が実施されない形式で挿入することが最善です。 XML 構文解析中に文字データ・タイプが使用されなくなり、文字置換が発生しないようにするには、**ENABLE_XMLCHAR** 構成パラメーターを **NO** に設定します。
2. データベースに接続します。

   ```
   CONNECT TO xmltut~
   ```
3. INFO という XML 列を含む Customer という名前の表を作成します。

   ```
   CREATE TABLE Customer (Cid BIGINT NOT NULL PRIMARY KEY, Info XML)~
   ```

   XML データを保管または索引付けする場合、主キーは必要ありません。

   また、ALTER TABLE SQL ステートメントを使用して、表に 1 つ以上の XML 列を追加することもできます。

チュートリアルに戻る

演習 2: XML データに索引を作成する

この演習では、XML データに対する索引の作成方法を示します。XML データの索引を使用すると、XML 列の照会のパフォーマンスを改良できます。述部および文書をまたいだ結合で頻繁に使用される XML 要素または属性に索引を付けます。

リレーショナル索引および XML データに対する索引は両方とも、列を索引付けします。ただし、リレーショナル索引が列全体に索引を付けるのに対して、XML データに対する索引は列の一部に索引を付けます。 XML 列のどの部分に索引を付けるかを、XML パターン (限定された XPath 式) を指定することによって指示してください。また、索引付きの値が格納されるデータ・タイプを指定する必要もあります。一般に、選択するデータ・タイプは、照会で使用するデータ・タイプと同じである必要があります。

リレーショナル索引と同様に、述部および文書横断的な結合で頻繁に使用される XML 要素または属性に索引を付けることをお勧めします。

単一の XML 列のみ索引を付けられます。複合索引はサポートされていません。ただし、XML 列上では複数の索引を使用できます。

CREATE INDEX ステートメントのすべての節が XML データの索引に適用されるわけではありません。詳しくは、CREATE INDEX ステートメントを参照してください。

XML データに対する索引を作成するには、以下のステートメントを実行します。

```
CREATE INDEX cust_cid_xmlidx ON Customer(Info)
GENERATE KEY USING XMLPATTERN
  'declare default element namespace "http://posample.org"; /customerinfo/@Cid'
AS SQL DOUBLE~
```

このステートメントは、CUSTOMER 表の INFO 列から、<customerinfo> 要素の Cid 属性の値に索引を付けます。デフォルトで、XML データが索引付けされており、XML データが指定のデータ・タイプ SQL DOUBLE へのキャストに失敗した場合は、索引の項目は作成されません。このとき、エラーは戻されません。

指定する XML パターンは、大/小文字を区別します。例えば、XML 文書が属性「Cid」ではなく「cid」を含む場合、それらの文書はこの索引と一致しません。

チュートリアルに戻る

演習 3: XML タイプ列に XML 文書を挿入する

整形式 XML 文書は、INSERT SQL ステートメントを使用して、XML タイプ列に挿入されます。この演習では、INSERT SQL ステートメントを使用して整形式 XML 文書を XML 列に挿入する方法を示します。

この演習では、コマンド行プロセッサーを使用して、XML 文書を XML タイプ列に手動で挿入する方法を示します。ただし、通常は、XML 文書はアプリケーション・プログラムを使用して挿入されます。

XML データは XML、バイナリー、または文字タイプを使用して挿入できますが、コード・ページ変換の問題を回避するため、XML またはバイナリー・タイプを使用してください。この演習では、XML 文書は文字リテラルです。ほとんどの場合、ストリング・データを XML データ・タイプのターゲットに直接割り当てることはできません。まず XMLPARSE 関数を使用してそのデータを明示的に構文解析する必要があります。ただし、INSERT、UPDATE、または DELETE 操作では、XMLPARSE 関数を明示的に呼び出さなくても、ストリング・データを XML 列に直接割り当てることができます。以下の 3 つの例では、ストリング・データは暗黙的に構文解析されます。詳しくは、XML 構文解析の資料を参照してください。

演習 1 で作成した Customer 表に 3 つの XML 文書を挿入するには、以下のステートメントを実行します。

```
INSERT INTO Customer (Cid, Info) VALUES (1000,
'<customerinfo xmlns="http://posample.org" Cid="1000">
  <name>Kathy Smith</name>
  <addr country="Canada">
    <street>5 Rosewood</street>
    <city>Toronto</city>
    <prov-state>Ontario</prov-state>
    <pcode-zip>M6W 1E6</pcode-zip>
```

```
  </addr>
  <phone type="work">416-555-1358</phone>
</customerinfo>')~

INSERT INTO Customer (Cid, Info) VALUES (1002,
'<customerinfo xmlns="http://posample.org" Cid="1002">
  <name>Jim Noodle</name>
  <addr country="Canada">
    <street>25 EastCreek</street>
    <city>Markham</city>
    <prov-state>Ontario</prov-state>
    <pcode-zip>N9C 3T6</pcode-zip>
  </addr>
  <phone type="work">905-555-7258</phone>
</customerinfo>')~

INSERT INTO Customer (Cid, Info) VALUES (1003,
'<customerinfo xmlns="http://posample.org" Cid="1003">
  <name>Robert Shoemaker</name>
  <addr country="Canada">
    <street>1596 Baseline</street>
    <city>Aurora</city>
    <prov-state>Ontario</prov-state>
    <pcode-zip>N8X 7F8</pcode-zip>
  </addr>
  <phone type="work">905-555-2937</phone>
</customerinfo>')~
```

レコードが正常に挿入されたことを確認するには、以下のステートメントを実行します。

```
SELECT * from Customer~
```

チュートリアルに戻る

演習 4: XML 列に保管されている XML 文書を更新する

この演習では、XQuery の更新式を含む場合と含まない場合の、UPDATE SQL ステートメントを使用して XML 文書を更新する方法を示します。

XQuery の更新式を使用しない更新

XQuery の更新式が含まれていない UPDATE ステートメントを使用する場合、全文書の更新を実行する必要があります。

演習 3 で挿入した文書の 1 つで <street>、<city>、および <pcode-zip> 要素の値を更新するには、以下のステートメントを実行します。

```
UPDATE customer SET info =
'<customerinfo xmlns="http://posample.org" Cid="1002">
  <name>Jim Noodle</name>
  <addr country="Canada">
    <street>1150 Maple Drive</street>
    <city>Newtown</city>
    <prov-state>Ontario</prov-state>
    <pcode-zip>Z9Z 2P2</pcode-zip>
  </addr>
  <phone type="work">905-555-7258</phone>
</customerinfo>'
WHERE XMLEXISTS (
 'declare default element namespace "http://posample.org";
  $doc/customerinfo[@Cid = 1002]'
  passing INFO as "doc")~
```

XMLEXISTS 述部は、属性の Cid="1002" を含む文書のみを確実に置き換えます。XMLEXISTS での述部式 [@Cid = 1002] がストリング比較 ([@Cid = "1002"]) として指定されてはいないことに注意してください。これは、演習 2 で Cid 属性の索引を作成したときに DOUBLE データ・タイプを使用したためです。索引をこの照会と一致させるためには、述部式で Cid をストリングとして指定することはできません。

XML 文書が更新されたことを確認するには、以下のステートメントを実行します。

```
SELECT * from Customer~
```

Cid="1002" を含むレコードに、変更された <street>、<city>、および <pcode-zip> の値が含まれます。

表内の XML 文書を同じ表の非 XML 列の値を使用して識別できる場合、SQL 比較述部を使用して更新する行を識別することができます。前の例では、XML 文書の Cid 値が CUSTOMER 表の CID 列にも保管され、CID 列の SQL 比較述部を行の識別に使用することもできました。また、WHERE 文節を以下の文節に置換することもできます。

```
WHERE Cid=1002
```

XQuery の更新式を使用した更新

XQuery の更新式が含まれている UPDATE ステートメントを使用する場合、既存の XML 文書の一部を更新することができます。

既存の XML 文書内の顧客アドレスを更新するには、XQuery 変換式を使用する以下の SQL ステートメントを実行します。

```
UPDATE Customer set Info =
  XMLQUERY( 'declare default element namespace "http://posample.org";
  transform
  copy $mycust := $cust
  modify
    do replace $mycust/customerinfo/addr with
      <addr country="Canada">
        <street>25 EastCreek</street>
        <city>Markham</city>
        <prov-state>Ontario</prov-state>
        <pcode-zip>N9C 3T6</pcode-zip>
     </addr>
  return $mycust'
  passing INFO as "cust")
WHERE CID = 1002~
```

顧客の住所を更新するために、XMLQUERY 関数は置換式を使用する XQuery 変換式を実行してから、以下のように更新された情報を UPDATE ステートメントに戻します。

- XMLQUERY passing 節は ID cust を使用して、顧客情報を XML 列 INFO から XQuery 式に渡します。
- transform 式の **copy** 節で顧客情報の論理スナップショットが取得され、$mycust 変数に代入されます。
- transform 式の **modify** 節の中の replace 式は、顧客情報のコピー内にある住所情報を置換します。
- XMLQUERY は、$mycust 変数内の更新された顧客文書を戻します。

XML 文書に更新された顧客住所が含まれることを確認するには、次のステートメントを実行します。

```
SELECT Info FROM Customer WHERE Cid = 1002~
```

チュートリアルに戻る

演習 5: XML データを削除する

この演習では、SQL ステートメントを使用して、XML 文書全体あるいは XML 文書のセクションのみを削除する方法を示します。

XML 文書全体の削除

XML 文書全体を削除するには、DELETE SQL ステートメントを使用します。削除する特定の文書を識別するには、XMLEXISTS 述部を使用します。

属性 Cid=1003 を含む <customerinfo> 要素を持つ XML 文書のみを Info 列から削除するには、次のステートメントを実行します。

```
DELETE FROM Customer
WHERE XMLEXISTS (
 'declare default element namespace "http://posample.org";
  $doc/customerinfo[@Cid = 1003]'
  passing INFO as "doc")~
```

表内の XML 文書を同じ表の非 XML 列の値を使用して識別できる場合、SQL 比較述部を使用して削除する行を識別することができます。前の例では、XML 文書の Cid 値が CUSTOMER 表の CID 列にも保管され、SQL 比較述部を CID 列に適用して行を識別する以下の DELETE ステートメントを使用することによって、同じ操作を実行することもできました。

```
DELETE FROM Customer WHERE Cid=1003~
```

XML 文書が削除されたことを確認するには、以下の SQL ステートメントを実行します。

```
SELECT * FROM Customer~
```

2 つのレコードが戻されます。

XML 文書のセクションの削除

文書全体を削除する代わりに XML 文書の一部だけを削除するには、XQuery 更新削除式を含む UPDATE SQL ステートメントを使用します。

Cid の値が 1002 である顧客レコードからすべての電話情報を削除するには、XMLQUERY 関数を使用する以下の SQL ステートメントを実行します。

```
UPDATE Customer
SET info = XMLQUERY(
   'declare default element namespace "http://posample.org";
    transform
    copy $newinfo := $info
    modify do delete ($newinfo/customerinfo/phone)
    return $newinfo' passing info as "info")
WHERE cid = 1002~
```

<phone> 要素を削除するために、XMLQUERY 関数は削除式を使用する XQuery 変換式を実行してから、以下のように更新された情報を UPDATE ステートメントに戻します。

- XMLQUERY passing 節は ID `info` を使用して、顧客情報を XML 列 INFO から XQuery 式に渡します。
- transform 式の **copy** 節で顧客情報の論理スナップショットが取得され、`$newinfo` 変数に代入されます。
- transform 式の **modify** 節の中の delete 式は、顧客情報のコピー内にある <phone> 要素を削除します。
- XMLQUERY は、`$newinfo` 変数内の更新された顧客文書を戻します。

顧客レコードに <phone> 要素が表示されなくなったことを確認するには、以下のステートメントを実行します。

```
SELECT * FROM Customer WHERE Cid=1002~
```

チュートリアルに戻る

演習 6: XML データを照会する

この演習では、XML データを、SQL、XQuery の XQuery 式、またはその組み合わせを使用して照会する方法を示します。

SQL のみを使用する場合は、列レベルでのみ照会を行うことができます。つまり、列に格納された XML 文書全体を戻すことができますが、文書内で照会したり、文書の一部を戻したりすることはできません。XML 文書内で値を照会したり、文書の一部を戻すには、XQuery を使用する必要があります。

この演習の照会では、SQL コンテキストで XQuery を使用し、XQuery コンテキストで SQL を使用します。

重要: XQuery は大/小文字を区別しますが、SQL は大/小文字を区別しません。このため、XQuery を使用する場合は、表や SQL スキーマの名前など (どちらもデフォルトで大文字) の名前は慎重に指定してください。SQL コンテキストの場合も、XQuery 式は依然として大/小文字を区別します。

SQL コンテキストでの照会

XML 文書全体の検索

INFO という名前の列に格納された XML 文書および CID 主キー列からの値すべてを検索するには、次の SELECT ステートメントを実行します。

```
SELECT cid, info FROM customer~
```

この照会では、2 つの格納された XML 文書を戻します。

XML 値の検索およびフィルター処理

INFO 列の XML 文書内で照会するには、XMLQUERY 関数を使用して XQuery 式を起動する、以下の SELECT ステートメントを実行します。

```
SELECT XMLQUERY (
  'declare default element namespace "http://posample.org";
   for $d in $doc/customerinfo
   return <out>{$d/name}</out>'
```

```
   passing INFO as "doc")
FROM Customer as c
WHERE XMLEXISTS ('declare default element namespace "http://posample.org";
   $i/customerinfo/addr[city="Toronto"]' passing c.INFO as "i")~
```

XMLQUERY 関数では、デフォルトの名前空間が最初に指定されます。この名前空間は、直前に挿入された文書の名前空間と一致します。**for** 節は、Info 列の各文書の <customerinfo> 要素における反復を指定します。 INFO 列が **passing** 節を使用して指定されています。この節は、INFO 列を変数 doc にバインドし、これは **for** 節で参照されます。その後、**return** 節は <out> 要素を構成します。この要素には、**for** 節の反復ごとに取得された <name> 要素が含まれています。

WHERE 節は XMLEXISTS 述部を使用して、Info 列内の文書のサブセットだけを検討します。このフィルター処理は、<city> 要素 (指定されたパスにある) の値が Toronto である文書のみを生成します。

SELECT ステートメントは、次の構成要素を戻します。

```
<out xmlns="http://posample.org"><name>Kathy Smith</name></out>
```

パラメーターでの db2-fn:sqlquery の使用

db2-fn:sqlquery 関数で SQL 全選択に値を渡すには、以下の照会を実行します。

```
VALUES XMLQUERY (
  'declare default element namespace "http://posample.org";
  for $d in db2-fn:sqlquery(
    ''SELECT INFO FROM CUSTOMER WHERE Cid = parameter(1)'',
    $testval)/customerinfo
  return <out>{$d/name}</out>'
  passing 1000 as "testval" )~
```

XMLQUERY 関数は、ID testval を使用して値 1000 を XQuery 式に渡します。その後、XQuery 式は PARAMETER スカラー関数を使用することにより、db2-fn:sqlquery 関数に値を渡します。

XQuery 式は次の構成要素を戻します。

```
<out xmlns="http://posample.org">
   <name>Kathy Smith</name>
</out>
```

XQuery コンテキストでの照会

DB2 XQuery には、DB2 データベース専用の 2 つの組み込み関数があります。db2-fn:sqlquery および db2-fn:xmlcolumn です。db2-fn:sqlquery は、SQL 全選択の結果表であるシーケンスを取り出します。db2-fn:xmlcolumn は、XML 列からシーケンスを取り出します。

照会によって XQuery 式が直接呼び出された場合、その接頭部として大/小文字を区別しないキーワード XQUERY を付ける必要があります。

注: XQuery 式の結果の表示には特に、コマンド行プロセッサー環境をカスタマイズ設定できるオプションが複数あります。例えば、XQuery 式からの結果を簡単に読み取れるようにするには、以下のように **-i** オプションを設定します。

```
UPDATE COMMAND OPTIONS USING i ON~
```

XML 文書全体の検索

INFO 列に事前に挿入したすべての XML 文書を検索するには、XQuery を db2-fn:xmlcolumn または db2-fn:sqlquery のいずれかとともに使用できます。

db2-fn:xmlcolumn の使用

INFO 列のすべての XML 文書を検索するには、次の照会を実行します。

```
XQUERY db2-fn:xmlcolumn ('CUSTOMER.INFO')~
```

SQL ステートメントの名前は、デフォルトで自動的に大文字に変換されます。したがって、CREATE TABLE SQL ステートメントを使用して CUSTOMER 表を作成した場合、表および列の名前は大文字になります。XQuery は大/小文字を区別するため、db2-fn:xmlcolumn を使用して表および列の名前を指定する場合、正しい大/小文字を使用するよう、注意する必要があります。

この照会は、SQL 照会の `SELECT Info FROM Customer` と同等になります。

db2-fn:sqlquery の使用

INFO 列のすべての XML 文書を検索するには、次の照会を実行します。

```
XQUERY db2-fn:sqlquery ('SELECT Info FROM Customer')~
```

INFO および CUSTOMER 名は大文字で指定する必要はありません。これは、SELECT ステートメントが SQL コンテキストで処理され、そのため大/小文字を区別しないからです。

部分的な XML 文書の検索

XML 文書全体を検索するのではなく、XQuery を db2-fn:xmlcolumn または db2-fn:sqlquery のいずれかとともに使用することによって、文書の一部を検索し、文書に存在する値をフィルター処理できます。

db2-fn:xmlcolumn の使用

Toronto という値の <city> 要素 (指定されたパスに沿って) を持つ Info 列のすべての文書について、<name> ノードを含む要素を戻すには、以下の照会を実行します。

```
XQUERY declare default element namespace "http://posample.org";
  for $d in db2-fn:xmlcolumn('CUSTOMER.INFO')/customerinfo
  where $d/addr/city="Toronto"
  return <out>{$d/name}</out>~
```

db2-fn:xmlcolumn 関数は、CUSTOMER 表の INFO 列からシーケンスを検索します。**for** 節は、 CUSTOMER.INFO 列の各 <customerinfo> 要素に変数 `$d` をバインドします。 **where** 節は、項目を <city> 要素 (指定されたパスにある) の値が Toronto である項目にのみ制限します。 **return** 節は、戻された XML 値を構成します。この値は、要素 <out> であり、以下のように **where** 節で指定された条件を満たすすべての文書の <name> 要素が含まれます。

```
<out xmlns="http://posample.org">
<name>
              Kathy Smith
</name>
</out>
```

db2-fn:sqlquery の使用

XQuery 式内で全選択を実行するには、以下の照会を実行します。

```
XQUERY declare default element namespace "http://posample.org";
  for $d in db2-fn:sqlquery(
    'SELECT INFO
     FROM CUSTOMER
     WHERE Cid < 2000')/customerinfo
  where $d/addr/city="Toronto"
  return <out>{$d/name}</out>~
```

この例では、全選択で照会される XML 文書のセットが、非 XML の CID 列にある特定の値によってまず制限されます。この例は、db2-fn:sqlquery の利点を示しています。つまり、これにより、SQL 述部を XQuery 式内に適用できるということです。その後、SQL 照会の結果として生じる文書はさらに、XQuery 式の **where** 節で <city> 要素 (指定されたパスにある) の値が Toronto であるものに制限されます。

この照会は、db2-fn:xmlcolumn を使用した直前の例と同じ結果を導きます。

```
<out xmlns="http://posample.org">
<name>
              Kathy Smith
</name>
</out>
```

パラメーターでの db2-fn:sqlquery の使用

db2-fn:sqlquery 関数で SQL 全選択に値を渡すには、以下の照会を実行します。

```
XQUERY declare default element namespace "http://posample.org";
  let $testval := 1000
  for $d in db2-fn:sqlquery(
    'SELECT INFO FROM CUSTOMER WHERE Cid = parameter(1)',
    $testval)/customerinfo
  return <out>{$d/name}</out>~
```

XQuery 式では、**let** 節は $testval の値を 1000 に設定します。**for** 節では、式は PARAMETER スカラー関数を使用することにより、db2-fn:sqlquery 関数に値を渡します。

XQuery 式は次の構成要素を戻します。

```
<out xmlns="http://posample.org">
   <name>Kathy Smith</name>
</out>
```

チュートリアルに戻る

演習 7: XML スキーマに対して XML 文書を妥当性検査する

この演習では、XML 文書の妥当性検査の方法を示します。XML スキーマに対してのみ XML 文書を妥当性検査できます。DTD に対する妥当性検査はサポートされていません。ただし、DTD に照らした妥当性検査は行えませんが、DOCTYPE を含む文書または DTD を参照する文書を挿入することはできます。

IBM Rational® Application Developer で提供されているツールなどのように、DTD、表、および XML 文書を含むさまざまなソースから XML スキーマを生成するのに役立つツールを使用できます。

妥当性検査を行う前に、組み込み XML スキーマ・リポジトリー (XSR) で XML スキーマを登録する必要があります。このプロセスには、XML スキーマを構成する各 XML スキーマ文書を登録してから、登録を完了することが含まれます。XML スキーマを登録する 1 つの方法に、コマンドの利用があります。

スキーマ文書を登録し、posample.customer XML スキーマの登録を完了するには、以下のコマンドを実行します。 (この XML スキーマは 1 つのスキーマ文書によってのみ構成されているので、単一のコマンドを使用して文書の登録と登録の完了の両方を行うことができます。) このコマンドは、sqllib/samples/xml ディレクトリーへの絶対パスを指定します。システム上のパスが c:/sqllib/ で始まらない場合、適切にコマンドのファイル・パスを変更してください。

```
REGISTER XMLSCHEMA 'http://posample.org'
FROM 'file:///c:/sqllib/samples/xml/customer.xsd' AS posample.customer COMPLETE~
```

XSR に保管されているオブジェクトについての情報を含む、SYSCAT.XSROBJECTS カタログ・ビューを照会することで、XML スキーマが正常に登録されたことを検査できます。この照会およびその結果 (分かりやすくするために形式を整えています) は、以下のとおりです。

```
SELECT OBJECTSCHEMA, OBJECTNAME FROM SYSCAT.XSROBJECTS~


OBJECTSCHEMA           OBJECTNAME
--------------------   --------------------
POSAMPLE               CUSTOMER
```

これで、妥当性検査用に XML スキーマを使用することができます。通常、妥当性検査は、XMLVALIDATE 関数を使用して INSERT または UPDATE 操作時に実行できます。XMLVALIDATE を指定した INSERT または UPDATE 操作は、妥当性検査が成功した場合にのみ実行されます。

posample.customer XML スキーマに従って文書が妥当である場合、CUSTOMER 表の INFO 列に XML 文書を挿入するには、以下のステートメントを実行します。

```
INSERT INTO Customer(Cid, Info) VALUES (1003, XMLVALIDATE (XMLPARSE (DOCUMENT
'<customerinfo xmlns="http://posample.org" Cid="1003">
  <name>Robert Shoemaker</name>
  <addr country="Canada">
    <street>1596 Baseline</street>
    <city>Aurora</city>
    <prov-state>Ontario</prov-state>
    <pcode-zip>N8X 7F8</pcode-zip>
  </addr>
  <phone type="work">905-555-7258</phone>
  <phone type="home">416-555-2937</phone>
```

```
  <phone type="cell">905-555-8743</phone>
  <phone type="cottage">613-555-3278</phone>
</customerinfo>' PRESERVE WHITESPACE )
ACCORDING TO XMLSCHEMA ID posample.customer ))~
```

この例の XML 文書は、文字データとして渡されます。ただし、XMLVALIDATE は XML データでのみ動作します。XML 文書は文字データとして渡されるため、XMLPARSE 関数を使用してデータを明示的に構文解析する必要があります。XMLPARSE 関数は引数を XML 文書として解析し、XML 値を戻します。

DB2 データベース・サーバーは、一部の操作で暗黙的な構文解析を実行します。例えば、INSERT、UPDATE、DELETE、または MERGE ステートメントで、ストリング・データ・タイプ (文字、グラフィック、またはバイナリー) のホスト変数、パラメーター・マーカー、または SQL 式を XML 列に割り当てるときに、暗黙的な構文解析が行われます。

妥当性検査および挿入が成功したことを確認するには、INFO 列を照会します。

```
SELECT Info FROM Customer~
```

この照会では 3 つの XML 文書を戻し、そのうちの 1 つは挿入したばかりの文書です。

チュートリアルに戻る

演習 8: XSLT スタイルシートを使用した変換

この演習では、Extensible Stylesheet Language Transformation (XSLT) スタイルシートおよび XSLTRANSFORM 組み込み関数を使用して、データベース内の XML データを他の形式に変換する方法を示します。

任意の数の大学生レコードが含まれている XML 文書について考えてみましょう。各 student 要素には、生徒の ID、名前、姓、年齢、および在籍する大学が含まれています。以下の文書には 2 人の生徒が含まれています。

```
<?xml version="1.0"?>
<students xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
     <student studentID="1" givenName="Steffen" familyName="Siegmund"
        age="21" university="Rostock"/>
     <student studentID="2" givenName="Helena" familyName="Schmidt"
         age="23" university="Rostock"/>
</students>
```

さらに、XML レコード内の情報を抽出して、ブラウザーで表示できる HTML Web ページを作成するとします。情報を変換するには、以下の XSLT スタイルシートが必要です。

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
     xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:param name="headline"/>
<xsl:param name="showUniversity"/>
<xsl:template match="students">
     <html>
     <head/>
     <body>
     <h1><xsl:value-of select="$headline"/></h1>
     <table border="1">
```

```
     <th>
     <tr>
     <td width="80">StudentID</td>
     <td width="200">Given Name</td>
     <td width="200">Family Name</td>
     <td width="50">Age</td>
     <xsl:choose>
          <xsl:when test="$showUniversity =''true''">
               <td width="200">University</td>
          </xsl:when>
     </xsl:choose>
     </tr>
     </th>
     <xsl:apply-templates/>
     </table>
     </body>
     </html>
     </xsl:template>
          <xsl:template match="student">
               <tr>
               <td><xsl:value-of select="@studentID"/></td>
               <td><xsl:value-of select="@givenName"/></td>
               <td><xsl:value-of select="@familyName"/></td>
               <td><xsl:value-of select="@age"/></td>
               <xsl:choose>
                    <xsl:when test="$showUniversity = ''true'' ">
                         <td><xsl:value-of select="@university"/></td>
                    </xsl:when>
               </xsl:choose>
               </tr>
     </xsl:template>
</xsl:stylesheet>
```

データを変換するには、以下のようにします。

1. 以下のコマンドを実行して、XML 文書とスタイルシート文書を保管するための 2 つの表を作成します。

   ```
   CREATE TABLE XML_DATA (DOCID INTEGER, XML_DOC XML )~
   CREATE TABLE XML_TRANS (XSLID INTEGER, XSLT_DOC CLOB(1M))~
   ```

2. 以下の INSERT ステートメントを使用して、XML 文書および XSLT スタイルシート全体を表に挿入します。

   簡略のため、このステップの 2 番目の INSERT ステートメントでは、XSLT スタイルシートの切り捨てられたバージョンを示しています。ステートメントを使用する前に、切り捨てられたスタイルシートを、前にリストした XSLT スタイルシートに置き換えます。

   ```
   INSERT INTO XML_DATA VALUES
        (1,
         '<?xml version="1.0"?>
          <students xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
             <student studentID="1" givenName="Steffen" familyName="Siegmund"
               age="21" university="Rostock"/>
             <student studentID="2" givenName="Helena" familyName="Schmidt"
              age="23" university="Rostock"/>
         </students>'
    )~

   INSERT INTO XML_TRANS VALUES
        (1,
        '<?xml version="1.0" encoding="UTF-8"?>
   ```

```
        <xsl:stylesheet version="1.0"
             ...
        </xsl:stylesheet>'
   )~
```

3. XSLTRANSFORM 関数を呼び出して、以下のように XML 文書を変換します。

```
SELECT XSLTRANSFORM (XML_DOC USING XSLT_DOC AS CLOB(1M))
   FROM XML_DATA, XML_TRANS WHERE DOCID = 1 and XSLID = 1 ~
```

変換の出力は、以下の HTML ファイルになります。

```
<html>
<head>
<META http-equiv="Content-Type" content="text/html; charset=UTF-8">
</head>
<body>
<h1></h1>
<table border="1">
<th>
<tr>
<td width="80">StudentID</td>
<td width="200">Given Name</td>
<td width="200">Family Name</td>
<td width="50">Age</td>
</tr>
</th>
<tr>
<td>1</td>
<td>Steffen</td><td>Siegmund</td>
<td>21</td>
</tr>
<tr>
<td>2</td><td>Helena</td><td>Schmidt</td>
<td>23</td>
</tr>
</table>
</body>
</html>
```

XML レコードに含まれていない情報を追加したり、出力自体の性質を変更 (例えば標準 HTML 出力ではなく XHTML 出力を作成するなど) したりするために、ランタイム時に XSLT スタイルシートの動作を変更することもできます。振る舞いを変更するには、パラメーター・ファイルを使用して、XSLT プロセスにパラメーターを渡すことができます。このパラメーター・ファイル自体が XML 文書で、XSLT スタイルシート・ファイル内の同様のステートメントに対応する param ステートメントが含まれています。

スタイルシートで定義されているものの、以前の変換で使用されなかった、以下の 2 つのパラメーターについて考えてみましょう。

```
<xsl:param name="showUniversity"/>
<xsl:param name="headline"/>
```

これらのパラメーターを使用して XML 文書を変換するには、パラメーター・ファイルを表に保管して、XSLTRANSFORM 関数とともにそのファイルを使用します。

1. パラメーター・ファイルを保管するための表 PARAM_TAB を以下のように作成します。

```
CREATE TABLE PARAM_TAB (DOCID INTEGER, PARAM VARCHAR(1000))~
```

2. パラメーター・ファイルを以下のように作成します。

```
INSERT INTO PARAM_TAB VALUES
     (1,
     '<?xml version="1.0"?>
     <params xmlns="http://www.ibm.com/XSLTransformParameters">
          <param name="showUniversity" value="true"/>
          <param name="headline">The student list</param>
     </params>'
     )~
```

3. XSLTRANSFORM 関数を呼び出して、以下のように XML 文書を変換します。

```
SELECT XSLTRANSFORM (XML_DOC USING XSLT_DOC WITH PARAM AS CLOB(1M) )
   FROM XML_DATA X , PARAM_TAB P, XML_TRANS
   WHERE X.DOCID=P.DOCID and XSLID = 1 ~
```

この処理の出力は、以下の HTML ファイルになります。

```
<html>
<head>
<META http-equiv="Content-Type" content="text/html; charset=UTF-8">
</head>
<body>
<h1>The student list</h1>
<table border="1">
<th>
<tr>
<td width="80">StudentID</td>
<td width="200">Given Name</td>
<td width="200">Family Name</td>
<td width="50">Age</td>
<td width="200">University</td>
</tr>
</th>
<tr>
<td>1</td>
<td>Steffen</td><td>Siegmund</td>
<td>21</td>
<td>Rostock</td>
</tr>
<tr>
<td>2</td>
<td>Helena</td><td>Schmidt</td>
<td>23</td>
<td>Rostock</td>
</tr>
</table>
</body>
</html>
```

チュートリアルに戻る

第 3 章 XML ストレージ

タイプ XML の列に挿入する XML 文書は、デフォルトのストレージ・オブジェクトに入れることも、または直接基本表の行に入れることもできます。基本表の行での保管はユーザーの制御下にあり、小さな文書についてのみ使用できます。より大きな文書は、常にデフォルトのストレージ・オブジェクトに保管されます。

文書を基本表の行に保管する決定は、ストレージおよびパフォーマンス上の要件、および受け入れられたトレードオフに依存します。

XML ストレージ・オブジェクト
これは XML 文書を保管するためのデフォルトの方法です。必要なストレージ・スペースが 32 KB より大きい文書、またはページ・サイズよりも大きい文書は、ユーザーによるストレージの選択には関係なく、常にデフォルトのストレージ・オブジェクトに保管されます。デフォルトのストレージ・オブジェクトに保管することにより、最大 2 GB までのサイズの XML 文書を挿入および検索できます。

基本表行保管
必要なストレージ・スペースが 32 KB より小さい XML 文書については、XML 文書を基本表の行に直接保管することを選択できます。このオプションは、必要な入出力操作が少なくなるので、XML 文書を照会、挿入、更新、または削除する操作のパフォーマンスを向上させることができます。

表でデータ行の圧縮を使用可能にすると、デフォルトの XML ストレージ・オブジェクトおよび基本表の行に保管される XML 文書が圧縮の対象になります。圧縮を行うことによって、XML 文書に対する操作における入出力の効率が向上し、ストレージ・スペース要件が小さくなります。

XML ストレージ・オブジェクト

デフォルトでは、DB2 データベース・サーバーは、LOB データが他の表のコンテンツとは独立して格納されるのと同じようにして、XML ストレージ・オブジェクト中のタイプ XML の表列に含まれている XML 文書を保管します。

XML ストレージ・オブジェクトはその親表オブジェクトから分離していますが、その親表オブジェクトに従属しています。XML 表列の行に保管されている XML 値ごとに、DB2 は XML データ指定子 (XDS) というレコードを維持しています。このレコードは、ディスク上に保管されている XML データを、関連する XML ストレージ・オブジェクトから検索する場所を指定します。システム管理スペースに保管する場合、XML ストレージ・オブジェクトに関連したファイルはファイル・タイプ拡張子 .xda を持ちます。 XML ストレージ・オブジェクトは、XML データ域 (XDA) と呼ばれることもあります。

データベース中のサイズが 2 ギガバイトまでの XML 文書を保管できます。 XML データはかなり大きい場合があるので、他のデータに対するバッファリング・アクティビティーとは別に、XML データのバッファリング・アクティビティーをモニタ

ーする場合があるかもしれません。 XML ストレージ・オブジェクトのバッファー・プール・アクティビティー測定を支援するためにいくつかのモニター・エレメントが使用可能です。

XML ストレージ・オブジェクトを使用する XML 列のスペース所要量に関する追加情報は、「CREATE TABLE ステートメント」に INLINE LENGTH が指定されていない XML 列の「バイト・カウント」を参照してください。

XML の基本表行保管

オプションで、小から中サイズの XML 文書を、デフォルトの XML ストレージ・オブジェクトに保管する代わりに、基本表の行に保管できます。XML 文書の行保管は、構造化タイプのインスタンスを表の行にインラインで保管する方法に似ています。

基本表行保管を使用可能にする前に、各 XML 列のどれだけの行スペースを行保管専用にするかを決定する必要があります。専用にできるスペースの量は使用可能な最大行サイズに依存しています。さらにこの最大行サイズは、表が作成された表スペースのページ・サイズと、表の一部として指定した他の列に依存しています。使用可能な行スペースを計算するには、INLINE LENGTH が指定された XML 列について記載されている、『CREATE TABLE ステートメント』の『行サイズ』および『バイト・カウント』を参照してください。

基本表行保管の使用可能化

XML 列を含む表を作成する時、または XML 列を含む既存の表を変更する時に、XML 文書をデフォルトの XML ストレージ・オブジェクトにではなく基本表の行に保管するように指定できます。基本表行保管を使用可能にするには、行保管を使用する XML 列ごとに、CREATE TABLE または ALTER TABLE ステートメントで INLINE LENGTH キーワードを指定する必要があります。そのキーワードの後に基本表の行に保管する XML 文書の最大サイズをバイト単位で指定します。

既存の表の XML 列を変更しても、その列に既に保管されている XML 文書は、基本表の行に自動的に移動されない点に注意してください。 XML 文書を移動するには、UPDATE ステートメントですべての XML 文書を更新する必要があります。

制約事項

基本表行保管は、XML 文書の内部表記が、32 KB またはそれ以下 (行サイズがそれより少ない場合にはこれより小さくなります) から INLINE LENGTH オプションが指定された XML 列に必要なバイト・カウント・オーバーヘッドを引いた大きさである場合のみ使用できます。 32 KB というサイズは、表スペースのページ・サイズが 32 KB であることを想定しています。指定されたインライン長を超えた XML 文書を保管するときは、サイズ超過の文書は自動的にデフォルトの XML ストレージ・オブジェクトに保管されます。

XML 列のインライン長をいったん指定すると、XML 文書の行保管で使用するためのインライン長のサイズを増やすことはできますが、減らすことはできません。

例

以下の例は、SAMPLE データベースの PRODUCT 表の XML 列 DESCRIPTION で、XML 文書の基本表行保管を使用可能にするものです。この例では、基本表の行に保管する XML 文書の最大インライン長を 32000 バイトに設定し、オーバーヘッドのために必要な追加スペースのための余裕を残しています。値 32000 バイトを使用する場合、表スペースのページ・サイズは 32 KB であると想定されます。 XML 列が変更された後、UPDATE ステートメントにより XML 文書は基本表の行へ移動します。

```
ALTER TABLE PRODUCT
   ALTER COLUMN DESCRIPTION
   SET INLINE LENGTH 32000

UPDATE PRODUCT SET DESCRIPTION = DESCRIPTION
```

以下の例では、SAMPLE データベースの CUSTOMER 表に似た表 MYCUSTOMER が作成されますが、XML 列 Info に基本表行保管が指定されている点が異なります。内部表記が 2000 バイト以下である文書は、Info 列に挿入される時に基本表の行に保管されます。

```
CREATE TABLE MYCUSTOMER (Cid BIGINT NOT NULL,
   Info XML INLINE LENGTH 2000,
   History XML,
   CONSTRAINT PK_CUSTOMER PRIMARY KEY (Cid)) in IBMDB2SAMPLEXML
```

XML 文書用ストレージ要件

DB2 データベース中の XML 文書の占有するスペースの量は、未加工の XML 文書の初期サイズや他の要因により決定されます。

次のリストはこれらの要因の最も重要なものです。

文書構造
: 複雑なマークアップのタグ付けを含む XML 文書は、単純なマークアップによる文書と比較してストレージ・スペースのより大規模な量を必要とします。 例えば、それぞれが少量のテキストや短い属性値を持つ、多数のネストされた要素を持つ XML 文書は、主としてテキスト・コンテンツで構成される XML 文書よりも多くのストレージ・スペースを占有します。

ノード名
: 要素名、属性名、名前空間接頭部の長さなど、コンテンツでないデータもストレージ・サイズに影響を及ぼします。 未加工形式で 4 バイトを超えるこのタイプの情報単位は保管のため圧縮され、長いノード名に対して比較的高いストレージの効率性を提供します。

要素に対する属性の比率
: 通常、各要素に対する属性が多いほど、XML 文書のために必要となるストレージ・スペースの量は少なくなります。

文書コード・ページ
: 1 文字あたり 1 バイトを超える大きさを使用してエンコードする XML 文書は、1 バイト文字セットを使用する文書よりも大容量のストレージ・スペースを占有します。

圧縮 XML 列が含まれる表でデータ行の圧縮を使用可能にすると、XML 文書に必要なストレージ・スペースが少なくてすみます。

DB2 バージョン 9.5 以前の XML レコード形式を使用した XML 列が表に含まれる場合、表の XML ストレージ・オブジェクトのデータの圧縮はサポートされません。データ行の圧縮でこうした表を対象にした場合、表オブジェクトの表の行データのみが圧縮されます。表の XML ストレージ・オブジェクトのデータを圧縮に適したものにするには、ADMIN_MOVE_TABLE ストアード・プロシージャーを使用して表をマイグレーションしてから、データ行の圧縮を使用可能にします。

XML 文書をアーカイブする場合のデータ・タイプ

XML をシリアライズしたストリング・データをバイナリー・タイプまたは文字タイプの列に保管することは可能ですが、非 XML 列は XML データのアーカイブにのみ使用してください。XML データをアーカイブする場合の最適な列データ・タイプは、BLOB などのバイナリー・データ・タイプです。

文字列を使用してアーカイブすると、コード・ページ変換が起こり、文書が元の書式と矛盾する可能性があります。

第 4 章 XML データの挿入

XML 文書を挿入できるようにするには、XML 列を含む表を作成するか、既存の表に XML 列を追加する必要があります。

XML 列を持つ表の作成

XML 列を持つ表を作成するには、CREATE TABLE ステートメントで XML データ・タイプを持つ列を指定します。表には 1 つ以上の XML 列を含めることができます。

XML 列を定義する際、長さは指定しません。但し、DB2 データベースと交換されるシリアライズされた XML データは、タイプ XML の値あたり 2 GB に制限されるため、XML 文書の実効限度は 2 GB です。

LOB 列と同様に、XML 列は列の記述子のみを保持します。データは別個に保管されます。

注:

- 表でデータ行の圧縮を使用可能にすると、XML 文書に必要なストレージ・スペースが少なくてすみます。
- オプションで、小から中サイズの XML 文書を、デフォルトの XML ストレージ・オブジェクトに保管する代わりに、基本表の行に保管できます。

例: サンプル・データベースには、2 つの XML 列を持つ顧客データ用の表が含まれています。この定義は次のようになります。

```
CREATE TABLE Customer (Cid BIGINT NOT NULL PRIMARY KEY,
                       Info XML,
                       History XML)
```

例: VALIDATED 述部で、指定された XML 列の値が妥当性検査済みであるかを検査します。 VALIDATED 述部を使用して XML 列に表チェック制約を定義して、表に挿入される、または表で更新されるすべての文書が検査済みであることを確認します。

```
CREATE TABLE TableValid (id BIGINT,
                         xmlcol XML,
                         CONSTRAINT valid_check CHECK (xmlcol IS VALIDATED))
```

例: COMPRESS 属性を YES に設定すると、データ行の圧縮が使用可能になります。XML 列に保管された XML 文書は、行の圧縮の対象になります。行レベルでデータを圧縮することによって、繰り返しパターンを短いシンボル・ストリングに置換することができます。

```
CREATE TABLE TableXmlCol (id BIGINT,
                          xmlcol XML) COMPRESS YES
```

例: 以下の CREATE TABLE ステートメントによって、来院日付によってパーティション化された患者情報の表が作成されます。2000 年 1 月 1 日から 2006 年 12

月 31 日までのすべてのレコードが最初のパーティションに入ります。以後の新しいデータは、すべて 6 カ月ごとにパーティション化されます。

```
CREATE TABLE Patients ( patientID BIGINT, visit_date DATE, diagInfo XML,
    prescription XML )
    INDEX IN indexTbsp LONG IN ltbsp
    PARTITION BY ( visit_date )
       ( STARTING '1/1/2000' ENDING '12/31/2006',
         STARTING '1/1/2007' ENDING '6/30/2007',
            ENDING '12/31/2007',
            ENDING '6/30/2008',
            ENDING '12/31/2008',
            ENDING '6/30/2009' );
```

既存の表への XML 列の追加

既存の表に XML 列を追加するには、ALTER TABLE ステートメントに ADD 節を使用して、XML データ・タイプを持つ列を指定します。表には 1 つ以上の XML 列を含めることができます。

例 サンプル・データベースには、2 つの XML 列を持つ顧客データ用の表が含まれています。この定義は次のようになります。

```
CREATE TABLE Customer (Cid BIGINT NOT NULL PRIMARY KEY,
                       Info XML,
                       History XML)
```

Customer のコピーである MyCustomer という名前の表を作成し、顧客ごとの設定を記述するための XML 列を追加します。

```
CREATE TABLE MyCustomer LIKE Customer;
ALTER TABLE MyCustomer ADD COLUMN Preferences XML;
```

例: COMPRESS 属性を YES に設定すると、データ行の圧縮が使用可能になります。XML 列に保管された XML 文書は、行の圧縮の対象になります。行レベルでデータを圧縮することによって、繰り返しパターンを短いシンボル・ストリングに置換することができます。

```
ALTER TABLE MyCustomer ADD COLUMN Preferences XML COMPRESS YES;
```

例: 以下の CREATE TABLE ステートメントによって、来院日付によってパーティション化された患者情報の表が作成されます。2000 年 1 月 1 日から 2006 年 12 月 31 日までのすべてのレコードが最初のパーティションに入ります。以後の新しいデータは、すべて 6 カ月ごとにパーティション化されます。

```
CREATE TABLE Patients ( patientID INT, Name Varchar(20), visit_date DATE,
      diagInfo XML )
    PARTITION BY ( visit_date )
       ( STARTING '1/1/2000' ENDING '12/31/2006',
         STARTING '1/1/2007' ENDING '6/30/2007',
            ENDING '12/31/2007',
            ENDING '6/30/2008',
            ENDING '12/31/2008',
            ENDING '6/30/2009' );
```

以下の ALTER 表ステートメントは、患者の処方箋情報用の別の XML 列を追加します。

```
ALTER TABLE Patients ADD COLUMN prescription XML ;
```

XML 列への挿入

データを XML 列に挿入するには、SQL INSERT ステートメントを使用します。XML 列への入力は、XML 1.0 仕様に定義されているとおりの整形式 XML 文書であることが必要です。アプリケーション・データ・タイプは、XML、文字、またはバイナリー・タイプとすることができます。

DB2 データベース・サーバーがホスト変数データ・タイプを使用してエンコード情報の一部を判別できるように、XML データはリテラルではなく、ホスト変数から挿入することをお勧めします。

アプリケーション内の XML データは、シリアライズされたストリング形式となっています。そのデータを XML 列に挿入する際、データを XML 階層形式に変換しなければなりません。アプリケーションのデータ・タイプが XML データ・タイプである場合、DB2 データベース・サーバーはこの操作を暗黙的に実行します。アプリケーションのデータ・タイプが XML タイプではない場合、挿入操作を実行する際に XMLPARSE 関数を明示的に呼び出して、データをそのシリアライズされたストリング形式から XML 階層形式に変換できます。

文書の挿入時に、登録済みの XML スキーマに照らして XML 文書を妥当性検査することもできます。これは、XMLVALIDATE 関数を使用して行うことができます。

以下の例は、XML データを XML 列に挿入する方法を示しています。この例ではサンプルの Customer 表のコピーである表 MyCustomer を使用します。挿入される XML データは、ファイル c6.xml にあり、次のようになっています。

```
<customerinfo Cid="1015">
  <name>Christine Haas</name>
  <addr country="Canada">
    <street>12 Topgrove</street>
    <city>Toronto</city>
    <prov-state>Ontario</prov-state>
    <pcode-zip>N8X-7F8</pcode-zip>
  </addr>
  <phone type="work">905-555-5238</phone>
  <phone type="home">416-555-2934</phone>
</customerinfo>
```

例: JDBC アプリケーションで、XML データをバイナリー・データとしてファイル c6.xml から読み取り、そのデータを XML 列に挿入します。

```
PreparedStatement insertStmt = null;
String sqls = null;
int cid = 1015;
sqls = "INSERT INTO MyCustomer (Cid, Info) VALUES (?, ?)";
insertStmt = conn.prepareStatement(sqls);
insertStmt.setInt(1, cid);
File file = new File("c6.xml");
insertStmt.setBinaryStream(2, new FileInputStream(file), (int)file.length());
insertStmt.executeUpdate();
```

例: 静的組み込み C アプリケーションで、データをバイナリー XML ホスト変数から XML 列に挿入します。

```
EXEC SQL BEGIN DECLARE SECTION;
   sqlint64 cid;
   SQL TYPE IS XML AS BLOB (10K) xml_hostvar;
EXEC SQL END DECLARE SECTION;
...
```

```
cid=1015;
/* Read data from file c6.xml into xml_hostvar */
...
EXEC SQL INSERT INTO MyCustomer (Cid,Info) VALUES (:cid, :xml_hostvar);
```

XML 構文解析

XML 構文解析は、XML データをシリアライズされたストリング形式から階層形式に変換する処理です。

DB2 データベース・サーバーを使用して暗黙的に構文解析を行うことも、明示的に XML 構文解析を行うこともできます。

暗黙的な XML 構文解析 は、以下の場合に生じます。

- タイプ XML のホスト変数を使用してデータをデータベース・サーバーに渡すとき、またはタイプ XML のパラメーター・マーカーを使用するとき。

  データベース・サーバーが、ステートメント処理で使用するためにホスト変数またはパラメーター・マーカーの値をバインドするときに、構文解析を行います。

  この場合、暗黙的な構文解析を使用する必要があります。
- INSERT、UPDATE、DELETE、または MERGE ステートメントで、ストリング・データ・タイプ (文字、グラフィック、またはバイナリー) のホスト変数、パラメーター・マーカー、または SQL 式を XML 列に割り当てるとき。SQL コンパイラーが XMLPARSE 関数をステートメントに暗黙的に追加するとき、構文解析が生じます。

明示的な XML 構文解析 は、XMLPARSE 関数を入力 XML データに対して呼び出すことによって実行します。XMLPARSE の結果は、XML データ・タイプを受け入れるすべてのコンテキストで使用できます。例えば、その結果を XML 列に割り当てること、またはタイプ XML のストアード・プロシージャー・パラメーターとして使用することができます。

XMLPARSE 関数は、非 XML の文字またはバイナリー・データ・タイプを入力とします。組み込み動的 SQL アプリケーションでは、XMLPARSE の入力文書を表すパラメーター・マーカーを適切なデータ・タイプにキャストする必要があります。例:

```
INSERT INTO MyCustomer (Cid, Info)
 VALUES (?, xmlparse(document cast(? as clob(1k)) preserve whitespace))
```

組み込み静的 SQL アプリケーションでは、XMLPARSE 関数のホスト変数引数を XML タイプ (XML AS BLOB、XML AS CLOB、XML AS DBCLOB タイプ) として宣言することはできません。

XML 構文解析および空白処理

暗黙的または明示的な XML 構文解析中、データをデータベースに保管するときに、境界空白文字の保存または除去を制御できます。

XML 標準によれば、空白文字とは、読みやすさを促進するために文書中に置かれたスペース文字 (U+0020)、復帰 (U+000D)、改行 (U+000A)、またはタブ (U+0009) の

ことを指します。これらの記号がテキスト・ストリングの一部として出現する場合には、それは空白文字としては扱われません。

境界空白 は、要素と要素の間にある空白文字です。例えば、次の文書で、<a> と との間のスペース、および と との間のスペースは、境界空白です。

```
<a> <b> and between </b> </a>
```

XMLPARSE を明示的に呼び出すとき、STRIP WHITESPACE または PRESERVE WHITESPACE オプションを使用して境界空白の保存を制御します。デフォルトでは、境界空白は除去されます。

暗黙的な XML 構文解析の場合:

- 入力データ・タイプが XML タイプではないか、または XML データ・タイプにキャストされていない場合、DB2 データベース・サーバーは常に空白を除去します。
- 入力データ・タイプが XML データ・タイプの場合、CURRENT IMPLICIT XMLPARSE OPTION 特殊レジスターを使用して境界空白の保存を制御できます。この特殊レジスターを STRIP WHITESPACE または PRESERVE WHITESPACE に設定できます。デフォルトでは、境界空白は除去されます。

XML 妥当性検査を使用する場合、以下の状況において、DB2 データベース・サーバーは CURRENT IMPLICIT XMLPARSE OPTION 特殊レジスターを無視し、妥当性検査ルールだけを使用して空白の除去または保存を決めます。

```
xmlvalidate(? ACCORDING TO XMLSCHEMA ID schemaname)
xmlvalidate(?)
xmlvalidate(:hvxml ACCORDING TO XMLSCHEMA ID schemaname)
xmlvalidate(:hvxml)
xmlvalidate(cast(? as xml) ACCORDING TO XMLSCHEMA ID schemaname)
xmlvalidate(cast(? as xml))
```

上記の例では、? は XML データを表し、:hvxml は XML ホスト変数を表します。

XML 妥当性検査が空白処理にどのような影響を与えるかについては、XML 妥当性検査を参照してください。

XML 標準は、XML データ内の空白文字の除去または保存を制御する xml:space 属性を指定しています。 xml:space 属性は、暗黙的または明示的な XML 構文解析のための空白設定をオーバーライドします。

例えば、次の文書で の直前および直後にあるスペースは、XML 構文解析オプションに関係なく常に保存されます。それらのスペースが、属性 xml:space="preserve" のノード内にあるためです。

```
<a xml:space="preserve"> <b> <c>c</c>b </b></a>
```

ただし、次の文書で の直前および直後にあるスペースは、XML 構文解析オプションによって制御されます。それらのスペースが、属性 xml:space="default" のノード内にあるためです。

```
<a xml:space="default"> <b> <c>c</c>b </b></a>
```

非 Unicode データベースでの XML の構文解析

XML 文書が非 Unicode データベースに渡される場合、コード・ページ変換が、まず最初に文書がクライアントからターゲット・データベース・サーバーに渡される際に実行され、次に文書が DB2 XML パーサーに渡される際に実行されることがあります。タイプが XML のホスト変数またはパラメーター・マーカーを使用して XML 文書を渡す場合、コード・ページ変換は実行されません。文字データ・タイプ (CHAR、VARCHAR、CLOB、または LONG VARCHAR) を使用して XML 文書を渡す場合は、コード・ページ変換の結果として、XML データ中の、ターゲット・データベースのコード・ページの一部でない文字に関する代替文字に置き換わることがあります。

文字データ・タイプを使用して XML データを構文解析する場合に、代替文字に変換されないようにし、挿入される XML データが劣化しないようにするには、ソース文書中のすべてのコード・ポイントをターゲット・データベースのコード・ページの一部にします。このコード・ページの一部でない文字の場合、正しい Unicode コード・ポイントを指定した 10 進または 16 進文字エンティティー参照を使用できます。例えば、&#x003E または &#0062 を使用して > (より大) 符号文字を指定できます。

ENABLE_XMLCHAR 構成パラメーターを使用して、文字データ・タイプの場合に XML 構文解析を有効にするかどうかも制御できます。 ENABLE_XMLCHAR を「NO」に設定すると、文字データ・タイプの使用時に明示的および暗黙的な XML 構文解析が両方とも実行されないようになります。

XML 構文解析および DTD

入力データが内部文書タイプ宣言 (DTD) を含んでいるかまたは外部 DTD を参照する場合、XML 構文解析処理はそれらの DTD の構文も検査します。さらに、構文解析処理は以下を行います。

- 内部および外部 DTD で定義されたデフォルト値を適用する
- エンティティー参照およびパラメーター・エンティティーを展開する

例

以下の例は、XML 文書内の空白が様々な状態でどのように扱われるかを示します。

例: ファイル c8.xml には、次の文書が含まれています。

```
<customerinfo xml:space="preserve" Cid='1008'>
  <name>Kathy Smith</name>
  <addr country='Canada'>
    <street>14 Rosewood</street>
    <city>Toronto</city>
    <prov-state>Ontario</prov-state>
    <pcode-zip>M6W 1E6</pcode-zip>
  </addr>
  <phone type='work'>416-555-3333</phone>
</customerinfo>
```

JDBC アプリケーションで、ファイルから XML 文書を読み取り、サンプル Customer 表のコピーである表 MYCUSTOMER の XML 列 INFO にそのデータを挿入します。DB2 データベース・サーバーが、暗黙的な XML 構文解析操作を実行するようにします。

```
PreparedStatement insertStmt = null;
String sqls = null;
int cid = 1008;
sqls = "INSERT INTO MyCustomer (Cid, Info) VALUES (?, ?)";
insertStmt = conn.prepareStatement(sqls);
insertStmt.setInt(1, cid);
File file = new File("c8.xml");
insertStmt.setBinaryStream(2, new FileInputStream(file), (int)file.length());
insertStmt.executeUpdate();
```

空白処理は指定されていないので、通常であれば、デフォルトの動作として空白文字が除去されます。しかし、文書には xml:space="preserve" 属性が含まれているので、空白文字は保存されます。つまり、文書の中の復帰、改行、および要素間のスペースはそのまま残ります。

保管データを取り出した場合、その内容は次のようになります。

```
<customerinfo xml:space="preserve" Cid='1008'>
  <name>Kathy Smith</name>
  <addr country='Canada'>
    <street>14 Rosewood</street>
    <city>Toronto</city>
    <prov-state>Ontario</prov-state>
    <pcode-zip>M6W 1E6</pcode-zip>
  </addr>
  <phone type='work'>416-555-3333</phone>
</customerinfo>
```

例: 次の文書が BLOB ホスト変数 blob_hostvar 内にあると想定します。

```
<customerinfo xml:space="default" Cid='1009'>
  <name>Kathy Smith</name>
  <addr country='Canada'>
    <street>15 Rosewood</street>
    <city>Toronto</city>
    <prov-state>Ontario</prov-state>
    <pcode-zip>M6W 1E6</pcode-zip>
  </addr>
  <phone type='work'>416-555-4444</phone>
</customerinfo>
```

静的な組み込み C アプリケーションで、文書をホスト変数から表 MyCustomer の XML 列 Info に挿入します。ホスト変数は XML タイプではないので、XMLPARSE を明示的に実行する必要があります。境界空白を除去するには、STRIP WHITESPACE を指定します。

```
EXEC SQL BEGIN DECLARE SECTION;
   SQL TYPE BLOB (10K) blob_hostvar;
EXEC SQL END DECLARE SECTION;
...
EXEC SQL INSERT INTO MyCustomer (Cid, Info)
  VALUES (1009,
  XMLPARSE(DOCUMENT :blob_hostvar STRIP WHITESPACE));
```

文書には xml:space="default" 属性が含まれているので、XMLPARSE に指定された STRIP WHITESPACE が空白処理を制御します。つまり、文書の中の復帰、改行、および要素間のスペースは除去されます。

保管データを取り出した場合、次の内容の単一の行になります。

```
<customerinfo xml:space="default" Cid='1009'>
<name>Kathy Smith</name><addr country='Canada'><street>15 Rosewood</street>
<city>Toronto</city><prov-state>Ontario</prov-state><pcode-zip>M6W 1E6</pcode-zip>
</addr><phone type='work'>416-555-4444</phone></customerinfo>
```

例: C 言語アプリケーションで、ホスト変数 clob_hostvar に、内部 DTD を持つ次の文書が含まれています。

```
<!DOCTYPE prod [<!ELEMENT description (name,details,price,weight)>
   <!ELEMENT name (#PCDATA)>
   <!ELEMENT details (#PCDATA)>
   <!ELEMENT price (#PCDATA)>
   <!ELEMENT weight (#PCDATA)>
   <!ENTITY desc  "Anvil">
]>
<product pid=''110-100-01'' >
 <description>
 <name>&desc;</name>
 <details>Very heavy</details>
 <price>         9.99            </price>
 <weight>1 kg</weight>
 </description>
</product>'
```

サンプルの PRODUCT 表のコピーである表 MYPRODUCT にデータを挿入します。

```
EXEC SQL BEGIN DECLARE SECTION;
   SQL TYPE CLOB (10K) clob_hostvar;
EXEC SQL END DECLARE SECTION;
...
EXEC SQL insert into
  Product ( pid, name, Price, PromoPrice, PromoStart, PromoEnd, description )
  values ( '110-100-01','Anvil', 9.99, 7.99, '11-02-2004','12-02-2004',
  XMLPARSE ( DOCUMENT :clob_hostvar STRIP WHITESPACE ));
```

XMLPARSE は空白の除去を指定しているので、文書内の境界空白は除去されます。さらに、データベース・サーバーが XMLPARSE を実行するとき、エンティティー参照 &desc; がその値に置き換えられます。

保管データを取り出した場合、次の内容の単一の行になります。

```
<product pid="110-100-01"><description><name>Anvil
</name><details>Very heavy</details><price>         9.99            </price>
<weight>1 kg</weight></description></product>
```

XML データ保全性

XML 文書が特定の規則に従っていること、または特定の処理要件を満たしていることを確認する必要があるとき、追加の XML データ保全性検査を実行するか、またはアクションが実行される前に満たされる必要のある追加の条件を指定することができます。

XML データの保全性を保証するためのいくつかの異なる方法を使用できます。どの方法を選択するかは、特定のデータ保全性および処理の要件に依存します。

XML 文書を索引付けする場合、索引付けする XML パターンによってノードが修飾されるすべての文書にある XML 列内で、固有のものとなるように強制することもできます。詳しくは、『UNIQUE キーワードの意味体系』を参照してください。

XML 列のチェック制約

チェック制約を使用すると、XML 列に関して特定の制限事項を設けることができます。XML 列にデータを挿入または更新しようとすると、必ずこの制約が実施されます。この制約によって指定された基準が真と評価される場合にのみ、操作が実行されます。

XML 文書を扱う際の重要な考慮事項は、そうした文書が XML スキーマに対して既に妥当性検査されているかどうかという点です。特定の妥当性検査基準を満たしている文書のみを照会、挿入、更新、または削除することが必要な場合、VALIDATED 述部を使用してその基準を指定します。チェック制約は XML 文書の妥当性検査は行いません。XML 文書に対して既に妥当性検査が行われているかどうかだけを検査します。[2]

VALIDATED 述部は、*XML-expression* で指定された値 (XML データ・タイプでなければなりません) の妥当性検査状態を検査します。オプションの ACCORDING TO 節を指定しないと、妥当性検査で使用される XML スキーマは結果に影響を及ぼしません。チェック制約は XML 文書そのものを妥当性検査しません。文書の現在の妥当性検査状態を制約 (IS VALIDATED または IS NOT VALIDATED) によって検査するだけです。ACCORDING TO 節を指定する場合、*XML-expression* で指定された値を妥当性検査するのに使用する XML スキーマは、その ACCORDING TO 節によって特定される XML スキーマでなければなりません。XML スキーマを VALIDATED 述部で参照するには、その前に XML スキーマ・リポジトリーにそのスキーマを登録する必要があります。

注:

- チェック制約には、参照する XML スキーマと従属関係があります。XML スキーマの XSR オブジェクトが削除されると、そのスキーマを参照する制約も削除されます。
- XML 列は NOT NULL 制約をサポートしています。
- XML 列は、XML 妥当性検査用に定義されるインフォメーショナル制約をサポートしています。

チェック制約の評価

チェック制約は、IS VALIDATED 述部の結果に基づいて文書の妥当性検査状態を検査します。指定の条件が満たされると制約は真と評価され、満たされない場合には結果は偽と評価されます。*XML-expression* によって指定された値が NULL の場合、述部の結果は不明です。

XML-expression によって指定された値が NULL ではなく、さらに以下のいずれかの条件を満たす場合、VALIDATED 述部の結果は**真**になります。

- ACCORDING TO 節が指定されておらず、*XML-expression* で指定された値が妥当性検査されている場合。または

2. XML 列に XML 文書を保管する前にそれらの文書の妥当性検査を自動的に行う必要がある場合には、BEFORE トリガーを使用できます。

- ACCORDING TO 節が指定されていて、その ACCORDING TO 節によって特定されるいずれかの XML スキーマを使用して *XML-expression* によって指定される値が妥当性検査されている場合。

XML-expression によって指定された値が NULL ではなく、さらに以下のいずれかの条件を満たす場合、述部は**偽**になります。

- ACCORDING TO 節が指定されておらず、*XML-expression* で指定された値が妥当性検査されていない場合。または
- ACCORDING TO 節が指定されていて、その ACCORDING TO 節によって特定されるいずれかの XML スキーマを使用して *XML-expression* によって指定される値が妥当性検査されていない場合。

オプションの ACCORDING TO 節が指定されると、 *XML-expression* によって指定された値が妥当性検査されていない場合、または *XML-expression* によって指定された値が妥当性検査されたものの指定のいずれかの XML スキーマに準拠していない場合には、IS NOT VALIDATED は真を戻します。

同等の式

次の VALIDATED 述部は

```
value1 IS NOT VALIDATED optional-clause
```

以下の検索条件と同等です。

```
NOT(value1 IS VALIDATED optional-clause)
```

例

例: 妥当性検査された XML 文書だけを選択します。列 XMLCOL が表 T1 に定義されているとします。任意の XML スキーマによって妥当性検査された XML 値だけを取り出します。

```
SELECT XMLCOL FROM T1
  WHERE XMLCOL IS VALIDATED
```

例: 妥当性検査されていない場合には値の挿入または更新は行えないという規則を実施します。列 XMLCOL が表 T1 に定義されていて、XMLCOL にチェック制約を追加するとします。

```
ALTER TABLE T1 ADD CONSTRAINT CK_VALIDATED
  CHECK (XMLCOL IS VALIDATED)
```

例: 制約 INFO_CONSTRAINT はインフォメーショナル制約です。妥当性検査に合格しないと値の挿入または更新は行えないという規則は実施されません。

```
CREATE TABLE xmltab (ID INT,
     DOC XML, CONSTRAINT INFO_CONSTRAINT CHECK (DOC IS VALIDATED) NOT ENFORCED)
```

インフォメーショナル制約は、照会のパフォーマンスを向上されるために使用されます。

XML データのトリガー処理

トリガーは、INSERT、UPDATE、または DELETE ステートメントによって実行される操作に応答して作動します。XML データを処理する場合は、CREATE

TRIGGER ステートメントを使用して、XML 列に対する UPDATE オプションを指定した BEFORE トリガーまたは AFTER トリガーを作成することができます。XML 列が含まれている表に対する INSERT または DELETE オプションを指定した BEFORE トリガーまたは AFTER トリガーを作成することもできます。

トリガー本体では、影響を受ける行内にある XML タイプの列を参照する遷移変数を使用できるのは、XMLVALIDATE 関数を使用して妥当性検査を行う場合だけであり、XML 列値は NULL に設定されるか未変更のままになります。

INSERT または UPDATE ステートメントで BEFORE トリガーを使用すると、XML 列に XML 文書が保管される前に、それらの文書に対して自動的に妥当性検査を行えます。登録された XML スキーマに対する XML 文書の妥当性検査の実行は任意ですが、これにより妥当な XML 文書だけが挿入または更新されるようになるので、データ整合性が問題になる場合には実行を強くお勧めします。

このトリガーは、設定されている条件が満たされると起動されます。条件を指定しないと、トリガーは毎回起動されます。必要なときだけ XML スキーマに対して XML 文書の妥当性検査をトリガーするには、BEFORE トリガーの WHEN 節を使用して XML 列の条件を指定できます。WHEN 節で、トリガーを起動するか否かを決定するための前提条件となる XML 文書の妥当性検査状態を指定します。つまり、トリガーの起動条件として妥当性検査があらかじめ行われている (IS VALIDATED) のか、妥当性検査は行われていない (IS NOT VALIDATED) のかです。オプションとして、ACCORDING TO XMLSCHEMA 節を指定して 1 つ以上の XML スキーマを組み込むことができます。この節は、制約の評価をするにあたり考慮すべき XML スキーマをトリガーに指示します。

注: WHEN 節を指定したトリガーには、余分のオーバーヘッドが生じます。XML 文書の挿入前に妥当性検査を必ず行う場合には、WHEN 節を省略できます。

XML スキーマを参照するトリガーは、そのスキーマと従属関係があります。XML スキーマを参照するには、その前に XML スキーマ・リポジトリーにそのスキーマを登録する必要があります。トリガーと従属関係を持つ XML スキーマが後に XML スキーマ・リポジトリーから削除されると、トリガーには作動不能というマークが付けられます。

例 1: SAMPLE データベースの PRODUCT 表に XML 文書を挿入する前に、新規商品に関する説明が含まれるそれらの XML 文書を自動的に妥当性検査する BEFORE トリガーを作成します。このトリガーは、XML 文書を更新する前は毎回起動されます。

```
CREATE TRIGGER NEWPROD NO CASCADE BEFORE INSERT ON PRODUCT
  REFERENCING NEW AS N
  FOR EACH ROW MODE DB2SQL
  BEGIN ATOMIC
     SET (N.DESCRIPTION) = XMLVALIDATE(N.DESCRIPTION
     ACCORDING TO XMLSCHEMA URI 'http://posample.org/product.xsd');
  END
```

例 2: XML スキーマを product2.xsd へ発展させた後、元の XML スキーマ product.xsd に対して妥当であることが確認された保管済みの XML 文書は、発展したスキーマでも妥当であることが保証されます。しかし、こうした XML 文書に行われる更新に関しても、展開されたスキーマ product2.xsd に対して妥当であること

を確認したい場合があります。product2.xsd を XML スキーマ・リポジトリーに登録した後、BEFORE UPDATE トリガーを使用すると、更新を行う前に XML 文書に対して妥当性検査を行えます。

```
CREATE TRIGGER UPDPROD NO CASCADE BEFORE UPDATE ON PRODUCT
  REFERENCING NEW AS N
  FOR EACH ROW MODE DB2SQL
  BEGIN ATOMIC
    SET (N.DESCRIPTION) = XMLVALIDATE(N.DESCRIPTION
    ACCORDING TO XMLSCHEMA ID product2);
  END
```

例 3: 挿入または更新された顧客レコードを別の表に記録します。このためには、2 つのトリガー、つまり 1 つは新しく挿入されたレコード用の AFTER INSERT、もう 1 つは更新されたレコード用の AFTER UPDATE を作成する必要があります。この例では、サンプル Customer 表のコピーである MyCustomer 表の XML 列 Info にトリガーを作成します。これらのトリガーは、レコードが MyCustomer 表内で挿入または更新されるたびに、そのレコードがタイム・スタンプとカスタマー ID と共に CustLog という表に書き込まれるようにします。次の例 4 は、CustLog 表に書き込むと同時に実際のデータのコピーを保持する方法を示しています。

最初に、MyCustomer 表に AFTER INSERT トリガーを作成します。

```
CREATE TRIGGER INSAFTR
  AFTER INSERT ON MyCustomer
  REFERENCING NEW AS N
  FOR EACH ROW
  BEGIN ATOMIC
    INSERT INTO CustLog VALUES(N.CID, CURRENT TIMESTAMP, 'Insert');
  END
```

次に、MyCustomer 表に AFTER UPDATE トリガーを作成します。

```
CREATE TRIGGER UPDAFTR
  AFTER UPDATE OF Info
  ON MyCustomer
  REFERENCING NEW AS N
  FOR EACH ROW
  BEGIN ATOMIC
    INSERT INTO CustLog VALUES(N.CID, CURRENT TIMESTAMP, 'Update');
  END
```

例 4: この例は、挿入または更新された顧客レコードに対する監査ログの役割を果たすように表をセットアップする方法を示しています。例 3 と同様に、2 つのトリガー、つまり 1 つは新しく挿入されたレコード用の AFTER INSERT、もう 1 つは更新されたレコード用の AFTER UPDATE を作成します。サンプル Customer 表のコピーである MyCustomer 表の XML 列 Info にトリガーを作成します。MyCustomer 表でレコードが挿入または更新されるたびに、トリガーは、レコードがタイム・スタンプ、カスタマー ID、および XML タイプの列 Info の内容と共に CustLog という表に書き込まれるようにします。

最初に、MyCustomer 表に AFTER INSERT トリガーを作成します。

```
CREATE TRIGGER INSAFTR
  AFTER INSERT ON MyCustomer
  REFERENCING NEW AS N
  FOR EACH ROW
```

```
  BEGIN ATOMIC
    INSERT INTO CustLog VALUES(N.CID, CURRENT TIMESTAMP, 'Insert',
      (SELECT Info FROM MyCustomer WHERE CID = N.CID));
  END
```

次に、MyCustomer 表に AFTER UPDATE トリガーを作成します。

```
CREATE TRIGGER UPDAFTR
  AFTER UPDATE OF Info
  ON MyCustomer
  REFERENCING NEW AS N
  FOR EACH ROW
  BEGIN ATOMIC
    INSERT INTO CustLog VALUES(N.CID, CURRENT TIMESTAMP, 'Update',
      (SELECT Info FROM MyCustomer WHERE CID = N.CID));
  END
```

XML 妥当性検査

XML 妥当性検査は、XML 文書の構造、内容、およびデータ・タイプが有効かどうかを判別する処理です。 XML 妥当性検査は、XML 文書内の*無視できる空白文字*も除去します。

妥当性検査は任意ですが、これにより XML 文書が整形式であるだけでなく、XML スキーマによって指定された規則に順守していることが確認できるので、データ整合性が問題になる場合には実行を強くお勧めします。

XML スキーマに対してのみ XML 文書を妥当性検査できることに注意してください。DTD に対して XML 文書を妥当性検査することはできません。

XML 文書を妥当性検査するには、XMLVALIDATE 関数を使用します。 DB2 データベースで XML 文書の挿入または更新を行う SQL ステートメントで XMLVALIDATE を指定できます。また XML 文書に対して自動妥当性検査を行うには、XML 列に BEFORE トリガーを使用して XMLVALIDATE 関数を呼び出すことができます。XML 文書を強制的に妥当性検査する場合、チェック制約を作成します。

XMLVALIDATE 関数を呼び出すには、その前に XML スキーマを構成するすべてのスキーマ文書を組み込み XML スキーマ・リポジトリーに登録する必要があります。 XMLVALIDATE を使用して妥当性検査するには、XML 文書自体がデータベース内にある必要はありません。

XML 妥当性検査および無視できる空白文字

XML 標準によれば、*空白文字* とは、読みやすさを促進するために文書中に置かれたスペース文字 (U+0020)、復帰 (U+000D)、改行 (U+000A)、またはタブ (U+0009) のことを指します。これらの記号がテキスト・ストリングの一部として出現する場合には、それは空白文字としては扱われません。

無視できる空白文字 は、XML 文書から除去できる空白文字です。 XML スキーマ文書は、どの空白文字が無視できる空白文字であるかを決定します。 XML 文書が要素のみの複合タイプ (他の要素だけを含む要素) を定義している場合、要素同士の間の空白文字は無視できます。 XML スキーマがストリング以外のタイプを含む単純な要素を定義している場合、その要素内の空白文字は無視できます。

例: サンプルの product.xsd XML スキーマ文書内にある description 要素は、次のように定義されています。

```
<xs:element name="description" minOccurs="0" maxOccurs="unbounded">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="name" type="xs:string" minOccurs="0" />
      <xs:element name="details" type="xs:string" minOccurs="0" />
      <xs:element name="price" type="xs:decimal" minOccurs="0" />
      <xs:element name="weight" type="xs:string" minOccurs="0" />
 ...
 </xs:complexType>
</xs:element>
```

description 要素は他の要素しか含んでいないので、要素のみの複合タイプとなります。そのため、description 要素では、要素同士の間の空白文字は無視できる空白文字です。 price 要素も、ストリング以外のタイプを含む単純な要素なので、無視できる空白文字を含むことがあります。

XMLVALIDATE 関数では、妥当性検査に使用する XML スキーマ文書を明示的に指定できます。 XML スキーマ文書を指定しない場合、DB2 データベース・サーバーは入力文書内で、XML スキーマ文書を識別する xsi:schemaLocation または xsi:noNamespaceSchemaLocation 属性を検索します。 xsi:schemaLocation または xsi:noNamespaceSchemaLocation 属性は、XML スキーマ仕様で定義されており、*XML スキーマ・ヒント* と呼ばれます。 xsi:schemaLocation 属性は、XML スキーマ文書を見つけるために役立つ値の対を 1 つ以上含んでいます。それぞれの対の最初の値は名前空間であり、2 番目の値はその名前空間の XML スキーマを見つけることのできる場所を示すヒントとなります。 xsi:noNamespaceSchemaLocation 値は、ヒントだけを含んでいます。 XML スキーマ文書が XMLVALIDATE 関数に指定されている場合、その指定は xsi:schemaLocation または xsi:noNamespaceSchemaLocation 属性をオーバーライドします。

以下の例は、スキーマ product が XML スキーマ・リポジトリー (XSR) に登録されていることを前提としています。登録を完了するために、次のような CLP ステートメントを使用できます。

```
REGISTER XMLSCHEMA http://posample.org/product.xsd FROM product.xsd ¥
AS myschema.product
COMPLETE XMLSCHEMA myschema.product
```

またはその代わりに、XML スキーマが単一のスキーマ文書から構成されているので、次の単一のステートメントを使用して XML スキーマを登録し、登録を完了することもできます。

```
REGISTER XMLSCHEMA http://posample.org/product.xsd FROM product.xsd ¥
AS myschema.product COMPLETE
```

例: 次のように表 MyProduct を作成すると仮定します。

```
CREATE TABLE MyProduct LIKE Product
```

動的 SQL アプリケーションを使用して次の文書を MyProduct 表内の XML 列 Info に挿入し、この XML データを、MyProduct 表と同じデータベース・サーバー上の XML スキーマ・リポジトリーにある XML スキーマ文書 product.xsd に照らして妥当性検査するとします。

```
<product xmlns="http://posample.org" pid=''110-100-01'' >
 <description>
 <name>Anvil</name>
 <details>Very heavy</details>
 <price>         9.99            </price>
 <weight>1 kg</weight>
 </description>
</product>'
```

INSERT ステートメントの中で、XMLVALIDATE 関数は妥当性検査に使用する XML スキーマを指定します。

```
Insert into MyProduct
 (pid, name, Price, PromoPrice, PromoStart, PromoEnd, description)
 values ( '110-100-01','Anvil', 9.99, 7.99, '11-02-2004','12-02-2004',
 XMLVALIDATE(? ACCORDING TO XMLSCHEMA ID myschema.product))
```

保管データを取り出すと、XMLVALIDATE が無視できる空白文字を除去した箇所を確認できます。取り出したデータは、次の内容の単一の行です。

```
<product xmlns="http://posample.org" pid="110-100-01"><description><name>Anvil
</name><details>Very heavy</details><price>9.99</price><weight>1 kg</weight>
</description></product>
```

product スキーマは、name、details、price、および weight 要素の前後に空白を定義しています。price 要素内の空白文字は無視できる空白文字なので XMLVALIDATE はそれを除去します。

妥当性検査された文書だけを XML 列に挿入する必要がある場合、または妥当性検査された文書だけを XML 列から取り出す必要がある場合は、VALIDATED 述部を使用します。

XML 文書の挿入または更新前にその文書が妥当性検査されたかどうかを検査するには、XML 列に VALIDATED 述部を含むチェック制約を作成します。妥当性検査済みの文書だけを XML 列から取り出すには、または妥当性検査を受けずに挿入された文書だけを取り出すには、VALIDATED 述部を WHERE 節内で使用します。特定の XML スキーマに従って XML 文書が妥当性検査されたかどうかを確認する必要がある場合、それらの XML スキーマを VALIDATED 述部の ACCORDING TO XMLSCHEMA 節で指定します。

VALIDATED 述部は、トリガーの一部としても使用できます。XML 列での XML 文書の挿入または更新前に妥当性検査が行われていない XML 文書に対する妥当性検査をトリガーするには、XMLVALIDATE 関数を呼び出すため、WHEN 節内の XML 列に VALIDATED 述部を含む BEFORE トリガーを作成します。

例: MyCustomer 表の Info 列から妥当性検査済みの XML 文書だけを検索すると仮定します。次のような SELECT ステートメントを実行してください。

```
SELECT Info FROM MyCustomer WHERE Info IS VALIDATED
```

例: MyCustomer 表の Info 列に妥当性検査済みの XML 文書だけを挿入すると仮定します。チェック制約を定義して、この条件を適用できます。次の方法で、MyCustomer 表を変更します。

```
ALTER TABLE MyCustomer ADD CONSTRAINT CK_VALIDATED CHECK (Info IS VALIDATED)
```

ただし、このステートメントを発行すると、有効な文書だけが表に正常に挿入または更新されることになるので、前の例における VALIDATED 述部の使用は不要になります。

例: 次の文書を customer スキーマによって妥当性検査するものの、文書をデータベースに保管しないとします。

```
<customerinfo xml:space="default"
  xmlns="http://posample.org"
  Cid='1011'>
  <name>Kathy Smith</name>
  <addr country='Canada'>
  <street>25 Rosewood</street>
  <city>Toronto</city>
  <prov-state>Ontario</prov-state>
  <pcode-zip>M6W 1E6</pcode-zip>
  </addr>
  <phone type='work'>416-555-6676</phone>
</customerinfo>
```

文書をアプリケーション変数に割り当てたと仮定します。次のような VALUES ステートメントを使用して、妥当性検査を行うことができます。

```
VALUES XMLVALIDATE(? according to xmlschema id myschema.customer)
```

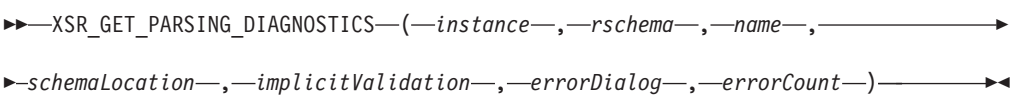
この文書は XML スキーマから見て有効なものなので、VALUES ステートメントはその文書を含む結果表を戻します。文書が無効な場合は、VALUES は SQL エラーを戻します。

XSR_GET_PARSING_DIAGNOSTICS ストアード・プロシージャー

ストアード・プロシージャー XSR_GET_PARSING_DIAGNOSTICS は、XML 文書の構文解析または妥当性検査の際に発生する、詳細なエラー情報を生成します。このストアード・プロシージャーを使用して、構文解析と妥当性検査の両方のエラーを報告することもできますし、構文解析エラーだけを報告することもできます。

XML ファイルの構文解析または妥当性検査時にエラーが発生する場合、XSR_GET_PARSING_DIAGNOSTICS ストアード・プロシージャーを DB2 コマンド・ウィンドウから呼び出すか、このストアード・プロシージャーをアプリケーションに追加します。例えば、XMLVALIDATE スカラー関数を使用して XML 文書を妥当性検査したときにエラーが発生する場合、このストアード・プロシージャーを使用して妥当性検査時に生じたエラーの詳細情報を生成します。

構文



ストアード・プロシージャーのスキーマは SYSPROC です。

許可

XML スキーマ許可: 妥当性検査に使用する XML スキーマは、使用する前に XML スキーマ・リポジトリーに登録しておく必要があります。ストアード・プロシージャーの許可 ID によって保持されている特権および権限には、少なくとも以下のいずれかが含まれていなければなりません。

- 妥当性検査で使用する XML スキーマに対する USAGE 特権
- DBADM 権限

プロシージャーのパラメーター

instance
: XML 文書の内容を含むタイプ BLOB(30M) の入力引数。 XML 文書を提供する必要があります。この値を NULL にすることはできません。

rschema
: XML スキーマ・リポジトリーに登録される 2 つの部分から成る XSR オブジェクト名の SQL スキーマの部分を指定する、VARCHAR(128) タイプの入力引数。この値は NULL にすることができます。この値が NULL の場合、SQL スキーマの部分は CURRENT SCHEMA 特殊レジスターの現行値と見なされます。

name
: XML スキーマ・リポジトリーに登録される 2 つの部分から成る XSR オブジェクト名のスキーマ名を指定する、VARCHAR(128) タイプの入力引数。 XML スキーマの完全な SQL ID は *rschema.name* です。これは、XSR 内のすべてのオブジェクト内で固有でなければなりません。この値は NULL にすることができます。

schemaLocation
: 1 次 XML スキーマ文書のスキーマ・ロケーションを示す、タイプ VARCHAR(1000) の入力引数。この引数は、XML スキーマの外部名です。つまり、1 次文書は、XML インスタンス文書内で `xsi:schemaLocation` 属性を使用して識別できます。この値は NULL にすることができます。

implicitValidation
: インスタンス文書のスキーマ・ロケーションを XML スキーマの検索に使用するかどうかを示す、タイプ INTEGER の入力引数。この値を NULL にすることはできません。使用しない場合は 0、使用する場合は 1 の値にします。

  0 インスタンス文書のスキーマ・ロケーションを使用しません。0 の値が渡された場合、次のいずれかの方式を使用してスキーマを指定できます。

  - XML スキーマ・リポジトリーに登録されているスキーマの XSR オブジェクト名を引数 *rschema* および *name* として指定します。
  - 引数 *schemaLocation* を使用してスキーマ・ロケーションを指定します。

  XSR オブジェクト名と *schemaLocation* の両方が指定された場合、XSR オブジェクト名が使用されます。どちらも指定しないと、妥当性検査は実行されません。XML 構文解析のみが実行され、XML 構文解析エラーが報告されます。

  1 インスタンス文書の `xsi:schemaLocation` 属性の値から取得したスキーマ・ロケーションを使用します。

入力文書に対する妥当性検査は、以前に XML スキーマ・リポジトリーに登録されている XML スキーマ文書に照らして実行されます。

implicitValidation 引数の値が 0 で、*rschema*、*name*、および *schemaLocation* 引数の値が NULL の場合、インスタンス文書は妥当性検査なしで構文解析されます。

errorDialog
構文解析および妥当性検査エラーをリストする UTF-8 XML 文書を含む、タイプ VARCHAR(32000) の出力引数。この文書は、少なくとも 1 つのエラーがある場合にのみ生成されます。

errorCount
XML 構文解析エラーおよび妥当性検査エラーの総数を指定する、タイプ INTEGER の出力引数。

使用法

XML 文書を登録済み XML スキーマに照らして妥当性検査するには、次の 3 つの方法があります。

- *rschema* および *name* 引数を使用して、XML スキーマの XSR オブジェクト名を指定します。
- *schemaLocation* 引数を使用してスキーマ・ロケーションを指定します。
- XML インスタンス文書が `xsi:schemaLocation` 属性の値としてスキーマを指定している場合、*implicitValidation* を 1 に設定します。

XSR_GET_PARSING_DIAGNOSTICS ストアード・プロシージャーの使用時に構文解析または妥当性検査のエラーが発生する場合、*errorDialog* および *errorCount* 出力パラメーターが設定されます。 *errorDialog* には、エラーをリストした XML 文書が含まれます。 CLI、Java または C++ を使用するアプリケーションから XSR_GET_PARSING_DIAGNOSTICS ストアード・プロシージャーを呼び出し、パラメーター・マーカーを使用してストアード・プロシージャーの出力を取得することができます。

詳細な XML 構文解析および妥当性検査エラーの表示

XSR_GET_PARSING_DIAGNOSTICS ストアード・プロシージャーから **errorDialog** 出力パラメーターに含まれる情報を利用して、XML 構文解析および妥当性検査のエラーを解決することができます。

XMLVALIDATE スカラー関数を使用して XML 文書を妥当性検査したときにエラーが発生する場合、 XSR_GET_PARSING_DIAGNOSTICS ストアード・プロシージャーを使用して詳細なエラー情報を生成します。以下の例は、XSR_GET_PARSING_DIAGNOSTICS ストアード・プロシージャーを簡単な XML スキーマおよび XML 文書とともに使用して、詳細な妥当性検査エラー情報を生成します。

XML スキーマのサンプル

この例では、以下の XML スキーマ定義 (XSD) を使用します。スキーマにはいくつかの要素が含まれますが、Age 要素には整数の型属性が割り当てられます。

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           targetNamespace="http://my.simpletest"
           xmlns="http://my.simpletest"
           elementFormDefault="qualified">
<xs:element name="Person">
 <xs:complexType>
   <xs:sequence>
     <xs:element name="Name">
      <xs:complexType>
       <xs:sequence>
        <xs:element name="FirstName" type="xs:string"/>
        <xs:element name="LastName" type="xs:string"/>
       </xs:sequence>
      </xs:complexType>
     </xs:element>
    <xs:element name="Age" type="xs:integer"/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

XML 文書のサンプル

次の XML 文書を、サンプル XML スキーマで妥当性検査します。文書は、一部のスキーマ規則に準拠していません。 Age 要素は数値ではなく、Notes® 要素は Person 要素のサブ要素としてスキーマで定義されていません。

```
<?xml version="1.0"?>
<Person xmlns="http://my.simpletest"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://my.simpletest http://my.simpletest/simple">
  <Name>
   <FirstName>Thomas</FirstName>
   <LastName>Watson</LastName>
  </Name>
  <Age>30x</Age>
  <Notes/>
<Person>
```

XML スキーマを登録するためのコマンド

XSR_GET_PARSING_DIAGNOSTICS ストアード・プロシージャーを使用して XML 文書を妥当性検査する前に、妥当性検査に使用する XML スキーマは、DB2 XML スキーマ・リポジトリー (XSR) に登録しておく必要があります。以下の REGISTER XMLSCHEMA コマンドは、スキーマが c:¥temp¥simpleschema.xsd にあり、SQL スキーマが USER1 であることを想定しています。

```
REGISTER XMLSCHEMA 'http://my.simpletest/simple'
   FROM 'file:///c:/temp/simpleschema.xsd'
   AS user1.simple COMPLETE
```

詳細な妥当性検査エラー情報を生成するための呼び出し

以下の呼び出しは、CLP から XSR_GET_PARSING_DIAGNOSTICS ストアード・プロシージャーを使用して、妥当性検査エラー情報を生成します。

```
CALL XSR_GET_PARSING_DIAGNOSTICS(
blob('<?xml version="1.0"?>
<Person xmlns="http://my.simpletest"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://my.simpletest http://my.simpletest/simple">
<Name>
```

```
  <FirstName>Thomas</FirstName>
  <LastName>Watson</LastName>
 </Name>
 <Age>30x</Age>
 <Notes />
</Person>
'),'','','',1,?,?)@
```

詳細な妥当性検査エラー情報

XSR_GET_PARSING_DIAGNOSTICS ストアード・プロシージャーの呼び出しは、以下の出力を戻します。**errorDialog** パラメーターの値は、詳細な妥当性検査エラー情報が含まれる XML 文書です。 XML 文書では、errText、location、lineNum および colNum 要素の内容は、特定のエラーおよびエラーの場所を識別します。以下の出力が、前の呼び出しが CLP コマンド行から実行されると標準出力に書き込まれます。

```
Value of output parameters
--------------------------
Parameter Name  : ERRORDIALOG
Parameter Value : <ErrorLog>
<XML_Error parser="XML4C">
  <errCode>238</errCode>
  <errDomain>http://apache.org/xml/messages/XML4CErrors</errDomain>
  <errText>Datatype error: Type:InvalidDatatypeValueException,
    Message:Value '30x' does not match regular expression facet '[+¥-]?[0-9]+' .
  </errText>
  <lineNum>1</lineNum>
  <colNum>272</colNum>
  <location>/Person/Age</location>
  <schemaType>http://www.w3.org/2001/XMLSchema:integer</schemaType>
  <tokenCount>2</tokenCount>
  <token1>30x</token1>
  <token2>13</token2>
</XML_Error>
<XML_Error parser="XML4C">
  <errCode>2</errCode>
  <errDomain>http://apache.org/xml/messages/XMLValidity</errDomain>
  <errText>Unknown element 'Notes' </errText>
  <lineNum>1</lineNum>
  <colNum>282</colNum>
  <location>/Person</location>
  <schemaType>http://www.w3.org/2001/XMLSchema:integer</schemaType>
  <tokenCount>2</tokenCount>
  <token1>Notes</token1>
  <token2>37</token2>
</XML_Error>
<XML_Error parser="XML4C">
  <errCode>7</errCode>
  <errDomain>http://apache.org/xml/messages/XMLValidity</errDomain>
  <errText>Element 'Notes' is not valid for content model: '(Name,Age)'
  </errText>
  <lineNum>1</lineNum>
  <colNum>292</colNum>
  <location>/Person</location>
  <schemaType>http://www.w3.org/2001/XMLSchema:anyType</schemaType>
  <tokenCount>2</tokenCount>
  <token1>Notes</token1>
  <token2>31</token2>
</XML_Error>
<DB2_Error>
  <sqlstate>2200M</sqlstate>
  <sqlcode>-16210</sqlcode>
  <errText>
    [IBM][CLI Driver][DB2/NT] SQL16210N  XML document contained a value "30x"
```

```
    that violates a facet constraint. Reason code = "13".  SQLSTATE=2200M
    </errText>
</DB2_Error>
</ErrorLog>

Parameter Name  : ERRORCOUNT
Parameter Value : 3
Return Status = 0
```

エラー・メッセージの拡張サポートのための ErrorLog XML スキーマ定義

ErrorLog XML スキーマ定義 (XSD) は、XML 文書の構文解析および妥当性検査エラーの結果として XSR_GET_PARSING_DIAGNOSTICS ストアード・プロシージャーによって生成される UTF-8 XML 文書を説明するものです。出力 XML 文書は *errorDialog* 引数に保管されます。

ErrorLogType

XML スキーマ定義のルート要素は ErrorLog であり、タイプは ErrorLogType になります。

XML スキーマ定義

```
<xs:complexType name="ErrorLogType">
  <xs:sequence>
    <xs:element name="XML_Error" type="XML_ErrorType" minOccurs="0"
          maxOccurs="unbounded"/>
    <xs:element name=name="XML_FatalError" type="XML_ErrorType" minOccurs="0"
          maxOccurs="unbounded"/>
    <xs:element name="DB2_Error" type="DB2_ErrorType"/>
  </xs:sequence>
</xs:complexType>
```

サブ要素

XML_Error
XML_FatalError

タイプ:

XML_ErrorType

使用上の注意:

XML_Error または XML_FatalError 要素には、XML パーサーによって生成されたエラー・メッセージが含まれます。 XML_Error および XML_FatalError 要素はどちらも同じ XML スキーマ・タイプです。XML_FatalError は、XML パーサーによる構文解析処理を異常終了させるエラーです。

```
 xs:complexType name="XML_ErrorType">
  <xs:sequence>
    <xs:element name="errCode"    type="xs:int"/>
    <xs:element name="errDomain"  type="xs:string"/>
    <xs:element name="errText"    type="xs:string"/>
    <xs:element name="lineNum"    type="xs:unsignedInt"/>
    <xs:element name="colNum"     type="xs:unsignedInt"/>
    <xs:element name="location"   type="xs:string"/>
    <xs:element name="schemaType" type="xs:string"/>
    <xs:element name="tokens">
      <xs:complexType>
        <xs:sequence minOccurs="0">
```

```
            <xs:element name="token" type="xs:string" minOccurs="0"
                 maxOccurs="unbounded"/>
          </xs:sequence>
          <xs:attribute name="count" type="xs:unsignedByte"
               use="required"/>
        </xs:complexType>
       </xs:element>
      </xs:sequence>
     <xs:attribute name="parser" type="xs:string" use="required"/>
    </xs:complexType>
```

XML_ErrorType 要素には、次のサブ要素が含まれます。

errCode
XML パーサーがエラー・コードを戻しました。

errDomain
XML パーサーがエラー・ドメインを戻しました。

errText
元の XML パーサー・エラー・メッセージ。

lineNum
エラーが発生した行番号。

colNum
エラーが発生した列番号。

location
Location は、エラーが発生する直前の XML 要素を指す XPath 式です。

schemaType
最後に構文解析された XML 要素の XML Schematype。

tokens 報告されるトークンの数を示す数値。

token Token は、DB2 エラー・メッセージを生成するために使用されるストリング値です。

属性

parser (必須)
parser 属性は、使用された基本の XML パーサーを指定します。

DB2_Error

タイプ:
DB2_ErrorType

使用上の注意:
DB2_Error 要素には、DB2 エラー・メッセージが含まれます。

```
<xs:complexType name="DB2_ErrorType">
  <xs:sequence>
    <xs:element name="sqlstate" type="xs:string"/>
    <xs:element name="sqlcode"  type="xs:int"/>
    <xs:element name="errText"  type="xs:string"/>
  </xs:sequence>
 <xs:attribute name="parser" type="xs:string" use="required"/>
</xs:complexType>
```

DB2_ErrorType 要素には、次のサブ要素が含まれます。

sqlstate
SQLSTATE

sqlcode
SQLCCODE

errText
DB2 エラー・メッセージ。

エラー・メッセージの拡張サポートのための XML スキーマ

ストアード・プロシージャー XSR_GET_PARSING_DIAGNOSTICS は、XML 文書の構文解析および妥当性検査に発生するエラーに関する詳細情報を生成します。この情報は、XML 文書として生成されます。ストアード・プロシージャーの妥当な XML 出力はスキーマによって定義されます。

次のリストは、ストアード・プロシージャー XSR_GET_PARSING_DIAGNOSTICS によって生成される XML 文書の ErrorLog XML スキーマを表しています。

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns="http://www.ibm.com/db2/XMLParser/Diagnosticsv10"
     xmlns:xs="http://www.w3.org/2001/XMLSchema"
     targetNamespace="http://www.ibm.com/db2/XMLParser/Diagnosticsv10"
     elementFormDefault="qualified" attributeFormDefault="unqualified">
  <xs:element name="ErrorLog">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="XML_Error" type="XML_ErrorType" minOccurs="0"
             maxOccurs="unbounded"/>
        <xs:element name="XML_FatalError" type="XML_ErrorType" minOccurs="0"/>
        <xs:element name="DB2_Error" type="DB2_ErrorType"/>
        <xs:any namespace="##any" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:complexType name="XML_ErrorType">
   <xs:attribute name="parser" type="xs:string" use="required"/>
    <xs:sequence>
      <xs:element name="errCode" type="xs:int"/>
      <xs:element name="errDomain" type="xs:string"/>
      <xs:element name="errText" type="xs:string"/>
      <xs:element name="lineNum" type="xs:unsignedInt"/>
      <xs:element name="colNum" type="xs:unsignedInt"/>
      <xs:element name="location" type="xs:string"/>
      <xs:element name="schemaType" type="xs:string"/>
      <xs:element name="tokens">
        <xs:complexType>
          <xs:sequence minOccurs="0">
            <xs:element name="token" type="xs:string" minOccurs="0"
                 maxOccurs="unbounded"/>
          </xs:sequence>
          <xs:attribute name="count" type="xs:unsignedByte" use="required"/>
        </xs:complexType>
      </xs:element>
      <xs:any namespace="##any"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="DB2_ErrorType">
   <xs:attribute name="parser" type="xs:string" use="required"/>
    <xs:sequence>
      <xs:element name="sqlstate" type="xs:string"/>
      <xs:element name="sqlcode" type="xs:int"/>
      <xs:element name="errText" type="xs:string"/>
```

```
      <xs:any namespace="##any"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

非 Unicode データベースでの XML の使用

バージョン 9.5 以降、XML データは Unicode コード・ページを使用しないデータベースで格納および検索できるようになりました。

内部的に、データベース・コード・ページには関係なく、XML データは常に DB2 データベース・サーバーによって Unicode 形式で管理されます。非 XML のリレーショナル・データは、データベース・コード・ページで管理されます。どちらかのデータ・タイプを他方のデータ・タイプにキャストするときや XML データ・タイプおよび SQL データ・タイプの両方が関係する比較をするときなど、SQL または XQuery ステートメントに XML データおよび SQL リレーショナル・データの両方が含まれる場合は、コード・ページ変換がしばしば必要となります。 XML データと XML データとの比較では、どちらのデータのセットも既に UTF-8 形式なので、コード・ページ変換は必要ありません。同様に、SQL データと SQL データとの比較では、どちらのデータのセットも既にデータベース・コード・ページなので、コード・ページ変換は必要ありません。

XML データおよび SQL データを含む操作では、データベースはすべてのデータ・タイプに対して同じエンコード方式を使用するので、コード・ページ変換の必要性は Unicode データベースでは除去されます。ただし、非 Unicode データベースでは、コード・ページ変換を含む操作の結果として破損やデータの消失が生じる可能性があります。変換中の XML データにデータベース・コード・ページの一部ではないコード・ポイントの文字が含まれる場合、文字置換が発生します。その結果、キャストまたは比較操作によって予期されない結果になることがあり、データベースから検索された XML データに不正な値が含まれることがあります。コード・ページ変換の問題を回避して保管された XML データの保全性を保証するためのさまざまな手段、および関係する操作について、次のセクションで説明します。

XML 文書の挿入およびコード・ページ変換

文字データ・タイプ (FOR BIT DATA タイプ以外の CHAR、VARCHAR、または CLOB のデータ・タイプ) の XML データが DB2 データベース・サーバーにホスト変数またはパラメーター・マーカーを介して挿入される場合には、データベース・コード・ページが要求を発行するクライアントまたはアプリケーションのコード・ページと異なる場合、コード・ページ変換が発生します。挿入された文字データがデータベース・コード・ページから Unicode (XML データが内部的に管理される形式) に変換される際に、2 度目の変換が発生します。

以下の表は、データベースとクライアントまたはアプリケーションから挿入された XML 文書ストリングとの間の、さまざまな可能なエンコード方式の組み合わせを示しています。クライアントは XML データを文字データ・タイプを介して挿入するので、XML 文書エンコードはクライアント・コード・ページと同じです。それぞれの組み合わせについて、XML 文書挿入中のコード・ページ変換の影響、およびその結果として生じる文字置換の可能性が説明されています。

表 2. データベースと挿入された XML 文書ストリングとの間のエンコード・シナリオ

シナリオ	XML 文書のエンコード方式	データベースのエンコード方式	コード・ページは一致しますか ?
1.	Unicode (UTF-8)	Unicode (UTF-8)	はい
2.	非 Unicode	Unicode (UTF-8)	いいえ
3.	非 Unicode	非 Unicode	はい
4.	Unicode (UTF-8)	非 Unicode	いいえ
5.	非 Unicode	非 Unicode	いいえ

1. シナリオ 1 で、XML 文書およびデータベースは Unicode エンコードを共有します。 XML 文書が挿入されるときに、文字変換は生じません。この方法で XML データを挿入することが、常に安全です。
2. シナリオ 2 で、非 Unicode の XML 文書は Unicode データベースに挿入されるために UTF-8 に変換されます。この処理では、文字置換は生じません。この方法で XML データを挿入することが、常に安全です。
3. シナリオ 3 で、XML 文書およびデータベースは同じ非 Unicode エンコードを共有します。この場合、XML 文書にはデータベース・コード・ページの一部であるコード・ポイントだけが含まれるので、コード・ページ変換中に文字置換は発生しません。この方法で XML データを挿入することが、常に安全です。
4. シナリオ 4 で、Unicode の XML 文書が非 Unicode のデータベースに挿入されます。文字データ・タイプの XML 文書がホスト変数またはパラメーター・マーカーを介して UTF-8 クライアントまたはアプリケーションから挿入される場合、コード・ページ変換が生じます。データベース・コード・ページ内に一致するコード・ポイントのない XML 文書内の文字は置換されます。
5. シナリオ 5 で、XML 文書がデータベース・サーバーに挿入されるとき、それら 2 つのエンコード方式が異なり、どちらも UTF-8 ではない場合について示されます。この場合、シナリオ 4 のようにして XML 文書が文字データ・タイプを使用して挿入されるのであれば、XML 文書にデータベース・コード・ページで無効な文字が含まれる場合に文字置換が発生します。

XML データを非 Unicode のデータベースに安全に挿入する

XML データの保全性を保証する上で最も安全な方法は、Unicode データベースを使用することです。しかし、それが不可能な場合、文字置換の発生を回避する他の方法もあります。以下のリストは、Unicode データベースが使用される場合と使用されない場合について、XML データを安全に挿入するためのさまざまな方法を示しています。

Unicode データベースを使用するか、またはデータベースとクライアントとが同じエンコード方式を使用することを確認します。

表 2 で例示されているように、以下の場合には、XML データに関するコード・ページ変換の問題は、常に回避されます。

- データベースが Unicode の場合
- Unicode であってもなくても、データベースとクライアントとが同じエンコード方式を共有する場合

文字データ・タイプと共にホスト変数またはパラメーター・マーカーを使用することを避けます。

Unicode データベースを使用できないとき、XML データのコード・ページ変換は、タイプ XML または他のバイナリー・データ・タイプのホスト変数またはパラメーター・マーカーを使用して XML データをバインドすることによっても避けられます。つまり、XML データに CHAR、VARCHAR、または CLOB 以外のデータ・タイプを指定すると、それをクライアントまたはアプリケーション・コード・ページから Unicode に直接渡すことが可能になり、データベース・コード・ページへの変換が回避されます。

ENABLE_XMLCHAR 構成パラメーターによって、文字データ・タイプを介して挿入することを許可するかどうかを制御できます。ENABLE_XMLCHAR を「NO」に設定すると、XML 文書の挿入中に文字データ・タイプの使用をブロックすることにより、文字置換が発生する可能性が回避され、格納された XML データの保全性が保証されます。 BLOB および FOR BIT DATA タイプは、コード・ページ変換に関して安全なので、引き続き許可されます。デフォルトで、ENABLE_XMLCHAR は YES に設定されているので、文字データ・タイプの挿入が可能です。

Unicode データベースが使用されるときはコード・ページ変換が問題とならないので、この場合 ENABLE_XMLCHAR 構成パラメーターは効果がありません。 ENABLE_XMLCHAR の設定に関係なく、XML 文書の挿入には文字データ・タイプを使用できます。

データベース・コード・ページに含まれない文字に対して文字エンティティー参照を使用します。

コード・ページ変換を避けることができず、XML データ・ストリームに文字データ・タイプを使用しなければならない場合、XML 文書内のすべての文字に対してデータベース・コード・ページ内に一致するコード・ポイントがあることを確認するのが最善です。ターゲット・データベース内に一致するコード・ポイントがない XML データ内の文字については、文字エンティティー参照を使用して文字の Unicode コード・ポイントを指定できます。文字エンティティー参照ではコード・ページ変換が常に回避されるので、正しい文字が XML データ内に保持されます。例えば、文字エンティティー参照 > および > は、それぞれ「より大」の記号 (">") を 16 進および 10 進で表したものです。

非 Unicode データベースでの XML データの照会

XML データをデータベースに挿入する際に、XML データの関係する照会の実行中のデータ保全性を保証する上で最も安全な方法は、Unicode データベースを使用することです。それが不可能な場合、すべての XML データがデータベース・コード・ページで表現可能であることを確認することにより、またはデータベース・コード・ページ内にない文字の文字エンティティー参照を使用することにより、文字置換を回避できます。

データベース・コード・ページで表現できない文字を含む XML 内容が照会に含まれている場合、次の 2 つのタイプの文字置換が発生することがあり、照会の結果が予期しないものとなる可能性があります。

デフォルトの置換文字による置換

コード・ページに対するデフォルトの置換文字は、 XML データ内にある

対応不可の文字の代わりに導入されます。例えば、中国語の文字が ASCII エンコード・データベース (ISO-8859-1) に渡される場合、クライアント上で表示された時点で元の文字は、ASCII コード・ポイント 0x1A に置換されます。それは、通常疑問符 ('?') として表示される制御文字です。 XML データがデータベース・コード・ページから Unicode に変換されるとき、置換文字は保存されます。

相当する文字のうち最も近いものによる置換 (「フォールディング」)
元の入力文字は、元の文字と必ずしも同じではなくても類似したターゲット・コード・ページの文字に置き換えられます。場合によっては、特殊な Unicode コード・ポイントの複数の文字がデータベース・コード・ページ内の単一のコード・ポイント (ターゲット・コード・ページ内で相当する文字のうち最も近いもの) にマップされることがあるので、データベースへの挿入後にそれらの値の間の区別はなくなります。このシナリオは、例 2 で例示されています。

例

以下の例は、UTF-8 エンコードのクライアントまたはアプリケーションを使用して非 Unicode データベース内の XML データを照会するときに、生じる可能性のあるコード・ページ変換の影響を例示しています。これらの例では、データベースがコード・ページ ISO8859-7 (ギリシャ語) を使用して作成されたと仮定しています。XQuery 式を使用して、表 T1 に保管された XML データを突き合わせます。保管された XML データは、Unicode のギリシャ語シグマ文字 (Σ_G) および Unicode の数学シグマ文字 (Σ_M) から構成されます。コード・ポイント 0xD3 は、ISO8859-7 データベース内のシグマ文字を識別します。

以下のコマンドを使用して、表 T1 が作成されて値が取り込まれます。

```
CREATE TABLE T1 (DOCID BIGINT NOT NULL, XMLCOL XML);
INSERT INTO T1 VALUES (1, XMLPARSE(
       document '<?xml version="1.0" encoding="utf-8" ?> <Specialchars>
 <sigma>Σ_G</sigma>
 <summation>Σ_M</summation>
 </Specialchars>'
       preserve whitespace));
```

例 1: 成功するコード・ページ変換 (文字がデータベース・コード・ページ内で表現可能)

```
XQUERY for $test in db2-fn:xmlcolumn("T1.XMLCOL")//*[. = "Σ_G"] return $test
```

この式は、目的の結果を生成します。

```
<sigma>Σ_G</sigma>
```

この場合、式 Σ_G はクライアント側でギリシャ語のシグマ文字に対する Unicode コード・ポイント (U+03A3) として始まり、ギリシャ語データベース・コード・ページ内のシグマ文字 (0xD3) に変換されてから、 XML 処理のための正しい Unicode 文字に戻されます。ギリシャ語のシグマ文字はデータベース・コード・ページ内で表現可能なので、この式から正しいマッチが得られます。この文字変換は、以下の表に示されています。

表 3. 文字データ変換 (例 1)

	クライアント (UTF-8)		データベース (ISO8859-7)		XML パーサー (UTF-8)
文字	U+03A3 (ギリシャ語シグマ)	→	0xD3 (ギリシャ語シグマ)	→	U+03A3 (ギリシャ語シグマ)

例 2: 成功しないコード・ページ変換 (文字がデータベース・コード・ページで表現できない)

```
XQUERY for $test in db2-fn:xmlcolumn("T1.XMLCOL")//*[. = "Σ_M"] return $test
```

この式は、目的の結果を生成しません。

```
<sigma>Σ_G</sigma>
```

この場合、式 Σ_M はクライアント側で数学記号シグマに対する Unicode コード・ポイント (U+2211) として始まり、ギリシャ語データベース・コード・ページ内のシグマ文字 (0xD3) に変換されてから、XML 比較の実行時に Σ_G 文字とマッチします。戻り式の場合、プロセスは例 1 と同じです。Unicode XML 文字 Σ_G は最初にギリシャ語データベース・コード・ページのシグマ文字 (Σ_A) に変換されて、その後クライアント UTF-8 コード・ページのギリシャ語シグマ文字 (Σ_G) に戻されます。この文字変換は、以下の表に示されています。

表 4. 文字データ変換 (例 2)

	クライアント (UTF-8)		データベース (ISO8859-7)		XML パーサー (UTF-8)
文字	U+2211 (数学シグマ)	→	0xD3 (ギリシャ語シグマ)	→	U+03A3 (ギリシャ語シグマ)

例 3: 文字エンティティー参照を使用してコード・ページ変換を迂回する

```
XQUERY for $test in db2-fn:xmlcolumn("T1.XMLCOL")//*[. = "&#2211;"]
   return $test
```

この式は、目的の結果を生成します。

```
<summation>Σ_M</summation>
```

この場合、式 Σ_M はクライアント側で数学記号シグマに対する Unicode コード・ポイント (U+2211) として始まり、それが文字参照 &#2211 としてエスケープされるので、XML パーサーに渡されるときに Unicode コード・ポイントは保持され、保管された XML 値 Σ_M に対する比較が成功します。文字変換の迂回は、以下の表に示されています。

表 5. 文字データ変換 (例 3)

	クライアント (UTF-8)		データベース (ISO8859-7)		XML パーサー (UTF-8)
文字	U+2211 (数学シグマに対する文字参照)	→	"&#2211" (数学シグマに対する文字参照)	→	U+2211 (数学シグマ)

例 4: 成功しないコード・ページ変換 (文字がデータベース・コード・ページで表現できない)

この例は例 1 と似ていますが、異なる点として ASCII エンコード・データベースが使用されて、コード・ページのデフォルト置換文字が XML 式に導入されます。

```
XQUERY for $test in db2-fn:xmlcolumn("T1.XMLCOL")//*[. = "Σ_G"] return $test
```

この照会は、表 T1 内の正しい値との突き合わせに失敗します。この場合、Unicode 文字 U+2211 (ギリシャ語シグマ) は ASCII コード・ページ内に一致するコード・ポイントがないので、デフォルトの置換文字が導入されます。この場合、それは疑問符 ('?') です。この文字変換は、以下の表に示されています。

表 6. 文字データ変換 (例 4)

	クライアント (UTF-8)		データベース (ISO8859-1)		XML パーサー (UTF-8)
文字	U+2211 (数学シグマ)	→	0x003F ('?')	→	0x003F ('?')

第 5 章 XML データの照会

データベース内に保管されている XML データの照会または検索は、主に 2 つの照会言語を用いて行えます。それぞれの言語を個別に使用するか、2 つを組み合わせて使用することで行えます。

以下のオプションが使用できます。

- XQuery 式のみ
- SQL ステートメントを呼び出す XQuery 式
- SQL ステートメントのみ
- XQuery 式を実行する SQL ステートメント

これらのさまざまな方法によって、XML および他のリレーショナル・データを SQL または XQuery コンテキストを用いて照会または検索できます。

これらの方法を使用して、XML 文書の部分または全体を照会および取り出しできます。照会は XML 文書の断片または全体を戻すことができ、述部を使用して照会から戻される結果を制限することができます。 XML データに対する照会は XML シーケンスを戻すので、照会の結果を XML データの作成に使用することもできます。

XQuery の概要

XQuery は、XML データを照会および変更するための特定の要件を満たすように、World Wide Web Consortium (W3C) によって設計された機能プログラミング言語です。

予測可能で、規則的な構造を持つリレーショナル・データとは異なり、XML データは非常に柔軟性があります。XML データは多くの場合、予測不能であり、散在しており、自己記述タイプです。

XML データの構造は予測不能であるため、 XML データに対して実行する必要のある照会は、多くの場合、典型的なリレーショナル照会とは異なります。XQuery 言語は、このような操作を実行するために必要な柔軟性を提供します。例えば、以下のような操作を実行するには、XQuery 言語を使用しなければならない場合があります。

- 階層のどのレベルにあるかが不明なオブジェクトの XML データを検索する。
- データに対して構造変換を実行する (例えば、階層を逆転させたい場合など)。
- タイプが混合した結果を戻す。
- 既存の XML データを更新する。

XQuery 照会のコンポーネント

XQuery では、式が照会の主要ビルディング・ブロックとなります。式はネスト可能で、照会の本体を形成します。照会には、その本体の前にプロローグも含めることができます。*プロローグ* には、照会の処理環境を定義する一連の宣言が含まれま

す。*照会本体* は、照会の結果を定義する式で構成されます。この式は、演算子またはキーワードを使用して結合される複数の XQuery 式で構成できます。

図 4は、典型的な照会の構造を示しています。この例では、プロローグに以下の 2 つの宣言が含まれます。照会を処理するために使用する XQuery 構文のバージョンを指定するバージョン宣言、および接頭部のない要素名およびタイプ名に使用する名前空間 URI を指定するデフォルト名前空間宣言。照会本体には、 `price_list` 要素を構成する式が含まれます。 `price_list` 要素の内容は、価格によって降順にソートされる `product` 要素のリストです。

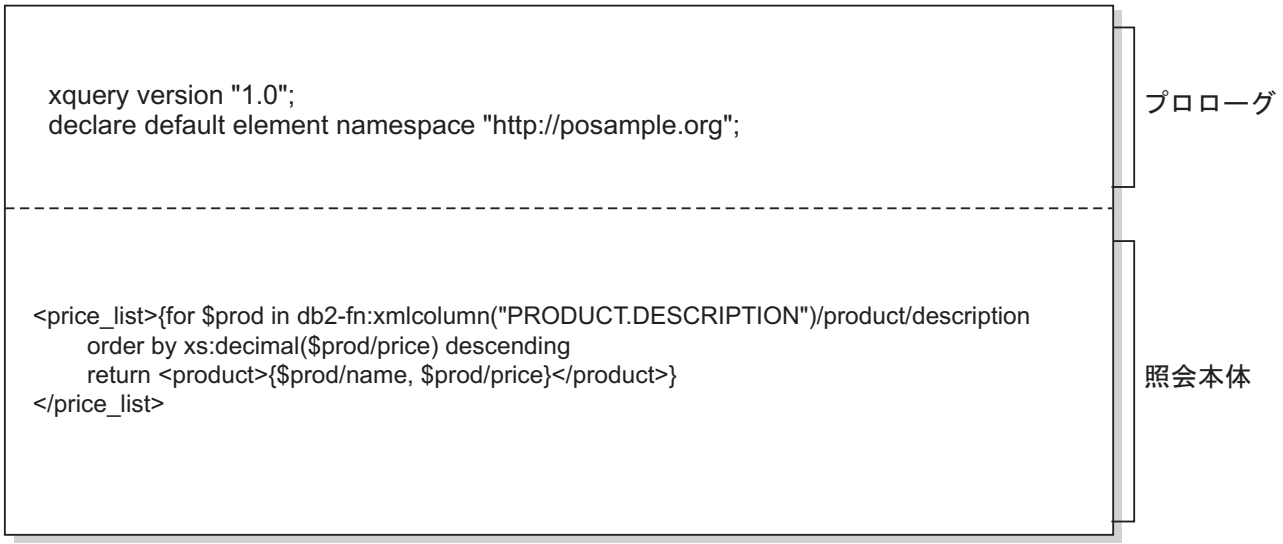


図 4. XQuery における典型的な照会の構造

XQuery 関数を使用した DB2 データの検索

XQuery では、照会において以下の関数のいずれかを呼び出して、DB2 データベースから入力 XML データを取得できます。db2-fn:sqlquery および db2-fn:xmlcolumn。

関数 db2-fn:xmlcolumn は、XML 列全体を検索します。一方 db2-fn:sqlquery は、SQL 全選択に基づく XML 値を検索します。

db2-fn:xmlcolumn

db2-fn:xmlcolumn 関数は、表またはビュー内の XML 列を識別するストリング・リテラル引数を使用し、その列にある XML 値のシーケンスを戻します。この関数の引数は、大/小文字を区別します。ストリング・リテラル引数は、修飾された XML タイプの列名にする必要があります。この関数により、検索条件を適用しなくても、XML データの列全体を抽出することができます。

以下の例において、照会は db2-fn:xmlcolumn 関数を使用して、BUSINESS.ORDERS 表の PURCHASE_ORDER 列内のすべての購入注文を取得します。続いてこの照会は、この入力データを操作して、これらの購入

注文の配送先住所から市区町村を抽出します。照会の結果は、注文商品が配送されるすべての市区町村のリストです。

```
db2-fn:xmlcolumn('BUSINESS.ORDERS.PURCHASE_ORDER')/shipping_address/city
```

db2-fn:sqlquery

db2-fn:sqlquery 関数は、fullselect を表すストリング引数を使用し、fullselect によって戻される XML 値の連結である XML シーケンスを戻します。fullselect では、単一列の結果セットを指定する必要があり、列はデータ・タイプが XML である必要があります。fullselect を指定することにより、SQL の機能を使用して XML データを XQuery に提供できます。関数は、パラメーターを使用した SQL ステートメントへの値の受け渡しをサポートします。

以下の例では、BUSINESS.ORDERS という表に PURCHASE_ORDER という XML 列が含まれています。この例の照会は、db2-fn:sqlquery 関数を使用して SQL を呼び出して、配送日付が 2005 年 6 月 15 日であるすべての購入注文を取得します。続いて、この照会は、この入力データを操作して、これらの購入注文の配送先住所から市区町村を抽出します。照会の結果は、6 月 15 日に注文商品が配送されるすべての市区町村のリストです。

```
db2-fn:sqlquery("
SELECT purchase_order FROM business.orders
WHERE ship_date = '2005-06-15' ")/shipping_address/city
```

重要: db2-fn:sqlquery 関数または db2-fn:xmlcolumn 関数によって戻される XML シーケンスには、アトミック値およびノードを含む任意の XML 値を含めることができます。これらの関数は、必ずしも整形式の文書のシーケンスを戻すとは限りません。例えば、関数が、XML データ・タイプのインスタンスとして、単一のアトミック値 (36 など) を戻す場合があります。

SQL および XQuery には、名前の大/小文字の区別に関して異なる規則があります。db2-fn:sqlquery 関数および db2-fn:xmlcolumn 関数の使用時には、これらの差異に注意する必要があります。

SQL は大/小文字を区別する言語ではない

デフォルトでは、SQL ステートメントで使用されるすべての通常 ID は、自動的に大文字に変換されます。このため、SQL の表および列の名前は、上記の例における BUSINESS.ORDERS および PURCHASE_ORDER のように、通例は大文字の名前です。SQL ステートメントでは、 business.orders および purchase_order のように小文字の名前を使用してこれらの列を参照できますが、これらは SQL ステートメントの処理中に自動的に大文字に変換されます。(名前を二重引用符で囲むことにより、SQL で*区切り ID* と呼ばれる大/小文字を区別する名前を作成することもできます。)

XQuery は大/小文字を区別する言語です

XQuery は、小文字の名前を大文字に変換しません。この違いにより、XQuery および SQL の同時使用時に混乱が生じることがあります。db2-fn:sqlquery に渡されるストリングは、SQL 照会として解釈され、SQL パーサーによって構文解析されて、これによりすべての名前が大文字に変換されます。このため、 db2-fn:sqlquery の例では、表名 business.orders、および列名 purchase_order および ship_date を、大文字または小文字のいずれでも表示できます。ただし、db2-fn:xmlcolumn のオペランドは、SQL 照会で

はありません。オペランドは、列名を表す、大/小文字を区別する XQuery ストリング・リテラルです。列の実際の名前は BUSINESS.ORDERS.PURCHASE_ORDER であるため、db2-fn:xmlcolumn のオペランドに、この名前を大文字で指定する必要があります。

SQL を使用して XML データを照会する方法の概要

XML データは、SQL 全選択または SQL/XML 照会関数の XMLQUERY および XMLTABLE を使用して照会できます。 XML データに対する SQL 照会の中で、XMLEXISTS 述部を使用することもできます。

XQuery ではなく SQL のみを使用して XML データを照会する場合は、全選択を発行すると、列レベルのみの照会が可能となります。このため、照会によって戻すことができるのは XML 文書の全体だけとなります。SQL だけを使用して文書の断片を戻すことはできません。

XML 文書内で照会を行うには、XQuery を使用する必要があります。 XQuery は、以下の SQL/XML 関数または述部のいずれかを使用して SQL から呼び出すことができます。

XMLQUERY
XQuery 式の結果を XML シーケンスとして戻す SQL スカラー関数。

XMLTABLE
XQuery 式の結果を表として戻す SQL 表関数。

XMLEXISTS
XQuery 式が空でないシーケンスを戻すかどうかを判別する SQL 述部。

XQuery と SQL の比較

DB2 データベースでは、SQL、XQuery、または SQL と XQuery の組み合わせを使用して、表の列への整形式 XML データの保管、およびデータベースからの XML データの検索をサポートします。両方の言語が基本照会言語としてサポートされています。また、どちらの言語も他の言語を呼び出すための関数を提供します。

XQuery
XQuery を直接呼び出す照会は、キーワード XQUERY で始まります。このキーワードは XQuery が使用されていることを示し、 XQuery 言語に適用される、大/小文字を区別する規則を、 DB2 サーバーが使用する必要があることを示します。エラー処理は、XQuery 式を処理するために使用されるインターフェースに基づきます。XQuery エラーは、SQL エラーが報告されるのと同じ方法で、SQLCODE および SQLSTATE を使用して報告されます。XQuery 式の処理で警告は戻されません。XQuery では、 DB2 表とビューから XML データを抽出する関数を呼び出すことでデータを取得します。XQuery は、SQL 照会から呼び出すこともできます。この場合、SQL 照会では、バインド変数の形式で XML データを XQuery に渡すことができます。XQuery は、XML データの処理や、要素および属性などの新規 XML オブジェクトの構成のために、さまざまな式をサポートします。 XQuery のプログラミング・インターフェースでは、SQL に類似した機能を提供して照会を準備し、照会結果を取得します。

SQL SQL により、XML データ・タイプの値を定義し、インスタンス化することができます。整形式 XML 文書を含むストリングは、XML 値に解析し、状況に応じて XML スキーマに対して妥当性検査し、表に挿入または表で更新することができます。または、他のリレーショナル・データを XML 値に変換する SQL コンストラクター関数を使用することで、XML 値を構成することもできます。XQuery を使用することで、XML データを照会することも、XML データをリレーショナル表に変換して SQL 照会で使用することもできます。データは、XML 値をストリング・データにシリアライズ化するだけでなく、SQL データ・タイプと XML データ・タイプの間でキャストすることもできます。

SQL/XML では、以下の関数と述部を指定して、SQL から XQuery を呼び出します。

XMLQUERY
XMLQUERY は、引数として XQuery 式を使用し、XML シーケンスを戻すスカラー関数です。この関数には、 SQL 値を XQuery 変数として XQuery 式に渡すために使用できるオプション・パラメーターが含まれています。XMLQUERY によって戻される XML 値は、SQL 照会のコンテキスト内でさらに処理することができます。

XMLTABLE
XMLTABLE は、XQuery 式を使用して XML データから SQL の表を生成する表関数で、この表は SQL でさらに処理することができます。

XMLEXISTS
XMLEXISTS は、XQuery 式が 1 つ以上の項目のシーケンスを戻すか (空のシーケンスではないか) どうかを判別する SQL 述部です。

XML データを照会する方式の比較

XML データは XQuery、SQL、またはこれらの組み合わせを使用して複数の方式で照会できるので、状況に応じて異なる方式を選択できます。以下のセクションでは、特定の照会方式にとって有利な状況について説明します。

XQuery のみ

以下の場合は、XQuery のみでの照会が適切な選択となります。

- アプリケーションが XML データだけにアクセスし、非 XML リレーショナル・データを照会する必要がない場合。
- 以前に XQuery で記述した照会を DB2 Database for Linux, UNIX, and Windows にマイグレーションする場合。
- 照会結果を、XML 文書を作成するための値として使用する場合。
- 照会の作成者が SQL よりも XQuery をよく知っている場合。

SQL を呼び出す XQuery

SQL を呼び出す XQuery を使用する照会は、(XQuery だけを使用する方式についての直前のセクションで示されたシナリオに加えて) 以下の場合に適切な選択となります。

- 照会に XML データとリレーショナル・データが関係する場合。SQL 述部と、リレーショナル列に定義された索引を照会の中で活用できます。
- XQuery 式を以下の結果に適用したい場合。
  - UDF 呼び出し (XQuery から直接呼び出すことができないため)。
  - SQL/XML 発行関数を使用してリレーショナル・データから作成される XML 値。
  - DB2 Net Search Extender を使用する照会。この製品は XML 文書の全文検索機能を提供しますが、SQL と共に使用する必要があります。

SQL のみ

XQuery を使用しないで SQL だけを使用して XML データを検索するときには、XML 列レベルでしか照会を行うことができません。このため、照会によって戻すことができるのは XML 文書の全体だけとなります。次のような場合、この使用法が適しています。

- XML 文書全体を検索する場合。
- 保管されている文書内の値に基づく照会を行う必要がない場合、または照会の述部が表の他の非 XML 列である場合。

XQuery 式を実行する SQL/XML 関数

SQL/XML 関数の XMLQUERY と XMLTABLE、および XMLEXISTS 述部は、XQuery 式を SQL コンテキスト内から実行できるようにします。 XQuery を SQL 内から実行する方法は、以下の場合に適切な選択となります。

- 既存の SQL アプリケーションから XML 文書内での照会を行えるようにする必要がある場合。 XML 文書内を照会するには、XQuery 式を実行する必要があり、これは SQL/XML を使用して行うことができます。
- XML データを照会するアプリケーションがパラメーター・マーカーを XQuery 式に渡す必要がある場合。 (パラメーター・マーカーは、最初に XMLQUERY または XMLTABLE 内の XQuery 変数にバインドされます。)
- 照会の作成者が XQuery よりも SQL をよく知っている場合。
- リレーショナルおよび XML データの両方を単一の照会で戻す必要がある場合。
- XML とリレーショナル・データとを結合する必要がある場合。
- XML データをグループ化または集約する場合。副選択の GROUP BY または ORDER BY 節を XML データに適用できます (例えば、XMLTABLE 関数によって XML データを検索して表形式に集めた後)。

XML 名前空間の指定

XML 文書で、XML 名前空間はオプションであり、XML 文書内のノード名の接頭部として使用されます。名前空間を使用する XML 文書内のノードにアクセスするには、 XQuery 式でも同じ名前空間をノード名の一部として指定する必要があります。

デフォルトの XML 名前空間は文書に対して指定することができ、 XML 名前空間は文書内の特定の要素に対して指定できます。

名前空間宣言はセミコロン (;) によって終了することに注意してください。つまり、コマンド行プロセッサーを db2 -t で起動するなど、セミコロンを含む SQL ステートメントおよび XQuery 式を使用して作業する場合、セミコロンをステートメント終了文字として使用できません。名前空間宣言を含むステートメントが誤って解釈されないようにするための -td オプションを使用することにより、セミコロン以外の終了文字を指定できます。チュートリアル内の例では、波形記号 (~) を終了文字として使用しますが (-td~)、 % も一般的に使用されます (-td%)。

例えば、pureXML のチュートリアルでは、 XML 文書のデフォルト要素名前空間を指定する XML 文書を使用します。以下の XML は、チュートリアルで使用される XML 文書の 1 つです。

```
<customerinfo xmlns="http://posample.org" Cid="1002">
  <name>Jim Noodle</name>
  <addr country="Canada">
    <street>25 EastCreek</street>
    <city>Markham</city>
    <prov-state>Ontario</prov-state>
    <pcode-zip>N9C 3T6</pcode-zip>
  </addr>
  <phone type="work">905-555-7258</phone>
</customerinfo>
```

XML 文書のルート・ノードは、文書のデフォルト要素名前空間を Universal Resource Identifier (URI) http://posample.org にバインドします。

```
<customerinfo xmlns="http://posample.org" Cid="1002">
```

チュートリアルで実行できる XQuery 式も、 **declare default element namespace** プロローグを含むことにより、URI をデフォルト要素名前空間としてバインドします。例えば、以下の SELECT ステートメント内の XQuery 式はデフォルト要素名前空間を宣言します。 SELECT ステートメントをチュートリアルで作成された CUSTOMER 表に対して実行すると、1 つの Customer ID が戻されます。

```
SELECT cid FROM customer
   WHERE XMLEXISTS('declare default element namespace "http://posample.org";
   $i/customerinfo/addr/city[ . = "Markham"]' passing INFO as "i")
```

同じ URI を XML 文書内でデフォルト要素名前空間として使用することにより、式は式内のノード名を正しい名前空間接頭部で修飾します。デフォルト要素名前空間宣言がないか、または異なる URI がデフォルト要素名前空間としてバインドされている場合、式はノード名を正しい名前空間で修飾しないので、データは戻されません。例えば、以下の SELECT ステートメントは直前のステートメントと似ていますが、デフォルト名前空間宣言がありません。このステートメントをチュートリアルで作成した CUSTOMER 表に対して実行すると、データは戻されません。

```
SELECT cid FROM customer
WHERE XMLEXISTS('$i/customerinfo/addr/city[ . = "Markham"]'
   passing INFO as "i")
```

名前空間接頭部をノード名と共に使用する

ノード名を名前空間で修飾するには、各ノード名に名前空間接頭部を追加できます。接頭部とノード名はコロンで分離します。ノード po:addr では、名前空間接頭部 po がローカル・ノード名 addr から分離されています。名前空間接頭部をノード名で修飾する場合、接頭部が URI にバインドされていることを確認する必要があります。例えば、以下の SELECT ステートメント内の XQuery 式は、名前空間 po

を宣言して、名前空間接頭部 po を URI http://posample.org にバインドします。このチュートリアルで作成した CUSTOMER 表に対して以下のステートメントを実行すると、1 つの結果が戻されます。

```
SELECT cid FROM customer
WHERE XMLEXISTS('
    declare namespace po = "http://posample.org";
   $i/po:customerinfo/po:addr/po:city[ . = "Markham"]' passing INFO as "i")
```

名前空間接頭部 po は、任意の接頭部とすることができます。大切なのは、接頭部にバインドされる URI です。例えば、以下の SELECT ステートメントの XQuery 式は名前空間接頭部 mytest を使用しますが、直前のステートメントの式と同等です。

```
SELECT cid FROM customer
WHERE XMLEXISTS('declare namespace mytest = "http://posample.org";
   $i/mytest:customerinfo/mytest:addr/mytest:city[ . = "Markham"]'
   passing INFO as "i")
```

ワイルドカードを名前空間接頭部として使用する

XML データで使用されるすべての名前空間と一致するように、ワイルドカード文字を XQuery 式で使用できます。以下の SELECT ステートメント内の XQuery 式は、すべての名前空間接頭部と一致するようにワイルドカード文字を使用しています。

```
SELECT cid FROM customer
WHERE XMLEXISTS('$i/*:customerinfo/*:addr/*:city[ . = "Markham"]'
   passing INFO as "i")
```

このチュートリアルで作成した CUSTOMER 表に対して SELECT ステートメントを実行すると、1 つの Customer ID が戻されます。

例: 要素の名前空間接頭部の変更

以下の例では、要素の QName に割り当てられた名前空間バインディングを追加または除去する方法を示します。

要素の QName は、オプションの名前空間接頭部およびローカル名です。名前空間接頭部とローカル名は、コロンで区切ります。名前空間接頭部を指定する場合、これは URI (Universal Resource Identifier) にバインドされ、URI の短縮形が提供されます。拡張 QName には、名前空間 URI およびローカル名が含まれます。

fn:QName、fn:local-name、および fn:namespace-uri などの関数を使用して、属性およびノード名を名前変更し、ノードの名前空間接頭部と関連付けられた URI を検索できます。

XQuery 名前空間宣言は、セミコロン (;) によって終了します。セミコロンをステートメント終了文字として使用することはできません。この例では、波形記号 (~) を終了文字として使用しますが、これは pureXML チュートリアルで使用されるのと同じ終了文字です。

要素への名前空間接頭部の追加

この例では、XML 文書を表から使用します。以下のステートメントは、表を作成し、XML 文書をその表に挿入します。

```
CREATE TABLE XMLTEST (ID BIGINT NOT NULL PRIMARY KEY, XMLDOC XML ) ~
INSERT INTO XMLTEST Values (4,
  '<depts>
    <dept id="A07">
        <emp id="31201" >
           <location region="31" />
        </emp>
        <emp type="new" id = "23322" >
          <moved:location xmlns:moved="http://oldcompany.com" region="43" />
        </emp>
    </dept>
    </depts> ') ~
```

以下の SELECT ステートメント内の XQuery 式は、名前空間接頭部を要素に追加します。 XQuery 式は fn:QName を使用して、名前空間バインディングを持つ QName を作成します。

XQuery **let** 節は、名前 emp および名前空間 http://example.com/new を持つ空の要素を作成します。 **transform** 式では、**rename** 式は XPath 式 $newdept/depts/dept/emp[@type="new"] で識別される要素の名前を変更します。要素には emp という名前が付けられますが、名前空間接頭部はありません。

```
SELECT XMLQUERY ( '
  let $newemp := fn:QName( "http://mycompany.com", "new:emp")
  return
  transform
    copy $newdept := $doc
    modify
    do rename $newdept/depts/dept/emp[@type="new"] as $newemp
  return
  $newdept  ' passing XMLDOC as "doc" )
from XMLTEST where ID = 4 ~
```

XQuery 式は、以下の XML データを、ノード new:emp の一部として指定された名前空間バインディングとともに戻します。変更された文書は、名前空間にバインドされる名前空間接頭部を含む有効な XML です。名前空間宣言は、新規に名前変更された要素に加えられます。

```
<depts>
 <dept id="A07">
  <emp id="31201">
   <location region="31" />
  </emp>
  <new:emp xmlns:new="http://mycompany.com" type="new" id="23322">
   <moved:location xmlns:moved="http://oldcompany.com" region="43" />
  </new:emp>
 </dept>
</depts>
```

要素からの名前空間接頭部の除去

名前空間バインディングを使用しない QName にノードを名前変更することによって、ノード名から名前空間接頭部を除去することができます。 SELECT ステートメント内の以下の XQuery 式は、for 文節および fn:namespace-uri 関数を使用して、各 emp ノード内の要素ノードの URI を識別します。URI が http://oldcompany.com である場合、名前変更式は fn:QName および fn:local-name 関数を使用して、要素ノードから名前空間接頭部を除去します。

```
SELECT XMLQUERY ( '
  transform
    copy $newdept := $x
```

```
    modify
     for $testemp in $newdept/depts/dept/*:emp/*
     return
     if ( fn:namespace-uri( $testemp ) eq "http://oldcompany.com")
     then
       do rename $testemp as fn:QName( "", fn:local-name($testemp) )
     else()
  return
  $newdept
 ' passing XMLDOC as "x" )
from XMLTEST where ID = 4 ~
```

XQuery 式は次の XML データを戻します。ノード名 new:location は location に置き換えられます。名前空間バインディングは、ノードから除去されません。

```
<depts>
  <dept id="A07">
    <emp id="31201">
      <location region="31" />
    </emp>
    <emp type="new" id="23322">
      <location xmlns:moved="http://oldcompany.com" region="43" />
    </emp>
  </dept>
</depts>
```

XMLQUERY 関数の概要

XMLQUERY は、SQL のコンテキストの中から XQuery 式を実行できるようにする SQL スカラー関数です。 XMLQUERY で指定された XQuery 式に、変数を渡すことができます。 XMLQUERY は、XML シーケンスである XML 値を戻します。このシーケンスは空であることもあり、1 つ以上の項目を含むこともあります。

XQuery 式を SQL コンテキスト内から実行して、以下を行うことができます。

- XML 文書の全体に対してではなく、保管された XML 文書の一部に対して操作を行う (XML 文書内を照会できるのは XQuery だけであり、SQL 単体では文書全体のレベルの照会を行います)
- XML データが SQL 照会に関与できるようにする
- リレーショナルおよび XML データの両方に対して操作を行う
- 戻された XML 値に対してさらに SQL 処理を適用する (例えば、副選択の ORDER BY 節による結果の順序付け)

詳しくは、照会方式の比較に関する資料を参照してください。

XQuery は大文字と小文字を区別するので、XMLQUERY に指定される XQuery 式と変数は注意深く指定しなければなりません。

SQL 式を渡すための全機能が必要ない場合には、**passing** 節で名前を明示的に指定せずに列名を渡すことができる、より単純な構文を使用することもできます。XMLEXISTS、XMLQUERY、および XMLTABLE を使用した列名の簡単な引き渡しを参照してください。

XMLQUERY によって戻される、空ではないシーケンス

XMLQUERY 関数内で指定されている XQuery 式が空ではないシーケンスを戻す場合、その関数も空ではないシーケンスを戻します。

例えば、CUSTOMER 表の XML 列 INFO に保管されている以下の 2 つの XML 文書を考えてみてください。

```
<customerinfo Cid="1002">
  <name>Jim Noodle</name>
  <addr country="Canada">
    <street>25 EastCreek</street>
    <city>Markham</city>
    <prov-state>Ontario</prov-state>
    <pcode-zip>N9C 3T6</pcode-zip>
  </addr>
  <phone type="work">905-555-7258</phone>
</customerinfo>

<customerinfo Cid="1003">
  <name>Robert Shoemaker</name>
    <addr country="Canada">
    <street>1596 Baseline</street>
    <city>Aurora</city>
    <prov-state>Ontario</prov-state>
    <pcode-zip>N8X 7F8</pcode-zip>
  </addr>
  <phone type="work">905-555-7258</phone>
  <phone type="home">416-555-2937</phone>
  <phone type="cell">905-555-8743</phone>
  <phone type="cottage">613-555-3278</phone>
</customerinfo>
```

以下の照会を発行する場合について考えます。

```
SELECT XMLQUERY ('$d/customerinfo/phone' passing INFO as "d")
FROM CUSTOMER
```

結果セットには、以下のような 2 つの行が含まれます (この出力は見やすくするために整形されています):

```
1
-----------

   <phone type="work">905-555-7258</phone>

   <phone type="work">905-555-7258</phone><phone type="home">416-555-2937</
   phone><phone type="cell">905-555-8743</phone><phone type="cottage">613-
   555-3278</phone>

 2 record(s) selected.
```

最初の行には 1 つの <phone> 要素のシーケンスが含まれるのに対して、2 番目の行には 4 つの <phone> 要素のシーケンスがあることに注意してください。この結果が生じるのは、2 番目の XML 文書に 4 つの <phone> 要素が含まれていて、XMLQUERY は XQuery 式を満たすすべての要素のシーケンスを戻すためです。 (2 番目の行にある結果は整形式の文書ではないことに注意してください。この結果を受け取るアプリケーションがこの動作に正しく対処できることを確認してください。)

直前の例は、XMLQUERY の一般的な使用方法を示しています。つまり、一度に 1 つの XML 文書に適用されて、結果となる表の各行が 1 つの文書からの結果を表すというものです。しかし、XMLQUERY は一度に複数の文書に適用することも可能であり、単一のシーケンスに複数の文書が含まれている場合も同様です。この場合、XMLQUERY をシーケンス内のすべての文書に適用した結果は、単一の行で戻されます。

例えば、上記のものと同じ文書が CUSTOMER 表の INFO 列に保管されていると仮定します。次の照会の db2-fn:xmlcolumn 関数は、INFO 列内の 2 つの XML 文書を含む 1 つのシーケンスを戻します。

```
VALUES
 (XMLQUERY
   ('db2-fn:xmlcolumn("CUSTOMER.INFO")/customerinfo/phone'))
```

その後、XML 文書のこの単一のシーケンスに XMLQUERY が適用されます。結果セットには以下のように 1 つの行だけが含まれます。

```
1
-----------

   <phone type="work">905-555-7258</phone><phone type="work">905-555-7258</
   phone><phone type="home">416-555-2937</phone><phone type="cell">905-555-
   8743</phone><phone type="cottage">613-555-3278</phone>

 1 record(s) selected.
```

INFO 列の XML 文書のすべての <phone> 要素は、XMLQUERY が単一の値 (db2-fn:xmlcolumn から戻される XML 文書のシーケンス) に対して実行されるので、単一の行で戻されます。

XMLQUERY によって戻される空のシーケンス

XMLQUERY 関数内で指定された XQuery 式が空のシーケンスを戻す場合、その関数も空のシーケンスを戻します。

例えば、次の照会で XMLQUERY は、CUSTOMER 表の INFO 列で <city> 要素の値が "Aurora" でない行ごとに空のシーケンスを戻します。

```
SELECT Cid, XMLQUERY ('$d//addr[city="Aurora"]' passing INFO as "d") AS ADDRESS
FROM CUSTOMER
```

CUSTOMER 表の行が 3 つあり、<city> 要素の値が "Aurora" なのは 1 つの XML 文書だけであると仮定します。直前の SELECT ステートメントの結果として次の表が生じます (出力は見やすくするために整形しています)。

表 7. 結果表

CID	ADDRESS
1001	
1002	
1003	<addr country="Canada"><street>1596 Baseline</street><city>Aurora</city><prov-state>Ontario</prov-state><pcode-zip>N8X-7F8</pcode-zip></addr>

<city> 要素の値が "Aurora" ではない行に対して、NULL 値ではなく、長さが 0 のシリアライズされた XML の空のシーケンスが戻されることに注意してください。しかし、3 行目については XQuery 式を満たすので <addr> 要素が戻されます。 3 行目には、空ではないシーケンスが戻されます。

XMLEXISTS などの述部を SELECT 節ではなく WHERE 節に適用して、空のシーケンスを含む行を戻さないようにできます。例えば、直前の照会はフィルター述部を XMLQUERY 関数から WHERE 節に移動して、次のように書き直すことができます。

```
SELECT Cid, XMLQUERY ('$d/customerinfo/addr' passing c.INFO as "d")
FROM Customer as c
WHERE XMLEXISTS ('$d//addr[city="Aurora"]' passing c.INFO as "d")
```

この照会から、次のような表が結果として生じます。

表 8. 結果表

CID	ADDRESS
1003	<addr country="Canada"><street>1596 Baseline</street><city>Aurora</city><prov-state>Ontario</prov-state><pcode-zip>N8X-7F8</pcode-zip></addr>

一般に XMLQUERY は、選択された文書の断片を戻すために SELECT 節の中で使用されます。 XMLQUERY の XQuery 式に指定された述部は、結果セットから行をフィルター操作するのではなく、どの断片を戻すかを決めるためだけに使用されます。結果セットから実際に行を除去するには、WHERE 節内の述部を適用する必要があります。 XMLEXISTS 述部は、保管されている XML 文書内の値に応じた述部を適用することができます。

XMLQUERY の結果を非 XML タイプにキャストする

XMLQUERY 関数の結果を SQL コンテキストに戻して、さらに処理 (比較や順序付け操作など) を加える場合、戻される XML 値を互換性のある SQL タイプにキャストする必要があります。 XMLCAST の指定により、XML と非 XML 値との間でキャストが可能です。

注:

1. XMLQUERY の結果を SQL データ・タイプにキャストできるのは、XMLQUERY に指定された XQuery 式が、原子化された 1 つの項目を含むシーケンスを戻す場合だけです。
2. 非 UTF-8 データベースでは、XMLQUERY の結果を SQL データ・タイプにキャストすると、戻り値が内部 UTF-8 エンコードからデータベースのコード・ページに変換される際に、コード・ページ変換が生じます。データベースのコード・ページの一部でないコード・ポイントを持つ戻り値は、代替文字に置き換えられます。代替文字により、XML 値と非 XML 値の比較で予期しない動作が発生することがあるので、保管されている XML データに、データベースのコード・ページに含まれているコード・ポイントのみが入っていることを注意深く確認する必要があります。

例: 照会内の XML 値と非 XML 値との比較

次の照会で、XMLQUERY によって戻されるシーケンスは XML タイプから文字タイプにキャストされて、PRODUCT 表の NAME 列と比較できるようになります。(XMLQUERY の結果として生じる XML 値がシリアライズされたストリングではない場合、XMLCAST 操作は失敗することがあります。)

```
SELECT R.Pid
FROM PURCHASEORDER P, PRODUCT R
WHERE R.NAME =
        XMLCAST( XMLQUERY ('$d/PurchaseOrder/item/name'
                           PASSING P.PORDER AS "d") AS VARCHAR(128))
```

例: XMLQUERY 結果による順序付け

次の照会で、製品 ID は、XML 文書に保管されている製品記述の <name> 要素の値によってソートされた順序で戻されます。 SQL は XML 値でソートすることができないので、SQL が順序付けできる値にシーケンスをキャストする必要があります (この例では文字)。

```
SELECT Pid
FROM PRODUCT
ORDER BY XMLCAST(XMLQUERY ('$d/product/description/name'
                           PASSING DESCRIPTION AS "d") AS VARCHAR(128))
```

データ・タイプ間のキャスト

特定のデータ・タイプの値を別のデータ・タイプへキャストする必要や、データ・タイプは同じでも長さ、精度、または位取りの異なるデータ・タイプへキャスト する必要が生じることがよくあります。

データ・タイプのプロモーションは、あるデータ・タイプから別のデータ・タイプへのプロモーションにおいて、値を新しいデータ・タイプへキャストすることが必要になる 1 つの例です。別のデータ・タイプへキャストできるデータ・タイプは、ソース・データ・タイプから宛先データ・タイプへキャスト*可能* であるといいます。

あるデータ・タイプから別のデータ・タイプへのキャストは、暗黙的に行われることもあれば、明示的に行うこともできます。関係するデータ・タイプによっては、cast 関数、CAST 仕様、または XMLCAST 仕様を使用して、データ・タイプを明示的に変更することができます。さらに、ソース関数から派生するユーザー定義関数を作成するときは、ソース関数のパラメーターのデータ・タイプが、作成しようとしている関数のデータ・タイプにキャスト可能でなければなりません。

組み込みデータ・タイプの間でサポートされているキャストを、 86 ページの表 9 に示します。第 1 列がキャスト・オペランドのデータ・タイプ (ソース・データ・タイプ) を表し、ヘッダー行に並べた各データ・タイプがキャスト操作のターゲット・データ・タイプを表します。 Y は、ソースとターゲットのデータ・タイプの組み合わせに対して CAST 仕様を使用できることを示します。 XMLCAST 仕様のみを使用できるケースでは、その旨注記されています。

データ・タイプが、文字または GRAPHIC データ・タイプにキャストされるときに切り捨てが行われる場合、非ブランク文字が切り捨てられると警告が戻されます。この切り捨て動作は、非ブランク文字が切り捨てられる場合にエラーが起こるときの、文字または GRAPHIC データ・タイプへの割り当てとは異なります。

特殊タイプに関する以下のキャストがサポートされています。(他に注意書きがなければ、CAST 仕様を使用しています。)

- 特殊タイプ *DT* から、そのソース・データ・タイプ *S* へのキャスト

- 特殊タイプ *DT* のソース・データ・タイプ *S* から、特殊タイプ *DT* へのキャスト
- 特殊タイプ *DT* から、それと同じ特殊タイプ *DT* へのキャスト
- データ・タイプ *A* から、特殊タイプ *DT* へのキャスト。ただし、 *A* は特殊タイプ *DT* のソース・データ・タイプ *S* へプロモート可能なもの
- INTEGER から、ソース・データ・タイプが SMALLINT である特殊タイプ *DT* へのキャスト
- DOUBLE から、ソース・データ・タイプが REAL である特殊タイプ *DT* へのキャスト
- DECFLOAT から、ソース・データ・タイプが CHAR である特殊タイプ DT へのキャスト
- VARCHAR から、ソース・データ・タイプが CHAR である特殊タイプ *DT* へのキャスト
- VARGRAPHIC から、ソース・データ・タイプが GRAPHIC である特殊タイプ *DT* へのキャスト
- Unicode データベースの場合、VARCHAR または VARGRAPHIC から、ソース・データ・タイプが CHAR または GRAPHIC である特殊タイプ *DT* へのキャスト
- ソース・データ・タイプが *S* である特殊タイプ *DT* から XML への、XMLCAST 仕様を使用したキャスト
- XML から、任意の組み込みデータ・タイプのソース・データ・タイプをもった特殊タイプ *DT* への、XMLCAST 仕様を使用したキャスト (XML 値の XML スキーマ・データ・タイプによる)

FOR BIT DATA 文字タイプを CLOB にキャストすることはできません。

ターゲットとして配列タイプが関係するキャストの場合、ソース配列値のエレメントのデータ・タイプは、ターゲット配列データのエレメントのデータ・タイプに対してキャスト可能でなければなりません (SQLSTATE 42846)。ターゲット配列タイプが通常配列の場合、ソース配列値は通常配列でなければならず (SQLSTATE 42821)、ソース配列値のカーディナリティーはターゲット配列データ・タイプの最大カーディナリティー以下でなければなりません (SQLSTATE 2202F)。ターゲット配列タイプが連想配列の場合、ソース配列値の索引のデータ・タイプは、ターゲット配列タイプの索引のデータ・タイプにキャスト可能でなければなりません。ユーザー定義配列タイプ値をキャストできるのは、同じユーザー定義配列タイプに対してのみです (SQLSTATE 42846)。

カーソル・タイプは、パラメーター・マーカーをカーソル・タイプにキャストする場合を除き、CAST 仕様のソース・データ・タイプにもターゲット・データ・タイプにもできません。

ターゲットとして行タイプが関係するキャストの場合、ソース行の値式の度合いとターゲット行タイプの度合いが一致し、ソース行の値式の各フィールドを対応するターゲット・フィールドにキャストできなければなりません。ユーザー定義行タイプ値をキャストできるのは、名前が同じ別のユーザー定義行タイプに対してのみです (SQLSTATE 42846)。

構造化タイプの値を何か別のものにキャストすることはできません。 *ST* のスーパータイプに対するすべてのメソッドは、*ST* に当てはまるので、構造化タイプ *ST* を、そのスーパータイプのいずれかにキャストすべきではありません。必要な操作が *ST* のサブタイプだけに当てはまる場合、サブタイプ処理式を使用して、*ST* をサブタイプの 1 つとして扱います。

キャストに関与したユーザー定義データ・タイプがスキーマ名によって修飾されていない場合、 *SQL* パス が、ユーザー定義データ・タイプを組み入れられた最初のスキーマをその名前で検出するために使用されます。

参照タイプに関して、以下のキャストがサポートされています。

- 参照タイプ *RT* から、表記データ・タイプ *S* へのキャスト
- 参照タイプ *RT* の表記データ・タイプ *S* から、参照タイプ *RT* へのキャスト
- ターゲット・タイプが *T* である参照タイプ *RT* から、ターゲット・タイプが *S* である参照タイプ *RS* へのキャスト (*S* は *T* のスーパータイプ)
- データ・タイプ *A* から、参照タイプ *RT* へのキャスト (ただし *A* は、参照タイプ *RT* の表記データ・タイプ *S* へプロモート可能なもの)

キャストに関与した参照データ・タイプのターゲット・タイプがスキーマ名によって修飾されていない場合、 *SQL* パス が、ユーザー定義データ・タイプを組み入れられた最初のスキーマをその名前で検出するために使用されます。

表 9. 組み込みデータ・タイプ間のサポートされるキャスト

	ターゲット・データ・タイプ																				
ソース・データ・タイプ	**SMALLINT**	**INTEGER**	**BIGINT**	**DECIMAL**	**REAL**	**DOUBLE**	**DECFLOAT**	**CHAR**	**CHAR FOR BIT DATA**	**VARCHAR**	**VARCHAR FOR BIT DATA**	**CLOB**	**GRAPHIC**	**VARGRAPHIC**	**DBCLOB**	**BLOB**	**DATE**	**TIME**	**TIMESTAMP**	**XML**	**BOOLEAN**
SMALLINT	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	-	Y[1]	Y[1]	-	-	-	-	-	Y[3]	Y[7]
INTEGER	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	-	Y[1]	Y[1]	-	-	-	-	-	Y[3]	Y[7]
BIGINT	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	-	Y[1]	Y[1]	-	-	-	-	-	Y[3]	Y[7]
DECIMAL	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	-	Y[1]	Y[1]	-	-	-	-	-	Y[3]	-
REAL	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	-	Y[1]	Y[1]	-	-	-	-	-	Y[3]	-
DOUBLE	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	-	Y[1]	Y[1]	-	-	-	-	-	Y[3]	-
DECFLOAT	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	-	Y[1]	Y[1]	-	-	-	-	-	-	-
CHAR	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y[1]	Y[1]	Y[1]	Y	Y	Y	Y	Y[4]	-
CHAR FOR BIT DATA	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	-	-	-	-	Y	Y	Y	Y	Y[3]	-
VARCHAR	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y[1]	Y[1]	Y[1]	Y	Y	Y	Y	Y[4]	-

表 9. 組み込みデータ・タイプ間のサポートされるキャスト (続き)

	ターゲット・データ・タイプ																				
ソース・データ・タイプ	SMALLINT	INTEGER	BIGINT	DECIMAL	REAL	DOUBLE	DECFLOAT	CHAR	CHAR FOR BIT DATA[2]	VARCHAR	VARCHAR FOR BIT DATA[2]	CLOB	GRAPHIC	VARGRAPHIC	DBCLOB	BLOB	DATE	TIME	TIMESTAMP	XML	BOOLEAN
VARCHAR FOR BIT DATA	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	-	-	-	-	Y	Y	Y	Y	Y[3]	-
CLOB	-	-	-	-	-	-	-	Y	-	Y	-	Y	Y[1]	Y[1]	Y[1]	Y	-	-	-	Y[4]	-
GRAPHIC	Y[1]	Y[1]	Y[1]	Y[1]	Y[1]	Y[1]	Y[1]	Y[1]	-	Y[1]	-	Y[1]	Y	Y	Y	Y	Y[1]	Y[1]	Y[1]	Y[3]	-
VARGRAPHIC	Y[1]	Y[1]	Y[1]	Y[1]	Y[1]	Y[1]	Y[1]	Y[1]	-	Y[1]	-	Y[1]	Y	Y	Y	Y	Y[1]	Y[1]	Y[1]	Y[3]	-
DBCLOB	-	-	-	-	-	-	-	Y[1]	-	Y[1]	-	Y[1]	Y	Y	Y	Y	-	-	-	Y[3]	-
BLOB	-	-	-	-	-	-	-	-	Y	-	Y	-	-	-	-	Y	-	-	-	Y[4]	-
DATE	-	Y	Y	Y	-	-	-	Y	Y	Y	Y	-	Y[1]	Y[1]	-	-	Y	-	Y	Y[3]	-
TIME	-	Y	Y	Y	-	-	-	Y	Y	Y	Y	-	Y[1]	Y[1]	-	-	-	Y	-	Y[3]	-
TIMESTAMP	-	-	Y	Y	-	-	-	Y	Y	Y	Y	-	Y[1]	Y[1]	-	-	Y	Y	Y	Y[3]	-
XML	Y[5]	Y[5]	Y[5]	Y[5]	Y[5]	Y[5]	Y[5]	Y[5]	Y[5]	Y[5]	Y[5]	Y[5]	Y[5]	Y[5]	Y[5]	Y[5]	Y[5]	Y[5]	Y[5]	Y	-
BOOLEAN	Y[7]	Y[7]	Y[7]	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	Y[7]

注

- ユーザー定義タイプおよび参照タイプに関してサポートされているキャストについては、この表の前にある説明を参照してください。
- 構造化タイプの値を何か別のものにキャストすることはできません。
- データ・タイプ LONG VARCHAR と LONG VARGRAPHIC は、引き続きサポートされていますが、非推奨になっており、将来のリリースで除去される可能性があります。

[1] キャストは、Unicode データベースの場合にのみサポートされます。

[2] FOR BIT DATA

[3] キャストは XMLCAST を使用しないと実行できません。

[4] ストリングを XML 列に割り当てる (INSERT または UPDATE) ときに、XMLPARSE 関数が暗黙的に処理されて、ストリングを XML に変換します。割り当てを正常に完了するには、そのストリングが整形式 XML 文書でなければなりません。

[5] キャストは XMLCAST を使用しないと実行できず、XML 値の基礎となる XML スキーマ・データ・タイプに依存します。詳細は、『XMLCAST』の項を参照してください。

[6] カーソル・タイプは、パラメーター・マーカーをカーソル・タイプにキャストする場合を除き、CAST 仕様のソース・データ・タイプにもターゲット・データ・タイプにもできません。

[7] CAST 仕様を使用する場合のみサポートされます。 cast 関数は存在しません。

表 10 は、識別されたターゲット・データ・タイプへキャストするときに適用する規則に関する情報を見つける場所を示しています。

表 10. データ・タイプへのキャストに関する規則

ターゲット・データ・タイプ	規則
SMALLINT	ソース・タイプが BOOLEAN の場合、TRUE が 1 にキャストされ、FALSE は 0 にキャストされます。その他すべてのソース・タイプの場合、「*SQL リファレンス 第 1 巻*」の『SMALLINT スカラー関数』を参照してください。
INTEGER	ソース・タイプが BOOLEAN の場合、TRUE は 1 にキャストされ、FALSE は 0 にキャストされます。その他すべてのソース・タイプについては、「*SQL リファレンス 第 1 巻*」の『INTEGER スカラー関数』を参照してください。
BIGINT	ソース・タイプが BOOLEAN の場合、TRUE は 1 にキャストされ、FALSE は 0 にキャストされます。その他すべてのソース・タイプについては、「*SQL リファレンス 第 1 巻*」の『BIGINT スカラー関数』を参照してください。
DECIMAL	「*SQL リファレンス 第 1 巻* 」の『DECIMAL スカラー関数』
NUMERIC	「*SQL リファレンス 第 1 巻* 」の『DECIMAL スカラー関数』
REAL	「*SQL リファレンス 第 1 巻* 」の『REAL スカラー関数』
DOUBLE	「*SQL リファレンス 第 1 巻* 」の『DOUBLE スカラー関数』
DECFLOAT	「*SQL リファレンス 第 1 巻* 」の『DECFLOAT スカラー関数』
CHAR	「*SQL リファレンス 第 1 巻* 」の『CHAR スカラー関数』
VARCHAR	「*SQL リファレンス 第 1 巻* 」の『VARCHAR スカラー関数』
CLOB	「*SQL リファレンス 第 1 巻* 」の『CLOB スカラー関数』
GRAPHIC	「*SQL リファレンス 第 1 巻* 」の『GRAPHIC スカラー関数』
VARGRAPHIC	「*SQL リファレンス 第 1 巻* 」の『VARGRAPHIC スカラー関数』
DBCLOB	「*SQL リファレンス 第 1 巻* 」の『DBCLOB スカラー関数』
BLOB	「*SQL リファレンス 第 1 巻* 」の『BLOB スカラー関数』

表 10. データ・タイプへのキャストに関する規則 (続き)

ターゲット・データ・タイプ	規則
DATE	「*SQL リファレンス 第 1 巻* 」の『DATE スカラー関数』
TIME	「*SQL リファレンス 第 1 巻* 」の『TIME スカラー関数』
TIMESTAMP	ソース・タイプが文字ストリングの場合、「*SQL リファレンス 第 1 巻* 」の『TIMESTAMP スカラー関数』を参照してください。そこでは、1 つのオペランドが指定されています。ソース・データ・タイプが DATE の場合、タイム・スタンプは指定された日付と時刻 00:00:00 から構成されます。
BOOLEAN	ソース・タイプが数値の場合、0 は FALSE にキャストされ、1 は TRUE にキャストされます。 NULL は NULL にキャストされます。

XML 以外の値から XML 値へのキャスト

表 11. XML 以外の値から XML 値への、サポートされているキャスト

	ターゲット・データ・タイプ	
ソース・データ・タイプ	XML	結果の XML スキーマ型
SMALLINT	Y	xs:short
INTEGER	Y	xs:int
BIGINT	Y	xs:long
DECIMAL または NUMERIC	Y	xs:decimal
REAL	Y	xs:float
DOUBLE	Y	xs:double
DECFLOAT	N	-
CHAR	Y	xs:string
VARCHAR	Y	xs:string
CLOB	Y	xs:string
GRAPHIC	Y	xs:string
VARGRAPHIC	Y	xs:string
DBCLOB	Y	xs:string
DATE	Y	xs:date
TIME	Y	xs:time
TIMESTAMP	Y	xs:dateTime[1]
BLOB	Y	xs:base64Binary
文字タイプ FOR BIT DATA	Y	xs:base64Binary
特殊タイプ		この図表は、特殊タイプのソース・タイプで使用してください。

表 11. XML 以外の値から XML 値への、サポートされているキャスト (続き)

	ターゲット・データ・タイプ	
ソース・データ・タイプ	**XML**	**結果の XML スキーマ型**

注

[1] ソース・データ・タイプの TIMESTAMP では、0 から 12 までのタイム・スタンプの精度をサポートします。 xs:dateTime の小数秒の最大精度は 6 です。TIMESTAMP のソース・データ・タイプのタイム・スタンプの精度が 6 を超えている場合、xs:dateTime にキャストすると値は切り捨てられます。

データ・タイプ LONG VARCHAR と LONG VARGRAPHIC は、引き続きサポートされていますが、非推奨になっており、将来のリリースで除去される可能性があります。

文字ストリング値を XML 値にキャストする場合、その結果の xs:string アトミック値に、不正な XML 文字が入っていてはなりません (SQLSTATE 0N002)。入力文字ストリングが Unicode でない場合、入力文字は Unicode に変換されます。

SQL バイナリー形式へキャストすると、その結果は、タイプが xs:base64Binary の XQuery アトミック値になります。

XML 値から XML 以外の値へのキャスト

XML 値から XML 以外の値への XMLCAST は、2 つのキャストに分かれます。つまり、ソースの XML 値を、SQL ターゲット・タイプに対応する XQuery タイプに変換する XQuery キャストと、その後に続く、対応する XQuery タイプから実際の SQL タイプへのキャストです。

XMLCAST がサポートされるのは、ターゲット・タイプに対応する、サポートされた XQuery ターゲット・タイプがあり、かつソース値のタイプから対応する XQuery ターゲット・タイプへの、サポートされた XQuery キャストがある場合です。XQuery キャストで使用されるターゲット・タイプは、対応する XQuery ターゲット・タイプを基にしたものであり、さらに別の制約事項を伴う場合があります。

以下の表は、そのような変換の結果の XQuery タイプを一覧で示しています。

表 12. XML 値から XML 以外の値への、サポートされているキャスト

	ソース・データ・タイプ	
ターゲット・データ・タイプ	**XML**	**対応する XQuery ターゲット・タイプ**
SMALLINT	Y	xs:short
INTEGER	Y	xs:int
BIGINT	Y	xs:long
DECIMAL または NUMERIC	Y	xs:decimal
REAL	Y	xs:float
DOUBLE	Y	xs:double
DECFLOAT	Y	一致するタイプがありません[1]
CHAR	Y	xs:string

表 12. XML 値から XML 以外の値への、サポートされているキャスト (続き)

	ソース・データ・タイプ	
ターゲット・データ・タイプ	**XML**	**対応する XQuery ターゲット・タイプ**
VARCHAR	Y	xs:string
CLOB	Y	xs:string
GRAPHIC	Y	xs:string
VARGRAPHIC	Y	xs:string
DBCLOB	Y	xs:string
DATE	Y	xs:date
TIME (時間帯なし)	Y	xs:time
TIMESTAMP (時間帯なし)	Y	xs:dateTime[2]
BLOB	Y	xs:base64Binary
CHAR FOR BIT DATA	N	キャスト不能
VARCHAR FOR BIT DATA	Y	xs:base64Binary
特殊タイプ		この図表は、特殊タイプのソース・タイプで使用してください。
行、参照、構造化されたデータ・タイプまたは抽象データ・タイプ (ADT)、その他	N	キャスト不能

注

[1] DB2 は XML スキーマ 1.0 をサポートしますが、これは DECFLOAT に一致する XML スキーマ・タイプを提供していません。XMLCAST の XQuery キャストの手順の処理は、以下のように処理されます。

- ソース値が XML スキーマの数値タイプで入力される場合、その数値タイプを使用します。
- ソース値が XML スキーマ・タイプ xs:boolean で入力される場合、xs:double を使用します。
- それ以外の場合、有効な数値形式の追加検査をして、xs:string を使用します。

[2] xs:dateTime の小数秒の最大精度は 6 です。ソース・データ・タイプの TIMESTAMP では、0 から 12 までのタイム・スタンプの精度をサポートします。TIMESTAMP のターゲット・データ・タイプのタイム・スタンプの精度が 6 より小さい場合、xs:dateTime からキャストすると値は切り捨てられます。TIMESTAMP のターゲット・データ・タイプのタイム・スタンプの精度が 6 を超える場合、xs:dateTime からキャストすると値にはゼロが埋め込まれます。

以下の制約の場合、制約から派生する XML スキーマ・データ・タイプが、XQuery キャストのターゲット・データ・タイプとして効果的に使用されます。

- CHAR および VARCHAR 以外のストリング・タイプに変換される XML 値は、文字またはバイトの切り捨てなしに、DB2 の該当タイプの長さ制限に収まらなければなりません。派生する XML スキーマ・タイプに使用される名前は、大文字の SQL タイプ名の後に、下線文字とストリングの最大長が続いたものになります。例えば、XMLCAST ターゲット・データ・タイプが CLOB(1M) の場合は CLOB_1048576 となります。

XML 値が CHAR または VARCHAR いずれかのタイプに変換されるときに、そのタイプの最大長が短くてすべてのデータを入れることができない場合、指定データ・タイプに収まるようにデータが切り捨てられます。このとき、エラーは返されません。非ブランク文字が切り捨てされる場合、警告 (SQLSTATE 01004) が返されます。値の切り捨てによってマルチバイト文字が切り捨てられることになる場合、マルチバイト文字全体が除去されます。そのため、切り捨てによって生成されるストリングは、予想より短くなる場合もあります。例えば文字 ñ は、UTF-8 では「C3 B1」という 2 バイトで示されます。この文字が VARCHAR(1) としてキャストされて「C3 B1」が 1 バイトに切り捨てられると、その文字の一部である「C3」が残ります。この文字の一部となる「C3」も除去されるため、最終結果は空ストリングになります。

- DECIMAL 値に変換される XML 値には、小数点の前 (左側) に (*precision* - *scale*) を超える桁数を含めることはできません。scale (位取り) を超える小数点以下の余分の桁は切り捨てられます。派生する XML スキーマ・タイプに使用される名前は、DECIMAL_*precision_scale* となります。ただし、*precision* は、ターゲットの SQL データ・タイプの精度であり、*scale* は、ターゲットの SQL データ・タイプの位取りです。例えば、XMLCAST ターゲット・データ・タイプが DECIMAL(9,2) の場合は、DECIMAL_9_2 となります。
- TIME 値に変換される XML 値内には、小数点以降にゼロ以外の数字をもった秒コンポーネントを置くことはできません。派生する XML スキーマ・タイプに使用される名前は、TIME です。

派生した XML スキーマ・タイプ名がメッセージ中に現れるのは、XML 値が、制約事項のいずれかに合致しない場合だけです。このタイプ名はエラー・メッセージの理解に役立ちますが、定義済みのどの XQuery タイプにも対応しません。入力値が、派生した XML スキーマ・タイプ (対応する XQuery ターゲット・タイプ) の基本タイプに準拠しない場合、エラー・メッセージには、そのタイプが代わりに示されることがあります。このような、派生した XML スキーマ・タイプ名のフォーマットは、将来変更される可能性があるので、プログラミング・インターフェースとして使用しないでください。

XQuery キャストでの XML 値の処理の前に、シーケンス中のすべての文書ノードは除去され、除去された文書ノードの直接の子はそれぞれ、そのシーケンス中の項目になります。文書ノードが複数の直接下位ノードをもっていた場合、改訂後のシーケンスの項目数は、元のシーケンスより多くなります。次に、XQuery fn:data 関数を使用して、文書ノードのない XML 値が原子化されます。その結果として生じる原子化シーケンス値は XQuery キャストで使用されます。原子化シーケンス値が空のシーケンスである場合、それ以上の処理を行うことなく、キャストから NULL 値が戻されます。原子化シーケンス値に複数の項目があると、エラーが戻されます (SQLSTATE 10507)。

XMLCAST のターゲット・タイプが SQL データ・タイプの DATE、TIME、または TIMESTAMP である場合、XQuery キャストの結果の XML 値も UTC に調整され、その値の時間帯コンポーネントは除去されます。

対応する XQuery ターゲット・タイプ値から SQL ターゲット・タイプへの変換時には、xs:base64Binary や xs:hexBinary などのバイナリーの XML データ・タイプは、文字フォームから実際のバイナリー・データに変換されます。

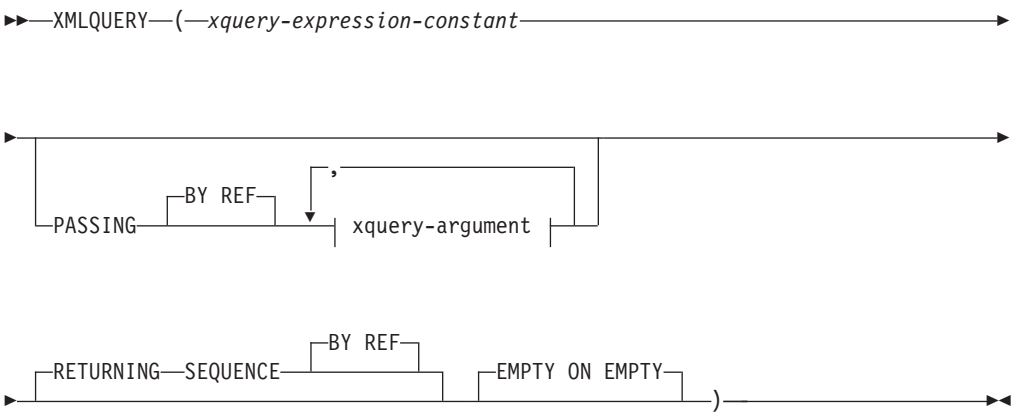
INF、-INF、または NaN の xs:double または xs:float 値を SQL データ・タイプ DOUBLE または REAL 値にキャストする (XMLCAST を使用して) と、エラーが戻されます (SQLSTATE 22003)。 -0 の xs:double または xs:float 値は、+0 に変換されます。

ソース・オペランドがユーザー定義特殊タイプでない場合、ターゲット・タイプはユーザー定義特殊タイプであっても構いません。そのような場合、XMLCAST 仕様を使用してソース値がユーザー定義特殊タイプ (つまり、ターゲット・タイプ) のソース・タイプにキャストされた後、CAST 仕様を使用してこの値がユーザー定義特殊タイプにキャストされます。

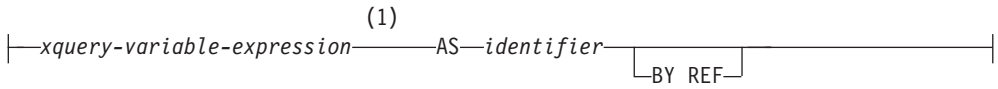
非 Unicode データベースでは、XML 値から XML 以外のターゲット・タイプへのキャストに、内部の UTF-8 形式からデータベース・コード・ページへのコード・ページ変換が含まれます。この変換は、XML 値のコード・ポイントがデータベース・コード・ページに存在しない場合に、置換文字を導入する結果になります。

XMLQUERY

場合によっては指定した入力引数を XQuery 変数として使用して、XMLQUERY 関数は XML 値を XQuery 式の評価から戻します。



xquery-argument:



注:

1 式のデータ・タイプを DECFLOAT にすることはできません。

スキーマは SYSIBM です。関数名を修飾名で指定することはできません。

xquery-expression-constant
サポートされる XQuery 言語構文を使用して、XQuery 式として解釈される SQL 文字ストリング定数を指定します。定数ストリングは、XQuery ステートメントとして構文解析される前に UTF-8 に変換されます。 XQuery 式は、オプション・セットの入力 XML 値を使用して実行し、XMLQUERY 式の値として

も戻される出力シーケンスを戻します。 *xquery-expression-constant* の値は、空ストリングまたはブランク文字のストリングにすることはできません (SQLSTATE 10505)。

PASSING
入力値、およびそれらの値を *xquery-expression-constant* で指定された XQuery 式に渡す方法を指定します。デフォルトでは、関数が呼び出された有効範囲内にあるすべての固有の列名が、列の名前を変数名として使用して XQuery 式に暗黙的に渡されます。指定の xquery-argument 内の *identifier* が有効範囲内の列名と一致する場合、明示的な xquery-argument はその暗黙的な列をオーバーライドして XQuery 式に渡されます。

BY REF
デフォルトの受け渡しメカニズムを、データ・タイプ XML の任意の *xquery-variable-expression* または戻り値の参照によると指定します。 XML 値を参照で渡す場合、XQuery の評価は、入力ノード・ツリーがあればそれを使用します。その場合は指定された入力式から直接、元のノードの ID および文書順序を含めすべてのプロパティーを保持したまま使用します。 2 つの引数が同じ XML 値を渡す場合、その 2 つの入力引数の間に含まれている何らかのノードに関係するノード ID 比較および文書順序比較は、同じ XML ノード・ツリー内のノードを参照する場合があります。

この節は、非 XML 値の受け渡しには影響を与えません。非 XML 値は、XML へのキャスト中に値の新規コピーを作成します。

xquery-argument
xquery-expression-constant により指定された XQuery 式に渡される引数を指定します。引数は、値およびその値が渡される方法を指定します。引数には、評価される SQL 式が組み込まれます。

- 結果の値は、XML 型である場合、*input-xml-value* になります。 NULL の XML 値は、XML の空シーケンスに変換されます。
- 結果の値は、XML 型でない場合、XML データ・タイプにキャスト可能でなければなりません。 NULL 値は、XML の空シーケンスに変換されます。変換される値は、*input-xml-value* になります。

xquery-expression-constant が評価されるとき、XQuery 変数は *input-xml-value* と等しい値、および AS 節により指定された名前で示されます。

xquery-variable-expression
実行中に *xquery-expression-constant* により指定された XQuery 式が使用できる値を持つ SQL 式を指定します。式には、シーケンス参照 (SQLSTATE 428F9) または OLAP 関数 (SQLSTATE 42903) を含めることはできません。式のデータ・タイプを DECFLOAT にすることはできません。

AS *identifier*
xquery-variable-expression により生成された値が、*xquery-expression-constant* に XQuery 変数として渡されることを指定します。変数名は *identifier* になります。 XQuery 言語の変数名に先行する先頭のドル記号 ($) は、*identifier* には含められません。 identifier は有効な XQuery 変数名でなければならず、XML NCName に制限されます (SQLSTATE 42634)。 identifier は、長さが 128 バイトを超えてはな

りません。同じ PASSING 節内の 2 つの引数が同じ identifier を使用することはできません (SQLSTATE 42711)。

BY REF
XML 入力値が参照により渡されるように指示します。 XML 値を参照で渡す場合、XQuery の評価は、入力ノード・ツリーがあればそれを使用します。その場合は指定された入力式から直接、元のノードの ID および文書順序を含めすべてのプロパティーを保持したまま使用します。2 つの引数が同じ XML 値を渡す場合、その 2 つの入力引数の間に含まれている何らかのノードに関係するノード ID 比較および文書順序比較は、同じ XML ノード・ツリー内のノードを参照する場合があります。 BY REF が *xquery-variable-expression* に続いて指定されない場合、XML 引数は、PASSING キーワードに続く構文により提供されるデフォルトの受け渡しメカニズムによって渡されます。このオプションは、非 XML 値に指定することはできません。非 XML 値が渡される場合、値は XML に変換されます。このプロセスによりコピーが作成されます。

RETURNING SEQUENCE
XMLQUERY 式がシーケンスを戻すことを指示します。

BY REF
XQuery 式の結果を参照により戻すことを指示します。この値にノードが含まれる場合、XQuery 式の戻り値を使用する式は、元のノードの ID および文書順序を含めすべてのノードのプロパティーを保持したまま、ノード参照を直接受け取ります。参照されるノードは、そのノード・ツリー内で接続されたままです。BY REF 節が指定されず、PASSING が指定されている場合、デフォルトの受け渡しメカニズムが使用されます。 BY REF が指定されず、PASSING も指定されていない場合、デフォルトの受け渡しメカニズムは BY REF です。

EMPTY ON EMPTY
XQuery 式の処理による空のシーケンスの結果を、空のシーケンスとして戻すことを指定します。

結果のデータ・タイプは XML であり、NULL にはできません。

XQuery 式の評価の結果がエラーになる場合、XMLQUERY 関数は XQuery エラーを戻します (SQLSTATE クラス '10')。

注

- **XMLQUERY の使用上の制限**: XMLQUERY 関数は、下記のものにはできません。
  - JOIN 演算子または MERGE ステートメントと関連した ON 節の一部 (SQLSTATE 42972)
  - CREATE INDEX EXTENSION ステートメントの GENERATE KEY USING または RANGE THROUGH 節の一部 (SQLSTATE 428E3)
  - CREATE FUNCTION (外部スカラー) ステートメント内の FILTER USING 節の一部、または CREATE INDEX EXTENSION ステートメント内の FILTER USING 節の一部 (SQLSTATE 428E4)
  - チェック制約の一部、または列生成式の一部 (SQLSTATE 42621)

- group-by 節の一部 (SQLSTATE 42822)
- 列関数の引数の一部 (SQLSTATE 42607)

- **副照会としての XMLQUERY**: 副照会として動作する XMLQUERY 式は、副照会を制限するステートメントにより制限される可能性があります。

XMLTABLE 関数の概要

XMLTABLE は、XQuery 式の評価から表を戻す SQL 表関数です。 XQuery 式は通常は値をシーケンスとして戻しますが、XMLTABLE を使うと、XQuery 式を実行して値を表として戻すことが可能になります。戻される表には、あらゆる SQL データ・タイプの列を含めることができます (XML を含む)。

XMLQUERY 関数のように、XMLTABLE で指定された XQuery 式に、変数を渡すことができます。 XQuery 式の結果は、結果として生じる表の列値を生成するために使用されます。結果として生じる表の構造は、XMLTABLE の COLUMNS 節によって定義されます。この節では、列名、データ・タイプ、および列値が生成される方法を指定して、列の特性を定義します。名前を明示的に指定しないでも列名を引き渡せるより簡単な構文も使用できます。 114 ページの『XMLEXISTS、XMLQUERY、または XMLTABLE を使用した列名の簡単な引き渡し』を参照してください。

結果として生じる表の列値は、XMLTABLE の PATH 節に XQuery 式を指定して生成できます。 XQuery 式が PATH 節に指定されていない場合、列値を生成するために、列名が XQuery 式として使用され、 XMLTABLE の中で先に指定された XQuery 式の結果が、列値を作成するときに外部コンテキスト項目になります。列値を生成する PATH 節の XQuery 式が空のシーケンスを戻した場合のために、オプションのデフォルト節を指定して列のデフォルト値を提供することもできます。

例えば、以下の SELECT ステートメントは、XMLTABLE 関数内の XQuery 式で、CUSTOMER 表の INFO 列を参照します。

```
SELECT X.*
FROM CUSTOMER C, XMLTABLE ('$INFO/customerinfo'
               COLUMNS
               CUSTNAME CHAR(30) PATH 'name',
               PHONENUM XML PATH 'phone')
     as X
WHERE C.CID < 1003
```

結果の表の列タイプが XML ではなく、かつ列の値を定義する XQuery 式の結果が空のシーケンスでなければ、XMLCAST が暗黙的に使用されて、XML 値がターゲット・データ・タイプの値に変換されます。

XMLTABLE 関数ではオプションで名前空間を宣言できます。 XMLNAMESPACES 宣言で名前空間を指定する場合、これらの名前空間のバインディングは XMLTABLE 関数呼び出し内のすべての XQuery 式に適用されます。名前空間のバインディングを XMLNAMESPACES 宣言を使用しないで宣言する場合、バインディングは名前空間宣言の後の行 XQuery 式にのみ適用されます。

XMLTABLE の利点

シーケンスの代わりに表を戻すと、以下のような操作を SQL 照会のコンテキスト内から実行できます。

- SQL 全選択の中から行う、XQuery 式の結果に対する繰り返し処理

  例えば次の照会の場合、XMLTABLE 内の XQuery 式 "db2-fn:xmlcolumn ("CUSTOMER.INFO")/customerinfo" を実行した結果として得られた表に対し、SQL 全選択が繰り返し処理を行います。

```
SELECT X.*
FROM XMLTABLE ('db2-fn:xmlcolumn("CUSTOMER.INFO")/customerinfo'
               COLUMNS "CUSTNAME" CHAR(30) PATH 'name',
                       "PHONENUM" XML PATH 'phone')
     as X
```

- 保管された XML 文書から表への値の挿入 (値の挿入については、XMLTABLE の例を参照してください)
- XML 文書からの値に対するソート

  例えば次の照会では、CUSTOMER 表の INFO 列にある XML 文書に保管されたカスタマー名によって結果がソートされます。

```
SELECT X.*
FROM XMLTABLE ('db2-fn:xmlcolumn("CUSTOMER.INFO")/customerinfo'
               COLUMNS "CUSTNAME" CHAR(30) PATH 'name',
                       "PHONENUM" XML PATH 'phone')
     as X
ORDER BY X.CUSTNAME
```

- XML 値を一部はリレーショナルとして、一部は XML として保管する (値の挿入については、XMLTABLE の例を参照してください)

重要: XMLTABLE の PATH オプションに指定された XQuery 式が戻す内容については、次のことが言えます。

- 複数の項目のシーケンスを戻す場合、列のデータ・タイプは XML でなければなりません。 XMLTABLE から戻された値を XML 列に挿入する場合、挿入される値が整形式 XML 文書であることを確認してください。複数の項目を戻すシーケンス処理の例については、値の挿入についての XMLTABLE 例を参照してください。
- 空のシーケンスを戻す場合、その列の値として NULL 値が戻されます。

例: XMLTABLE から戻される値の挿入

XMLTABLE SQL 表関数を使用して、保管された XML 文書内から値を検索し、それを表に挿入できます。

この手法は、分解の簡単な形式です。分解とは、XML 文書の断片をリレーショナル表の列に保管するプロセスのことです。 (より一般的なタイプの分解は、アノテーション付き XML スキーマ分解機能で使用可能になります。アノテーション付き XML スキーマ分解を使用すると、複数の XML 文書を複数の表に同時に分解できます。)

例えば、次の 2 つの XML 文書を CUSTOMER という名前の表に保管したとします。

```
<customerinfo Cid="1001">
  <name>Kathy Smith</name>
  <addr country="Canada">
    <street>25 EastCreek</street>
    <city>Markham</city>
    <prov-state>Ontario</prov-state>
    <pcode-zip>N9C 3T6</pcode-zip>
  </addr>
  <phone type="work">905-555-7258</phone>
</customerinfo>

<customerinfo Cid="1003">
  <name>Robert Shoemaker</name>
    <addr country="Canada">
    <street>1596 Baseline</street>
    <city>Aurora</city>
    <prov-state>Ontario</prov-state>
    <pcode-zip>N8X 7F8</pcode-zip>
  </addr>
  <phone type="work">905-555-7258</phone>
  <phone type="home">416-555-2937</phone>
  <phone type="cell">905-555-8743</phone>
  <phone type="cottage">613-555-3278</phone>
</customerinfo>
```

これらの文書からの値を、次のように定義された表に挿入するとします。

```
CREATE TABLE CUSTPHONE (custname char(30), numbers XML)
```

この場合、XMLTABLE を使用する次の INSERT ステートメントにより、CUSTPHONE 表に XML 文書からの値が取り込まれます。

```
INSERT INTO CUSTPHONE
  SELECT X.*
  FROM XMLTABLE ('db2-fn:xmlcolumn("CUSTOMER.INFO")/customerinfo'
                 COLUMNS
                   "CUSTNAME" CHAR(30) PATH 'name',
                   "PHONENUM" XML PATH 'document{<allphones>{phone}</allphones>}'
                ) as X
```

XMLTABLE の PHONENUM 列の PATH 式に対して、XQuery のノード・コンストラクター関数「document{<allphones>{phone}</allphones>}」が指定されていることに注意してください。 XML 列 (この例では NUMBERS 列) に挿入される値は整形式 XML 文書でなければならないので、文書コンストラクターが必要です。この例では、<customerinfo> 文書の Cid="1003" であるすべての <phone> 要素は、次の 4 項目を含む単一のシーケンスで戻されます。

```
{<phone type="work">905-555-7258</phone>,<phone type="home">416-555-2937</phone>,
<phone type="cell">905-555-8743</phone>, <phone type="cottage">613-555-3278</phone>}
```

このシーケンス自体は整形式 XML 文書ではないので NUMBERS XML 列に挿入できません。 phone 値が正常に挿入されるようにするために、シーケンス内のすべての項目が単一の整形式文書に構成されます。

結果として生じる表は、次のようになります (見やすくするために出力は整形されています)。

表 13. 結果表

CUSTNAME	NUMBER
Kathy Smith	<allphones> <phone type="work">905-555-7258</phone> </allphones>
Robert Shoemaker	<allphones> <phone type="work">905-555-7258</phone> <phone type="home">416-555-2937</phone> <phone type="cell">905-555-8743</phone> <phone type="cottage">613-555-3278</phone> </allphones>

例: XMLTABLE を使用して項目のオカレンスごとに 1 行を戻す

XML 文書に 1 つの要素の複数オカレンスがあり、この要素のオカレンスごとに 1 行を生成するには、XMLTABLE を使用します。

例えば、次の 2 つの XML 文書を CUSTOMER という名前の表に保管したとします。

```
<customerinfo Cid="1001">
  <name>Kathy Smith</name>
  <addr country="Canada">
    <street>25 EastCreek</street>
    <city>Markham</city>
    <prov-state>Ontario</prov-state>
    <pcode-zip>N9C 3T6</pcode-zip>
  </addr>
  <phone type="work">905-555-7258</phone>
</customerinfo>

<customerinfo Cid="1003">
  <name>Robert Shoemaker</name>
    <addr country="Canada">
    <street>1596 Baseline</street>
    <city>Aurora</city>
    <prov-state>Ontario</prov-state>
    <pcode-zip>N8X 7F8</pcode-zip>
  </addr>
  <phone type="work">905-555-7258</phone>
  <phone type="home">416-555-2937</phone>
  <phone type="cell">905-555-8743</phone>
  <phone type="cottage">613-555-3278</phone>
</customerinfo>
```

すべての <phone> 値が別個の行に保管される表を作成するには、XMLTABLE を次のように使用します。

```
SELECT X.*
FROM CUSTOMER C, XMLTABLE ('$cust/customerinfo/phone' PASSING C.INFO as "cust"
                           COLUMNS "CUSTNAME" CHAR(30) PATH '../name',
                           "PHONETYPE" CHAR(30) PATH '@type',
                           "PHONENUM" CHAR(15) PATH '.'
                          ) as X
```

この照会により、2 つの XML 文書に関して次の結果が生じます。

表 14. 結果表

CUSTNAME	PHONETYPE	PHONENUM
Kathy Smith	work	905-555-7258
Robert Shoemaker	work	905-555-7258
Robert Shoemaker	home	416-555-2937
Robert Shoemaker	cell	905-555-8743
Robert Shoemaker	cottage	613-555-3278

要素 <name> が "Robert Shoemaker" の XML 文書に関して、要素 <phone> の値が、個別の行として戻されていることに注目してください。

同じ文書で、次のように <phone> 要素を XML として抽出することもできます。

```
SELECT X.*
FROM CUSTOMER C, XMLTABLE ('$cust/customerinfo/phone' PASSING C.INFO as "cust"
                           COLUMNS "CUSTNAME" CHAR(30) PATH '../name',
                           "PHONETYPE" CHAR(30) PATH '@type',
                           "PHONENUM" XML PATH '.'
                          ) as X
```

この照会により、2 つの XML 文書に関して次の結果が生じます (出力は見やすくするために整形しています)。

表 15. 結果表

CUSTNAME	PHONETYPE	PHONENUM
Kathy Smith	work	<phone type="work">416-555-1358</phone>
Robert Shoemaker	work	<phone type="work">905-555-7258</phone>
Robert Shoemaker	home	<phone type="home">416-555-2937</phone>
Robert Shoemaker	cell	<phone type="cell">905-555-8743</phone>
Robert Shoemaker	cottage	<phone type="cottage">613-555-3278</phone>

例: XMLTABLE を使用して XML 文書の複数のツリーから要素を処理する

複数の階層、つまりツリーが含まれる XML 文書では、文書の 1 つのツリーの要素に、文書の別のツリーの要素を関連付けることができます。 XPath 式を使用して、XML 文書内の関連要素を処理できます。

以下の例では、XML 文書内の情報に基づいて、設備および備品を新しい建物に移動する場合の表を生成します。 XML 文書には、新しい建物でのアイテムの場所に関連した情報が含まれます。

以下のステートメントは、XML 文書を保管する表 MOVE を作成します。

```
CREATE TABLE MOVE (ID BIGINT NOT NULL PRIMARY KEY, MOVEINFO XML )
```

以下の XML データには、会社のアイテムおよびアイテムの場所に関する情報が含まれます。 XML 情報は 2 つのツリーに分けられます。 1 つのツリーには、所属、タグ番号およびアイテムの説明など、アイテムに関する情報が含まれます。もう 1 つのツリーには、新しい部門の場所、オフィスに割り当てられるフロア、公共のエリア、および保管エリアなど、移動に関する情報が含まれます。

以下のステートメントは、XML 文書を表 MOVE に挿入します。

```
INSERT into MOVE (ID, MOVEINFO) values ( 1, '
<listing>
  <items>
  <item dept="acct" tag="12223">
    <name>laser printer</name>
    <area>common</area>
  </item>
  <item dept="recptn" tag="23665">
    <name>monitor, CRT</name>
    <area>storage</area>
  </item>
  <item dept="acct" tag="42345">
    <name>CPU, desktop</name>
    <area>office</area>
  </item>
  <item dept="recptn" tag="33301">
    <name>monitor, LCD</name>
    <area>office</area>
  </item>
  <item dept="mfg" tag="10002">
    <name>cabinet, 3 dwr</name>
    <area>office</area>
  </item>
  <item dept="acct" tag="65436">
    <name>table, round 1m</name>
    <area>storage</area>
  </item>
  </items>

  <locations>
  <building dept="recptn" >
     <wing>main</wing>
     <floor area="storage">1</floor>
     <floor area="common">1</floor>
     <floor area="office">2</floor>
  </building>

  <building dept="mfg" >
     <wing>east</wing>
     <floor area="storage">1</floor>
     <floor area="common">2</floor>
     <floor area="office">2</floor>
  </building>

  <building dept="acct" >
     <wing>west</wing>
     <floor area="storage">2</floor>
     <floor area="common">1</floor>
     <floor area="office">2</floor>
  </building>
  </locations>

</listing>
')
```

以下の SELECT ステートメントは、アイテム情報と場所の情報を結合し、アイテム情報、部門、新しい場所がリストされた表を作成します。

マッピングのキーは、相対 XPath 軸 `$x/../../` を使用して、アイテム情報と場所の情報を突き合わせます。

```
SELECT T.*
   from MOVE,
  XMLTABLE( '$doc/listing/items/item' PASSING MOVE.MOVEINFO AS "doc"
```

```
  COLUMNS
    ITEM_ID  VARCHAR(10) PATH 'let $x := . return $x/@tag' ,
    DESC     VARCHAR(20) PATH 'let $x := . return $x/name' ,
    DEPT     VARCHAR(15) PATH 'let $x := . return $x/@dept' ,
    WING     VARCHAR(10) PATH 'let $x := . return $x/../../locations/
               building[@dept = $x/@dept]/wing',
    FLOOR    VARCHAR(10) PATH 'let $x := . return $x/../../locations/
               building/floor[@area = $x/area
                              and ../@dept = $x/@dept ]'
  )as T
```

サンプル・データに対して実行すると、このステートメントは以下のデータを戻します。

```
ITEM_ID    DESC                 DEPT           WING       FLOOR
---------- -------------------- -------------- ---------- ----------
12223      printer, laser       acct           west       1
23665      monitor, CRT         recptn         main       1
42345      CPU, desktop         acct           west       2
33301      monitor, LCD         recptn         main       2
10002      cabinet, 3 dwr       mfg            east       2
65436      table, round 1m      acct           west       2
```

例: XMLTABLE を使用した階層データの処理

XML 文書には、ネスト・レベルの変数番号を使用してデータの階層を含めることができます。 XPath 式を使用して、階層データを処理できます。

以下の例では、コンピューターの部品のリストおよび各部品の親コンポーネントを作成します。この情報は、コンピューターのコンポーネントおよびコンポーネントの関係が含まれる XML 文書に基づいています。

以下のステートメントは、XML 文書を保管する表 BOMLIST を作成します。

```
CREATE TABLE BOMLIST (Cid BIGINT NOT NULL PRIMARY KEY, ITEMS XML )
```

この XML 文書にはコンポーネントおよびサブコンポーネントのリストが含まれます。コンポーネントのリストは、コンポーネントのサブコンポーネントについて説明する階層に関連付けられています。コンポーネントがサブコンポーネントで構成されている場合、コンポーネントの各サブコンポーネントはコンポーネントのサブ要素としてリストされます。

以下のステートメントは、XML 文書を表 BOMLIST に挿入します。

```
CREATE TABLE BOMLIST (Cid BIGINT NOT NULL PRIMARY KEY, ITEMS XML )
insert into BOMLIST (Cid, ITEMS) values ( 1, '
<item desc="computersystem" model="L1234123">
  <part desc="computer" partnum="5423452345">
    <part desc="motherboard" partnum="5423452345">
      <part desc="CPU" partnum="6109486697">
        <part desc="register" partnum="6109486697"/>
      </part>
      <part desc="memory" partnum="545454232">
        <part desc="transistor" partnum="6109486697"/>
      </part>
    </part>

    <part desc="diskdrive" partnum="6345634563456">
      <part desc="spindlemotor" partnum="191986123"/>
    </part>
    <part desc="powersupply" partnum="098765343">
      <part desc="powercord" partnum="191986123"/>
```

```
    </part>
  </part>

  <part desc="monitor" partnum="898234234">
    <part desc="cathoderaytube" partnum="191986123"/>
  </part>

  <part desc="keyboard" partnum="191986123">
    <part desc="keycaps" partnum="191986123"/>
  </part>

  <part desc="mouse" partnum="98798734">
    <part desc="mouseball" partnum="98798734"/>
  </part>
</item>
')
```

以下の SELECT ステートメントは、文書全体をナビゲートして、パーツおよび親パーツをリストする表を作成します。

特徴は、XMLTABLE 関数を使用して表 B を作成することです。 XPath 軸 $doc//part の // を使用することによって、Item ノードのすべての part 要素がナビゲートされます。

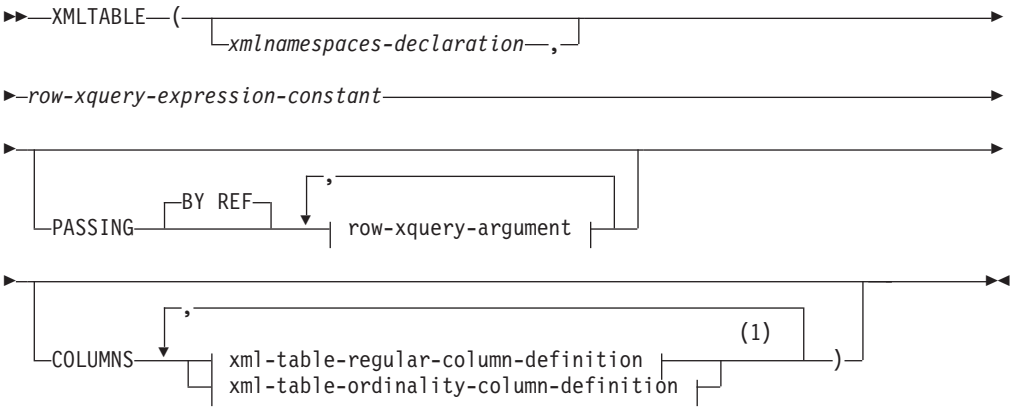
```
SELECT
  A.ITEMNAME,
  B.PART,
  B.PARENT
  FROM BOMLIST ,
  XMLTABLE('$doc/item' PASSING BOMLIST.ITEMS AS "doc"
    COLUMNS
    ITEMNAME VARCHAR(20)  PATH './@desc',
    ITEM     XML          PATH '.'
  )AS A,
  XMLTABLE('$doc//part' PASSING A.ITEM AS "doc"
    COLUMNS
    PART   VARCHAR(20) PATH './@desc',
    PARENT VARCHAR(20) PATH '../@desc'
  )AS B
```

以下のステートメントがデータに対して実行されると、以下の表が表示され、パーツおよび親パーツがリストされます。

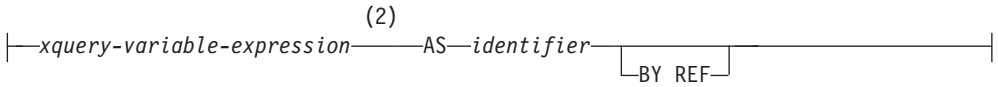
```
ITEMNAME             PART                 PARENT
-------------------- -------------------- --------------------
computersystem       computer             computersystem
computersystem       motherboard          computer
computersystem       CPU                  motherboard
computersystem       register             CPU
computersystem       memory               motherboard
computersystem       transistor           memory
computersystem       diskdrive            computer
computersystem       spindlemotor         diskdrive
computersystem       powersupply          computer
computersystem       powercord            powersupply
computersystem       monitor              computersystem
computersystem       cathoderaytube       monitor
computersystem       keyboard             computersystem
computersystem       keycaps              keyboard
computersystem       mouse                computersystem
computersystem       mouseball            mouse
```

XMLTABLE

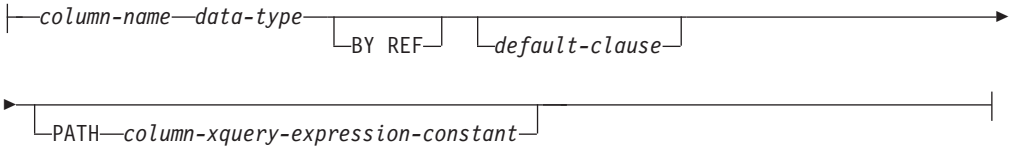
場合によっては指定した入力引数を XQuery 変数として使用して、XMLTABLE 関数は結果表を XQuery 式の評価から戻します。行 XQuery 式の結果シーケンス内の各シーケンス項目は、結果表の行を表しています。



row-xquery-argument:



xml-table-regular-column-definition:



xml-table-ordinality-column-definition:

注:

1 xml-table-ordinality-column-definition 節を複数回指定することはできません (SQLSTATE 42614)。

2 式のデータ・タイプを DECFLOAT にすることはできません。

スキーマは SYSIBM です。関数名を修飾名で指定することはできません。

xmlnamespaces-declaration

row-xquery-expression-constant および *column-xquery-expression-constant* の静的コンテキストの一部になる、1 つ以上の XML 名前空間宣言を指定します。XMLTABLE の引数である XQuery 式の静的に認識される名前空間のセットは、事前設定された静的に認識される名前空間のセットと、この節で指定された

名前空間宣言を組み合わせたものです。 XQuery 式内の XQuery プロローグは、これらの名前空間をオーバーライドする場合があります。

xmlnamespaces-declaration を指定しない場合、事前設定された静的に認識される名前空間のセットだけが XQuery 式に適用されます。

row-xquery-expression-constant
サポートされる XQuery 言語構文を使用して、XQuery 式として解釈される SQL 文字ストリング定数を指定します。定数ストリングは、データベース・コード・ページまたはセクション・コード・ページに変換されることなく、UTF-8 に直接変換されます。 XQuery 式は、オプション・セットの入力 XML 値を使用して実行し、シーケンス内の各項目についての行が生成される出力 XQuery シーケンスを戻します。 *row-xquery-expression-constant* の値は、空ストリングまたはすべてがブランクのストリングにすることはできません (SQLSTATE 10505)。

PASSING
入力値、およびそれらの値を *row-xquery-expression-constant* で指定された XQuery 式に渡す方法を指定します。デフォルトでは、関数が呼び出された有効範囲内にあるすべての固有の列名が、列の名前を変数名として使用して XQuery 式に暗黙的に渡されます。指定の row-xquery-argument 内の *identifier* が有効範囲内の列名と一致する場合、明示的な row-xquery-argument はその暗黙的な列をオーバーライドして XQuery 式に渡されます。

BY REF
デフォルトでは XML 入力引数を参照により渡すことを指定します。 XML 値を参照で渡す場合、XQuery の評価は、入力ノード・ツリーがあればそれを使用します。その場合は指定された入力式から直接、元のノードの ID および文書順序を含めすべてのプロパティーを保持したまま使用します。 2 つの引数が同じ XML 値を渡す場合、その 2 つの入力引数の間に含まれている何らかのノードに関係するノード ID 比較および文書順序比較は、同じ XML ノード・ツリー内のノードを参照する場合があります。

この節は、非 XML 値の受け渡しには影響を与えません。非 XML 値は、XML へのキャスト中に値の新規コピーを作成します。

row-xquery-argument
row-xquery-expression-constant により指定された XQuery 式に渡される引数を指定します。引数は、値およびその値が渡される方法を指定します。引数には、結果を XQuery 式に渡す前に評価される SQL 式が組み込まれます。

- 結果の値は、XML 型である場合、*input-xml-value* になります。 NULL の XML 値は、XML の空シーケンスに変換されます。
- 結果の値は、XML 型でない場合、XML データ・タイプにキャスト可能でなければなりません。 NULL 値は、XML の空シーケンスに変換されます。変換される値は、*input-xml-value* になります。

row-xquery-expression-constant が評価されるとき、XQuery 変数は *input-xml-value* と等しい値、および AS 節により指定された名前で示されます。

xquery-variable-expression
実行中に *row-xquery-expression-constant* により指定された XQuery 式が使用できる値を持つ SQL 式を指定します。式には、NEXT VALUE

式、PREVIOUS VALUE 式 (SQLSTATE 428F9)、または OLAP 関数 (SQLSTATE 42903) を含めることはできません。式のデータ・タイプを DECFLOAT にすることはできません。

AS *identifier*
xquery-variable-expression により生成された値が、*row-xquery-expression-constant* に XQuery 変数として渡されることを指定します。変数名は *identifier* になります。 XQuery 言語の変数名に先行する先頭のドル記号 ($) は、*identifier* には含められません。identifier は有効な XQuery 変数名でなければならず、XML NCName に制限されます。 identifier は、長さが 128 バイトを超えてはなりません。同じ PASSING 節内の 2 つの引数が同じ identifier を使用することはできません (SQLSTATE 42711)。

BY REF
XML 入力値が参照により渡されるように指示します。 XML 値を参照で渡す場合、XQuery の評価は、入力ノード・ツリーがあればそれを使用します。その場合は指定された入力式から直接、元のノードの ID および文書順序を含めすべてのプロパティーを保持したまま使用します。2 つの引数が同じ XML 値を渡す場合、その 2 つの入力引数の間に含まれている何らかのノードに関係するノード ID 比較および文書順序比較は、同じ XML ノード・ツリー内のノードを参照する場合があります。 BY REF が *xquery-expression-variable* に続いて指定されない場合、XML 引数は、PASSING キーワードに続く構文により提供されるデフォルトの受け渡しメカニズムによって渡されます。このオプションは、非 XML 値に指定することはできません (SQLSTATE 42636)。非 XML 値が渡される場合、値は XML に変換されます。このプロセスによりコピーが作成されます。

COLUMNS
結果表の出力列を指定します。この節が指定されない場合、*row-xquery-expression-constant* 内の XQuery 式を評価して得られたシーケンス項目に基づく値が指定されて、データ・タイプ XML の単一の無名列が参照によって戻されます (PATH '.' を指定した場合と同じ結果になります)。結果列を参照するには、関数に続く *correlation-clause* に *column-name* が指定されている必要があります。

xml-table-regular-column-definition
結果表の出力列を指定します。これには列名、データ・タイプ、XML 受け渡しメカニズム、および行のシーケンス項目から値を抽出する XQuery 式が含まれます。

column-name
結果表の列の名前を指定します。名前を修飾したり、表の複数の列に対して同じ名前を使用することはできません (SQLSTATE 42711)。

data-type
列のデータ・タイプを指定します。使用可能な型の構文および説明については、CREATE TABLE を参照してください。 *data-type* は、XML データ・タイプから、指定された *data-type* へのサポートされる XMLCAST がある場合に、XMLTable で使用できます。

BY REF
XML 値を、データ・タイプ XML の列の参照により戻すことを指定します。デフォルトでは、XML 値は BY REF により戻されます。 XML 値を参照で戻す場合、XML 値は、入力ノード・ツリーがあればそれを組み込みます。その場合は結果の値から直接、元のノードの ID および文書順序を含めすべてのプロパティーを保持したまま組み込みます。このオプションは、非 XML 列に指定することはできません (SQLSTATE 42636)。非 XML 列が処理される場合、値は XML から変換されます。このプロセスによりコピーが作成されます。

default-clause
列のデフォルト値を指定します。 *default-clause* の構文および説明については、CREATE TABLE を参照してください。 XMLTABLE 結果列の場合、*column-xquery-expression-constant* に含まれる XQuery 式の処理が空のシーケンスを戻す場合は、デフォルトが適用されます。

PATH *column-xquery-expression-constant*
サポートされる XQuery 言語構文を使用して、XQuery 式として解釈される SQL 文字ストリング定数を指定します。定数ストリングは、データベース・コード・ページまたはセクション・コード・ページに変換されることなく、UTF-8 に直接変換されます。 *column-xquery-expression-constant* は XQuery 式を指定しますが、これは *row-xquery-expression-constant* 内の XQuery 式の評価の結果である項目に関連して列値を決定します。外部で提供されたコンテキスト項目として *row-xquery-expression-constant* の処理の結果による項目がある場合、*column-xquery-expression-constant* が評価され、出力シーケンスが戻されます。列値は、以下のようにこの出力シーケンスに基づいて決定されます。

- 出力シーケンスに含まれている項目がゼロの場合、*default-clause* は列の値を提供します。
- 空のシーケンスが戻され、*default-clause* が指定されていない場合、NULL 値が列に割り当てられます。
- 空でないシーケンスが戻される場合、値は列に指定された *data-type* に対する XMLCAST です。この XMLCAST の処理によりエラーが戻される場合があります。

column-xquery-expression-constant の値は、空ストリングまたはすべてがブランクのストリングにすることはできません (SQLSTATE 10505)。この節が指定されない場合、デフォルトの XQuery 式は単に *column-name* になります。

xml-table-ordinality-column-definition
結果表の順序を示す列を指定します。

column-name
結果表の列の名前を指定します。名前を修飾したり、表の複数の列に対して同じ名前を使用することはできません (SQLSTATE 42711)。

FOR ORDINALITY
column-name が結果表の順序を示す列になるように指定します。この列のデータ・タイプは BIGINT です。 結果表のこの列の値は、

row-xquery-expression-constant 内の XQuery 式を評価した結果シーケンスにおける行の項目の順序番号です。

いずれかの XQuery 式の評価の結果がエラーになる場合、XMLTABLE 関数は XQuery エラーを戻します (SQLSTATE クラス '10')。

例

以下は、注文の購入注文項目で状況が「NEW」の結果である表のリストです。

```
SELECT U."PO ID", U."Part #", U."Product Name",
     U."Quantity", U."Price", U."Order Date"
  FROM PURCHASEORDER P,
     XMLTABLE('$po/PurchaseOrder/item' PASSING P.PORDER AS "po"
              COLUMNS "PO ID"         INTEGER       PATH '../@PoNum',
                      "Part #"        CHAR(10)      PATH 'partid',
                      "Product Name"  VARCHAR(50)   PATH 'name',
                      "Quantity"      INTEGER       PATH 'quantity',
                      "Price"         DECIMAL(9,2)  PATH 'price',
                      "Order Date"    DATE          PATH '../@OrderDate'
              ) AS U
   WHERE P.STATUS = 'Unshipped'
```

XML データを照会するときの XMLEXISTS 述部

XMLEXISTS 述部は、XQuery 式が 1 つ以上の項目のシーケンスを戻すかどうかを判別します。この述部に指定された XQuery 式が空のシーケンスを戻す場合、XMLEXISTS は false を戻します。その他の場合には、true を戻します。

XMLEXISTS 述部は、SELECT ステートメントの WHERE 節で使用できます。この使用法は、保管された XML 文書の値を使用して SELECT 照会の操作対象となる行のセットを絞り込むというものです。

例えば、次の SQL 照会は、XMLEXISTS 述部を使用して戻される行を <city> 要素の値が "Toronto" である XML 文書を含むものだけに制限する方法を示しています。 (XQuery 式は大/小文字を区別しますが、SQL は大/小文字を区別しないことに注意してください。)

```
SELECT Cid
FROM CUSTOMER
WHERE XMLEXISTS ('$d//addr[city="Toronto"]' passing INFO as "d")
```

XMLEXISTS の XQuery 式で、XQuery 変数に値を渡すことができることに注目してください。この例では、CUSTOMER 表の INFO 列の文書に、XQuery 変数 $d がバインドされています。 **passing** 節に名前を明示的に指定しないでも列名を引き渡せるより簡単な構文も使用できます。114 ページの『XMLEXISTS、XMLQUERY、または XMLTABLE を使用した列名の簡単な引き渡し』を参照してください。

期待される結果が戻されるように、XMLEXISTS での XQuery 式が正しく指定されていることを確認してください。例えば、CUSTOMER 表の XML INFO 列に複数の文書が保管されているものの、1 つの文書のみ値が 1000 の Cid 属性 (指定されたパスにある) を含むと仮定します。

```
<customerinfo Cid="1000">
  <name>Kathy Smith</name>
  <addr country="Canada">
```

```
    <street>5 Rosewood</street>
    <city>Toronto</city>
    <prov-state>Ontario</prov-state>
    <pcode-zip>M6W 1E6</pcode-zip>
  </addr>
  <phone type="work">416-555-1358</phone>
</customerinfo>
```

次の 2 つの照会は、XQuery 式が少し異なるために、異なる結果を戻します。

```
SELECT *
FROM CUSTOMER
WHERE XMLEXISTS ('$d/customerinfo[@Cid=1000]' passing INFO as "d")

SELECT *
FROM CUSTOMER
WHERE XMLEXISTS ('$d/customerinfo/@Cid=1000' passing INFO as "d")
```

最初の照会は、上記の XML 文書を含む行を予期されたとおりに戻します。しかし、2 番目の照会は、指定された XQuery 式で XMLEXISTS 述部が常に true を戻すために、CUSTOMER 表のすべての行を戻します。 2 番目の照会の XQuery 式はブール項目のシーケンスを戻し、これは空ではないシーケンスなので、XMLEXISTS は常に true を戻すことになります。その結果、CUSTOMER 表のすべての行が選択されて、予期したとおりの結果が得られません。

XMLEXISTS 述部の使用法

XMLEXISTS に値述部 (*expression*) を持つ XPath 式が含まれている場合、[*expression*] が結果となるように、述部を大括弧で囲みます。値述部を大括弧で囲むと、意味体系上期待されるように *expression* が評価されます。

XMLEXISTS 述部の動作

以下のシナリオは、空でないシーケンス自体には単一値 *false* が含まれていたとしても、その空でないシーケンスのためにどのように XMLEXISTS が *true* と評価されるようになるのかを示しています。索引の突き合わせは行われず、照会は、期待されるよりずっと多くの結果セットを戻します。この問題は、値述部を大括弧 ([]) で適切に囲むことにより回避できます。

以下の 1 つの表、1 つの索引、および 2 つの照会について考慮します。

```
CREATE TABLE mytable (id BIGINT, xmlcol XML);
   CREATE INDEX myidx ON mytable(xmlcol)
   GENERATE KEY USING XMLPATTERN '//text()' AS SQL VARCHAR(255);

SELECT xmlcol FROM mytable
   WHERE XMLEXISTS('$doc/CUSTOMER/ORDERS/ORDERKEY/text()="A512" '
   PASSING xmlcol AS "doc")

SELECT xmlcol FROM mytable
   WHERE XMLEXISTS('$doc/CUSTOMER[ORDERS/ORDERKEY/text()="A512"] '
   PASSING xmlcol AS "doc") ;
```

この動作の原因は次の理由によります: XMLEXISTS は XQuery 式を評価し、結果が空のシーケンスである場合には *false (for XMLEXISTS)* を戻し、結果が空でないシーケンスである場合には *true (for XMLEXISTS)* を戻します。その後、照会評価におけるおそらく直感的ではない次のステップが続きます: 最初の照会で、式は「オーダー・キーを A512 と比較する」ように指示します。その式の結果は、オーダー・キーの実際の値により、*false* または *true* のいずれかの値になります。そのため、XMLEXISTS 関数は、単一

の項目、つまり *false* の項目または *true* の項目のいずれかを含む戻りシーケンスを常に認識します。 1 つの項目を持つシーケンスは空でないシーケンスであるため、XMLEXISTS はすべてにおいて**常に** *true (for XMLEXISTS)* を戻し、そのために照会はすべての行を戻します。すべての行が条件に適合するように XMLEXISTS が使用される場合には、索引は活用できません。

以下に空でないシーケンスの 5 つの例を示します。その中の 3 つは項目が 1 つしかないシーケンスです。

```
(42, 3,4,78, 1966)
(true)
(abd, def)
(false)
(5)
```

空でないそうしたシーケンスにより、XMLEXISTS 自体が認識する空でないシーケンスが (*false*) を戻す場合でも、XMLEXISTS は値 *true (for XMLEXISTS)* を戻します。

2 番目の照会では、XMLEXISTS の内部の式は、「orderkey が A512 に等しいオーダーを持つ顧客を戻す」ように指示します。文書内にそのような顧客が存在しない場合、結果は実際、空のシーケンスになります。この照会は索引を使用し、期待される結果を戻します。

XMLEXISTS 述部の使用法

expression 全体が大括弧の中に置かれた場合、それは「[*expression*] である場合に XML データを戻す」という意味に固定され、XML データが *expression* を満たさない場合には常に空のシーケンスが戻されるはずです。

値の比較は常に大括弧内部にあるため、XMLEXISTS 述部使用法の次の各サンプル・フラグメントは予想通りに処理されます。

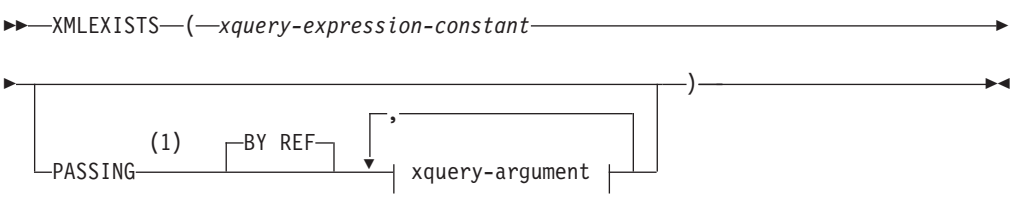
```
... WHERE XMLEXISTS('$doc[CUSTOMER/ORDERS/ORDERKEY/text()="A512"] '
       PASSING xmlcol as "doc") ;
   ... WHERE XMLEXISTS('$doc/CUSTOMER[ORDERS/ORDERKEY/text()="A512"] '
       PASSING xmlcol AS "doc") ;
   ... WHERE XMLEXISTS('$doc/CUSTOMER/ORDERS[ORDERKEY/text()="A512"] '
       PASSING xmlcol AS "doc") ;
```

値比較がない照会に対してもこの指針は正しく働きます。例えば、*COMMENT* 子要素を持つすべての顧客の文書を戻す場合には、次のようにします。

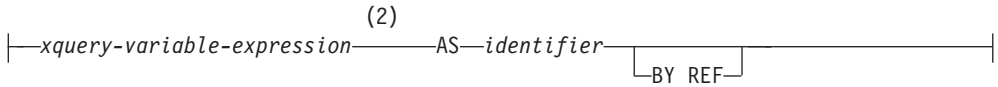
```
... WHERE XMLEXISTS('$doc[CUSTOMER/COMMENT ] '
   PASSING xmlcol AS "doc") ;
```

XMLEXISTS 述部

XMLEXISTS 述部は、XQuery 式が 1 つ以上の項目のシーケンスを戻すかどうかをテストします。



xquery-argument:



注:

1 データ・タイプを DECFLOAT にすることはできません。

2 式のデータ・タイプを DECFLOAT にすることはできません。

xquery-expression-constant
XQuery 式として解釈される SQL 文字ストリング定数を指定します。定数ストリングは、データベース・コード・ページまたはセクション・コード・ページに変換されることなく、UTF-8 に直接変換されます。 XQuery 式は、オプション・セットの入力 XML 値を使用して実行し、XMLEXISTS 述部の結果を決定するためにテストされる出力シーケンスを戻します。 *xquery-expression-constant* の値は、空ストリングまたはブランク文字のストリングにすることはできません (SQLSTATE 10505)。

PASSING
入力値、およびそれらの値を *xquery-expression-constant* で指定された XQuery 式に渡す方法を指定します。デフォルトでは、関数が呼び出された有効範囲内にあるすべての固有の列名が、列の名前を変数名として使用して XQuery 式に暗黙的に渡されます。指定の xquery-argument 内の *identifier* が有効範囲内の列名と一致する場合、明示的な xquery-argument はその暗黙的な列をオーバーライドして XQuery 式に渡されます。

BY REF
デフォルトの受け渡しメカニズムを、データ・タイプ XML の任意の *xquery-variable-expression* の参照によると指定します。 XML 値を参照で渡す場合、XQuery の評価は、入力ノード・ツリーがあればそれを使用します。その場合は指定された入力式から直接、元のノードの ID および文書順序を含めすべてのプロパティーを保持したまま使用します。 2 つの引数が同じ XML 値を渡す場合、その 2 つの入力引数の間に含まれている何らかのノードに関係するノード ID 比較および文書順序比較は、同じ XML ノード・ツリー内のノードを参照する場合があります。

この節は、非 XML 値の受け渡しには影響を与えません。非 XML 値は、XML へのキャスト中に値の新規コピーを作成します。

xquery-argument
xquery-expression-constant により指定された XQuery 式に渡される引数を指定します。引数は、値およびその値が渡される方法を指定します。引数には、評価される SQL 式が組み込まれます。

- 結果の値は、XML 型である場合、*input-xml-value* になります。 NULL の XML 値は、XML の空シーケンスに変換されます。
- 結果の値は、XML 型でない場合、XML データ・タイプにキャスト可能でなければなりません。 NULL 値は、XML の空シーケンスに変換されます。変換される値は、*input-xml-value* になります。

xquery-expression-constant が評価されるとき、XQuery 変数は *input-xml-value* と等しい値、および AS 節により指定された名前で示されます。

xquery-variable-expression
実行中に *xquery-expression-constant* により指定された XQuery 式が使用できる値を持つ SQL 式を指定します。式には、シーケンス参照 (SQLSTATE 428F9) または OLAP 関数 (SQLSTATE 42903) を含めることはできません。式のデータ・タイプを DECFLOAT にすることはできません。

AS *identifier*
xquery-variable-expression により生成された値が、*xquery-expression-constant* に XQuery 変数として渡されることを指定します。変数名は *identifier* になります。 XQuery 言語の変数名に先行する先頭のドル記号 ($) は、*identifier* には含められません。 identifier は有効な XQuery 変数名でなければならず、XML NCName に制限されます。 identifier は、長さが 128 バイトを超えてはなりません。同じ PASSING 節内の 2 つの引数が同じ identifier を使用することはできません (SQLSTATE 42711)。

BY REF
XML 入力値が参照により渡されるように指示します。 XML 値を参照で渡す場合、XQuery の評価は、入力ノード・ツリーがあればそれを使用します。その場合は指定された入力式から直接、元のノードの ID および文書順序を含めすべてのプロパティーを保持したまま使用します。2 つの引数が同じ XML 値を渡す場合、その 2 つの入力引数の間に含まれている何らかのノードに関係するノード ID 比較および文書順序比較は、同じ XML ノード・ツリー内のノードを参照する場合があります。 BY REF が *xquery-variable-expression* に続いて指定されない場合、XML 引数は、PASSING キーワードに続く構文により提供されるデフォルトの受け渡しメカニズムによって渡されます。このオプションは、非 XML 値に指定することはできません。非 XML 値が渡される場合、値は XML に変換されます。このプロセスによりコピーが作成されます。

注

XMLEXISTS 述部は、以下のものにすることはできません。

- JOIN 演算子または MERGE ステートメントと関連した ON 節の一部 (SQLSTATE 42972)
- CREATE INDEX EXTENSION ステートメントの GENERATE KEY USING または RANGE THROUGH 節の一部 (SQLSTATE 428E3)
- CREATE FUNCTION (外部スカラー) ステートメント内の FILTER USING 節の一部、または CREATE INDEX EXTENSION ステートメント内の FILTER USING 節の一部 (SQLSTATE 428E4)
- チェック制約の一部、または列生成式の一部 (SQLSTATE 42621)
- group-by 節の一部 (SQLSTATE 42822)
- 列関数の引数の一部 (SQLSTATE 42607)

副照会に関係する XMLEXISTS 述部は、副照会を制限するステートメントにより制限されることがあります。

例

```
SELECT c.cid FROM customer c
  WHERE XMLEXISTS('$d/*:customerinfo/*:addr[ *:city = "Aurora" ]'
    PASSING info AS "d")
```

SQL ステートメントと XQuery 式におけるパラメーターの引き渡し

SQL ステートメントと XQuery 式を組み合わせて実行する場合、ステートメントと式の実行を変更するためにそのステートメントと式の間でデータを受け渡すことができます。

XMLEXISTS および XMLQUERY への定数およびパラメーター・マーカーの引き渡し

XMLEXISTS 述部と XMLQUERY スカラー関数は、1 つの SQL ステートメントから複数の XQuery 式を実行します。定数およびパラメーター・マーカーを使用して、SQL ステートメントのデータを、その SQL ステートメント内で実行される XQuery 式内の変数に渡します。

XMLEXISTS および XMLQUERY の中で、XQuery 変数を XQuery 式の一部として指定できます。値は PASSING 節を介して渡されます。これらの値は SQL 式です。 XQuery 式に渡されるこれらの値は非 XML 値なので、それらを DB2 XQuery でサポートされるタイプに暗黙的または明示的にキャストする必要があります。サポートされるキャストについて詳しくは、データ・タイプ間でのキャストに関する資料を参照してください。

定数およびパラメーター・マーカーを XMLQUERY に渡す方法は XMLEXISTS の場合と同じですが、XMLEXISTS の方がより一般的に使用されます。これは、XMLQUERY 内のパラメーター化された述部を SELECT 節と共に使用する場合は、結果セットから行が削除されないためです。その代わりに、述部は文書のどの断片を戻すかを決定するために使用されます。結果セットから実際に行を除去するには、WHERE 節内で XMLEXISTS 述部を使用する必要があります。空のシーケンスを含む行は、結果セットの一部として戻されないことになります。ここで説明した例は、XMLEXISTS によるより一般的な使用法を示しています。

例: 暗黙的なキャスト

次の照会で、XML タイプではない SQL 文字ストリング定数 'Aurora' は、XMLEXISTS 述部で XML タイプに暗黙的にキャストされます。暗黙的なキャストの後に、定数は XML スキーマ・サブタイプの xs:string になり、変数 $cityName にバインドされます。こうして、この定数が XQuery 式の述部で使用できるようになります。

```
SELECT XMLQUERY ('$d/customerinfo/addr' passing c.INFO as "d")
FROM Customer as c
WHERE XMLEXISTS('$d//addr[city=$cityName]'
                 passing c.INFO as "d",
                 'Aurora' AS "cityName")
```

例: 明示的なキャスト

次の照会では、パラメーター・マーカーのタイプを判別できないため、パラメーター・マーカーをデータ・タイプに明示的にキャストする必要があります。 SQL VARCHAR タイプに明示的にキャストされたパラメーター・マーカーは、その後 xs:string XML スキーマ・タイプに暗黙的にキャストされます。

```
SELECT XMLQUERY ('$d/customerinfo/addr' passing c.INFO as "d")
FROM Customer as c
WHERE XMLEXISTS('$d//addr[city=$cityName]'
                passing c.INFO as "d",
                CAST (? AS VARCHAR(128)) AS "cityName")
```

XMLEXISTS、XMLQUERY、または XMLTABLE を使用した列名の簡単な引き渡し

XMLEXISTS 述部、XMLQUERY スカラー関数、または XMLTABLE 表関数の使用を簡単にするには、**passing** 節で列名を指定しないで、XMLEXISTS、XMLQUERY、または XMLTABLE で指定した XQuery 式でその列名をパラメーターとして使用できます。

使用しているパラメーター名が引き渡される列名と異なる場合には、パラメーターとして列名を渡す **passing** 節を使用する必要があります。

passing 節で明示的に変数が指定され、その名前が XQuery 式で参照される変数と競合する場合には、**passing** 節の変数が優先されます。 CUSTOMER 表に INFO および CUST という名前の 2 つの XML 列があるとします。以下の例では、INFO 列から XML データを取得します。

```
SELECT XMLQuery('$CUST/customerinfo/name' PASSING INFO as "CUST") FROM customer
```

passing 節で指定された変数 CUST を使用して、XQuery 式の列 INFO が置換されます。CUSTOMER 表の列 CUST は使用されません。

例: XMLQUERY および XMLEXISTS

これらの例では、列名が二重引用符で囲まれているため、大文字と小文字が区別されます。二重引用符で囲まない場合、通常の列名に関する規則が適用されます。列名では大/小文字が区別されず、大文字で保管されます。

以下の例は、PURCHASEORDER 表から同じ文書のシーケンスを戻すいくつかの SELECT ステートメントを示しています。XML 文書は、列 PORDER にあります。最初の 2 つの SELECT ステートメントは **passing** 節を使用して、列名 PORDER を XMLQUERY スカラー関数内の XQuery 式に渡します。3 番目の SELECT は PORDER 列名を受け渡しパラメーターとして暗黙的に使用します。

```
SELECT XMLQuery('$PORDER/PurchaseOrder/item/name'  PASSING porder AS "PORDER")
   FROM purchaseorder
SELECT XMLQuery('$PORDER/PurchaseOrder/item/name' PASSING porder AS "porder")
   FROM purchaseorder
SELECT XMLQuery('$PORDER/PurchaseOrder/item/name') FROM purchaseorder
```

以下の 2 つの例は、XMLQUERY と XMLEXISTS の両方を使用するいくつかの関数呼び出しを示しています。どちらの例も、CUSTOMER 表から同じ文書のシーケンスを戻します。

以下の例では、**passing** 節を使用して XMLQUERY スカラー関数および XMLEXISTS 述部で INFO 列名をパラメーターとして明示的に指定します。

```
SELECT XMLQUERY ('$INFO/customerinfo/addr' passing Customer.INFO as "INFO")
FROM Customer
WHERE XMLEXISTS('$INFO//addr[city=$cityName]'
                passing Customer.INFO as "INFO",
                'Aurora' AS "cityName")
```

以下の例では、XMLQUERY 関数は passing 節を使用せず、XMLEXISTS **passing** 節は INFO 列を指定しません。列名 INFO は、XMLQUERY スカラー関数と XMLEXISTS 述部の両方の XQuery 式に暗黙的に渡されます。

```
SELECT XMLQUERY ('$INFO/customerinfo/addr')
FROM Customer
WHERE XMLEXISTS('$INFO//addr[city=$cityName]'
                passing 'Aurora' AS "cityName")
```

例: XMLTABLE

以下の 2 つの例は、XMLTABLE 表関数を使用する 2 つの INSERT ステートメントを示しています。どちらの例でも、表 T1 の同じデータが表 CUSTOMER に挿入されます。表 T1 には、CUSTLIST という名前の XML 列が含まれています。XMLTABLE 関数は列 T1.CUSTLIST のデータを取得して、Cid、Info、および History という 3 つの列を持つ表を戻します。INSERT ステートメントは、XMLTABLE 関数からのデータを表 CUSTOMER の 3 つの列に挿入します。

以下の例では、**passing** 節を使用して、XMLTABLE 表関数で CUSTLIST 列名をパラメーターとして明示的に指定します。

```
INSERT INTO customer SELECT X.* FROM T1,
    XMLTABLE( '$custlist/customers/customerinfo' passing T1.custlist as "custlist"
    COLUMNS
    "Cid" BIGINT PATH '@Cid',
    "Info" XML PATH 'document{.}',
    "History" XML PATH 'NULL') as X
```

以下の例では、XMLTABLE 表関数は **passing** 節を使用しません。 XMLTABLE は、表 T1 の列名 CUSTLIST を受け渡しパラメーターとして暗黙的に使用します。

```
INSERT INTO customer SELECT X.* FROM T1,
    XMLTABLE( '$custlist/customers/customerinfo'
    COLUMNS
    "Cid" BIGINT PATH '@Cid',
    "Info" XML PATH 'document{.}',
    "History" XML PATH 'NULL') as X
```

XQuery から SQL へのパラメーターの引き渡し

XQuery 式内の db2-fn:sqlquery 関数は SQL 全選択を実行して、XML ノード・シーケンスを取り出します。db2-fn:sqlquery を使用する際に PARAMETER 関数を用いて、db2-fn:sqlquery で指定された、XQuery 式から SQL 全選択ステートメントに引き渡されたデータを参照します。

PARAMETER 関数を使用すると、db2-fn:sqlquery においてパラメーターを SQL 全選択式の一部として指定できます。db2-fn:sqlquery 呼び出しで PARAMETER 関数を使用する場合、その PARAMETER 関数が使用することになる XQuery 式も指定する必要があります。SQL 全選択の処理中、各 PARAMETER 関数呼び出しは、

db2-fn:sqlquery 関数呼び出しにおいて対応する XQuery 式の結果値で置き換えられます。PARAMETER 関数によって提供される値は、同じ SQL ステートメント内で複数回参照できます。

db2-fn:sqlquery 関数呼び出しの一部である XQuery 式は値を戻します。全選択に渡されるこれらの値は XML 値なので、それらを DB2 SQL でサポートされるタイプに暗黙的または明示的にキャストする必要があります。サポートされるキャストについて詳しくは、db2-fn:sqlquery 資料およびデータ・タイプ間でのキャストに関する資料を参照してください。

例: db2-fn:sqlquery へのパラメーターの引き渡し

以下は、db2-fn:sqlquery を使用した XQuery 式の例です。db2-fn:sqlquery 関数の処理中、parameter(1) への参照の両方が、注文日の属性の値 $po/@OrderDate を戻します。

DB2 SAMPLE データベースに対してこの XQuery 式を実行すると、販売促進期間中に販売されたすべての部品に関する購入 ID、部品 ID、および購入日を戻します。

```
xquery
for $po in db2-fn:xmlcolumn('PURCHASEORDER.PORDER')/PurchaseOrder,
  $item in $po/item/partid
for $p in db2-fn:sqlquery(
  "select description
  from product
  where promostart < parameter(1)
  or
  promoend  >  parameter(1)",
  $po/@OrderDate )
where $p//@pid = $item
return
<RESULT>
  <PoNum>{data($po/@PoNum)}</PoNum>
  <PartID>{data($item)} </PartID>
  <PoDate>{data($po/@OrderDate)}</PoDate>
</RESULT>
```

XQuery によるデータ検索

XQuery の仕様では、XQuery 式の結果は 0 または 1 つ以上の項目を含むシーケンスと定義されています。 XQuery 式は、XQuery を 1 次言語として使用するか、XMLQUERY SQL 関数を使用した SQL を 1 次言語として用いて実行できます。どちらかの方法を使用して XQuery 式を実行しても、XML シーケンスが戻されます。

結果セットに結果のシーケンスが現れる方法は、SQL または XQuery のどちらを 1 次言語として使用したかに応じて異なります。

XQuery を 1 次言語とした場合

XQuery を 1 次言語として XQuery 式を実行した場合、結果はタイプ XML の 1 つの列がある結果表としてクライアント・アプリケーションに戻されます。この結果表の各行は、XQuery 式を評価した結果として得られたシーケンスの項目です。アプリケーションがカーソルを使用してこの結果表からフェッチするとき、それぞれのフェッチは、結果のシーケンスからシリアライズされた項目を検索することになります。

XMLQUERY を使用した SQL を 1 次言語とした場合

XMLQUERY は XML 値を戻すスカラー関数です。戻される値は、0 または 1 つ以上の項目のシーケンスとなります。結果シーケンスのすべての項目は、単一のシリアライズされた値としてアプリケーションに戻されます。

XQuery または XMLQUERY を使用する照会から結果をフェッチするには、他の結果セットで通常行うとおりに、アプリケーション内から結果をフェッチします。アプリケーション変数を結果セットにバインドして、結果セットの最後までフェッチします。 XQuery 式 (直接または XMLQUERY を介して発行したもの) が空のシーケンスを戻す場合、結果セット内の行も空です。

照会の結果セットの管理

XQuery を使用して照会するときに戻される XML 値が整形式 XML 文書となることがアプリケーションにより求められるとき (例えば、これらの値をタイプ XML の列に挿入する予定であるとき)、要素または文書のコンストラクターを XQuery 式に含めることによって、値が必ず整形式 XML 文書となるようにすることができます。

例: XQuery および XMLQUERY からの結果セットの相違点

この例は、2 つの照会方法による結果セットの違いを示しています。

例えば、以下の 2 つの XML 文書が XML 列に保管される場合、すべての <phone> 要素を検索するには、XQuery または XMLQUERY のどちらかを使用します。ただし、これら 2 つの方法によって戻される結果セットは異なるので、結果セットからフェッチするときはアプリケーションで適切に処理するようにしてください。

```
<customerinfo Cid="1000">
  <name>Kathy Smith</name>
  <addr country="Canada">
    <street>5 Rosewood</street>
    <city>Toronto</city>
    <prov-state>Ontario</prov-state>
    <pcode-zip>M6W 1E6</pcode-zip>
  </addr>
  <phone type="work">416-555-1358</phone>
</customerinfo>

<customerinfo Cid="1003">
  <name>Robert Shoemaker</name>
    <addr country="Canada">
    <street>1596 Baseline</street>
    <city>Aurora</city>
    <prov-state>Ontario</prov-state>
    <pcode-zip>N8X 7F8</pcode-zip>
  </addr>
  <phone type="work">905-555-7258</phone>
  <phone type="home">416-555-2937</phone>
  <phone type="cell">905-555-8743</phone>
  <phone type="cottage">613-555-3278</phone>
</customerinfo>
```

次の例のように、XQuery を 1 次言語として使用して XQuery 式を実行します。

```
XQUERY
  db2-fn:xmlcolumn ('CUSTOMER.INFO')/customerinfo/phone
```

結果セットは、次のように 5 つの行を返します。

```
<phone type="work">416-555-1358</phone>
<phone type="work">905-555-2937</phone>
<phone type="home">416-555-2937</phone>
<phone type="cell">905-555-8743</phone>
<phone type="cottage">613-555-3278</phone>
```

次の例のように、XMLQUERY を介して XQuery 式を実行します。

```
SELECT XMLQUERY ('$doc/customerinfo/phone' PASSING INFO AS "doc")
FROM CUSTOMER
```

結果セットは、次のように 2 つの行を返します。表に含まれる 2 番目の行のすべての <phone> 要素は、単一のスカラー値 (XML シーケンス) に連結されます。

```
<phone type="work">416-555-1358</phone>
<phone type="work">905-555-2937</phone><phone type="home">416-555-2937</
phone><phone type="cell">905-555-8743</phone><phone type="cottage">613-
555-3278</phone>
```

この結果セットの 2 番目の行には、整形式 XML 文書ではない値が含まれることに注意してください。

結果セットにこれらの相異が生じるのは、XMLQUERY がスカラー関数であるためです。これは表の各行および表の行からの結果シーケンスに対して実行されて、結果セットの行を形成します。それに対して XQuery は、シーケンスの各項目を結果セットの別個の行として戻します。

例: 照会の結果セットの管理

この例では、直前の SQL 照会に変更を加えて、XQuery 文書ノード・コンストラクターを組み込み、結果の行すべてが整形式文書になることを保証することができます。

```
SELECT XMLQUERY ('document{<phonelist>{$doc/customerinfo/phone}</phonelist>}'
  PASSING INFO AS "doc")
FROM CUSTOMER
```

前に示したものと同じ文書がデータベース内に存在すると想定すると、この照会の結果セットは、以下のような 2 つの行を返します (見やすくするために出力は整形されています)。

```
<phonelist><phone type="work">416-555-1358</phone></phonelist>
<phonelist><phone type="work">905-555-7258</phone><phone
type="home">416-555-2937</phone><phone type="cell">905-555-8743</
phone><phone type="cottage">613-555-3278</phone></phonelist>
```

照会に一致する索引のガイドラインの概要

このセクションでは、XML データに対する索引に一致する照会のいくつかのガイドラインと例を示します。

照会で索引を使用できるかどうかは、作成した索引 (複数の場合もある) が照会 (索引突き合わせとも言う) と互換性があるか、およびオプティマイザーが照会評価中に索引スキャンを実行する選択をするかどうかによります。 EXPLAIN 機能のアクセス・プランで、照会評価に索引スキャンが含まれるかが示されます。

照会は、少なくとも次の条件を満たさなければ XML データに対する索引を使用できません。

- 照会検索条件のデータ・タイプが索引付きデータ・タイプと一致する。
- 照会検索条件に索引付けされたノードのサブセットが含まれている。

SQL および XQuery オプティマイザー

オプティマイザーは照会の評価を計画し、評価中にどの索引を使用するかを選択します。照会のコンパイル中、照会は、XML 索引定義内のすべてのパターンと突き合わされ、照会のいくつかの部分に答えるために十分な情報を持つ索引候補を検出します。

オプティマイザーは照会評価中、以下のステップのいずれかを行う場合があります。

- 索引を使用せずに、XML 文書が含まれる表をスキャンする
- リレーショナル索引を使用する
- リレーショナル索引 ANDing または索引 ORing を使用する
- 新規 XML 索引演算子を使用する
- 単一の XML パターンの評価のために XML データに対する索引を使用する
- 単一の照会の複雑な XML パターンを評価するために XML データに対する索引の ANDing および ORing を使用する

Explain 機能

Visual Explain ツールと EXPLAIN 機能は、照会を評価するために選択されるアクセス・プランを提供することができます。アクセス・プランを確認すると、照会評価中に 1 つの索引を使用したかそれとも複数の索引を使用したかが、以下の演算子で分かります。

IXAND
複数の索引スキャンで生成された行 ID を AND 演算する。

XISCAN
XML データに対する索引をスキャンする。

XANDOR
AND 演算された述部を複数の XML 索引に適用できるようにする。

索引定義の厳密さ

照会の評価に索引が使用されるかどうかは、照会と比較する索引定義の厳密さによって異なります。以下に、一緒に使用できる照会と索引の例をいくつか示しています。

範囲述部を持つ照会の索引

以下の照会は、*35000* より高い給料を得ている従業員の会社情報を、XML 列 *companydocs* を持つ表 *companyinfo* から取得します。

```
SELECT companydocs FROM companyinfo
   WHERE XMLEXISTS('$x/company/emp[@salary > 35000]'
   PASSING companydocs AS "x")
```

互換性を持たせるため、 XML データに対する索引は、索引付けするノード間に従業員給料属性ノードを含め、値を DOUBLE または DECIMAL タイプで格納する必要があります。

照会は、以下の XML データに対する索引のいずれかを使用することができます。例を示します。

```
CREATE INDEX empindex on companyinfo(companydocs)
   GENERATE KEY USING XMLPATTERN '//@salary' AS SQL DECIMAL(10,2)

CREATE INDEX empindex on companyinfo(companydocs)
   GENERATE KEY USING XMLPATTERN '/company/emp/@salary'
   AS SQL DECIMAL(10,2)
```

複数の照会が使用できる索引

以下の照会は、ID *31664* を持つ従業員の会社情報を取得します。

```
SELECT companydocs FROM companyinfo
   WHERE XMLEXISTS('$x/company/emp[@id="31664"]'
   PASSING companydocs AS "x")
```

次の照会は、ID *K55* を持つ部門の会社情報を取得します。

```
SELECT companydocs FROM companyinfo
   WHERE XMLEXISTS('$x/company/emp/dept[@id="K55"]
   PASSING companydocs AS "x")
```

両方の照会と互換性を持たせるため、XML データに対する索引は、索引付きノード間に従業員 ID 属性ノードと部門 ID 属性ノードを含める必要があり、値を VARCHAR タイプで索引に格納する必要があります。

照会は次の XML データに対する索引を使用することができます。

```
CREATE INDEX empdeptindex on companyinfo(companydocs)
   GENERATE KEY USING XMLPATTERN '//@id' AS SQL VARCHAR(25)
```

XQuery 述部を制限する際の名前空間の組み込み

顧客情報を含む XML 列を持つ以下の表と、その XML 列に作成された索引について考慮します。

```
CREATE TABLE customer(xmlcol XML) %

CREATE UNIQUE INDEX customer_id_index ON customer(xmlcol)
   GENERATE KEY USING XMLPATTERN
   'DECLARE DEFAULT ELEMENT NAMESPACE
   "http://mynamespace.org/cust";/Customer/@id'
   AS SQL DOUBLE %
```

注: セミコロン (;) はすでに名前空間区切り文字の働きをしているため、このセクションではステートメント終止符にパーセント記号 (%) が使用されています。

次の照会は、索引との突き合わせに失敗します。

```
SELECT xmlcol FROM customer
   WHERE XMLEXISTS('$xmlcol/*:Customer[@id=1042]'
   PASSING xmlcol AS "xmlcol") %
```

照会で索引を使用できるようにするには、照会の厳密さが索引と同じかそれ以上である必要があります。索引 customer_id_index は、特定の 1 つの名前空間 (*http://mynamespace.org/cust*) にある customer 要素のみを扱います。**:* は、照会の中で任意の名前空間を示すために使用されるので、索引は使用されません。これは、**:* が索引定義内の名前空間と一致することを期待している場合には、直感とは異なる場合があります。

照会で索引を使用するには、索引の厳密さがより低くなるか、または照会の厳密さがより高くなる必要があります。

厳密さがより低い次の索引 *customer_id_index2* は、同じ照会で正常に使用できます。

```
CREATE UNIQUE INDEX customer_id_index2 ON customer(xmlcol)
      GENERATE KEY USING XMLPATTERN '/*:Customer/@id' AS SQL DOUBLE %
```

厳密さがより高い次の照会は、初期の索引 *customer_id_index* を使用できます。

```
SELECT xmlcol FROM customer
   WHERE XMLEXISTS('
   DECLARE NAMESPACE ns = "http://mynamespace.org/cust";
   $xmlcol/ns:Customer[@id=1042]'
   PASSING xmlcol AS "xmlcol") %
```

適切な名前空間が照会で明示的に指定されている場合、照会の厳密さは索引と同じになるため、索引 *customer_id_index* を使用できます。索引 *customer_id_index2* も、その厳密さがこの例の照会より低いため、使用できます。

text() ノードを指定する際の考慮事項

XML パターン式に text() ノードを組み込むと、索引項目の生成に影響を与える場合があります。索引定義および述部には /text() を一貫して使用してください。

text() ノードの指定が索引キーに及ぼす影響

次の XML 文書フラグメントのサンプルを考慮します。

```
<company name="Company1">
      <emp id="31201" salary="60000" gender="Female">
     <name><first>Laura</first><last>Brown</last></name>
     <dept id="M25">
   Finance
       </dept>
      </emp>
</company>
```

パターンの末尾に text() を指定して以下の索引が作成された場合、XML 文書フラグメント・サンプル内の *name* 要素にはテキスト自体が含まれていないため、索引項目は挿入されません。テキストは子要素である *first* および *last* のみにあります。

```
CREATE INDEX nameindex on company(companydocs)
      GENERATE KEY USING XMLPATTERN '/company/emp/name/text()' AS SQL
      VARCHAR(30)
```

しかし、パターンの末尾に要素 *name* を指定して次の索引が作成されると、*first* および *last* 子要素のテキストは挿入された索引項目内で連結されます。

```
CREATE INDEX nameindex on company(companydocs)
      GENERATE KEY USING XMLPATTERN '/company/emp/name'
      AS SQL VARCHAR(30)
```

text() ノードがあるかないかは、非リーフ要素の索引項目の生成には影響がありますが、リーフ要素には影響がありません。リーフ要素に索引付けする場合、text() の指定は推奨されていません。 text() を指定すると、索引の突き合わせを正常に行うためには、照会でも text() を使用する必要があります。さらに、スキーマの妥当性検査は要素だけに適用され、テキスト・ノードには適用されません。

text() が含まれない非リーフ・ノードである要素と一致する XML パターンを指定する際、注意が必要です。子孫要素のテキスト・ノードの連結により、予期しない結果が生じる場合があります。特に、XML パターンで *//** を指定すると、おそらく非リーフ要素に索引付けされることになります。

場合によっては VARCHAR を使用する索引に対して、連結が役立ちます。例えば、以下の文書フラグメント内の */title* の索引は、タイトル内の太字書式設定を無視するのに役立つかもしれません。

```
<title>This is a <bold>great</bold> book about XML</title>
```

特定の従業員名を検索するための照会述部は、次のように記述できます。

```
db2-fn:xmlcolumn('COMPANY.COMPANYDOCS')/company/emp[name='LauraBrown']
```

空白は、述部および文書では意味を持ちます。述部で「Laura」と「Brown」の間にスペースが挿入された場合、以下の照会では何も戻されません。XML 文書フラグメントのサンプル自体には、ファーストネームとラストネームの間にはスペースが含まれていないからです。

```
db2-fn:xmlcolumn('COMPANY.COMPANYDOCS')/company/emp[name='Laura Brown']
```

複合等価述部を持つ照会の索引

以下の照会は、Finance または Marketing 部門にいる従業員の会社情報を取得します。

```
SELECT companydocs FROM companyinfo WHERE
   XMLExists('$x/company/emp[dept/text()='Finance'
   or dept/text()='Marketing']'
   PASSING companydocs AS "x")
```

互換性を持たせるため、XML データに対する索引は、索引付きノード間で各従業員の部門のテキスト・ノードを索引付けする必要があり、値を VARCHAR タイプで保管する必要があります。

照会は次の XML データに対する索引を使用することができます。

```
CREATE INDEX empindex on companyinfo(companydocs)
   GENERATE KEY USING XMLPATTERN '/company/emp/dept/text()'
   AS SQL VARCHAR(30)
```

リテラルのデータ・タイプ

リテラルのデータ・タイプが索引のデータ・タイプと一致しなければ、照会でその索引を使用することはできません。

リテラルのデータ・タイプの突き合わせ

次の照会は、ID *31201* を持つ従業員の会社情報を取得します。

```
SELECT companydocs FROM companyinfo
   WHERE XMLEXISTS('$x/company/emp[@id="31201"]'
   PASSING companydocs AS "x")
```

互換性を持たせるため、XML データに対する索引は、索引付きノード間に従業員 ID 属性ノードを含める必要があり、値を VARCHAR タイプで索引に格納する必要があります。

```
CREATE INDEX empindex on companyinfo(companydocs)
   GENERATE KEY USING XMLPATTERN '/company/emp/@id'
   AS SQL VARCHAR(5)
```

類似した索引が AS SQL DOUBLE と定義されている場合、照会述部にストリング比較が含まれているため、これは照会では使用できません。述部 *@id="31201"* で使用されている二重引用符は、これをストリング比較にするので、ストリング索引 (VARCHAR) のみで評価され、数値索引 (DOUBLE) では評価できません。

数値述部とストリング述部の違いを強調するため、次の比較演算子述部を考慮します。

```
@id > 3
@id > "3"
```

数値述部 *@id > 3* は、ストリング述部 *@id > "3"* とは異なります。数値述部 *@id > 3* は、*@id* 値が *10* である場合に満たされますが、ストリング述部 *@id > "3"* は満たされません。なぜならストリング比較では *"3"* は *"10"* より大きいからです。

結合述部の変換

結合述部は、両側で適切なデータ・タイプに変換する必要があります。

索引の使用を不可能にする結合述部

2 つの表があり、一方には顧客情報の XML 列が、他方には購入注文の XML 列が含まれています。

```
CREATE TABLE customer(info XML);

CREATE TABLE PurchaseOrder(POrder XML);
```

顧客情報を含む XML 文書には、属性 @cid、数値カスタマー ID (*cid*) が含まれています。購入注文情報を含む XML 文書にも *@cid* 属性が含まれているので、各オーダーは特定の顧客と一意的に関連付けられます。顧客およびオーダーを *cid* で頻繁に検索することが予想されるため、索引を定義することには意味があります。

```
CREATE UNIQUE INDEX idx1 ON customer(info)
   GENERATE KEY USING XMLPATTERN '/customerinfo/@cid' AS SQL INTEGER;

CREATE INDEX idx2 ON PurchaseOrder(POrder)
   GENERATE KEY USING XMLPATTERN '/porder/@cid' AS SQL INTEGER;
```

特定の郵便番号の全顧客の購入注文を検索するとします。直感的には次のように照会を作成するでしょう。

```
XQUERY
   for $i in db2-fn:xmlcolumn("CUSTOMER.INFO")/customerinfo
   for $j in db2-fn:xmlcolumn("PURCHASEORDER.PORDER")/porder[@cid = $i/@cid]
   where $i/zipcode = "95141"
   return $j;
```

結合述部が @cid = $i/@cid である場合には、購入注文の *cid* が顧客の *cid* と等しくなる必要があることに注意してください。

この照会は正しい結果を戻しますが、2 つの索引のいずれも使用できません。照会は、両方の表の表スキャンが含まれるネストされたループ結合として実行されます。表スキャンの反復を避けるため、*customer* に対して表スキャンを 1 回行い、郵便番号が *95141* であるすべての顧客を検索する方が望ましいでしょう。その後 @cid を使用して購入注文表を索引検索します。なお、*zipcode* には索引がないため、customer 表をスキャンすることが必要です。

索引の使用は誤っているため、使用されません。索引が使用されてしまうと、DB2 は一致する購入注文の一部を失い、不完全な結果を戻す場合があります。これは、@cid 属性の一部の値が非数値となる可能性があるためです。例えば @cid は *YPS* と等しくなる場合があり、その場合には AS SQL INTEGER で定義された数値索引に組み込まれません。

注: 索引付きノードの値が、指定された索引データ・タイプに変換できない場合、その値の索引項目は挿入されず、エラーまたは警告は出されません。

結合述部を用いた索引の使用可能化

すべての @cid 値が数値であることが確かである場合、索引を使用できるようにすることができます。結合述部を明示的に変換して索引のタイプと一致させると、索引が使用されます。

```
XQUERY
   for $i in db2-fn:xmlcolumn("CUSTOMER.INFO")/customerinfo
   for $j in db2-fn:xmlcolumn("PURCHASEORDER.PORDER")/porder
   where $i/@cid/xs:int(.) = $j/@cid/xs:int(.)
   and $i/zipcode = "95141"
   return $j;
```

この変換は、本質的には、 INTEGER に変換可能な @cid 属性のみの突き合わせを考慮するように DB2 に指示しています。この指示に従うことで、必要な突き合わせがすべて AS SQL INTEGER で定義された索引内に確実に表されているため、その索引を使用しても安全です。文書のいずれかに非数値の @cid 値が存在する場合、変換は実行時エラーで失敗します。

なお、XQuery の中では、キャストは個別のものにしか作動しません。特に要素の場合 (以下の例の *a*、*b*、および *c*)、次のように変換することをお勧めします。

```
/a/b/c/xs:double(.)
```

要素を次のように変換すると、いずれかの要素 *b* の下に複数の要素 *c* が存在する場合、実行時エラーが発生します。

```
/a/b/xs:double(c)
```

AS SQL VARCHAR に定義された索引の場合、対応する結合述部は、比較される値を fn:string() 関数を使用して xs:string データ・タイプに変換する

必要があります。同じことが DATE および TIMESTAMP 索引にも当てはまります。次の例は、ストリング結合における fn:string() 関数の使用方法を示しています。

```
XQUERY
   for $i in db2-fn:xmlcolumn("CUSTOMER.INFO")/customerinfo
   for $j in db2-fn:xmlcolumn("PURCHASEORDER.PORDER")/porder
   where $i/zipcode/fn:string(.) = $j/supplier/zip/fn:string(.)
   return <pair>{$i}{$j}</pair>
```

結合述部の変換規則についての要約

以下の表には、どのように結合述部を、索引の使用を可能にするために両側で適切なデータ・タイプに変換する必要があるかについての要約を示しています。

表 16. 結合述部の変換規則

索引 SQL タイプ	結合述部から XML タイプへの変換
DOUBLE	xs:double
DECIMAL	xs:decimal
INTEGER	xs:int
VARCHAR *integer*、VARCHAR HASHED	xs:string
DATE	xs:date
TIMESTAMP	xs:dateTime

不確定の照会評価

索引スキャンが含まれていない場合に照会の評価が不確定になり、エラーが戻る場合があります。照会の評価に索引スキャンが含まれている場合には、エラーの原因となるキャストできない XML フラグメントは索引に含まれないため、同じ照会でもエラーとならずに一致する XML データを戻す場合があります。

この例では、XQuery 式が含まれる以下の VALUES ステートメントを使用します。この式は、数値比較を使用して、ID が 17 の XML 文書を戻します。

```
VALUES( XMLQUERY('
  for $i in db2-fn:xmlcolumn("T.DOC")
   where $i/emp/id = 17
   return $i'))
```

表 T は、単一の XML 列 DOC で構成され、2 つの XML 文書を含みます。以下のステートメントは、表を作成し、文書を挿入します。

```
CREATE TABLE t (doc XML) ;
INSERT INTO t VALUES ( '<emp><id>17</id></emp>' );
INSERT INTO t VALUES ( '<emp><id>ABC</id></emp>' );
```

表の中の一致する文書を検出するために SQL INTEGER または SQL DOUBLE で定義された XML データに対する索引 が使用される場合を除き、 XQuery 式はエラーを返します。以下の CREATE INDEX ステートメントによって索引を作成します。

```
CREATE INDEX EMPDBL ON t (doc) GENERATE KEY USING XMLPATTERN '/emp/id'
    as SQL INTEGER IGNORE INVALID VALUES
```

表に索引が存在しない場合、**where** 文節の数値比較は、表内の一致する文書と一致しない文書の両方に適用されます。一致しない文書に比較が適用された場合、値 ABC は数値に変換できないため、実行時エラーとなります。従業員 ID に索引を付ける XML データに対する索引 が表に存在する場合、式はこの索引を使用し、エラーなしで最初の XML 文書を戻します。 IGNORE INVALID VALUES 節が索引作成中の無効パターン値を無視し、それらの値の索引付けを行わないように指定しているので、2 番目の文書にあるキャストできない値は XML データに対する索引 に含まれません。無効値の無視は、デフォルト・オプションです。

EXPLAIN 機能により提供されるアクセス・プランには、照会の評価に索引スキャンが含まれるかどうかが示されます。

索引作成時にすべてのパターン値が有効であることを確かめるには、REJECT INVALID VALUES 節を使用します。索引の作成中または更新中に無効なパターン値があると、エラーになります。

XML 文書での全文検索

ネイティブに保管された XML データの全文検索は、DB2 Net Search Extender によって可能です。

DB2 Net Search Extender

DB2 Net Search Extender は、XML データ・タイプを完全にサポートし、XML 列に保管された文書に対する全文索引作成機能を備えています。 XML 列にテキスト索引を作成することにより、XML 文書内のすべてのテキストを照会したり、あいまい検索またはワイルドカード検索などの検索を実行できます。 DB2 Net Search Extender は、Linux、UNIX、および Windows 用の DB2 データ・サーバー製品全体の一部ですが、別個にインストールする必要があります。

次の例は、DEPTDOC 列に保管された XML 文書の /dept/description パス内にある単語「marketing」を検索する、簡単な全文検索を示しています。

```
SELECT DEPTDOC
FROM DEPT
WHERE contains (DEPTDOC, SECTIONS("/dept/description") "marketing") = 1
```

DB2 Net Search Extender が提供する contains 関数は、ストリング「marketing」をパス /dept/description の下のすべてのテキスト内 (要素または属性名および要素または属性値を含む) で検索します。

全文検索を使用するには、SQL を使用する必要があります。ただし、SQL 照会からの結果を XQuery コンテキストに戻してさらに処理することもできます。次の例は、全文検索を使用する SQL 照会からの結果を XQuery 式で使用する方法を示しています。

```
XQUERY for $i in db2-fn:sqlquery ('SELECT DEPTDOC FROM DEPT
  WHERE contains
    (DEPTDOC, SECTIONS("/dept/description") "marketing") = 1')//employee
  return $i/name
```

この例では、全文検索を活用した SQL 照会の結果が XQuery FLWOR 式の for 節に戻されます。その後、for 節はすべての <employee> 要素を戻し、return 節は検索された <employee> 要素内の <name> 要素を戻します。

DB2 Net Search Extender について詳しくは、DB2 Net Search Extender のマニュアル、または Web の www.ibm.com/software/data/db2/extenders/netsearch を参照してください。

以前の DB2 クライアントへの XML 列のデータの取り込み

XML 列のデータを DB2 バージョン 9.1 より前のリリースのクライアントに取り込む場合、データベース・クライアントは XML データを処理できません。

DRDA® 処理中、XML データをサポートできないクライアントをデータベース・サーバーが認識すると、デフォルトでは、DB2 データベース・サーバーは XML データ値を BLOB 値として記述し、そのデータを BLOB データとしてクライアントに送信します。 BLOB データは、完全な XML 宣言を持つシリアライズされた XML データのストリング表記です。

BLOB データ・タイプ以外のデータ・タイプでデータを受け取る場合、以下のいずれかの方法を使用します。

- データを CLOB データとして取り出す場合、db2set コマンドを使用してサーバー上で DB2_MAP_XML_AS_CLOB_FOR_DLC レジストリー変数を YES に設定するように、データベース・サーバーの管理者に依頼します。

  重要: データベース・サーバー上で DB2_MAP_XML_AS_CLOB_FOR_DLC レジストリー変数を YES に設定すると、そのインスタンス内のいずれかのデータベースに接続する以前のリリース・レベルの**すべての** DB2 クライアントは、XML データを CLOB データとして受け取ります。

  重要: データベース・サーバー上で DB2_MAP_XML_AS_CLOB_FOR_DLC レジストリー変数が YES に設定されていると、クライアントは、XML 宣言が**ない**、XML データのシリアライズされたストリング表記である CLOB データを受け取ります。
- データを CLOB、CHAR、または VARCHAR データで取り出すには、列データに対して XMLSERIALIZE 関数を呼び出して、クライアントに送信する前にデータを指定するデータ・タイプに変換するように DB2 データベース・サーバーに指示します。

データベース・サーバーから以前のリリース・レベルのクライアントにデータを取り込む際に XMLSERIALIZE を呼び出さない場合、データを取り出す列は、BLOB または CLOB 列とまったく同様に動作するわけではありません。例えば、BLOB 列では LIKE 述部を使用できますが、BLOB または CLOB データを戻す XML 列上では LIKE 述部は使用できません。

XML 値を構成するための SQL/XML 発行関数

結果として得られる XML 値の構成要素に対応した発行関数を組み合せることにより、XML 値 (必ずしも整形式 XML 文書である必要はない) を構成できます。結果の順序と同じ順序で関数を指定する必要があります。

SQL/XML 発行関数を使用して作成された値は、XML として戻されます。 XML 値の使用目的によっては、その値を明示的にシリアライズして他の SQL データ・タイプに変換する必要がある場合があります。詳しくは、XML のシリアライゼーションに関する資料を参照してください。

以下の SQL/XML 発行関数を使用して、XML 値を構成できます。各関数の構文に関する説明は、481 ページの『付録 B. SQL/XML 発行関数』を参照してください。

XMLAGG 集約関数
XML 値のセット中、NULL 以外の値ごとに 1 つの項目を含めた、XML シーケンスを戻します。

XMLATTRIBUTES スカラー関数
引数から XML 属性を作成します。この関数は、XMLELEMENT 関数の引数としてのみ使用できます。

XMLCOMMENT スカラー関数
入力引数の内容を含む、単一の XQuery コメント・ノードを持った XML 値を戻します。

XMLCONCAT スカラー関数
さまざまな数の XML 入力引数を連結したシーケンスを戻します。

XMLDOCUMENT スカラー関数
ゼロ個以上の下位ノードを持った単一の XQuery 文書ノードを持つ XML 値を戻します。この関数は文書ノードを作成します。これは、定義上、すべての XML 文書が持つ必要のあるものです。文書ノードは XML のシリアライズされた表記では不可視ですが、DB2 表に保管されるすべての文書は文書ノードを含んでいなければなりません。

XMLELEMENT スカラー関数
XML 要素ノードである XML 値を戻します。XMLELEMENT 関数が作成するのは、要素ノードだけで、文書ノードではないことに注意してください。挿入されることになる XML 文書を作成するとき、要素ノードを作成するだけでは不十分です。文書は、XMLDOCUMENT 関数で作成された文書ノードを含んでいる必要があります。

XMLFOREST スカラー関数
XML 要素ノードのシーケンスである XML 値を戻します。

XMLGROUP 集約関数
表を表す、または照会の結果を表す単一の最上位要素を戻します。デフォルトでは、結果セット内の各行は行サブ要素にマップされ、それぞれの入力式は行サブ要素のサブ要素にマップされます。オプションとして、結果内の各行を行サブ要素にマップし、それぞれの入力式を行サブ要素の属性にマップすることもできます。

XMLNAMESPACES 宣言
引数から名前空間宣言を作成します。この宣言は、XMLELEMENT、XMLFOREST、および XMLTABLE 関数の引数としてのみ使用できます。

XMLPI スカラー関数
単一の XQuery 処理命令ノードを持った XML 値を戻します。

XMLROW スカラー関数
表を表す、または照会の結果を表す行要素のシーケンスを戻します。デフォルトでは、各入力式は行要素のサブ要素に変換されます。オプションとして、各入力式を行要素の属性に変換することもできます。

XMLTEXT スカラー関数
入力引数の内容を含む、単一の XQuery テキスト・ノードを持った XML 値を戻します。

XSLTRANSFORM スカラー関数
XML データを、他の XML スキーマを含む他の形式に変換します。

Null 要素値

XMLELEMENT または XMLFOREST を使用して XML 値が構成されるとき、要素の内容を判別する際に NULL 値が検出される可能性があります。 XMLELEMENT および XMLFOREST の EMPTY ON NULL および NULL ON NULL オプションによって、要素の内容が NULL のときに空の要素を生成するかまたは要素を生成しないかを指定できます。 XMLEXISTS のデフォルトの NULL 処理は、EMPTY ON NULL です。 XMLFOREST のデフォルトの NULL 処理は、NULL ON NULL です。

XML 値の発行の例

以下の例は、SQL/XML 発行関数および XQuery 式を使用して XML 値を構成する方法を示しています。

例: 定数値による XML 文書の構成

この簡単な例は、SQL/XML 発行関数を使用した発行に適した XML 定数値を構成する方法を示しています。

簡単な例として、次の XML 要素を考えてみます。

```
<elem1 xmlns="http://posample.org" id="111">
  <!-- example document -->
  <child1>abc</child1>
  <child2>def</child2>
</elem1>
```

文書は、以下から構成されます。

- 3 つの要素ノード (elem1、child1、および child2)
- 名前空間宣言
- <elem1> 上の "id" 属性
- コメント・ノード

この文書を構成するには、以下のステップを実行します。

1. XMLELEMENT を使用して、"elem1" という名前の要素ノードを作成します。
2. XMLNAMESPACES を使用して、デフォルトの名前空間宣言を <elem1> の XMLELEMENT 関数呼び出しに追加します。
3. XMLATTRIBUTES を使用して "id" という名前の属性を作成し、それを XMLNAMESPACES 宣言の後に置きます。

4. XMLCOMMENT を使用して <elem1> の XMLELEMENT 関数呼び出しの中にコメント・ノードを作成します。
5. XMLFOREST 関数を使用して "child1" および "child2" という名前の要素のフォレストを <elem1> の XMLELEMENT 関数呼び出しの中に作成します。

上記のステップを結合すると、次の照会になります。

```
VALUES XMLELEMENT (NAME "elem1",
                   XMLNAMESPACES (DEFAULT 'http://posample.org'),
                   XMLATTRIBUTES ('111' AS "id"),
                   XMLCOMMENT ('example document'),
                   XMLFOREST('abc' as "child1",
                             'def' as "child2"))
```

例: 単一の表からの値による XML 文書の作成

この例は、単一の表からの SQL/XML 発行関数を使用した発行に適した XML 値を構成する方法を示しています。

この例は、単一の表に保管された値から XML 文書を構成する方法を示しています。次の照会では、XMLELEMENT 関数によって、各 <item> 要素が PRODUCT 表の name 列の値を使って構成されます。その後、XMLAGG によってすべての <item> 要素が集約され、構成された <allProducts> 要素の中にまとめられます。

```
SELECT XMLELEMENT (NAME "allProducts",
                   XMLAGG(XMLELEMENT (NAME "item", p.name)))
FROM Product p
```

```
<allProducts>
  <item>Snow Shovel, Basic 22 inch</item>
  <item>Snow Shovel, Deluxe 24 inch</item>
  <item>Snow Shovel, Super Deluxe 26 inch</item>
  <item>Ice Scraper, Windshield 4 inch</item>
</allProducts>
```

行要素のシーケンスを含む同様の XML 文書を、XMLAGG で要素を集約する代わりに、XMLROW 関数を使用することにより構成できます。

```
SELECT XMLELEMENT (NAME "products",
                   XMLROW(NAME as "po:item"))
FROM Product
```

結果出力は以下のようになります。

```
<products>
   <row>
      <po:item>Snow Shovel, Basic 22 inch</po:item>
   </row>
</products>
<products>
   <row>
      <po:item>Snow Shovel, Deluxe 24 inch</po:item>
   </row>
</products>
<products>
      <row><po:item>Snow Shovel, Super Deluxe 26 inch</po:item>
   </row>
</products>
<products>
      <row><po:item>Ice Scraper, Windshield 4 inch</po:item>
   </row>
</products>

  4 record(s) selected.
```

例: 複数の表からの値による XML 文書の作成

この例は、複数の表からの SQL/XML 発行関数を使用した発行に適した XML 値を構成する方法を示しています。

この例は、複数の表に保管された値から XML 文書を構成する方法を示しています。次の照会では、XMLFOREST 関数によって、<prod> 要素が要素 (name および numInStock) のフォレストから構成されます。このフォレストは、PRODUCT および INVENTORY 表の値から作成されます。その後、すべての <prod> 要素は構成された <saleProducts> 要素に集約されます。

```
SELECT XMLELEMENT (NAME "saleProducts",
                   XMLAGG (XMLELEMENT (NAME "prod",
                           XMLATTRIBUTES (p.Pid AS "id"),
                           XMLFOREST (p.name as "name",
                                      i.quantity as "numInStock"))))
FROM PRODUCT p, INVENTORY i
WHERE p.Pid = i.Pid
```

直前の照会は、次の XML 文書を生成します。

```
<saleProducts>
  <prod id="100-100-01">
    <name>Snow Shovel, Basic 22 inch</name>
    <numInStock>5</numInStock>
  </prod>
  <prod id="100-101-01">
    <name>Snow Shovel, Deluxe 24 inch</name>
    <numInStock>25</numInStock>
  </prod>
  <prod id="100-103-01">
    <name>Snow Shovel, Super Deluxe 26 inch</name>
    <numInStock>55</numInStock>
  </prod>
  <prod id="100-201-01">
    <name>Ice Scraper, Windshield 4 inch</name>
    <numInStock>99</numInStock>
  </prod>
</saleProducts>
```

例: NULL 要素を含む表の行からの値による XML 文書の作成

この例は、NULL 要素を含む表の行からの SQL/XML 発行関数を使用した発行に適した XML 値を構成する方法を示しています。

この例では、INVENTORY 表の LOCATION 列で 1 つの行に NULL 値が含まれると想定します。そのため、次の照会は <loc> 要素を戻しません。XMLFOREST はデフォルトで NULL を NULL として扱うためです。

```
SELECT XMLELEMENT (NAME "newElem",
                   XMLATTRIBUTES (PID AS "prodID"),
                   XMLFOREST (QUANTITY as "quantity",
                              LOCATION as "loc"))
FROM INVENTORY
```

```
<newElem prodID="100-100-01"><quantity>5</quantity></newElem>
```

同じ照会で EMPTY ON NULL オプションが指定されていると、空の <loc> 要素が戻されます。

```
SELECT XMLELEMENT (NAME "newElem",
                   XMLATTRIBUTES (PID AS "prodID"),
                   XMLFOREST (QUANTITY as "quantity",
                              LOCATION as "loc" OPTION EMPTY ON NULL))
FROM INVENTORY
```

```
<newElem prodID="100-100-01"><quantity>5</quantity><loc /></newElem>
```

例: XQuery を使用したデータの発行

この例は、SQL/XML 発行関数を使用した発行だけでなく、XQuery 式を使用した発行にも適した XML 値を構成する方法を示しています。

以下の XQuery 式は、delete 更新式を使用して簡単な顧客リストを作成します。この式は、カスタマー情報からカスタマー ID、アドレス、アシスタント情報、および職場以外の電話番号を除去して、country 属性を address ノード要素から customerinfo ノード要素へ移動します。

```
xquery
<phonelist>
  {for $d in db2-fn:xmlcolumn("CUSTOMER.INFO")/customerinfo
  return
    transform
    copy $mycust := $d
    modify (
      do delete ( $mycust/@Cid ,
        $mycust/addr ,
        $mycust/assistant ,
        $mycust/phone[@type!="work"] ),
      do insert attribute country { $mycust/addr/@country } into $mycust )
    return $mycust }
</phonelist>
```

address 要素が削除された場合でも、アドレス情報は **modify** 節の中でアクセス可能であり、address 要素の country 属性が挿入式で使用されます。

照会は以下の結果を返します。

```
<phonelist>
 <customerinfo country="Canada">
  <name>Kathy Smith</name>
  <phone type="work">416-555-1358</phone>
 </customerinfo>
 <customerinfo country="Canada">
  <name>Kathy Smith</name>
  <phone type="work">905-555-7258</phone>
 </customerinfo>
 <customerinfo country="Canada">
  <name>Jim Noodle</name>
  <phone type="work">905-555-7258</phone>
 </customerinfo>
 <customerinfo country="Canada">
  <name>Robert Shoemaker</name>
  <phone type="work">905-555-7258</phone>
 </customerinfo>
 <customerinfo country="Canada">
  <name>Matt Foreman</name>
  <phone type="work">905-555-4789</phone>
 </customerinfo>
 <customerinfo country="Canada">
  <name>Larry Menard</name>
  <phone type="work">905-555-9146</phone>
 </customerinfo>
</phonelist>
```

SQL/XML 発行関数における特殊文字の処理

SQL/XML 発行関数には、特殊文字の処理におけるデフォルトの動作があります。

SQL 値から XML 値へ

特定の文字は XML 文書内で特殊文字と見なされ、そのエンティティー表記を使用してエスケープ形式で記述する必要があります。これらの特殊文字を以下に示します。

表 17. 特殊文字とそのエンティティー表記

特殊文字	エンティティーの表記
<	<
>	>
&	&
"	"

SQL/XML 発行関数を使用して SQL 値を XML 値として発行する場合、これらの特殊文字はエスケープされてその事前定義エンティティーに置換されます。

SQL ID と QName

SQL 値から XML 値を発行または作成する際、SQL ID から XML 修飾名または QName へのマップが必要になる場合があります。ただし、区切り文字付き SQL ID で許可される文字セットは、QName で許可される文字セットとは異なります。この違いは、SQL ID で使用される文字の一部は QName では無効であるということを意味します。そのため、これらの文字は、QName ではエンティティー表記に置換されます。

例えば、区切り文字付き SQL ID の「phone@work」を考えてみます。 @ 文字は QName では有効文字ではないため、文字はエスケープされ、QName は phone@work になります。

このデフォルトのエスケープ動作は列名のみに適用される点に注意してください。XMLELEMENT で要素名として指定される SQL ID、または XMLFOREST および XMLATTRIBUTES の AS 節で別名として指定される SQL ID の場合、エスケープのデフォルトはありません。これらの場合には、有効な QName を指定する必要があります。有効な名前の詳細については、W3C XML namespace specifications を参照してください。

XML シリアライゼーション

XML シリアライゼーションは、XML データを XQuery および XPath データ・モデルの表現 (DB2 データベース内での階層形式) から、アプリケーション内でのシリアライズされたストリング形式に変換する処理です。

DB2 データベース・マネージャーを使用して暗黙的にシリアライゼーションを行うことも、XMLSERIALIZE 関数を呼び出して XML シリアライゼーションを明示的に要求することもできます。 XML シリアライゼーションが最も一般的に使用されるのは、XML データがデータベース・サーバーからクライアントに送られるときです。

暗黙的なシリアライゼーションはコーディングがより容易なので、ほとんどの場合により好ましい方法となります。XML データをクライアントに送ることにより、DB2 クライアントは XML データを適切に処理できるようになります。明示的なシリアライゼーションでは追加の処理が必要となります。この処理は、暗黙的なシリアライゼーションではクライアントによって自動的に処理されます。

一般には、データを XML データとしてクライアントに送る方が効率が良いので、暗黙的なシリアライゼーションがより好ましい方法です。ただし、(後ほど説明する)特定の状況では、明示的な XMLSERIALIZE を行う方が優れています。

XML データの変換先として最適なデータ・タイプは BLOB データ・タイプです。バイナリー・データを検索するとエンコード方式に関する問題が少なくなるためです。

暗黙的な XML シリアライゼーション

暗黙的なシリアライゼーションでは、クライアントが XML データ・タイプをサポートする場合、データはクライアントに送られるときに XML タイプになります。CLI および組み込み SQL アプリケーションでは、DB2 データベース・サーバーが XML 宣言を適切なエンコード方式の指定と共にデータに追加します。 Java および .NET アプリケーションでは、DB2 データベース・サーバーは XML 宣言を追加しません。ただし、データを取り出して `DB2Xml` オブジェクトに入れてから、何らかの方法を使用してデータをそのオブジェクトから取り出す場合、IBM Data Server Driver for JDBC and SQLJ が XML 宣言を追加します。

例: C プログラムで、カスタマー ID「1000」の customerinfo 文書を暗黙的にシリアライズして、シリアライズされた文書を取り出してバイナリー XML ホスト変数に入れます。取り出されたデータは UTF-8 コード化スキームによるもので、XML 宣言を含んでいます。

```
EXEC SQL BEGIN DECLARE SECTION;
 SQL TYPE IS XML AS BLOB (1M) xmlCustInfo;
EXEC SQL END DECLARE SECTION;
...
EXEC SQL SELECT INFO INTO :xmlCustInfo
  FROM Customer
  WHERE Cid=1000;
```

明示的な XML シリアライゼーション

XMLSERIALIZE を明示的に呼び出した後に、データはデータベース・サーバー内で非 XML データ・タイプとなり、そのデータ・タイプでクライアントに送られます。

XMLSERIALIZE では、以下を指定できます。

- データがシリアライズされるときの、変換後の SQL データ・タイプ

  データ・タイプは、文字またはバイナリー・データ・タイプです。
- 出力データに次の明示的なエンコード方式指定が含まれるかどうか (EXCLUDING XMLDECLARATION または INCLUDING XMLDECLARATION)

  ```
  <?xml version="1.0" encoding="UTF-8"?>
  ```

XMLSERIALIZE からの出力は、Unicode UTF-8 でエンコードされたデータとなります。

シリアライズされたデータを取り出してバイナリーではないデータ・タイプにする場合、データはアプリケーションのエンコード方式に変換されますが、エンコード方式の指定は変更されません。そのため、データのエンコード方式はエンコード方式の指定と一致しない可能性が大きくなります。この場合、エンコード方式の名前に依存しているアプリケーション・プロセスで XML データを構文解析できなくなります。

一般には、データを XML データとしてクライアントに送る方が効率が良いので、暗黙的なシリアライゼーションがより好ましい方法です。ただし、以下の状況では、明示的な XMLSERIALIZE を行う方が優れています。

- XML 文書が非常に大きいとき

  XML ロケーターがないため、XML 文書が非常に大きいときは、XMLSERIALIZE を使用してデータを LOB タイプに変換し、LOB ロケーターを使用できるようにする必要があります。
- クライアントが XML データをサポートしないとき

  クライアントが XML データ・タイプをサポートしない、以前のバージョンである場合、暗黙的な XML シリアライゼーションを使用するときは、 DB2 データベース・サーバーはデータをクライアントに送る前にそのデータを以下のデータ・タイプの 1 つに変換します。
  - デフォルトで、BLOB データ・タイプ
  - サーバー上で db2set コマンドを使用して DB2_MAP_XML_AS_CLOB_FOR_DLC レジストリー変数を YES に設定した場合、CLOB データ・タイプ

  取り出したデータを別のデータ・タイプにする場合、XMLSERIALIZE を実行できます。

例: サンプル表 Customer 内の XML 列 Info に、以下のデータと階層的に同等のものが入った文書が入っています。

```
<customerinfo Cid='1000'>
  <name>Kathy Smith</name>
  <addr country='Canada'>
  <street>5 Rosewood</street>
  <city>Toronto</city>
  <prov-state>Ontario</prov-state>
  <pcode-zip>M6W 1E6</pcode-zip>
  </addr>
  <phone type='work'>416-555-1358</phone>
</customerinfo>
```

データを取り出してホスト変数に入れる前に、XMLSERIALIZE を呼び出してデータをシリアライズし、BLOB タイプに変換します。

```
SELECT XMLSERIALIZE(Info as BLOB(1M)) from Customer
  WHERE CID=1000
```

XSLT スタイルシートを使用した変換

XML データを他の形式に変換するための標準的な方法は、XSLT (Extensible Stylesheet Language Transformations) を使用することです。組み込み XSLTRANSFORM 関数を使用すると、XML 文書を HTML、プレーン・テキスト、または別の XML スキーマに変換できます。

XSLT はスタイルシートを使用して、XML を他のデータ形式に変換します。XPath 照会言語および XSLT の組み込み関数を使用すると、XML 文書の一部または全部の変換、およびデータの選択や再配置を行えます。通常 XSLT は XML から HTML への変換に使用されますが、ある XML スキーマに準拠する XML 文書を別のスキーマに準拠する文書に変換するのにも使用できます。また XSLT は XML データを関連のない形式 (コンマ区切りテキストや、troff などの整形言語等) に変換する際にも使用できます。XSLT には、主に 2 つの適用可能な分野があります。

- フォーマット (XML の HTML または FOP などの整形言語への変換);
- データ変換 (ある XML スキーマから別のスキーマへ、または SOAP などのデータ交換形式へのデータの照会、再編成、および変換)。

どちらの場合にも、XML 文書全体、または選択部分のみを変換する必要がある場合があります。XSLT は XPath 仕様を取り入れていて、ソース XML 文書からの任意のデータの照会と取り出しが可能です。また XSLT テンプレートは、ファイル・ヘッダーおよび命令ブロックなどの追加情報を含んでいるか、それらの情報を作成する場合があります。こうした情報は、出力ファイルに追加されます。

XSLT の動作方法

XSLT スタイルシートは、XSL (Extensible Stylesheet Language) で作成されている XML スキーマです。XSL は、C または Perl などのアルゴリズム言語ではなくテンプレート言語で、この特徴は XSL の能力を制限するものではありますが、この言語の目的にまさに適しています。 XSL スタイルシートには、1 つ以上の `template` 要素が含まれています。これは、指定の XML 要素または照会をターゲット・ファイルで検出した場合に実行するアクションについて説明しています。標準的な XSLT テンプレート要素では、適用先の要素の指定から開始されます。例えば、次のようになります。

```
<xsl:template match="product">
```

これは、このテンプレートの内容が、ターゲットの XML ファイルで検出される任意の `<product>` タグの内容を置き換えるのに使用されることを宣言します。 XSLT ファイルはそのようなテンプレートのリストで構成されており、順序は問いません。

以下の例は、XSLT テンプレートの典型的な要素を示しています。この場合、アイス・スクレーパーについて説明したレコードなど、品目情報が含まれる XML 文書がターゲットです。

```
<?xml version="1.0"?>
<product pid="100-201-01">
   <description>
      <name>Ice Scraper, Windshield 4 inch</name>
      <details>Basic Ice Scraper 4 inches wide, foam handle</details>
      <price>3.99</price>
   </description>
</product>
```

このレコードには、フロント・ガラスのアイス・スクレーパーの部品番号、説明、価格などの情報が含まれています。こうした情報の一部は、<name> などの要素内に入れられています。部品番号などのように属性内に組み込まれているものもあります (この場合の <product> 要素の pid 属性)。この情報を Web ページとして表示するには、以下の XSLT テンプレートを適用できます。

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="html"/>
   <xsl:template match="/">
       <html>
            <body>
                <h1><xsl:value-of select="/product/description/name"/></h1>
                <table border="1">
                        <th>
       <xsl:apply-templates select="product"/>
                        </th>
                    </table>
            </body>
       </html>
   </xsl:template>
     <xsl:template match="product">
        <tr>
                    <td width="80">product ID</td>
                    <td><xsl:value-of select="@pid"/></td>
              </tr>
              <tr>
                    <td width="200">product name</td>
                    <td><xsl:value-of select="/product/description/name"/></td>
              </tr>
              <tr>
                    <td width="200">price</td>
                    <td>$<xsl:value-of select="/product/description/price"/></td>
              </tr>
              <tr>
                    <td width="50">details</td>
                    <td><xsl:value-of select="/product/description/details"/></td>
              </tr>
  </xsl:template>
</xsl:stylesheet>
```

XSLT プロセッサーは、前に示したテンプレートとターゲット文書の両方を入力として受信し、以下の HTML 文書を出力します。

```
<html>
<body>
<h1>Ice Scraper, Windshield 4 inch</h1>
<table border="1">
<th>
<tr>
<td width="80">product ID</td><td>100-201-01</td>
</tr>
<tr>
<td width="200">product name</td><td>Ice Scraper, Windshield 4 inch</td>
</tr>
<tr>
<td width="200">price</td><td>$3.99</td>
</tr>
<tr>
<td width="50">details</td><td>Basic Ice Scraper 4 inches wide, foam handle</td>
</tr>
```

```
</th>
</table>
</body>
</html>
```

XSLT プロセッサーは、指定の条件に関して着信 XML 文書を検査します (通常、1 つのテンプレートについて 1 つの条件)。条件が真の場合にはテンプレート・コンテンツは出力に挿入され、偽の場合にはプロセッサーはそのテンプレートを無視します。スタイルシートは出力に独自のデータを追加することもできます。例えば、HTML 表におけるタグや "product ID." などのストリングです。

XPath を使用して、テンプレート条件を定義したり (例 `<xsl:template match="product">`)、XML ストリームの任意の場所にあるデータの選択と挿入を行ったり (例 `<h1><xsl:value-of select="/product/description/name"/></h1>`) することができます。

XSLTRANSFORM の使用

XSLTRANSFORM 関数を使用して、XSLT スタイルシートを XML データに適用できます。この関数に XML 文書の名前と XSLT スタイルシートを提供すると、この関数によってその XML 文書にスタイルシートが適用されて、結果が戻されます。

XSLT スタイルシート内で XSLT の document 関数を指定する場合は、**DB2_XSLT_ALLOWED_PATH** レジストリー変数を、ダウンロードする追加の XML 文書が入っているディレクトリーに設定してください。

ランタイムにおける XSLT スタイルシートへのパラメーターの引き渡し

XML 文書を変換するために組み込み XSLTRANSFORM 関数を使用する際、パラメーターをランタイムで渡すことができます。

XSLTRANSFORM 関数の重要なフィーチャーの 1 つは、ランタイムに XSLT パラメーターを受け入れることができる点です。この機能がないと、(XML データに対する 1 つの照会のそれぞれが変形である) 複数の XSLT スタイルシートで構成される大規模なライブラリーを維持する必要があります。あるいは、使用しているスタイルシートを新しい照会に合わせてそれぞれ手動で編集する必要が生じます。パラメーターの引き渡しにより、変更せずに使用できる汎用スタイルシートを設計して、ライブラリーのパラメーター・ファイルを累積したり、作業中にそうしたファイルを作成したりすることも可能です。

XSLT パラメーターは別個の XML 文書に含めます。例えば、以下のようになります。

```
<?xml version="1.0"?>
<params xmlns="http://www.ibm.com/XSLTransformParameters">
        <param name="headline">BIG BAZAAR super market</param>
        <param name="supermarketname" value="true"/>
</params>
```

それぞれの `<param>` 要素はパラメーターに名前を付け、`value` 属性内 (または値が長い場合には要素自体の中) にその値が入ります。上記の例では、その両方が示されています。

XSLT テンプレート・ファイルで許可されるパラメーターは、以下のように <xsl:param> 要素を使用して変数として定義します。

```
<xsl:param name="headline"/>
<xsl:param name="supermarketname"/>
```

この例では、$headline 変数または $supermarketname 変数をスタイルシート内の任意の場所に呼び出すことできます。これらの変数には、パラメーター・ファイルで定義されたデータが入れられます (この例では、それぞれストリング "BIG BAZAAR super market" と値 "true")。

例: フォーマット設定エンジンとしての XSLT の使用

以下は、組み込み XSLTRANSFORM 関数をフォーマット設定エンジンとして使用する方法を示している例です。

この例は、XSLT をフォーマット設定エンジンとして使用する方法を示しています。セットアップのために、まず以下の 2 つの例の文書をデータベースに挿入します。

```
INSERT INTO XML_TAB VALUES
(1,
    '<?xml version="1.0"?>
<students xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation = "/home/steffen/xsd/xslt.xsd">
<student studentID="1" firstName="Steffen" lastName="Siegmund"
    age="23" university="Rostock"/>
</students>',
    '<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
           xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:param name="headline"/>
<xsl:param name="showUniversity"/>
<xsl:template match="students">
     <html>
             <head/>
                <body>
                 <h1><xsl:value-of select="$headline"/></h1>
                 <table border="1">
                   <th>
                       <tr>
                               <td width="80">StudentID</td>
                               <td width="200">First Name</td>
                               <td width="200">Last Name</td>
                               <td width="50">Age</td>
                               <xsl:choose>
       <xsl:when test="$showUniversity ='true'">
                                  <td width="200">University</td>
                                   </xsl:when>
       </xsl:choose>
                           </tr>
                         </th>
                         <xsl:apply-templates/>
                    </table>
                   </body>
                 </html>
      </xsl:template>
       <xsl:template match="student">
             <tr>
                 <td><xsl:value-of select="@studentID"/></td>
                 <td><xsl:value-of select="@firstName"/></td>
                 <td><xsl:value-of select="@lastName"/></td>
                 <td><xsl:value-of select="@age"/></td>
                 <xsl:choose>
```

```
                    <xsl:when test="$showUniversity = 'true' ">
                      <td><xsl:value-of select="@university"/></td>
                    </xsl:when>
                  </xsl:choose>
       </tr>
        </xsl:template>
</xsl:stylesheet>'
);
```

次に、XSLTRANSFORM 関数を呼び出して XML データを HTML に変換して表示します。

```
SELECT XSLTRANSFORM (XML_DOC USING XSL_DOC AS CLOB(1M)) FROM XML_TAB;
```

以下が、その結果の文書になります。

```
<html>
<head>
<META http-equiv="Content-Type" content="text/html; charset=UTF-8">
</head>
<body>
<h1></h1>
<table border="1">
<th>
<tr>
<td width="80">StudentID</td>
<td width="200">First Name</td>
<td width="200">Last Name</td>
<td width="50">Age</td>
</tr>
</th>
 <tr>
<td>1</td>
<td>Steffen</td><td>Siegmund</td>
<td>23</td>
</tr>
  </table>
</body>
</html>
```

この例では、出力は HTML で、パラメーターは、生成される HTML と、その HTML に渡されるデータにのみ影響を与えます。したがってこの例は、エンド・ユーザー出力用フォーマット設定エンジンとしての XSLT の使用について示しています。

例: データ交換での XSLT の使用

以下は、組み込み XSLTRANSFORM 関数を使用して、データ交換のために XML 文書を変換する方法を示している例です。

この例では、スタイルシートでパラメーターを用いてデータ交換に XSLT を使用し、ランタイムに異なるデータ交換形式を生成する方法を示しています。

xsl:param 要素を取り込むスタイルシートを使用して、パラメーター・ファイルからデータをキャプチャーします。

```
INSERT INTO Display_productdetails values(1, '<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
                    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:param name="headline"/>
<xsl:param name="supermarketname"/>
<xsl:template match="product">
   <html>
```

```
            <head/>
               <body>
               <h1><xsl:value-of select="$headline"/></h1>
               <table border="1">
                       <th>
                       <tr>
                        <td width="80">product ID</td>
                        <td width="200">product name</td>
                        <td width="200">price</td>
                        <td width="50">details</td>
                        <xsl:choose>
   <xsl:when test="$supermarket =''true'' ">
                                <td width="200">BIG BAZAAR super market</td>
                               </xsl:when>
    </xsl:choose>
                        </tr>
                       </th>
                       <xsl:apply-templates/>
               </table>
               </body>
               </html>
       </xsl:template>
      <xsl:template match="product">
             <tr>
                   <td><xsl:value-of select="@pid"/></td>
                   <td><xsl:value-of select="/product/description/name"/></td>
                   <td><xsl:value-of select="/product/description/price"/></td>
                   <td><xsl:value-of select="/product/description/details"/></td>
                  </tr>
     </xsl:template>
  </xsl:stylesheet>'
);
```

このパラメーター・ファイルには、XSLT テンプレート内のパラメーターに対応するパラメーターが含まれています。その内容は、以下のとおりです。

```
CREATE TABLE PARAM_TAB (DOCID INTEGER, PARAM VARCHAR (10K));

INSERT INTO PARAM_TAB VALUES
(1,
'<?xml version="1.0"?>
<params xmlns="http://www.ibm.com/XSLTransformParameters">
        <param name="supermarketname" value="true"/>
        <param name="headline">BIG BAZAAR super market</param>
</params>'
);
```

次に、以下のコマンドを使用してランタイムにこのパラメーター・ファイルを適用できます。

```
SELECT XSLTRANSFORM (XML_DOC USING XSL_DOC WITH PARAM AS CLOB (1M))
   FROM product_details X, PARM_TAB P WHERE X.DOCID=P.DOCID;
```

結果は HTML ですが、そのコンテンツはこのパラメーター・ファイルと、XML 文書のコンテンツに対して行われた検査によって決定します。

```
<html>
<head>
<META http-equiv="Content-Type" content="text/html; charset=UTF-8">
</head>
<body>
<h1></h1>
<table border="1">
<th>
<tr>
<td width="80">product ID</td>
```

```
<td width="200">product Name</td>
<td width="200">price</td>
<td width="50">Details</td>
</tr>
</th>
</table>
</body>
</html>
```

他のアプリケーションでは、XSLTRANSFORM の出力は HTML ではなく他の XML 文書、または EDI ファイルなどの別のデータ形式を使用したファイルとなる場合があります。

データ交換アプリケーションの場合、パラメーター・ファイルには、E メール・アドレス、ポート・アドレス、または特定のトランザクションに対して固有な他の重要なデータなどの EDI または SOAP ファイル・ヘッダー情報を含めることができます。上記の例で使用されている XML は在庫レコードのため、クライアントの購買システムとのやりとりのために XSLT を使用してこのレコードを再パッケージすることは容易に想定できます。

例: XSLT を使用した名前空間の除去

ユーザーが受け取る XML 文書には、不要または正しくない名前空間情報が含まれている場合があります。 XSLT スタイル・シートを使用して、文書内の名前空間情報を除去または操作することができます。

以下の例では、XSLT を使用して XML 文書から名前空間情報を除去する方法を示します。この例では、XML 文書および XSLT スタイルシートを XML 列に保管し、XSLTRANSFORM 関数を使用して XSLT スタイルシートのいずれかで XML 文書を変換します。

以下の CREATE ステートメントは、表 XMLDATA および XMLTRANS を作成します。XMLDATA にはサンプル XML 文書が含まれ、XMLTRANS には XSLT スタイルシートが含まれます。

```
CREATE TABLE XMLDATA (ID BIGINT NOT NULL PRIMARY KEY, XMLDOC XML );
CREATE TABLE XMLTRANS (XSLID BIGINT NOT NULL PRIMARY KEY, XSLT XML );
```

以下の INSERT ステートメントを使用して、サンプル XML 文書を XMLDATA 表に追加します。

```
insert into XMLDATA (ID, XMLDOC) values ( 1, '
<newinfo xmlns="http://mycompany.com">
<!-- merged customer information -->
  <customerinfo xmlns="http://oldcompany.com" xmlns:d="http://test" Cid="1004">
     <name>Matt Foreman</name>
  <addr country="Canada">
     <street>1596 Baseline</street>
     <city>Toronto</city>
     <prov-state>Ontario</prov-state>
     <pcode-zip>M3Z 5H9</pcode-zip>
  </addr >
  <phone type="work" >905-555-4789</phone>
  <h:phone xmlns:h="http://test1" type="home">416-555-3376</h:phone>
  <d:assistant>
      <name>Gopher Runner</name>
      <h:phone xmlns:h="http://test1" type="home">416-555-3426</h:phone>
```

```
  </d:assistant>
  </customerinfo>
</newinfo>
');
```

すべての名前空間を除去する XSLT スタイルシートの例

以下の例では、XSLT スタイルシートを使用して、表 XMLDATA に保管されている XML 文書からすべての名前空間情報を除去します。この例では、スタイルシートを表 XMLTRANS に保管し、SELECT ステートメントを使用してスタイルシートを XML 文書に適用します。

INSERT ステートメントを使用して、スタイルシートを XMLTRANS 表に追加します。

```
insert into XMLTRANS (XSLID, XSLT) values ( 1, '
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>

 <!-- keep comments -->
 <xsl:template match="comment()">
    <xsl:copy>
      <xsl:apply-templates/>
    </xsl:copy>
  </xsl:template>

  <xsl:template match="*">
    <!-- remove element prefix -->
    <xsl:element name="{local-name()}">
      <!-- process attributes -->
      <xsl:for-each select="@*">
        <!-- remove attribute prefix -->
        <xsl:attribute name="{local-name()}">
          <xsl:value-of select="."/>
        </xsl:attribute>
      </xsl:for-each>
      <xsl:apply-templates/>
    </xsl:element>
  </xsl:template>
</xsl:stylesheet>
') ;
```

以下の SELECT ステートメントは、XSLT スタイルシートを使用してサンプル XML 文書を変換します。

```
SELECT XSLTRANSFORM (XMLDOC USING XSLT ) FROM XMLDATA, XMLTRANS
   WHERE ID = 1 and XSLID = 1
```

XSLTRANSFORM コマンドは、最初の XSLT スタイルシートを使用して XML 文書を変換し、除去されたすべての名前空間情報とともに以下の XML を戻します。

```
<?xml version="1.0" encoding="UTF-8"?>
<newinfo>
 <!-- merged customer information -->
 <customerinfo Cid="1004">
  <name>Matt Foreman</name>
  <addr country="Canada">
   <street>1596 Baseline</street>
   <city>Toronto</city>
   <prov-state>Ontario</prov-state>
   <pcode-zip>M3Z 5H9</pcode-zip>
  </addr>
  <phone type="work">905-555-4789</phone>
  <phone type="home">416-555-3376</phone>
```

```
  <assistant>
   <name>Gopher Runner</name>
   <phone type="home">416-555-3426</phone>
  </assistant>
 </customerinfo>
</newinfo>
```

要素の名前空間バインディングを保持する XSLT スタイルシートの例

以下の例では、phone 要素ノードのみの名前空間バインディングを保持する XSLT スタイルシートを使用します。ノードの名前は、XSLT 変数 mynode で指定されています。この例では、スタイルシートを表 XMLTRANS に保管し、SELECT ステートメントを使用してスタイルシートを XML 文書に適用します。

以下の INSERT ステートメントを使用して、スタイルシートを XMLTRANS 表に追加します。

```
insert into XMLTRANS (XSLID, XSLT) values ( 2, '
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>
  <xsl:variable name ="mynode">phone</xsl:variable>

  <!-- keep comments -->
  <xsl:template match="comment()">
      <xsl:copy>
        <xsl:apply-templates/>
      </xsl:copy>
   </xsl:template>

  <xsl:template xmlns:d="http://test" xmlns:h="http://test1" match="*">
  <xsl:choose>

  <!-- keep namespace prefix for node names $mynode -->
    <xsl:when test="local-name() = $mynode " >
     <xsl:element name="{name()}">
       <!-- process node attributes -->
       <xsl:for-each select="@*">
       <!-- remove attribute prefix  -->
         <xsl:attribute name="{local-name()}">
           <xsl:value-of select="."/>
         </xsl:attribute>
       </xsl:for-each>
      <xsl:apply-templates/>
     </xsl:element>
    </xsl:when>

  <!-- remove namespace prefix from node -->
    <xsl:otherwise>
     <xsl:element name="{local-name()}">
       <!-- process node attributes -->
       <xsl:for-each select="@*">
       <!-- remove attribute prefix  -->
         <xsl:attribute name="{local-name()}">
           <xsl:value-of select="."/>
         </xsl:attribute>
       </xsl:for-each>
      <xsl:apply-templates/>
     </xsl:element>
     </xsl:otherwise>
  </xsl:choose>
  </xsl:template>
</xsl:stylesheet>
');
```

以下の SELECT ステートメントは、2 番目の XSLT スタイルシートを使用してサンプル XML 文書を変換します。

```
SELECT XSLTRANSFORM (XMLDOC USING XSLT) FROM XMLDATA, XMLTRANS
   WHERE ID = 1 and XSLID = 2 ;
```

XSLTRANSFORM コマンドは、2 番目の XSLT スタイルシートを使用して XML 文書を変換し、phone 要素のみを対象にした名前空間とともに以下の XML を戻します。

```
<?xml version="1.0" encoding="UTF-8"?>
<newinfo>
 <!-- merged customer information -->
 <customerinfo Cid="1004">
  <name>Matt Foreman</name>
  <addr country="Canada">
   <street>1596 Baseline</street>
   <city>Toronto</city>
   <prov-state>Ontario</prov-state>
   <pcode-zip>M3Z 5H9</pcode-zip>
  </addr>
  <phone type="work">905-555-4789</phone>
  <h:phone xmlns:h="http://test1" type="home">
   416-555-3376
  </h:phone>
  <assistant>
   <name>Gopher Runner</name>
   <h:phone xmlns:h="http://test1" type="home">
    416-555-3426
   </h:phone>
  </assistant>
 </customerinfo>
</newinfo>
```

例: XSLT の document 関数の使用

pureXML では、XSLT の組み込み関数を使用することは推奨されていません。それでも、XSLT の document 関数を使用することにした場合は、**DB2_XSLT_ALLOWED_PATH** レジストリー変数を、XML 文書が入っているディレクトリーのリストにセットアップして、参照を URI のリストに限定する必要があります。デフォルトでは、XSLT の document 関数はいずれの XML 文書にもアクセスできません。

以下の例は、XSLT の document 関数をセキュアな仕方で使用する方法を示しています。

この例では、XML 文書および XSLT スタイルシートを XML 列に保管し、XSLTRANSFORM 関数を使用して XSLT スタイルシートのいずれかで XML 文書を変換します。この例では、142 ページの『例: XSLT を使用した名前空間の除去』で作成された XMLDATA 表と XMLTRANS 表を使用しています。

Linux および UNIX 環境の場合の例

この例では、XML ファイルは /home/user/xml_files ディレクトリーにあります。**DB2_XSLT_ALLOWED_PATH** レジストリー変数をこのディレクトリーに設定します。

```
db2set DB2_XSLT_ALLOWED_PATH="/home/user/xml_files"
```

インスタンスを再始動して、このレジストリー変数の設定を有効にします。

INSERT ステートメントを使用して、スタイルシートを XMLTRANS 表に追加します。

```
INSERT INTO XMLTRANS (XSLID, XSLT)
 VALUES ( 3, '<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
               <xsl:template match="/">
                <xsl:value-of select="document(''/home/user/xml_files/n.xml'')/*"/>
               </xsl:template>
              </xsl:stylesheet>') ;
```

以下の SELECT ステートメントは、XSLID = 3 の XSLT スタイルシートを使用してサンプル XML 文書を変換します。

```
SELECT XSLTRANSFORM (XMLDOC USING XSLT ) FROM XMLDATA, XMLTRANS
   WHERE ID = 1 and XSLID = 3
```

XSLTRANSFORM コマンドは、XSLT スタイルシートを使用して XML 文書を変換し、変換された XML 文書を戻します。 XSLT の document 関数で指定された文書をデータベース・マネージャーが開けない場合、その document 関数への呼び出しは無視されます。

Windows 環境の場合の例

この例では、XML ファイルは C:¥Documents and Settings¥user¥xml_files ディレクトリーにあります。以下のいずれかの方法で、**DB2_XSLT_ALLOWED_PATH** レジストリー変数をこのディレクトリーに設定します。

- db2set DB2_XSLT_ALLOWED_PATH="C:¥Documents%20and %20Settings¥user¥xml_files"
- db2set DB2_XSLT_ALLOWED_PATH="file:///C:/Documents%20and%20Settings/user/xml_files"

ブランク・スペースを表すには、%20 を使用します。

インスタンスを再始動して、このレジストリー変数の設定を有効にします。

INSERT ステートメントを使用して、スタイルシートを XMLTRANS 表に追加します。

```
INSERT INTO XMLTRANS (XSLID, XSLT)
 VALUES ( 4, '<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
               <xsl:template match="/">
                <xsl:value-of select="document(''C:\Documents and Settings\user\xml_files\t.xml'')/*"/>
               </xsl:template>
              </xsl:stylesheet>') ;
```

以下の SELECT ステートメントは、XSLID = 4 の XSLT スタイルシートを使用してサンプル XML 文書を変換します。

```
SELECT XSLTRANSFORM (XMLDOC USING XSLT ) FROM XMLDATA, XMLTRANS
   WHERE ID = 1 and XSLID = 4
```

XSLTRANSFORM コマンドは、XSLT スタイルシートを使用して XML 文書を変換し、変換された XML 文書を戻します。 XSLT の document 関数で指定された文書をデータベース・マネージャーが開けない場合、その document 関数への呼び出しは無視されます。

XML 文書の変換に関する重要な考慮事項

組み込み XSLTRANSFORM 関数を使用して XML 文書を変換する際に当てはまる、いくつかの重要な考慮事項と制限事項があります。

XML 文書を変換する際、以下のことに注意してください。

- ソース XML 文書は、ルートが 1 つだけで、整形式でなければなりません。
- デフォルトでは XSLT 変換は UTF-8 文字を生成するので、文字データ・タイプで定義された列に挿入されると出力ストリームで文字が失われる可能性があります。

制約事項

- W3C XSLT バージョン 1.0 勧告だけがサポートされています。
- パラメーターすべてと結果タイプは SQL タイプでなければなりません。ファイル名にすることはできません。
- 複数のスタイルシート文書を使用して行う変換 (xsl:include または xsl:import 宣言を使用) はサポートされていません。

使用上の注意

XML 文書の変換は、XSLTRANSFORM 関数、XQuery の更新式、および外部アプリケーション・サーバーによる XSLT 処理など、多数の方式を使用して行えます。DB2 の XML 列に格納された文書の場合、多数の変換を実行するには XSLT よりも XQuery 更新式を使う方がより効率的です。これは、XSLT を使用する場合は常に変換対象の XML 文書の構文解析が必要になるからです。

XSLT を使用して XML 文書の変換を行うことに決めた場合は、その文書の変換をデータベース内で行うか、あるいはアプリケーション・サーバー内で行うかについて、注意深く決定する必要があります。

XSLT スタイルシート内で XSLT の document 関数を指定する場合は、**DB2_XSLT_ALLOWED_PATH** レジストリー変数を、ダウンロードする追加の XML 文書が入っているディレクトリーに設定してください。

保管および検索後の XML 文書の変更点

DB2 データベース内に XML 文書を保管してから、そのコピーをデータベースから検索すると、取り出された文書は元の文書と完全には一致しないことがあります。この性質は XML および SQL/XML 標準で定義されていて、Xerces オープン・ソース XML パーサーの性質と一致します。

文書が保存されるときに、その文書に対していくつかの変更が生じます。それらの変更は、以下のとおりです。

- XMLVALIDATE を実行すると、データベース・サーバーは以下を行います。
  - XMLVALIDATE 呼び出しで指定された XML スキーマから、デフォルトの値を入力文書に追加する
  - 無視できる空白文字を入力文書から除去する
- XML 妥当性検査を要求しない場合、データベース・サーバーは以下を行います。

– 境界空白を保持するよう要求されていない場合、これを除去する
– XML 1.0 仕様で指定されているように、行末の正規化を行う
– XML 1.0 仕様で指定されているように、属性値の正規化を行う

この処理により、属性内の改行 (U+000A) 文字がスペース文字 (U+0020) に置換されます。

データを XML 列から取り出すときに、追加の変更が生じます。それらの変更は、以下のとおりです。

- データベース・サーバーに送られる前のデータに XML 宣言がある場合、その XML 宣言は保持されません。

  暗黙的なシリアライゼーションを行う場合、CLI および組み込み SQL アプリケーションでは、 DB2 データベース・サーバーが XML 宣言を適切なエンコード方式の指定と共にデータに追加します。 Java および .NET アプリケーションでは、DB2 データベース・サーバーは XML 宣言を追加しません。ただし、データを取り出して `DB2Xml` オブジェクトに入れてから、何らかの方法を使用してデータをそのオブジェクトから取り出す場合、IBM Data Server Driver for JDBC and SQLJ が XML 宣言を追加します。

  XMLSERIALIZE 関数を実行する場合、INCLUDING XMLDECLARATION オプションが指定されていれば、 DB2 データベース・サーバーは、UTF-8 エンコードのエンコード方式を指定した XML 宣言を追加します。
- 文書の内容または属性値の中で、特定の文字が定義済み XML エンティティーに置換されます。それらの文字と定義済みエンティティーは、以下のとおりです。

文字	Unicode 値	エンティティーの表記
& 記号	U+0026	&
小なり記号	U+003C	<
大なり記号	U+003E	>

- 属性値またはテキスト値の中で、特定の文字が数値表現に置換されます。それらの文字と数値表現は、以下のとおりです。

文字	Unicode 値	エンティティーの表記
文字タブ	U+0009	
改行	U+000A
復帰	U+000D
復帰改行	U+0085	…
行区切り記号	U+2028	

- 属性値の中で、引用符 (U+0022) 文字はその定義済み XML エンティティー " に置換されます。
- 入力文書に DTD 宣言がある場合、その宣言は保持されず、DTD を基にしたマークアップは生成されません。
- 入力文書に CDATA セクションが含まれる場合、それらのセクションは出力の中に保持されません。

第 6 章 XML データの索引付け

XML データに対する索引を使用すると、XML 列に格納されている XML 文書の照会の効率を向上させることができます。

ユーザーが指定する 1 つ以上の表列で索引キーが構成される従来のリレーショナル索引とは対照的に、XML データに対する索引は特定の XML パターン式を使用して、単一の列に格納された XML 文書内にあるパスおよび値に索引付けを行います。その列のデータ・タイプは XML でなければなりません。

文書の先頭へのアクセスを可能にする代わりに、XML データに対する索引内の索引項目は XML パターン式に基づく索引キーを作成して、文書内のノードに対するアクセスを可能にします。 XML 文書の複数の部分が 1 つの XML パターンを満たす場合があるため、複数の索引キーが、単一の文書用の索引に挿入される場合があります。

XML データに対する索引の作成には CREATE INDEX ステートメントを使用し、XML データに対する索引のドロップには DROP INDEX ステートメントを使用します。 CREATE INDEX ステートメントの GENERATE KEY USING XMLPATTERN 節で、索引付けする対象を指定します。

XML 以外の列の索引において CREATE INDEX ステートメントで使用されるキーワードの一部は、XML データに対する索引に適用されません。 XML データに対する索引では、UNIQUE キーワードも異なる意味を持ちます。

例: XML データに対する索引の作成

表 companyinfo には companydocs という名前の XML 列があり、そこには以下のような XML 文書フラグメントが含まれているとします。

Company1 の文書

```
<company name="Company1">
  <emp id="31201" salary="60000" gender="Female">
    <name>
      <first>Laura</first>
      <last>Brown</last>
    </name>
    <dept id="M25">
      Finance
    </dept>
  </emp>
</company>
```

Company2 の文書

```
<company name="Company2">
  <emp id="31664" salary="60000" gender="Male">
    <name>
      <first>Chris</first>
      <last>Murphy</last>
    </name>
    <dept id="M55">
      Marketing
    </dept>
  </emp>
```

```
    <emp id="42366" salary="50000" gender="Female">
      <name>
        <first>Nicole</first>
        <last>Murphy</last>
      </name>
      <dept id="K55">
        Sales
      </dept>
    </emp>
  </company>
```

companyinfo 表のユーザーは、従業員 ID を使用して従業員情報をよく検索します。次のような索引を使用すると、この検索をより効率的にできます。

```
CREATE INDEX empindex on companyinfo(companydocs)          1
  GENERATE KEY USING XMLPATTERN '/company/emp/@id'         2
   AS SQL DOUBLE                                     3
```

図 5. XML データに対する索引の例

図 5 の注:

1 XML データに対する索引は companyinfo 表の companydocs 列に定義されています。 companydocs は XML データ・タイプでなければなりません。

2 GENERATE KEY USING XMLPATTERN 節は、索引付けする対象に関する情報を提供します。この節は XML 索引の仕様と呼ばれます。 XML 索引の仕様には XML パターン節が含まれます。この例の XML パターン節は、各従業員要素の id 属性の値に索引付けすることを示します。

3 AS SQL DOUBLE は、索引の値は DOUBLE 値として格納されることを示します。

索引 XML パターン式

XML 列に格納された XML 文書の XML パターン式を満たす部分のみが索引付けされます。 XML パターンに索引付けするには、CREATE INDEX ステートメントと一緒に SPECIFICATION ONLY 指定の索引節を指定します。

SPECIFICATION ONLY 指定の索引節は、GENERATE KEY USING XMLPATTERN で始まり、XML パターンと XML データに対する索引のデータ・タイプがその後に続きます。代わりに、GENERATE KEYS USING XMLPATTERN 節を指定することもできます。

CREATE INDEX ステートメントごとに 1 つの SPECIFICATION ONLY 指定の索引節のみが許可されます。 1 つの XML 列に対して複数の XML 索引を作成できます。

XML パターン式

索引付けされる文書の部分を識別するには、XML パターンを使用して XML 文書内のノードのセットを指定します。このパターン式は XQuery 言語で定義されるパス式に似ていますが、XQuery 言語のサブセットのみがサポートされるという点で異なります。

パス式の各ステップは、スラッシュ (/) で分離されます。 /descendant-or-self::node()/ の省略構文である二重スラッシュ (//) も指定できます。それぞれのステップにおいて、フォワード軸 (child::、@、attribute::、descendant::、self::、および descendant-or-self::) が 1 つ選択され、XML 名テストまたは XML 種類テストがそれに続きます。フォワード軸を指定しない場合、child 軸がデフォルトとして使用されます。

XML 名テストが使用される場合、パス内のステップにおいて突き合わせするノード名を指定するために、修飾 XML 名またはワイルドカードが使用されます。ノード名に突き合わせる代わりに、XML 種類テストを使用してパターン内で突き合わせされるノードの種類を指定することもできます。テキスト・ノード、コメント・ノード、処理命令ノード、その他の任意のタイプのノードを指定できます。

パターン式に、サポートされる関数への呼び出しを含めて、大/小文字を区別しないなどの特殊なプロパティーを持つ索引を作成することができます。関数ステップは 1 つの XMLPATTERN 節につき 1 つのみ指定できます。

以下の例は、パターン式が異なるものの論理的に等価であるステートメントを示しています。

ステートメント 1 とステートメント 2 は、論理的に等価です。ステートメント 1 は省略した構文を使用しています。

```
CREATE INDEX empindex on company(companydocs)
       GENERATE KEY USING XMLPATTERN '/company/emp/@id' AS SQL DOUBLE
```

図 6. ステートメント 1

```
CREATE INDEX empindex on company(companydocs)
       GENERATE KEY USING XMLPATTERN '/child::company/child::emp/attribute::id'
       AS SQL DOUBLE
```

図 7. ステートメント 2

ステートメント 3 とステートメント 4 は、論理的に等価です。ステートメント 3 は省略した構文を使用しています。

```
CREATE INDEX idindex on company(companydocs)
       GENERATE KEY USING XMLPATTERN '//@id' AS SQL DOUBLE
```

図 8. ステートメント 3

```
CREATE INDEX idindex on company(companydocs)
       GENERATE KEY USING XMLPATTERN '/descendant-or-self::node()/attribute::id'
       AS SQL DOUBLE
```

図 9. ステートメント 4

ステートメント 5 は XML 種類テストを使用して、指定されたパターン内のテキスト・タイプ・ノードを突き合わせています。

図 10. ステートメント 5

```
CREATE INDEX empindex on company(companydocs)
       GENERATE KEY USING XMLPATTERN '/company/emp/name/last/text()' AS SQL
       VARCHAR(25)
```

ステートメント 6 および 7 は、大/小文字を区別しない索引を作成します。ステートメント 7 は索引に格納する値を、どのロケールに変換すべきか指定しています。

```
CREATE INDEX empindex on company(companydocs)
       GENERATE KEY USING XMLPATTERN '/company/emp/name/last/fn:upper-case(.)'
       AS SQL VARCHAR(25)
```

図 11. ステートメント 6

図 12. ステートメント 7

```
CREATE INDEX empindex on company(companydocs)
       GENERATE KEY USING XMLPATTERN '/company/emp/name/last/fn:upper-case(.,
        "en_US")' AS SQL VARCHAR(25)
```

ステートメント 8 は、XML 文書構造内の従業員のミドルネームの存在を示す索引を作成しています。

図 13. ステートメント 8

```
CREATE INDEX empindex on company(companydocs)
       GENERATE KEY USING XMLPATTERN
       '/company/emp/name/fn:exists(middle)' AS SQL VARCHAR(1)
```

ステートメント 9 は VARCHAR 索引を作成しています。

```
CREATE INDEX varcharidx on company(companydocs)
       GENERATE KEY USING XMLPATTERN '/company/emp/name/last'
       AS SQL VARCHAR(30)
```

図 14. ステートメント 9

DB2 V10.1 以降、オプティマイザーは、fn:starts-with 関数を述部に含む照会に VARCHAR タイプの索引の使用を選択できます。fn:starts-with 関数は、ストリングの先頭が特定のサブストリングであるかどうかを判別します。既存の VARCHAR 索引に対する変更は不要です。また、新規索引の場合も CREATE INDEX ステートメントに特殊な構文を使用する必要はありません。

条件に適合するパスとノード

XML 列 companydocs に XML 文書が格納されている company という名前の表で考察してみます。XML 文書には、`'/company/emp/dept/@id'` および `'/company/emp/@id'` という 2 つのパスを持つ階層があります。 XML パターンに単一パスを指定する場合、文書内のノードのセットが条件に適合すると考えられます。

例えば、従業員要素において特定の従業員 ID 属性 (@id) を検索する場合、XML パターン `'/company/emp/@id'` に索引を作成できます。そうすることで、`'/company/emp[@id=42366]'` という形式の述部を持つ照会は、この XML 列の索引を利用できます。この場合、文書内の従業員要素はすべて従業員 ID 属性を持つ可

能性があるため、 CREATE INDEX ステートメントの XMLPATTERN '/company/emp/@id' に指定した単一パスは、文書内のさまざまなノードを指すことになります。

```
CREATE INDEX empindex on company(companydocs)
        GENERATE KEY USING XMLPATTERN '/company/emp/@id' AS SQL DOUBLE
```

XML パターンにワイルドカード式、descendant 軸、または descendant-or-self 軸を使用する場合は、パスとノードのセットが条件に適合する可能性があります。以下の例では descendant-or-self 軸を指定しており、部門 ID 属性および従業員 ID 属性の両方が @id を持つため、 XML パターンの '//@id' はそれら両方のパスを指すことになります。

```
CREATE INDEX idindex on company(companydocs)
        GENERATE KEYS USING XMLPATTERN '//@id' AS SQL DOUBLE
```

例えば、名前要素において特定の従業員の姓 (<last>) を大/小文字を区別せずに検索する場合、 XML パターン '/company/emp/name/last/fn:upper-case(.)' に索引を作成できます。 そうすることで、'/company/emp/name/last[fn:upper-case(.)="SMITH"]" という形式の述部を持つ照会は、この XML 列の索引を利用できます。この場合、XML パターンのコンテキスト・ステップには、単一のパス '/company/emp/name/last' を指定しています。文書内の各名前要素が <last> 要素を持つ可能性があるため、このパスの場合、文書内のさまざまなノードが条件に適合する可能性があります。従業員の姓はすべて大文字形式で索引付けされます。

```
CREATE INDEX empindex on company(companydocs)
        GENERATE KEY USING XMLPATTERN '/company/emp/name/last/fn:upper-case(.)'
        AS SQL VARCHAR(25)
```

XML パターンに fn:exists 関数を含めた場合、索引付けの対象項目が XML 文書構造内に存在するかどうかの真偽を表す単一文字 T または F が、索引の値として格納されます。以下の例では、従業員のミドルネームはすべてブール値の形式で索引付けされます。

```
CREATE INDEX midnameidx on company(companydocs)
        GENERATE KEY USING XMLPATTERN '/company/emp/name/fn:exists(middle)'
        AS SQL VARCHAR(1)
```

大/小文字を区別しない XML 索引の使用例

fn:upper-case() 関数を使用して、大/小文字を区別しない項目として値を格納する索引を作成すると、ストリング・データを検索する照会の速度を上げることができます。

大/小文字を区別しない索引の作成

この例は、 XML 列を持つ表を作成し、その表にデータを挿入し、大/小文字を区別しない索引を作成する方法を示しています。

まず、以下を使用して、XML タイプの CONTACTINFO という列を持つ CLIENTS という表を作成します。

```
CREATE TABLE clients (
  ID            INT PRIMARY KEY NOT NULL,
  NAME          VARCHAR(50),
  STATUS        VARCHAR(10),
  CONTACTINFO   XML
);
```

以下を使用して、2 つのレコードを CLIENTS 表に挿入します。

```
INSERT INTO clients VALUES('0092', 'Johny Peterson', 'Standard',
'<Client>
    <address type="permanent">
         <street>8734 Zuze Ave.</street>
         <city>New York</city>
         <state>New York</state>
         <zip>95443</zip>
     </address>
</Client>');

INSERT INTO clients VALUES('0093', 'Rose Locke', 'Golden',
'<Client>
    <address type="PERMANENT">
         <street>1121 Oxford Street</street>
         <city>Albany</city>
         <state>new york</state>
         <zip>19232</zip>
     </address>
</Client>');
```

例えば、/Client/address/state パスに対して clients_state_idx という大/小文字を区別しない索引を作成できます。 fn:upper-case() 関数の第一パラメーターは必ずコンテキスト・アイテム式 (.) でなければなりません。

```
CREATE INDEX clients_state_idx ON clients(contactinfo)
   GENERATE KEYS USING XMLPATTERN '/Client/address/state/fn:upper-case(.)'
   AS SQL VARCHAR(50);
```

属性に対しても大/小文字を区別しない索引を作成することができます。例えば、address の属性 type、/Client/address/@type パスに対して client_address_type_idx という索引を作成できます。

```
CREATE INDEX client_address_type_idx ON clients(contactinfo)
   GENERATE KEYS USING XMLPATTERN '/Client/address/@type/fn:upper-case(.)'
   AS SQL VARCHAR(50);
```

clients_state_idx 索引および client_address_type_idx 索引の両方とも、索引キー値は米国英語のエンコード・セットの大文字形式で格納されます。例えば、先に挿入した 1 番目のデータ・レコードの場合、 /Client/address/state パスに関連付けられた値は New York ですが、NEW YORK として格納されます。/Client/address/@type 属性に関連付けられた値は、小文字のストリング permanent ですが、PERMANENT として格納されます。

大/小文字を区別しない索引を使用する照会の実行

以下の条件を索引パターンおよび述部が満たす場合に限り、オプティマイザーは大/小文字を区別しない索引を考慮します。

- CREATE INDEX ステートメントの GENERATE KEYS USING XMLPATTERN 節のコンテキスト・ステップのパスが、照会述部の XML パスと一致している。
- ロケール名が CREATE INDEX ステートメントに指定された場合は、そのロケール名が、照会述部の fn:upper-case() 関数で指定されたロケールと一致している。
- 照会述部で使用された fn:upper-case() の第一パラメーターが、コンテキスト・アイテム式 (.) である。

以下の照会では、大/小文字を区別しない索引 clients_state_idx が存在する場合は、オプティマイザーはその索引の使用を選択する可能性があります。 state 要素の値

が New York (大文字か小文字かにかかわらず) であるレコードを検出する方法として、オプティマイザーは、表スキャンを実行する代わりに clients_state_idx 索引のスキャンを選択する可能性があります。ただし、その方が作業量が少なくて済む場合です。

```
XQUERY db2-fn:xmlcolumn('CLIENTS.CONTACTINFO')
   /Client/address/state[fn:upper-case(.)="NEW YORK"];

--------------------------------
<state>New York</state>
<state>new york</state>

2 record(s) selected.
```

ロケール・パラメーターの指定

大/小文字を区別しない索引を作成する際に、fn:upper-case 関数のオプションのロケール・パラメーターを使用できます。例えば、以下のステートメントは、address の属性 type (パス /Client/address/@type) に対して tr_TR ロケールで索引を作成しています。

```
CREATE INDEX client_address_type_idx_tr ON clients(contactinfo)
   GENERATE KEYS USING XMLPATTERN '/Client/address/@type/fn:upper-case(., "tr_TR")'
   AS SQL VARCHAR(50);
```

両端の引用符を省略するなどしてロケール・ストリングを正しく指定していない場合、ロケール名にはデフォルト値が使用されることに注意してください。索引作成時に使用されたロケールを調べるには、**db2look** コマンドまたは **DESCRIBE** ステートメントを使用します。例: DESCRIBE INDEXES FOR TABLE CLIENTS SHOW DETAIL。

オプティマイザーがこの client_address_type_idx_tr 索引の使用を選択する可能性があるのは、照会の照会述部の fn:upper-case() にもロケール tr_TR が指定されている場合のみです。例:

```
SELECT id FROM clients client1
  WHERE XMLEXISTS('$XMLDOC/Client/address/@type[fn:upper-case(., "tr_TR")="PERMANENT"]'
  PASSING client1.contactinfo as "XMLDOC")

ID
-----------
          92
          93

  2 record(s) selected.
```

以下の例のような照会では、異なるロケールが指定されているため、索引 client_address_type_idx_tr は使用されません。

```
SELECT id FROM clients client1
  WHERE XMLEXISTS('$XMLDOC/Client/address/@type[fn:upper-case(., "en_US")="PERMANENT"]'
  PASSING client1.contactinfo as "XMLDOC")

ID
-----------
          92
          93

  2 record(s) selected.
```

fn:exists を指定した索引の使用例

fn:exists 関数を使用すると、要素または属性がデータ値 (空シーケンス以外) を持つかどうかの真偽を表すブール値 (単一文字 T または F) を格納する索引を作成できます。

fn:exists を使用した索引の作成

この例は、 XML 列を持つ表を作成し、その表にデータを挿入し、fn:exists 関数を使用して索引を作成する方法を示しています。

まず、以下を使用して、XML タイプの INCOMEINFO という列を持つ INCOME という表を作成します。

```
CREATE TABLE income (
  ID           INT PRIMARY KEY NOT NULL,
  INCOMEINFO   XML
);
```

以下を使用して、3 つのレコードを INCOME 表に挿入します。

```
INSERT INTO income VALUES('1',
'<Employee>
    <salary type="regular">
         <base>5500.00</base>
         <bonus>1000.00</bonus>
    </salary>
</Employee>');

INSERT INTO income VALUES('2',
'<Employee>
    <salary type="contractor">
         <base>7600.00</base>
    </salary>
</Employee>');

INSERT INTO income VALUES('3',
'<Employee>
    <salary>
         <base>2600.00</base>
         <bonus>500.00</bonus>
    </salary>
</Employee>');
```

例えば、以下のように、手当 (bonus) を得ている従業員を確認するために、パス `/Employee/salary/fn:exists(bonus)` を使用して **exists_bonus_idx** という索引を作成できます。

```
CREATE INDEX exists_bonus_idx ON
   income(incomeinfo) GENERATE KEYS USING XMLPATTERN
   '/Employee/salary/fn:exists(bonus)' AS SQL VARCHAR(1);
```

また、以下のように、給与 (salary) 要素の属性の存在を確認するために、パス `/Employee/salary/fn:exists(@*)` を使用して **exists_any_attrib_idx** という索引を作成することもできます。

```
CREATE INDEX exists_any_attrib_idx ON
   income(incomeinfo) GENERATE KEYS USING XMLPATTERN
   '/Employee/salary/fn:exists(@*)' AS SQL VARCHAR(1);
```

これらの索引を使用する照会の実行

fn:exists 関数を使用して作成された索引をオプティマイザーが考慮するのは、以下の両方の記述が当てはまる場合のみです。

- 索引パターンのパスが、照会述部の XML パスと一致している。
- 照会述部が、 CREATE INDEX ステートメントの fn:exists のパラメーターに指定されていた要素または属性を検索している。

以下の照会では、オプティマイザーは、表スキャンを実行する代わりに索引 **exists_bonus_idx** の使用を選択する可能性があります。ただし、その方が作業量が少なくて済む場合です。

```
XQUERY db2-fn:xmlcolumn('INCOME.INCOMEINFO')
    /Employee/salary[fn:exists(bonus)];

--------------------------------
<salary type="regular"><base>5500.00</base><bonus>1000.00</bonus></salary>
<salary><base>2600.00</base><bonus>500.00</bonus></salary>

2 record(s) selected.
```

この照会を次の形式に書き直した場合も、 **exists_bonus_idx** 索引が考慮されます。

```
XQUERY db2-fn:xmlcolumn('INCOME.INCOMEINFO')/Employee/salary[bonus];

--------------------------------
<salary type="regular"><base>5500.00</base><bonus>1000.00</bonus></salary>
<salary><base>2600.00</base><bonus>500.00</bonus></salary>

2 record(s) selected.
```

手当を得ていない全従業員を検出する以下の 2 つの照会について、**exists_bonus_idx** 索引は考慮されます。

```
XQUERY db2-fn:xmlcolumn('INCOME.INCOMEINFO')
   /Employee/salary[not(fn:exists(bonus))];

--------------------------------
<salary type="contractor"><base>7600.00</base></salary>

1 record(s) selected.
```

```
XQUERY db2-fn:xmlcolumn('INCOME.INCOMEINFO')
   /Employee/salary[fn:not(fn:exists(bonus))];
--------------------------------
<salary type="contractor"><base>7600.00</base></salary>

1 record(s) selected.
```

以下の照会では、オプティマイザーは、索引 **exists_any_attrib_idx** の使用を選択する可能性があります。この索引は、給与要素に属性があるかどうかをチェックします。この例で使用しているデータでは、 type 属性のみ存在しています。従って、この場合は、属性 `@type` が XML パス `/Employee/salary` に存在する場合のみ照会述部は TRUE となります。

```
XQUERY db2-fn:xmlcolumn('INCOME.INCOMEINFO')
   /Employee/salary[fn:exists(@type)];
--------------------------------
<salary type="regular"><base>5500.00</base><bonus>1000.00</bonus></salary>
<salary type= "contractor"><base>7600.00</base></salary>

2 record(s) selected.
```

以下の形式で記述された照会では、オプティマイザーは **exists_any_attrib_idx** 索引も考慮します。

```
XQUERY db2-fn:xmlcolumn('INCOME.INCOMEINFO')/Employee/salary[bonus and @type];

 --------------------------------
<salary type="regular"><base>5500.00</base><bonus>1000.00</bonus></salary>

1 record(s) selected.


XQUERY db2-fn:xmlcolumn('INCOME.INCOMEINFO')
  /Employee/salary[bonus and base > 3000];

 --------------------------------
<salary type="regular"><base>5500.00</base><bonus>1000.00</bonus></salary>

1 record(s) selected.


XQUERY db2-fn:xmlcolumn('INCOME.INCOMEINFO')
   /Employee/salary[bonus and bonus > 600];

 --------------------------------
<salary type="regular"><base>5500.00</base><bonus>1000.00</bonus></salary>

1 record(s) selected.


XQUERY db2-fn:xmlcolumn('INCOME.INCOMEINFO')
   /Employee/salary[bonus][bonus > 600];

 --------------------------------
<salary type="regular"><base>5500.00</base><bonus>1000.00</bonus></salary>

1 record(s) selected.


XQUERY for $e in db2-fn:xmlcolumn('INCOME.INCOMEINFO')
   /Employee/salary where $e/bonus return $e;

 --------------------------------
<salary type="regular"><base>5500.00</base><bonus>1000.00</bonus></salary>
<salary><base>2600.00</base><bonus>500.00</bonus></salary>

1 record(s) selected.
```

UNIQUE キーワードを使用して作成した索引

UNIQUE キーワードを使用し、さらに XMLPATTERN パターン節に fn:exists も指定して作成した索引の場合、ユニーク・セマンティクスの制約を受けるのは、索引パターンのコンテキスト・ステップのみにおける XML ワイルドカード (および "//" などのその他の構文) の出現です。 fn:exists の入力引数における出現ではありません。例えば、以下のステートメントは有効です。

```
CREATE UNIQUE INDEX i2 ON tbx1(x1) GENERATE KEYS USING XMLPATTERN
'/node/node1/fn:exists(*)' AS SQL VARCHAR(1)
```

一方、この次のステートメントはエラーを返します。索引パターンのコンテキスト・ステップがユニークではないためです。

```
CREATE UNIQUE INDEX i2 ON tbx1(x1) GENERATE KEYS USING XMLPATTERN
'/node/*/fn:exists(a)' AS SQL VARCHAR(1)
```

fn:starts-with を指定した照会での索引の使用例

DB2 V10.1 以降、述部に fn:starts-with 関数を含む照会に、オプティマイザーは、VARCHAR タイプの索引の使用を選択して照会速度を上げることができます。

既存の VARCHAR 索引に対する変更は不要です。また、新規の VARCHAR 索引を作成する場合も CREATE INDEX ステートメントに特殊な構文を使用する必要はありません。

fn:starts-with 関数は、ストリングの先頭が特定のサブストリングであるかどうかを判別します。

VARCHAR タイプ索引の作成

この例は、 XML 列を持つ表を作成し、その表にデータを挿入し、VARCHAR タイプの索引を作成する方法を示しています。

まず、以下を使用して、XML タイプの CDINFO という列を持つ FAVORITE_CDS という表を作成します。

```
CREATE TABLE favorite_cds (
  NAME          CHAR(20) NOT NULL,
  CDID          BIGINT,
  CDINFO        XML
);
```

以下を使用して、FAVORITE_CDS 表にレコードを挿入します。

```
INSERT INTO favorite_cds VALUES('John Peterson', 01,
'<FAVORITECDS>
    <CD>
         <TITLE>Top hits</TITLE>
         <ARTIST>Good Singer</ARTIST>
         <COMPANY>Top Records</COMPANY>
         <YEAR>1999</YEAR>
    </CD>
    <CD>
         <TITLE>More top hits</TITLE>
         <ARTIST>Better Singer</ARTIST>
         <COMPANY>Better Music </COMPANY>
         <YEAR>2005</YEAR>
    </CD>
    <CD>
         <TITLE>Even more top hits</TITLE>
         <ARTIST>Best Singer</ARTIST>
         <COMPANY>Best Music</COMPANY>
         <YEAR>2010</YEAR>
    </CD>
</FAVORITECDS>');
```

例えば、以下のように、 **year_idx** という VARCHAR 索引を /FAVORITECDS/CD/YEAR パスに対して作成できます。

```
CREATE INDEX year_idx ON favorite_cds (cdinfo)
   GENERATE KEYS USING XMLPATTERN '/FAVORITECDS/CD/YEAR'
   AS SQL VARCHAR(20);
```

また、例えば、**company_idx** という VARCHAR 索引を /FAVORITECDS/CD/COMPANY パスに対して作成することもできます。以下の例では、fn:upper-case 関数を使用して企業名に対して大/小文字を区別しない索引を作成しています。

```
CREATE INDEX company_idx ON favorite_cds (cdinfo)
   GENERATE KEYS USING XMLPATTERN '/FAVORITECDS/CD/COMPANY/fn:upper-case(.)'
   AS SQL VARCHAR(20);
```

fn:starts-with を述部に含む照会の実行

以下の照会では、1990 年代の CD (この例では 199 で始まる年の CD を fn:starts-with 関数で検出しています) を検出する方法として、オプティマイザーは、表スキャンを実行する代わりに VARCHAR 索引 **year_idx** の使用を選択する可能性があります。オプティマイザーが **year_idx** 索引の使用を選択するのは、その方が作業量が少なくて済む場合です。

```
XQUERY for $y in db2-fn:xmlcolumn
   ('FAVORITE_CDS.CDINFO')/FAVORITECDS/CD
    [YEAR/fn:starts-with(., "199")] return $y

--------------------------------
<CD>
   <TITLE>Top hits</TITLE>
   <ARTIST>Good Singer</ARTIST>
   <COMPANY>Top Records</COMPANY>
   <YEAR>1999</YEAR>
</CD>
1 record(s) selected.
```

照会が、fn:starts-with 関数の第 1 パラメーターにコンテキスト・アイテム式 (.) を指定した形式で記述されていれば、オプティマイザーは 1 つのみの索引スキャンを選択できますが、それ以外の形式の場合は、2 つの索引スキャンが必要になる可能性があることに注意してください。このため、以下のような形式で記述された照会は、実行がより遅くなる可能性があります。

```
XQUERY for $y in db2-fn:xmlcolumn
   ('FAVORITE_CDS.CDINFO')/FAVORITECDS/CD
    [fn:starts-with(YEAR, "199")] return $y
```

次の照会では、企業名が BES で始まるレコードを検出する方法として、オプティマイザーは、表スキャンを実行する代わりに VARCHAR 索引 **company_idx** の使用を選択する可能性があります。オプティマイザーが **company_idx** 索引の使用を選択するのは、その方が作業量が少なくて済む場合です。

```
XQUERY for $y in db2-fn:xmlcolumn
   ('FAVORITE_CDS.CDINFO')/FAVORITECDS/CD
    [COMPANY/fn:starts-with(fn:upper-case(.), "BES")] return $y

--------------------------------
    <CD>
         <TITLE>Even more top hits</TITLE>
         <ARTIST>Best Singer</ARTIST>
         <COMPANY>Best Music</COMPANY>
         <YEAR>2010</YEAR>
    </CD>
1 record(s) selected.
```

コンテキスト・ステップおよび関数式ステップ

コンテキスト・ステップおよび関数式ステップは、XML データに対する関数索引を作成する際に指定される XML 索引パターンの一部です。コンテキスト・ステップは、索引付けする要素ノードまたは属性ノードのセットの XML 索引パターンを定義します。関数式ステップは、索引に格納するキー値を定義する関数を指定します。

一般的に、例えば fn:exists や fn:upper-case 関数を使用して XML データに関数索引を作成する際に GENERATE KEYS USING XMLPATTERN 節に指定する XML パターン式には 2 つの部分があります。

1 つ目の部分はコンテキスト・ステップ と呼ばれるものです。コンテキスト・ステップは、索引項目を作成する要素ノードまたは属性ノードの XML パスを指定します。コンテキスト・ステップの構文は、一般的な XML 索引の索引パターン式と同じ構文に従いますが、 fn:upper-case や fn:exists などの特定の関数と一緒に使用する場合には、いくつかの制限があります。詳しくは、 CREATE INDEX ステートメントの使用に関する情報を参照してください。

2 つ目の部分は*関数式ステップ* と呼ばれるものです。関数式ステップは、fn:exists や fn:upper-case などの関数と、そのパラメーターを指定します。関数式ステップは、コンテキスト・ステップによって指定された各ノードについて、索引に格納する実際のキー値を生成します。

例えば、索引 XML パターン `/a/b/fn:upper-case(.)` の場合:

- コンテキスト・ステップは `/a/b` です。
- 関数式ステップは `fn:upper-case(.)` です。

もう 1 つの例として、索引 XML パターン `/a/b/fn:exists(c)` の場合

- コンテキスト・ステップは `/a/b` です。
- 関数式ステップは `fn:exists(c)` です。

XML 名前空間の宣言

修飾 XML 名 (QName) は、XML パターン式内で要素および属性のタグを定義するために使用されます。 QName の修飾子は、名前空間 URI と関連付けられている名前空間接頭部です。

XML パターンをオプションの名前空間宣言を使用して指定して、名前空間接頭部を名前空間 URI ストリング・リテラルにマップするか、または XML パターンのデフォルトの名前空間 URI を定義することができます。次いで、XML パターン内の要素および属性の名前を修飾するために名前空間接頭部を使用して、それらを文書内で使用される名前空間と一致させることができます。

以下の例では、省略した名前空間接頭部 *m* が *http://www.mycompanyname.com/* にマップします。

```
CREATE INDEX empindex on department(deptdocs)
     GENERATE KEYS USING XMLPATTERN
     'declare namespace m="http://www.mycompanyname.com/";
     /m:company/m:emp/m:name/m:last' AS SQL VARCHAR(30)
```

この CREATE INDEX ステートメントがコマンド行プロセッサー (CLP) から発行された場合、セミコロンはデフォルトのステートメント終止符であるので、埋め込まれたセミコロンが問題となります。この問題を避けるため、次のいずれかの回避策を使用してください。

- 行の中でセミコロンが空白文字以外の最後の文字とならないようにする (例えば、セミコロンの後に空の XQuery コメントを追加するなどして)。
- コマンド行から CLP でのデフォルトのステートメント終止符を変更する。

複数の名前空間宣言を同じ XMLPATTERN 式で指定することもできますが、名前空間接頭部は名前空間宣言のリスト内で固有でなければなりません。さらに、ユーザーには、接頭部がない要素に対するデフォルトの名前空間を宣言するというオプションがあります。要素の名前空間または名前空間接頭部が明示的に指定されていない場合には、デフォルトの名前空間が使用されます。デフォルトの名前空間宣言は属性には適用されません。ユーザーがデフォルトの名前空間を指定しなかった場合、名前空間は *no namespace* になります。デフォルトの名前空間は 1 つしか宣言できません。この名前空間宣言の動作は、XQuery 規則に従います。

前の例は、デフォルトの名前空間を使用して次のように作成することもできます。

```
CREATE INDEX empindex on department(deptdocs)
   GENERATE KEY USING XMLPATTERN
   'declare default element namespace "http://www.mycompany.com/";
    /company/emp/name/last') AS SQL VARCHAR(30)
```

次の例では、*@id* 属性は *no namespace* 名前空間を持ちます。なぜなら、デフォルトの名前空間 *http://www.mycompany.com/* は *company* および *emp* 要素のみに適用され、*@id* 属性には適用されないからです。 XML 文書ではデフォルトの名前空間宣言は属性には適用されないため、これは基本的な XQuery 規則に従います。

```
CREATE INDEX empindex on department(deptdocs)
   GENERATE KEY USING XMLPATTERN
   'declare default element namespace "http://www.mycompany.com/";
    /company/emp/@id' AS SQL VARCHAR(30)
```

@id 属性は *company* および *emp* 要素と同じ名前空間を持つはずなので、このステートメントは次のように書き直すこともできます。

```
CREATE INDEX empindex on department(deptdocs)
   GENERATE KEY USING XMLPATTERN
   'declare default element namespace "http://www.mycompany.com/";
    declare namespace m="http://www.mycompanyname.com/";
    /company/emp/@m:id' AS SQL VARCHAR(30)
```

インスタンス文書で使用される索引と、名前空間接頭部を作成するために使用される名前空間接頭部は、索引付けのために一致する必要はありませんが、完全に拡張された QName は一致する必要があります。接頭部名そのものではなく、接頭部の拡張後の名前空間の値が重要です。例えば、索引の名前空間接頭部が *m="http//www.mycompany.com/"* と定義されていて、インスタンス文書で使用される名前空間接頭部が *c="http//www.mycompany.com/"* である場合には、省略した名前空間接頭部である *m* と *c* の両方が同じ名前空間に拡張するため、インスタンス文書内の *c:company/c:emp/@id* が索引付けされます。

索引 XML パターン式と関連付けられたデータ・タイプ

CREATE INDEX ステートメントで指定された各 XML パターン式は、データ・タイプと関連付けられていなければなりません。次の SQL データ・タイプがサポートされています: VARCHAR、 DATE、TIMESTAMP、INTEGER、DECIMAL、および DOUBLE。

式の結果を複数のデータ・タイプで解釈するように選択することができます。例えば、値 *123* には文字表現がありますが、数値 *123* としても解釈できます。パス */company/emp/@id* を文字ストリングと数値の両方で索引付けする場合には、2 つの

索引を作成する必要があり、1 つを VARCHAR データ・タイプ用、もう 1 つを DOUBLE データ・タイプ用にします。文書内の値は索引ごとに指定されたデータ・タイプに変換されます。

次の例は、同じ XML 列 *deptdocs* に異なるデータ・タイプを持つ 2 つの索引を作成する方法を示しています。

```
CREATE INDEX empindex1 on department(deptdocs)
         GENERATE KEY USING XMLPATTERN '/company/emp/@id' AS SQL VARCHAR(10)

CREATE INDEX empindex2 on department(deptdocs)
        GENERATE KEY USING XMLPATTERN '/company/emp/@id' AS SQL DOUBLE
```

サポートされる SQL データ・タイプの説明をします。

VARCHAR(*integer*)

VARCHAR データは、UTF-8 コード・ページで XML 列の索引に格納されます。データ・タイプ VARCHAR が指定された長さである *integer* (バイト単位) と共に使用された場合、指定された長さは制約として処理されます。文書が表に挿入されたり、索引が作成される時に文書が表に存在する場合、指定された長さより長い値で索引付けされるノードがある場合には、文書の挿入または索引の作成は失敗します。挿入または作成が成功した場合、索引に文字ストリング値全体がすべて確実に格納され、索引では範囲スキャンと等価検索の両方をサポートできます。長さ *integer* は、1 からページ・サイズに依存した最大値の範囲の値です。許可された最大長のリストについては、CREATE INDEX ステートメントを参照してください。ストリング比較には XQuery の意味体系が使用されます。そこでは末尾ブランクは意味を持ちます。これは、比較時に末尾ブランクが意味を持たない SQL の意味体系とは異なります。

```
CREATE INDEX empindex1 on department(deptdocs)
     GENERATE KEY USING XMLPATTERN '/company/emp/@id' AS SQL VARCHAR(50)
```

VARCHAR HASHED

VARCHAR HASHED は任意の長さの文字ストリングの索引付けを処理するために指定できます。文書に索引付けされる文字ストリングが含まれており、それがページ・サイズに依存する最大値を基にして索引に許可された最大長 *integer* を超えている場合には、代わりに VARCHAR HASHED を指定できます。この場合、システムはストリング全体に対する 8 バイトのハッシュ・コードを生成し、索引付きストリングの長さには制限はなくなります。VARCHAR HASHED を指定した場合には、索引に実際の文字データの代わりにハッシュ・コードが入るため、範囲スキャンは実行できません。これらのハッシュ文字ストリングを使用する索引は、等価検索にのみ使用可能です。ストリング等価比較には XQuery の意味体系が使用されます。そこでは末尾ブランクは意味を持ちます。これは、比較時に末尾ブランクが意味を持たない SQL の意味体系とは異なります。ストリングのハッシュは、等価における XQuery の意味体系を保持しますが、等価における SQL の意味体系は保持しません。

```
CREATE INDEX empindex on company(companydocs)
    GENERATE KEY USING XMLPATTERN '/company/emp/name/last' AS SQL
    VARCHAR HASHED
```

DOUBLE

すべての数値は変換され、DOUBLE データ・タイプで索引に格納されます。無制限の 10 進タイプおよび 64 ビット整数は、DOUBLE として格納されると精度を失う場合があります。索引 SQL データ・タイプ DOUBLE の値には、特殊

数値 *NaN*、*INF*、*-INF*、*+0*、および *-0* が含まれる場合があります (SQL データ・タイプ DOUBLE 自体はこれらの値をサポートしていなくても同様です)。

INTEGER

XML スキーマ・タイプ xs:int に適合するすべての数値が変換され、 INTEGER データ・タイプで索引に格納されます。 XML スキーマには、xs:int や xs:integer などの複数の整数データ・タイプが定義されていることに注意してください。xs:int の境界は、 SQL タイプ INTEGER の境界に対応していますが、xs:integer には境界がありません。 xs:int の境界を超えた値が検出されると、エラーが返されます。

```
CREATE INDEX intidx on favorite_cds(cdinfo)
   GENERATE KEYS USING XMLPATTERN '/favoritecds/cd/year'
   AS SQL INTEGER
```

語 INT を、 SQL データ・タイプ INTEGER のシノニムとして使用できます。

DECIMAL(*integer*, *integer*)

すべての数値は変換され、DECIMAL データ・タイプで索引に格納されます。最初の整数は数値の精度、つまり数字の総桁数です。これは、1 から 31 の範囲で指定できます。 2 番目の整数は、数値の位取り、つまり、小数点の右側の桁数です。これは、0 から数値の精度までの範囲で指定できます。精度と位取りが指定されない場合、デフォルト値の 5,0 が使用されます。

```
CREATE INDEX decidx on favorite_cds(cdinfo) GENERATE KEYS USING XMLPATTERN
   '//price' AS SQL DECIMAL(5,2)
```

語 DEC、NUMERIC、および NUM を、DECIMAL のシノニムとして使用できます。

DATE

DATE データ・タイプ値は、索引に格納される前に、UTC (協定世界時) またはズールー時に正規化されます。DATE の XML スキーマ・データ・タイプは、SQL データ・タイプより精度が高い点に注意してください。範囲外の値が検出されると、エラーが戻されます。

```
CREATE INDEX BirthdateIndex ON personnel(xmlDoc)
   GENERATE KEY USING XMLPATTERN '/Person/Confidential/Birthdate' AS SQL DATE
```

TIMESTAMP

TIMESTAMP データ・タイプ値は、索引に格納される前に、UTC (協定世界時) またはズールー時に正規化されます。タイム・スタンプの XML スキーマ・データ・タイプは、SQL データ・タイプより精度が高い点に注意してください。範囲外の値が検出されると、エラーが戻されます。

```
CREATE INDEX LastLogonIndex ON machines(xmlDoc)
   GENERATE KEY USING XMLPATTERN '/Machine/Employee/LastLogon' AS SQL TIMESTAMP
```

XML データに対する索引のデータ・タイプ変換

値を XML データに対する索引に挿入する前に、まず索引 SQL データ・タイプに対応する索引 XML タイプに変換する必要があります。

VARCHAR(*integer*) および VARCHAR HASHED では、値は XQuery 関数の fn:string を使用して xs:string 値に変換されます。 VARCHAR(*integer*) の長さ属性が、変換後の xs:string 値への制約として適用されます。 VARCHAR HASHED の

索引 SQL データ・タイプは、変換後の xs:string 値にハッシュ・アルゴリズムを適用して、索引に挿入されるハッシュ・コードを生成します。 VARCHAR タイプのデータは、最初にスキーマ・データ・タイプに正規化されずに索引に直接格納されます。

DOUBLE、INTEGER、DECIMAL、DATE、および TIMESTAMP 索引では、値は XQuery キャスト式を使用して索引 XML タイプに変換されます。 DATE および TIMESTAMP データ・タイプ値は、索引に格納される前に、UTC (協定世界時) またはズールー時に正規化されます。 XQuery 規則によると有効で、システム制限のために索引データ・タイプに変換できない XML データは、索引付けエラーになります。索引 SQL データ・タイプ DOUBLE の値には、特殊数値 *NaN*、*INF*、*-INF*、*+0*、および *-0* が含まれる場合があります (SQL データ・タイプ DOUBLE 自体はこれらの値をサポートしていなくても同様です)。

対応する索引データ・タイプ

表 18. 対応する索引データ・タイプ

XML データ・タイプ	SQL データ・タイプ
xs:string	VARCHAR(*integer*) および VARCHAR HASHED
xs:double	DOUBLE
xs:date	DATE
xs:dateTime	TIMESTAMP
xs:int	INTEGER
xs:decimal	DECIMAL(*integer*, *integer*)

無効な XML 値

XML パターン値は、CREATE INDEX ステートメントの *xmlpattern-clause* で生成される索引付きの値です。

データ・タイプ DOUBLE、INTEGER、DECIMAL、DATE、および TIMESTAMP を使用する索引では、XML パターン値は XQuery キャスト式を使用して索引 XML データ・タイプに変換されます。ターゲットの索引 XML データ・タイプ用の有効な字句形式を持たない XML 値は、無効な XML 値とみなされます。

例えば、`ABC` は xs:double データ・タイプに対して無効な XML 値です。索引が無効な XML 値を処理する方法は、 CREATE INDEX ステートメントの *xmltype-clause* に REJECT INVALID VALUES オプションまたは IGNORE INVALID VALUES オプションのいずれを指定するかによって異なります。

REJECT INVALID VALUES

すべての XML パターン値が、索引 XML データ・タイプの字句定義のコンテキストにおいて有効でなければならないことを指定します。また、値は、索引 XML データ・タイプの値スペースの範囲内になければなりません。各データ・タイプの字句定義および値スペースの詳細へのリンクについては、後述の関連資料のセクションを参照してください。例えば、REJECT INVALID VALUES 節を指定して INTEGER タイプの索引を作成する場合、 3.5、3.0、3e0、'A123'、'hello' などの XML パターン値はエラーを返し

ます (SQLSTATE 23525)。索引が既に存在していれば、XML データが表に挿入されたり、表の中で更新されたりすることはありません (SQLSTATE 23525)。索引が存在しない場合、索引は作成されません (SQLSTATE 23526)。

例えば、ユーザーが数字による従業員 ID を DOUBLE データ・タイプで索引付けする索引 EMPID を作成すると仮定します。 `31201` などの数値が索引付けされます。しかし、文書のうちの 1 つにおいて、間違えて部門 ID 値の `M55` が従業員 ID 属性値の 1 つとして使用された場合、`M55` は DOUBLE 値として無効であるため、文書の挿入は失敗してエラー・メッセージが出されます。

```
CREATE INDEX EMPID ON DEPARTMENT(DEPTDOCS)
   GENERATE KEY USING XMLPATTERN '//@id' AS SQL DOUBLE
   REJECT INVALID VALUES
```

IGNORE INVALID VALUES

ターゲット索引 XML データ・タイプにとって無効な字句形式の XML パターン値を無視し、格納されている XML 文書内の対応する値を CREATE INDEX ステートメントで索引付けしないように指定します。デフォルトでは、無効値は無視されます。挿入と更新の操作では、無効な XML パターン値の索引は生成されませんが、XML 文書は表に挿入されます。このようなデータ・タイプを指定しても XML パターン値の制約とは見なされないので、エラーや警告にはなりません (特定の XML 索引データ・タイプを検索する XQuery 式は、それらの値を処理の対象にしません)。

無視できる XML パターン値についての規則は、指定された SQL データ・タイプによって決まります。

- SQL データ・タイプが文字ストリングのデータ・タイプである場合、どのような文字シーケンスでも有効であるため、XML パターン値が無視されることはありません。
- SQL データ・タイプが数値のデータ・タイプである場合、 XML データ・タイプ xs:double の字句形式に適合しない XML パターン値は無視されます。索引の数値 SQL データ・タイプに対応する XML データ・タイプの、より具体的な字句形式に XML パターン値が適合しない場合は、エラーが返されます。例えば、 SQL データ・タイプが INTEGER の場合、XML パターン値 3.5、3.0、および 3e0 は、xs:double の字句形式には適合していますが xs:int の字句形式には適合していないため、エラー (SQLSTATE 23525) が返されます。同じ索引に対する 'A123' や 'hello' などの XML パターン値は無視されます。
- SQL データ・タイプが日時のデータ・タイプである場合、対応する XML データ・タイプ (xs:date または xs:dateTime) の字句形式に適合しない XML パターン値は無視されます。

XML パターン値が、適切な字句形式に適合していない場合に、その値がデータ・タイプの値スペースの外にあったり、指定された SQL データ・タイプの最大長または精度や位取りを超過していたりすると、エラーが返されます。索引が存在しない場合、索引は作成されません (SQLSTATE 23526)。

データ・タイプに対する無効な XML パターン値が無視される場合、ユーザーは同じ XML 列に対してデータ・タイプの異なる複数の索引が可能なので、ターゲットの索引 XML データ・タイプはフィルターのような働き

をし、制約とはなりません。例えば、ユーザーが、同じパターンにデータ・タイプが異なる 2 つの索引を作成すると仮定します。索引 ALLID は、VARCHAR データ・タイプを使用し、文書内のすべての ID (部門 ID と従業員 ID の両方) に索引付けをします。索引 EMPID は数値の従業員 ID のみを索引付けし、DOUBLE データ・タイプをフィルターとして使用します。

明示的な IGNORE INVALID VALUES オプションの使用

```
CREATE INDEX ALLID ON DEPARTMENT(DEPTDOCS)
   GENERATE KEY USING XMLPATTERN '//@id' AS SQL VARCHAR(10)
   IGNORE INVALID VALUES

CREATE INDEX EMPID ON DEPARTMENT(DEPTDOCS)
   GENERATE KEY USING XMLPATTERN '//@id' AS SQL DOUBLE
   IGNORE INVALID VALUES
```

デフォルトを使用する論理的に同等のステートメント

```
CREATE INDEX ALLID ON DEPARTMENT(DEPTDOCS)
   GENERATE KEY USING XMLPATTERN '//@id' AS SQL VARCHAR(10)

CREATE INDEX EMPID ON DEPARTMENT(DEPTDOCS)
   GENERATE KEY USING XMLPATTERN '//@id' AS SQL DOUBLE
```

部門 ID 値 `M25` は有効な VARCHAR データ・タイプ値で、索引 ALLID に挿入されます。ただし、`M25` は DOUBLE データ・タイプに変換できないため、値は EMPID には挿入されず、エラーまたは警告は出されません。値は、表に格納されている文書に対して挿入されます。

値 `M25` は DOUBLE の索引 EMPID には存在しないものの、`M25` が含まれる文書はアクセスされないため、照会で DOUBLE の索引を使用してすべての一致する数値を取得でき、変換エラーは発生しません。

ただし、照会で DOUBLE の索引 EMPID を使用せず、`//@id=25` 述部を使用して文書をスキャンする場合には、値 `M25` がパターンに一致し、文書にまだ存在するものの、数値ではないので、変換エラーが発生します。

文書内のすべての値は、xs:string (SQL VARCHAR) データ・タイプに有効である点に注意してください。

文書リジェクトまたは CREATE INDEX ステートメント失敗

以下のタイプの索引付けエラーの場合、INSERT または UPDATE ステートメントでは XML 文書はリジェクトされます (SQLSTATE 23525、sqlcode -20305)。データが入っている表に対する CREATE INDEX ステートメントで、XML 文書がその表に既に存在する場合、CREATE INDEX ステートメントは失敗し (SQLSTATE 23526、sqlcode -20306)、文書は表に格納されたままになります。

VARCHAR(*integer*) 長さ制約エラー

1 つ以上の XML パターン式から生成された索引値の長さが、VARCHAR データ・タイプのユーザー指定の長さ制約を超えています。

サポートされないリスト・データ・タイプ・ノード・エラー

XML 値の中の 1 つ以上の XML ノード値が、指定された索引で索引付けできないリスト・データ・タイプ・ノードです。リスト・データ・タイプ・ノードは、XML データに対する索引ではサポートされません。

変換エラー

ソース値が索引 XML データ・タイプに対して無効であり、 REJECT INVALID VALUES オプションが CREATE INDEX ステートメントと共に指定された場合、エラーが戻されます。また、ソース値が DB2 データベース・サーバーの内部制限のためにスキーマ・データ・タイプまたは索引 XML データ・タイプのいずれかの DB2 の表記に変換できない有効な XML 値である場合も、エラーが出されます。一貫性のある結果を維持するために、エラーが発行される必要があります。索引を使用した照会が実行される予定であった場合、値は有効な XML 値なので、照会の正しい結果にはサポートされる制限を越える値が含まれることがあります。照会から不完全な結果が戻されることがないようにするため、エラーが発行されて一貫性のある結果が維持されます。

表 19. DB2 の内部制限のいくつかの例

XML データ・タイプ	XML スキーマ	DB2 の範囲 (最小 : 最大)
xs:date	年には最大限度なし 負の日付がサポートされる	0001-01-01: 9999-12-31
xs:dateTime	年には最大限度なし 負の日付がサポートされる 小数秒では任意の精度がサポートされる	0001-01-01T00:00:00.000000Z: 9999-12-31T23:59:59.999999Z
xs:integer	最小または最大の範囲に制限なし	-9223372036854775808: 9223372036854775807

DB2 データベース・サーバーは XML 値の全範囲はサポートしません。サポートされない値範囲には以下のものが含まれます。

- 年が 9999 より大または 0 より小である date または dateTime 値
- 小数秒の精度が 6 桁より大である date または dateTime 値
- 範囲外の数値

索引 XML データ・タイプに変換するためのサマリー表

データをターゲットの索引 XML データ・タイプに正常に変換するためには、ソース値がスキーマ・データ・タイプおよび索引 XML データ・タイプに従って字句において有効でなければならず、値がスキーマ・データ・タイプおよび索引 XML データ・タイプにおける DB2 の制限内になければなりません。

以下の表に、索引 XML データ・タイプに変換されるときに値が処理される方法を要約します。

表 20. 索引 XML データ・タイプに変換するためのサマリー表

索引 XML データ・タイプに準じた有効な値 (xs:string データ・タイプではすべての値が有効)	**DB2 の制限内の索引 XML データ・タイプの値**	**索引付けの結果**
なし	N/A	REJECT INVALID VALUES: エラー IGNORE INVALID VALUES (デフォルト): 値は無視され、索引付けされない。
あり	あり	値は索引付けされる。
あり	なし	エラー: 値は DB2 の制限外にある。

XML スキーマおよび索引キーの生成

XML スキーマを調べて、XML スキーマのデータ・タイプ仕様と一致するデータ・タイプで XML 列の索引を作成できるようにします。索引用にどの XML パターンを選択するかを決定する際には、実行する照会も考慮に入れる必要があります。

XML スキーマが使用される場合には、XML 列に格納される XML 文書の構造は、XML 文書の要素と属性のデータ・タイプが XML スキーマによって制約されるよう、妥当性検査されます。文書がスキーマの仕様と一致しない場合、文書はパーサーによりリジェクトされます。例えば、スキーマにおいて、属性が DOUBLE データ・タイプに制約されることを指定していて、文書の属性の値が *ABC* である場合には、文書はリジェクトされます。 XML スキーマが使用されない場合には、文書データはパーサーにより妥当性検査されず、型なしデータと見なされます。

例えば、次の XML スキーマが使用されると想定します。

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<xsd:element name="product" type="ProdType"/>

<xsd:simpleType name="ColorType">
  <xsd:restriction base="xsd:string">
    <xsd:maxLength value='20'/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:complexType name="ProdType">
  <xsd:sequence>
    <xsd:element name="name"    type="xsd:string" />
    <xsd:element name="SKU"     type="xsd:string" />
    <xsd:element name="price"   type="xsd:integer" />
    <xsd:element name="comment" type="xsd:string" />
  </xsd:sequence>
  <xsd:attribute name="color"   type="ColorType" />
  <xsd:attribute name="weight"  type="xsd:integer" />
</xsd:complexType>
</xsd:schema>
```

発行する必要がある照会を確認した後、price および color に索引が必要であると決定するかもしれません。照会を分析すると、CREATE INDEX ステートメントにど

のような XML パターン式を組み込むかを決定する助けになります。 XML スキーマから、索引にどのデータ・タイプを選択すべきかについての指針が得られます。例えば、price 要素は 10 進数なので索引 *priceindex* には DECIMAL の数値データ・タイプを選択でき、 color 属性はストリングなので索引 *colorindex* には VARCHAR のデータ・タイプを選択できるということがわかります。

```
XQUERY for $i in db2-fn:xmlcolumn('COMPANY.PRODUCTDOCS')/product[price > 5.00]
   return $i/name
XQUERY for $i in db2-fn:xmlcolumn('COMPANY.PRODUCTDOCS')/product[@color = 'pink']
   return $i/name
```

```
CREATE INDEX priceindex on company(productdocs)
      GENERATE KEY USING XMLPATTERN '/product/price' AS DECIMAL(7,2)
CREATE INDEX colorindex on company(productdocs)
      GENERATE KEY USING XMLPATTERN '//@color' AS SQL VARCHAR(80)
```

スキーマでは、ストリング・データ・タイプの他の制約も指定できます。例えばこの例では、*ColorType* の下に示されている *maxLength* で、ストリングがユニコードで 20 文字に制限されます。 CREATE INDEX ステートメントは VARCHAR の長さを文字数ではなくバイト数で指定するため、スキーマの長さを 4 の係数で乗算し、索引のスキーマで許可される最長のストリングを格納するために必要な最大のバイト数を計算することができます。この事例では、4*20 = 80 であるため、*colorindex* には VARCHAR(80) が選択されます。

スキーマがストリング・データ・タイプの長さ制限を指定しておらず、文書内の値の最大ストリング長が不明である場合、索引により使用されるページ・サイズにより許可される最大長を使用できます。索引はさまざまな長さのストリングを格納しますが、各ストリングに必要な実際のバイト数のみが格納されるため、必要以上に長い最大長を指定した場合の保管ペナルティーはありません。ただし、索引スキャン中に最大キー・サイズを処理するために、メモリー内により大きなキー・バッファーを割り振る必要があります。 VARCHAR データ・タイプを指定する XML 列の索引に許可される最大長のリストについては、CREATE INDEX ステートメントを参照してください。

VARCHAR データ・タイプの最大長が文書値に索引付けするほど長くない場合には、長さ制限がない VARCHAR HASHED データ・タイプを使用できます。ただし、VARCHAR HASHED を使用する索引は、等価検索のみに使用でき、範囲スキャンには使用できません。 VARCHAR(*integer*) で指定した長さより長いストリングを含む文書はリジェクトされる点に注意してください。

XML スキーマでは、属性および要素のデフォルト値を指定することもできます。対応する値が XML 文書に指定されておらず、文書が妥当性検査される場合には、文書を保管する際にスキーマのデフォルト値が使用されます。これらのデフォルト値は、元の入力文書にあった他の値と共に索引付けされます。文書が妥当性検査されない場合には、デフォルト値は文書に追加されず、索引付けされません。

UNIQUE キーワードの意味体系

XML 以外の列の索引に対して使用されるものと同じ UNIQUE キーワードが、XML 列の索引にも使用されますが、その意味は異なります。

リレーショナル索引では、CREATE INDEX ステートメントの UNIQUE キーワードは、表のすべての行を強制的にユニークにします。 XML データに対する索引で

は、UNIQUE キーワードは、 XML パターンでノードが修飾されたすべての文書を、単一の XML 列内で強制的にユニークにします。 1 つの文書の挿入は、ユニーク索引への複数の値の挿入を生じさせる場合があります。これらの値は、その文書内で、また同一 XML 列内の他のすべての文書の中で固有でなければなりません。なお、いくつかの文書を挿入しても、索引にはまったく値が挿入されない場合があることにも注意してください。このような文書のユニーク性は強制されません。

索引に対して指定された SQL データ・タイプに XML 値が変換された後、索引のデータ・タイプ、ノードの XML パス、およびノードの値にユニーク性が強制されます。

UNIQUE キーワードを指定する場合には注意が必要です。索引に対して指定したデータ・タイプへの変換により、精度または範囲の損失が起きる可能性があり、同じキー値に複数の異なる値がハッシュされる可能性もあるため、XML 文書の中で固有に見える複数の値が、重複キー・エラーを引き起こすおそれもあります。次の状態の時に重複キー・エラーが発生する可能性があります。

- VARCHAR HASHED が指定されている場合、固有の文字ストリングが同じハッシュ・コードにハッシュされて重複キー・エラーとなる場合があります。
- タイプ DOUBLE の索引の場合、精度の低下または DOUBLE データ・タイプの範囲を超えた値により、挿入時に重複キー・エラーが発生する場合があります。例えば、BIGINT および無制限の 10 進値は、DOUBLE データ・タイプとして索引に格納されると、精度が低下します。これらの種類のデータは、それぞれタイプ INTEGER または DECIMAL の索引を作成することをお勧めします。

VARCHAR(*integer*) が指定された場合、誤った重複キー・エラーが発生しないように、XML 文書からの文字ストリング全体が索引に格納されます。さらに、文字ストリングの固有性は、末尾ブランクに意味がある XQuery の意味体系に従います。そのため、SQL では重複であっても末尾ブランクでは異なる値は、XML データに対する索引では固有値と見なされます。

```
CREATE UNIQUE INDEX EMPINDEX ON company(companydocs)
    GENERATE KEY USING XMLPATTERN '/company/emp/name/last' AS SQL
    VARCHAR(100)
```

UNIQUE 索引の場合、XML パターンは単一の完全パスを指定する必要があり、以下のいずれも含まれない可能性があります。

- descendant 軸
- descendant-or-self 軸
- /descendant-or-self::node()/ (//)
- XML 名テストにおけるワイルドカード
- XML 種類テストにおける node() または processing instruction()

XML データの索引付けに関連したデータベース・オブジェクト

XML データに対する論理的および物理的な索引

XML データに対する索引を作成する際、論理索引および物理索引という 2 つの B ツリー索引が作成されます。

論理索引には、CREATE INDEX ステートメントで指定された XML パターン情報が含まれています。物理索引には、論理索引をサポートするための DB2 により生成されたキー列があり、索引付き文書値が含まれています。その文書値は、CREATE INDEX ステートメントの *xmltype-clause* で指定されたデータ・タイプに変換されています。

XML データに対する索引は、論理レベルで (CREATE INDEX や DROP INDEX ステートメントなどで) 処理を行います。基礎となる物理索引の DB2 による処理は、ユーザーには認識されません。物理索引は、索引メタデータを戻すアプリケーション・プログラミング・インターフェースでは認識されない点に注意してください。

SYSCAT.INDEXES カタログ・ビューでは、論理索引には CREATE INDEX ステートメントで指定した索引名があり、その索引タイプは *XVIL* です。物理索引はシステム生成名と *XVIP* の索引タイプを持っています。論理索引は常に作成され、最初に索引 ID (IID) が割り当てられます。物理索引はその後即時に作成され、連続した次の索引 ID が割り当てられます。

論理索引と物理索引の関係が次の例に示されています。ここでは、*EMPINDEX* および *IDINDEX* という 2 つの XML データに対する索引について考慮してみます。*EMPINDEX* では、論理索引は、名前 *EMPINDEX*、索引 ID *3*、および索引タイプ *XVIL* を持っています。対応する物理索引は、システム生成名 *SQL060414134408390*、索引 ID *4*、および索引タイプ *XVIP* を持っています。

表 21. 論理索引と物理索引の関係

索引名 (INDNAME)	索引 ID (IID)	表名 (TABNAME)	索引タイプ (INDEXTYPE)
SQL060414133259940	1	COMPANY	XRGN
SQL060414133300150	2	COMPANY	XPTH
EMPINDEX	3	COMPANY	XVIL
SQL060414134408390	4	COMPANY	XVIP
IDINDEX	5	COMPANY	XVIL
SQL060414134408620	6	COMPANY	XVIP

カタログ・ビュー

以下の各カタログ・ビューの詳細については、『関連参照』のセクションを参照してください。

SYSCAT.INDEXES
各行は索引 (XML データに対する論理的および物理的な索引を含む) を表します。

SYSCAT.INDEXXMLPATTERNS
各行は、XML データに対する索引内のパターン節を表します。

データベース・パーティション環境における索引

CREATE INDEX ステートメントと ALTER INDEX ステートメントは、任意のデータベース・パーティションから発行できます。

複数のデータ・パーティションにパーティション化されている表で索引を作成する場合、その索引も複数のデータ・パーティションでパーティション化されます。

監査

XML 列の索引は、監査のために既存の索引オブジェクト・タイプを使用します。論理索引のみが監査され、物理索引はされません。

XML 列に関連付けられた他のデータベース・オブジェクト

XML 列に関連し、内部およびシステムで生成される 2 つの索引があります。これらの索引は、SYSCAT.INDEXES カタログ・ビューに表示されます。さらに、XML データを含むデータ・パーティション表の索引パーティションが、SYSCAT.INDEXPARTITIONS カタログ・ビューに表示されます。

XML パスの索引および XML 領域の索引

XML 列を作成するたびに、DB2 により、XML 列上に XML パスの索引が自動的に作成されます。さらに DB2 は、表内のすべての XML 列に対して XML 領域の単一の索引も作成します。

XML パスの索引は、XML 列内に格納された XML 文書内に存在するすべての固有のパスを記録します。

XML 領域の索引は、XML 文書が内部でどのように領域 (ページ内のノードのセット) に分割されているかをキャプチャーします。領域はページ内のノードのセットであるため、ページ内により多くのノードを保管できるより大きなページ・サイズが使用された場合には、領域索引項目の数を減らして、パフォーマンスを向上させることができます。

データ・パーティション表の場合、XML 領域の索引は常にパーティション化索引になり、XML 列のパス索引は常に非パーティション化索引になります。

SYSCAT.INDEXES の XML 領域索引は常にパーティション化されており、SYSINDEX の単一項目は論理的な表現であるため、この領域索引の表スペース ID およびオブジェクト ID は論理値になります。表スペース ID およびオブジェクト ID は、関連付けられたパーティション表の ID と等しくなります。

パーティション表のデータ・パーティションと関連付けられたすべての索引パーティションでは、SYSIBM.SYSINDEXPARTITIONS カタログ表に、その索引パーティションに関する情報および統計を含む項目があります。この情報には、SYSCAT.INDEXPARTITIONS カタログ・ビューを介してアクセスします。非パーティション索引については、このカタログ表に載りません。

XML パスおよび XML 領域両方の索引は SYSCAT.INDEXES に記録されます。これらの索引は、索引メタデータを戻すアプリケーション・プログラミング・インターフェースでは認識されない点に注意してください。

XML 列に関連したこれらの内部索引は、XML データに対するユーザー作成の索引とは異なっています。例えば XML 列に格納されている XML データを索引付けする場合、XML 列の論理索引のみを、CREATE INDEX および DROP INDEX などのステートメントを使用して処理します。

カタログ・ビュー

以下のカタログ・ビューの詳細については、『関連参照』のセクションを参照してください。

SYSCAT.INDEXES
各行は索引 (XML パスおよび XML 領域の索引を含む) を表します。 XML パスの索引は SYSCAT.INDEXES.INDEXTYPE に *XPTH* として示され、XML 領域の索引は SYSCAT.INDEXES.INDEXTYPE に *XRGN* として示されます。

SYSCAT.INDEXPARTITIONS
各行は、パーティション表のデータ・パーティションと関連付けられた索引パーティションを表します。

XML データに対する索引の再作成

XML データに対する索引は以下の場合に再作成されます。

- **REORG INDEX** コマンドまたは **REORG INDEXES** コマンドの実行時。
- **REORG TABLE** コマンドの実行時。
- **IMPORT** コマンドを REPLACE オプションを指定して発行した場合。
- 照会、挿入、削除、または更新操作で表または索引へのアクセスを試行し、無効とマークされた索引オブジェクトを検出した場合

使用上の注意

- ネイティブ XML データ・ストア機能に関連したすべての索引は、表用の同じ索引オブジェクトにリレーショナル索引として入っています。これには、存在するすべての XML パスの索引、XML 領域の索引、および XML データに対する索引が含まれます。単一索引は単独では再作成されません。索引の再作成が必要になった場合、索引オブジェクト内のすべての索引はまとめて再作成されます。
- パーティション表に対する **REORG TABLE** コマンドで LOBGLOBDATA オプションが指定されていない場合は XML パスの索引は作成されません。

CREATE INDEX

CREATE INDEX ステートメントは、DB2 表に対する索引を定義する場合に使用します。索引は、XML データまたはリレーショナル・データに対して定義できます。また CREATE INDEX ステートメントは、SPECIFICATION ONLY 指定の索引 (データ・ソース表に索引があることをオプティマイザーに通知するメタデータ) を作成する場合にも使用します。

呼び出し方法

このステートメントは、アプリケーション・プログラムに組み込んだり、動的 SQL ステートメントを使用して発行したりすることができます。これは、DYNAMICRULES の実行動作がパッケージに効力を持つ場合にのみ、動的に準備できる実行可能ステートメントです (SQLSTATE 42509)。

許可

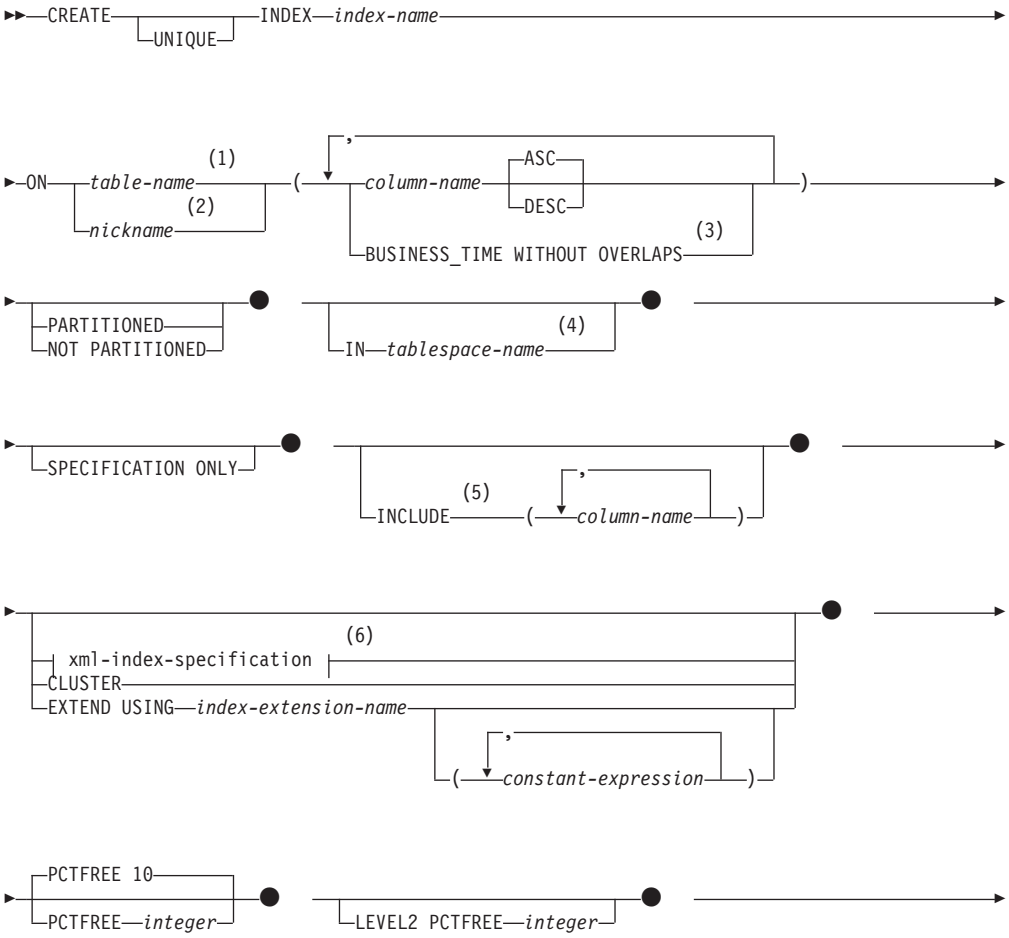
ステートメントの許可 ID によって保持されている特権には、少なくとも以下のいずれかの権限が含まれていなければなりません。

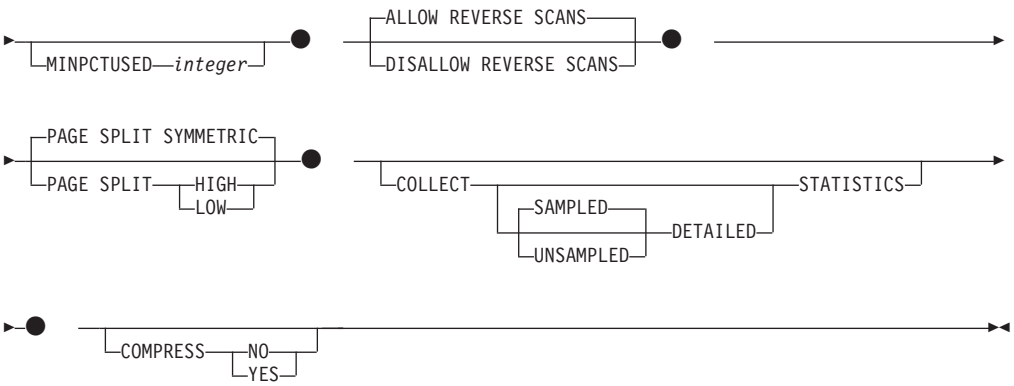
- 以下のいずれか
  - その索引の定義されている表またはニックネームに対する CONTROL 特権。
  - その索引の定義されている表またはニックネームに対する INDEX 特権。

  および以下のいずれか
  - データベースに対する IMPLICIT_SCHEMA 権限 (索引の暗黙または明示のスキーマ名が存在しない場合)
  - スキーマに対する CREATEIN 特権 (索引のスキーマ名が既存のスキーマを指している場合)
- DBADM 権限

宣言済み一時表の索引を作成するのに、明示特権は必要ありません。

構文





注:

1 フェデレーテッド・システムでは、*table-name* には、フェデレーテッド・データベース内の表を指定します。データ・ソース表を指定することはできません。

2 *nickname* を指定すると、 CREATE INDEX ステートメントは、SPECIFICATION ONLY 指定の索引を作成します。この場合、INCLUDE、*xml-index-specification*、CLUSTER、EXTEND USING、PCTFREE、MINPCTUSED、DISALLOW REVERSE SCANS、ALLOW REVERSE SCANS、PAGE SPLIT、または COLLECT STATISTICS は指定できません。

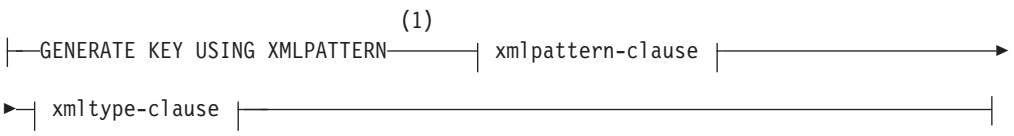
3 BUSINESS_TIME WITHOUT OVERLAPS 節は UNIQUE が指定されている場合のみ指定できます。

4 IN *tablespace-name* 節は、パーティション表の非パーティション索引に対してのみ、指定できます。

5 INCLUDE 節は UNIQUE が指定されている場合のみ指定できます。

6 *xml-index-specification* が指定された場合、*column-name* DESC、INCLUDE、または CLUSTER は指定できません。

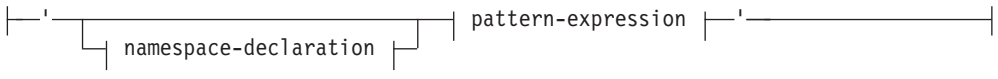
xml-index-specification:



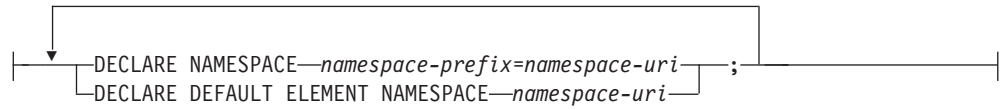
注:

1 代替構文 GENERATE KEYS USING XMLPATTERN が使用できます。

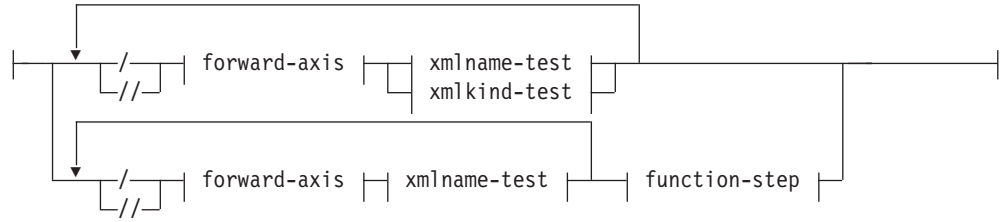
xmlpattern-clause:



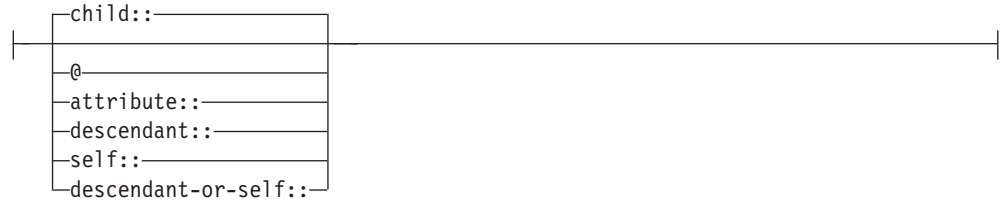
namespace-declaration:



pattern-expression:



forward-axis:

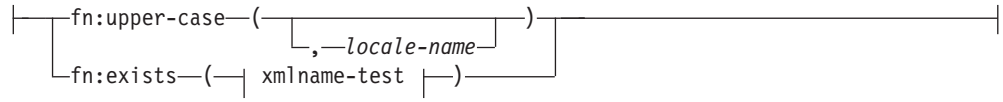


xmlname-test:

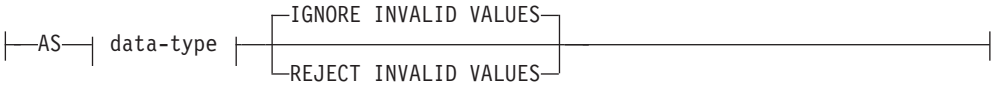
xml-wildcard:

xmlkind-test:

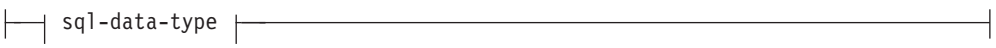
function-step:



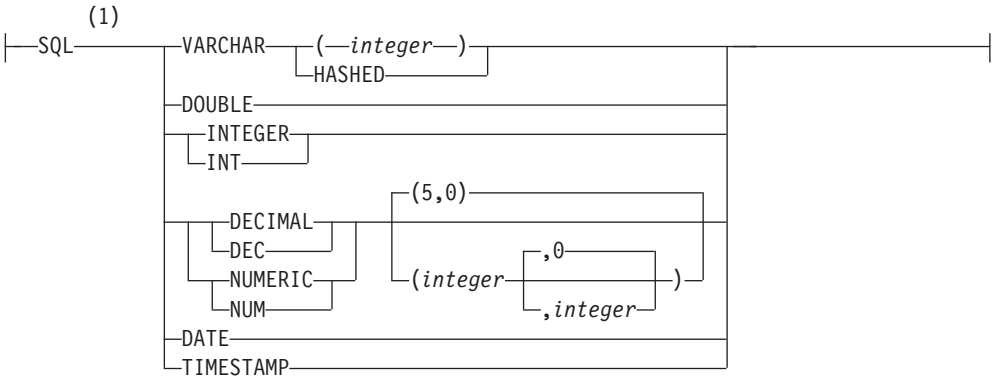
xmltype-clause:



data-type:



sql-data-type:



注:

1 XML パターンの末尾に fn:upper-case のような関数名を指定した場合、サポートされる索引のデータ・タイプは、ここに示す索引のデータ・タイプのサブセットになる場合があります。有効な索引のデータ・タイプは、xmlpattern-clause の説明の中で確認できます。

説明

UNIQUE

ON *table-name* を指定する場合、 UNIQUE により、表には索引キーの値が同じである複数の行を含めることができなくなります。行の更新、または新しい行の挿入を行う SQL ステートメントの終了時に、固有性が確保されます。

この固有性は、CREATE INDEX ステートメントの実行の過程でも検査されます。重複するキー値を含む行が既に表に含まれている場合、索引は作成されません。

索引が XML 列にある (つまり索引が XML データに対する索引である) 場合、表のすべての行について、*pattern-expression* が指定された値に対して固有性が適用されます。指定された *sql-data-type* へ値が変換された後、固有性がそれぞれの値に強制されます。指定された *sql-data-type* への変換によって精度や範囲に関する欠落が生じることや、異なる値がハッシュ化されたときに同じキー値になる可能性があるので、XML 文書で固有な値に見える値でも「キーが重複している」というエラーになる場合があります。文字ストリングの固有性は、末尾

のブランクも意味を持つ XQuery のセマンティクスに依存しています。そのため、SQL では重複であっても末尾ブランクでは異なる値は、XML データに対する索引では固有値と見なされます。

UNIQUE を使用する場合、NULL 値は他の値と同様に扱われます。例えば、キーが NULL 可能の単一列である場合、その列では 1 つの NULL 値しか含めることができません。

UNIQUE オプションの指定があり、しかも表に分散キーがある場合、索引キーの列は分散キーのスーパーセットである必要があります。つまり、ユニーク索引キーに対して指定される列は、分散キーのすべての列を含んでいる必要があります (SQLSTATE 42997)。

UNIQUE オプションの指定があり、しかも表に表パーティション・キーがある場合、索引キーの列は表パーティション・キーのスーパーセットである必要があります。つまり、ユニーク索引キーに対して指定される列は、表パーティション・キーのすべての列を含んでいる必要があります (SQLSTATE 42990)。

主キーまたはユニーク・キーは、ディメンションのサブセットにはなりません (SQLSTATE 429BE)。

ON *nickname* が指定されている場合、索引キーのデータ・ソース表の各行に固有値が含まれている場合に限り、UNIQUE を指定するようにします。固有性は検査されません。

XML データに対する索引の場合、UNIQUE は、*pattern-expression* のコンテキスト・ステップが単一の完全パスを指定し、descendant または descendant-or-self 軸、「//」、*xml-wildcard*、*node()*、または *processing-instruction()* を含まない場合にのみ指定できます (SQLSTATE 429BS)。

パーティション・データベース環境では、1 つ以上の XML 列がある表に以下の規則が適用されます。

- 分散表は、XML データに対するユニーク索引を持つことができません。
- XML データに対するユニーク索引は、分散キーのない、単一ノード複数パーティション・データベース上の表のみでサポートされます。
- XML データに対するユニーク索引が表にある場合、その表に変更を加えて分散キーを追加することはできません。

INDEX *index-name*

索引または SPECIFICATION ONLY 指定の索引を指定します。名前 (暗黙または明示の修飾子を含む) は、カタログに記述されている索引または SPECIFICATION ONLY 指定の索引、または宣言済み一時表の既存の索引を指定するものであってはなりません (SQLSTATE 42704)。修飾子は、SYSIBM、SYSCAT、SYSFUN、または SYSSTAT であってはなりません (SQLSTATE 42939)。

宣言済みグローバル一時表の索引の暗黙または明示の修飾子は、SESSION でなければなりません (SQLSTATE 428EK)。

ON *table-name* **または** *nickname*

table-name には、索引を作成する対象の表を指定します。表は、基本表 (ビューではない) か、作成済み一時表か、宣言済み一時表か、現行のサーバーに存在するマテリアライズ照会表か、または宣言済み一時表でなければなりません。宣言済み一時表の名前は、SESSION によって修飾される必要があります。

table-name に、カタログ表を指定することはできません (SQLSTATE 42832)。 UNIQUE が指定されており、 *table-name* が型付き表である場合には、副表を指定することはできません (SQLSTATE 429B3)。

nickname は、SPECIFICATION ONLY 指定の索引を作成する対象のニックネームです。この *nickname* により、索引が SPECIFICATION ONLY 指定の索引によって記述されているデータ・ソース表、またはそのような表に基づくデータ・ソース・ビューのいずれかが参照されます。この *nickname* は、カタログにリストされていなければなりません。

column-name
索引の場合、*column-name* には索引キーを構成する列を指定します。 SPECIFICATION ONLY 指定の索引の場合、*column-name* は、フェデレーテッド・サーバーがデータ・ソース表の列を参照するときの名前になります。

各 *column-name* は、表の列を指定する非修飾名でなければなりません。指定期間数の 2 倍に列数を足した値は 64 を超えてはなりません (SQLSTATE 54008)。*table-name* が型付き表の場合、列の数は 63 を超えることはできません (SQLSTATE 54008)。*table-name* が副表であるならば、副表内の少なくとも 1 つの *column-name* はスーパー表から継承するのではなく、新たに指定する必要があります (SQLSTATE 428DS)。同じ *column-name* を使用することはできません (SQLSTATE 42711)。

指定された列の保管長の合計は、ページ・サイズに対する索引キーの長さの上限を超えてはなりません。キー長の制限については、『SQL の制限』を参照してください。 *table-name* が型付き表である場合は、索引キーの長さ制限は 4 バイト減ります。この長さの上限は、列のデータ・タイプや列が NULL 可能か否かによって変動するシステムのオーバーヘッドにより、より小さい値になることがあります。この制限に影響を与えるオーバーヘッドの詳細については、『CREATE TABLE』の『バイト・カウント』を参照してください。

この長さは、列のデータ・タイプや NULL 可能か否かによって変動するシステムのオーバーヘッドにより、より小さい値になることがあります。この制限に影響を与えるオーバーヘッドの詳細については、『CREATE TABLE』の『バイト・カウント』を参照してください。

列の長さ属性がページ・サイズに対する索引キーの長さの上限を超えない場合でも、LOB 列、または LOB に基づく特殊タイプの列は、索引の一部として使用できません (SQLSTATE 54008)。構造化タイプ列は、EXTEND USING 節も指定されている場合にのみ指定できます (SQLSTATE 42962)。 EXTEND USING 節が指定される場合、列は 1 つしか指定できず、列のタイプは構造化タイプか、あるいは LOB に基づいていない特殊タイプでなければなりません (SQLSTATE 42997)。

索引が 1 つの列しか持たず、その列が XML データ・タイプを持ち、GENERATE KEY USING XMLPATTERN 節も指定されている場合、索引は XML データに対する索引になります。 XML データ・タイプを持つ列は、GENERATE KEY USING XMLPATTERN 節も同時に指定されている場合にのみ指定できます (SQLSTATE 42962)。 GENERATE KEY USING XMLPATTERN 節が指定される場合、列は 1 つしか指定できず、列のタイプは XML でなければなりません。

ASC
索引項目が、列の値の昇順で保持されるように指定します。これがデフォルト設定です。 ASC は、EXTEND USING で定義される索引に指定することはできません (SQLSTATE 42601)。

DESC
索引項目が、列の値の降順で保持されるように指定します。 DESC は、EXTEND USING で定義される索引に指定することはできません。また、索引が XML データに対する索引である場合にも指定できません (SQLSTATE 42601)。

BUSINESS_TIME WITHOUT OVERLAPS
BUSINESS_TIME WITHOUT OVERLAPS は、UNIQUE として定義された索引に対してのみ指定できます (SQLSTATE 428HW)。これは、残りの指定キーの値が、任意の期間においてユニークになることを示します。 BUSINESS_TIME WITHOUT OVERLAPS は、リストの最後の項目としてのみ指定できます。BUSINESS_TIME WITHOUT OVERLAPS を指定すると、期間 BUSINESS_TIME の終了列および開始列が自動的に昇順で索引キーに追加され、時間の重なり合いがないように強制されます。BUSINESS_TIME WITHOUT OVERLAPS を指定する際は、期間 BUSINESS_TIME の列をキー列として、パーティション・キーの列として、または分散キーの列として指定することはできません (SQLSTATE 428HW)。

PARTITIONED
パーティション索引を作成する必要があることを示します。 *table-name* は、データ・パーティションを指定して定義されている表を示していなければなりません (SQLSTATE 42601)。

表がパーティション化されており、PARTITIONED と NOT PARTITIONED がどちらも指定されていない場合、索引はパーティション化されたものとして作成されます (例外が幾つかあります)。以下のいずれかの状態が当てはまる場合には、非パーティション索引がパーティション索引の代わりに作成されます。

- UNIQUE が指定されており、かつ索引キーにすべての表パーティション・キー列が含まれているわけではない。
- 空間インデックスが作成されている。
- XML データに対する索引が定義されている。

非パーティション索引の定義と重複する定義を持つパーティション索引は、重複索引とは見なされません。詳しくは、このトピックにある 193 ページの『規則』というセクションを参照してください。

PARTITIONED キーワードは、以下の索引には指定できません。

- 非パーティション表上の索引 (SQLSTATE 42601)
- XML データに対して定義された索引 (SQLSTATE 42613)
- 索引キーにすべての表パーティション・キー列が含まれているわけではないユニーク索引 (SQLSTATE 42990)
- 空間インデックス (SQLSTATE 42997)

MQT などの、デタッチされた従属表のあるパーティション表上にはパーティション索引を作成できません (SQLSTATE 55019)。

パーティション索引の索引パーティションの表スペース配置は、以下の規則に従って判別されます。

- CREATE TABLE ステートメントの INDEX IN 節の *partition-tablespace-options* を使用して、索引化している表が作成された場合、その INDEX IN 節で指定された表スペース内に索引パーティションが作成されます。
- 索引化している表の CREATE TABLE ステートメントの INDEX IN 節に *partition-tablespace-options* が指定されていない場合は、索引化されている対応するデータ・パーティションと同じ表スペース内に、索引パーティションのパーティション索引が作成されます。

CREATE INDEX ステートメントの IN 節では、パーティション索引はサポートされていません (SQLSTATE 42601)。 CREATE TABLE ステートメントの INDEX IN 節の *tablespace-clauses* は、パーティション索引では無視されます。BUSINESS_TIME WITHOUT OVERLAPS が索引キーに指定されている場合、パーティション・キー列に期間 BUSINESS_TIME の開始列または終了列が含まれていてはなりません (SQLSTATE 428HW)。

NOT PARTITIONED
非パーティション索引が、表に関して定義されたすべてのデータ・パーティションにわたって作成されることを指定します。*table-name* は、データ・パーティションを指定して定義されている表を示していなければなりません (SQLSTATE 42601)。

パーティション索引の定義と重複する定義を持つ非パーティション索引は、重複索引とは見なされません。詳しくは、このトピックにある 193 ページの『規則』というセクションを参照してください。

非パーティション索引の表スペース配置は、以下の規則に従って判別されます。

- CREATE INDEX ステートメントの IN 節を指定した場合、非パーティション索引はその IN 節で指定された表スペースに配置されます。
- CREATE INDEX ステートメントの IN 節を指定しない場合には、非パーティション索引の表スペース配置は以下の規則に従って判別されます。
  - 索引化されている表が CREATE TABLE ステートメントの INDEX IN 節の *tablespace-clauses* を使用して作成された場合、その INDEX IN 節で指定された表スペースに非パーティション索引が配置されます。
  - 索引化されている表が CREATE TABLE ステートメントの INDEX IN 節の *tablespace-clauses* を使用しないで作成された場合、その表の最初の可視または接続済みデータ・パーティションにある表スペース内に非パーティション索引が作成されます。表の最初の可視または接続済みデータ・パーティションとは、範囲指定に基づいてソートされたデータ・パーティションのリスト中の最初のパーティションです。また、ステートメントの許可 ID にはデフォルトの表スペースに対する USE 特権は必要ありません。

IN *tablespace-name*
パーティション表に対する非パーティション索引を作成する表スペースを指定します。この節は、パーティション索引および非パーティション表の索引には指定できません (SQLSTATE 42601)。索引に専用の表スペースを指定すると、表の作成時に INDEX IN 節を使用して行った指定は、オーバーライドされます。

tablespace-name で指定する表スペースは、表のデータ表スペースと同じデータベース・パーティション・グループになければならず、パーティション表の他の

表スペースと同じスペース管理方式でなければなりません (SQLSTATE 42838)。ステートメントの許可 ID には、その表スペースに対する USE 特権が必要です。

IN 節が指定されない場合、索引は CREATE TABLE ステートメントの INDEX IN 節で指定された表スペースに作成されます。 INDEX IN 節が指定されなかった場合、表のデータ・パーティションのうち、最初の可視パーティションまたはアタッチされたパーティションの表スペースが使用されます。これは、範囲指定に基づいてソートされたデータ・パーティションのリスト中の最初のパーティションです。 IN 節が指定されなければ、ステートメントの許可 ID にはデフォルトの表スペースに対する USE 特権は必要ありません。

SPECIFICATION ONLY

nickname で参照するデータ・ソース表に適用される、SPECIFICATION ONLY 指定の索引を作成するために、このステートメントが使われることを示します。SPECIFICATION ONLY は、*nickname* を指定した場合に、指定しなければなりません (SQLSTATE 42601)。 *table-name* を指定した場合には、指定することはできません (SQLSTATE 42601)。

SPECIFICATION ONLY 指定の索引がユニーク索引に適用される場合、DB2 は、リモート表内の列値が固有であるかどうかを検査しません。リモート列値が固有でない場合、索引列を含むニックネームに対する照会が、誤ったデータやエラーを戻す可能性があります。

作成済み一時表または宣言済み一時表の索引が作成される場合には、この節は使用できません (SQLSTATE 42995)。

INCLUDE

このキーワードは、一連の索引キー列に追加する列を指定する節を、新たに指定します。この節によって組み込まれる列は、固有性を強制するために使用されることはありません。これらの組み込み列を使用して、索引のみのアクセスを実行することにより、一部の照会のパフォーマンスが向上する可能性があります。この列は、固有性を強制するために使用される列とは区別する必要があります (SQLSTATE 42711)。 INCLUDE が指定された場合には、UNIQUE を指定する必要があります (SQLSTATE 42613)。列の数および長さ属性の合計に対する制限は、ユニーク・キーと索引にあるすべての列にも適用されます。

この節は、作成済み一時表や宣言済み一時表には使用できません (SQLSTATE 42995)。

column-name

索引には組み込まれているものの、ユニーク索引キーの一部ではない列を指定します。ユニーク索引キーの列に定義された規則と同様の規則が適用されます。 *column-name* の後にキーワード ASC または DESC を指定しても構いませんが、順序に影響はありません。

INCLUDE は、EXTEND USING で定義される索引に指定することはできません。 *nickname* が指定されている場合、また索引が XML 列で定義されている場合にも指定できません (SQLSTATE 42601)。

xml-index-specification

XML 列に保管された XML 文書からどのように索引キーが生成されるかを指定

します。 *xml-index-specification* は、索引列が複数存在する場合、または列が XML データ・タイプを持たない場合は、指定できません。

この節は、XML 列にのみ適用されます (SQLSTATE 429BS)。

GENERATE KEY USING XMLPATTERN *xmlpattern-clause*
索引の対象となる XML 文書の部分を指定します。 XML パターン値は、*xmlpattern-clause* によって生成される索引付け対象値です。リスト・データ・タイプのノードは索引ではサポートされていません。ノードが *xmlpattern-clause* によって修飾され、そのノードがリスト・データ・タイプであることを示す XML スキーマが存在している場合、このリスト・データ・タイプのノードを索引付けすることはできません (CREATE INDEX ステートメントでは SQLSTATE 23526、または INSERT および UPDATE ステートメントでは SQLSTATE 23525)。

xmlpattern-clause
索引の対象となるノードを特定するパターン式が入ります。オプションの *namespace-declaration* と必須の *pattern-expression* で構成されます。

namespace-declaration
パターン式に修飾名が含まれている場合は、*namespace-declaration* を指定して名前空間接頭部を定義する必要があります。非修飾名には、デフォルトの名前空間を定義できます。

DECLARE NAMESPACE *namespace-prefix=namespace-uri*
NCName である *namespace-prefix* を、ストリング・リテラルである *namespace-uri* にマップします。*namespace-declaration* には、複数の *namespace-prefix* から *namespace-uri* へのマッピングを含めることができます。 *namespace-prefix* は、*namespace-declaration* のリストの中で固有でなければなりません (SQLSTATE 10503)。

DECLARE DEFAULT ELEMENT NAMESPACE *namespace-uri*
非修飾の要素名またはタイプに対するデフォルトの名前空間 URI を宣言します。デフォルトの名前空間が宣言されない場合、要素やタイプの非修飾名はどの名前空間にも属さないことになります。宣言できるデフォルトの名前空間は 1 つだけです (SQLSTATE 10502)。

pattern-expression
XML 文書内の、索引の対象となるノードを指定します。*pattern-expression* には、パターン・マッチング文字 (*) を入れることができます。XQuery のパス式に似ていますが、DB2 によってサポートされる XQuery 言語のサブセットをサポートしています。

/ (スラッシュ)
パス式のステップを分離します。

// (二重スラッシュ)
これは、/descendant-or-self::node()/ の短縮構文です。 *// (二重スラッシュ)* は、UNIQUE を指定した場合は使用できません。

forward-axis

child::
コンテキスト・ノードの子を指定します。これは、他のフォワード軸が指定されない場合のデフォルトです。

@ コンテキスト・ノードの属性を指定します。これは、attribute:: の短縮構文です。

attribute::
コンテキスト・ノードの属性を指定します。

descendant::
コンテキスト・ノードの子孫を指定します。 *descendant::* は、UNIQUE を指定した場合は使用できません。

self::
コンテキスト・ノード自体を単独で指定します。

descendant-or-self::
コンテキスト・ノードとそのコンテキスト・ノードの子孫を指定します。 descendant-or-self:: は、UNIQUE を指定した場合は使用できません。

xmlname-test
パス内のステップに、XML の修飾名 (xml-qname) またはワイルドカード (xml-wildcard) を使用してノード名を指定します。

xml-ncname
XML 1.0 で定義された XML 名。コロン文字を組み込むことはできません。

xml-qname
以下の 2 とおりの形式のいずれかで XML の修飾名 (QName としても知られる) を指定します。

- xml-nsprefix:xml-ncname。xml-nsprefix は、有効範囲内名前空間を特定する xml-ncname。
- xml-ncname。暗黙の xml-nsprefix としてデフォルトの名前空間が適用されるよう指定する。

xml-wildcard
xml-qname を、以下の 3 とおりの形式のいずれかでワイルドカードとして指定します。

- * (単一のアスタリスク文字)。これは、あらゆる xml-qname に対応する。
- *xml-nsprefix*:*。これは、指定の名前空間内のあらゆる xml-ncname に対応する。
- *:*xml-ncname*。これは、あらゆる有効範囲内名前空間の特定の XML 名に対応する。

UNIQUE も指定する場合、パターン式のコンテキスト・ステップで *xml-wildcard* を使用することはできません。

xmlkind-test
これらのオプションは、パターン・マッチングするノードのタイプを指定するのに使用します。ユーザーは以下のオプションを使用できます。

node()
あらゆるノードに一致します。 *node()* は、UNIQUE を指定した場合は使用できません。

text()
あらゆるテキスト・ノードに一致します。

comment()
あらゆるコメント・ノードに一致します。

processing-instruction()
あらゆる処理命令ノードに一致します。
processing-instruction() は、UNIQUE を指定した場合は使用できません。

function-step
これらの関数呼び出しは、大/小文字の区別なしなど、特殊なプロパティーを持つ索引を指定する場合に使用します。
XMLPATTERN 節ごとに 1 つの関数ステップしか許可されません。関数ステップは、エレメントまたは属性にのみ適用できます。関数ステップの直前に *xmlkind-test* オプションを配置することはできません。関数は、XMLPATTERN の途中では使用できず、最終ステップでのみ使用できます。現行では、fn:upper-case 関数と fn:exists 関数のみがサポートされています。

関数名として接頭部 fn: を指定する代わりに、別の有効なネーム・スペースを指定することもできますし、fn: を完全に省略することもできます。

fn:upper-case
強制的に索引値を大文字形式で格納します。 fn:upper-case の最初のパラメーターは必須で、コンテキスト項目式 (' . ') でなければなりません。2 番目のパラメーターはロケールで、これはオプションです。 fn:upper-case がパターンに出現する場合、サポートされる索引タイプは VARCHAR および VARCHAR HASHED のみです。

fn:exists
XML 文書内でエレメント項目または属性項目があるかどうかを検査します。項目が存在する場合、この述部は TRUE を返します。 fn:exists のパラメーターは必須で、エレメントまたは属性でなければなりません。この関数が索引パスで使用される場合、索引タイプは VARCHAR(1) として定義する必要があります。

xmltype-clause

AS *data-type*
索引値を保管前に変換するデータ・タイプを指定します。値は、指定した索引 SQL データ・タイプに対応する索引 XML データ・タイプに変換されます。

表 22. 対応する索引データ・タイプ

索引 XML データ・タイプ	索引 SQL データ・タイプ
xs:string	VARCHAR(*integer*)、VARCHAR HASHED
xs:double	DOUBLE
xs:int	INTEGER
xs:decimal	DECIMAL
xs:date	DATE
xs:dateTime	TIMESTAMP

VARCHAR(*integer*) および VARCHAR HASHED の場合、値は、XQuery 関数 fn:string を使用して xs:string 値に変換されます。VARCHAR(*integer*) の長さ属性が、変換後の xs:string 値への制約として適用されます。 VARCHAR HASHED の索引 SQL データ・タイプは、ハッシュ・アルゴリズムを変換後の xs:string 値に適用し、索引に挿入されるハッシュ・コードを生成します。

データ・タイプ DOUBLE、DATE、INTEGER、DECIMAL、および TIMESTAMP を使用する索引の場合、この値は、XQuery キャスト式を使用して索引 XML データ・タイプに変換されます。

索引が固有であれば、索引のタイプに値が変換された後、値は必ず固有になります。

data-type
以下のデータ・タイプがサポートされています。

sql-data-type
サポートされる SQL データ・タイプは以下のとおりです。

VARCHAR(*integer***)**
この書式の VARCHAR が指定されると、DB2 は *integer* を制約として使用します。索引付けされる文書ノードに、*integer* より長い値があれば、索引が既に存在する場合、文書は表に挿入されません。索引が存在しない場合、索引は作成されません。 *integer* は、1 とページ・サイズ依存の最大値の間の値です。ページ・サイズごとの最大値は、表 23 のようになります。

表 23. ページ・サイズごとの文書ノードの最大長

ページ・サイズ	文書ノードの最大長 (単位：バイト)
4KB	817
8KB	1841
16KB	3889
32KB	7985

XQuery セマンティクスがストリングの比較に使用されますが、この場合末尾ブランクも意味を持ちます。これは、SQL セマンティクス (比較時に末尾ブランクが無視される) とは異なっています。

VARCHAR HASHED
任意の長さの文字ストリングの索引作成を扱う VARCHAR HASHED を指定します。索引にされるストリングの長さには制限はありません。 DB2 は、ストリング全体に対応して 8 バイトのハッシュ・コードを生成します。これらのハッシュされた文字ストリングを使用する索引は、同等性検索にのみ使用できます。
XQuery セマンティクスが、ストリングの等価比較に使用されますが、この場合末尾ブランクは意味を持ちます。これは、SQL セマンティクス (比較時に末尾ブランクが無視される) とは異なっています。ストリング上のハッシュは、等価の XQuery セマンティクスを保持しますが、SQL セマンティクスは保持しません。

DOUBLE
データ・タイプ DOUBLE が数値の索引作成に使用されるよう指定します。無制限の DECIMAL タイプと 64 ビットの整数は、DOUBLE 値として保存される場合、精度を失う場合があります。 DOUBLE の値には、特別な数値 *NaN*、*INF*、*-INF*、*+0*、および *-0* を含めることができます。ただし、SQL データ・タイプ DOUBLE 自体はこれらの値をサポートしていません。

INTEGER
データ・タイプ INTEGER が XML 値の索引作成に使用されるよう指定します。 xs:integer の XML スキーマ・データ・タイプは、整数の SQL データ・タイプより広範囲の値を許容することに注意してください。範囲外の値が検出されると、エラーが戻されます。値が xs:double の字句形式に準拠するものの、xs:int の字句形式には準拠しない場合 (3.5、3.0、3E1 など)、エラーも返されます。

DECIMAL(*integer*, *integer*)
データ・タイプ DECIMAL が XML 値の索引作成に使用されるよう指定します。 DECIMAL タイプは、*precision* と *scale* という 2 つのパラメーターを取ります。最初のパラメーター *precision* は、総桁数を指定する、1 から 31 の範囲の値を持つ整数定数です。 2 番目のパラメーター *scale* は、ゼロ以上で precision 以下の整数定数です。 *scale* は、小数点より右側の数字の桁数を指定します。

10 進数の末尾から桁が切り捨てられることはありません。小数点文字より右側の数字の桁数が scale より多い場合、エラーが返されます。また、小数点文字の左側に

ある有効数字 (数値の整数部分) の桁数が precision よりも多い場合は、エラーが返されます。

DATE
データ・タイプ DATE が XML 値の索引作成に使用されるよう指定します。 xs:date の XML スキーマ・データ・タイプは、SQL データ・タイプに対応する DB2 pureXML xs:date データ・タイプよりも広範囲の値を許容することに注意してください。範囲外の値が検出されると、エラーが戻されます。

TIMESTAMP
データ・タイプ TIMESTAMP が XML 値の索引作成に使用されるよう指定します。 xs:dateTime の XML スキーマ・データ・タイプは、SQL データ・タイプに対応する DB2 pureXML xs:dateTime データ・タイプよりも広範囲の値および秒未満の精度を許容することに注意してください。範囲外の値が検出されると、エラーが戻されます。

IGNORE INVALID VALUES
これを指定すると、ターゲット索引の XML データ・タイプで字句形式が無効な XML パターン値は無視され、格納されている XML 文書内の対応する値の索引は CREATE INDEX ステートメントによって生成されません。デフォルトでは、無効値は無視されます。挿入と更新の操作では、無効な XML パターン値の索引は生成されませんが、XML 文書は表に挿入されます。このようなデータ・タイプを指定しても XML パターン値の制約とは見なされないので、エラーや警告にはなりません (特定の XML 索引データ・タイプを検索する XQuery 式は、それらの値を処理の対象にしません)。

どの XML パターン値を無視できるかについての規則は、指定 SQL データ・タイプによって決まります。

- SQL データ・タイプが VARCHAR(*integer*) または VARCHAR HASHED である場合、すべての文字シーケンスが有効であるため、XML パターン値が無視されることはありません。
- SQL データ・タイプが DOUBLE、DECIMAL、または INTEGER である場合、XML データ・タイプ xs:double の字句形式に準拠しない XML パターン値は無視されます。 SQL データ・タイプが DECIMAL または INTEGER で、XML パターン値が XML データ・タイプ xs:double の字句形式には準拠するものの、それぞれ xs:decimal または xs:int の字句形式には準拠しない場合、エラーが返されます。例えば、SQL データ・タイプが INTEGER である場合、XML パターン値 3.5、3.0、および 3e0 は xs:double の字句形式には準拠するものの、xs:int の字句形式には準拠しないため、エラーが返されます (SQLSTATE 23525)。「A123」、「hello」などの XML パターン値は、同じ索引において無視されます。

- SQL データ・タイプが日時のデータ・タイプである場合、対応する XML データ・タイプ (xs:date または xs:dateTime) の字句形式に適合しない XML パターン値は無視されます。

XML パターン値が、適切な字句形式に適合していない場合に、その値がデータ・タイプの値スペースの外にあったり、指定された SQL データ・タイプの最大長または精度や位取りを超過していたりすると、エラーが返されます。索引が存在しない場合、索引は作成されません (SQLSTATE 23526)。

REJECT INVALID VALUES
索引の XML データ・タイプの字句定義のコンテキストでは、すべての XML パターン値が有効でなければなりません。また、値は、索引の XML データ・タイプの値スペースの範囲に含まれていなければなりません。各データ・タイプの字句定義および値スペースの詳細を示すリンクについては、下記の『関連資料』セクションを参照してください。例えば、REJECT INVALID VALUES 節を指定するときに INTEGER タイプの索引を作成する場合、
3.5、3.0、3e0、「A123」、「hello」などの XML パターン値ではエラーが返されます (SQLSTATE 23525)。索引が既に存在していれば、XML データが表に挿入されたり、表の中で更新されたりすることはありません (SQLSTATE 23525)。索引が存在しない場合、索引は作成されません (SQLSTATE 23526)。

CLUSTER
索引を表のクラスター索引として指定します。クラスター索引のクラスター係数は、関連する表にデータが挿入されるときに、動的に保守され適切な値に調整されます。これは、この索引のキー値が同じ範囲にある行と物理的に近い位置に、新しい行の挿入を試みることによって行われます。ただし、表のクラスター索引は 1 つだけなので、 CLUSTER が表の既存の索引の定義に使用されていて、CLUSTER が指定できないということもありえます (SQLSTATE 55012)。追加モードを使用するように定義されている表では、クラスター索引を作成できない場合があります (SQLSTATE 428D8)。

CLUSTER は、*nickname* が指定されている場合、または索引が XML データに対する索引である場合は使用できません (SQLSTATE 42601)。この節は、作成済み一時表や宣言済み一時表 (SQLSTATE 42995)、あるいは範囲がクラスター化された表 (SQLSTATE 429BG) では使用できません。

EXTEND USING *index-extension-name*
この索引を管理するのに使用する *index-extension* を指定します。この節を指定する場合、1 つだけ *column-name* を指定しなければならず、この列は構造化タイプまたは特殊タイプでなければなりません (SQLSTATE 42997)。
index-extension-name (索引拡張名) は、カタログに記述されている索引拡張を指定する名前でなければなりません (SQLSTATE 42704)。特殊タイプの場合には、列が、索引拡張でソース・キー・パラメーターに対応するタイプと完全に一致していなければなりません。構造化タイプ列では、対応するソース・キー・パラメーターのタイプが、列タイプのタイプまたはスーパータイプと同じでなければなりません (SQLSTATE 428E0)。

この節は、作成済み一時表や宣言済み一時表には使用できません (SQLSTATE 42995)。

この節は DB2 pureScale®環境では使用できません (SQLSTATE 56038)。

constant-expression
索引拡張に必要な引数の値を指定します。各式は、対応する索引拡張パラメーターの定義されたデータ・タイプ (長さまたは精度、およびスケールも含む) に完全に一致するデータ・タイプを持つ定数値でなければなりません (SQLSTATE 428E0)。この節は、データベース・コード・ページ内で、32 768 バイト以内の長さでなければなりません (SQLSTATE 22001)。

PCTFREE *integer*
索引を構築する際に、各索引ページで何 % をフリー・スペースとして残すかを指定します。ページの最初の項目は、制限なしで追加されます。索引ページに項目を追加する場合には、各ページに少なくとも *integer* パーセントをフリー・スペースとして残します。 *integer* の値は 0 から 99 です。 10 よりも大きな値を指定しても、非リーフ・ページには 10% のフリー・スペースしか残されません。

PCTFREE の明示的な値が指定されず、**DB2_INDEX_PCTFREE_DEFAULT** が設定されていない場合、PCTFREE のデフォルト値が 10 になります。

PCTFREE は、*nickname* が指定されている場合は使用できません (SQLSTATE 42601)。この節は、作成済み一時表や宣言済み一時表には使用できません (SQLSTATE 42995)。

LEVEL2 PCTFREE *integer*
索引を作成する際に、索引レベル 2 の各ページで何 % をフリー・スペースとして残すかを指定します。 *integer* の値は 0 から 99 です。 LEVEL2 PCTFREE が設定されない場合は、すべての非リーフ・ページに最低 10 % または PCTFREE で指定された分 (%) のフリー・スペースが残されます。 LEVEL2 PCTFREE が設定された場合は、 *integer* で指定された分 (%) のフリー・スペースがレベル 2 の中間ページに残され、レベル 3 以上の中間ページには最低 10 % または *integer* で指定された分 (%) のフリー・スペースが残されます。

nickname が指定されている場合は、LEVEL2 PCTFREE は使用できません (SQLSTATE 42601)。この節は、作成済み一時表や宣言済み一時表には使用できません (SQLSTATE 42995)。

MINPCTUSED *integer*
索引のリーフ・ページをオンラインでマージするかどうか、および索引のリーフ・ページで使用されるスペースの最小パーセンテージの限界値を指定します。索引のリーフ・ページからキーを除去した後、そのページで使用されているスペースのパーセントが *integer* のパーセントを下回る場合、このページにある残りのキーを近隣のページのキーにマージするよう試行されます。いずれかのページに十分なスペースがあれば、マージが行われ、いずれかのページが削除されます。 *integer* の値は 0 から 99 です。パフォーマンス上の理由のため、50 以下の値をお勧めします。このオプションを指定すると、更新および削除のパフォーマンスに影響があります。排他的表ロックが掛けられている場合、更新および削除操作の実行中に限りマージされます。排他的表ロックがない場合には、更新および削除操作の間にキーは疑似的に削除されたものとしてマークされ、マージは実行されません。リーフ・ページをマージするには、CREATE INDEX の MINPCTUSED を使うのではなく、 REORG INDEXES の CLEANUP ONLY ALL オプションを使用することを考慮してください。

MINPCTUSED は、*nickname* が指定されている場合は使用できません (SQLSTATE 42601)。この節は、作成済み一時表や宣言済み一時表には使用できません (SQLSTATE 42995)。

DISALLOW REVERSE SCANS
索引において、前方向スキャン、すなわち索引作成時に定義された順序でのスキャンだけをサポートすることを指定します。

DISALLOW REVERSE SCANS は *nickname* と同時には指定できません (SQLSTATE 42601)。

ALLOW REVERSE SCANS
索引が前方向スキャンと反対方向スキャンの両方、すなわち、索引作成時に定義された順序での索引のスキャンと、その反対の順序でのスキャンをサポートすることを指定します。

ALLOW REVERSE SCANS は *nickname* と同時には指定できません (SQLSTATE 42601)。

PAGE SPLIT
索引分割の振る舞いを指定します。デフォルトは SYMMETRIC です。

SYMMETRIC
ページを大まかに半分で分割するよう指定します。

HIGH
索引キーの値が特定のパターンに従って挿入されている場合に、索引ページのスペースを効率的に使う索引ページの分割の振る舞いを指定します。索引キー値のサブセットについては、索引の左端の列 (複数の場合もある) には常に同じ値が含まれていなければならず、索引の右端の列 (複数の場合もある) には挿入ごとに増加する値が含まれていなければなりません。

LOW
索引キーの値が特定のパターンに従って挿入されている場合に、索引ページのスペースを効率的に使う索引ページの分割の振る舞いを指定します。索引キー値のサブセットについては、索引の左端の列 (複数の場合もある) には常に同じ値が含まれていなければならず、索引の右端の列 (複数の場合もある) には挿入ごとに減少する値が含まれていなければなりません。

COLLECT STATISTICS
索引の作成時に基本索引統計が収集されるように指定します。

SAMPLED
これを指定すると、索引項目を処理して拡張索引統計を収集するときに、サンプリング技法が使用されます。このオプションは、パフォーマンスの考慮と正確な統計を取る必要性の間でバランスを取るために使用されます。キーワード COLLECT の直後に DETAILED が指定されている場合、このオプションはデフォルトです。

UNSAMPLED
これを指定すると、索引項目を処理して拡張索引統計を収集するときに、サンプリングが使用されません。代わりに、各索引項目は個別に検査されます。このオプションを使用すると、CPU とメモリーの使用量がかなり増加します。

DETAILED
索引の作成時に、拡張索引統計 (CLUSTERFACTOR および PAGE_FETCH_PAIRS) も収集されるように指定します。

COMPRESS
索引圧縮が使用可能かどうかを示します。デフォルトでは、データ行圧縮が使用可能な場合には索引圧縮も使用可能で、データ行圧縮が使用不可な場合には索引圧縮も使用不可です。このオプションを使用して、デフォルト動作をオーバーライドできます。COMPRESS は、*nickname* が指定されている場合は使用できません (SQLSTATE 42601)。

YES
索引圧縮を使用可能にすることを指定します。索引に対する挿入と更新の操作で、圧縮が行われるようになります。

NO 索引圧縮を使用不可にすることを指定します。

規則

- 既存の索引に一致する索引を作成しようとすると、CREATE INDEX ステートメントはエラーになります (SQLSTATE 01550)。

  多数の要因を使用して、2 つの索引が一致するかどうかが判別されます。これらの要因は、さまざまな方法で、2 つの索引が一致するかどうかを判別する規則に結合されます。以下の要因を使用して、2 つの索引が一致するかどうかが判別されます。

  1. INCLUDE 列を含む索引列の集合が、両方の索引で同じである。
  2. INCLUDE 列を含む索引キー列の順序が、両方の索引で同じである。
  3. 新しい索引のキー列が同じであるか、既存の索引内にあるキー列のスーパーセットである。
  4. 列の配列属性が、両方の索引で同じである。
  5. 既存の索引がユニークである。
  6. 両方の索引が非ユニークである。

  これらの要因の以下の組み合わせにより、考慮対象の 2 つの索引がいつ重複しているかを判別する規則が形成されます。

  – 1 + 2 + 4 + 5
  – 1 + 2 + 4 + 6
  – 1 + 2 + 3 + 5

  例外:

  – 比較している索引の一方がパーティション化されていて、もう一方がパーティション化されていない場合、索引名が異なると、たとえその他の索引一致条件を満たしているとしても、それらの索引は重複しているとは見なされません。
  – XML データについての索引の場合、索引記述は、たとえ索引にされる XML 列、XML パターン、およびオプションも含めたデータ・タイプが同一であっても、索引名が異なっていれば重複しているとはみなされません。

- システム保守 MQT 上のユニーク索引は、サポートされません (SQLSTATE 42809)。

- COLLECT STATISTICS オプションは、ニックネームが指定される場合にはサポートされません (SQLSTATE 42601)。

注

- 索引作成中の同時読み取り/書き込みアクセス、およびデフォルトの索引作成動作は、非パーティション表上の索引、非パーティション索引、パーティション索引、および DB2 pureScale 環境内の索引では、次のように異なります。
  - 非パーティション索引では、索引を作成している間に、表への同時読み取り/書き込みアクセスが許可されます。ただし EXTEND USING 節が指定されている場合を除きます。索引が作成されると、索引の作成時に表に加えられた変更は、新しい索引に送られ適用されます。表への書き込みアクセスは、新しい索引が使用できるようになってから、索引の作成が完了するまでの短時間ブロックされます。
  - パーティション索引では、索引を作成している間に、表への同時読み取り/書き込みアクセスが許可されます。ただし EXTEND USING 節が指定されている場合を除きます。索引パーティションが作成されると、その索引パーティションの作成時にパーティションに加えられた変更は、新しい索引パーティションに送られ適用されます。データ・パーティションへの書き込みアクセスは、残りのデータ・パーティションで索引作成が完了するまでブロックされます。最後のデータ・パーティションの索引パーティションが作成され、トランザクションがコミットされると、すべてのデータ・パーティションで読み書きできるようになります。
  - DB2 pureScale 環境では、同時読み取りアクセスがデフォルトの動作です。索引作成時の同時書き込みアクセスは許可されていません。

  このデフォルトの動作を回避するには、LOCK TABLE ステートメントを使用して、 CREATE INDEX ステートメントが発行される前に表を明示的にロックします。 (表は SHARE か EXCLUSIVE モードのいずれかでロックできます。読み取りアクセスが許可されているかどうかによります。)
- 指定した表に既にデータが含まれる場合、CREATE INDEX はそのデータの索引項目を作成します。表にまだデータが含まれていない場合、CREATE INDEX は索引記述を作成します (索引項目は、データが表に挿入される時点で作成されます)。
- b
- 索引が作成され、データが表にロードされた時点で、 RUNSTATS コマンドを実行することをお勧めします。 RUNSTATS コマンドは、データベース表、列、および索引について収集された統計値を更新します。これらの統計値は、表への最適アクセス・パスを判別するために使用されます。 RUNSTATS コマンドを実行することによって、データベース・マネージャーが新しい索引の特性を判別することができます。 CREATE INDEX ステートメントが発行される前にデータをロードする場合には、 CREATE INDEX ステートメントの COLLECT STATISTICS オプションを、 RUNSTATS コマンドの代わりに使用することをお勧めします。
- まだ存在していないスキーマ名を用いて索引を作成すると、ステートメントの許可 ID に IMPLICIT_SCHEMA 権限がある場合に限り、そのスキーマが暗黙に作成されます。スキーマの所有者は SYSIBM になります。スキーマに対する CREATEIN 特権が PUBLIC に付与されます。

- オプティマイザーは、実際の索引を作成する前に、複数の索引を推奨することがあります。
- 索引のあるデータ・ソース表に SPECIFICATION ONLY 指定の索引を定義している場合、その索引名と SPECIFICATION ONLY 指定の索引の名前は一致していなくても構いません。
- オプティマイザーは SPECIFICATION ONLY 指定の索引を使用して、その指定を適用するデータ・ソース表へのアクセスを改善します。
- ***索引統計の収集***: 索引統計の収集方法を変更するために、UNSAMPLED DETAILED オプションを使用できます。ただしこれは、DETAILED では正確な統計が生成されないことが明らかである場合にのみ、使用してください。
- ***代替構文***: 以下の構文は許容されますが、無視されます。
  - CLOSE
  - DEFINE
  - FREEPAGE
  - GBPCACHE
  - PIECESIZE
  - TYPE 2
  - using-block

  以下の構文はデフォルトの振る舞いとして受け入れられます。
  - COPY NO
  - DEFER NO

例

- *例 1:* PROJECT 表に対して UNIQUE_NAM という名前の索引を作成します。この索引の目的は、プロジェクト名 (PROJNAME) の値が同じ 2 つの項目が表に作成されないようにすることです。索引項目は昇順に並べます。

```
CREATE UNIQUE INDEX UNIQUE_NAM
   ON PROJECT(PROJNAME)
```

- *例 2:* EMPLOYEE 表に対して JOB_BY_DPT という名前の索引を作成します。索引項目は、各部門 (WORKDEPT) の中ではジョブ名 (JOB) 順に昇順で並べます。

```
CREATE INDEX JOB_BY_DPT
   ON EMPLOYEE (WORKDEPT, JOB)
```

- *例 3:* ニックネーム EMPLOYEE は、CURRENT_EMP というデータ・ソース表を参照します。このニックネームを作成した後、索引が CURRENT_EMP で定義されます。索引キー用に選んだ列は WORKDEBT と JOB です。この索引を記述する SPECIFICATION ONLY 指定の索引を作成します。この指定を参照することにより、オプティマイザーは、索引が存在することと索引に含まれるキーを知ることになります。この情報を利用して、オプティマイザーは、表をアクセスするときの方法を改善することができます。

```
CREATE UNIQUE INDEX JOB_BY_DEPT
  ON EMPLOYEE (WORKDEPT, JOB)
  SPECIFICATION ONLY
```

- *例 4:* 構造化タイプ列の位置に、拡張索引タイプ SPATIAL_INDEX を作成します。索引拡張 GRID_EXTENSION の記述が SPATIAL_INDEX を保守するのに使用されます。リテラルが GRID_EXTENSION に指定されて、索引格子サイズを作成します。

  ```
  CREATE INDEX SPATIAL_INDEX ON CUSTOMER (LOCATION)
    EXTEND USING (GRID_EXTENSION (x'000100100010001000400010'))
  ```

- *例 5:* TAB1 という名前の表に IDX1 という名前の索引を作成し、索引 IDX1 の基本索引統計を収集します。

  ```
  CREATE INDEX IDX1 ON TAB1 (col1) COLLECT STATISTICS
  ```

- *例 6:* TAB1 という名前の表に IDX2 という名前の索引を作成し、索引 IDX2 の詳細な索引統計を収集します。

  ```
  CREATE INDEX IDX2 ON TAB1 (col2) COLLECT DETAILED STATISTICS
  ```

- *例 7:* TAB1 という名前の表に IDX3 という名前の索引を作成し、サンプリングを使用して索引 IDX3 の詳細な索引統計を収集します。

  ```
  CREATE INDEX IDX3 ON TAB1 (col3) COLLECT SAMPLED DETAILED STATISTICS
  ```

- *例 8:* 表スペース IDX_TBSP 内の MYNUMBERDATA というパーティション表に A_IDX というユニーク索引を作成します。

  ```
  CREATE UNIQUE INDEX A_IDX ON MYNUMBERDATA (A) IN IDX_TBSP
  ```

- *例 9:* 表スペース IDX_TBSP 内の MYNUMBERDATA というパーティション表に B_IDX という非ユニーク索引を作成します。

  ```
  CREATE INDEX B_IDX ON MYNUMBERDATA (B)
    NOT PARTITIONED IN IDX_TBSP
  ```

- *例 10:* COMPANYDOCS という XML 列を含む、COMPANYINFO という表に、XML データに対する索引 を作成します。 XML 列 COMPANYDOCS には、以下のような数多くの XML 文書が含まれます。

  ```
  <company name="Company1">
    <emp id="31201" salary="60000" gender="Female">
      <name>
        <first>Laura</first>
        <last>Brown</last>
      </name>
      <dept id="M25">
        Finance
      </dept>
    </emp>
  </company>
  ```

  COMPANYINFO 表のユーザーは、頻繁に、従業員 ID を使用して従業員情報を検索する必要があります。以下のような索引を作成すると、そうした検索がより効率的になる可能性があります。

  ```
  CREATE INDEX EMPINDEX ON COMPANYINFO(COMPANYDOCS)
    GENERATE KEY USING XMLPATTERN '/company/emp/@id'
      AS SQL DOUBLE
  ```

- *例 11:* 以下の索引は、論理的には前の例で作成したものと同等ですが、短縮していない構文を使用している点が異なります。

  ```
  CREATE INDEX EMPINDEX ON COMPANYINFO(COMPANYDOCS)
    GENERATE KEY USING XMLPATTERN '/child::company/child::emp/attribute::id'
      AS SQL DOUBLE
  ```

- *例 12:* 本のタイトルだけを VARCHAR(100) として索引にし、DOC という列に索引を作成します。本のタイトルは、どれも他のすべての本と異なる固有なものなので、索引もユニークでなければなりません。

```
CREATE UNIQUE INDEX MYDOCSIDX ON MYDOCS(DOC)
  GENERATE KEY USING XMLPATTERN '/book/title'
    AS SQL VARCHAR(100)
```

- *例 13:* 章番号を DOUBLE として索引にし、DOC という列に索引を作成します。この例には、名前空間宣言が含まれます。

```
CREATE INDEX MYDOCSIDX ON MYDOCS(DOC)
  GENERATE KEY USING XMLPATTERN
    'declare namespace b="http://www.example.com/book/";
      declare namespace c="http://acme.org/chapters";
        /b:book/c:chapter/@number'
    AS SQL DOUBLE
```

- *例 14:* 表 PROJECT に IDXPROJEST という名前のユニーク索引を作成し、列 PRSTAFF を組み込んで見積平均スタッフ配置情報の索引のみのアクセスを可能にします。

```
CREATE UNIQUE INDEX IDXPROJEST ON PROJECT (PROJNO) INCLUDE (PRSTAFF)
```

XML データに対する索引への照会のサンプル

XML データに対する索引は、それらを使用することを目的とした照会と一致している必要があります。以下の例は、XML データに対する索引を使用できる照会または使用できない照会を示しています。

XML データに対する索引を使用できる照会のサンプル

さまざまな種類の異なる述部がある照会では、XML データに対する索引を活用することができます。照会で使用できる索引に一致する XQuery 述部のいくつかの例をこのセクションで示します。照会の次に、一致する索引を示します。

例 1。等価のものを対象とする照会を発行します。次のようにして ID *42366* を持つ従業員を検索します。

```
XQUERY for $i in db2-fn:xmlcolumn('COMPANY.COMPANYDOCS')/company/emp[@id='42366']
       return $i

CREATE INDEX empindex on company(companydocs)
      GENERATE KEY USING XMLPATTERN '/company/emp/@id' AS SQL VARCHAR(5)
```

例 2。範囲を対象として照会します。給料が *35000* より高い従業員を検索します。

```
XQUERY
   for $i in db2-fn:xmlcolumn('COMPANY.COMPANYDOCS')/company/emp[@salary > 35000]
   return $i

CREATE INDEX empindex on company(companydocs)
      GENERATE KEY USING XMLPATTERN '//@salary' AS SQL DECIMAL(10,2)
```

例 3。論理和 (OR) を含む照会を発行します。Finance 部門または Marketing 部門にいる従業員を検索します。

```
XQUERY
   for $i in
   db2-fn:xmlcolumn('COMPANY.COMPANYDOCS')/company/emp[dept/text()='Finance'
   or dept/text()='Marketing']
   return $i
```

```
CREATE INDEX empindex on company(companydocs)
      GENERATE KEY USING XMLPATTERN '/company/emp/dept/text()' AS SQL
      VARCHAR(30)
```

例 4。異なる照会が同じ索引に合致する場合もあります。

ID *31201* を持つ従業員を検索します。

```
XQUERY for $i in db2-fn:xmlcolumn('COMPANY.COMPANYDOCS')/company/emp[@id='31201']
        return $i
```

ID *K55* を持つ部門を検索します。

```
XQUERY for $i in db2-fn:xmlcolumn('COMPANY.COMPANYDOCS')/company/emp/dept[@id='K55']
        return $i
```

```
CREATE INDEX empindex on company(companydocs)
      GENERATE KEY USING XMLPATTERN '//@id' AS SQL VARCHAR(25)
```

例 5。照会述部にパスを含めることができます。Sales 部門にいてラストネームが *Murphy* である従業員を検索します。

```
XQUERY
   for $i in db2-fn:xmlcolumn('COMPANY.COMPANYDOCS')/company/emp[name/last='Murphy'
     and dept/text()='Sales']
   return $i
```

```
CREATE INDEX empindex on company(companydocs)
      GENERATE KEY USING XMLPATTERN '/company/emp/name/last' AS SQL
      VARCHAR(100)
```

```
CREATE INDEX deptindex on company(companydocs)
      GENERATE KEY USING XMLPATTERN '/company/emp/dept/text()' AS SQL
      VARCHAR(30)
```

例 6。照会中に階層包含を使用します。照会で索引を使用して、文書階層内の異なるレベルで ANDing を実行できます。照会では、どの下位ノードが同じ祖先に属するかを判別して適切なフィルター操作をするために、索引を使用することもできます。

給料が *60000* に等しい従業員がいる会社を検索し、女性の従業員がいる会社を検索します。 『XML データの索引付けの概要』トピックの XML フラグメントのサンプルでは、Company1 と Company2 の両方が条件に適合します。

```
XQUERY for $i in
   db2-fn:xmlcolumn('COMPANY.COMPANYDOCS')/company[emp/@salary=60000 and
   emp/@gender='Female']
   return $i
```

給料が *60000* に等しく、女性である従業員を検索します。 Company1 の Laura Brown のみが条件に適合します。

```
XQUERY for $i in
   db2-fn:xmlcolumn('COMPANY.COMPANYDOCS')/company/emp[@salary=60000
   and @gender='Female']
   return $i
```

```
CREATE INDEX empindex on company(companydocs)
      GENERATE KEY USING XMLPATTERN '/company/emp/@salary' AS DECIMAL(10,2)
```

```
CREATE INDEX genderindex on company(companydocs)
      GENERATE KEY USING XMLPATTERN '/company/emp/@gender' AS SQL
      VARCHAR(10)
```

例 7。照会述部の厳密さが少なくとも索引パターンと同じであるかまたは索引パターンより厳密である場合、照会で descendant-or-self 軸 (//) を使用して索引を活用できます。

部門 ID *K55* を持つ従業員を検索します。

```
XQUERY
   for $i in
   db2-fn:xmlcolumn('COMPANY.COMPANYDOCS')/company//emp[.//dept//@id='K55' ]
   return $i

CREATE INDEX empindex on company(companydocs)
      GENERATE KEY USING XMLPATTERN '//emp//@id' AS SQL VARCHAR(25)
```

例 8。マルチディメンション・クラスタリング (MDC) 表の場合、照会で MDC ブロック索引と、XML データの索引を使用できます。XML 列を持つ MDC 表が作成され、 WORKDEPT をディメンションとして使用すると想定します。

```
CREATE TABLE employee (empno char(6), workdept char(3), doc xml)
    ORGANIZE BY (workdept)
```

列 DOC の XML データに、以下の CREATE INDEX ステートメントで定義されているような索引があるとします。

```
CREATE INDEX hdate on employee(doc)
      GENERATE KEY USING XMLPATTERN '//hirdate'AS SQL DATE COLLECT STATISTICS
```

次の照会のアクセス・プランでは、WORKDEPT にはブロック索引、DOC には XML データの索引 (hdate) を使用できます。

```
SELECT COUNT (*) FROM y.employee y
      WHERE workdept='A00'
      AND XMLEXISTS('$p/employee[hiredata > "1964-01-01"]
      PASSING y.doc as "p")
```

XML データに対する索引を使用できない照会のサンプル

照会が XML データに対する索引を使用できないいくつかの条件があります。意図した索引を示されているように活用できない XQuery 述部のいくつかの例が、このセクションにリストされています。

例 1。照会により要求されたデータ・タイプが索引付きデータ・タイプと一致しなければ、照会は索引を使用できません。次の例では、照会は従業員 ID をストリングとして要求しますが、その ID は数値として索引付けされています。

```
XQUERY for $i in db2-fn:xmlcolumn('COMPANY.COMPANYDOCS')/company/emp[@id='31664']
        return $i

CREATE INDEX empindex on company(companydocs)
      GENERATE KEY USING XMLPATTERN '/company/emp/@id' AS SQL DOUBLE
```

例 2。索引を作成するために使用される XML パターン式は、照会述部より厳密である可能性があります。この例では、照会は部門 ID と従業員 ID の両方を取得しますが、索引には従業員 ID しか含まれていないため、照会では索引を使用できません。

```
XQUERY for $i in db2-fn:xmlcolumn('COMPANY.COMPANYDOCS')//@id
        return $i

CREATE INDEX empindex on company(companydocs)
      GENERATE KEY USING XMLPATTERN '/company/emp/@id' AS SQL VARCHAR(5)
```

次の照会は、従業員 ID *31201* または部門 ID *K55* を持つ従業員を取得します。ID は従業員 ID または部門 ID のいずれかとなり得ますが、索引には部門 ID のみが含まれるため、索引は作成されても使用できません。

```
XQUERY
   for $i in
   db2-fn:xmlcolumn('COMPANY.COMPANYDOCS')//emp[.//@id='31201' or .//@id='K55']
   return $i

CREATE INDEX empindex on company(companydocs)
      GENERATE KEY USING XMLPATTERN '//dept//@id' AS SQL VARCHAR(5)
```

XML データに対する索引の制約事項

XML データの索引作成には制約事項があり、それにはデータ・タイプ・サポートの制約事項、並行性レベルの制約事項、XML リスト要素の制約事項、および索引圧縮の制約事項が含まれています。

並行性レベル

XML 列および関連した索引の処理中、一部の並行性レベルのサポートは制限されます。以下の表は、どの並行性レベルがサポートされるかを指定しています。

表 24. 少なくとも 1 つの XML 列がある非パーティション表に関してサポートされている並行性レベル

コマンド	並行性レベル	サポートされている
REORG INDEXES ALL FOR TABLE	コマンド節: ALLOW [NO | READ | WRITE] ACCESS	はい。
REORG TABLE	コマンド節: ALLOW [READ | NO] ACCESS	はい。 XML データの索引が存在する可能性があります。
REORG TABLE INPLACE (XML データに対する索引が少なくとも 1 つ表に存在する)	コマンド節: ALLOW [READ | WRITE] ACCESS	いいえ。
REORG TABLE RECLAIM EXTENTS	コマンド節: ALLOW [NO | READ | WRITE] ACCESS	はい。

表 25. 非パーティション索引と、少なくとも 1 つの XML 列があるパーティション表に関してサポートされている並行性レベル

コマンド	並行性レベル	サポートされている
REORG INDEX (XML データの非パーティション索引上)	コマンド節: ALLOW [NO | READ | WRITE] ACCESS	はい。
REORG INDEXES ALL FOR TABLE CLEANUP|RECLAIM EXTENTS	コマンド節: ALLOW [NO | READ | WRITE] ACCESS	はい。
REORG INDEXES ALL FOR TABLE REBUILD[1]	コマンド節: ALLOW [READ | WRITE] ACCESS	いいえ。
REORG TABLE	コマンド節: ALLOW NO ACCESS (パーティション表の場合、サポートされているアクセスは ALLOW NO ACCESS のみ)	はい。

表 25. 非パーティション索引と、少なくとも 1 つの XML 列があるパーティション表に関してサポートされている並行性レベル (続き)

コマンド	並行性レベル	サポートされている
REORG TABLE INPLACE	コマンド節: ALLOW [NO | READ | WRITE] ACCESS	いいえ。パーティション表の場合、**INPLACE** パラメーターはサポートされません。
REORG TABLE RECLAIM EXTENTS	コマンド節: ALLOW [NO | READ | WRITE] ACCESS	はい。

注:

1. デフォルトの動作では、索引を再作成します。**REBUILD** 節の指定がない場合、索引は再作成されます。

表 26. パーティション索引と XML 列があるパーティション表に関してサポートされている並行性レベル

コマンド	並行性レベル	サポートされている
REORG INDEXES ALL FOR TABLE	コマンド節: ALLOW NO ACCESS	はい。
REORG INDEXES ALL FOR TABLE CLEANUP|RECLAIM EXTENTS	コマンド節: ALLOW [READ | WRITE] ACCESS	はい。
REORG INDEXES ALL FOR TABLE REBUILD ON DATA PARTITION[1]	コマンド節: ALLOW [READ | WRITE] ACCESS	はい。
REORG INDEXES ALL FOR TABLE REBUILD[1]	コマンド節: ALLOW [READ | WRITE] ACCESS	いいえ。
REORG TABLE	コマンド節: ALLOW NO ACCESS (データ・パーティション表の場合、サポートされているアクセスは ALLOW NO ACCESS のみ)	はい。
REORG TABLE INPLACE	コマンド節: ALLOW [NO | READ | WRITE] ACCESS	いいえ。データ・パーティション表の場合、**INPLACE** パラメーターはサポートされません。
REORG TABLE RECLAIM EXTENTS	コマンド節: ALLOW [NO | READ | WRITE] ACCESS	はい。

注:

1. デフォルトの動作では、索引を再作成します。**REBUILD** 節の指定がない場合、索引は再作成されます。

これらの節とオプションについては、**CREATE INDEX** ステートメントと **REORG INDEX/TABLE** コマンドを参照してください。

マルチディメンション・クラスタリング (MDC) 表および挿入時クラスタリング (ITC) 表の場合、ALLOW WRITE ACCESS を使用したオンライン索引再編成 (再作成) はサポートされていません。

XML リスト要素

リスト・データ・タイプのノードを索引付けすることはできません。ノードが *xmlpattern-clause* で修飾され、ノードがリスト・データ・タイプであることを指定する XML スキーマが存在する場合、ノードを索引付けすることはできません。リスト・データ・タイプのノードに **CREATE INDEX** ステートメントを発行すると、エラー (SQLSTATE 23526、SQLCODE -20306) が戻ります。 **INSERT** および **UPDATE** ステートメントを発行しても、エラー (SQLSTATE 23525、SQLCODE -20305) が戻ります。

UNIQUE な、XML データに対する索引

パーティション・データベース環境では、1 つ以上の XML 列がある表に以下の規則が適用されます。

- 分散キーがある表が、XML データのユニーク索引を持つことはできません。
- XML データに対するユニーク索引は、単一パーティション・データベースの分散キーのない表のみでサポートされます。
- XML データに対するユニーク索引が表にある場合、その表に変更を加えて分散キーを追加することはできません。

パーティション表の場合、XML データに対するユニーク・パーティション索引はサポートされません。そのような索引を作成しようとすると、エラー・メッセージ SQL20303N (SQLSTATE=42990) を受け取ります。

XML 列の索引の作成においても、ネイティブ XML データ・ストア全体に課される制限が適用されます。

XML 索引付けの一般的な問題

XML データの索引付けの際に問題が発生した場合、以下の問題シナリオの 1 つが適用されることがあります。

SQL20305N および SQL20306N エラー・メッセージの問題判別

これらのエラー・メッセージは、XML ノードの値を索引付けできない場合に発行されます。 SQL20305N メッセージは、INSERT と UPDATE ステートメントによって、およびインポートとロード・ユーティリティーによって発行されます。 SQL20306N メッセージは、値の取り込まれた基本表に対して発行された CREATE INDEX ステートメントによって発行されます。

メッセージは、エラーの理由コードを出力します。`? SQL20305` または `? SQL20306` をコマンド行プロセッサーから発行して、対応する理由コードの説明およびユーザー応答を検索します。生成された XQuery ステートメントは、**db2diag** ログ・ファイルに出力されて、失敗した XML ノード値の位置指定に役立ちます。

SQL20305N がロード・ユーティリティーによって発行された場合、失敗したノード値を位置指定するために生成される XQuery ステートメントは **db2diag** ログ・ファイルに書き込まれません。これらの XQuery ステートメントを生成するために、インポート・ユーティリティーはロードされなかった失敗した行の上で実行する必要があります。詳しくは、DB2 インフォメーション・センターで『XML データのロード』および『XML データのロード時の索引付けエラーの解決』を参照してください。

SQL20305N が INSERT または UPDATE ステートメントによって発行される場合、 『INSERT または UPDATE ステートメントにより発行された SQL20305N メッセージのトラブルシューティング』を参照してください。 SQL20306N が CREATE INDEX ステートメントによって発行される場合、『データが挿入された表に対する CREATE INDEX ステートメントによって出された SQL20306N メッセージのトラブルシューティング』を参照してください。

INSERT または UPDATE ステートメントにより発行された SQL20305N メッセージのトラブルシューティング

SQL20305N エラー・メッセージの原因を調べるため、「SQL20305N および SQL20306N エラー・メッセージの問題判別」を参照してから、以下のステップに従ってください。

手順

1. 索引名を判別し、XML パターン節に索引付けします。
   a. エラー・メッセージにある *index-id* を使用して以下の照会を発行することにより、索引名 (*index-name*、*index-schema*) を SYSCAT.INDEXES から取得します。

      ```
      SELECT INDNAME,INDSCHEMA
         FROM SYSCAT.INDEXES
         WHERE IID = index-id AND
         TABSCHEMA ='schema' AND TABNAME ='table-name'
      ```

   b. *index-name* および *index-schema* を使用して以下の照会を発行することにより、索引のデータ・タイプと XML パターンを SYSCAT.INDEXES から取得します。

      ```
      SELECT DATATYPE, PATTERN
         FROM SYSCAT.INDEXXMLPATTERNS
         WHERE INDSCHEMA = 'index-schema' AND
         INDNAME = 'index-name'
      ```

2. 入力文書で失敗したノード値を検索するため、**db2diag** ログ・ファイルでストリング SQL20305N を検索し、理由コード番号を突き合わせます。理由コードの後に一連の指示があり、続いてエラーの原因となっている文書内の値を見つけるために使用できる、生成された XQuery ステートメントがあります。小さいノード値では、値全部が XQuery 述部で使用されます。長すぎて **db2diag** ログ・ファイルに出力できないノード値の場合、XQuery 述部の中で、値の開始バイトが fn:starts-with 関数で使用され、値の終了バイトが fn:ends-with 関数で使用されます。
3. 文書はリジェクトされて表には存在しないため、XQuery ステートメントをそれに対して実行することはできません。この問題を解決するため、元の表と同じ列を持つ新規表を作成し、新規表に失敗した文書を挿入します。新規表には索引を作成しないでください。
4. 生成された XQuery ステートメントを **db2diag** ログ・ファイルからコピーし、XQuery にある表名を新規に作成した表名に置き換えます。
5. XQuery ステートメントを実行して、文書全体と、失敗の原因となった値が含まれる文書の一部を取得します。文書のどこでエラーが発生したかを示すコンテキスト情報を示すため、XQuery ステートメントは、失敗の原因となっているノード値の親で開始する文書フラグメントを出力します。

6. 索引 XML パターンを使用し、検査するための、一致する XML ノードのセットを識別します。生成された XQuery ステートメントは名前空間でワイルドカードを使用するため、異なる名前空間を持つ複数の問題値が適格となる可能性があります (ただしこれはよくあることではありません)。このようになった場合には、索引 XML パターンで名前空間宣言を使用し、一致する XML ノードの正しいセットを判別する必要があります。結果をフィルターに掛ける際に述部で完全な値が使用されなかった場合、索引 XML パターンを使用して、XQuery ステートメントにより返されて適格となる問題値を検証する必要があります。
7. 文書内の失敗となっている値を見つけたら、入力文書を変更して問題を訂正し、INSERT または UPDATE ステートメントを再サブミットします。

例: INSERT ステートメント・エラー

以下の例では、hello world が無効な DOUBLE 値で、生成された XQuery 述部で値全体が使用されています。スキーマ情報が適用されないエラー・メッセージで、プレースホルダーとして *N が使用されている点に注意してください。

```
CREATE TABLE t1 (x XML);

CREATE INDEX ix1 ON t1(x)
   GENERATE KEY USING XMLPATTERN '/root/x/text()'
   AS SQL DOUBLE REJECT INVALID VALUES;

DB20000I  The SQL command completed successfully.

INSERT INTO t1 VALUES (XMLPARSE (DOCUMENT
   'The beginning of the documenthello world'
   STRIP WHITESPACE));

DB21034E  The command was processed as an SQL statement because it was not a
valid Command Line Processor command.  During SQL processing it returned:

SQL20305N  An XML value cannot be inserted or updated because of an error
detected when inserting or updating the index identified by "IID = 23" on
table "ADUA.T".  Reason code = "5".  For reason codes related to an XML schema
the XML schema identifier = "*N" and XML schema data type = "*N".
SQLSTATE=23525
```

db2diag ログ・ファイルの出力は以下のようになります (フォーマットを多少変更しています)。

```
2007-03-06-12.02.08.116046-480 I4436A1141         LEVEL: Warning
PID     : 1544348              TID  : 1801        PROC : db2sysc
INSTANCE: adua                 NODE : 000         DB   : ADTEST
APPHDL  : 0-18                 APPID: *LOCAL.adua.070306200203
AUTHID  : ADUA
EDUID   : 1801                 EDUNAME: db2agent (ADTEST)
FUNCTION: DB2 UDB, Xml Storage and Index Manager,
          xmlsIkaProcessErrorMsg, probe:651
MESSAGE : ZRC=0x80A50411=-2136669167=XMS_XML_IX_INSERT_UPDATE_ERROR
          "XML node value error during insert or update XML index"
DATA #1 : String, 36 bytes
SQL Code: SQL20305N ; Reason Code: 5
DATA #2 : String, 321 bytes
To locate the value in the document that caused the error, create
a new table with the same columns as the original table and insert
the failing document in the table. Do not create any indexes on
the new table. Replace the table name in the query below with the
newly created table name and execute the following XQuery.
DATA #3 : String, 187 bytes
xquery for $i in db2-fn:xmlcolumn("ADUA.T.X")[/*:root/*:x/text()="hello world"]
return
```

```
<Result>
 <ProblemDocument> {$i} </ProblemDocument>
 <ProblemValue>{$i/*:root/*:x/text()/..} </ProblemValue>
</Result>;
```

失敗したノード値を検出するには、以下のようにします。

1. 元の表と同じ列を持つ新規表を作成します。

   ```
   CREATE TABLE t2 LIKE t1;
   ```

2. 失敗した文書を新規表に挿入します。

   ```
   INSERT INTO t2 VALUES (XMLPARSE (DOCUMENT
   'The beginning of the documenthello world'
   STRIP WHITESPACE));
   ```

3. 生成された XQuery ステートメントを **db2diag** ログ・ファイルからコピーし、XQuery にある表名を新規表の名前に置き換えます。

   ```
   xquery for $i in db2-fn:xmlcolumn("ADUA.T2.X")[/*:root/*:x/text()="hello world"]
   return
   {$i}
   {$i/*:root/*:x/text()/..}
   ;
   ```

4. 新規表に対して XQuery ステートメントを実行します。照会ステートメントの結果は以下のようになります (フォーマットを多少変更しています)。

   ```
   <Result>
      <ProblemDocument>
         <root>The beginning of the document<x>hello world</x></root>
      </ProblemDocument>
      <ProblemValue><x>hello world</x></ProblemValue>
   </Result>
   ```

エラーを訂正します。

DOUBLE データ・タイプに正常にキャストする数値を <x> 要素が持つように、文書を変更できます。

```
INSERT INTO t1 VALUES (
   XMLPARSE (DOCUMENT
   '<root>The beginning of the document<x>123</x></root>'
   STRIP WHITESPACE))
```

データが挿入された表に対する CREATE INDEX ステートメントによって出された SQL20306N メッセージのトラブルシューティング

SQL20306N エラー・メッセージの原因を調べるため、「SQL20305N および SQL20306N エラー・メッセージの問題判別」を参照してから、以下のステップに従ってください。

手順

1. 保管された文書で失敗したノード値を検索するため、**db2diag** ログ・ファイルでストリング SQL20306N を検索し、理由コード番号を突き合わせます。 理由コードの後に一連の指示があり、続いてエラーの原因となった文書内の値を見つけるために使用できる、生成された XQuery ステートメントがあります。
   - 小さいノード値では、値全部が XQuery 述部で使用されます。

- 長すぎて **db2diag** ログ・ファイルに出力できないノード値の場合、XQuery 述部の中で、値の開始バイトが fn:starts-with 関数で使用され、値の終了バイトが fn:ends-with 関数で使用されます。

2. XQuery ステートメントを実行して、文書全体と、失敗の原因となった値が含まれる文書の一部を取得します。 文書のどこでエラーが発生したかを示すコンテキスト情報を示すため、XQuery ステートメントは、失敗の原因となっているノード値の親で開始する文書フラグメントを出力します。
3. 索引 XML パターンを使用し、検査するための、一致する XML ノードのセットを識別します。生成された XQuery ステートメントは名前空間でワイルドカードを使用するため、異なる名前空間を持つ複数の問題値が適格となる可能性があります (ただしこれはよくあることではありません)。このようになった場合には、索引 XML パターンで名前空間宣言を使用し、一致する XML ノードの正しいセットを判別する必要があります。結果をフィルターに掛ける際に述部で完全な値が使用されなかった場合、索引 XML パターンを使用して、XQuery ステートメントにより返されて適格となる問題値を検証する必要があります。
4. 文書内の失敗となっている値を見つけたら、CREATE INDEX XML パターンを変更して問題を訂正するか、または XQuery 述部を使用して、失敗となっている値が含まれる文書を更新または削除します。

例: CREATE INDEX の失敗

この例では、保管された文書の修飾されたテキスト値が索引 XML パターンの VARCHAR(4) 長さ制約を超えるため、CREATE INDEX ステートメントは失敗します。大きい値では、生成された XQuery は、述部で fn:starts-with および fn:ends-with 関数を使用します。スキーマ情報が適用されないエラー・メッセージで、プレースホルダーとして *N が使用されている点に注意してください。

```
INSERT INTO t VALUES (XMLPARSE (DOCUMENT '
   <x>This is the beginning of the document
       <y>test
          <z>rld12345678901234567890123412345678901234567890123
   4567890123456789012345678900987654321123456789009876 5
   432112344567788098765432112345678904 55</z>
       </y>
   </x>' strip whitespace))

DB20000I  The SQL command completed successfully.

CREATE INDEX i1 ON t(x)
   GENERATE KEY USING XMLPATTERN '/x/y//text()'
   AS SQL VARCHAR(4)

DB21034E  The command was processed as an SQL statement because it was not a
valid Command Line Processor command.  During SQL processing it returned:

SQL20306N  An index on an XML column cannot be created because of an error
detected when inserting the XML values into the index.  Reason code = "1". For
reason codes related to an XML schema the XML schema identifier = "*N" and XML
schema data type = "*N".  SQLSTATE=23526
```

db2diag ログ・ファイルの出力は以下のようになります (フォーマットを多少変更しています)。

```
2007-03-06-12.08.48.437571-480 I10148A1082        LEVEL: Warning
PID     : 1544348              TID  : 1801        PROC : db2sysc
INSTANCE: adua                 NODE : 000         DB   : ADTEST
APPHDL  : 0-30                 APPID: *LOCAL.adua.070306200844
```

```
AUTHID  : ADUA
EDUID   : 1801                 EDUNAME: db2agent (ADTEST)
FUNCTION: DB2 UDB, Xml Storage and Index Manager,
          xmlsIkaProcessErrorMsg, probe:361
MESSAGE : ZRC=0x80A50412=-2136669166=XMS_XML_CRIX_ERROR
          "XML node value error during create XML Index"
DATA #1 : String, 36 bytes
SQL Code: SQL20306N ; Reason Code: 1
DATA #2 : String, 72 bytes
To locate the value in the document that caused the error, execute
the following XQuery.
DATA #3 : String, 435 bytes
xquery for $doc in db2-fn:xmlcolumn("ADUA.T.X")[/*:x/*:y/*:z/text()
[fn:starts-with(., "rld12345678901234567890123412345678901234567890123")
and fn:ends-with(., "567890098765432112344567788098765432112345678904555")]]
return
<Result>
 <ProblemDocument> {$doc} </ProblemDocument>
 <ProblemValue> {$doc/*:x/*:y/*:z/text()/..} </ProblemValue>
</Result>;
```

照会ステートメントの結果は以下のようになります (フォーマットを多少変更しています)。

```
<Result>
    <ProblemDocument>
         <x>This is the beginning of the document
           <y>test
              <z>rld12345678901234567890123412345678901234567890123
       456789012345678901234567890098765432112345678900987 65
       432112344567788098765432112345678904555</z>
           </y>
       </x>
</ProblemDocument>
 <ProblemValue>
     <z>rld12345678901234567890123412345678901234567890123
     456789012345678901234567890098765432112345678900987 65
     432112344567788098765432112345678904555</z>
 </ProblemValue>
</Result>
```

エラーを訂正します。

CREATE INDEX XML パターンを以下のように変更すると、VARCHAR の最大長を増やすことができます。

```
CREATE INDEX i1 ON t(x)
   GENERATE KEY USING XMLPATTERN '/x/y//text()'
   AS SQL VARCHAR(200)
```

第 7 章 XML データの更新

XML 列内のデータを更新するには、SQL UPDATE ステートメントを使用します。特定の行を更新したいときは、WHERE 節を組み込みます。列値の全体が置換されます。 XML 列への入力は、整形式 XML 文書であることが必要です。アプリケーション・データ・タイプは、XML、文字、またはバイナリー・タイプとすることができます。

XML 列を更新するとき、登録済みの XML スキーマに照らして入力 XML 文書を妥当性検査することもできます。これは、XMLVALIDATE 関数を使用して行うことができます。

XML 列値を使用して、更新する行を指定できます。 XML 文書内の値を検索するには、 XQuery 式を使用する必要があります。XQuery 式を指定する方法の 1 つは XMLEXISTS 述部です。これにより XQuery 式を指定し、式の結果が空のシーケンスになるかどうかを判別することができます。 XMLEXISTS が WHERE 節内に指定されているとき、XQuery 式が空ではないシーケンスを戻すと、行が更新されます。

以下の例は、XML データを XML 列内で更新する方法を示しています。この例ではサンプルの CUSTOMER 表のコピーである表 MYCUSTOMER を使用します。この例は、1004 というカスタマー ID 値を持つ行が既に MYCUSTOMER に含まれていることを前提としています。既存の列データを更新する XML データはファイル c7.xml に保管されていて、その内容は次のようであると想定します。

```
<customerinfo Cid="1004">
  <name>Christine Haas</name>
  <addr country="Canada">
    <street>12 Topgrove</street>
    <city>Toronto</city>
    <prov-state>Ontario</prov-state>
    <pcode-zip>N9Y-8G9</pcode-zip>
  </addr>
  <phone type="work">905-555-5238</phone>
  <phone type="home">416-555-2934</phone>
</customerinfo>
```

例: JDBC アプリケーションで、XML データをファイル c7.xml からバイナリー・データとして読み取り、それを使用して XML 列内のデータを更新します。

```
PreparedStatement updateStmt = null;
String sqls = null;
int cid = 1004;
sqls = "UPDATE MyCustomer SET Info=? WHERE Cid=?";
updateStmt = conn.prepareStatement(sqls);
updateStmt.setInt(1, cid);
File file = new File("c7.xml");
updateStmt.setBinaryStream(2, new FileInputStream(file), (int)file.length());
updateStmt.executeUpdate();
```

例: 組み込み C アプリケーションで、バイナリー XML ホスト変数を使って XML 列内のデータを更新します。

```
EXEC SQL BEGIN DECLARE SECTION;
   sqlint64 cid;
   SQL TYPE IS XML AS BLOB (10K) xml_hostvar;
EXEC SQL END DECLARE SECTION;
...
cid=1004;
/* Read data from file c7.xml into xml_hostvar */
...
EXEC SQL UPDATE MyCustomer SET Info=:xml_hostvar WHERE Cid=:cid;
```

上記の例の場合、<customerinfo> 要素内の Cid 属性の値が CID リレーショナル列にも保管されるようになっています。そのため、UPDATE ステートメント内の WHERE 節はリレーショナル列 CID を使用して更新する行を指定します。更新の対象として選択する行を決める値が XML 文書自体の中にしかない場合には、XMLEXISTS 述部を使用できます。例えば、前述の組み込み C アプリケーション例にある UPDATE ステートメントは、次のように変更して XMLEXISTS を使用するようにできます。

```
EXEC SQL UPDATE MyCustomer SET Info=:xml_hostvar
       WHERE XMLEXISTS ('$doc/customerinfo[@Cid = $c]'
                        passing INFO as "doc", cast(:cid as integer) as "c");
```

例: 以下の例では、MYCUSTOMER 表の既存の XML データを更新します。SQL UPDATE ステートメントが MYCUSTOMER 表の行を操作し、行の INFO 列の文書を、変換式によって変更された文書の論理スナップショットに置き換えます。

```
UPDATE MyCustomer
SET info = XMLQUERY(
   'transform
    copy $newinfo := $info
    modify do insert <status>Current</status>
      as last into $newinfo/customerinfo
    return $newinfo' passing info as "info")
WHERE cid = 1004
```

変換式での更新式の使用

DB2 XQuery の更新式は、変換式の **modify** 節で使用する必要があります。更新式は、変換式の **copy** 節によって作成される、コピーされたノードを操作します。

以下の式が更新式です。

- 削除式
- 挿入式
- 名前変更式
- 置換式
- **return** 節に更新式を含む FLWOR 式
- **then** または **else** 節に更新式を含む条件式
- すべてのオペランドが更新式か空のシーケンスの、コンマで区切られた 2 つ以上の更新式

無効な更新式の場合、DB2 XQuery はエラーを戻します。例えば、条件式の一方の分岐に更新式が含まれ、もう一方の分岐には更新式でも空のシーケンスでもないものが含まれる場合、DB2 XQuery はエラーを戻します。

変換式は、既存のノードを変更するわけではないので、更新式ではありません。変換式は、既存のノードの変更されたコピーを作成します。変換式の結果は、変換式の **modify** 節内の更新式によって作成されたノードと、既存のノードのコピーを含むことができます。

XQuery 更新操作の処理

変換式では、**modify** 節で複数の更新を指定できます。例えば、既存の値を置き換える式と、新しい要素を挿入する式の、2 つの更新式を **modify** 節に含めることができます。 **modify** 節に複数の更新式が含まれている場合、各更新式は単独で評価され、変換式の **copy** 節によって作成された特定のノードに関連付けられた変更操作のリストができます。

modify 節内で、更新式は他の更新式によって追加される新しいノードを変更できません。例えば、更新式が新しい要素ノードを追加する場合、別の更新式は新しく作成されたそのノードのノード名を変更することはできません。

変換式の **modify** 節で指定されたすべての変更操作が収集され、以下の順序で実際に適用されます。

1. 以下の更新操作が非決定論的順序で行われます。
   - **before**、**after**、**as first**、**as last** などの順序付けキーワードを使用していない挿入操作。
   - すべての名前変更操作。
   - キーワード **value of** が指定されていて、ターゲット・ノードが属性ノード、テキスト・ノード、コメント・ノード、または処理命令ノードである置換操作。
2. **before**、**after**、**as first**、**as last** などの順序付けキーワードを使用している挿入操作。
3. キーワード **value of** が指定されていない置換操作。
4. キーワード **value of** が指定されていてターゲット・ノードが要素ノードである置換操作。
5. すべての削除操作。

変更操作がこの順序で適用されることによって、一連の複数の変更が決定論的結果になります。更新操作の順序によって、一連の複数の変更が決定論的結果になることが保証されることの例については、212 ページの『例』にある最後の XQuery 式を参照してください。

無効な XQuery 更新操作

変換式の処理中に以下のいずれかの状態が検出された場合、DB2 XQuery はエラーを戻します。

- 同一ノードに 2 つ以上の名前変更操作が適用される。
- **value of** キーワードを使用する複数の置換操作が同じノードに適用される。
- **value of** キーワードを使用しない複数の置換操作が同じノードに適用される。
- 変換式の結果が有効な XDM インスタンスでない。

無効な XDM インスタンスの例として、2 つの属性を持つ 1 つの要素が含まれていて、両方の属性の名前が同じであるインスタンスがあります。

- XDM インスタンスに、不整合な名前空間バインディングが含まれる。

  不整合な名前空間バインディングとは、例えば次のようなものです。

  – 属性ノードの QName での名前空間バインディングが、その親要素ノードでの名前空間バインディングと一致しない。
  – 同じ親を持つ 2 つの属性ノードでの名前空間バインディングが、相互に一致しない。

例

以下の例では、変換式の **copy** 節が変数 $product を要素ノードのコピーにバインドし、変換式の **modify** 節が 2 つの更新式を使用して、コピーされたノードを変更します。

```
xquery
transform
copy $product := db2-fn:sqlquery(
  "select description from product where pid='100-100-01'")/product
modify(
  do replace value of $product/description/price with 349.95,
  do insert <status>Available</status> as last into $product )
return $product
```

以下の例では、SQL UPDATE ステートメント内で XQuery 変換式を使用して、CUSTOMER 表の XML データを変更します。この SQL UPDATE ステートメントは、CUSTOMER 表の行を操作します。変換式が行の INFO 列から XML 文書のコピーを作成し、そのコピーに status 要素を追加します。UPDATE ステートメントが、行の INFO 列の文書を、変換式によって変更された文書のコピーに置き換えます。

```
UPDATE customer
SET info = xmlquery( 'transform
   copy $newinfo := $info
   modify do insert <status>Current</status> as last into $newinfo/customerinfo
   return $newinfo' passing info as "info")
WHERE cid = 1003
```

以下の例では、DB2 SAMPLE データベースの CUSTOMER 表を使用します。CUSTOMER 表の XML 列 INFO には、顧客の住所と電話に関する情報が含まれています。

以下の例の SQL SELECT ステートメントは、CUSTOMER 表の行を操作します。変換式の **copy** 節が、INFO 列から XML 文書のコピーを作成します。削除式が、住所情報と work 以外の電話番号を文書のコピーから削除します。 **return** は、CUSTOMER 表のオリジナル文書にあるカスタマー ID 属性と country 属性を使用します。

```
SELECT XMLQUERY( 'transform
  copy $mycust := $d
  modify
    do delete ( $mycust/customerinfo/addr,
       $mycust/customerinfo/phone[@type != "work"] )
  return
  <custinfo>
    <Cid>{data($d/customerinfo/@Cid)}</Cid>
```

```
    {$mycust/customerinfo/*}
    <country>{data($d/customerinfo/addr/@country)}</country>
  </custinfo>'
  passing INFO as "d")
FROM CUSTOMER
WHERE CID = 1003
```

SAMPLE データベースに対して実行すると、このステートメントは以下の結果を戻します。

```
<custinfo>
  <Cid>1003</Cid>
  <name>Robert Shoemaker</name>
  <phone type="work">905-555-7258</phone>
  <country>Canada</country>
</custinfo>
```

以下の例の XQuery 式は、一連の複数の変更が決定論的結果になることが、更新操作の順序によって保証されることを示しています。挿入式が phone 要素の後に status 要素を追加し、置換式が phone 要素を email 要素に置き換えます。

```
xquery
let $email  := <email>jnoodle@my-email.com</email>
let $status := <status>current</status>
return
  transform
  copy $mycust := db2-fn:sqlquery('select info from customer where cid = 1002')
  modify (
    do replace $mycust/customerinfo/phone with $email,
    do insert $status after $mycust/customerinfo/phone[@type = "work"] )
  return $mycust
```

modify 節内では、置換式が挿入式の前にあります。ただし、コピーされたノード・シーケンス $mycust を更新するときは、決定論的結果になるように、置換更新操作の前に挿入更新操作が行われます。 SAMPLE データベースに対して実行すると、この式は以下の結果を戻します。

```
<customerinfo Cid="1002">
  <name>Jim Noodle</name>
  <addr country="Canada">
    <street>25 EastCreek</street>
    <city>Markham</city>
    <prov-state>Ontario</prov-state>
    <pcode-zip>N9C 3T6</pcode-zip>
  </addr>
  <email>jnoodle@my-email.com</email>
  <status>current</status>
</customerinfo>
```

最初に置換操作が行われると、ノード・シーケンスに phone 要素が存在しなくなるので、phone 要素の後に status 要素を挿入するという操作は、成り立たなくなってしまいます。

更新操作の順序については、 211 ページの『XQuery 更新操作の処理』を参照してください。

他の表の情報を使用した XML 文書の更新

他のデータベース列のデータを使用して XML 文書を更新することができます。例えば更新されたカスタマー情報が含まれる表がある場合、SQL/XML ステートメントおよび XQuery 式を使用して XML 文書内のカスタマー情報を更新できます。

以下の SQL ステートメントを使用すると、顧客の新しい電話番号が含まれるサンプル表が作成されます。

```
CREATE TABLE NewPhones (
   CID BIGINT NOT NULL PRIMARY KEY, PhoneNo VARCHAR(20), Type VARCHAR(10))~

INSERT INTO NewPhones (CID, PhoneNo, Type) VALUES (1001, '111-222-3333', 'cell' )~
INSERT INTO NewPhones (CID, PhoneNo, Type) VALUES (1002, '222-555-1111', 'home' )~
INSERT INTO NewPhones (CID, PhoneNo, Type) VALUES (1003, '333-444-2222', 'home' )~
```

以下の SQL ステートメントを使用すると、CUSTOMER 表の顧客の電話番号が更新されます。このステートメントは、XMLQUERY 関数および XQuery 式を使用します。XQuery 式は、FLWOR 式と、挿入式を持つ変換式で構成されています。

```
UPDATE CUSTOMER SET INFO = XMLQUERY(
   'let $myphone := db2-fn:sqlquery(''SELECT XMLELEMENT(Name "phone",
        XMLATTRIBUTES( NewPhones.Type as "type" ), NewPhones.PhoneNo )
        FROM NewPhones WHERE CID = parameter(1)'', $mycid )
   return
     transform
     copy $mycust := $d
     modify
      do insert $myphone after $mycust/customerinfo/phone[last()]
     return
       $mycust'
  passing INFO as "d", 1002 as "mycid" )
WHERE CID = 1002~
```

XMLQUERY 関数は、phone 要素ノードをカスタマー情報に追加する XQuery 式を実行し、変更された情報を UPDATE ステートメントへ返します。 XQuery の FLWOR 式の **let** 節において、db2-fn:sqlquery 関数は SQL 全選択ステートメントを実行します。全選択は、XMLQUERY から XQuery 式へ渡されたカスタマー ID を使用します。

全選択ステートメントは XML データ・タイプを戻す必要があります。SELECT ステートメントから返された PHONENO および TYPE データから XML データ・タイプを作成するため、XMLELEMENT および XMLATTRIBUTES 関数は、提供されたデータに基づいて phone 要素ノードを作成します。

この例では、**let** 節の db2-fn:sqlquery で実行された全選択は、以下の phone 要素ノードを作成します。

```
<phone type="home">222-555-1111<phone>
```

以下の SQL SELECT を実行すると、職場と自宅の電話番号が含まれるカスタマー情報が表示されるようになります。

```
SELECT INFO FROM CUSTOMER WHERE CID = 1002~
```

表からの XML データの削除

XML 文書を含む行を削除するには、DELETE SQL ステートメントを使用します。特定の行を削除したいときは、WHERE 節を組み込みます。

XML 列内の値に基づいて、削除する行を指定できます。 XML 文書内の値を検索するには、 XQuery 式を使用する必要があります。XQuery 式を指定する方法の 1 つは XMLEXISTS 述部です。これにより XQuery 式を指定し、式の結果が空のシーケンスになるかどうかを判別することができます。 XMLEXISTS が WHERE 節内に指定されているとき、XQuery 式が空ではないシーケンスを戻すと、行が削除されます。

XML 列は、NULL であるか、整形式 XML 文書を含むかのどちらかであることが必要です。行を削除しないで XML 列から XML 文書を削除するには、列が NULL 可能として定義されている場合、SET NULL を指定した UPDATE SQL ステートメントを使用して、その列を NULL に設定します。既存の XML 文書から属性または要素などのオブジェクトを削除するには、XQuery 更新式を含む UPDATE SQL ステートメントを使用します。XQuery 更新式により、既存の XML 文書のコピーに変更を加えることができます。その後 UPDATE ステートメントは、XQuery 更新式により戻された変更済みのコピーを、指定された行に関する XML 列に適用します。

以下の例は、XML データを XML 列から削除する方法を示しています。この例では、サンプルの Customer 表のコピーである表 MyCustomer を使用します。MyCustomer に Customer のすべてのデータが取り込まれていることを前提としています。

例: 表 MyCustomer から、Cid 列の値が 1002 である行を削除します。

```
DELETE FROM MyCustomer WHERE Cid=1002
```

例: 表 MyCustomer から、city 要素の値が Markham である行を削除します。このステートメントは、カスタマー ID が 1002 である行を削除します。

```
DELETE FROM MyCustomer
 WHERE XMLEXISTS ('$d//addr[city="Markham"]' passing INFO as "d")
```

例: 表 MyCustomer から、city 要素の値が Markham である行の XML 文書を削除しますが、その行は残します。このステートメントは、カスタマー ID が 1002 である行の Info 列から XML データを削除します。

```
UPDATE MyCustomer SET Info = NULL
  WHERE XMLEXISTS ('$d//addr[city="Markham"]' passing INFO as "d")
```

例: 以下の例では、MyCustomer 表の既存の XML データから電話情報を削除します。この SQL UPDATE ステートメントは、MyCustomer 表の行を操作します。XQuery 変換式によって、行の INFO 列から XML 文書のコピーが作成された後、XQuery 削除式を使用してその文書のコピーから勤務先電話番号が削除されます。UPDATE ステートメントが、行の INFO 列の文書を、変換式によって変更された文書のコピーに置き換えます。

```
UPDATE MyCustomer
SET info = XMLQUERY(
   'transform
```

```
    copy $newinfo := $info
    modify do delete ($newinfo/customerinfo/phone[@type="work"])
    return $newinfo' passing info as "info")
WHERE cid = 1004
```

第 8 章 XML スキーマ・リポジトリー

XML スキーマ・リポジトリー (XSR) は、XML 列に保管される XML インスタンス文書を処理するために使用されるすべての XML 要素のためのリポジトリーです。XSR の目的は、これらの XML 要素上の従属関係のタスクをサポートすることです。

XML インスタンス文書には、関連した XML スキーマ、DTD、または他の外部エンティティーを指す Uniform Resource Identifier (URI) への参照が含まれている場合があります。この URI は、インスタンス文書を処理するために必要です。 DB2 データベース・システムは、XSR を使用して、URI ロケーションの参照を変更することなく、外部に参照される XML の従属関係を管理します。

関連した XML スキーマ、DTD、または外部エンティティーを保管するこのメカニズムがなければ、必要なときにデータベースが外部リソースにアクセスできなくなるおそれや、あるいはデータベースに保管されている、妥当性検査済みでアノテーションの付いた XML 文書に対する必要な変更が行われずに、外部のリソースが変更されてしまうおそれがあります。さらに XSR は、外部文書を見つけるために必要な余分のオーバーヘッドや、起こりうるパフォーマンスへの影響をなくします。

各データベースには、データベース・カタログに常駐し、カタログ表、カタログ・ビュー、およびこれらのカタログ表にデータを入力するためのいくつかの組み込みストアード・プロシージャーで構成される XML スキーマ・リポジトリーが含まれています。

XSR オブジェクト

XML スキーマ・リポジトリー (XSR) は、XML スキーマ、DTD、または外部エンティティーに含まれる情報のコピーを XSR オブジェクトとして作成する機能をサポートします。この情報を使用して、XML 列に保管された XML インスタンス文書の妥当性検査を行い、処理します。

新規 XSR オブジェクトは、その使用前に登録プロセスで明示的に XSR に追加する必要があり、それによって XML スキーマ、DTD、または外部エンティティーを識別できます。 XSR オブジェクトは、Java アプリケーション、ストアード・プロシージャー、またはコマンド行プロセッサーから登録できます。

最も広く使用されている XSR オブジェクトは XML スキーマです。XSR の各 XML スキーマは 1 つ以上の XML スキーマ文書で構成される場合があります。XML スキーマが複数の文書で構成される場合、登録プロセスを開始するために使用される文書が基本 XML スキーマ文書です。 XML スキーマが 1 つの文書のみで構成される場合、その文書が基本 XML スキーマ文書です。

XSR ストアード・プロシージャーおよびコマンドの構文についての説明は、511 ページの『付録 C. XSR ストアード・プロシージャーおよび XSR コマンド』を参照してください。

XSR オブジェクト登録

XML スキーマや DTD などの外部エンティティーは、XML 文書の処理に使用する前に、XML スキーマ・リポジトリー (XSR) に登録しなければなりません。 XSR に登録すると XSR オブジェクトが作成されます。

ほとんどの XML スキーマを登録するには、アプリケーション・ヒープ・サイズ構成パラメーター (applheapsz) を増やす必要があります。 Windows 32 ビット・オペレーティング・システム上で非常に複雑な XML スキーマを登録するには、エージェント・スタック・サイズ構成パラメーター (agent_stack_sz) も増やすことが必要になる場合があります。これらの構成パラメーターのいずれかの変更方法については、以下の関連リンクを参照してください。

XML スキーマの場合、XSR オブジェクトの登録には以下のステップが関係します。

1. 1 次 XML スキーマ文書を XML スキーマ・リポジトリーに登録します。
2. この XSR オブジェクトに含める追加の XML スキーマ文書を指定します。このステップは、XML スキーマが複数のスキーマ文書によって構成されている場合にだけ必要です。
3. XML スキーマ・リポジトリーで登録プロセスを完了します。

DTD および外部エンティティーの場合、XML スキーマ・リポジトリーでの XSR オブジェクト登録は単一ステップのプロセスです。

XSR オブジェクト登録ステップは、以下のいずれかから実行できます。

- Java アプリケーション
- ストアード・プロシージャー
- コマンド行プロセッサー

CLP コマンドによって必要なファイル情報を渡すことができないため、これらのコマンドを使用してホスト・アプリケーションから XML スキーマを登録できないことに注意してください。CLI/ODBC または JDBC ドライバーによって、DB2 データベースに接続しているアプリケーションから XML スキーマを登録するには、ストアード・プロシージャーを使用してください。

これらの方法の説明では、2 つの XML スキーマ文書 ("PO.xsd" と "address.xsd" で、両方とも C:¥TEMP にローカルに格納される) で構成される XML スキーマの例を使用します。ユーザーは、"user1.POschema" という SQL の 2 部構成の名前でこのスキーマを登録します。この XML スキーマには、schemaProp.xml という名前のプロパティー・ファイルが関連付けられています。このプロパティー・ファイルも、同じ C:¥TEMP ディレクトリーにローカルに格納されます。 2 つの XML スキーマ文書の方には、関連付けられたプロパティーはありません。ユーザーはこのスキーマを外部に公開する URI を "http://myPOschema/PO" と定義します。

特権

DBADM 権限を持つユーザーはだれでも XSR オブジェクトを登録できます。他のすべてのユーザーの場合、その特権は登録プロセス中に提供される SQL スキーマに基づきます。 SQL スキーマが存在しない場合は、スキーマの登録にそのデータ

ベース上の IMPLICIT_SCHEMA 権限が必要です。 SQL スキーマが存在する場合は、スキーマを登録するユーザーにその SQL スキーマに対する CREATEIN 特権が必要です。

XML スキーマの場合、(例えば XSR_REGISTER ストアード・プロシージャーから) XSR オブジェクト登録プロセスを開始するユーザーは、追加の XML スキーマ文書を指定し (該当する場合)、登録プロセスを完了するユーザーでもなければなりません。

XSR オブジェクトに対する USAGE 特権は、XSR オブジェクトの作成者に自動的に付与されます。

ストアード・プロシージャーで XSR オブジェクトを登録する

データベースを作成すると、XML スキーマを登録するのに使用されるストアード・プロシージャーも作成されます。ストアード・プロシージャーを使用する方法で XML スキーマを登録するには、CALL ステートメントで XSR_REGISTER、XSR_ADDSCHEMADOC、および XSR_COMPLETE ストアード・プロシージャーを呼び出します。

文書の登録または追加の際には、XML スキーマ文書が正確かどうかは検査されません。文書の検査は、XML スキーマの登録を完了して初めて行われます。

XML スキーマの登録

1. SYSPROC.XSR_REGISTER ストアード・プロシージャーを呼び出すことによって、基本 XML スキーマ文書の登録を行います。

   ```
   CALL SYSPROC.XSR_REGISTER ('user1', 'POschema', 'http://myPOschema/PO',
                              :content_host_var, NULL)
   ```

2. 登録を完了する前に、基本 XML スキーマと共に含める追加の XML スキーマ文書があれば、それらを追加します。追加の各スキーマ文書を含めることができるのは、それぞれ 1 回ずつのみであることに注意してください。この例では、XML スキーマは 2 つの XML スキーマ文書から構成され、両方とも登録しなければならないので、このステップはオプションではありません。
   XSR_ADDSCHEMADOC ストアード・プロシージャーを使用して、追加の XML スキーマ文書を追加します。次の例は、アドレスのためのスキーマ構成を XSR オブジェクトに追加します。

   ```
   CALL SYSPROC.XSR_ADDSCHEMADOC ('user1', 'POschema', 'http://myPOschema/address',
                              :content_host_var, NULL)
   ```

3. SYSPROC.XSR_COMPLETE ストアード・プロシージャーを呼び出して、登録を完了します。以下の例では、最後のパラメーターは、XML スキーマが分解に使用されないことを示します (値 1 は、これが分解に使用されることを示します)。

   ```
   CALL SYSPROC.XSR_COMPLETE ('user1', 'POschema', :schemaproperty_host_var, 0)
   ```

特権

DBADM 権限を持つユーザーはだれでも XML スキーマを登録できます。他のすべてのユーザーの場合、その特権は登録プロセス中に提供される SQL スキーマに基づきます。 SQL スキーマが存在しない場合は、スキーマの登録にそのデータベー

ス上の IMPLICIT_SCHEMA 権限が必要です。 SQL スキーマが存在する場合は、スキーマを登録するユーザーにその SQL スキーマに対する CREATEIN 特権が必要です。

XSR オブジェクトに対する USAGE 特権は、XSR オブジェクトの作成者に自動的に付与されます。

コマンド行プロセッサーで XSR オブジェクトを登録する

コマンド行プロセッサーで XML スキーマを登録するには、REGISTER XMLSCHEMA、ADD XMLSCHEMA DOCUMENT、および COMPLETE XMLSCHEMA コマンドを使用します。

文書の登録または追加の際には、XML スキーマ文書が正確かどうかは検査されません。文書の検査は、スキーマの登録を完了して初めて行われます。

CLP コマンドによって必要なファイル情報を渡すことができないため、これらのコマンドを使用してホスト・アプリケーションから XML スキーマを登録できないことに注意してください。CLI/ODBC または JDBC ドライバーによって、DB2 データベースに接続しているアプリケーションから XML スキーマを登録するには、ストアード・プロシージャーを使用してください。

XML スキーマの登録

1. REGISTER XMLSCHEMA コマンドを実行することによって、基本 XML スキーマ文書の登録を行います。

   ```
   REGISTER XMLSCHEMA 'http://myPOschema/PO'
   FROM 'file://c:/TEMP/PO.xsd'
   AS user1.POschema
   ```

2. 登録を完了する前に、基本 XML スキーマと共に含める追加の XML スキーマ文書があれば、それらを追加することができます。 ADD XMLSCHEMA DOCUMENT コマンドを使用して、追加の XML スキーマ文書を追加します。追加の各スキーマ文書を含めることができるのは、それぞれ 1 回ずつのみであることに注意してください。次の例では、address スキーマのためのスキーマ構成をストレージに追加しています。

   ```
   ADD XMLSCHEMA DOCUMENT TO user1.POschema
      ADD 'http://myPOschema/address'
      FROM 'file://c:/TEMP/address.xsd'
   ```

3. COMPLETE XMLSCHEMA コマンドを実行して、登録を完了します。

   ```
   COMPLETE XMLSCHEMA user1.POschema
   WITH 'file://c:TEMP/schemaProp.xml'
   ```

特権

DBADM 権限を持つユーザーはだれでも XML スキーマを登録できます。他のすべてのユーザーの場合、その特権は登録プロセス中に提供される SQL スキーマに基づきます。 SQL スキーマが存在しない場合は、スキーマの登録にそのデータベース上の IMPLICIT_SCHEMA 権限が必要です。 SQL スキーマが存在する場合は、スキーマを登録するユーザーにその SQL スキーマに対する CREATEIN 特権が必要です。

XSR オブジェクトに対する USE 特権は、XSR オブジェクトの作成者に自動的に付与されます。

XML スキーマの登録および除去の Java サポート

IBM Data Server Driver for JDBC and SQLJ により、XML スキーマとそのコンポーネントを登録および除去するための Java アプリケーション・プログラムを作成できるメソッドが提供されます。

メソッドは、以下のとおりです。

DB2Connection.registerDB2XMLSchema
XML スキーマを 1 つ以上の XML スキーマ文書を使用して DB2 に登録します。このメソッドには 2 つの形式があります。1 つは InputStream オブジェクトから入力される XML スキーマ文書用の形式で、もう 1 つは `String` 内の XML スキーマ文書用の形式です。

DB2Connection.deregisterDB2XMLObject
DB2 から XML スキーマ定義を除去します。

DB2Connection.updateDB2XmlSchema
登録済みの XML スキーマ内の XML スキーマ文書を登録済みの別の XML スキーマからの XML スキーマ文書で置き換えます。オプションで、内容がコピーされた XML スキーマをドロップします。このメソッドは、DB2 Database for Linux, UNIX, and Windows への接続の場合のみ使用できます。

これらのメソッドを呼び出すには、これらのメソッドをサポートするストアード・プロシージャーが DB2 データベース・サーバーにインストールされていなければなりません。

例: XML スキーマの登録: 以下の例では、入力ストリームから読み取られた単一の XML スキーマ文書 (customer.xsd) を使用して、XML スキーマを DB2 に登録するために registerDB2XmlSchema を使用する方法を示します。登録済みのスキーマの SQL スキーマ名は SYSXSR です。追加のプロパティーは登録されません。

```
public static void registerSchema(
  Connection con,
  String schemaName)
  throws SQLException {
  // Define the registerDB2XmlSchema parameters
  String[] xmlSchemaNameQualifiers = new String[1];
  String[] xmlSchemaNames = new String[1];
  String[] xmlSchemaLocations = new String[1];
  InputStream[] xmlSchemaDocuments = new InputStream[1];
  int[] xmlSchemaDocumentsLengths = new int[1];
  java.io.InputStream[] xmlSchemaDocumentsProperties = new InputStream[1];
  int[] xmlSchemaDocumentsPropertiesLengths = new int[1];
  InputStream xmlSchemaProperties;
  int xmlSchemaPropertiesLength;
  //Set the parameter values
  xmlSchemaLocations[0] = "";
  FileInputStream fi = null;
  xmlSchemaNameQualifiers[0] = "SYSXSR";
  xmlSchemaNames[0] = schemaName;
  try {
    fi = new FileInputStream("customer.xsd");
    xmlSchemaDocuments[0] = new BufferedInputStream(fi);
  } catch (FileNotFoundException e) {
         e.printStackTrace();
```

```
  }
  try {
     xmlSchemaDocumentsLengths[0] = (int) fi.getChannel().size();
     System.out.println(xmlSchemaDocumentsLengths[0]);
  } catch (IOException e1) {
         e1.printStackTrace();
  }
  xmlSchemaDocumentsProperties[0] = null;
  xmlSchemaDocumentsPropertiesLengths[0] = 0;
  xmlSchemaProperties = null;
  xmlSchemaPropertiesLength = 0;
  DB2Connection ds = (DB2Connection) con;
  // Invoke registerDB2XmlSchema
  ds.registerDB2XmlSchema(
    xmlSchemaNameQualifiers,
    xmlSchemaNames,
    xmlSchemaLocations,
    xmlSchemaDocuments,
    xmlSchemaDocumentsLengths,
    xmlSchemaDocumentsProperties,
    xmlSchemaDocumentsPropertiesLengths,
    xmlSchemaProperties,
    xmlSchemaPropertiesLength,
    false);
}
```

例: XML スキーマの除去: 以下の例では、 XML スキーマを DB2 から除去するために deregisterDB2XmlObject を使用する方法を示します。登録済みのスキーマの SQL スキーマ名は SYSXSR です。

```
public static void deregisterSchema(
  Connection con,
  String schemaName)
  throws SQLException {
  // Define and assign values to the deregisterDB2XmlObject parameters
  String xmlSchemaNameQualifier = "SYSXSR";
  String xmlSchemaName = schemaName;
  DB2Connection ds = (DB2Connection) con;
  // Invoke deregisterDB2XmlObject
  ds.deregisterDB2XmlObject(
    xmlSchemaNameQualifier,
    xmlSchemaName);
}
```

例: XML スキーマの更新: 以下の例は、DB2 Database for Linux, UNIX, and Windows への接続のみに適用されます。 XML スキーマの内容を別の XML スキーマの内容で更新するために updateDB2XmlSchema を使用する方法が示されています。コピーされたスキーマは、リポジトリー内に維持されます。登録済みの両方のスキーマの SQL スキーマ名は SYSXSR です。

```
public static void updateSchema(
  Connection con,
  String schemaNameTarget,
  String schemaNameSource)
  throws SQLException {
  // Define and assign values to the updateDB2XmlSchema parameters
  String xmlSchemaNameQualifierTarget = "SYSXSR";
  String xmlSchemaNameQualifierSource = "SYSXSR";
  String xmlSchemaNameTarget = schemaNameTarget;
  String xmlSchemaNameSource = schemaNameSource;
  boolean dropSourceSchema = false;
  DB2Connection ds = (DB2Connection) con;
  // Invoke updateDB2XmlSchema
  ds.updateDB2XmlSchema(
    xmlSchemaNameQualifierTarget,
```

```
    xmlSchemaNameTarget,
  xmlSchemaNameQualifierSource,
  xmlSchemaNameSource,
    dropSourceSchema);
}
```

登録された XSR オブジェクトを変更する

XML スキーマ・リポジトリーで一度登録されると、XSR オブジェクトは、分解、ドロップ、またはコメントとの関連付けを有効または無効にするように変更できます。加えて、登録済み XSR オブジェクトに対する使用特権を付与するまたは取り消すことができます。

このタスクについて

XML スキーマ・リポジトリーは、XML スキーマ、DTD、または他の外部エンティティーに対する XML 文書の従属関係を管理するために使用します。これらの各 XML スキーマ、DTD、または外部エンティティーはまず、XML スキーマ・リポジトリー内の新規 XSR オブジェクトとして登録する必要があります。

XML スキーマの展開

XML スキーマ・リポジトリー (XSR) に登録されている XML スキーマは、新しい互換性のある XML スキーマに展開することができ、その際に既に保管されている XML インスタンス文書を再度妥当性検査する必要はありません。XSR に登録されている XML スキーマだけが更新されます。保管されている XML インスタンス文書は、その URI ID を含め、変更されません。

始める前に

スキーマを展開するには、新しい XML スキーマは元の XML スキーマと互換性がなければなりません。2 つのスキーマに互換性がない場合、XSR_UPDATE ストアード・プロシージャーまたは UPDATE XMLSCHEMA コマンドはエラーを戻し、スキーマの展開は行われません。*XML スキーマの展開のための互換性要件* を参照してください。

このタスクについて

XML スキーマを XSR に展開するには:

手順

1. XSR_REGISTER ストアード・プロシージャーを呼び出すか、REGISTER XMLSCHEMA コマンドを実行して、新しい XML スキーマを XSR に登録します。
2. 最後に、XSR_UPDATE ストアード・プロシージャーを呼び出すか、UPDATE XMLSCHEMA コマンドを実行して、XSR の新しい XML スキーマを更新します。

次のタスク

スキーマの展開により、元の XML スキーマが正常に置き換えられました。いったん展開されると、更新済みの XML スキーマだけが使用可能になります。

XML スキーマの展開のための互換性要件

XML スキーマ・リポジトリー (XSR) 内の XML スキーマの展開プロセスでは、元の XML スキーマと、それを更新するために使用される新しい XML スキーマとが、よく似ている必要があります。

2 つの XML スキーマに互換性がない場合、更新は失敗し、エラー・メッセージが出されます。更新プロセスを続行するには、以下の 10 の互換性の基準を満たす必要があります。説明されている要件を満たさないスキーマの例が示されています。

属性内容

元の XML スキーマの複合タイプの内部で宣言または参照されている属性は、新しい XML スキーマにも記述される必要があります。必須属性が元の XML スキーマに含まれていない場合、それらの属性は新しい XML スキーマにも含められません。

例 1

元の XML スキーマ:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <xs:element name="root">
    <xs:complexType>
      <xs:attribute name="a" type="xs:string"/>
      <xs:attribute name="b" use="optional" type="xs:string"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

新しい XML スキーマ:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <xs:element name="root">
    <xs:complexType>
      <xs:attribute name="a" type="xs:string"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

例 2

元の XML スキーマ:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <xs:element name="root">
    <xs:complexType>
      <xs:attribute name="a" type="xs:string"/>
  </xs:complexType>
  </xs:element>
</xs:schema>
```

新しい XML スキーマ:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <xs:element name="root">
    <xs:complexType>
      <xs:attribute name="a" type="xs:string"/>
```

```
      <xs:attribute name="b" type="xs:string" use="required" />
    </xs:complexType>
  </xs:element>
</xs:schema>
```

要素内容

元の XML スキーマの複合タイプの内部で宣言または参照されている要素は、新しい XML スキーマにも記述される必要があります。必須要素が元の XML スキーマに含まれていない場合、それらの要素は新しい XML スキーマに含められません。オプションの要素だけが追加されます。

例 1

元の XML スキーマ:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <xs:element name="root">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="a" type="xs:string"/>
        <xs:element name="b" minOccurs="0" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

新しい XML スキーマ:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <xs:element name="root">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="a" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

例 2

元の XML スキーマ:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <xs:element name="root">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="a" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

新しい XML スキーマ:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <xs:element name="root">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="a" type="xs:string"/>
        <xs:element name="b" type="xs:string"/>
```

```
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

例 3

元の XML スキーマ:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <xs:element name="a" type="xs:string"/>
  <xs:element name="b" substitutionGroup="a"/>
  <xs:element name="root">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="a"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

新しい XML スキーマ:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <xs:element name="a" type="xs:string"/>
  <xs:element name="b" type="xs:string"/>
  <xs:element name="root">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="a"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

ファセットの競合

新しい XML スキーマの単純タイプのファセット値は、元の XML スキーマに定義されている単純タイプの値の範囲と互換性がなければなりません。

例 1

元の XML スキーマ:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <xs:element name="foo" >
       <xs:simpleType>
        <xs:restriction base="xs:decimal" />
        </xs:simpleType>
      </xs:element>
</xs:schema>
```

新しい XML スキーマ:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <xs:element name="foo">
    <xs:simpleType>
      <xs:restriction base="xs:decimal">
        <xs:totalDigits value="7"/>
      </xs:restriction>    </xs:simpleType>
  </xs:element>
</xs:schema>
```

例 2

元の XML スキーマ:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <xs:element name="foo">
    <xs:simpleType>
      <xs:restriction base="xs:decimal">
        <xs:totalDigits value="7"/>
        <xs:fractionDigits value="3"/>
        <xs:maxInclusive value="300.00"/>
        <xs:minInclusive value="1.0"/>      </xs:restriction>
    </xs:simpleType>
  </xs:element>
</xs:schema>
```

新しい XML スキーマ:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <xs:element name="foo">
    <xs:simpleType>
      <xs:restriction base="xs:decimal">
        <xs:totalDigits value="5"/>
        <xs:fractionDigits value="2"/>
        <xs:pattern value="(0|1|2|3|4|5|6|7|8|9|¥.)*"/>
        <xs:maxInclusive value="100.00"/>
        <xs:minInclusive value="10.00"/>      </xs:restriction>
    </xs:simpleType>
  </xs:element>
</xs:schema>
```

非互換タイプ

すでに挿入されている XML 文書が新しい XML スキーマに対する妥当性検査に失敗した場合、または新しいスキーマに元の XML スキーマとは異なる単純タイプのアノテーションが含まれている場合、新しいスキーマの要素または属性のタイプには互換性がありません。

例

元の XML スキーマ:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
  <xs:element name="a" type="xs:string"/>
</xs:schema>
```

新しい XML スキーマ:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <xs:element name="a" type="xs:integer"/>
</xs:schema>
```

混合内容から非混合内容

複合タイプの内容モデルが元の XML スキーマで混合として宣言されている場合、それを新しい XML スキーマで非混合と宣言することはできません。

例

元の XML スキーマ:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <xs:element name="root">
    <xs:complexType>
      <xs:complexContent mixed="true">
        <xs:restriction base="xs:anyType">
          <xs:attribute name="a" type="xs:string"/>
        </xs:restriction>
      </xs:complexContent>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

新しい XML スキーマ:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <xs:element name="root">
    <xs:complexType>
      <xs:complexContent mixed="false">
        <xs:restriction base="xs:anyType">
          <xs:attribute name="a" type="xs:string"/>
        </xs:restriction>
      </xs:complexContent>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Nillable から非 Nillable

元の XML スキーマの要素宣言で nillable 属性が有効である場合、新しい XML スキーマでも有効にする必要があります。

例

元の XML スキーマ:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <xs:element name="root">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="a" nillable="true" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

新しい XML スキーマ:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <xs:element name="root">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="a" nillable="false" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

除去された要素

元の XML スキーマで宣言されているグローバル要素は、新しい XML スキーマでも宣言される必要があり、抽象化することはできません。

例 1

元の XML スキーマ:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <xs:element name="a" type="xs:string"/>
  <xs:element name="b" type="xs:string"/>
</xs:schema>
```

新しい XML スキーマ:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <xs:element name="a" type="xs:string"/>
</xs:schema>
```

例 2

元の XML スキーマ:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <xs:element name="a" type="xs:string"/>
  <xs:element name="b" type="xs:string"/>
</xs:schema>
```

新しい XML スキーマ:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <xs:element name="a" type="xs:string"/>
  <xs:element name="b" abstract="true" type="xs:string"/>
</xs:schema>
```

除去されたタイプ

元の XML スキーマのグローバル・タイプが別のタイプから派生したものである場合、グローバル・タイプは新しい XML スキーマでも記述される必要があります。

例

元の XML スキーマ:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <xs:element name="root" type="t1"/>
  <xs:complexType name="t1">
    <xs:complexContent>
      <xs:extension base="xs:anyType">
        <xs:attribute name="a" use="required"/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="t2">
    <xs:complexContent>
      <xs:extension base="t1">
        <xs:attribute name="b" use="required"/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType></xs:schema>
```

新しい XML スキーマ:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <xs:element name="root" type="t1"/>
  <xs:complexType name="t1">
    <xs:complexContent>
      <xs:extension base="xs:anyType">
        <xs:attribute name="a" use="required"/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
</xs:schema>
```

単純から複合

元の XML スキーマで単純内容を含む複合タイプは、更新された XML スキーマで複合内容を含むように再定義することはできません。

例

元の XML スキーマ:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <xs:element name="root">
    <xs:complexType>
      <xs:simpleContent>
        <xs:extension base="xs:string">
          <xs:attribute name="a" type="xs:string"/>
        </xs:extension>
      </xs:simpleContent>    </xs:complexType>
  </xs:element>
</xs:schema>
```

新しい XML スキーマ:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <xs:element name="root">
    <xs:complexType>
      <xs:complexContent base="xs:anyType">
        <xs:extension base="xs:anyType">
          <xs:attribute name="a" type="xs:string"/>
        </xs:extension>
      </xs:complexContent>    </xs:complexType>
  </xs:element>
</xs:schema>
```

単純内容

元の XML スキーマと新しい XML スキーマで定義されている単純タイプは同じ基本タイプを共有する必要があります。

例

元の XML スキーマ:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <xs:element name="foo" >
       <xs:simpleType>
        <xs:restriction base="xs:decimal" />
        </xs:simpleType>
      </xs:element>
</xs:schema>
```

新しい XML スキーマ:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <xs:element name="foo" >
       <xs:simpleType>
        <xs:restriction base="xs:string" />
        </xs:simpleType>
      </xs:element>
</xs:schema>
```

シナリオ: XML スキーマの展開

以下のシナリオは、XML スキーマ・リポジトリー (XSR) に登録されている XML スキーマを展開するプロセスについて示しています。

小さな店舗の経営者であるジェーンは、店舗のすべての製品が多数の XML 文書にリストされている 1 つのデータベースを保守しています。こうした XML の製品リストは、以下のスキーマに準拠しています。

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<xsd:complexType name="prodType">
  <xsd:sequence>
    <xsd:element name="name" type="xsd:string" />
    <xsd:element name="sku" type="xsd:string" />
    <xsd:element name="price" type="xsd:integer" />
  </xsd:sequence>
  <xsd:attribute name="color" type="xsd:string" />
  <xsd:attribute name="weight" type="xsd:integer" />
</xsd:complexType>

<xsd:element name="products">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="product" type="prodType" maxOccurs="unbounded" />
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

</xsd:schema>
```

この XML スキーマは、次のコマンドを使用して XSR に最初に登録されています。

```
REGISTER XMLSCHEMA 'http://product'
FROM 'file://c:/schemas/prod.xsd'
AS STORE.PROD

COMPLETE XMLSCHEMA STORE.PROD
```

XML スキーマの登録後、そのスキーマに対して XML 整形式製品リストが妥当性検査されて、店舗のデータベースに挿入されました。

ジェーンは、各製品の名前とともに説明、在庫商品識別番号 (SKU)、および価格がリストに含まれている方が良いと判断しました。新しい XML スキーマを作成して、それに対して既存の XML 文書すべてを再び妥当性検査するのではなく、元の XML スキーマを更新して追加された製品説明に合わせる方をジェーンが選んだとします。新しい "description" 要素を、元の XML スキーマに追加する必要があります。

```
<xsd:complexType name="prodType">
  <xsd:sequence>
    <xsd:element name="name" type="xsd:string" />
    <xsd:element name="sku" type="xsd:string" />
    <xsd:element name="price" type="xsd:integer" />
    <xsd:element name="description" type="xsd:string" minOccurs="0" />
  </xsd:sequence>
  <xsd:attribute name="color" type="xsd:string" />
  <xsd:attribute name="weight" type="xsd:integer" />
</xsd:complexType>
```

挿入するこの XML スキーマ・セグメントで、"minOccurs" 属性は "0" に設定されています。これは重要な点です。そうしなければ、"description" がコンテンツ・モデルの必須要素となり、元のスキーマに対して妥当性検査が行われてからデータベース表に挿入された既存のすべての XML 文書がもはや有効な状態ではなくなってしまうからです。XML スキーマを展開するには、そのスキーマの元のバージョンと新規バージョンとで互換性がなければなりません。詳細は、「*XML スキーマの展開に関する互換性要件*」を参照してください。

更新に行うには、その前に新しい XML スキーマを XSR に登録する必要があります。

```
REGISTER XMLSCHEMA 'http://newproduct'
FROM 'file://c:/schemas/newprod.xsd'
AS STORE.NEWPROD

COMPLETE XMLSCHEMA STORE.NEWPROD
```

これで、ジェーンは XSR_UPDATE ストアード・プロシージャーを使用して更新を実行できます。

```
CALL SYSPROC.XSR_UPDATE(
  'STORE',
  'PROD',
  'STORE',
  'NEWPROD',
  1)
```

元の XML スキーマが展開されます。以前 XML スキーマ STORE.PROD に対して妥当性検査が行われた XML インスタンス文書に関する XSR で管理されている、外部従属関係のすべてが、XML スキーマ STORE.NEWPROD のコンテンツに基づいて更新されます。*dropnewschema* パラメーターはゼロ以外の値を渡されて設定されるので、新しいスキーマ STORE.NEWPROD は元のスキーマが更新されると削除されます。

元の XML スキーマに対する妥当性検査が既に完了している既存の XML 文書については、この更新手順を行っても、妥当性検査が再び行われることはありません。むしろ、更新中に検査が実行され、元の XML スキーマと新しいスキーマに互換性があることが確かめられるため、元の XML スキーマに対する妥当性検査が前に行われた文書は、新しいスキーマに対しても妥当であることが確認できます。上記の例では、2 つの XML スキーマで互換性を持たせるためには、新しい "description" 要素の "minOccurs" 属性を "0" に設定することが必要です。スキーマ展開後に挿入される XML 文書には、新しい更新版の STORE.PROD に対する妥当性検査が行われます。このような文書には店舗の各製品に "description" 要素を含めることができます。

XML スキーマ情報の抽出例

XSR に登録された XML スキーマのリスト

以下の例は、XML スキーマ・リポジトリーに完全に登録済みの XML スキーマを、SQL ステートメントから照会する方法を示しています。 XML スキーマが完全に登録される前に、登録を完了しておく必要があります。

例 1: すべての登録済み XML スキーマをリストする

この例では、XSR に登録されたすべての XML スキーマの SQL スキーマおよび SQL ID が戻されます。

```
SELECT OBJECTNAME, OBJECTSCHEMA
   FROM SYSCAT.XSROBJECTS
   WHERE OBJECTTYPE='S' AND STATUS='C'
```

例 2: ターゲット名前空間およびスキーマ・ロケーションを戻す

この例では、すべての登録済み XML スキーマのターゲット名前空間およびスキーマ・ロケーション (*targetNamespace* および *schemaLocation*) の URI が戻されます。

```
SELECT TARGETNAMESPACE, SCHEMALOCATION
   FROM SYSCAT.XSROBJECTS
   WHERE OBJECTTYPE='S' AND STATUS='C'
```

例 3: オブジェクト情報文書を戻す

この例では、すべての登録済みスキーマのオブジェクト情報文書 (*schemaInfo*) が戻されます。この XML 文書はスキーマ登録時に生成され、XSR に登録された XML スキーマの一部である各 XML スキーマ文書を記述します。

```
SELECT OBJECTINFO
   FROM SYSCAT.XSROBJECTS
   WHERE OBJECTTYPE='S' AND STATUS='C'
```

XSR に登録された XML スキーマのすべてのコンポーネントの取得

以下の例は、登録済みの XML スキーマを構成するすべてのコンポーネント XML スキーマ文書を、XML スキーマ・リポジトリーから取得する方法を示しています。

例 1: 登録済みの XML スキーマのコンポーネント XML スキーマ文書を、ターゲット名前空間およびスキーマ・ロケーション (*targetNamespace* および *schemaLocation*) と共に戻す

```
SELECT COMPONENT, TARGETNAMESPACE, SCHEMALOCATION
   FROM SYSCAT.XSROBJECTCOMPONENTS
   WHERE OBJECTSCHEMA  = ? AND OBJECTNAME = ?
```

コンポーネント XML スキーマ文書は、BLOB 値として戻されます。

例 2: オブジェクト名 CUSTOMER を指定した登録済み XML スキーマの XML スキーマ文書を戻す。SAMPLE データベースに対して実行すると、このステートメントは CUSTOMER 表の INFO 列にある XML 文書の妥当性検査に使用される XML スキーマ文書を戻します。

```
SELECT XMLPARSE(document COMPONENT) FROM SYSCAT.XSROBJECTCOMPONENTS
WHERE OBJECTNAME = 'CUSTOMER'
```

XML スキーマ文書は、XML 値として戻されます。

XML 文書の XML スキーマの取得

以下の例は、XML 文書と関連付けられた XML スキーマを、XML スキーマ・リポジトリーから取得する方法を示しています。

例 1: XML 文書の XML スキーマのオブジェクト ID の取得

```
SELECT DOC, XMLXSROBJECTID(DOC)
   FROM T
```

例 2: XML 文書の XML スキーマのオブジェクト ID および 2 部の SQL ID の取得

```
SELECT XMLXSROBJECTID(DOC),
   CAT.OBJECTSCHEMA, CAT.OBJECTNAME
   FROM T, SYSCAT.XSROBJECTS AS CAT
   WHERE XMLXSROBJECTID(DOC) = CAT.OBJECTID
```

第 9 章 XML データ移動

XML データ移動のサポートは、ロード、インポート、およびエクスポート・ユーティリティーによって提供されます。 ADMIN_MOVE_TABLE ストアード・プロシージャーを使用すると、表をオフラインにしないでも XML 列を含む表を移動することができます。

XML データのインポート

インポート・ユーティリティーを使用して、 XML 文書を通常のリレーショナル表に挿入できます。インポートできるのは、整形式 XML 文書だけです。

IMPORT コマンドの XML FROM オプションを使用して、インポートする XML 文書の場所を指定します。 XMLVALIDATE オプションは、インポートされた文書を妥当性検査する方法を指定します。インポートされた XML データが、IMPORT コマンドで指定されたスキーマに対して、ソース XML 文書内のスキーマ・ロケーション・ヒントによって識別されるスキーマに対して、またはメイン・データ・ファイル内の XML Data Specifier によって識別されるスキーマに対して、妥当性検査されるように選択できます。また、XMLPARSE オプションを使用して、XML 文書がインポートされるときの空白文字の処理方法を指定できます。 `xmlchar` および `xmlgraphic` ファイル・タイプ修飾子によって、インポートされる XML データのエンコード特性を指定できます。

XML データのロード

ロード・ユーティリティーは、大量の XML データを表に挿入するための効果的な方法を提供します。このユーティリティーはまた、ユーザー定義のカーソルからロードする機能など、インポート・ユーティリティーでは使用できない特定のオプションを使用することができます。

IMPORT コマンドと同じように LOAD コマンドでも、ロードする XML データの場所、XML データの妥当性検査オプション、および空白文字の処理方法を指定できます。 IMPORT と同様に、`xmlchar` および `xmlgraphic` ファイル・タイプ修飾子を使用して、ロードされた XML データのエンコード特性を指定できます。

XML データのエクスポート

データは、XML データ・タイプの列を 1 列以上含む表からエクスポートできます。エクスポートされた XML データは、エクスポートされたリレーショナル・データを含むメイン・データ・ファイルとは別個のファイルに格納されます。各エクスポート XML 文書についての情報は、エクスポートされたメインのデータ・ファイル内で XML データ指定子 (XDS) によって表されます。 XDS は、XML 文書が保管されるシステム・ファイルの名前、このファイル内の XML 文書の正確な場所と長さ、および XML 文書の妥当性検査に使用される XML スキーマを指定するストリングです。

EXPORT コマンドの XMLFILE、XML TO、および XMLSAVESCHEMA パラメーターを使用することにより、エクスポートされた XML 文書の格納方法についての

詳細を指定できます。 xmlinsepfiles、xmlnodeclaration、xmlchar、および xmlgraphic ファイル・タイプ修飾子により、エクスポートされた XML データの保管場所およびエンコード方式に関する詳細を指定できます。

オンラインでの表の移動

ADMIN_MOVE_TABLE ストアード・プロシージャーは、データをオンラインでアクセス可能な状態のままで、そのアクティブ表のデータを同じ名前の新しい表オブジェクトに移動できます。表には、XML データ・タイプの 1 つ以上の列を含めることができます。コスト、スペース、移動のパフォーマンス、トランザクション・オーバーヘッドよりも可用性の方を重視する場合には、オフラインの表移動ではなく、オンラインの表移動を使用してください。

このプロシージャーは、1 回呼び出すことも、複数回 (プロシージャーが実行する各操作につき 1 回ずつ) 呼び出すこともできます。複数の呼び出しを行う場合、移動のキャンセルや、ターゲット表が更新のためにオフラインになるタイミングの制御などの追加オプションを使用できます。

XML データの移動に関する重要な考慮事項

XML データをインポートまたはエクスポートする際には、考慮すべきいくつかの制限事項、前提条件、および注意事項があります。XML データをインポートまたはエクスポートする前に、こうした考慮事項を検討してください。

XML データをエクスポートまたはインポートする際に、以下の考慮事項に注意を払ってください。

- エクスポートされた XML データは、エクスポートされたリレーショナル・データを含むメイン・データ・ファイルとは別個の場所に常に保管されます。
- デフォルトで、エクスポート・ユーティリティーは XML データを Unicode で書き込みます。 xmlchar ファイル・タイプ修飾子を使用して、XML データを文字コード・ページで書き込むようにするか、xmlgraphic ファイル・タイプ修飾子を使用して、アプリケーションのコード・ページに関わりなく XML データを UTF-16 (グラフィック・コード・ページ) で書き込むようにします。
- XML データは Unicode 以外のデータベースで格納でき、XML 列に挿入されるデータは挿入前にデータベース・コード・ページから UTF-8 に変換されます。XML 構文解析中に代替文字が使用されないようにするため、挿入する文字データはデータベース・コード・ページに含まれるコード・ポイントのみを使用して構成する必要があります。 enable_xmlchar 構成パラメーターを no に設定すると、XML 構文解析中に文字データ・タイプの挿入がブロックされて、BIT DATA、BLOB や XML など、コード・ページ変換を実施しないデータ・タイプの挿入が制限されます。
- XML データをインポートまたはロードする際、インポートする XML 文書にエンコード属性を含む宣言タグが含まれていない限り、XML データは Unicode であると想定されます。xmlchar ファイル・タイプ修飾子を使用して、インポートする XML 文書が文字コード・ページでエンコードされるように指定できます。一方、xmlgraphic ファイル・タイプ修飾子は、インポートする XML 文書が UTF-16 でエンコードされることを指定します。

- インポート・ユーティリティーとロード・ユーティリティーは、整形式でない文書を含む行を拒否します。
- XMLVALIDATE オプションがインポート・ユーティリティーまたはロード・ユーティリティーに指定されている場合、マッチング・スキーマに対する妥当性検査が成功した文書は、表に挿入されるときに、妥当性検査に使用されたスキーマに関する情報のアノテーションが付けられます。マッチング・スキーマに対する妥当性検査が失敗した文書を含む行は、拒否されます。
- XMLVALIDATE オプションがインポート・ユーティリティーまたはロード・ユーティリティーに指定され、複数の XML スキーマを使用して XML 文書を妥当性検査する場合、カタログ・キャッシュ・サイズ構成パラメーター **catalogcache_sz** を増やす必要がある場合があります。**catalogcache_sz** の値を増やすことが適切でないか不可能な場合、単一のインポート・コマンドまたはロード・コマンドを、もっと少ないスキーマ文書しか使用しない複数のコマンドに分けることができます。
- XQuery ステートメントを指定して XML データをエクスポートする場合、整形式 XML 文書ではない Query および XPath データ・モデル (XDM) のインスタンスをエクスポートできます。 XML データ・タイプで定義された列には完全な整形式の XML 文書だけしか含めることができないので、エクスポートされた整形式ではない XML 文書は XML 列に直接インポートできません。
- ロード中の **CPU_PARALLELISM** 設定は、統計が収集されている場合、1 に縮小されます。
- XML ロード操作を続行するには、共有ソート・メモリーを使用することが必要です。 **SHEAPTHRES_SHR** または **INTRA_PARALLEL** を有効にするか、または接続コンセントレーターをオンにします。デフォルトでは **SHEAPTHRES_SHR** は設定されているので、共有ソート・メモリーはデフォルト構成で使用できます。
- XML 列を含む表をロードする場合、LOAD コマンドの **SOURCEUSEREXIT** オプションまたは **SAVECOUNT** パラメーターを指定することはできません。
- LOAD コマンドを使用する際、LOB ファイルの場合と同様に、XML ファイルもサーバー・サイドに存在する必要があります。
- XML データをパーティション・データベース環境にある複数のデータベース・パーティションにロードする場合、XML データを含むファイルはすべてのデータベース・パーティションにアクセス可能でなければなりません。例えば、ファイルをコピーするか、NFS マウントを作成して、ファイルをアクセス可能にすることができます。

Query および XPath のデータ・モデル

SQL で使用できる XQuery 関数を使用するか、または XQuery を直接呼び出すことにより、データベース表内で XML データにアクセスできます。 Query および XPath データ・モデル (XDM) のインスタンスは、整形式 XML 文書、ノードのシーケンス、アトミック値のシーケンス、またはノードとアトミック値の任意の組み合わせとすることができます。

個々の XDM インスタンスは、EXPORT コマンドによって 1 つ以上の XML ファイルに書き込むことができます。

インポートおよびエクスポート時の LOB および XML ファイルの性質

LOB および XML ファイルは、データをインポートおよびエクスポートする時に使用できる特定の性質および互換性を共有します。

エクスポート

データをエクスポートするときに LOBS TO オプションを付けて 1 つ以上の LOB パスを指定する場合、エクスポート・ユーティリティーはパスの間を循環し、それぞれの連続した LOB 値を適切な LOB ファイルに書き込みます。同様に、1 つ以上の XML パスが XML TO オプションで指定された場合、エクスポート・ユーティリティーはそれらのパスの間で循環して、それぞれの連続した XQuery および XPath データ・モデル (XDM) インスタンスを該当する XML ファイルに書き込みます。デフォルトでは、LOB 値および XDM インスタンスは、エクスポートされるリレーショナル・データが書き込まれているパスと同じパスに書き込まれます。
LOBSINSEPFILES または XMLINSEPFILES ファイル・タイプ修飾子が設定されているのでないかぎり、LOB ファイルと XML ファイルは両方とも複数の値を同じファイルに連結できます。

LOBFILE オプションを指定すると、エクスポート・ユーティリティーによって生成される LOB ファイルのベース名を指定することができます。同様に、XMLFILE オプションを指定すると、エクスポート・ユーティリティーによって生成される XML ファイルのベース名を指定することができます。エクスポート・データ・ファイルの名前に拡張子 .lob を付けたものが、デフォルトの LOB ファイルのベース名になります。エクスポート・データ・ファイルの名前に拡張子 .xml を付けたものが、デフォルトの XML ファイルのベース名になります。したがって、エクスポート LOB ファイルまたは XML ファイルの完全名は、ベース名、3 桁の数値の拡張子、そして拡張子 .lob または .xml をこの順番でつなぎ合わせたもので構成されます。

インポート

データをインポートするとき、LOB ロケーション指定子 (LLS) には XML ターゲット列との互換性があり、XML データ指定子 (XDS) には LOB ターゲット列との互換性があります。 LOBS FROM オプションが指定されない場合、インポートする LOB ファイルは、入力リレーショナル・データ・ファイルと同じパスにあると見なされます。同様に、XML FROM オプションが指定されない場合、インポートする XML ファイルは、入力リレーショナル・データ・ファイルと同じパスにあると見なされます。

エクスポートの例

以下の例では、LOB 値はすべて /mypath/t1export.del.001.lob ファイルに書き込まれ、XDM インスタンスはすべて /mypath/t1export.del.001.xml ファイルに書き込まれます。

```
EXPORT TO /mypath/t1export.del OF DEL MODIFIED BY LOBSINFILE
SELECT * FROM USER.T1
```

以下の例では、最初の LOB 値は /lob1/t1export.del.001.lob ファイルに書き込まれ、2 番目は /lob2/t1export.del.002.lob ファイルに書き込まれ、3 番目は

/lob1/t1export.del.001.lob に付加され、4 番目は /lob2/t1export.del.002.lob に付加され、以降このパターンで付加されていきます。

```
EXPORT TO /mypath/t1export.del OF DEL LOBS TO /lob1,/lob2
MODIFIED BY LOBSINFILE SELECT * FROM USER.T1
```

以下の例では、最初の XDM インスタンスは /xml1/xmlbase.001.xml ファイルに書き込まれ、2 番目は /xml2/xmlbase.002.xml ファイル、3 番目は /xml1/xmlbase.003.xml、4 番目は /xml2/xmlbase.004.xml、というパターンで書き込まれていきます。

```
EXPORT TO /mypath/t1export.del OF DEL XML TO /xml1,/xml2 XMLFILE xmlbase
   MODIFIED BY XMLINSEPFILES SELECT * FROM USER.T1
```

インポートの例

XML 列を 1 つ含む「mytable」という表があり、以下の IMPORT コマンドがあるとします。

```
IMPORT FROM myfile.del of del LOBS FROM /lobpath XML FROM /xmlpath
MODIFIED BY LOBSINFILE XMLCHAR replace into mytable
```

「myfile.del」に以下のデータが含まれるとします。

```
mylobfile.001.lob.123.456/
```

この場合、インポート・ユーティリティーは、/lobpath/mylobfile.001.lob ファイルから、ファイル・オフセットを 123 から開始し、長さ 456 バイトで XML 文書のインポートを試行します。

ファイル「mylobfile.001.lob」は XML パスではなく LOB パスにあると見なされます。値が XML データ指定子 (XDS) ではなく LOB ロケーション指定子 (LLS) によって参照されているからです。

XMLCHAR ファイル・タイプ修飾子が指定されているので、文書は文字コード・ページでエンコードされるものと見なされます。

XML データ指定子

エクスポート、インポート、およびロード・ユーティリティーを使用して移動される XML データは、メイン・データ・ファイルとは別のファイルに格納する必要があります。 XML データは、メイン・データ・ファイル内で XML データ指定子 (XDS) を使用して表されます。

XDS は XDS という名前の XML タグによって表されるストリングであり、列内の実際の XML データについての情報を記述する属性が付随しています。そのような情報には、実際の XML データが含まれているファイルの名前、およびそのファイル内の XML データのオフセットおよび長さが含まれます。 XDS の属性について、以下のリストで説明します。

FIL XML データが入っているファイルの名前。Named PIPE を指定することはできません。Named PIPE からの XML 文書のインポートとロードはサポートされていません。

OFF FIL 属性で名前指定されているファイル中の XML データのバイト・オフセット。このオフセットは 0 から始まります。

LEN FIL 属性で名前指定されているファイル中の XML データのバイト単位の長さ。

SCH この XML 文書の妥当性検査に使用する XML スキーマの完全修飾 SQL ID。この SQL ID のスキーマと名前のコンポーネントは、それぞれこの XML スキーマに対応する SYSCAT.XSROBJECTS カタログ表の行の「OBJECTSCHEMA」および「OBJECTNAME」値として保管されます。

XDS はデータ・ファイル中で文字フィールドとして解釈され、ファイル形式の文字の列に関する構文解析動作の対象になります。例えば、区切り文字付き ASCII ファイル形式 (DEL) の場合、区切り文字が XDS 中にあれば、二重にしなければなりません。属性値の中の特殊文字 <、>、&、'、" は常にエスケープしなければなりません。大文字と小文字の区別のあるオブジェクト名は、" 文字エンティティーで囲まなければなりません。

例

値が abc&"def".del の FIL 属性について考えてみましょう。区切り文字付き ASCII ファイルにこの XDS を組み込むには、区切り文字が " 文字であれば、" 文字を二重にして、特殊文字をエスケープします。

```
<XDS FIL=""abc&amp;&quot;def&quot;.del"" />
```

以下の例は、区切り文字付き ASCII データ・ファイル中にある XDS を示しています。XML データはファイル xmldocs.xml.001 に保管され、バイト・オフセットは 100 で始まり長さは 300 バイトになります。この XDS は二重引用符で区切られる ASCII ファイル中にあるので、XDS タグ自体の中の二重引用符は二重にしなければなりません。

```
"<XDS FIL = ""xmldocs.xml.001"" OFF=""100"" LEN=""300"" />"
```

以下の例は、完全修飾 SQL ID ANTHONY.purchaseOrderTest を示しています。XDS 中で、ID の大文字と小文字の区別がある部分は " 文字エンティティーで囲まなければなりません。

```
"<XDS FIL='/home/db2inst1/xmlload/a.xml' OFF='0' LEN='6758'
   SCH='ANTHONY.&amp;quot;purchaseOrderTest&amp;quot;' />"
```

XML データのエクスポート

XML データをエクスポートする際、結果として作成される QDM (XQuery データ・モデル) インスタンスは、エクスポートされるリレーショナル・データを含むメイン・データ・ファイルとは別のファイル (1 つまたは複数) に書き込まれます。XMLFILE オプションおよび XML TO オプションのどちらも指定されていない場合でも、そのようになります。

デフォルトで、エクスポートされた QDM インスタンスはすべて同じ XML ファイルに連結されます。 XMLINSEPFILES ファイル・タイプ修飾子を使用して、各 QDM インスタンスが別のファイルに書き込まれるように指定できます。

ただし XML データは、メイン・データ・ファイル内で XML データ指定子 (XDS) によって表されます。 XDS は XDS という名前の XML タグによって表されるストリングであり、列内の実際の XML データについての情報を記述する属性が付随

しています。そのような情報には、実際の XML データが含まれているファイルの名前、およびそのファイル内の XML データのオフセットおよび長さが含まれます。

エクスポートされた XML ファイルの宛先パスおよびベース名は、XML TO および XMLFILE オプションを使用して指定できます。 XML TO または XMLFILE オプションが指定されている場合、エクスポートされた XML ファイルの名前として XDS の FIL 属性に格納される名前の形式は xmlfilespec.xxx.xml です (xmlfilespec は XMLFILE オプションに指定された値、xxx はエクスポート・ユーティリティーによって生成された XML ファイルの順序番号)。そうでない場合、エクスポートされた XML ファイル名の形式は exportfilename.xxx.xml となります (exportfilename は EXPORT コマンドのために指定されるエクスポートされた出力ファイルの名前、xxx はエクスポート・ユーティリティーによって生成された XML ファイルの順序番号)。

デフォルトで、エクスポートされた XML ファイルはエクスポートされたデータ・ファイルのパスに書き込まれます。エクスポートされた XML ファイルのデフォルトのベース名は、エクスポートされたデータ・ファイル名に 3 桁の順序番号、および .xml 拡張子を付加したものです。

例

以下の例では、4 つの列および 2 つの行を含む表 USER.T1 を想定します。

```
C1 INTEGER
C2 XML
C3 VARCHAR(10)
C4 XML
```

表 27. USER.T1

C1	C2	C3	C4
2	<?xml version="1.0" encoding="UTF-8" ?><note time="12:00:00"><to>You</ to><from> Me</ from><heading>note1</heading> <body>Hello World!</body></note>	'char1'	<?xml version="1.0" encoding="UTF-8" ?><note time="13:00:00"><to>Him</ to><from> Her</ from><heading>note2</heading>< body>Hello World!</body></note>
4	NULL	'char2'	?xml version="1.0" encoding="UTF-8" ?><note time="14:00:00">to>Us</ to><from> Them</ from><heading>note3</heading> <body>Hello World!</body></note>

例 1

以下のコマンドは、USER.T1 の内容を Delimited ASCII (DEL) 形式でファイル "/mypath/t1export.del" にエクスポートします。 XML TO および XMLFILE オプションが指定されていないので、列 C2 および C4 に含まれる XML 文書はメインのエクスポート・ファイル "/mypath" と同じパスに書き込まれます。これらのファイルのベース名は、"t1export.del.xml" です。 XMLSAVESCHEMA オプションは、XML スキーマ情報がエクスポート手順の実行中に保存されることを示します。

```
EXPORT TO /mypath/t1export.del OF DEL XMLSAVESCHEMA SELECT * FROM USER.T1
```

エクスポート・ファイル "/mypath/t1export.del" には、以下が含まれます。

```
2,"<XDS FIL='t1export.del.001.xml' OFF='0' LEN='144' />","char1",
"<XDS FIL='t1export.del.001.xml' OFF='144' LEN='145' />"
4,,"char2","<XDS FIL='t1export.del.001.xml' OFF='289'
LEN='145' SCH='S1.SCHEMA_A' />"
```

エクスポートされた XML ファイル "/mypath/t1export.del.001.xml" には、以下が含まれます。

```
<?xml version="1.0" encoding="UTF-8" ?><note time="12:00:00"><to>You</to>
   <from>Me</from><heading>note1</heading><body>Hello World!</body>
   </note><?xml version="1.0" encoding="UTF-8" ?><note time="13:00:00"><to>Him
   </to><from>Her</from><heading>note2</heading><body>Hello World!
   </body></note><?xml version="1.0" encoding="UTF-8" ?><note time="14:00:00">
   <to>Us</to><from>Them</from>heading>note3</heading><body>
   Hello World!</body></note>
```

例 2

以下のコマンドは、USER.T1 の内容を DEL 形式でファイル "t1export.del" にエクスポートします。列 C2 および C4 に含まれる XML 文書は、パス "/home/user/xmlpath" に書き込まれます。 XML ファイルはベース名 "xmldocs" を使用して名前が付けられ、複数のエクスポートされた XML 文書が同じ XML ファイルに書き込まれます。 XMLSAVESCHEMA オプションは、XML スキーマ情報がエクスポート手順の実行中に保存されることを示します。

```
EXPORT TO /mypath/t1export.del OF DEL XML TO /home/user/xmlpath
XMLFILE xmldocs XMLSAVESCHEMA SELECT * FROM USER.T1
```

エクスポートされた DEL ファイル "/home/user/t1export.del" には、以下が含まれます。

```
2,"<XDS FIL='xmldocs.001.xml' OFF='0' LEN='144' />","char1",
"<XDS FIL='xmldocs.001.xml' OFF='144' LEN='145' />"
4,,"char2","<XDS FIL='xmldocs.001.xml' OFF='289'
LEN='145' SCH='S1.SCHEMA_A' />"
```

エクスポートされた XML ファイル "/home/user/xmlpath/xmldocs.001.xml" には、以下のものが含まれます。

```
<?xml version="1.0" encoding="UTF-8" ?><note time="12:00:00"><to>You</to>
   <from>Me</from><heading>note1</heading><body>Hello World!</body>
   </note><?xml version="1.0" encoding="UTF-8" ?><note time="13:00:00">
   <to>Him</to><from>Her</from><heading>note2</heading><body>
   Hello World!</body></note><?xml version="1.0" encoding="UTF-8" ?>
   <note time="14:00:00"><to>Us</to><from>Them</from><heading>
   note3</heading><body>Hello World!</body></note>
```

例 3

以下のコマンドは例 2 と似ていますが、それぞれのエクスポートされた XML 文書が別の XML ファイルに書き込まれることが異なります。

```
EXPORT TO /mypath/t1export.del OF DEL XML TO /home/user/xmlpath
XMLFILE xmldocs MODIFIED BY XMLINSEPFILES XMLSAVESCHEMA
SELECT * FROM USER.T1
```

エクスポート・ファイル "/mypath/t1export.del" には、以下が含まれます。

```
2,"<XDS FIL='xmldocs.001.xml' />","char1","XDS FIL='xmldocs.002.xml' />"
4,,"char2","<XDS FIL='xmldocs.004.xml' SCH='S1.SCHEMA_A' />"
```

エクスポートされた XML ファイル "/home/user/xmlpath/xmldocs.001.xml" には、以下のものが含まれます。

```
<?xml version="1.0" encoding="UTF-8" ?><note time="12:00:00"><to>You</to>
   <from>Me</from><heading>note1</heading><body>Hello World!</body>
   </note>
```

エクスポートされた XML ファイル "/home/user/xmlpath/xmldocs.002.xml" には、以下のものが含まれます。

```
?xml version="1.0" encoding="UTF-8" ?>note time="13:00:00">to>Him/to>
   from>Her/from>heading>note2/heading>body>Hello World!/body>
   /note>
```

エクスポートされた XML ファイル "/home/user/xmlpath/xmldocs.004.xml" には、以下のものが含まれます。

```
<?xml version="1.0" encoding="UTF-8" ?><note time="14:00:00"><to>Us</to>
   <from>Them</from><heading>note3</heading><body>Hello World!</body>
   </note>
```

例 4

次のコマンドは、 XQuery の結果を XML ファイルに書き込みます。

```
EXPORT TO /mypath/t1export.del OF DEL XML TO /home/user/xmlpath
XMLFILE xmldocs MODIFIED BY XMLNODECLARATION select
xmlquery( '$m/note/from/text()' passing by ref c4 as "m" returning sequence)
   from USER.T1
```

エクスポートされた DEL ファイル "/mypath/t1export.del" には、以下が含まれます。

```
"<XDS FIL='xmldocs.001.xml' OFF='0' LEN='3' />"
"<XDS FIL='xmldocs.001.xml' OFF='3' LEN='4' />"
```

エクスポートされた XML ファイル "/home/user/xmlpath/xmldocs.001.xml" には、以下のものが含まれます。

```
HerThem
```

注: 特定の XQuery の結果は、整形式 XML 文書を生成しません。そのため、この例でエクスポートされたファイルを XML 列に直接インポートすることはできません。

XML データのインポート

インポート・ユーティリティーを使用することにより、DB2 Database for Linux, UNIX, and Windows ソース・データ・オブジェクトの表名またはニックネームを使用して XML データを XML 表列にインポートできます。

データを XML 表列にインポートするとき、XML FROM オプションを使用して入力 XML データ (1 つまたは複数) のパスを指定できます。例えば、以前にエクスポートされた XML ファイル "/home/user/xmlpath/xmldocs.001.xml" では、以下のコマンドを使用してデータを表にインポートして戻すことができます。

```
IMPORT FROM t1export.del OF DEL XML FROM /home/user/xmlpath INSERT INTO USER.T1
```

スキーマに対して挿入された文書を妥当性検査する

XMLVALIDATE オプションにより、XML 文書がインポートされる際に、それらが XML スキーマに準拠しているかどうかの妥当性検査を行えます。次の例では、XML 文書がエクスポートされた時点で保存されたスキーマ情報に準拠しているかどうかに関して、着信 XML 文書が妥当性検査されます。

```
IMPORT FROM t1export.del OF DEL XML FROM /home/user/xmlpath XMLVALIDATE
     USING XDS INSERT INTO USER.T1
```

構文解析オプションの指定

XMLPARSE オプションを使用して、インポートされた XML 文書の空白文字を保存するかストリップするかを指定できます。次の例では、XML 文書がエクスポートされたときに保管された XML スキーマ情報に対してすべてのインポートされた XML 文書が妥当性検査されて、これらの文書は空白文字を保存しながら構文解析されます。

```
IMPORT FROM t1export.del OF DEL XML FROM /home/user/xmlpath XMLPARSE PRESERVE
WHITESPACE XMLVALIDATE USING XDS INSERT INTO USER.T1
```

XML データのロード

ロード・ユーティリティーを使用して、大量の XML データを表に効率的に移動できます。

データを XML 表列にロードする場合、XML FROM オプションを使用して入力 XML データ (1 つまたは複数) のパスを指定できます。例えば、データを XML ファイル /home/user/xmlpath/xmlfile1.xml からロードするには、以下のコマンドを使用できます。

```
LOAD FROM data1.del OF DEL XML FROM /home/user/xmlpath INSERT INTO USER.T1
```

区切られた ASCII 入力ファイル data1.del には、ロードする XML データの場所を説明する XML データ指定子 (XDS) が含まれます。例えば、以下の XDS は、ファイル xmldata.ext 内のオフセット 123 バイトにある、長さが 456 バイトの XML 文書について説明しています。

```
<XDS FIL='xmldata.ext' OFF='123' LEN='456' />
```

宣言済みカーソルを使用した XML データのロードがサポートされています。以下の例ではカーソルを宣言し、そのカーソルと **LOAD** コマンドを使用して、表 CUSTOMERS のデータを表 LEVEL1_CUSTOMERS に追加します。

```
DECLARE cursor_income_level1 CURSOR FOR
   SELECT * FROM customers
   WHERE XMLEXISTS('$DOC/customer[income_level=1]');

LOAD FROM cursor_income_level1 OF CURSOR INSERT INTO level1_customers;
```

XML データを XML 列にロードする際に、**LOAD** コマンドの ANYORDER ファイル・タイプ修飾子がサポートされています。

ロード中、タイプ XML の列については分散統計は収集されません。

パーティション・データベース環境での XML データのロード

データベース・パーティション間で分散される表の場合、複数の XML データ・ファイルから XML データを表に並行してロードできます。XML データをファイルから表にロードする場合、ロードが実行されるすべてのデータベース・パーティションで XML データ・ファイルが読み取り可能でなければなりません。

スキーマに対して挿入された文書を妥当性検査する

XMLVALIDATE オプションにより、 XML 文書がロードされるときに、それらを XML スキーマに対して妥当性検査できます。次の例では、区切られた ASCII 入力ファイル data2.del 内で XDS によって識別されるスキーマに対して、着信 XML 文書が妥当性検査されます。

```
LOAD FROM data2.del OF DEL XML FROM /home/user/xmlpath XMLVALIDATE
     USING XDS INSERT INTO USER.T2
```

この場合、XDS には XML スキーマの完全修飾 SQL ID "S1.SCHEMA_A" のある SCH 属性が妥当性検査で使用するために含まれます。

```
<XDS FIL='xmldata.ext' OFF='123' LEN='456' SCH='S1.SCHEMA_A' />
```

構文解析オプションの指定

XMLPARSE オプションを使用して、ロードされた XML 文書の空白文字を保存するかストリップするかを指定できます。次の例では、SQL ID "S2.SCHEMA_A" のあるスキーマに対してすべてのロードされた XML 文書が妥当性検査されて、これらの文書は空白文字を保存しながら構文解析されます。

```
LOAD FROM data2.del OF DEL XML FROM /home/user/xmlpath XMLPARSE PRESERVE
WHITESPACE XMLVALIDATE USING SCHEMA S2.SCHEMA_A INSERT INTO USER.T1
```

XML データのロード時の索引付けエラーの解決

索引付けエラーのために失敗するロード操作は、**db2diag** ログ・ファイルとインポート・ユーティリティーを一緒に使用して XML データの問題となる値を識別し、訂正することで解決できます。

このタスクについて

ロード操作でエラー・メッセージ SQL20305N (sqlcode -20305) が返される場合、これは 1 つ以上の XML ノード値に索引付けできなかったことを示します。エラー・メッセージはエラーの理由コードを出力します。コマンド行プロセッサーに ? SQL20305N と入力して、対応する理由コードの説明とユーザー応答を検索します。

挿入操作中の索引付けの問題については、生成される XQuery ステートメントが **db2diag** ログ・ファイルに出力されて、文書内の失敗した XML ノード値を見つける上で役立ちます。失敗した XML ノード値を見つけるために XQuery ステートメントを使用する方法についての詳細は、『XML 索引付けの一般的な問題』を参照してください。

しかし、ロード操作中の索引付けの問題については、生成される XQuery ステートメントは **db2diag** ログ・ファイルに出力されません。これらの XQuery ステートメントを生成するには、ロードされなかった失敗した行に対してインポート・ユーティリティーを実行する必要があります。リジェクトされた行は表には存在しないた

め、XQuery ステートメントを失敗した文書に対して実行することはできません。この問題を解決するため、同じ定義を持つ新規表を、索引を付けずに作成する必要があります。その後、失敗した行を新規表にロードでき、さらに XQuery ステートメントを新規表に対して実行し、文書内の失敗した XML ノード値を見つけることができるようになります。

索引付けのエラーを解決するには、以下のステップを実行します。

手順

1. 出力情報にあるレコード番号を使用して、ロード操作中にどの行がリジェクトされたかを調べます。
2. リジェクトされた行のみが含まれる .del ファイルを作成します。
3. 元の表 (T1) と同じ列を持つ新規表 (例えば T2) を作成します。新規表には索引を作成しないでください。
4. リジェクトされた行を新規表 T2 にロードします。
5. 元の表 T1 のそれぞれのリジェクトされた行について以下を行います。
   a. リジェクトされた行を T1 にインポートして、SQL20305N メッセージを受け取ります。エラーが最初に出されたところでインポートは停止します。
   b. **db2diag** ログ・ファイルを調べ、生成された XQuery ステートメントを見つけます。入力文書で失敗したノード値を検索するため、**db2diag** ログ・ファイルでストリング 'SQL20305N' を検索し、理由コード番号を突き合わせます。理由コードの後に一連の指示があり、続いてエラーの原因となった文書内の問題値を見つけるために使用できる、生成された XQuery ステートメントがあります。
   c. 新規表 T2 を使用するように XQuery ステートメントを変更します。
   d. T2 に対して XQuery ステートメントを実行し、文書内の問題値を見つけます。
   e. 文書が含まれる .xml ファイルの中の問題値を修正します。
   f. ステップ a に戻り、リジェクトされた行を再度 T1 にインポートします。インポートの停止の原因となった行は、今度は正常に挿入されるはずです。他にもリジェクトされた行が .del ファイルにある場合、インポート・ユーティリティーは次のエラーで停止し、別の SQL20305N メッセージが出力されます。インポートが正常に実行されるようになるまで、これらのステップを続けます。

例

以下の例では、索引 BirthdateIndex が *date* データ・タイプに対して作成されています。 REJECT INVALID VALUES オプションが指定されているため、*/Person/Confidential/Birthdate* の XML パターン値は *date* データ・タイプに対してすべて有効となる必要があります。このデータ・タイプにキャストできない XML パターン値がある場合、エラーが返されます。

以下の XML 文書を使用すると 5 つの行がロードされるはずですが、Birthdate 値を索引付けできないため、最初と 4 番目の行はリジェクトされます。ファイル `person1.xml` では、値 `March 16, 2002` が正しい日付形式ではありません。ファイル `person4.xml` では、値 `20000-12-09` の年にゼロが余分にあるため、有効な XML

日付値ではありますが、DB2 で許可される年 (0001 から 9999) の範囲外となっています。例を簡潔にするため、出力例の一部は編集されています。

ロードする 5 つの XML ファイルは以下のとおりです。

person1.xml (Birthdate 値は無効です)

```
<?xml version="1.0"?>
<Person gender="Male">
  <Name>
    <Last>Cool</Last>
    <First>Joe</First>
  </Name>
  <Confidential>
    <Age unit="years">5</Age>
    <Birthdate>March 16, 2002</Birthdate>
    <SS>111-22-3333</SS>
  </Confidential>
  <Address>5224 Rose St. San Jose, CA 95123</Address>
</Person>
```

person2.xml (Birthdate 値は有効です)

```
<?xml version="1.0"?>
<Person gender="Male">
  <Name>
    <Last>Cool</Last>
    <First>Joe</First>
  </Name>
  <Confidential>
    <Age unit="years">5</Age>
    <Birthdate>2002-03-16</Birthdate>
    <SS>111-22-3333</SS>
  </Confidential>
  <Address>5224 Rose St. San Jose, CA 95123</Address>
</Person>
```

person3.xml (Birthdate 値は有効です)

```
<?xml version="1.0"?>
<Person gender="Female">
  <Name>
    <Last>McCarthy</Last>
    <First>Laura</First>
  </Name>
  <Confidential>
    <Age unit="years">6</Age>
    <Birthdate>2001-03-12</Birthdate>
    <SS>444-55-6666</SS>
  </Confidential>
  <Address>5960 Daffodil Lane, San Jose, CA 95120</Address>
</Person>
```

person4.xml (Birthdate 値は無効です)

```
<?xml version="1.0"?>
<Person gender="Female">
  <Name>
    <Last>Wong</Last>
    <First>Teresa</First>
  </Name>
  <Confidential>
    <Age unit="years">7</Age>
    <Birthdate>20000-12-09</Birthdate>
```

```
    <SS>555-66-7777</SS>
  </Confidential>
  <Address>5960 Tulip Court, San Jose, CA 95120</Address>
</Person>
```

person5.xml (Birthdate 値は有効です)

```
<?xml version="1.0"?>
<Person gender="Male">
  <Name>
    <Last>Smith</Last>
    <First>Chris</First>
  </Name>
  <Confidential>
    <Age unit="years">10</Age>
    <Birthdate>1997-04-23</Birthdate>
    <SS>666-77-8888</SS>
  </Confidential>
  <Address>5960 Dahlia Street, San Jose, CA 95120</Address>
</Person>
```

入力ファイル person.del には以下が含まれます。

```
1, <XDS FIL='person1.xml'/>
2, <XDS FIL='person2.xml'/>
3, <XDS FIL='person3.xml'/>
4, <XDS FIL='person4.xml'/>
5, <XDS FIL='person5.xml'/>
```

DDL および LOAD ステートメントは以下のとおりです。

```
CREATE TABLE T1 (docID INT, XMLDoc XML);

CREATE INDEX BirthdateIndex ON T1(xmlDoc)
   GENERATE KEY USING XMLPATTERN '/Person/Confidential/Birthdate' AS SQL DATE
   REJECT INVALID VALUES;

LOAD FROM person.del OF DEL INSERT INTO T1
```

前にリストした一連の XML ファイルのロード中に発生する索引付けエラーを解決するため、以下のステップを行います。

1. 出力情報にあるレコード番号を使用して、ロード操作中にどの行がリジェクトされたかを調べます。以下の出力では、レコード番号 1 およびレコード番号 4 はリジェクトされました。

   ```
   SQL20305N  An XML value cannot be inserted or updated because of an error
   detected when inserting or updating the index identified by "IID = 3" on table
   "LEECM.T1".  Reason code = "5".  For reason codes related to an XML schema the
   XML schema identifier = "*N" and XML schema data type = "*N".  SQLSTATE=23525

   SQL3185W  The previous error occurred while processing data from row "F0-1" of
   the input file.

   SQL20305N  An XML value cannot be inserted or updated because of an error
   detected when inserting or updating the index identified by "IID = 3" on table
   "LEECM.T1".  Reason code = "4".  For reason codes related to an XML schema the
   XML schema identifier = "*N" and XML schema data type = "*N".  SQLSTATE=23525

   SQL3185W  The previous error occurred while processing data from row "F0-4" of
   the input file.

   SQL3227W  Record token "F0-1" refers to user record number "1".

   SQL3227W  Record token "F0-4" refers to user record number "4".
   ```

```
SQL3107W  There is at least one warning message in the message file.

Number of rows read         = 5
Number of rows skipped      = 0
Number of rows loaded       = 3
Number of rows rejected     = 2
Number of rows deleted      = 0
Number of rows committed    = 5
```

2. リジェクトされた行が含まれる新規ファイル reject.del を作成します。

```
1, <XDS FIL='person1.xml'/>
4, <XDS FIL='person4.xml'/>
```

3. 元の表 T1 と同じ列を持つ新規表 T2 を作成します。新規表には索引を作成しないでください。

```
CREATE TABLE T2 LIKE T1
```

4. リジェクトされた行を新規表 T2 にロードします。

```
LOAD FROM reject.del OF DEL INSERT INTO T2;
```

5. 元の表 T1 のリジェクトされた行 1 について以下を行います。

   a. リジェクトされた行を T1 にインポートして、-20305 メッセージを受け取ります。

```
IMPORT FROM reject.del OF DEL INSERT INTO T1
SQL3109N  The utility is beginning to load data from file "reject.del".

SQL3306N  An SQL error "-20305" occurred while inserting a row into the
          table.

SQL20305N  An XML value cannot be inserted or updated because of an error
detected when inserting or updating the index identified by "IID = 3" on
table "LEECM.T1".  Reason code = "5".  For reason codes related to an XML
schema the XML schema identifier = "*N" and XML schema data type = "*N".
SQLSTATE=23525

SQL3110N  The utility has completed processing.  "1" rows were read from
the input file.
```

   b. **db2diag** ログ・ファイルを調べ、生成された XQuery ステートメントを見つけます。

```
FUNCTION: DB2 UDB, Xml Storage and Index Manager, xmlsDumpXQuery, probe:608
DATA #1 : String, 36 bytes
SQL Code: SQL20305N ; Reason Code: 5
DATA #2 : String, 265 bytes
To locate the value in the document that caused the error, create a
table with one XML column and insert the failing document in the table.
Replace the table and column name in the query below with the created
table and column name and execute the following XQuery.
DATA #3 : String, 247 bytes
xquery for $i in db2-fn:xmlcolumn(
  "LEECM.T1.XMLDOC")[/*:Person/*:Confidential/*:Birthdate="March 16, 2002"]
return
<Result>
 <ProblemDocument> {$i} </ProblemDocument>
 <ProblemValue>{$i/*:Person/*:Confidential/*:Birthdate/..} </ProblemValue>
</Result>;
```

   c. 新規表 T2 を使用するように XQuery ステートメントを変更します。

```
xquery for $i in db2-fn:xmlcolumn(
  "LEECM.T2.XMLDOC")[/*:Person/*:Confidential/*:Birthdate="March 16, 2002"]
return
```

```
<Result>
 <ProblemDocument> {$i} </ProblemDocument>
 <ProblemValue>{$i/*:Person/*:Confidential/*:Birthdate/..} </ProblemValue>
</Result>;
```

d. 表 T2 に対して XQuery ステートメントを実行し、文書内の問題値を見つけます。

```
<Result><ProblemDocument><Person gender="Male">
  <Name>
    <Last>Cool</Last>
    <First>Joe</First>
  </Name>
  <Confidential>
    <Age unit="years">5</Age>
    <Birthdate>March 16, 2002</Birthdate>
    <SS>111-22-3333</SS>
  </Confidential>
  <Address>5224 Rose St. San Jose, CA 95123</Address>
</Person></ProblemDocument><ProblemValue><Confidential>
    <Age unit="years">5</Age>
    <Birthdate>March 16, 2002</Birthdate>
    <SS>111-22-3333</SS>
  </Confidential></ProblemValue></Result>
```

e. 文書が含まれるファイル person1.xml の中の問題値を修正します。 March 16, 2002 は正しい日付形式ではないため、2002-03-16 に変更されます。

```
<?xml version="1.0"?>
<Person gender="Male">
  <Name>
    <Last>Cool</Last>
    <First>Joe</First>
  </Name>
  <Confidential>
    <Age unit="years">5</Age>
    <Birthdate>2002-03-16</Birthdate>
    <SS>111-22-3333</SS>
  </Confidential>
  <Address>5224 Rose St. San Jose, CA 95123</Address>
</Person>
```

f. ステップ a. に戻り、リジェクトされた行を再度表 T1 にインポートします。

6. (ステップ 5 の最初の繰り返し)

a. リジェクトされた行を表 T1 にインポートします。インポート・ファイルから 2 つの行が読み取られたので、最初の行は正常にインポートされます。新しいエラーが 2 番目の行で発生します。

```
IMPORT FROM reject.del OF DEL INSERT INTO T1
SQL3109N  The utility is beginning to load data from file "reject.del".

SQL3306N  An SQL error "-20305" occurred while inserting a row into the
          table.

SQL20305N  An XML value cannot be inserted or updated because of an error
detected when inserting or updating the index identified by "IID = 3" on
table "LEECM.T1".  Reason code = "4".  For reason codes related to an XML
schema the XML schema identifier = "*N" and XML schema data type = "*N".
SQLSTATE=23525

SQL3110N  The utility has completed processing.  "2" rows were read from
the input file.
```

b. **db2diag** ログ・ファイルを調べ、生成された XQuery ステートメントを見つけます。

```
FUNCTION: DB2 UDB, Xml Storage and Index Manager, xmlsDumpXQuery, probe:608
DATA #1 : String, 36 bytes
SQL Code: SQL20305N ; Reason Code: 4
DATA #2 : String, 265 bytes
To locate the value in the document that caused the error, create a
table with one XML column and insert the failing document in the table.
Replace the table and column name in the query below with the created
table and column name and execute the following XQuery.
DATA #3 : String, 244 bytes
xquery for $i in db2-fn:xmlcolumn("LEECM.T1.XMLDOC")
   [/*:Person/*:Confidential/*:Birthdate="20000-12-09"]
return
<Result>
 <ProblemDocument> {$i} </ProblemDocument>
 <ProblemValue>{$i/*:Person/*:Confidential/*:Birthdate/..} </ProblemValue>
</Result>;
```

c. 表 T2 を使用するように XQuery ステートメントを変更します。

```
xquery for $i in db2-fn:xmlcolumn("LEECM.T2.XMLDOC")
   [/*:Person/*:Confidential/*:Birthdate="20000-12-09"]
return
<Result>
 <ProblemDocument> {$i} </ProblemDocument>
 <ProblemValue>{$i/*:Person/*:Confidential/*:Birthdate/..} </ProblemValue>
</Result>;
```

d. XQuery ステートメントを実行し、文書内の問題値を見つけます。

```
<Result><ProblemDocument><Person gender="Female">
  <Name>
    <Last>Wong</Last>
    <First>Teresa</First>
  </Name>
  <Confidential>
    <Age unit="years">7</Age>
    <Birthdate>20000-12-09</Birthdate>
    <SS>555-66-7777</SS>
  </Confidential>
  <Address>5960 Tulip Court, San Jose, CA 95120</Address>
</Person></ProblemDocument><ProblemValue><Confidential>
    <Age unit="years">7</Age>
    <Birthdate>20000-12-09</Birthdate>
    <SS>555-66-7777</SS>
  </Confidential></ProblemValue></Result>
```

e. 文書が含まれるファイル person4.xml の中の問題値を修正します。値 20000-12-09 の年にゼロが余分にあるため、DB2 で許可される年 (0001 から 9999) の範囲外となっています。値が 2000-12-09 に変更されます。

```
<?xml version="1.0"?>
<Person gender="Female">
  <Name>
    <Last>Wong</Last>
    <First>Teresa</First>
  </Name>
  <Confidential>
    <Age unit="years">7</Age>
    <Birthdate>2000-12-09</Birthdate>
    <SS>555-66-7777</SS>
  </Confidential>
  <Address>5960 Tulip Court, San Jose, CA 95120</Address>
</Person>
```

f. ステップ a に戻り、リジェクトされた行を再度 T1 にインポートします。

7. (ステップ 5 の 2 回目の繰り返し)

a. リジェクトされた行を T1 にインポートします。

```
IMPORT FROM reject.del OF DEL INSERT INTO T1
SQL3109N  The utility is beginning to load data from file "reject.del".

SQL3110N  The utility has completed processing.  "2" rows were read from
the input file.

SQL3221W  ...Begin COMMIT WORK. Input Record Count = "2".

SQL3222W  ...COMMIT of any database changes was successful.

SQL3149N  "2" rows were processed from the input file.  "2" rows were
successfully inserted into the table.  "0" rows were rejected.

Number of rows read         = 2
Number of rows skipped      = 0
Number of rows inserted     = 2
Number of rows updated      = 0
Number of rows rejected     = 0
Number of rows committed    = 2
```

これで問題は解決されました。person.del のすべての行は、正常に表 T1 に挿入されます。

第 10 章 アプリケーション・プログラミング言語サポート

XML データを DB2 データベース表に保管したり、表からデータを検索したり、XML パラメーターのあるストアード・プロシージャーまたはユーザー定義関数を呼び出したりするアプリケーションを作成することができます。

以下の言語のいずれかを使用して、アプリケーションを作成することができます。
- C または C++ (組み込み SQL または CLI)
- COBOL
- Java (JDBC または SQLJ)
- C# および Visual Basic (IBM Data Server Provider for .NET)
- PHP
- Perl

アプリケーション・プログラムは、文書全体または文書の一部を XML 列から検索することができます。ただし、XML 列に保管できるのは、文書全体のみです。

ストアード・プロシージャーおよびユーザー定義関数は、入力または出力パラメーターで XML 値を受け渡すことができます。 XML データは、ストアード・プロシージャーに IN、OUT、または INOUT パラメーターとして受け渡されたとき、マテリアライズされます。Java ストアード・プロシージャーを使用している場合、XML 引数の数やサイズ、および並行して実行されている外部ストアード・プロシージャーの数に応じて、ヒープ・サイズ (**java_heap_sz** 構成パラメーター) を大きくする必要がある可能性があります。XML または XML AS CLOB パラメーターのあるストアード・プロシージャーまたはユーザー定義関数を呼び出すには、 CALL ステートメントを互換データ・タイプで実行します。

アプリケーションが XML 値を DB2 データベース・サーバーに提供するとき、データベース・サーバーは、データを XML のシリアライズされたストリング・フォーマットから、Unicode の UTF-8 エンコード方式で XML 階層フォーマットに変換します。

アプリケーションが XML 列からデータを検索するとき、 DB2 データベース・サーバーは、データを XML 階層フォーマットから XML のシリアライズされたストリング・フォーマットに変換します。さらに、データベース・サーバーは、出力データを UTF-8 からアプリケーション・エンコード方式に変換する必要がある場合もあります。

XML データを検索するときには、コード・ページ変換のデータ損失への影響を認識しておく必要があります。データ損失は、ソース・コード・ページの文字をターゲット・コード・ページで表せない場合に生じることがあります。

アプリケーションは、XML 列から XML 文書全体またはシーケンスを検索することができます。

XML 文書全体をフェッチするときには、文書をアプリケーション変数に取り出します。

XML シーケンスを検索する場合、以下のようないくつかの選択項目があります。

- XQuery 式を直接実行する。

  アプリケーションで XQquery 式を実行するには、ストリング「XQUERY」を XQuery 式の前に付加し、結果ストリングを動的に実行します。

  XQuery 式を直接実行するとき、DB2 データベース・サーバーは、XQuery ステートメントの結果であるシーケンスを結果表として戻します。結果表の各行は、シーケンス内の項目です。

- SQL SELECT または単一行 SELECT INTO 操作内で、引数として XQuery 式を渡して、 XMLQUERY または XMLTABLE 組み込み関数を呼び出す。

  この技法は、静的または動的 SQL、および任意のアプリケーション・プログラミング言語で使用できます。 XMLQUERY は、アプリケーション変数にシーケンス全体を戻すスカラー関数です。 XMLTABLE は、シーケンスの各項目を結果表の行として戻す表関数です。結果表の列は、検索されたシーケンス項目の値です。

パラメーター・マーカーおよびホスト変数

パラメーター・マーカーまたはホスト変数は、XQuery 式で指定された SQL 内を含め、XQuery 式内にはどこにも指定できません。例えば、XQuery 関数 db2-fn:sqlquery により、SQL 全選択を XQuery 式と共に指定して、以下のように製品の詳細記述を取り出すことができます。

```
xquery
db2-fn:sqlquery("select description from product where pid='100-103-01'")
                /product/description/details/text()
```

パラメーター・マーカーまたはホスト変数は、SQL 全選択内であっても、XQuery 式内に指定することはできません。以下の式は誤っており、サポートされていません (これは SQLSTATE 42610、sqlcode -418 を戻します)。

```
xquery
db2-fn:sqlquery("select description from product where pid=?")
                /product/description/details/text()
```

アプリケーション値を XQuery 式に渡すためには、SQL/XML 関数の XMLQUERY および XMLTABLE を使用します。これらの関数の PASSING 節により、XQuery 式の評価時にアプリケーション値を使用することができます。

以下の照会は、前の誤った照会を SQL/XML を使用して作成し直し、同等の結果を達成する方法を示しています。

```
SELECT XMLQUERY ('$descdoc/product/description/details/text()'
   passing description as "descdoc")
FROM product
WHERE pid=?
```

CLI

CLI アプリケーションでの XML データの取り扱い - 概要

CLI アプリケーションは、SQL_XML データ・タイプを使用して XML データを検索および保管できます。このデータ・タイプは、整形式 XML 文書を保管する列を

定義するために使用する、 DB2 データベースのネイティブ XML データ・タイプに相当します SQL_XML タイプは、SQL_C_BINARY、 SQL_C_CHAR、SQL_C_WCHAR、および SQL_C_DBCHAR の各 C タイプにバインドできます。文字タイプの代わりにデフォルトの SQL_C_BINARY タイプを使用して、文字タイプを使用したときのコード・ページ変換から生じるデータ損失または破壊の可能性を回避してください。

XML データを XML 列に保管するには、SQL_XML SQL タイプへの XML 値を含むバイナリー (SQL_C_BINARY) または文字 (SQL_C_CHAR、SQL_C_WCHAR、または SQL_C_DBCHAR) バッファーを SQL_XML SQL タイプにバインドして、INSERT または UPDATE SQL ステートメントを実行します。 XML データをデータベースから検索するには、結果セットをバイナリー (SQL_C_BINARY) または文字 (SQL_C_CHAR、SQL_C_WCHAR、または SQL_C_DBCHAR) タイプにバインドします。エンコードの問題があるため、文字タイプは注意して使用してください。

XML 値が取得されてアプリケーション・データ・バッファーに入れられるとき、DB2 サーバーは XML 値に対する暗黙的なシリアライゼーションを実行して、それを保管されている階層フォームからシリアライズされたストリング・フォームに変換します。文字タイプのバッファーでは、XML 値は文字タイプに関連したアプリケーション文字コード・ページに対して暗黙的にシリアライズされます。

デフォルトでは、XML 宣言はシリアライズされた出力ストリングに含まれています。このデフォルトの動作は、SQL_ATTR_XML_DECLARATION ステートメントまたは接続属性を設定することにより、または XMLDeclaration CLI/ODBC 構成キーワードを `db2cli.ini` ファイル内に設定することにより、変更できます。

CLI アプリケーションで XQuery 式および SQL/XML 関数を発行して実行できます。 SQL/XML 関数は、他の SQL ステートメントと同様に発行および実行できます。大文字小文字の区別のないキーワード **`XQUERY`** を XQuery 式に接頭部として追加するか、XQuery 式に関連付けられたステートメント・ハンドルの SQL_ATTR_XQUERY_STATEMENT ステートメント属性を設定する必要があります。

注: DB2 バージョン 9.7 フィックスパック 5 以降、SQL_XML データ・タイプは i V7R1 サーバー以降のリリース用の DB2 でサポートされています。

CLI アプリケーションでの XML 列の挿入および更新

表の XML 列でデータを更新または挿入するとき、入力データはシリアライズされたストリング・フォーマットでなければなりません。

XML データでは、`SQLBindParameter()` を使用してパラメーター・マーカーを入力データ・バッファーにバインドするとき、入力データ・バッファーのデータ・タイプを SQL_C_BINARY、SQL_C_CHAR、SQL_C_DBCHAR、または SQL_C_WCHAR として指定できます。

SQL_C_BINARY タイプの XML データを含むデータ・バッファーをバインドするとき、 CLI はその XML データを内部エンコード・データとして処理します。これは文字タイプを使用する場合に、リソースの追加使用と文字変換の際のデータ損失の可能性を回避できるため、より推奨される方法です。

重要: XML データがアプリケーション・コード・ページのコード化スキームではないコード化スキームおよび CCSID でエンコードされた場合、内部エンコードをデータに含めて、そのデータを SQL_C_BINARY としてバインドすることにより文字変換を防止する必要があります。

SQL_C_CHAR、SQL_C_DBCHAR、または SQL_C_WCHAR タイプの XML データを含むデータ・バッファーをバインドするとき、 CLI はその XML データを外部エンコード・データとして処理します。 CLI は、データのエンコード方式を次のように決定します。

- C タイプが SQL_C_WCHAR の場合、CLI はデータが UCS-2 としてエンコードされていると想定します。
- C タイプが SQL_C_CHAR または SQL_C_DBCHAR の場合、 CLI はデータがアプリケーション・コード・ページのコード化スキームでエンコードされていると想定します。

データベース・サーバーがデータを XML 列に保管する前にそれを暗黙的に構文解析するようにするには、 SQLBindParameter() 内のパラメーター・マーカーのデータ・タイプを SQL_XML として指定する必要があります。

XMLPARSE を使用して文字タイプを明示的に構文解析すると、エンコードの問題が生じることがあるので、暗黙的な構文解析が推奨されています。

次の例は、推奨される SQL_C_BINARY タイプを使用して、XML 列内の XML データを更新する方法を示しています。

```
char xmlBuffer[10240];
integer length;

// Assume a table named dept has been created with the following statement:
// CREATE TABLE dept (id CHAR(8), deptdoc XML)

// xmlBuffer contains an internally encoded XML document that is to replace
// the existing XML document
length = strlen (xmlBuffer);
SQLPrepare (hStmt, "UPDATE dept SET deptdoc = ? WHERE id = '001'", SQL_NTS);
SQLBindParameter (hStmt, 1, SQL_PARAM_INPUT, SQL_C_BINARY, SQL_XML, 0, 0,
                  xmlBuffer, 10240, &length);
SQLExecute (hStmt);
```

CLI アプリケーション内での XML データ検索

表の XML 列からデータを選択するとき、出力データはシリアライズされたストリング・フォーマットとなります。

XML データでは、SQLBindCol() を使用して照会結果セット内の列をアプリケーション変数にバインドするとき、アプリケーション変数のデータ・タイプを SQL_C_BINARY、SQL_C_CHAR、SQL_C_DBCHAR、または SQL_C_WCHAR として指定できます。 XML 列から結果セットを検索するとき、アプリケーション変数を SQL_C_BINARY タイプにバインドすることをお勧めします。文字タイプにバインドすると、コード・ページ変換によりデータ損失が生じる可能性があります。データ損失は、ソース・コード・ページの文字をターゲット・コード・ページで表せない場合に生じることがあります。変数を SQL_C_BINARY C タイプにバインドすることにより、これらの問題を回避できます。

XML データは、内部エンコード・データとしてアプリケーションに戻されます。CLI は、データのエンコード方式を次のように決定します。

- C タイプが SQL_C_BINARY の場合、CLI はデータを UTF-8 コード化スキームで戻します。
- C タイプが SQL_C_CHAR または SQL_C_DBCHAR の場合、 CLI はデータをアプリケーション・コード・ページのコード化スキームで戻します。
- C タイプが SQL_C_WCHAR の場合、CLI はデータを UCS-2 コード化スキームで戻します。

データベース・サーバーは、データをアプリケーションに戻す前に、そのデータに対する暗黙的なシリアライゼーションを実行します。 XMLSERIALIZE 関数を呼び出すことにより、XML データを特定のデータ・タイプに明示的にシリアライズすることができます。ただし、XMLSERIALIZE によって文字タイプに明示的にシリアライズするとエンコードの問題が生じることがあるので、暗黙的なシリアライゼーションが推奨されています。

次の例は、XML データを XML 列から検索して、バイナリー・アプリケーション変数にする方法を示しています。

```
char xmlBuffer[10240];
// xmlBuffer is used to hold the retrieved XML document
integer length;

// Assume a table named dept has been created with the following statement:
// CREATE TABLE dept (id CHAR(8), deptdoc XML)

length = sizeof (xmlBuffer);
SQLExecute (hStmt, "SELECT deptdoc FROM dept WHERE id='001'", SQL_NTS);
SQLBindCol (hStmt, 1, SQL_C_BINARY, xmlBuffer, &length, NULL);
SQLFetch (hStmt);
SQLCloseCursor (hStmt);
// xmlBuffer now contains a valid XML document encoded in UTF-8
```

CLI アプリケーションでのデフォルトの XML タイプ処理の変更

CLI は、XML 列およびパラメーター・マーカーを記述するか、またはそこに SQL_C_DEFAULT を指定するとき、デフォルト・タイプが戻されることを予期しないアプリケーションのために互換性を提供する CLI/ODBC 構成キーワードをサポートします。それ以前の CLI および ODBC アプリケーションは、 XML 列またはパラメーターを記述するときに、デフォルトの SQL_XML タイプを認識または予期しないことがあります。いくつかの CLI または ODBC アプリケーションは、 XML 列およびパラメーター・マーカーに対して SQL_C_BINARY 以外のデフォルト・タイプを期待することもあります。これらのタイプのアプリケーションに互換性を提供するために、 CLI は MapXMLDescribe および MapXMLCDefault キーワードをサポートしています。

MapXMLDescribe は、XML 列またはパラメーター・マーカーが記述されている場合にどの SQL データ・タイプが戻されるかを制御します。

MapXMLCDefault は、CLI 関数で XML 列およびパラメーター・マーカーに対して SQL_C_DEFAULT が指定されている場合に使用される C タイプを指定します。

組み込み SQL

組み込み SQL アプリケーションにおける XML ホスト変数の宣言

データベース・サーバーと組み込み SQL アプリケーションの間で XML データを交換するには、アプリケーションのソース・コードの中でホスト変数を宣言する必要があります。

このタスクについて

DB2 V9.1 では、XML データをツリー形式でノードの構造化セットに保管する XML データ・タイプが導入されています。この XML データ・タイプを持つ列は SQL_TYP_XML 列 SQLTYPE として記述され、アプリケーションは、これらの列またはパラメーターとの間の入力あるいは出力のために、さまざまな言語固有のデータ・タイプをバインドできます。 XML 列には、SQL、SQL/XML 拡張、または XQuery を使用して直接アクセスすることができます。 XML データ・タイプは、単に列に適用されるだけではありません。関数が XML 値の引数を取ったり、XML 値を生成したりすることもできます。同様に、ストアード・プロシージャーは、入力パラメーターおよび出力パラメーターの両方として XML 値を取ることができます。さらに、XQuery 式は、XML 列にアクセスするかどうかに関係なく、XML 値を生成します。

XML データは本来、文字であり、使用される文字セットを定義するエンコード方式を持っています。 XML データのエンコード方式は、 XML 文書をシリアライズされたストリングで表示したものを含む基本アプリケーション・タイプから導出して、外部的に決めることができます。さらに、内部的に決めることもでき、その場合にはデータの解釈が必要になります。 Unicode でエンコードされた文書には、データ・ストリームの先頭の Unicode 文字コードで構成されるバイト・オーダー・マーク (BOM) が推奨されます。 BOM は、バイト・オーダーおよび Unicode エンコード・フォームを定義するシグニチャーとして使用されます。

データの取り出しおよび挿入には、XML ホスト変数に加え、CHAR、VARCHAR、CLOB、BLOB といった、既存の文字およびバイナリー・タイプも使用できます。しかし、こうしたタイプは、XML ホスト変数とは違い、暗黙的な XML 構文解析の対象になることはありません。その代わり、デフォルトで空白文字の除去を行う明示的な XMLPARSE 関数が挿入され、適用されます。

組み込み SQL アプリケーションの開発における XML および XQuery に関する制約事項

組み込み SQL アプリケーションで XML ホスト変数を宣言するには以下のようにします。

アプリケーションの宣言セクションで、以下のように、XML ホスト変数を LOB データ・タイプとして宣言します。

-
```
SQL TYPE IS XML AS CLOB(n) <hostvar_name>
```

  ここで、<hostvar_name> は、アプリケーションの混合コード・ページでエンコードされる XML データを含む CLOB ホスト変数です。

- `SQL TYPE IS XML AS DBCLOB(n) <hostvar_name>`

  ここで、<hostvar_name> は、アプリケーションのグラフィック・コード・ページでエンコードされる XML データを含む DBCLOB ホスト変数です。

- `SQL TYPE IS XML AS BLOB(n) <hostvar_name>`

  ここで、<hostvar_name> は、内部でエンコードされる XML データを含む BLOB ホスト変数です。[1]

- `SQL TYPE IS XML AS CLOB_FILE <hostvar_name>`

  ここで、<hostvar_name> は、アプリケーションの混合コード・ページでエンコードされる XML データを含む CLOB ファイルです。

- `SQL TYPE IS XML AS DBCLOB_FILE <hostvar_name>`

  ここで、<hostvar_name> は、アプリケーションのグラフィック・コード・ページでエンコードされる XML データを含む DBCLOB ファイルです。

- `SQL TYPE IS XML AS BLOB_FILE <hostvar_name>`

  ここで、<hostvar_name> は、内部でエンコードされる XML データを含む BLOB ファイルです。[1]

注:

1. XML 1.0 仕様によってエンコード方式を決めるためのアルゴリズム (`http://www.w3.org/TR/REC-xml/#sec-guessing-no-ext-info`) を参照してください。

例: 組み込み SQL アプリケーションでの XML ホスト変数の参照

以下のサンプル・アプリケーションは、C および COBOL で XML ホスト変数を参照する方法を示しています。

例: 組み込み SQL C アプリケーション

```
The following code example has been formatted for clarity:
EXEC SQL BEGIN DECLARE;
  SQL TYPE IS XML AS CLOB( 10K ) xmlBuf;
  SQL TYPE IS XML AS BLOB( 10K ) xmlblob;
  SQL TYPE IS CLOB( 10K ) clobBuf;
EXEC SQL END DECLARE SECTION;

// as XML AS CLOB
// The XML value written to xmlBuf will be prefixed by an XML declaration
// similar to: <?xml version = "1.0" encoding = "ISO-8859-1" ?>
// Note: The encoding name will depend upon the application codepage
EXEC SQL SELECT xmlCol INTO :xmlBuf
   FROM myTable
   WHERE id = '001';
EXEC SQL UPDATE myTable
   SET xmlCol = :xmlBuf
```

```
   WHERE id = '001';

// as XML AS BLOB
// The XML value written to xmlblob will be prefixed by an XML declaration
// similar to: <?xml version = "1.0" encoding = "UTF-8"?>
EXEC SQL SELECT xmlCol INTO :xmlblob
   FROM myTable
   WHERE id = '001';
EXEC SQL UPDATE myTable
   SET xmlCol = :xmlblob
   WHERE id = '001';

// as CLOB
// The output will be encoded in the application character codepage,
// but will not contain an XML declaration
EXEC SQL SELECT XMLSERIALIZE (xmlCol AS CLOB(10K)) INTO :clobBuf
   FROM myTable
   WHERE id = '001';
EXEC SQL UPDATE myTable
   SET xmlCol = XMLPARSE (:clobBuf PRESERVE WHITESPACE)
   WHERE id = '001';
```

例: 組み込み SQL COBOL アプリケーション

```
The following code example has been formatted for clarity:
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
  01 xmlBuf USAGE IS SQL TYPE IS XML as CLOB(5K).
  01 clobBuf USAGE IS SQL TYPE IS CLOB(5K).
  01 xmlblob  USAGE IS SQL TYPE IS BLOB(5K).
EXEC SQL END DECLARE SECTION END-EXEC.

* as XML
EXEC SQL SELECT xmlCol INTO :xmlBuf
   FROM myTable
   WHERE id = '001' END-EXEC.
EXEC SQL UPDATE myTable
   SET xmlCol = :xmlBuf
   WHERE id = '001' END-EXEC.

* as BLOB
EXEC SQL SELECT xmlCol INTO :xmlblob
   FROM myTable
   WHERE id = '001' END-EXEC.
EXEC SQL UPDATE myTable
   SET xmlCol = :xmlblob
   WHERE id = '001' END-EXEC.

* as CLOB
EXEC SQL SELECT XMLSERIALIZE(xmlCol AS CLOB(10K)) INTO :clobBuf
   FROM myTable
   WHERE id= '001' END-EXEC.
EXEC SQL UPDATE myTable
   SET xmlCol = XMLPARSE(:clobBuf) PRESERVE WHITESPACE
   WHERE id = '001' END-EXEC.
```

組み込み SQL アプリケーションにおける XQuery 式の実行

始める前に

表に XML データを保管し、XQuery 式を使用して、組み込み SQL アプリケーションで XML 列にアクセスすることができます。 XML データへのアクセスには、データを文字またはバイナリー・データ・タイプにキャストする代わりに、 XML ホスト変数を使用します。 XML ホスト変数を使用しない場合、XML データにア

クセスするのに最適な代替方法は、コード・ページ変換を避けるために FOR BIT DATA または BLOB データ・タイプを使用することです。

- 組み込み SQL アプリケーション内で XML ホスト変数を宣言する。

このタスクについて

- 静的 SQL SELECT INTO ステートメントで XML 値を検索するには、XML タイプの使用が必要。
- XML 値が予期されているところで CHAR、VARCHAR、CLOB あるいは BLOB ホスト変数を入力に使用すると、その値は、デフォルトで空白処理 (STRIP) を行う XMLPARSE 関数操作の対象になります。そうでない場合、XML ホスト変数は必須です。

XQuery 式を組み込み SQL アプリケーション内で直接実行するには、「XQUERY」キーワードを式の前に付加します。静的 SQL の場合は、XMLQUERY 関数を使用します。 XMLQUERY 関数が呼び出される場合、XQuery 式に「XQUERY」という接頭部は付きません。

これらの例は、サンプル・データベースの表 CUSTOMER に含まれる XML 文書からのデータを戻します。

例 1: C および C++ 動的 SQL で、「XQUERY」キーワードを前に付加して直接 XQuery 式を実行する

C および C++ アプリケーションでは、XQuery 式は、以下のような方法で実行できます。

```
EXEC SQL INCLUDE SQLCA;
EXEC SQL BEGIN DECLARE SECTION;
  char stmt[16384];
  SQL TYPE IS XML AS BLOB( 10K ) xmlblob;
EXEC SQL END DECLARE SECTION;

sprintf( stmt, "XQUERY (for $a in db2-fn:xmlcolumn("CUSTOMER.INFO")
        /*:customerinfo[*:addr/*:city = "Toronto"]/@Cid return data($a))");

EXEC SQL PREPARE s1 FROM :stmt;
EXEC SQL DECLARE c1 CURSOR FOR s1;
EXEC SQL OPEN c1;

while( sqlca.sqlcode == SQL_RC_OK )
{
  EXEC SQL FETCH c1 INTO :xmlblob;
  /* Display results */
}

EXEC SQL CLOSE c1;
EXEC SQL COMMIT;
```

例 2: XMLQUERY 関数と XMLEXISTS 述部を使用して、静的 SQL で XQuery 式を実行する

XMLQUERY 関数を含む SQL ステートメントは、以下のように静的に準備できます。

```
EXEC SQL INCLUDE SQLCA;
EXEC SQL BEGIN DECLARE SECTION;
  SQL TYPE IS XML AS BLOB( 10K ) xmlblob;
EXEC SQL END DECLARE SECTION;

EXEC SQL DECLARE C1 CURSOR FOR SELECT XMLQUERY(data($INFO/*:customerinfo/@Cid)')
FROM customer
WHERE XMLEXISTS('$INFO/*:customerinfo[*:addr/*:city = "Toronto"]');

EXEC SQL OPEN c1;

while( sqlca.sqlcode == SQL_RC_OK )
{
  EXEC SQL FETCH c1 INTO :xmlblob;
  /* Display results */
```

```
}

EXEC SQL CLOSE c1;
EXEC SQL COMMIT;
```

例 3: COBOL 組み込み SQL アプリケーションで XQuery 式を実行する

COBOL アプリケーションでは、XQuery 式は、以下のような方法で実行できます。

```
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
     01 stmt pic x(80).
   01 xmlBuff USAGE IS SQL TYPE IS XML AS BLOB (10K).
EXEC SQL END DECLARE SECTION END-EXEC.

MOVE "XQUERY (for $a in db2-fn:xmlcolumn("CUSTOMER.INFO")/*:customerinfo
           [*:addr/*:city = "Toronto"]/@Cid return data($a)))" TO stmt.
EXEC SQL PREPARE s1 FROM :stmt END-EXEC.
EXEC SQL DECLARE c1 CURSOR FOR s1 END-EXEC.
EXEC SQL OPEN c1 USING :host-var END-EXEC.

*Call the FETCH and UPDATE loop.
Perform Fetch-Loop through End-Fetch-Loop
   until SQLCODE does not equal 0.

EXEC SQL CLOSE c1 END-EXEC.
EXEC SQL COMMIT END-EXEC.

Fetch-Loop Section.
    EXEC SQL FETCH c1 INTO :xmlBuff END-EXEC.
    if SQLCODE not equal 0
       go to End-Fetch-Loop.
* Display results
End-Fetch-Loop. exit.
```

XML および XQuery を使用した組み込み SQL アプリケーションの開発に関する推奨事項

組み込み SQL アプリケーションで XML および XQuery を使用するにあたっては、以下の推奨事項および制約事項が適用されます。

- アプリケーションは、直列化されたストリング・フォーマットですべての XML データにアクセスする必要がある。
  - 数値、日付時間データなどを含め、すべてのデータを、直列化されたストリング・フォーマットで表現する必要がある。
- 外部化される XML データは 2 GB に制限される。
- XML データを含むカーソルはすべて非ブロッキングになる (取り出し操作のたびにデータベース・サーバー要求が出される)。
- 文字ホスト変数に直列化された XML データが含まれる場合は常に、アプリケーション・コード・ページがデータのエンコード方式として使用されるとみなされるため、それが、データ中に存在する内部エンコード方式と一致しなくてはならない。
- XML ホスト変数の基本タイプとしては、LOB データ・タイプを指定しなくてはならない。
- 静的 SQL には、以下の推奨事項および制約事項が適用されます。
  - 文字およびバイナリー・ホスト変数は、SELECT INTO 操作による XML 値の取得には使用できない。
  - XML データ・タイプの入力が予期されているところで CHAR、VARCHAR、CLOB、BLOB ホスト変数を使用すると、それらは、デフォルトで空白処理を行う特性 ('STRIP WHITESPACE') を持つ XMLPARSE 操作の対象になります。他の非 XML ホスト変数タイプはすべて拒否されます。

– 静的 XQuery 式のサポートはない。XQuery 式をプリコンパイルしようとすると、エラーが発生して失敗します。XQuery 式は、XMLQUERY 関数によってのみ実行できます。

- XQuery 式は、ストリング「XQUERY」を式の前に付けることで、動的に実行できる。

SQLDA での XML 値の識別

基本タイプが XML データを保持していることを示すには、SQLVAR の sqlname フィールドは以下のように更新する必要があります。

- sqlname.length は 8 でなければならない。
- sqlname.data の最初の 2 バイトは X'0000' でなければならない。
- sqlname.data の 3 番目、4 番目のバイトは X'0000' にするべきである。
- sqlname.data の 5 番目のバイトは X'01' でなければならない (最初の 2 つの条件が満たされた場合にのみ、XML サブタイプ標識として参照される)。
- 残りのバイトは X'000000' にするべきである。

XML サブタイプ標識が、SQLTYPE が非 LOB である SQLVAR で設定された場合、実行時に SQL0804 エラー (rc=115) が戻されます。

注: SQL_TYP_XML は DESCRIBE ステートメントからのみ戻せます。このタイプは、他のどの要求に対しても使用することはできません。アプリケーションは、有効な文字タイプまたはバイナリー・タイプを収容するように SQLDA を変更し、sqlname フィールドを、データが XML であることを示すよう適切に設定する必要があります。

Java

Java アプリケーションにおけるバイナリー XML 形式

バージョン 4.9 以降のリリースでは、データ・サーバーがバイナリー XML データをサポートする場合、 IBM Data Server Driver for JDBC and SQLJ は、 XML データをバイナリー XML データ (拡張可能動的バイナリー XML DB2 クライアント/サーバー・バイナリー XML 形式のデータ) としてデータ・サーバーに送信したりデータ・サーバーから取得したりすることができます。

IBM Data Server Driver for JDBC and SQLJ は、テキスト XML 形式またはバイナリー XML 形式のどちらでデータ転送を行うか制御するために、 DriverManager および DataSource の各インターフェースに xmlFormat プロパティーを備えています。 xmlFormat に XML_FORMAT_BINARY を設定するとバイナリー XML 形式が有効になります。データ・サーバーがバイナリー XML 形式をサポートする場合は、XML_FORMAT_BINARY がデフォルトの形式です。そうでない場合は、XML_FORMAT_TEXTUAL がデフォルトです。 XML データの形式について、アプリケーションが関与する必要はありません。バイナリー XML データの保管および取得には、 IBM Data Server Driver for JDBC and SQLJ バージョン 4.9 以降が必要です。SQLJ アプリケーションでバイナリー XML データを使用している場合、sqlj4.zip パッケージもバージョン 4.9 以降が必要になります。

以下の例は、DataSource インターフェースを使用してデータ伝送をバイナリー XML 形式に設定するステートメントを示しています。

```
import com.ibm.db2.jcc.DB2SimpleDataSource;
...
DB2SimpleDataSource ds = new DB2SimpleDataSource();
ds.setXmlFormat(DB2BaseDataSource.XML_FORMAT_BINARY);
```

データ伝送をテキスト XML 形式に設定するには、以下のようなステートメントを使用します。

```
ds.setXmlFormat(DB2BaseDataSource.XML_FORMAT_TEXTUAL);
```

Connection インターフェースには xmlFormat プロパティーのために定義された setXXX メソッドはありません。このため、Connection インターフェースを使用して xmlFormat 値を設定するためには、以下の例に示すように、DriverManager.getConnection メソッドを実行する際に xmlFormat をプロパティーとして指定する必要があります。

```
properties.put("xmlFormat", DB2BaseDataSource.XML_FORMAT_BINARY);
DriverManager.getConnection(url, properties);
```

IBM Data Server Driver for JDBC and SQLJ は、 XML オブジェクト・インターフェースを通じてのみ、アプリケーションに対してバイナリー XML データを提供します。ユーザーは、バイナリー XML 形式のデータを見ることはありません。

バイナリー XML データを使用する場合、IBM Data Server Driver for JDBC and SQLJ に渡される XML データからは、外部エンティティー、内部エンティティー、または内部 DTD のいずれも参照できません。外部 DTD は、過去にデータ・ソースに登録されていた場合にのみサポートされます。

バイナリー XML 形式は、入力データあるいは出力データが SAX、StAX、DOM などの非テキスト表記である場合に最も効率的です。例えば、以下のメソッドは XML データを非テキスト表記で取り出します。

- getSource(SAXSource.class)
- getSource(StAXSource.class)
- getSource(DOMSource.class)

以下のメソッドは XML 列を非テキスト表記のデータで更新します。

- setResult(SAXResult.class)
- setResult(StAXResult.class)
- setResult(DOMResult.class)

SAX 表記は、バイナリー形式からテキスト形式へのデータの余分な変換を行わないため、バイナリー XML 形式のデータを取り出すには最も効率的な方法です。

xmlFormat を XML_FORMAT_BINARY (1) に設定するとします。以下の JDBC の例では、IBM Data Server Driver for JDBC and SQLJ はデータをバイナリー XML 形式で取得し、アプリケーションは取得されたデータを解析するのに SAX パーサーを使用しています。

```
...
Statement stmt = conn.createStatement();
ResultSet rs = stmt.executeQuery("SELECT XMLCOL FROM XMLTABLE");
ContentHandler handler = new MyContentHandler();
while (rs.next()) {
```

```
 SQLXML sqlxml = rs.getSQLXML(1);
 SAXSource source = sqlxml.getSource(SAXSource.class);
 XMLReader reader = source.getXMLReader();
 reader.setContentHandler(handler);
 reader.parse(source.getInputSource());
}
...
```

次の SQLJ の例も同じ動作を実行します。

```
#sql iterator  SqlXmlIter(java.sql.SQLXML);
{
 ...
 SqlXmlIter SQLXMLiter = null;
 java.sql.SQLXML outSqlXml = null;
 ContentHandler handler = new MyContentHandler();
 #sql [ctx] SQLXmlIter = {SELECT XMLCOL FROM XMLTABLE};
 #sql {FETCH :SqlXmlIter INTO :outSqlXml};
 while (!SQLXMLIter.endFetch()) {
  SAXSource source = outSqlXml.getSource(SAXSource.class);
  XMLReader reader = source.getXMLReader();
  reader.setContentHandler(handler);
  reader.parse(source.getInputSource());
  #sql {FETCH :SqlXmlIter INTO :outSqlXml};
 }
 ...
}
```

JDBC

JDBC アプリケーションでの XML データ

JDBC アプリケーションでは、XML 列へのデータの保管、および XML 列からのデータを取り出しが可能です。

データベース表では、XML 組み込みデータ・タイプを使用して XML データをノードの構造化セットとしてツリー形式で列に保管します。

JDBC アプリケーションは、以下のいずれかの形式で、データ・サーバーに XML データを送ったり、データ・サーバーから XML データを取り出したりすることができます。

- テキスト XML データ
- バイナリー XML データ (データ・サーバーがサポートする場合)

JDBC アプリケーションでは以下を行うことができます。

- setXXX メソッドを使用して XML 文書全体を XML 列に保管します。
- getXXX メソッドを使用して XML 文書全体を XML 列から取り出します。
- SQL XMLQUERY 関数を使用してシーケンスをデータベース内のシリアライズされたシーケンスに取り出してから、getXXX メソッドを使用してデータをアプリケーション変数に取り出すことによって、XML 列の文書からシーケンスを取り出します。
- ストリング 'XQUERY' が前に付加されている XQuery 式を使用して、シーケンスの要素をデータベース内の結果表に取り出すことによって、XML 列の文書からシーケンスを取り出します。結果表の各行はシーケンス内のアイテムを表します。続いて、getXXX メソッドを使用してデータをアプリケーション変数に取り出します。

- SQL XMLTABLE 関数を使用して結果表を定義し、その表を取り出すことによって、XML 列の文書からユーザー定義表としてシーケンスを取り出します。続いて、getXXX メソッドを使用して、結果表からのデータをアプリケーション変数に取り出します。

JDBC 4.0 の java.sql.SQLXML オブジェクトを使用して、XML 列内のデータの取り出しおよび更新を行うことができます。 ResultSetMetaData.getColumnTypeName などのメタデータ・メソッドを呼び出すと、XML 列タイプの整数値 java.sql.Types.SQLXML が戻されます。

JDBC アプリケーションでの XML 列の更新

JDBC アプリケーションでは、 DB2 データ・サーバーの表の XML 列に、XML テキスト・データを使用してデータを更新したり挿入したりできます。データ・サーバーがバイナリー XML データをサポートしている場合、表の XML 列のデータを更新したり挿入したりするのに、バイナリー XML データ (拡張可能動的バイナリー XML DB2 クライアント/サーバー・バイナリー XML 形式のデータ) を使用できます。

次の表には、データの XML 列への書き込みに使用できるメソッドと対応する入力データ・タイプがリストされています。

表 28. XML 列の更新用のメソッドとデータ・タイプ

メソッド	入力データ・タイプ
PreparedStatement.setAsciiStream	InputStream
PreparedStatement.setBinaryStream	InputStream
PreparedStatement.setBlob	Blob
PreparedStatement.setBytes	byte[]
PreparedStatement.setCharacterStream	Reader
PreparedStatement.setClob	Clob
PreparedStatement.setObject	byte[]、Blob、Clob、SQLXML、DB2Xml (非推奨)、InputStream、Reader、String
PreparedStatement.setSQLXML[1]	SQLXML
PreparedStatement.setString	String

注:

1. このメソッドには、JDBC 4.0 以降が必要です。

XML データのエンコードは、データ自体から (*内部的にエンコードされた* と呼びます) か、または外部ソースから (*外部的にエンコードされた* と呼びます) 導出されます。データベース・サーバーにバイナリー・データとして送信される XML データは、内部的にエンコードされたデータとして処理されます。データ・ソースに文字データとして送信される XML データは、外部的にエンコードされたデータとして処理されます。

Java アプリケーションの外部エンコード方式は、常に Unicode エンコード方式です。

外部的にエンコードされたデータに、内部エンコード方式を含めることができます。つまり、このデータはデータ・ソースに文字データとして送信されますが、こ

のデータにエンコード方式の情報を含めることができます。データ・ソースにより、以下のように内部エンコード方式と外部エンコード方式の間の非互換性が処理されます。

- データ・ソースが DB2 Database for Linux, UNIX, and Windows の場合、外部エンコード方式と内部エンコード方式の間に互換性がなく、外部エンコード方式と内部エンコード方式が Unicode でない場合は、データベース・ソースによりエラーが生成されます。外部エンコード方式と内部エンコード方式が Unicode の場合は、データベース・ソースでは内部エンコード方式が無視されます。
- データベース・ソースが DB2 for z/OS® の場合、データベース・ソースでは内部エンコード方式が無視されます。

XML 列のデータは UTF-8 エンコード方式で保管されます。データベース・ソースにより、内部エンコード方式または外部エンコード方式から UTF-8 へのデータの変換が処理されます。

例: 以下の例は、データを SQLXML オブジェクトから XML 列へ挿入する方法を示しています。データがストリング・データであるため、データベース・ソースにより外部的にエンコードされるデータとして処理されます。

```
public void insertSQLXML()
{
 Connection con = DriverManager.getConnection(url);
  SQLXML info = con.createSQLXML();
                            // Create an SQLXML object
  PreparedStatement insertStmt = null;
  String infoData =
    "<customerinfo xmlns=""http://posample.org"" " +
    "Cid=""1000"">...</customerinfo>";
  info.setString(infoData);
                            // Populate the SQLXML object
  int cid = 1000;
  try {
    sqls = "INSERT INTO CUSTOMER (CID, INFO) VALUES (?, ?)";
    insertStmt = con.prepareStatement(sqls);
    insertStmt.setInt(1, cid);
    insertStmt.setSQLXML(2, info);
                            // Assign the SQLXML object value
               // to an input parameter
    if (insertStmt.executeUpdate() != 1) {
      System.out.println("insertSQLXML: No record inserted.");
    }
  }
  catch (IOException ioe) {
   ioe.printStackTrace();
  }
  catch (SQLException sqle) {
    System.out.println("insertSQLXML: SQL Exception: " +
      sqle.getMessage());
    System.out.println("insertSQLXML: SQL State: " +
      sqle.getSQLState());
    System.out.println("insertSQLXML: SQL Error Code: " +
      sqle.getErrorCode());

  }
}
```

例: 以下の例は、データをファイルから XML 列へ挿入する方法を示しています。データはバイナリー・データとして挿入されるため、データベース・サーバーにより内部エンコード方式が使用されます。

```
public void insertBinStream(Connection conn)
{
  PreparedStatement insertStmt = null;
  String sqls = null;
  int cid = 0;
  Statement stmt=null;
  try {
    sqls = "INSERT INTO CUSTOMER (CID, INFO) VALUES (?, ?)";
    insertStmt = conn.prepareStatement(sqls);
    insertStmt.setInt(1, cid);
    File file = new File(fn);
    insertStmt.setBinaryStream(2,
      new FileInputStream(file), (int)file.length());
    if (insertStmt.executeUpdate() != 1) {
      System.out.println("insertBinStream: No record inserted.");
    }
  }
  catch (IOException ioe) {
   ioe.printStackTrace();
  }
  catch (SQLException sqle) {
    System.out.println("insertBinStream: SQL Exception: " +
      sqle.getMessage());
    System.out.println("insertBinStream: SQL State: " +
      sqle.getSQLState());
    System.out.println("insertBinStream: SQL Error Code: " +
      sqle.getErrorCode());

  }
}
```

例: 以下の例は、バイナリー XML データをファイルから XML 列に挿入する方法を示しています。

```
...
SQLXML info = conn.createSQLXML();
OutputStream os = info.setBinaryStream ();
FileInputStream fis = new FileInputStream("c7.xml");
int read;
while ((read = fis.read ()) != -1) {
  os.write (read);
}

PreparedStatement insertStmt = null;
String sqls = null;
int cid = 1015;
sqls = "INSERT INTO MyCustomer (Cid, Info) VALUES (?, ?)";
insertStmt = conn.prepareStatement(sqls);
insertStmt.setInt(1, cid);
insertStmt.setSQLXML(2, info);
insertStmt.executeUpdate();
```

JDBC アプリケーションでの XML データの取り出し

JDBC アプリケーションでは、ResultSet.getXXX または ResultSet.getObject メソッドを使用して XML 列からデータを取り出すことができます。

JDBC アプリケーションでは、 DB2 表の XML 列からデータを XML テキスト・データとして取り出せます。データ・サーバーがバイナリー XML データをサポートする場合、表の XML 列からデータをバイナリー XML データ (拡張可能動的バイナリー XML DB2 クライアント/サーバー・バイナリー XML 形式のデータ) として取り出せます。

XML データを取り出すには、以下の手法のいずれかを使用することができます。

- ResultSet.getSQLXML メソッドを使用してデータを取り出します。次に、SQLXML.getXXX メソッドを使用して、データを互換性のある出力データ・タイプに取り出します。この手法には、JDBC 4.0 以降が必要です。

  例えば、SQLXML.getBinaryStream メソッドまたは SQLXML.getSource メソッドを使用してデータを取り出せます。
- ResultSet.getObject 以外の ResultSet.getXXX メソッドを使用して、互換性のあるデータ・タイプにデータを取り出します。
- ResultSet.getObject メソッドを使用してデータを取り出してから、それを DB2Xml タイプにキャストし、DB2Xml オブジェクトに割り当てます。次に、DB2Xml.getDB2XXX または DB2Xml.getDB2XmlXXX メソッドを使用して、互換性のある出力データ・タイプにデータを取り出します。

  JDBC 4.0 をサポートするバージョンの IBM Data Server Driver for JDBC and SQLJ を使用していない場合、この手法を使用する必要があります。

次の表には、XML データを取り出すための ResultSet メソッドおよび対応する出力データ・タイプがリストされています。

表 29. XML データを取り出すための ResultSet メソッドおよびデータ・タイプ

メソッド	出力データ・タイプ
ResultSet.getAsciiStream	InputStream
ResultSet.getBinaryStream	InputStream
ResultSet.getBytes	byte[]
ResultSet.getCharacterStream	Reader
ResultSet.getObject	オブジェクト
ResultSet.getSQLXML	SQLXML
ResultSet.getString	String

次の表には、java.sql.SQLXML または com.ibm.db2.jcc.DB2Xml オブジェクトからのデータの取り出しに呼び出すことができるメソッド、および対応する出力データ・タイプと XML 宣言でのエンコード方式のタイプがリストされています。

表 30. SQLXML および DB2Xml メソッド、データ・タイプ、および追加されるエンコード仕様

メソッド	出力データ・タイプ	追加される XML 内部エンコード宣言のタイプ
SQLXML.getBinaryStream	InputStream	なし
SQLXML.getCharacterStream	Reader	なし
SQLXML.getSource	Source[1]	なし
SQLXML.getString	String	なし
DB2Xml.getDB2AsciiStream	InputStream	なし
DB2Xml.getDB2BinaryStream	InputStream	なし
DB2Xml.getDB2Bytes	byte[]	なし
DB2Xml.getDB2CharacterStream	Reader	なし
DB2Xml.getDB2String	String	なし
DB2Xml.getDB2XmlAsciiStream	InputStream	US-ASCII

表 30. SQLXML および DB2Xml メソッド、データ・タイプ、および追加されるエンコード仕様 (続き)

メソッド	出力データ・タイプ	追加される XML 内部エンコード宣言のタイプ
DB2Xml.getDB2XmlBinaryStream	InputStream	getDB2XmlBinaryStream *targetEncoding* パラメーターで指定
DB2Xml.getDB2XmlBytes	byte[]	DB2Xml.getDB2XmlBytes *targetEncoding* パラメーターで指定
DB2Xml.getDB2XmlCharacterStream	Reader	ISO-10646-UCS-2
DB2Xml.getDB2XmlString	String	ISO-10646-UCS-2

注:

1. 戻されるクラスは getSource の呼び出し側によって指定されますが、このクラスは javax.xml.transform.Source を拡張するものです。

戻されるデータに対してアプリケーションが XMLSERIALIZE 関数を実行すると、関数の実行後に、そのデータには XML データ・タイプではなく XMLSERIALIZE 関数で指定されたデータ・タイプが含まれます。そのため、ドライバーは指定されたタイプとしてそのデータを処理し、内部エンコード宣言をすべて無視します。

例: 以下の例は、データを XML 列から SQLXML オブジェクトに取り出してから、SQLXML.getString メソッドを使用して、ストリングにデータを取り出す方法を示しています。

```
public void fetchToSQLXML(long cid, java.sql.Connection conn)
  {
    System.out.println(">> fetchToSQLXML: Get XML data as an SQLXML object " +
    "using getSQLXML");
    PreparedStatement selectStmt = null;
    String sqls = null, stringDoc = null;
    ResultSet rs = null;

    try{
      sqls = "SELECT info FROM customer WHERE cid = " + cid;
      selectStmt = conn.prepareStatement(sqls);
      rs = selectStmt.executeQuery();

      // Get metadata
      // Column type for XML column is the integer java.sql.Types.OTHER
      ResultSetMetaData meta = rs.getMetaData();
      int colType = meta.getColumnType(1);
      System.out.println("fetchToSQLXML: Column type = " + colType);
      while (rs.next()) {
        // Retrieve the XML data with getSQLXML.
        // Then write it to a string with
        // explicit internal ISO-10646-UCS-2 encoding.
        java.sql.SQLXML xml = rs.getSQLXML(1);
        System.out.println (xml.getString());
      }
      rs.close();
    }
    catch (SQLException sqle) {
      System.out.println("fetchToSQLXML: SQL Exception: " +
        sqle.getMessage());
      System.out.println("fetchToSQLXML: SQL State: " +
        sqle.getSQLState());
      System.out.println("fetchToSQLXML: SQL Error Code: " +
        sqle.getErrorCode());
    }
  }
```

例: 以下の例は、データを XML 列から SQLXML オブジェクトに取り出してから、SQLXML.getBinaryStream メソッドを使用して、データをバイナリー・データとして InputStream に取り出す方法を示しています。

```
String sql = "SELECT INFO FROM Customer WHERE Cid='1000'";
PreparedStatement pstmt = con.prepareStatement(sql);
ResultSet resultSet = pstmt.executeQuery();
// Get the result XML as a binary stream
SQLXML sqlxml = resultSet.getSQLXML(1);
InputStream binaryStream = sqlxml.getBinaryStream();
```

例: 以下の例は、データを XML 列からストリング変数に取り出す方法を示しています。

```
public void fetchToString(long cid, java.sql.Connection conn)
  {
    System.out.println(">> fetchToString: Get XML data " +
    "using getString");
    PreparedStatement selectStmt = null;
    String sqls = null, stringDoc = null;
    ResultSet rs = null;

    try{
      sqls = "SELECT info FROM customer WHERE cid = " + cid;
      selectStmt = conn.prepareStatement(sqls);
      rs = selectStmt.executeQuery();

      // Get metadata
      // Column type for XML column is the integer java.sql.Types.OTHER
      ResultSetMetaData meta = rs.getMetaData();
      int colType = meta.getColumnType(1);
      System.out.println("fetchToString: Column type = " + colType);

      while (rs.next()) {
        stringDoc = rs.getString(1);
        System.out.println("Document contents:");
        System.out.println(stringDoc);
 }
    catch (SQLException sqle) {
      System.out.println("fetchToString: SQL Exception: " +
        sqle.getMessage());
      System.out.println("fetchToString: SQL State: " +
        sqle.getSQLState());
      System.out.println("fetchToString: SQL Error Code: " +
        sqle.getErrorCode());
    }
  }
```

例: 以下の例は、データを XML 列から DB2Xml オブジェクトに取り出してから、DB2Xml.getDB2XmlString メソッドを使用して、ISO-10646-UCS-2 エンコード仕様で追加された XML 宣言を含むストリングにデータを取り出す方法を示しています。

```
public void fetchToDB2Xml(long cid, java.sql.Connection conn)
  {
    System.out.println(">> fetchToDB2Xml: Get XML data as a DB2XML object " +
    "using getObject");
    PreparedStatement selectStmt = null;
    String sqls = null, stringDoc = null;
    ResultSet rs = null;

    try{
      sqls = "SELECT info FROM customer WHERE cid = " + cid;
      selectStmt = conn.prepareStatement(sqls);
      rs = selectStmt.executeQuery();
```

```
      // Get metadata
      // Column type for XML column is the integer java.sql.Types.OTHER
      ResultSetMetaData meta = rs.getMetaData();
      int colType = meta.getColumnType(1);
      System.out.println("fetchToDB2Xml: Column type = " + colType);
      while (rs.next()) {
        // Retrieve the XML data with getObject, and cast the object
        // as a DB2Xml object. Then write it to a string with
        // explicit internal ISO-10646-UCS-2 encoding.
        com.ibm.db2.jcc.DB2Xml xml =
          (com.ibm.db2.jcc.DB2Xml) rs.getObject(1);
        System.out.println (xml.getDB2XmlString());
      }
      rs.close();
    }
    catch (SQLException sqle) {
      System.out.println("fetchToDB2Xml: SQL Exception: " +
        sqle.getMessage());
      System.out.println("fetchToDB2Xml: SQL State: " +
        sqle.getSQLState());
      System.out.println("fetchToDB2Xml: SQL Error Code: " +
        sqle.getErrorCode());
    }
  }
```

Java アプリケーションでの XML パラメーターを指定したルーチンの呼び出し

Java アプリケーションは、DB2 Database for Linux, UNIX, and Windows あるいは DB2 for z/OS データ・ソースにある、XML パラメーターが指定されたストアード・プロシージャーを呼び出すことができます。

ネイティブ SQL プロシージャーの場合、ストアード・プロシージャー定義内の XML パラメーターは XML タイプです。DB2 Database for Linux, UNIX, and Windows データ・ソース上の外部ストアード・プロシージャーおよびユーザー定義関数の場合は、ルーチン定義内の XML パラメーターは XML AS CLOB タイプです。 XML パラメーターのあるストアード・プロシージャーまたはユーザー定義関数を呼び出す場合は、呼び出しステートメントで互換データ・タイプを使用する必要があります。

JDBC プログラムから XML 入力パラメーターのあるルーチンを呼び出すには、java.sql.SQLXML または com.ibm.db2.jcc.DB2Xml タイプのパラメーターを使用します。 XML 出力パラメーターを登録するには、パラメーターを java.sql.Types.SQLXML または com.ibm.db2.jcc.DB2Types.XML タイプで登録します。 (com.ibm.db2.jcc.DB2Xml および com.ibm.db2.jcc.DB2Types.XML タイプは推奨されません。)

例: 3 つの XML パラメーター (IN パラメーター、OUT パラメーター、および INOUT パラメーター) を使用するストアード・プロシージャーを呼び出す JDBC プログラム。この例では JDBC 4.0 以降が必要です。

```
java.sql.SQLXML in_xml = xmlvar;
java.sql.SQLXML out_xml = null;
java.sql.SQLXML inout_xml = xmlvar;
                                  // Declare an input, output, and
                                  // INOUT XML parameter
Connection con;
CallableStatement cstmt;
ResultSet rs;
```

```
...
cstmt = con.prepareCall("CALL SP_xml(?,?,?)");
                                  // Create a CallableStatement object
cstmt.setObject (1, in_xml);      // Set input parameter
cstmt.setObject (3, inout_xml);   // Set inout parameter
cstmt.registerOutParameter (2, java.sql.Types.SQLXML);
                                  // Register out and input parameters
cstmt.registerOutParameter (3, java.sql.Types.SQLXML);
cstmt.executeUpdate();            // Call the stored procedure
out_xml = cstmt.getSQLXML(2);     // Get the OUT parameter value
inout_xml = cstmt.getSQLXML(3);   // Get the INOUT parameter value
System.out.println("Parameter values from SP_xml call: ");
System.out.println("Output parameter value ");
MyUtilities.printString(out_xml.getString());
                           // Use the SQLXML.getString
                           // method to convert the out_xml
                           // value to a string for printing.
                           // Call a user-defined method called
                           // printString (not shown) to print
                           // the value.
System.out.println("INOUT parameter value ");
MyUtilities.printString(inout_xml.getString());
                           // Use the SQLXML.getString
                           // method to convert the inout_xml
                           // value to a string for printing.
                           // Call a user-defined method called
                           // printString (not shown) to print
                           // the value.
```

SQLJ プログラムから XML パラメーターのあるルーチンを呼び出すには、java.sql.SQLXML または com.ibm.db2.jcc.DB2Xml タイプのパラメーターを使用します。

例: 3 つの XML パラメーター (IN パラメーター、OUT パラメーター、および INOUT パラメーター) を使用するストアード・プロシージャーを呼び出す SQLJ プログラム。この例では JDBC 4.0 以降が必要です。

```
java.sql.SQLXML in_xml = xmlvar;
java.sql.SQLXML out_xml = null;
java.sql.SQLXML inout_xml = xmlvar;
                                  // Declare an input, output, and
                                  // INOUT XML parameter
...
#sql [myConnCtx] {CALL SP_xml(:IN in_xml,
                              :OUT out_xml,
                              :INOUT inout_xml)};
                                  // Call the stored procedure
System.out.println("Parameter values from SP_xml call: ");
System.out.println("Output parameter value ");
MyUtilities.printString(out_xml.getString());
                           // Use the SQLXML.getString
                           // method toconvert the out_xml value
                           // to a string for printing.
                           // Call a user-defined method called
                           // printString (not shown) to print
                           // the value.
System.out.println("INOUT parameter value ");
MyUtilities.printString(inout_xml.getString());
                           // Use the SQLXML.getString
                           // method to convert the inout_xml
                           // value to a string for printing.
                           // Call a user-defined method called
                           // printString (not shown) to print
                           // the value.
```

SQLJ

SQLJ アプリケーションでの XML データ

SQLJ アプリケーションでは、XML 列へのデータの保管、および XML 列からのデータの取り出しが可能です。

DB2 表で、ツリー形式でノードを構造化した集合として XML データを列に保管するには、XML 組み込みデータ・タイプを使用します。

SQLJ アプリケーションは、以下のいずれかの形式で、データ・サーバーに XML データを送ったり、データ・サーバーから XML データを取り出したりすることができます。

- テキスト XML データ
- データ・サーバーがサポートする場合はバイナリー XML データ (拡張可能動的バイナリー XML DB2 クライアント/サーバー・バイナリー XML 形式 のデータ)

SQLJ アプリケーションでは以下を行うことができます。

- INSERT、UPDATE、または MERGE ステートメントを使用して XML 文書全体を XML 列に保管します。
- 単一行の SELECT ステートメントまたはイテレーターを使用して XML 文書全体を XML 列から取得します。
- SQL XMLQUERY 関数を使用してシーケンスをデータベース内に取り出してから、単一行の SELECT ステートメントまたはイテレーターを使用して、シリアライズされた XML ストリング・データをアプリケーション変数に取り出すことによって、XML 列の文書からシーケンスを取り出します。
- ストリング 'XQUERY' が前に付加されている XQuery 式を使用して、シーケンスの要素をデータベース内の結果表に取り出すことによって、XML 列の文書からシーケンスを取り出します。結果表の各行はシーケンス内のアイテムを表します。続いて、単一行の SELECT ステートメントまたはイテレーターを使用してデータをアプリケーション変数に取り出します。
- SQL XMLTABLE 関数を使用して結果表を定義し、その表を取り出すことによって、XML 列の文書からユーザー定義表としてシーケンスを取り出します。続いて、単一行の SELECT ステートメントまたはイテレーターを使用して、データを結果表からアプリケーション変数に取り出します。
- テキスト XML データの形式で、XML データを更新したり取り出したりすることができます。あるいは、バイナリー XML データをサポートするデータ・サーバーとの接続では、バイナリー XML データとして XML データを更新したり取り出したりすることができます。 Datasource または Connection のプロパティー xmlFormat を使用して、データ形式がテキスト XML かバイナリー XML であるかを制御します。XML データの形式について、アプリケーションが関与する必要はありません。DB2 for z/OS データ・サーバーでのバイナリー XML データの保管および取り出しには、IBM Data Server Driver for JDBC and SQLJ バージョン 4.9 以降が必要です。 DB2 Database for Linux, UNIX, and Windows データ・サーバーでのバイナリー XML データの保管および取り出しには、IBM Data Server Driver for JDBC and SQLJ バージョン 4.11 以降が必要です。

JDBC 4.0 の java.sql.SQLXML オブジェクトを使用して、XML 列内のデータの取り出しおよび更新を行うことができます。 ResultSetMetaData.getColumnType などのメタデータ・メソッドを呼び出すと、XML 列タイプの整数値 java.sql.Types.SQLXML が戻されます。

SQLJ アプリケーションでの XML 列の更新

SQLJ アプリケーションでは、DB2 データ・サーバーにある表の XML 列で、XML テキスト・データを使用してデータを更新または挿入することができます。データ・サーバーがバイナリー XML データをサポートしている場合、表の XML 列のデータを更新したり挿入したりするのに、バイナリー XML データ (拡張可能動的バイナリー XML DB2 クライアント/サーバー・バイナリー XML 形式のデータ) を使用できます。

XML 列の更新に使用できるホスト式のデータ・タイプは以下のとおりです。

- java.sql.SQLXML (SDK for Java バージョン 6 以降、および IBM Data Server Driver for JDBC and SQLJ バージョン 4.0 以降が必要)
- com.ibm.db2.jcc.DB2Xml (非推奨)
- String
- byte
- Blob
- Clob
- sqlj.runtime.AsciiStream
- sqlj.runtime.BinaryStream
- sqlj.runtime.CharacterStream

XML データのエンコードは、データ自体から (*内部的にエンコードされた* と呼びます) か、または外部ソースから (*外部的にエンコードされた* と呼びます) 導出されます。データベース・サーバーにバイナリー・データとして送信される XML データは、内部的にエンコードされたデータとして処理されます。データ・ソースに文字データとして送信される XML データは、外部的にエンコードされたデータとして処理されます。外部エンコード方式は JVM のデフォルトのエンコード方式です。

Java アプリケーションの外部エンコード方式は、常に Unicode エンコード方式です。

外部的にエンコードされたデータに、内部エンコード方式を含めることができます。つまり、このデータはデータ・ソースに文字データとして送信されますが、このデータにエンコード方式の情報を含めることができます。データ・ソースにより、以下のように内部エンコード方式と外部エンコード方式の間の非互換性が処理されます。

- データ・ソースが DB2 Database for Linux, UNIX, and Windows の場合、外部エンコード方式と内部エンコード方式の間に互換性がなく、外部エンコード方式と内部エンコード方式が Unicode でない場合は、データ・ソースによりエラーが生成されます。外部エンコード方式と内部エンコード方式が Unicode の場合は、データ・ソースでは内部エンコード方式が無視されます。
- データ・ソースが DB2 for z/OS の場合は、データ・ソースでは内部エンコード方式が無視されます。

XML 列のデータは UTF-8 エンコード方式で保管されます。

例: 次のステートメントを使用して、データを String ホスト式 xmlString から表の XML 列に挿入するとします。 xmlString は文字タイプのため、内部エンコード方式が指定されているかどうかに関係なく、外部エンコード方式が使用されます。

```
#sql [ctx] {INSERT INTO CUSTACC VALUES (1, :xmlString)};
```

例: データを xmlString から CP500 エンコード方式のバイト配列にコピーするとします。データには、CP500 のエンコード方式宣言を持つ XML 宣言が含まれています。次に、byte[] ホスト式から表の XML 列にデータを挿入します。

```
byte[] xmlBytes = xmlString.getBytes("CP500");
#sql[ctx] {INSERT INTO CUSTACC VALUES (4, :xmlBytes)};
```

バイト・ストリングは、内部的にエンコードされたデータとみなされます。データはその内部コード化スキームから UTF-8 に変換され、必要な場合はデータ・ソース上に階層形式で保管されます。

例: データを xmlString から US-ASCII エンコード方式のバイト配列にコピーするとします。 sqlj.runtime.AsciiStream ホスト式を構成し、データを sqlj.runtime.AsciiStream ホスト式からデータ・ソース上にある表の XML 列に挿入します。

```
byte[] b = xmlString.getBytes("US-ASCII");
java.io.ByteArrayInputStream xmlAsciiInputStream =
  new java.io.ByteArrayInputStream(b);
sqlj.runtime.AsciiStream sqljXmlAsciiStream =
  new sqlj.runtime.AsciiStream(xmlAsciiInputStream,  b.length);
#sql[ctx] {INSERT INTO CUSTACC VALUES (4, :sqljXmlAsciiStream)};
```

sqljXmlAsciiStream はストリーム・タイプであるため、内部エンコード方式が使用されます。データはその内部エンコード方式から UTF-8 エンコード方式に変換され、データ・ソース上に階層形式で保管されます。

例: sqlj.runtime.CharacterStream ホスト式: sqlj.runtime.CharacterStream ホスト式を構成し、データを sqlj.runtime.CharacterStream ホスト式から表の XML 列に挿入します。

```
java.io.StringReader xmlReader =
  new java.io.StringReader(xmlString);
sqlj.runtime.CharacterStream sqljXmlCharacterStream =
  new sqlj.runtime.CharacterStream(xmlReader,  xmlString.length());
#sql [ctx] {INSERT INTO CUSTACC VALUES (4, :sqljXmlCharacterStream)};
```

sqljXmlCharacterStream は文字タイプのため、内部エンコード方式が指定されているかどうかに関係なく、外部エンコード方式が使用されます。

例: 文書を XML 列から java.sql.SQLXML ホスト式に取り出し、このデータを表の XML 列に挿入します。

```
java.sql.ResultSet rs = s.executeQuery ("SELECT * FROM CUSTACC");
rs.next();
java.sql.SQLXML xmlObject = (java.sql.SQLXML)rs.getObject(2);
#sql [ctx] {INSERT INTO CUSTACC VALUES (6, :xmlObject)};
```

データを取り出した後、このデータは UTF-8 エンコード方式のままなので、このデータを他の XML 列に挿入する場合も変換は実行されません。

例: 文書を XML 列から com.ibm.db2.jcc.DB2Xml ホスト式に取り出し、このデータを表の XML 列に挿入します。

```
java.sql.ResultSet rs = s.executeQuery ("SELECT * FROM CUSTACC");
rs.next();
com.ibm.db2.jcc.DB2Xml xmlObject = (com.ibm.db2.jcc.DB2Xml)rs.getObject(2);
#sql [ctx] {INSERT INTO CUSTACC VALUES (6, :xmlObject)};
```

データを取り出した後、このデータは UTF-8 エンコード方式のままなので、このデータを他の XML 列に挿入する場合も変換は実行されません。

SQLJ アプリケーションでの XML データの取り出し

SQLJ アプリケーションで、データをデータベース表の XML 列から取り出す場合は、出力データは明示的または暗黙的に直列化される必要があります。

データの XML 列からの取り出しに使用できるホスト式またはイテレーターのデータ・タイプは以下のとおりです。

- java.sql.SQLXML (SDK for Java バージョン 6 以降、および IBM Data Server Driver for JDBC and SQLJ バージョン 4.0 以降が必要)
- com.ibm.db2.jcc.DB2Xml (非推奨)
- String
- byte[]
- sqlj.runtime.AsciiStream
- sqlj.runtime.BinaryStream
- sqlj.runtime.CharacterStream

アプリケーションで、データの取り出しの前に XMLSERIALIZE 関数が呼び出されない場合は、データは UTF-8 から文字データ・タイプ用の外部アプリケーション・エンコード方式か、またはバイナリー・データ・タイプ用の内部エンコード方式に変換されます。 XML 宣言は追加されません。ホスト式が java.sql.SQLXML または com.ibm.db2.jcc.DB2Xml タイプのオブジェクトの場合は、追加のメソッドを呼び出して、データをこのオブジェクトから取り出す必要があります。呼び出すメソッドにより、出力データのエンコード方式、およびエンコード方式の指定を含む XML 宣言が追加されるかどうかが決まります。

次の表には、java.sql.SQLXML または com.ibm.db2.jcc.DB2Xml オブジェクトからのデータの取り出しに呼び出すことができるメソッド、および対応する出力データ・タイプと XML 宣言でのエンコード方式のタイプがリストされています。

表 31. SQLXML および DB2Xml メソッド、データ・タイプ、および追加されるエンコード仕様

メソッド	出力データ・タイプ	追加される XML 内部エンコード宣言のタイプ
SQLXML.getBinaryStream	InputStream	なし
SQLXML.getCharacterStream	Reader	なし
SQLXML.getSource	Source	なし
SQLXML.getString	String	なし
DB2Xml.getDB2AsciiStream	InputStream	なし
DB2Xml.getDB2BinaryStream	InputStream	なし
DB2Xml.getDB2Bytes	byte[]	なし
DB2Xml.getDB2CharacterStream	Reader	なし
DB2Xml.getDB2String	String	なし

表 31. *SQLXML および DB2Xml メソッド、データ・タイプ、および追加されるエンコード仕様 (続き)*

メソッド	出力データ・タイプ	追加される XML 内部エンコード宣言のタイプ
DB2Xml.getDB2XmlAsciiStream	InputStream	US-ASCII
DB2Xml.getDB2XmlBinaryStream	InputStream	getDB2XmlBinaryStream *targetEncoding* パラメーターで指定
DB2Xml.getDB2XmlBytes	byte[]	DB2Xml.getDB2XmlBytes *targetEncoding* パラメーターで指定
DB2Xml.getDB2XmlCharacterStream	Reader	ISO-10646-UCS-2
DB2Xml.getDB2XmlString	String	ISO-10646-UCS-2

戻されるデータに対してアプリケーションが XMLSERIALIZE 関数を実行すると、関数の実行後に、そのデータには XML データ・タイプではなく XMLSERIALIZE 関数で指定されたデータ・タイプが含まれます。そのため、ドライバーは指定されたタイプとしてそのデータを処理し、内部エンコード宣言をすべて無視します。

例: データを XML 列から String ホスト式に取り出します。

```
#sql iterator  XmlStringIter (int, String);
#sql [ctx] siter = {SELECT C1, CADOC from CUSTACC};
#sql {FETCH :siter INTO :row, :outString};
```

String タイプは文字タイプであるため、データは UTF-8 から外部エンコード方式に変換され (これがデフォルトの JVM エンコード方式)、XML 宣言なしで返されます。

例: データを XML 列から byte[] ホスト式に取り出します。

```
#sql iterator  XmlByteArrayIter (int, byte[]);
XmlByteArrayIter biter = null;
#sql [ctx] biter =  {SELECT c1, CADOC from CUSTACC};
#sql {FETCH :biter INTO :row, :outBytes};
```

byte[] タイプはバイナリー・タイプであるため、UTF-8 エンコード方式からのデータ変換は実行されず、データは XML 宣言なしで返されます。

例: 文書を XML 列から java.sql.SQLXML ホスト式に取り出しますが、バイナリー・ストリームのデータが必要な場合です。

```
#sql iterator  SqlXmlIter (int, java.sql.SQLXML);
SqlXmlIter SQLXMLiter = null;
java.sql.SQLXML outSqlXml = null;
#sql [ctx] SqlXmlIter = {SELECT c1, CADOC from CUSTACC};
#sql {FETCH :SqlXmlIter INTO :row, :outSqlXml};
java.io.InputStream XmlStream = outSqlXml.getBinaryStream();
```

FETCH ステートメントでは、データが UTF-8 エンコード方式で SQLXML オブジェクトに取り出されます。 SQLXML.getBinaryStream では、データがバイナリー・ストリームで保管されます。

例: 文書を XML 列から com.ibm.db2.jcc.DB2Xml ホスト式に取り出しますが、UTF-8 の内部エンコード方式の指定が含まれた XML 宣言を持つバイト・ストリングとしてデータが必要な場合です。

```
#sql iterator  DB2XmlIter (int, com.ibm.db2.jcc.DB2Xml);
DB2XmlIter db2xmliter = null;
com.ibm.db2.jcc.DB2Xml outDB2Xml = null;
```

```
#sql [ctx] db2xmliter = {SELECT c1, CADOC from CUSTACC};
#sql {FETCH :db2xmliter INTO :row, :outDB2Xml};
byte[] byteArray = outDB2XML.getDB2XmlBytes("UTF-8");
```

FETCH ステートメントでは、データが UTF-8 エンコード方式で DB2Xml オブジェクトに取り出されます。 UTF-8 引数のある getDB2XmlBytes メソッドでは、UTF-8 エンコード方式の指定を含む XML 宣言が追加され、データがバイト配列で保管されます。

PHP

IBM データ・サーバー用の PHP アプリケーション開発

PHP: Hypertext Preprocessor (PHP) とは、Web アプリケーションの開発のために広く使用されているインタープリター型プログラミング言語です。 PHP は学習しやすく、実用的なソリューションに焦点を合わせており、Web アプリケーションで一般に最も必要とされる機能をサポートしているため、Web 開発で広く使用される言語となりました。

PHP はモジュラー言語であり、拡張モジュールを使用することによって、使用できる機能をカスタマイズできます。これらの拡張モジュールを使用すれば、XML の読み取り、書き込み、および操作、SOAP クライアント/サーバーの作成、およびサーバーとブラウザーとの間の通信の暗号化などのタスクを単純化できます。ただし、PHP の最も一般的な拡張モジュールは、データベースへの読み取り/書き込みアクセスを提供するものであり、これにより動的なデータベース・ドリブンの Web サイトを簡単に作成できます。

IBM は、IBM データ・サーバー・データベースにアクセスするための、リストされた PHP 拡張モジュールを提供しています。

ibm_db2
: プロシージャー型アプリケーション・プログラミング・インターフェース (API)。これは通常のデータベースの作成、読み取り、更新、および書き込み操作に加え、データベース・メタデータへの広範なアクセスも行います。ibm_db2 拡張モジュールは、PHP 4 または PHP 5 のいずれかでコンパイルできます。この拡張モジュールは、IBM によって作成、保守、およびサポートされています。

pdo_ibm
: PDO (PHP Data Objects) 拡張モジュール用のドライバー。これは、PHP 5.1 で導入された標準オブジェクト指向データベース・インターフェースによる、 IBM データ・サーバー・データベースへのアクセスを提供します。

これらの拡張は、IBM Data Server Driver Package (DS Driver) バージョン 1.7.0 に含まれています。 IBM DB2 バージョン 9.7 for Linux, UNIX, and Windows への接続において、このバージョンまたはそれ以降のバージョンがサポートされています。 ibm_db2 拡張モジュールのバージョンをチェックするには、**php --re ibm_db2** コマンドを発行することができます。

ibm_db2 と pdo_ibm の最新バージョンは、PHP Extension Community Library (PECL) (http://pecl.php.net/) から入手できます。

PHP アプリケーションは、リストされた IBM データ・サーバーのデータベースにアクセスすることができます。

- IBM DB2 バージョン 9.1 for Linux, UNIX, and Windows、 Fix Pack 2 以降
- IBM DB2 Universal Database™ (DB2 UDB) バージョン 8 for Linux, UNIX, and Windows、 Fixpak 15 以降
- IBM DB2 for IBM i V5R3 へのリモート接続
- IBM DB2 for IBM i バージョン 5.4 以降へのリモート接続
- IBM DB2 for z/OS、バージョン 8 以降へのリモート接続

3 番目の拡張モジュールである Unified ODBC は、これまで DB2 データベース・システムへのアクセスを提供してきました。ただし新しいアプリケーションの場合、ibm_db2 および pdo_ibm は Unified ODBC を上回るパフォーマンスおよび安定度における大きな利点があるため、これらのいずれかを使用することができます。 ibm_db2 拡張モジュール API を使用すれば、Unified ODBC 用に以前に作成されたアプリケーションの移植は、ほぼ、アプリケーションのソース・コード全体にわたって `odbc_`関数名を `db2_` に変更するだけで容易に行うことができます。

PHP アプリケーションを使用した XML データ検索

ibm_db2 API は、IBM データ・サーバー・データベースにアクセスするための PHP 拡張モジュールです。 ibm_db2 API は DB2 データベースへの接続および XML 列からの XML データの戻しをサポートしています。

ibm_db2 API は、XMLTABLE などの DB2 関数の使用および SQL ステートメントでの XQuery 式の実行もサポートしています。 SQL ステートメントで、XMLQUERY 関数を使用して XQuery 式を呼び出します。

ibm_db2 API を使用した PHP アプリケーションの開発については、*Perl*、*PHP*、*Python* および *Ruby on Rails* アプリケーションの開発の『ibm_db2 を使用した PHP でのアプリケーション開発』を参照してください。

PHP ダウンロードおよび関連リソース

IBM データ・サーバー用の PHP アプリケーション開発に役立つ多くのリソースが入手可能です。

表 32. PHP ダウンロードおよび関連リソース

ダウンロード	
完全な PHP ソース・コード[1]	http://www.php.net/downloads.php
PHP Extension Community Library (PECL) からの ibm_db2 および pdo_ibm	http://pecl.php.net/
IBM Data Server Driver Package (DS Driver)	http://www.ibm.com/software/data/support/data-server-clients/index.html
Zend Server	http://www.zend.com/en/products/server/downloads
PHP Manual	http://www.php.net/docs.php
ibm_db2 API 資料	http://www.php.net/ibm_db2
PDO API 資料	http://php.net/manual/en/book.pdo.php

表 32. PHP ダウンロードおよび関連リソース (続き)

ダウンロード	
PHP Web サイト	http://www.php.net/

1. Windows バイナリーを含みます。ほとんどの Linux ディストリビューションには、既にプリコンパイルされた PHP が付属しています。

Perl

pureXML と Perl

DBD::DB2 ドライバーは、DB2 pureXML をサポートしています。 pureXML のサポートにより、DBD::DB2 ドライバーを通してデータへのより直接的なアクセスが可能になります。また、アプリケーションとデータベース間の通信がより透過性のあるものとなり、アプリケーション・ロジックの削減に役立ちます。

pureXML のサポートがあるため、DB2 データベースに XML 文書を直接挿入できます。データベースに XML 文書を挿入すると pureXML パーサーが自動的に実行されるので、アプリケーションが XML 文書を解析する必要はもはやありません。文書の解析処理がアプリケーション外で行われるため、アプリケーションのパフォーマンスが向上し、保守の労力も削減されます。DBD::DB2 ドライバーで XML 保管データを取り出す作業も簡単です。BLOB またはレコードを使用してデータにアクセスできます。

DB2 Perl Database Interface に関する情報や、最新の DBD::DB2 ドライバーをダウンロードする方法については、http://www.ibm.com/software/data/db2/perlを参照してください。

例

例は、pureXML を使用する Perl プログラムです。

```
#!/usr/bin/perl
use DBI;
use strict ;

# Use DBD:DB2 module:
#   to create a simple DB2 table with an XML column
#   Add one row of data
#   retreive the XML data as a record or a LOB (based on $datatype).

# NOTE: the DB2 SAMPLE database must already exist.

my $database='dbi:DB2:sample';
my $user='';
my $password='';

my $datatype = "record" ;
#  $datatype = "LOB" ;

my $dbh = DBI->connect($database, $user, $password)
  or die "Can't connect to $database: $DBI::errstr";

# For LOB datatype, LongReadLen = 0 -- no data is retrieved on initial fetch
$dbh->{LongReadLen} = 0 if $datatype eq "LOB" ;
```

```
# SQL CREATE TABLE to create test table
my $stmt = "CREATE TABLE xmlTest (id INTEGER, data XML)";
my $sth = $dbh->prepare($stmt);
$sth->execute();

#insert one row of data into table
insertData() ;

# SQL SELECT statement returns home phone element from XML data
$stmt = qq(
 SELECT XMLQUERY ('
 ¥$d/*:customerinfo/*:phone[¥@type = "home"] '
 passing data as "d")
 FROM xmlTest
) ;


# prepare and execute SELECT statement
$sth = $dbh->prepare($stmt);
$sth->execute();

# Print data returned from select statement
if($datatype eq "LOB") {
    printLOB() ;
}
else {
 printRecord() ;
}

# Drop table
$stmt = "DROP TABLE xmlTest" ;
$sth = $dbh->prepare($stmt);
$sth->execute();

warn $DBI::errstr if $DBI::err;

$sth->finish;
$dbh->disconnect;


##############


sub printRecord {
 print "output data as as record¥n" ;

 while( my @row = $sth->fetchrow )
 {
  print $row[0] . "¥n";
 }

 warn $DBI::errstr if $DBI::err;
}


sub printLOB {
 print "output as Blob data¥n" ;

 my $offset = 0;
 my $buff="";
 $sth->fetch();
 while( $buff = $sth->blob_read(1,$offset,1000000)) {
  print $buff;
  $offset+=length($buff);
  $buff="";
 }
 warn $DBI::errstr if $DBI::err;
```

```
}


sub insertData {

 # insert a row of data
 my $xmlInfo = qq(¥'
 <customerinfo xmlns="http://posample.org" Cid="1011">
   <name>Bill Jones</name>
   <addr country="Canada">
     <street>5 Redwood</street>
     <city>Toronto</city>
     <prov-state>Ontario</prov-state>
     <pcode-zip>M6W 1E9</pcode-zip>
   </addr>
   <phone type="work">416-555-9911</phone>
   <phone type="home">416-555-1212</phone>
 </customerinfo>
 \') ;

 my $catID = 1011 ;

 # SQL statement to insert data.
 my $Sql = qq(
  INSERT INTO xmlTest (id, data)
      VALUES($catID, $xmlInfo )
 );

 $sth = $dbh->prepare( $Sql )
   or die "Can't prepare statement: $DBI::errstr";

 my $rc = $sth->execute
   or die "Can't execute statement: $DBI::errstr";

 # check for problems
 warn $DBI::errstr if $DBI::err;
}
```

Perl でのデータベース接続

DBD::DB2 ドライバーは、DBI API によって定義された標準データベース接続機能のサポートを提供します。

Perl が DBI モジュールをロードできるようにするには、アプリケーションに `use DBI;` 行を含める必要があります。

DBI モジュールは、**DBI->connect** ステートメントを使用してデータベース・ハンドルを作成すると、 DBD::DB2 ドライバーを自動的にロードします。リストされた構文を使用します。

```
my $dbhandle = DBI->connect('dbi:DB2:dsn', $userID, $password);
```

詳細は次のとおりです。

$dbhandle
: connect ステートメントが戻すデータベース・ハンドル。

dsn
: (ローカル接続用) DB2 データベース・ディレクトリーにカタログされている DB2 別名

(リモート接続用) リモート・ホストへの接続のためのホスト名、ポート番号、プロトコル、ユーザー ID、およびパスワードを含む、完全な接続ストリング

$userID
データベースへの接続で使用するユーザー ID

$password
データベースへの接続で使用するユーザー ID のパスワード

DBI API について詳しくは、http://search.cpan.org/~timb/DBI/DBI.pmhttp://search.cpan.org/~timb/DBI/DBI.pmを参照してください。

例

例 1: ローカル・ホスト上のデータベースに接続します (クライアントとサーバーが同じワークステーション上にあります)

```
use DBI;

$DATABASE = 'dbname';
$USERID = 'username';
$PASSWORD = 'password';

my $dbh = DBI->connect("dbi:DB2:$DATABASE", $USERID, $PASSWORD, {PrintError => 0})
or die "Couldn't connect to database: " . DBI->errstr;

$dbh->disconnect;
```

例 2: リモート・ホスト上のデータベースに接続します (クライアントとサーバーが別々のワークステーション上にあります)

```
use DBI;

$DSN="DATABASE=sample; HOSTNAME=host; PORT=60000; PROTOCOL=TCPIP; UID=username;
PWD=password";

my $dbh = DBI->connect("dbi:DB2:$DSN", $USERID, $PASSWORD, {PrintError => 0})
or die "Couldn't connect to database: " . DBI->errstr;

$dbh->disconnect;
```

Perl の制約事項

Perl でのアプリケーション開発で使用可能なサポートには、いくつかの制約事項が適用されます。

Perl DBI モジュールがサポートするのは、動的 SQL だけです。複数回ステートメントを実行する必要がある場合には、ステートメントを準備する **prepare** 呼び出しを発行して、 Perl アプリケーションのパフォーマンスを改善することができます。

Perl ではマルチスレッド・データベース・アクセスはサポートされていません。

ワークステーションにインストールする DBD::DB2 ドライバー・バージョンの制限に関する最新情報については、 DBD::DB2 パッケージにある CAVEATS ファイルを参照してください。

ルーチン

SQL プロシージャー

SQL プロシージャーにおける XML および XQuery のサポート

SQL プロシージャーは、データ・タイプが XML のパラメーターと変数をサポートします。他のデータ・タイプの変数と同様に、SQL ステートメントで使用できます。加えて、データ・タイプ XML の変数は、XMLEXISTS 式、XMLQUERY 式および XMLTABLE 式内の XQuery 式へのパラメーターとして渡すことができます。

以下の例は、SQL プロシージャーの XML パラメーターおよび変数の宣言、使用、および割り当てを示しています。

```
CREATE TABLE T1(C1 XML) %

CREATE PROCEDURE proc1(IN parm1 XML, IN parm2 VARCHAR(32000))
LANGUAGE SQL
BEGIN
   DECLARE var1 XML;

   /* check if the value of XML parameter parm1
      contains an item with a value less than 200 */
   IF(XMLEXISTS('$x/ITEM[value < 200]' passing by ref parm1 as "x"))THEN

     /* if it does, insert the value of parm1 into table T1 */
     INSERT INTO T1 VALUES(parm1);

   END IF;

   /* parse parameter parm2's value and assign it to a variable */
   SET var1 = XMLPARSE(document parm2 preserve whitespace);

   /* insert variable var1 into table T1
   INSERT INTO T1 VALUES(var1);

END %
```

この例には 1 つの XML 列を持つ表 T1 があります。 SQL プロシージャーは、データ・タイプ XML の parm1 および parm2 という 2 つのパラメーターを受け入れます。 SQL プロシージャー内では XML 変数は var1 という名前で宣言されます。

SQL プロシージャーのロジックは、XML パラメーター parm1 の値に 200 より小さい値を持つ項目が含まれるかどうかを検査します。含まれる場合、XML 値は直接、表 T1 の列 C1 に挿入されます。

その後、XMLPARSE 関数を使ってパラメーター parm2 の値が構文解析され、XML 変数 var1 に割り当てられます。そしてこの XML 変数の値も表 T1 の列 C1 に挿入されます。

XQuery 操作に制御フロー・ロジックをインプリメントすることができるので、データベースに保管されている XML データを照会してこれにアクセスする複雑なアルゴリズムを開発する作業が容易になります。

SQL プロシージャーにおける XQuery 式のカーソル

SQL プロシージャーは、XQuery 式でのカーソルの定義をサポートします。XQuery 式でカーソルを使用すると、式によって戻される XQuery 順序のエレメントを繰り返して使用できます。

動的または静的に定義できる SQL ステートメントで定義されたカーソルとは異なり、XQuery 式でのカーソルは動的にのみ定義できます。カーソルを動的に宣言するには、タイプ CHAR または VARCHAR の変数を宣言して、カーソルの結果セットを定義する XQuery 式を含める必要があります。XQuery 式は、カーソルをオープンして、結果セットを解決してから準備する必要があります。

XQuery 式用にカーソルを動的に宣言し、そのカーソルをオープンして、その後 XML データをフェッチする SQL プロシージャーの例を以下に示します。

```
CREATE PROCEDURE xmlProc(IN inCust XML, OUT resXML XML)
SPECIFIC xmlProc
LANGUAGE SQL
BEGIN
    DECLARE SQLSTATE CHAR(5);
    DECLARE stmt_text VARCHAR (1024);
    DECLARE customer XML;
    DECLARE cityXml XML;
    DECLARE city VARCHAR (100);
    DECLARE stmt STATEMENT;
    DECLARE cur1 CURSOR FOR stmt;

   -- Get the city of the input customer
   SET cityXml = XMLQUERY('$cust/customerinfo//city' passing inCust as "cust");
   SET city = XMLCAST(cityXml as VARCHAR(100));

  -- Iterate over all the customers from the city using an XQUERY cursor
  -- and collect the customer name values into the output XML value

   SET stmt_text = 'XQUERY for $cust
                    in db2-fn:xmlcolumn("CUSTOMER.INFO")
                       /*:customerinfo/*:addr[*:city= "' || city ||'"]
                    return <Customer>{$cust/../@Cid}{$cust/../*:name}</Customer>';

  -- Use the name of the city for the input customer data as a prefix
  SET resXML = cityXml;

  PREPARE stmt FROM stmt_text;
  OPEN cur1;

  FETCH cur1 INTO customer;
  WHILE (SQLSTATE = '00000') DO
    SET resXML = XMLCONCAT(resXML, customer);
    FETCH cur1 INTO customer;
  END WHILE;

  set resXML = XMLQUERY('<result> {$res} </result>'
                        passing resXML as "res");

END
```

この SQL プロシージャーは、XML データが入力パラメーターとして提供されている顧客と同じ市区町村に住んでいる、CUSTOMER という名前の表で定義された顧客の ID と名前を収集します。

この SQL プロシージャーは、以下のようにして CALL ステートメントを実行すると呼び出すことができます。

```
CALL xmlProc(xmlparse(document '<customerinfo Cid="5002">
                                  <name>Jim Noodle</name>
                                  <addr country="Canada">
                                     <street>25 EastCreek</street>
                                     <city>Markham</city>
                                     <prov-state>Ontario</prov-state>
                                     <pcode-zip>N9C-3T6</pcode-zip>
                                  </addr>
                                  <phone type="work">905-566-7258</phone>
                                </customerinfo>' PRESERVE WHITESPACE),?)
```

この SQL プロシージャーを作成して SAMPLE データベースに対して実行すると、2 人の顧客の XML データが戻ります。

パラメーター・マーカーは XML 値をサポートしないので、この制限の回避策は 1 つ以上のローカル変数の値が組み込まれた連結ステートメント・フラグメントから動的 SQL ステートメントを構成することです。

例:

```
DECLARE person_name VARCHAR(128);

SET person_name = "Joe";
SET stmt_text = 'XQUERY for $fname in db2-fn:sqlquery
   ("SELECT doc
      FROM T1
      WHERE DOCID=1")//fullname where $fname/first = ''' person_name || ''';
```

この例では、SQL 全選択を組み込んだ XQuery ステートメントの変数割り当てに結果セットを戻します。結果セットには、ファーストネームが Joe という人の氏名が含まれます。機能的にいえば、SQL の部分は、表 T1 の列 doc から ID 1 の XML 文書を選択します。そして XQuery の部分は、値 first が Joe となっている XML 文書の fullname の値を選択します。

SQL プロシージャー内の XML パラメーターおよび変数値に対するコミットおよびロールバックの効果

SQL プロシージャー内のコミットおよびロールバックは、パラメーターの値とデータ・タイプ XML の変数に影響を及ぼします。SQL プロシージャーの実行では、コミットまたはロールバックの操作と同時に、XML パラメーターおよび XML 変数に割り当てられていた値は以後無効になります。

コミットまたはロールバック操作後に、データ・タイプ XML の SQL 変数または SQL パラメーターを参照しようとすると、エラー (SQL1354N、560CE) が生じる原因になります。

コミットまたはロールバック操作が生じた後に、XML パラメーターおよび変数を正常に参照するには、最初に新しい値を割り当てる必要があります。

SQL プロシージャーに ROLLBACK および COMMIT ステートメントを追加するときは、XML パラメーターおよび変数値の使用の可能性をご検討ください。

SQL 関数

SQL 関数でのデータ・タイプ XML のパラメーターおよび変数

DB2 データベース・システムは、CREATE FUNCTION (SQL スカラー、表、行) ステートメントまたは CREATE FUNCTION (ソース派生またはテンプレート) ステートメントを使用して作成するインライン SQL 関数での XML データ・タイプをサポートしています。

CREATE FUNCTION (SQL スカラー、表、行) ステートメントを使用して作成したインラインのユーザー定義関数では、他のデータ・タイプの変数と同様に、SQL ステートメントで XML 変数を使用できます。例えばユーザー定義関数では、XMLEXISTS 述部や XMLQUERY、XMLTABLE などの関数にある XQuery 式に、XML データ・タイプの変数をパラメーターとして渡すことができます。

CREATE FUNCTION (ソースまたはテンプレート) ステートメントによって作成した、ソース関数がユーザー定義の SQL スカラー関数であるユーザー定義関数では、XML データ・タイプを入力パラメーター、出力パラメーター、または入出力パラメーターとして使用することができます。

XML 値は、ユーザー定義関数内の参照によって割り当てられます。

データ・タイプ XML のパラメーターと変数は、コンパイルされた SQL 関数ではサポートされていません。

例

以下の関数の例は、XML データ・タイプを入力パラメーターおよび変数として使用したインライン SQL スカラー関数を示しています。この関数は、XQuery 式を使用して XML 文書から電話番号要素を抽出し、戻します。

```
CREATE FUNCTION phone_number ( dept_doc XML )
RETURNS XML
LANGUAGE SQL
NO EXTERNAL ACTION
BEGIN ATOMIC
DECLARE tmp_xml XML;
 IF (XMLEXISTS('$test/department/phone' passing by ref dept_doc as "test"))
 THEN
  SET tmp_xml = XMLQUERY('document
       {<phone_list>{$doc/department/phone}</phone_list>}'
    PASSING dept_doc as "doc");
 ELSE
 SET tmp_xml = XMLPARSE(document '<phone_list><phone>N/A</phone></phone_list>');
END IF;
RETURN tmp_xml;
END
```

以下の SELECT ステートメントでは、PHONE_NUMBER 関数を使用して、従業員情報が記載された表の XML 文書から電話番号を取得しています。

```
SELECT PHONE_NUMBER(info) FROM employees WHERE empid = 12356
```

SELECT ステートメントでは、この表が以下の CREATE TABLE ステートメントで作成された表に類似しており、さらに以下の INSERT ステートメントによって挿入された情報に類似するデータを含んでいるものとみなします。

```
CREATE TABLE employees (empid BIGINT, info XML )
```

```
INSERT INTO EMPLOYEES VALUES ( 12356, '
    <department id="marketing">
        <empid>12356</empid>
        <phone>555-123-4567</phone>
     </department> ')
```

SELECT ステートメントは上記の表と情報を使用して、以下の電話番号情報を戻します。

```
<phone_list><phone>555-123-4567</phone></phone_list>
```

インライン化された SQL 関数とコンパイルされた SQL 関数

SQL 関数のインプリメンテーションのタイプには、インライン化された SQL 関数とコンパイル済み SQL 関数の 2 つがあります。

インライン化された SQL 関数

インライン化された SQL 関数とは、RETURN ステートメントまたはインライン・コンパウンド・ステートメントである本体を持つ、CREATE FUNCTION ステートメントを使用して作成される SQL 関数です。インライン・コンパウンド・ステートメントは、BEGIN ATOMIC キーワードと END キーワードを使用して定義します。

インライン化された SQL 関数には、SQL ステートメント、およびインライン SQL PL ステートメント (SQL PL ステートメントのサブセット) を含めることができます。

コンパイル済み SQL 関数

コンパイル済み SQL 関数とは、RETURN ステートメントまたはコンパイル済みコンパウンド・ステートメントである本体を持つ、CREATE FUNCTION ステートメントを使用して作成される SQL 関数です。コンパイル済みコンパウンド・ステートメントは、BEGIN キーワードと END キーワードを使用して定義します。

ATOMIC 節を省略すると、SQL 関数がコンパイルされるので、インライン化された SQL 関数に比べて、より多くの SQL PL フィーチャーを組み込んだり参照したりできます。コンパイル済み SQL 関数には、インライン化された SQL 関数ではサポートされていない以下のフィーチャーがあります。

- 以下のような SQL PL ステートメント
  - CASE ステートメント
  - REPEAT ステートメント
- カーソルの処理
- 動的 SQL
- 条件ハンドラー

外部ルーチン

外部ルーチンでの XML データ・タイプのサポート

下記のプログラミング言語で書かれている外部プロシージャーおよび関数は、データ・タイプ XML のパラメーターおよび変数をサポートします。

- C

- C++
- COBOL
- Java
- .NET CLR 言語

OLE および OLEDB 外部ルーチンは、データ・タイプ XML のパラメーターをサポートしません。

XML データ・タイプの値は、CLOB データ・タイプと同じ方法で外部ルーチンのコード中に示されます。

データ・タイプ XML の外部ルーチン・パラメーターを宣言するときは、データベース内でそのルーチンを作成するときに使用する CREATE PROCEDURE および CREATE FUNCTION ステートメントで、XML データ・タイプを CLOB データ・タイプとして保管することを指定する必要があります。 CLOB 値のサイズは、XML パラメーターで表される XML 文書のサイズに近くなければなりません。

次の CREATE PROCEDURE ステートメントは、parm1 という XML パラメーターを使用して C プログラミング言語でインプリメントされた外部プロシージャーの CREATE PROCEDURE ステートメントを示しています。

```
CREATE PROCEDURE myproc(IN parm1 XML AS CLOB(2M), IN parm2 VARCHAR(32000))
LANGUAGE C
FENCED
PARAMETER STYLE SQL
EXTERNAL NAME 'mylib!myproc';
```

次の例に示されているような外部 UDF の作成時にも、それに似た考慮事項が当てはまります。

```
CREATE FUNCTION myfunc (IN parm1 XML AS CLOB(2M))
RETURNS SMALLINT
LANGUAGE C
PARAMETER STYLE SQL
DETERMINISTIC
NOT FENCED
NULL CALL
NO SQL
NO EXTERNAL ACTION
EXTERNAL NAME 'mylib1!myfunc'
```

XML データは、ストアード・プロシージャーに IN、OUT、または INOUT パラメーターとして渡されるときにマテリアライズされます。Java ストアード・プロシージャーを使用している場合、XML 引数の数量とサイズ、および並行に実行されている外部ストアード・プロシージャーの数に基づいて、ヒープ・サイズ (**java_heap_sz** 構成パラメーター) を増やすことが必要になる場合があります。

外部ルーチン・コード内部では、XML パラメーターおよび変数値へのアクセス、その設定、および変更は、データベース・アプリケーションの場合と同じやり方で行われます。

Java ルーチン用のドライバーの指定

Java ルーチンの開発および呼び出しには、JDBC または SQLJ ドライバーを指定する必要があります。

Java ルーチンは IBM Data Server Driver for JDBC and SQLJ バージョン 4.0 を使用します。

IBM Data Server Driver for JDBC and SQLJ バージョン 4.0 db2jcc4.jar には、JDBC バージョン 4.0 のさまざまな機能が含まれています。ドライバーは、DB2 バージョン 9.5 以上でサポートされています。

デフォルトでは、DB2 データベース・システムは、IBM Data Server Driver for JDBC and SQLJ を使用します。このドライバーは、Java ルーチンに以下のものが含まれる場合には、前提条件になります。

- データ・タイプ XML のパラメーター
- データ・タイプ XML の変数
- XML データへの参照
- XML 関数への参照
- 他の任意のネイティブ XML フィーチャー

例: Java (JDBC) プロシージャーでの XML および XQuery サポート

Java プロシージャーの基本、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) を使用した Java でのプログラミング、および XQuery を理解したなら、XML データを照会する Java プロシージャーの作成および使用を始めることができます。

ここでの Java プロシージャーの例では、以下について示します。

- パラメーター・スタイル JAVA プロシージャーの CREATE PROCEDURE ステートメント
- パラメーター・スタイル JAVA プロシージャーのソース・コード
- データ・タイプ XML の入出力パラメーター
- 照会での XML 入力パラメーターの使用
- XQuery の結果、XML 値の出力パラメーターへの割り当て
- SQL ステートメントの結果、XML 値の出力パラメーターへの割り当て

前提条件

Java プロシージャーの例を使用した作業を開始する前に、以下のトピックを参照することもできます。

- Java ルーチン
- ルーチン
- Java ルーチン・コードのビルド

次の例では、xmlDataTable という名前の表を使用します。その定義および含まれているデータは以下のとおりです。

```
CREATE TABLE xmlDataTable
(
   num INTEGER,
   xdata XML
)@

INSERT INTO xmlDataTable VALUES
   (1, XMLPARSE(DOCUMENT '<doc>
```

```
                              <type>car</type>
                              <make>Pontiac</make>
                              <model>Sunfire</model>
                              </doc>' PRESERVE WHITESPACE)),
(2, XMLPARSE(DOCUMENT '<doc>
                              <type>car</type>
                              <make>Mazda</make>
                              <model>Miata</model>
                              </doc>' PRESERVE WHITESPACE)),
(3, XMLPARSE(DOCUMENT '<doc>
                              <type>person</type>
                              <name>Mary</name>
                              <town>Vancouver</town>
                              <street>Waterside</street>
                              </doc>' PRESERVE WHITESPACE)),
(4, XMLPARSE(DOCUMENT '<doc>
                              <type>person</type>
                              <name>Mark</name>
                              <town>Edmonton</town>
                              <street>Oak</street>
                              </doc>' PRESERVE WHITESPACE)),
(5, XMLPARSE(DOCUMENT '<doc>
                              <type>animal</type>
                              <name>dog</name>
                              </doc>' PRESERVE WHITESPACE)),
(6, NULL),
(7, XMLPARSE(DOCUMENT '<doc>
                              <type>car</type>
                              <make>Ford</make>
                              <model>Taurus</model>
                              </doc>' PRESERVE WHITESPACE)),
(8, XMLPARSE(DOCUMENT '<doc>
                              <type>person</type>
                              <name>Kim</name>
                              <town>Toronto</town>
                              <street>Elm</street>
                              </doc>' PRESERVE WHITESPACE)),
(9, XMLPARSE(DOCUMENT '<doc>
                              <type>person</type>
                              <name>Bob</name>
                              <town>Toronto</town>
                              <street>Oak</street>
                              </doc>' PRESERVE WHITESPACE)),
(10, XMLPARSE(DOCUMENT '<doc>
                               <type>animal</type>
                               <name>bird</name>
                               </doc>' PRESERVE WHITESPACE))@
```

手順 独自の Java プロシージャーを作成するときには、以下の例を参考にしてください。

- 『Java 外部コード・ファイル 』
- 293 ページの『例 1: XML パラメーターを使用するパラメーター・スタイル JAVA プロシージャー』

Java 外部コード・ファイル

例では、Java プロシージャー・インプリメンテーションを示します。例は、CREATE PROCEDURE ステートメントと、関連 Java クラスのビルド元プロシージャーの外部 Java コード・インプリメンテーションという 2 つの部分から成っています。

以下の例のプロシージャー・インプリメンテーションに含まれる Java ソース・ファイルは、stpclass.java という名前で、myJAR という名前の JAR ファイルに組み込まれています。ファイルの形式は以下のとおりです。

```
using System;
import java.lang.*;
import java.io.*;
import java.sql.*;
import java.util.*;
import com.ibm.db2.jcc.DB2Xml;

 public class stpclass
 {          ...
    // Java procedure implementations
            ...
 }
```

ファイルに先頭では、Java クラス・ファイルの import が示されています。ファイル内の、タイプ XML のパラメーターまたは変数を含むプロシージャーが使用される場合は、 com.ibm.db2.jcc.DB2Xml import が必要です。

クラス・ファイルの名前、および特定のプロシージャー・インプリメンテーションを含む JAR 名をメモしておくことは重要です。各プロシージャーの CREATE PROCEDURE ステートメントの EXTERNAL 節でその情報を指定して、DB2 データベース・システムが実行時にそのクラスを見つけられるようにする必要があるからです。

例 1: XML パラメーターを使用するパラメーター・スタイル JAVA プロシージャー

この例では、以下について説明します。

- パラメーター・スタイル JAVA のプロシージャーの CREATE PROCEDURE ステートメント
- XML パラメーターを使用するパラメーター・スタイル JAVA プロシージャーの Java コード

このプロシージャーは入力パラメーター、inXML を取り、その値を含む行を表に挿入し、SQL ステートメントと XQuery 式の両方を使用して XML データを照会して、2 つの出力パラメーター、outXML1 と outXML2 を設定します。

```
CREATE PROCEDURE  xmlProc1   ( IN inNUM  INTEGER,
                               IN inXML  XML as CLOB (1K),
                               OUT out1XML XML as CLOB (1K),
                               OUT out2XML XML as CLOB (1K)
                                  )
DYNAMIC RESULT SETS 0
DETERMINISTIC
LANGUAGE JAVA
PARAMETER STYLE JAVA
MODIFIES SQL DATA
FENCED
THREADSAFE
DYNAMIC RESULT SETS 0
PROGRAM TYPE SUB
NO DBINFO
EXTERNAL NAME 'myJar:stpclass.xmlProc1'@

//*************************************************************************
//  Stored Procedure: XMLPROC1
//
```

```
//  Purpose:   Inserts XML data into XML column; queries and returns XML data
//
//  Parameters:
//
//   IN:      inNum -- the sequence of XML data to be insert in xmldata table
//            inXML -- XML data to be inserted
//   OUT:     out1XML -- XML data to be returned
//            out2XML -- XML data to be returned
//
//*************************************************************************

public void xmlProc1(int inNum,
                     DB2Xml inXML ,
                     DB2Xml[] out1XML,
                     DB2Xml[] out2XML
                    )
throws Exception
{
       Connection con = DriverManager.getConnection("jdbc:default:connection");

       // Insert data including the XML parameter value into a table
       String query = "INSERT INTO xmlDataTable (num, inXML ) VALUES ( ?, ? )" ;
       String xmlString = inXML.getDB2String() ;

       stmt = con.prepareStatement(query);
       stmt.setInt(1, inNum);
       stmt.setString (2, xmlString );
       stmt.executeUpdate();
       stmt.close();

       // Query and retrieve a single XML value from a table using SQL
       query = "SELECT xdata from xmlDataTable WHERE num = ? " ;

       stmt = con.prepareStatement(query);
       stmt.setInt(1, inNum);
       ResultSet rs = stmt.executeQuery();

       if ( rs.next() )
       {  out1Xml[0] = (DB2Xml) rs.getObject(1);  }

       rs.close() ;
       stmt.close();

       // Query and retrieve a single XML value from a table using XQuery
       query = "XQUERY for $x in db2-fn:xmlcolumn(¥"xmlDataTable.xdata¥")/doc
                              where $x/make = ¥'Mazda¥'
                              return <carInfo>{$x/make}{$x/model}</carInfo>";

       stmt = con.createStatement();

       rs = stmt.executeQuery( query );

       if ( rs.next() )
       {  out2Xml[0] = (DB2Xml) rs.getObject(1) ;  }

       rs.close();
       stmt.close();
       con.close();

       return ;
}
```

例: C# .NET CLR プロシージャーでの XML および XQuery サポート

プロシージャーの基本、.NET 共通言語ランタイム・ルーチンの本質部分、XQuery および XML を理解したなら、XML フィーチャーを持つ CLR プロシージャーの作成および使用を始めることができます。

次の例は、XML データの更新および照会方法に加えて、タイプ XML のパラメーターを使用する C# .NET CLR プロシージャーを示します。

前提条件

CLR プロシージャーの例を使用した作業を開始する前に、概念について説明している以下のトピックを参照することもできます。

- .NET 共通言語ランタイム (CLR) ルーチン
- DB2 コマンド・ウィンドウから .NET CLR ルーチンを作成する
- ルーチン使用の利点

次の例では、以下のように定義された xmlDataTable という名前の表を使用します。

```
CREATE TABLE xmlDataTable
(
   num INTEGER,
   xdata XML
)

INSERT INTO xmlDataTable VALUES
   (1, XMLPARSE(DOCUMENT '<doc>
                              <type>car</type>
                              <make>Pontiac</make>
                              <model>Sunfire</model>
                              </doc>' PRESERVE WHITESPACE)),
   (2, XMLPARSE(DOCUMENT '<doc>
                              <type>car</type>
                              <make>Mazda</make>
                              <model>Miata</model>
                              </doc>' PRESERVE WHITESPACE)),
   (3, XMLPARSE(DOCUMENT '<doc>
                              <type>person</type>
                              <name>Mary</name>
                              <town>Vancouver</town>
                              <street>Waterside</street>
                              </doc>' PRESERVE WHITESPACE)),
   (4, XMLPARSE(DOCUMENT '<doc>
                              <type>person</type>
                              <name>Mark</name>
                              <town>Edmonton</town>
                              <street>Oak</street>
                              </doc>' PRESERVE WHITESPACE)),
   (5, XMLPARSE(DOCUMENT '<doc>
                              <type>animal</type>
                              <name>dog</name>
                              </doc>' PRESERVE WHITESPACE)),
   (6, NULL),
   (7, XMLPARSE(DOCUMENT '<doc>
                              <type>car</type>
                              <make>Ford</make>
                              <model>Taurus</model>
                              </doc>' PRESERVE WHITESPACE)),
   (8, XMLPARSE(DOCUMENT '<doc>
                              <type>person</type>
                              <name>Kim</name>
```

```
                                <town>Toronto</town>
                                <street>Elm</street>
                                </doc>' PRESERVE WHITESPACE)),
        (9, XMLPARSE(DOCUMENT '<doc>
                                <type>person</type>
                                <name>Bob</name>
                                <town>Toronto</town>
                                <street>Oak</street>
                                </doc>' PRESERVE WHITESPACE)),
        (10, XMLPARSE(DOCUMENT '<doc>
                                 <type>animal</type>
                                 <name>bird</name>
                                 </doc>' PRESERVE WHITESPACE))@
```

手順 独自の C# CLR プロシージャーを作成するときには、以下の例を参考にしてください。

- 『C# 外部コード・ファイル』
- 297 ページの『例 1: XML フィーチャーを持つ C# パラメーター・スタイル GENERAL プロシージャー』

C# 外部コード・ファイル

例は、CREATE PROCEDURE ステートメントと、関連アセンブリーのビルド元プロシージャーの外部 C# コード・インプリメンテーションという 2 つの部分から成っています。

以下の例のプロシージャー・インプリメンテーションに含まれる C# ソース・ファイルは、 gwenProc.cs という名前であり、以下の形式になっています。

```
using System;
using System.IO;
using System.Data;
using IBM.Data.DB2;
using IBM.Data.DB2Types;

namespace bizLogic
{
   class empOps
   {          ...
     // C# procedures
              ...
   }
}
```

ファイルの先頭には、このファイルに組み込むものを示します。ファイル内のプロシージャーのいずれかに SQL が含まれる場合は、 IBM.Data.DB2 を含める必要があります。ファイル内のプロシージャーのいずれかにタイプ XML のパラメーターまたは変数が含まれる場合は、IBM.Data.DB2Types を含める必要があります。このファイルには、ネーム・スペース宣言を組み込み、プロシージャーを内容とするクラス empOps を組み込みます。ネーム・スペースの使用はオプションです。ネーム・スペースを使用する場合は、 CREATE PROCEDURE ステートメントの EXTERNAL 節に指定するアセンブリー・パス名の中にネーム・スペースを入れなければなりません。

ファイルの名前、ネームスペース、特定のプロシージャー・インプリメンテーションを含むクラスの名前をメモしておくことは重要です。各プロシージャーの CREATE PROCEDURE ステートメントの EXTERNAL 節でその情報を指定して、DB2 データベース・システムがアセンブリーと CLR プロシージャーのクラスを見

つけられるようにする必要があるからです。

例 1: XML フィーチャーを持つ C# パラメーター・スタイル GENERAL プロシージャー

この例では、以下について説明します。

- パラメーター・スタイル GENERAL のプロシージャーの CREATE PROCEDURE ステートメント
- XML パラメーターを使用するパラメーター・スタイル GENERAL プロシージャーの C# コード

このプロシージャーは、整数 inNum と inXML という 2 つのパラメーターを取ります。これらの値は表 xmlDataTable に挿入されます。次に、XML 値が XQuery を使用して検索されます。もう 1 つの XML 値が SQL を使用して検索されます。検索された XML 値は 2 つの出力パラメーター、outXML1 と outXML2 に割り当てられます。結果セットは戻されません。

```
CREATE PROCEDURE  xmlProc1   ( IN inNUM  INTEGER,
                               IN inXML  XML as CLOB (1K),
                               OUT inXML  XML as CLOB (1K),
                               OUT inXML  XML as CLOB (1K)
                             )
LANGUAGE CLR
PARAMETER STYLE GENERAL
DYNAMIC RESULT SETS 0
FENCED
THREADSAFE
DETERMINISTIC
NO DBINFO
MODIFIES SQL DATA
PROGRAM TYPE SUB
EXTERNAL NAME 'gwenProc.dll:bizLogic.empOps!xmlProc1' ;

//*************************************************************************
//  Stored Procedure: xmlProc1
//
//  Purpose:   insert XML data into XML column
//
//  Parameters:
//
//   IN:    inNum -- the sequence of XML data to be insert in xmldata table
//          inXML -- XML data to be inserted
//   OUT:   outXML1 -- XML data returned - value retrieved using XQuery
//          outXML2 -- XML data returned - value retrieved using SQL
//*************************************************************************

    public static void xmlProc1 (     int      inNum, DB2Xml   inXML,
                                  out DB2Xml   outXML1, out DB2Xml   outXML2 )
    {
       // Create new command object from connection context
       DB2Parameter parm;
       DB2Command cmd;
       DB2DataReader reader = null;
       outXML1 = DB2Xml.Null;
       outXML2 = DB2Xml.Null;

       // Insert input XML parameter value into a table
       cmd = DB2Context.GetCommand();
       cmd.CommandText = "INSERT INTO "
                       + "xmlDataTable( num , xdata ) "
                       + "VALUES( ?, ? )";

       parm = cmd.Parameters.Add("@num", DB2Type.Integer );
```

```
    parm.Direction = ParameterDirection.Input;
    cmd.Parameters["@num"].Value = inNum;
    parm = cmd.Parameters.Add("@data", DB2Type.Xml);
    parm.Direction = ParameterDirection.Input;
    cmd.Parameters["@data"].Value = inXML ;
    cmd.ExecuteNonQuery();
    cmd.Close();

    // Retrieve XML value using XQuery
                  and assign value to an XML output parameter
    cmd = DB2Context.GetCommand();
    cmd.CommandText = "XQUERY for $x " +
                      "in db2-fn:xmlcolumn(¥"xmlDataTable.xdata¥")/doc "+
                      "where $x/make = ¥'Mazda¥' " +
                      "return <carInfo>{$x/make}{$x/model}</carInfo>";
    reader = cmd.ExecuteReader();
    reader.CacheData= true;

    if (reader.Read())
    {  outXML1 = reader.GetDB2Xml(0);  }
    else
    {  outXML1 = DB2Xml.Null;  }

    reader.Close();
    cmd.Close();

    // Retrieve XML value using SQL
                  and assign value to an XML output parameter value
    cmd = DB2Context.GetCommand();
    cmd.CommandText = "SELECT xdata "
                       + "FROM xmlDataTable "
                       + "WHERE num = ?";

    parm = cmd.Parameters.Add("@num", DB2Type.Integer );
    parm.Direction = ParameterDirection.Input;
    cmd.Parameters["@num"].Value = inNum;
    reader = cmd.ExecuteReader();
    reader.CacheData= true;

    if (reader.Read())
    {  outXML2 = reader.GetDB2Xml(0);  }
    else
    {  outXML = DB2Xml.Null;  }

    reader.Close() ;
    cmd.Close();

    return;
}
```

例: C プロシージャーでの XML および XQuery サポート

プロシージャーの基本、C ルーチンの本質部分、XQuery および XML を理解したなら、XML 機能を持つ C プロシージャーの作成および使用を始めることができます。

次の例は、XML データの更新および照会方法に加えて、タイプ XML のパラメーターを使用する C プロシージャーを示します。

前提条件

C プロシージャーの例を使用した作業を開始する前に、概念について説明している以下のトピックを参照することもできます。

- ルーチン使用の利点

次の例では、以下のように定義された xmlDataTable という名前の表を使用します。

```
CREATE TABLE xmlDataTable
(
   num INTEGER,
   xdata XML
)

INSERT INTO xmlDataTable VALUES
   (1, XMLPARSE(DOCUMENT '<doc>
                              <type>car</type>
                              <make>Pontiac</make>
                              <model>Sunfire</model>
                              </doc>' PRESERVE WHITESPACE)),
   (2, XMLPARSE(DOCUMENT '<doc>
                              <type>car</type>
                              <make>Mazda</make>
                              <model>Miata</model>
                              </doc>' PRESERVE WHITESPACE)),
   (3, XMLPARSE(DOCUMENT '<doc>
                              <type>person</type>
                              <name>Mary</name>
                              <town>Vancouver</town>
                              <street>Waterside</street>
                              </doc>' PRESERVE WHITESPACE)),
   (4, XMLPARSE(DOCUMENT '<doc>
                              <type>person</type>
                              <name>Mark</name>
                              <town>Edmonton</town>
                              <street>Oak</street>
                              </doc>' PRESERVE WHITESPACE)),
   (5, XMLPARSE(DOCUMENT '<doc>
                              <type>animal</type>
                              <name>dog</name>
                              </doc>' PRESERVE WHITESPACE)),
   (6, NULL),
   (7, XMLPARSE(DOCUMENT '<doc>
                              <type>car</type>
                              <make>Ford</make>
                              <model>Taurus</model>
                              </doc>' PRESERVE WHITESPACE)),
   (8, XMLPARSE(DOCUMENT '<doc>
                              <type>person</type>
                              <name>Kim</name>
                              <town>Toronto</town>
                              <street>Elm</street>
                              </doc>' PRESERVE WHITESPACE)),
   (9, XMLPARSE(DOCUMENT '<doc>
                              <type>person</type>
                              <name>Bob</name>
                              <town>Toronto</town>
                              <street>Oak</street>
                              </doc>' PRESERVE WHITESPACE)),
   (10, XMLPARSE(DOCUMENT '<doc>
                               <type>animal</type>
                               <name>bird</name>
                               </doc>' PRESERVE WHITESPACE))
```

手順 独自の C プロシージャーを作成するときには、以下の例を参考にしてください。

- 300 ページの『C 外部コード・ファイル』
- 300 ページの『例 1: XML フィーチャーを持つ C パラメーター・スタイル SQL プロシージャー』

C 外部コード・ファイル

例は、CREATE PROCEDURE ステートメントと、関連アセンブリーのビルド元プロシージャーの外部 C コード・インプリメンテーションという 2 つの部分から成っています。

以下の例のプロシージャー・インプリメンテーションに含まれる C ソース・ファイルは、 gwenProc.SQC という名前であり、以下の形式になっています。

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <sqlda.h>
#include <sqlca.h>
#include <sqludf.h>
#include <sql.h>
#include <memory.h>

// C procedures
               ...
```

ファイルの先頭には、このファイルに組み込むものを示します。組み込み SQL ルーチンには、XML サポートに必要な余分の組み込みファイルはありません。

ファイルの名前、およびプロシージャー・インプリメンテーションに対応する関数の名前をメモしておくことは重要です。各プロシージャーの CREATE PROCEDURE ステートメントの EXTERNAL 節でその情報を指定して、DB2 データベース・マネージャーがその C プロシージャーに該当するライブラリーとエントリー・ポイントを見つけられるようにする必要があるからです。

例 1: XML フィーチャーを持つ C パラメーター・スタイル SQL プロシージャー

この例では、以下について説明します。

- パラメーター・スタイル SQL のプロシージャーの CREATE PROCEDURE ステートメント
- XML パラメーターを使用するパラメーター・スタイル SQL プロシージャーの C コード

このプロシージャーは 2 つの入力パラメーターを取ります。最初の入力パラメーターの名前は inNum で、タイプは INTEGER です。 2 番目の入力パラメーターの名前は inXML で、タイプは XML です。入力パラメーターの値を使用して、行を表 xmlDataTable に挿入します。次に、XML 値が SQL ステートメントを使用して検索されます。もう 1 つの XML 値が XQuery 式を使用して検索されます。検索された XML 値はそれぞれ 2 つの出力パラメーター、out1XML と out2XML に割り当てられます。結果セットは戻されません。

```
CREATE PROCEDURE  xmlProc1   ( IN inNUM  INTEGER,
                               IN inXML  XML as CLOB (1K),
                               OUT inXML  XML as CLOB (1K),
                               OUT inXML  XML as CLOB (1K)
                             )
LANGUAGE C
PARAMETER STYLE SQL
DYNAMIC RESULT SETS 0
FENCED
```

```
  THREADSAFE
  DETERMINISTIC
  NO DBINFO
  MODIFIES SQL DATA
  PROGRAM TYPE SUB
  EXTERNAL NAME 'gwenProc!xmlProc1' ;

  //*************************************************************************
  //  Stored Procedure: xmlProc1
  //
  //  Purpose:   insert XML data into XML column
  //
  //  Parameters:
  //
  //   IN:    inNum -- the sequence of XML data to be insert in xmldata table
  //          inXML -- XML data to be inserted
  //   OUT:   out1XML -- XML data returned - value retrieved using XQuery
  //          out2XML -- XML data returned - value retrieved using SQL
  //*************************************************************************

#ifdef __cplusplus
extern "C"
#endif
SQL_API_RC SQL_API_FN testSecA1(sqlint32* inNum,
                                SQLUDF_CLOB* inXML,
                                SQLUDF_CLOB* out1XML,
                                SQLUDF_CLOB* out2XML,
                                SQLUDF_NULLIND *inNum_ind,
                                SQLUDF_NULLIND *inXML_ind,
                                SQLUDF_NULLIND *out1XML_ind,
                                SQLUDF_NULLIND *out2XML_ind,
                                SQLUDF_TRAIL_ARGS)
{
  char *str;
  FILE *file;

  EXEC SQL INCLUDE SQLCA;

  EXEC SQL BEGIN DECLARE SECTION;
       sqlint32 hvNum1;
       SQL TYPE IS XML AS CLOB(200) hvXML1;
       SQL TYPE IS XML AS CLOB(200) hvXML2;
       SQL TYPE IS XML AS CLOB(200) hvXML3;
  EXEC SQL END DECLARE SECTION;

  /* Check null indicators for input parameters */
  if ((*inNum_ind < 0) ||  (*inXML_ind < 0)) {
     strcpy(sqludf_sqlstate, "38100");
     strcpy(sqludf_msgtext, "Received null input");
     return 0;
  }

  /* Copy input parameters to host variables */
  hvNum1 = *inNum;
  hvXML1.length = inXML->length;
  strncpy(hvXML1.data, inXML->data, inXML->length);

  /* Execute SQL statement */
  EXEC SQL
     INSERT INTO xmlDataTable (num, xdata) VALUES (:hvNum1, :hvXML1);

  /* Execute SQL statement */
  EXEC SQL
     SELECT xdata INTO :hvXML2
        FROM xmlDataTable
           WHERE num = :hvNum1;

  sprintf(stmt5, "SELECT XMLQUERY('for $x in $xmldata/doc
```

```
                                        return <carInfo>{$x/model}</carInfo>'
                                        passing by ref xmlDataTable.xdata
                                            as ¥"xmldata¥" returning sequence)
                        FROM xmlDataTable WHERE num = ?");

  EXEC SQL PREPARE selstmt5 FROM :stmt5 ;
  EXEC SQL DECLARE c5 CURSOR FOR selstmt5;
  EXEC SQL OPEN c5 using :hvNum1;
  EXEC SQL FETCH c5 INTO :hvXML3;

  exit:

  /* Set output return code */
  *outReturnCode = sqlca.sqlcode;
  *outReturnCode_ind = 0;

  return 0;
}
```

ルーチンのパフォーマンス

ルーチンのパフォーマンスは様々な要因による影響を受けます。これには、ルーチンのタイプおよびインプリメンテーション、ルーチン内の SQL ステートメントの数、ルーチン内の SQL の複雑さの度合い、ルーチンに対するパラメーターの数、ルーチンのインプリメンテーションにおけるロジックの有効性、ルーチン内のエラー処理などが含まれます。

ユーザーはしばしば、アプリケーションのパフォーマンスを向上させるためにルーチンをインプリメントすることを選ぶため、ルーチンのパフォーマンスを最大限に活用することは重要です。

以下の表は、ルーチンのパフォーマンスに影響を与える一般的な要因のいくつかの概略を示し、それぞれの要因を変更することによってルーチンのパフォーマンスを向上させる方法についての推奨事項を示しています。特定のルーチン・タイプに影響を与えるパフォーマンス要因の詳細については、その特定のルーチン・タイプのパフォーマンスおよびチューニングのトピックを参照してください。

表 33. パフォーマンスの考慮事項およびルーチンのパフォーマンスの推奨事項

パフォーマンスの考慮事項	パフォーマンスの推奨事項
ルーチン・タイプ: プロシージャー、関数、メソッド	• プロシージャー、関数、およびメソッドはさまざまな目的で使用され、さまざまな場所で参照されます。それらには機能上の違いがあるため、パフォーマンスを直接比較することは困難です。 • 一般に、プロシージャーは関数として再作成できる場合があり (特に、プロシージャーがスカラー値を戻す場合と照会データのみを戻す場合)、それによってパフォーマンスがわずかに向上することがあります。しかし、それらの利点は一般に SQL ロジックをインプリメントするために必要な SQL を単純化することによって得られます。 • 複雑な初期化を伴うユーザー定義関数では、スクラッチパッドを利用して、最初の呼び出し時に必要とされる値を保管できます。そうすれば、それらの値は以後の呼び出しで使用することができます。

表 33. パフォーマンスの考慮事項およびルーチンのパフォーマンスの推奨事項 (続き)

パフォーマンスの考慮事項	パフォーマンスの推奨事項
ルーチンのインプリメンテーション: 組み込みまたはユーザー定義	• 同等のロジックの場合、組み込みルーチンがパフォーマンスに最も優れており、その次に優れているのは組み込みルーチンです。それは、それらのルーチンはユーザー定義ルーチンよりもデータベース・エンジンと密接な関係を保つからです。 • ユーザー定義ルーチンは、適切にコード化されており、ベスト・プラクティスに従っているのであれば、良いパフォーマンスで実行できます。
ルーチンのインプリメンテーション: SQL または外部ルーチンのインプリメンテーション	• SQL ルーチンは外部ルーチンよりも効率的です。なぜなら、SQL ルーチンは DB2 データベース・サーバーによって直接実行されるからです。 • SQL プロシージャーは一般に、論理的に同等の外部プロシージャーよりもパフォーマンスに優れています。 • 単純なロジックの場合、SQL 関数のパフォーマンスは、同等の外部関数のパフォーマンスと同程度になります。 • 数学アルゴリズムおよびストリング処理関数などの SQL をほとんど必要としない複合ロジックの場合、C などの低レベルのプログラミング言語で記述した外部ルーチンを使用する方が適切です。なぜなら SQL サポートへの依存が少ないからです。 • パフォーマンスをはじめ、サポートされる外部ルーチン・プログラミング言語オプションのフィーチャーの比較については、『ルーチン・インプリメンテーションの比較』を参照してください。

表 33. パフォーマンスの考慮事項およびルーチンのパフォーマンスの推奨事項 (続き)

パフォーマンスの考慮事項	パフォーマンスの推奨事項
外部ルーチンのインプリメンテーションのプログラミング言語	• 外部ルーチンのインプリメンテーションを選択する際に考慮すべきパフォーマンス・フィーチャーの比較については、『外部ルーチン API とプログラミング言語の比較』を参照してください。 • Java (JDBC および SQLJ API) – メモリー要件が非常に大きい Java ルーチンを作成する場合は、FENCED NOT THREADSAFE 節を指定するのが最適です。消費するメモリー領域が平均的な Java ルーチンでは、FENCED THREADSAFE 節を指定できます。 – FENCED THREADSAFE Java ルーチンの呼び出しの場合、DB2 データベース・システムは、ルーチンを十分に実行できる大きさの Java ヒープを持つ、スレッド化された Java fenced モードのプロセスを選択しようとします。独自のプロセスで大量のヒープを消費するルーチンを分離できないと、マルチスレッド化された Java **db2fmp** プロセスで Java ヒープ不足エラーが発生する可能性があります。対照的に、FENCED THREADSAFE ルーチンは、少数の JVM を共有できるため、良いパフォーマンスが得られます。 • C および C++ – 一般に、C および C++ ルーチンのパフォーマンスは、その他の外部ルーチンのインプリメンテーションよりも優れており、SQL ルーチンと同程度です。 – 最適な C および C++ ルーチンを実行するには、それらが 32 ビットの DB2 インスタンスにデプロイされる場合には 32 ビット・フォーマットでコンパイルし、64 ビットの DB2 インスタンスにデプロイされる場合には 64 ビット・フォーマットでコンパイルする必要があります。 • COBOL – 一般に、COBOL のパフォーマンスも十分ですが、ルーチンのインプリメンテーションとして COBOL は推奨されていません。

表 33. パフォーマンスの考慮事項およびルーチンのパフォーマンスの推奨事項 (続き)

パフォーマンスの考慮事項	パフォーマンスの推奨事項
ルーチン内の SQL ステートメントの数	• ルーチンには複数の SQL ステートメントが含まれている必要があります。そうしないと、ルーチン呼び出しのオーバーヘッドのパフォーマンス・コスト効率が低くなります。 • 複数のデータベース照会を行い、中間結果を処理し、処理したデータのサブセットを最終的に戻すようなロジックは、ルーチンのカプセル化に最適のロジックです。このタイプのロジックの例としては、複雑なデータ・マイニング、および関連データの検索を必要とする大規模な更新があります。負荷の高い SQL 処理はデータベース・サーバー上で行われ、呼び出し側には少量のデータ結果セットしか戻されません。
ルーチン内の SQL ステートメントの複雑さ	• ルーチンに非常に複雑な照会を組み込んで、データベース・サーバーのメモリーおよびパフォーマンスの機能を十分に生かして活用できるようにするのは、良い方法です。 • SQL ステートメントが複雑すぎることを心配しないでください。
ルーチン内の静的または動的 SQL 実行	• 一般に、静的 SQL のパフォーマンスは動的 SQL よりも優れています。ルーチンでは、静的 SQL または動的 SQL を使用する場合に、それ以上の違いはありません。
ルーチンに対するパラメーターの数	• ルーチンに対するパラメーターの数を最小限にすると、ルーチンとルーチン呼び出し側との間で受け渡されるバッファーの数が最小限になるため、ルーチンのパフォーマンスを向上できます。

表 33. パフォーマンスの考慮事項およびルーチンのパフォーマンスの推奨事項 (続き)

パフォーマンスの考慮事項	パフォーマンスの推奨事項
ルーチン・パラメーターのデータ・タイプ	• ルーチン定義で CHAR パラメーターではなく VARCHAR パラメーターを使用することにより、ルーチンのパフォーマンスを向上させることができます。CHAR データ・タイプではなく VARCHAR データ・タイプを使用すると、パラメーターの引き渡しの前に DB2 データベース・システムによってパラメーターにスペースが埋め込まれなくなります。これで、ネットワークを経由したパラメーターの転送に要する時間が短縮されます。 例えば、クライアント・アプリケーションが CHAR(200) パラメーターを予期するルーチンにストリング "A SHORT STRING" を渡す場合、DB2 データベース・システムはパラメーターに 186 個のスペースを埋め込み、ストリングを NULL で終了してから、200 文字のストリングと NULL 終止符全体をネットワーク経由でルーチンに送信する必要があります。 それと比べて、VARCHAR(200) パラメーターを予期するルーチンに同じストリング *"A SHORT STRING"* を渡すと、DB2 データベース・システムは単に 14 文字ストリングと NULL 終止符をネットワーク経由で渡します。
ルーチンに対するパラメーターの初期化	• ルーチンに対する入力パラメーターを常に初期化することは、特に入力ルーチン・パラメーター値が NULL の場合には適切です。 NULL 値のルーチン・パラメーターの場合、フルサイズのバッファーではなく、小さいバッファーまたは空のバッファーをルーチンに渡すことができます。それにより、パフォーマンスを向上させることができます。
ルーチン内のローカル変数の数	• ルーチン内で宣言されるローカル変数の数を最小限にすることにより、ルーチン内で実行される SQL ステートメントの数を最小限にして、パフォーマンスを向上させることができます。 • 一般に、使用する変数を可能な限り少なくすることを目標としてください。変数を再利用することで混乱が生じない場合は、変数を再利用してください。
ルーチン内のローカル変数の初期化	• 可能であれば、単一の SQL ステートメント内で複数のローカル変数を初期化するのは、良い方法です。そうすれば、ルーチンの合計の SQL 実行時間を節約できます。
プロシージャーが戻す結果セットの数	• ルーチンによって戻される結果セットの数を減らせると、ルーチンのパフォーマンスを向上させることができます。

表 33. パフォーマンスの考慮事項およびルーチンのパフォーマンスの推奨事項 (続き)

パフォーマンスの考慮事項	パフォーマンスの推奨事項
ルーチンによって戻される結果セットのサイズ	• ルーチンによって戻される結果セットごとに、結果を定義する照会によって、戻された列および戻された行数をできるだけフィルタリングするようにしてください。不要なデータの列または行を戻すことは効率的ではなく、ルーチンのパフォーマンスが最適ではなくなる可能性があります。
ルーチン内のロジックの有効性	• アプリケーションと同様に、ルーチンのパフォーマンスも、インプリメントが不十分なアルゴリズムによって制限されることがあります。ルーチンをプログラミングする際にはできる限り効率的であるようにし、一般に推奨されているコーディングのベスト・プラクティスをできる限り適用するようにしてください。 • SQL を分析し、可能な限り照会を単純なフォームにしてください。これは多くの場合、CASE ステートメントの代わりに CASE 式を使用したり、複数の SQL ステートメントを CASE 式をスイッチとして使用する単一のステートメントに縮小したりすることによって行えます。

表 33. パフォーマンスの考慮事項およびルーチンのパフォーマンスの推奨事項 (続き)

パフォーマンスの考慮事項	パフォーマンスの推奨事項
ルーチンのランタイム・モード (FENCED または NOT FENCED 節の指定)	**NOT FENCED 節の使用法:** • 一般に、ルーチンを NOT FENCED 節で作成すること (DB2 データベース・マネージャーと同じプロセスで実行することを指定) は、FENCED 節で作成すること (エンジンのアドレス・スペースの外部の特殊な DB2 データベース・プロセスで実行することを指定) より望ましいと言えます。 • ルーチンを not fenced として実行すると、ルーチンのパフォーマンスの向上を期待できますが、unfenced ルーチンのユーザー・コードが意図せずにまたは故意にデータベースを破損したり、データベース制御構造に損傷を与えることがあります。 NOT FENCED 節を使用するのは、パフォーマンスの利点を最大限にする必要があるとき、およびルーチンが安全であると判断される場合に限らなければなりません。 (C/C++ ルーチンを NOT FENCED として登録するリスクの評価およびその軽減については、『ルーチンのセキュリティー』を参照してください。) ルーチンがデータベース・マネージャーのプロセスで実行できるほど安全でない場合は、ルーチンの作成時に FENCED 節を使用してください。安全でない可能性があるコードの作成および実行を制限するために、DB2 データベース・システムでは、ユーザーが NOT FENCED ルーチンを作成するには、特殊権限 CREATE_NOT_FENCED_ROUTINE を持っていなければなりません。 • NOT FENCED ルーチンの実行中に異常終了が発生する場合、ルーチンが NO SQL として登録されていると、データベース・マネージャーは適切なリカバリーを試行します。しかし、NO SQL として定義されていないルーチンの場合、データベース・マネージャーは失敗します。 • ルーチンが GRAPHIC または DBCLOB データを使用する場合は、NOT FENCED ルーチンを WCHARTYPE NOCONVERT オプションでプリコンパイルする必要があります。

表 33. パフォーマンスの考慮事項およびルーチンのパフォーマンスの推奨事項 (続き)

パフォーマンスの考慮事項	パフォーマンスの推奨事項
ルーチンのランタイム・モード (FENCED または NOT FENCED 節の指定)	**FENCED THREADSAFE 節の使用法** • FENCED THREADSAFE 節で作成されたルーチンは、その他のルーチンと同じプロセスで実行します。具体的には、Java 以外のルーチンはあるプロセスを共有し、Java ルーチンは他の言語で作成されたルーチンとは分離した、別のプロセスを共有します。この分離により、Java ルーチンは、他の言語で作成された、エラーを起こしやすいルーチンから保護されます。また、Java ルーチンのプロセスには JVM が含まれています。これは、メモリー・コストが高くなり、他のルーチン・タイプでは使用されません。 FENCED THREADSAFE ルーチンの複数の呼び出しではリソースを共有するため、それぞれが独自の専用プロセスで実行する FENCED NOT THREADSAFE ルーチンよりもシステムのオーバーヘッドが減ります。 • ご使用のルーチンが他のルーチンと同じプロセスで実行しても安全であると感じる場合、それを登録する際に THREADSAFE 節を使用してください。 NOT FENCED ルーチンと同様に、C/C++ ルーチンを FENCED THREADSAFE として登録するリスクの評価およびその軽減の詳細については、『ルーチンのセキュリティーに関する考慮事項』のトピックを参照してください。 • FENCED THREADSAFE ルーチンが異常終了する場合、このルーチンを実行しているスレッドだけが終了されます。プロセス内のその他のルーチンは実行を続けます。しかし、このスレッドが異常終了する原因になった障害は、プロセス内の他のルーチンのスレッドに悪影響を及ぼし、トラップ、ハング、およびデータの破損の原因となることがあります。あるスレッドが異常終了した後は、そのプロセスは新規のルーチンの呼び出しに使用されません。すべてのアクティブ・ユーザーがこのプロセスでジョブを完了すると、それは終了されます。 • Java ルーチンを登録する際に、特に指定されない限り、THREADSAFE であると見なされます。その他の LANGUAGE タイプはすべて、デフォルトで NOT THREADSAFE です。 LANGUAGE OLE および OLE DB を使用するルーチンは THREADSAFE として指定できません。 • NOT FENCED ルーチンは THREADSAFE でなければなりません。ルーチンを NOT FENCED NOT THREADSAFE として登録することはできません (SQLCODE -104)。 • UNIX のユーザーは、**db2fmp** (Java) または **db2fmp** (C) を探すことにより、Java および C の THREADSAFE プロセスを参照できます。

表 33. パフォーマンスの考慮事項およびルーチンのパフォーマンスの推奨事項 (続き)

パフォーマンスの考慮事項	パフォーマンスの推奨事項
ルーチンのランタイム・モード (FENCED または NOT FENCED 節の指定)	**FENCED NOT THREADSAFE モード** • FENCED NOT THREADSAFE ルーチンはそれぞれ、独自の専用プロセスで実行します。多数のルーチンを実行している場合、このことはデータベース・システムのパフォーマンスに悪影響を及ぼす可能性があります。ルーチンが他のルーチンと同じプロセスで実行できるほど安全でない場合は、ルーチンを登録する際に NOT THREADSAFE 節を使用してください。 • UNIX では、NOT THREADSAFE プロセスは `db2fmp (pid)` (*pid* は fenced モード・プロセスを使用するエージェントのプロセス ID) またはプール NOT THREADSAFE **db2fmp** の場合は `db2fmp (idle)` として表示されます。
ルーチン内の SQL アクセスのレベル: NO SQL、CONTAINS SQL、READS SQL DATA、MODIFIES SQL DATA	• 低レベルの SQL アクセス節で作成されたルーチンは、高レベルの SQL アクセス節で作成されたルーチンよりもパフォーマンスに優れています。そのため、最も限定的なレベルの SQL アクセス節でルーチンを宣言してください。例えば、ルーチンが SQL データの読み取りだけを行う場合、ルーチンを MODIFIES SQL DATA 節で作成するのではなく、より限定的な READS SQL DATA 節で作成します。
ルーチンの決定論 (DETERMINISTIC または NOT DETERMINISTIC 節の指定)	• DETERMINISTIC または NOT DETERMINISTIC 節でルーチンを宣言しても、ルーチンのパフォーマンスに影響を与えません。
ルーチンによってとられる外部アクションの数および複雑さ (EXTERNAL ACTION 節の指定)	• 外部ルーチンによって実行される外部アクションの数および外部アクションの複雑さによっては、ルーチンのパフォーマンスの妨げとなることがあります。この原因となる要因としては、ネットワーク・トラフィック、書き込みまたは読み取り用のファイルへのアクセス、外部アクションの実行に要する時間、および外部アクションのコードまたは動作のハングに関連したリスクがあります。
入力パラメーターが NULL のときのルーチン呼び出し (CALLED ON NULL INPUT 節の指定)	• NULL の入力パラメーター値を受け取った場合はロジックが実行されず、ルーチンが即時に戻される結果になるのであれば、NULL の入力パラメーター値が検出されたときにルーチンが完全には呼び出されないように、ルーチンを変更することができます。ルーチン入力パラメーターを受け取った場合に呼び出しを早期に終了するルーチンを作成するには、ルーチンを作成して、CALLED ON NULL INPUT 節を指定します。

表 33. パフォーマンスの考慮事項およびルーチンのパフォーマンスの推奨事項 (続き)

パフォーマンスの考慮事項	パフォーマンスの推奨事項
タイプ XML のプロシージャー・パラメーター	• データ・タイプ XML のパラメーターを渡すことは、C または JAVA プログラミング言語でインプリメントされた外部プロシージャーで行う場合、SQL プロシージャーで行う場合と比べてかなり効率が低くなります。データ・タイプ XML の 1 つ以上のパラメーターを渡すときには、外部プロシージャーではなく SQL プロシージャーを使用することを考慮してください。 • XML データは、ストアード・プロシージャーに IN、OUT、または INOUT パラメーターとして渡されるときにマテリアライズされます。Java ストアード・プロシージャーを使用している場合、XML 引数の数量とサイズ、および並行に実行されている外部ストアード・プロシージャーの数に基づいて、ヒープ・サイズ (**java_heap_sz** 構成パラメーター) を増やすことが必要になる場合があります。

いったんルーチンを作成してデプロイすると、環境およびルーチンに固有のどのような要因がルーチンのパフォーマンスに影響を与えるのかを判別するのが難しくなる可能性があります。そのため、パフォーマンスを念頭においてルーチンを設計することが重要です。

サンプル・アプリケーション

pureXML サンプル

pureXML フィーチャーでは、整形式 XML 文書を階層フォーマットで表の列内に保管することができます。XML 列は、新規の XML データ・タイプを使用して定義します。 pureXML フィーチャーは、DB2 データベース・システムに完全に統合されているので、保管された XML データは DB2 機能の効力によりアクセスされ、管理されます。そのような機能には、管理サポート、アプリケーション開発サポート、および、XQuery、SQL または SQL/XML 関数の組み合わせに対するサポートを介する XML の効率のよい検索および取り出しなどがあります。

XML サポートを説明するためのさまざまなサンプルが用意されています。それらは、大きく分けると次のように分類されます。

管理サンプル

このサンプルは、以下のフィーチャーを解説します。

- XML スキーマのサポート: スキーマの登録および XML 文書の妥当性検査。
- XML データの索引付けのサポート: さまざまなノード・タイプの XML 値に対する索引付け。
- XML のユーティリティーに対するサポート: XML データ・タイプの import、export、runstats、db2look、および db2batch のサポート。

アプリケーション開発サンプル

このサンプルは、以下のフィーチャーを解説します。

- XML の挿入、更新、および削除: XML タイプの列への XML 値の挿入、既存値の更新、および既存値の削除。
- XML の構文解析、妥当性検査、およびシリアライゼーションのサポート: 互換データ・タイプの暗黙および明示的な構文解析、XML 文書の妥当性検査、XML データのシリアライズ化。
- SQL および XQuery のハイブリッド使用: XMLTABLE、XMLQUERY、および XMLEXISTS 述部などの SQL/XML 関数の使用。
- SQL および外部プロシージャーでの XML データ・タイプのサポート: データ・タイプ XML のパラメーターの組み込みによる、SQL および外部プロシージャーへの XML データの引き渡し。
- アノテーション付きの XML スキーマの分解のサポート: アノテーション付きの XML スキーマに基づいた XML 文書の分解。
- XML パブリッシング関数: XML 値の構成での関数の使用。

XQuery サンプル
このサンプルは、軸の使用、FLWOR 式、および、XQuery および SQL/XML を使用して作成された照会を解説します。

このサンプルは、次のようなロケーションに置かれています。

- Windows の場合: `%DB2PATH%¥sqllib¥samples¥xml` (`%DB2PATH%` は DB2 データベース・サーバーのインストール先を指定する変数)
- UNIX の場合: `$HOME/sqllib/samples/xml` ($HOME は、インスタンス所有者のホーム・ディレクトリー)

pureXML - 管理のサンプル

これらのサンプルは、XML スキーマ・サポート、ユーティリティー・サポート、XML データ索引付けサポートを含む、さまざまな管理フィーチャー向けの pureXML サポートを示しています。

これらのサンプルは、さまざまなプログラミング言語で用意されており、以下のロケーションにある言語固有サブディレクトリーにあります。

- Windows の場合: `%DB2PATH%¥sqllib¥samples¥xml` (`%DB2PATH%` は DB2 データベース・サーバーのインストール先を指定する変数)
- UNIX の場合: `$HOME/sqllib/samples/xml` ($HOME は、インスタンス所有者のホーム・ディレクトリー)

表 34. XML スキーマ・サポート - スキーマ登録、妥当性検査、および互換性のあるスキーマ発展のサンプル

言語別のサンプル	サンプル・プログラム名	プログラムの説明
CLI	`xsupdate.c`	元のスキーマと新規スキーマの互換性を維持しながら、登録済み XML スキーマを更新します。
C	`xmlschema.sqc`	XML スキーマをデータベースに登録してから、その登録したスキーマを使用して、XML 文書の妥当性検査および挿入を行います。

表 34. XML スキーマ・サポート - スキーマ登録、妥当性検査、および互換性のあるスキーマ発展のサンプル (続き)

言語別のサンプル	サンプル・プログラム名	プログラムの説明
CLP	xmlschema.db2	XML スキーマをデータベースに登録してから、その登録したスキーマを使用して、XML 文書の妥当性検査および挿入を行います。
	xsupdate.db2	元のスキーマと新規スキーマの互換性を維持しながら、登録済み XML スキーマを更新します。
JDBC	XmlSchema.java	XML スキーマをデータベースに登録してから、その登録したスキーマを使用して、XML 文書の妥当性検査および挿入を行います。
	XsUpdate.java	元のスキーマと新規スキーマの互換性を維持しながら、登録済み XML スキーマを更新します。
SQLJ	XmlSchema.sqlj	XML スキーマをデータベースに登録してから、その登録したスキーマを使用して、XML 文書の妥当性検査および挿入を行います。

表 35. ユーティリティー・サポート: XML データ・タイプのインポート、エクスポート、runstats、db2look、reorg、および db2batch サポートのサンプル

言語別のサンプル	サンプル・プログラム名	プログラムの説明
C	xmlrunstats.sqc	XML タイプ列を含む表に対して RUNSTATS を実行します。
	lobstoxml.sqc	IMPORT および EXPORT コマンドを使用して LOB データを XML 列に移動させます。
	impexpxml.sqc	XML 文書をインポートおよびエクスポートします。
	xmlload.sqc	さまざまな LOAD コマンド・オプションを使用して XML 文書を DB2 表にロードします。

表 35. ユーティリティー・サポート: XML データ・タイプのインポート、エクスポート、runstats、db2look、reorg、および db2batch サポートのサンプル (続き)

言語別のサンプル	サンプル・プログラム名	プログラムの説明
CLP	xmlrunstats.db2	XML タイプ列を含む表に対して RUNSTATS を実行します。
	xmlolic.db2	表に対する定義済みの索引の再編成、および範囲パーティション表に対する非パーティション索引を再編成する方法。
	xmldb2batch.db2	XML データ・タイプの db2batch サポート
	xmldb2look.db2	XML データ・タイプの db2look サポート
	lobstoxml.db2	IMPORT および EXPORT コマンドを使用して LOB データを XML 列に移動させます。
	impexpxml.db2	XML 文書をインポートおよびエクスポートします。
	xmlload.db2	さまざまな LOAD コマンド・オプションを使用して XML 文書を DB2 表にロードします。
JDBC	XmlRunstats.java	XML タイプ列を含む表に対して RUNSTATS を実行します。

表 36. XML データ索引付けサポート: XML データに対する索引付けのサンプル

言語別のサンプル	サンプル・プログラム名	プログラムの説明
C	xmlindex.sqc	索引を作成し、それを XQuery 照会で使用します。
	xmlconst.sqc	XML パターンを使用して UNIQUE および VARCHAR 長さ制約がある索引を作成します。
CLI	xmlindex.c	索引を作成し、それを XQuery 照会で使用します。
	xmlconst.c	XML パターンを使用して UNIQUE および VARCHAR 長さ制約がある索引を作成します。
CLP	xmlindex.db2	索引を作成し、それを XQuery 照会で使用します。
	xmlconst.db2	XML パターンを使用して UNIQUE および VARCHAR 長さ制約がある索引を作成します。
JDBC	XmlIndex.java	索引を作成し、それを XQuery 照会で使用します。
	XmlConst.java	XML パターンを使用して UNIQUE および VARCHAR 長さ制約がある索引を作成します。

表 36. XML データ索引付けサポート: XML データに対する索引付けのサンプル (続き)

言語別のサンプル	サンプル・プログラム名	プログラムの説明
SQLJ	XmlIndex.sqlj	索引を作成し、それを XQuery 照会で使用します。
	XmlConst.sqlj	XML パターンを使用して UNIQUE および VARCHAR 長さ制約がある索引を作成します。

pureXML - アプリケーション開発のサンプル

これらのサンプルは、アプリケーション開発機能の XML サポートを示しています。これには、挿入、更新、および削除、XML の構文解析、妥当性検査、およびシリアライゼーション、SQL/XML の混成使用、SQL および外部ストアード・プロシージャーにおける XML データ・タイプ・サポート、XML 分解、SQL/XML パブリッシング関数などがあります。

これらのサンプルは、さまざまなプログラミング言語で用意されており、以下のロケーションにある言語固有サブディレクトリーにあります。

- Windows の場合: %DB2PATH%¥sqllib¥samples¥xml (%DB2PATH% は DB2 データベース・サーバーのインストール先を指定する変数)
- UNIX の場合: $HOME/sqllib/samples/xml ($HOME は、インスタンス所有者のホーム・ディレクトリー)

表 37. pureXML - アプリケーション開発のサンプル

言語別のサンプル	サンプル・プログラム名	プログラムの説明
CLI	xmlinsert.c	XML 文書を XML データ・タイプの列に挿入します。
	xmlupdel.c	表の XML 文書を更新および削除します。
	xmlread.c	表に保管された XML データを読み取ります。
	reltoxmldoc.c	さまざまな SQL/XML パブリッシング関数を使用して、リレーショナル表に保管されたデータから直接 XML 文書を作成します。
	xmltotable.c	XMLTABLE、XMLQUERY、および XMLEXISTS 述部などの SQL/XML 関数を使用して、XML 文書からのデータをリレーショナル表に挿入します。
	simple_xmlproc.c	XML タイプ・パラメーターがある単純なストアード・プロシージャー。
	simple_xmlproc_client.c	simple_xmlproc.c のルーチンを呼び出すためのクライアント・プログラム

表 37. pureXML - アプリケーション開発のサンプル (続き)

言語別のサンプル	サンプル・プログラム名	プログラムの説明
	simple_xmlproc_create.db2	simple_xmlproc.c のストアード・プロシージャーを登録するための CLP スクリプト
	simple_xmlproc_drop.db2	simple_xmlproc.c のストアード・プロシージャーをドロップするための CLP スクリプト
C	xmlinsert.sqc	XML 文書を XML データ・タイプの列に挿入します。
	xmludfs.sqc	スカラー関数、ソース派生関数、SQL を本体として持つ UDF、および表 UDF を使用している場合は、XML データ・タイプを入力パラメーターとして渡し、XML データ・タイプのローカル変数を宣言し、値を返します。
	xmludfs.c	スカラー関数、ソース派生関数、SQL を本体として持つ UDF、および表 UDF を使用している場合は、XML データ・タイプを入力パラメーターとして渡し、XML データ・タイプのローカル変数を宣言し、値を返します。
	xmlupdel.sqc	表の XML 文書を更新および削除します。
	xmlread.sqc	表に保管された XML データを読み取ります。
	reltoxmltype.sqc	さまざまな SQL/XML パブリッシング関数を使用して、リレーショナル表に保管されたデータから XML オブジェクトを作成します。
	xmldecomposition.sqc	XML ファイルに保管されたデータを分解し、そのデータを表に挿入します。 XML 文書の分解時に使用する挿入順序を指定します。
	recxmldecomp.sqc	XSR に再帰的 XML スキーマを登録して、分解のために使用可能にします。
	simple_xmlproc.sqc	XML タイプ・パラメーターがある単純なストアード・プロシージャー。
	simple_xmlproc_client.db2	simple_xmlproc.sqc のルーチンを呼び出すための CLP スクリプト

表 37. pureXML - アプリケーション開発のサンプル (続き)

言語別のサンプル	サンプル・プログラム名	プログラムの説明
	simple_xmlproc_create.db2	simple_xmlproc.sqc のストアード・プロシージャーを登録するための CLP スクリプト
	simple_xmlproc_drop.db2	simple_xmlproc.sqc のストアード・プロシージャーをドロップするための CLP スクリプト
	xmltrig.sqc	トリガー処理機能を使用して、着信 XML 文書の自動妥当性検査を強制実行します。
	xmlintegrate.sqc	XMLROW 関数と XMLGROUP 関数を使用して、XML にリレーショナル・データをマッピングします。XMLQuery のデフォルトの引き渡しメカニズムと、XMLTABLE のデフォルトの列仕様を例示します。
	xmlcheckconstraint.sqc	IS VALIDATED および IS NOT VALIDATED 述部を使用して、XML 列に対するチェック制約付きの表を作成します。また、ACCORDING TO XMLSCHEMA 節を使用して 1 つ以上のスキーマを指定します。
	xmlxslt.sqc	XSLTRANSFORM 関数を使用して、データベース内にある XML 文書を HTML、プレーン・テキスト、または他の形式の XML に変換します (スタイル・シートを使用)。
CLP	xmlinsert.db2	XML 文書を XML データ・タイプの列に挿入します。
	xrpart.db2	範囲パーティション表での XML の使用、およびローカル索引とグローバル索引のサポート。
	xmlpartition.db2	パーティション・データベース環境、MDC、および範囲パーティション表での XML の使用。
	xmlmdc.db2	データを MDC 表から非 MDC 表へ移動し、ブロック索引、グローバル索引、および高速化した挿入と削除を使用します。

表 37. pureXML - アプリケーション開発のサンプル (続き)

言語別のサンプル	サンプル・プログラム名	プログラムの説明
	xmludfs.db2	スカラー関数、ソース派生関数、SQL を本体として持つ UDF、および表 UDF を使用している場合は、XML データ・タイプを入力パラメーターとして渡し、値を返します。
	xmldbafn.db2	インライン DBA 関数の使用による、XML 文書の推定インライン長の判別。
	xmlolic.db2	表に対する定義済みの索引の再編成、および範囲パーティション表に対する非パーティション索引を再編成する方法。
	xmlindgtt.db2	XML データ・タイプの宣言済みのグローバル一時表の使用。
	xmlupdel.db2	表の XML 文書を更新および削除します。
	reltoxmldoc.db2	さまざまな SQL/XML パブリッシング関数を使用して、リレーショナル表に保管されたデータから直接 XML 文書を作成します。
	reltoxmltype.db2	さまざまな SQL/XML パブリッシング関数を使用して、リレーショナル表に保管されたデータから XML オブジェクトを作成します。
	xmldecomposition.db2	XML ファイルに保管されたデータを分解し、そのデータを表に挿入します。 XML 文書の分解時に使用する挿入順序を指定します。
	recxmldecomp.db2	XSR に再帰的 XML スキーマを登録して、分解のために使用可能にします。
	simple_xmlproc.db2	XML タイプ・パラメーターがある単純なストアード・プロシージャー
	xmltotable.db2	XMLTABLE、XMLQUERY、および XMLEXISTS 述部などの SQL/XML 関数を使用して、XML 文書からのデータをリレーショナル表に挿入します。
	xmltrig.db2	トリガー処理機能を使用して、着信 XML 文書の自動妥当性検査を強制実行します。

表 37. pureXML - アプリケーション開発のサンプル (続き)

言語別のサンプル	サンプル・プログラム名	プログラムの説明
	xmlintegrate.db2	XMLROW 関数と XMLGROUP 関数を使用して、XML にリレーショナル・データをマッピングします。XMLQuery のデフォルトの引き渡しメカニズムと、XMLTABLE のデフォルトの列仕様を例示します。
	xmlcheckconstraint.db2	IS VALIDATED および IS NOT VALIDATED 述部を使用して、XML 列に対するチェック制約付きの表を作成します。また、ACCORDING TO XMLSCHEMA 節を使用して 1 つ以上のスキーマを指定します。
	xmlxslt.db2	XSLTRANSFORM 関数を使用して、データベース内にある XML 文書を HTML、プレーン・テキスト、または他の形式の XML に変換します (スタイル・シートを使用)。
JDBC	XmlInsert.java	XML 文書を XML データ・タイプの列に挿入します。
	XmlMdc.java	データを MDC 表から非 MDC 表へ移動し、ブロック索引、グローバル索引、および高速化した挿入と削除を使用します。
	XmlUdfs.java	スカラー関数、ソース派生関数、SQL を本体として持つ UDF、および表 UDF を使用している場合は、XML データ・タイプを入力パラメーターとして渡し、値を返します。
	XmlUpDel.java	表の XML 文書を更新および削除します。
	XmlRead.java	表に保管された XML データを読み取ります。
	RelToXmlDoc.java	SQL/XML パブリッシング関数を使用して、リレーショナル表に保管されたデータから直接 XML 文書を作成します。
	RelToXmlType.java	さまざまな SQL/XML パブリッシング関数を使用して、リレーショナル表に保管されたデータから XML オブジェクトを作成します。

表 37. *pureXML - アプリケーション開発のサンプル (続き)*

言語別のサンプル	サンプル・プログラム名	プログラムの説明
	XmlDecomposition.java	XML ファイルに保管されたデータを分解し、そのデータを表に挿入します。 XML 文書の分解時に使用する挿入順序を指定します。
	RecXmlDecomp.java	XSR に再帰的 XML スキーマを登録して、分解のために使用可能にします。
	Simple_XmlProc.java	XML タイプ・パラメーターがある単純なストアード・プロシージャー。
	Simple_XmlProc_Client.java	Simple_XmlProc.java のルーチンを呼び出すためのクライアント・プログラム
	Simple_XmlProc_Create.db2	Simple_XmlProc.java のストアード・プロシージャーを登録するための CLP スクリプト
	Simple_XmlProc_Drop.db2	Simple_XmlProc.java のストアード・プロシージャーをドロップするための CLP スクリプト
	XmlToTable.java	XMLTABLE、XMLQUERY、および XMLEXISTS 述部などの SQL/XML 関数を使用して、XML 文書からのデータをリレーショナル表に挿入します。
	XmlTrig.java	トリガー処理機能を使用して、着信 XML 文書の自動妥当性検査を強制実行します。
	XmlCheckConstraint.java	IS VALIDATED および IS NOT VALIDATED 述部を使用して、XML 列に対するチェック制約付きの表を作成します。また、ACCORDING TO XMLSCHEMA 節を使用して 1 つ以上のスキーマを指定します。
SQLJ	XmlInsert.sqlj	XML 文書を XML データ・タイプの列に挿入します。
	XmlUpDel.sqlj	表の XML 文書を更新および削除します。
	XmlRead.sqlj	表に保管された XML データを読み取ります。
	RelToXmlDoc.sqlj	SQL/XML パブリッシング関数を使用して、リレーショナル表に保管されたデータから直接 XML 文書を作成します。

表 37. pureXML - アプリケーション開発のサンプル (続き)

言語別のサンプル	サンプル・プログラム名	プログラムの説明
	RelToXmlType.sqlj	SQL/XML パブリッシング関数を使用して、リレーショナル表に保管されたデータから XML オブジェクトを作成します。
	XmlToTable.sqlj	XMLTABLE、XMLQUERY、および XMLEXISTS 述部などの SQL/XML 関数を使用して、XML 文書からのデータをリレーショナル表に挿入します。
	XmlIntegrate.sqlj	XMLROW 関数と XMLGROUP 関数を使用して、XML にリレーショナル・データをマッピングします。XMLQuery のデフォルトの引き渡しメカニズムと、XMLTABLE のデフォルトの列仕様を例示します。
	XmlXslt.sqlj	XSLTRANSFORM 関数を使用して、データベース内にある XML 文書を HTML、プレーン・テキスト、または他の形式の XML に変換します (スタイル・シートを使用)。
PHP	XmlFlwor_DB2.php	XQuery FLWOR 式を使用します。
	XmlIndex_DB2.php	索引を作成し、それを XQuery で使用します。
	XmlInsert_DB2.php	XML 文書を XML データ・タイプの列に挿入します。
	XmlRead_DB2.php	表に保管された XML データを読み取ります。
	XmlRelToXmlDOC_DB2.php	SQL/XML パブリッシング関数を使用して、リレーショナル表に保管されたデータから直接 XML 文書を作成します。
	XmlRelToXmlType_DB2.php	SQL/XML パブリッシング関数を使用して、リレーショナル・データおよび XML データから XML 文書を作成します。
	XmlRunstats_DB2.php	XML タイプ列を含む表に対して RUNSTATS を実行します。
	XmlSchema_DB2.php	XML スキーマをデータベースに登録してから、その登録したスキーマを使用して、XML 文書の妥当性検査および挿入を行います。
	XmlSQLXQuery_DB2.php	SQL/XML 照会を使用します。

表 37. *pureXML - アプリケーション開発のサンプル (続き)*

言語別のサンプル	サンプル・プログラム名	プログラムの説明
	XmlUniqueIndexes_DB2.php	UNIQUE および VARCHAR 長さ制約がある索引を作成します。
	XmlUpAndDel_DB2.php	表の XML 文書を更新および削除します。
	XmlToTable_DB2.php	SQL/XML を使用して、XML 文書からのデータをリレーショナル表に挿入します。
	XmlXPath_DB2.php	単純な XPath 照会を実行します。
	XmlXQuery_DB2.php	ネストされた XQuery FLWOR 式を実行します。

第 11 章 XML パフォーマンス

以下のトピックでは、pureXML フィーチャーを使用する場合に参考にすることができるパフォーマンス・チューニング考慮事項などを記載しています。

関連情報:

DB2 の pureXML のパフォーマンスのための 15 のベスト・プラクティス

pureXML およびデータ編成スキーム

pureXML を、データベース・パーティション環境、表パーティション化、およびマルチディメンション・クラスタリングなどのデータ編成スキームとともに使用すると、照会パフォーマンスが大幅に向上し、データ保守操作 (再編成、挿入、および削除など) のオーバーヘッドは大幅に少なくなります。

データを編成する方法を指定するアルゴリズムを含む CREATE TABLE ステートメントには、以下の 3 つの節があります。

- DISTRIBUTE BY。パーティション・データベース環境のデータベース・パーティション間で均等にデータを広げます (データベース・パーティション化)。
- PARTITION BY。表の表パーティション化スキームを指定します (表パーティション化)。
- ORGANIZE BY。表データをクラスター化するために使用する、各列、または列のグループのディメンションを指定します (マルチディメンション・クラスタリング)。

pureXML を使用する際の制約事項については、『pureXML に対する制約事項』および『XML データに対する索引の制約事項』を参照してください。

パーティション・データベース環境

パーティション・データベース環境で表を使用する場合、XML 列を含む表を、複数のマシンの複数パーティション・データベースに保管できます。 XML データは、DB2 pureXML を使用して管理できます。

XML データがデータベース・パーティション間に分散されている場合、複数のマシンの複数のプロセッサーは情報要求を処理できます。データ検索および更新要求は、サブ要求に自動的に分解され、適切なデータベース・パーティション間で並行して実行されます。

パーティション表

パーティション表には、XML データ・タイプの 1 つ以上の列、および XML データの索引を 1 つ以上含めることができます。 XML データのユーザー作成索引は、パーティションまたは非パーティションで可能です。 ALTER TABLE ステートメントの ATTACH PARTITION および DETACH PARTITION 節を表データのロールインおよびロールアウトで使用するとき、パーティション索引は、環境でのデータ保守操作のオーバーヘッドを削減します。

マルチディメンション・クラスタリング表

マルチディメンション・クラスタリング (MDC) 表には、XML データ・タイプを持つ 1 つ以上の列を含めることができ、XML 列の XML データに 1 つ以上の索引を作成することができます。 XML データの索引を MDC ブロック索引とともに使用して、照会パフォーマンスを向上させることができます。さらに、XML データの索引を MDC ブロック索引とともに使用して、索引 ANDing を実行することもできます。

パーティション・データベース環境における XQuery 変換の使用例

パーティション・データベース環境では XQuery 変換式がサポートされており、抽出、変換、およびロード (ETL) の各操作を使用するシナリオで使用可能です。

例えば、XML データを 2 つのソース表 ORDERS と PRODUCT 間の結合の結果から抽出し、必要な形式に変換し、ターゲット表 SALES に XML データとして挿入する場合、XQuery 変換式を使用できます。ただし、式への入力が、複数のソース表の結合ではなく、単一の表のみを参照する場合に、変換式は最良のパフォーマンスが得られます。

複数の表で XQuery 変換式を使用する場合、結合結果がソース表のサイズに比べて小さければ、変換式を結合結果に直接適用する単一ステートメントを記述しても、通常は、許容可能なパフォーマンスが得られます。しかし、結合によって作成される行の数がソース表の行の数に匹敵しているか、またはより大きい場合、結合および変換式を 2 つのステートメントに分割することを検討してください。

以下の例は、結合および XQuery の変換操作を使用する単一ステートメントを 2 つのステートメントに分割する一般的な手法を示します。 2 つのステートメントにより、パーティション・データベース環境で変換操作は確実に並列化されます。この手法は、以下のステップを使用します。

1. ターゲット表と同じパーティション・グループ内に、ターゲット表と同じ分散キーを持つ新しい中間表を作成します。この中間表を、宣言済みのグローバル一時表にできます。
2. 複数のソース表のデータを取り出し、それを中間表に挿入します。
3. 中間表の XML データを変換して、その変換済みデータをターゲット表に挿入します。

パーティション・データベース環境で使用可能な並列処理を活用することで、ステップ 2 と 3 の両方をスケーラブルなものにできます。ステップ 3 では、ネストされた変換式は用いるべきではありません。

例で使用される表およびデータ

例の中では、ソース表 ORDERS および PRODUCTS、ターゲット表 SALES、および宣言済みグローバル一時表 TEMPSALES が使用されています。この例では、ORDER 表からデータを取り出し、そのデータを PRODUCTS 表の価格情報と結合し、結果データを形式設定し、その形式設定済みデータを SALES 表に挿入します。

ORDERS 表には日次注文が含まれ、注文 ID (OID) と注文情報が XML 列 (ORDERDETAIL) に入っています。各 XML 文書には、顧客 ID (Cid) と注文された製品が含まれています。注文された各製品の情報には、製品 ID、数量、および配送要件が含まれます。以下の CREATE ステートメントによって、サンプルの ORDERS 表が作成されます。

```
CREATE TABLE ORDERS(OID BIGINT, ORDERDETAIL XML)DISTRIBUTE BY HASH (OID);
```

以下の INSERT ステートメントによって、ORDERS 表に注文サンプルが挿入されます。

```
INSERT into ORDERS
values (5003, '<order>
                <cid>1001</cid>
                <product>
                  <pid>2344</pid><qty>10</qty>
                  <delivery>Overnight</delivery>
                </product>
                <product>
                   <pid>537</pid><qty>3</qty>
                   <delivery>Ground</delivery>
                </product>
               </order>');
```

PRODUCTS 表には製品情報として製品 ID(PID)、製品価格 (PRICE)、および XML 列 (PRODDETAIL) に製品の詳細情報が含まれます。以下の CREATE ステートメントによって、サンプルの PRODUCTS 表が作成されます。

```
CREATE TABLE PRODUCTS(PID BIGINT, PRICE FLOAT, PRODDETAIL XML);
```

以下の INSERT ステートメントによって、PRODUCTS 表に製品データが挿入されます。

```
INSERT into PRODUCTS
values(2344, 4.99, '<product>
                      <name>10 D-Cell batteries</name>
                      <desc>D Cell battery, 10-pack</desc>
                    </product>')

INSERT into PRODUCTS
values(537, 8.99, '<product>
                     <name>Ice Scraper, small</name>
                     <desc>Basic ice scraper, 4 inches wide</desc>
                   </product>');
```

SALES 表には、注文 ID (OID)、顧客 ID (CID)、製品 ID 情報 (PID)、注文の合計額 (ITEMTOTAL)、XML 列に格納された各注文の詳細情報 (SALEDETAIL) が含まれます。SALES 表の各行には、注文された個々の製品に関する情報が含まれています。SALES 表は、注文 ID (OID) 列に従って分散されています。以下の CREATE ステートメントによって、サンプルの SALES 表が作成されます。

```
CREATE TABLE SALES(OID BIGINT, CID BIGINT, PID BIGINT, ITEMTOTAL FLOAT,
    SALESDETAIL XML) DISTRIBUTE BY HASH (OID);
```

単一ステートメントでの表結合および XQuery 変換式

以下の INSERT ステートメントは ORDER データと PRODUCT データを結合し、変換式を結果の XML 文書に適用し、更新された文書を SALES 表に挿入します。

```
INSERT into SALES
select T.OID, T.CID, T.PID, T.ITEMTOTAL,
   XMLQUERY('
```

```
     copy $new := $temp
     modify (do delete ($new/info/cid,
     $new/info/product/pid,
     $new/info/product/qty),
       do insert <orderdate>{fn:current-date()}</orderdate>
       as first into $new/info)
    return $new' passing T.SALESDETAIL as "temp")
from(
SELECT O.OID, OX.CID, OX.PID, P.PRICE * OX.QTY, OX.SALESDETAIL
FROM PRODUCTS P,
     ORDERS O,
     XMLTABLE('for $i in $details/order/product
               return document{<info> {$details/order/cid} {$i} </info>}'
               passing O.ORDERDETAIL as "details"
     columns
        CID bigint path './info/cid',
        PID bigint path './info/product/pid',
        QTY int    path './info/product/qty',
        SALESDETAIL xml path '.') as OX
WHERE P.PID = OX.PID) as T(OID, CID, PID, ITEMTOTAL, SALESDETAIL);
```

次のセクションでは、前のステートメントで実行された表結合および XQuery 変換を、2 つの別個のステートメントでどのように実行できるかを示します。

別個のステートメントでの表結合および XQuery 変換式

325 ページの『単一ステートメントでの表結合および XQuery 変換式』のステートメントの後半の結合の結果が、ORDER 表と PRODUCT 表のサイズと比較して十分に大きい場合、2 つのステートメントに分割するとパフォーマンスが向上します。単一ステートメントを 2 つに分割するとき、最初のステートメントは ORDER 表と PRODUCT 表との結合の結果を一時表に挿入します。 2 番目のステートメントは、変換を一時表の XML 文書に適用し、更新された文書を SALES 表に挿入します。

一時表は SALES 表と同様で、中間照会結果を格納します。日次注文を処理するには、宣言済みグローバル一時表 TEMPSALES が使用されます。DB2 オプティマイザーが一時表と SALES 表を更新するために使用する照会を適切に最適化するには、一時表 TEMPSALES とターゲット表 SALES の分散キーが同じで、同じパーティション・グループに属している必要があります。以下の DECLARE ステートメントによって、一時表 TEMPSALES が作成されます。

```
DECLARE GLOBAL TEMPORARY TABLE SESSION.TEMPSALES LIKE SALES
    DISTRIBUTE BY HASH (OID);
```

以下の SELECT ステートメントには、PRODUCTS 表と ORDERS 表の間の結合が含まれています。このステートメントは、PRODUCTS 表と ORDERS 表の情報を使用して ITEMTOTAL 列の値を計算し、TEMPSALES 表に挿入される XML 文書を構成します。結合は、325 ページの『単一ステートメントでの表結合および XQuery 変換式』内の INSERT ステートメントでの結合と同じです。

```
INSERT INTO SESSION.TEMPSALES
SELECT O.OID, OX.CID, OX.PID, P.PRICE * OX.QTY, OX.SALESDETAIL
FROM PRODUCTS P,
     ORDERS O,
     XMLTABLE('for $i in $details/order/product
               return document{<info> {$details/order/cid} {$i} </info>}'
               passing O.ORDERDETAIL as "details"
     columns
        CID bigint path './info/cid',
```

```
        PID bigint path './info/product/pid',
        QTY int    path './info/product/qty',
        SALESDETAIL xml path '.') as OX
WHERE P.PID = OX.PID;
```

INSERT ステートメントによって使用される前述の SELECT ステートメントには、XQuery 変換式は含まれていません。

一時表には、SALES 表に必要なリレーショナル情報が含まれています。一時表の XML 文書では製品情報を除去し、注文日付を追加する必要があります。

以下の INSERT ステートメントによって、一時表から日次販売情報が選択され、その情報が SALES 表に挿入されます。この INSERT ステートメントには、325 ページの『単一ステートメントでの表結合および XQuery 変換式』内の INSERT ステートメントと同じ XQuery 変換式を使用する SELECT ステートメントが含まれています。この式は、XML 文書を SALES 表に挿入する前に、一時表のそれぞれの文書を変更します。TEMPSALES 表と SALES 表は同じパーティション・グループにあり、共通の分散キーを共有しているので、挿入を並列化できます。

```
INSERT into SALES
select T.OID, T.CID, T.PID, T.ITEMTOTAL,
   XMLQUERY('
     copy $new := $temp
     modify (do delete ($new/info/cid,
     $new/info/product/pid,
     $new/info/product/qty),
       do insert <orderdate>{fn:current-date()}</orderdate>
       as first into $new/info)
    return $new' passing T.SALESDETAIL as "temp")
from SESSION.TEMPSALES T;
```

XML データが含まれる宣言済み一時表の使用

宣言済みのグローバル一時表は、非永続表の作成が必要とされるソリューションを実装する際に役立ちます。例えば、アプリケーションは宣言済み一時表を作成して、中間結果を処理できます。アプリケーションのセッションが終了すると、その一時表はドロップされます。

宣言済みのグローバル一時表は、それを宣言したセッションに対してのみ存在します。この表を、他のセッションと共有することはできません。セッションが終了すると、この表の行が削除され、一時表の記述もドロップされます。宣言済み一時表にはカタログ競合の問題は存在しません。宣言済み一時表にはカタログ項目がないからです。

宣言済みのグローバル一時表に XML 列を含め、XML データを照会および更新することが可能です。また宣言済みのグローバル一時表は、パーティション・データベース環境でも使用できます。一時表にパーティション・キーとして指定された列が含まれる場合、一時表内の XML データで実行される処理をデータベース・パーティション間で分散できます。

例

以下の例では、会社の従業員に関する表に XML データが含まれるとします。従業員データは、以下の従業員情報のようになります。

```
<company name="MyFirstComany">
  <emp id="31201" salary="60000" gender="Female">
    <name>
      <first>Laura</first>
      <last>Brown</last>
    </name>
    <dept id="M25">Finance</dept>
  </emp>
</company>
```

以下の CREATE ステートメントにより、XML 列が含まれる 1 つの従業員表が作成されます。

```
CREATE TABLE COMPANYINFO (ID INT, DOC XML)
```

以下の 2 つのステートメントは、従業員表とともに使用できるグローバル一時表を作成します。どちらのステートメントも、COMPANYINFO 表と同じ列名と記述を持つ一時表を作成します。最初のステートメントは、列定義を使用して一時表を作成します。2 番目のステートメントは、LIKE 節を使用して一時表を作成します。この表の列は、指定された表の列と同じ名前と記述になります。 ON COMMIT DELETE ROWS 節は、COMMIT 操作の実行時に WITH HOLD カーソルが表でオープンされていないと、表のすべての行が削除されるように指定します。

```
DECLARE GLOBAL TEMPORARY TABLE INSTMPTB (ID INT, DOC XML)
    ON COMMIT DELETE ROWS in USR_TBSP

DECLARE GLOBAL TEMPORARY TABLE INSTMPTB LIKE COMPANYINFO
    ON COMMIT DELETE ROWS in USR_TBSP
```

以下の DECLARE ステートメントは、XML 文書の基本表の行ストレージが含まれる一時表を作成します。XML 文書は、指定されたインライン長未満であると、基本表に格納されます。

```
DECLARE GLOBAL TEMPORARY TABLE TEMPTB (ID INT, DOC XML INLINE LENGTH 3000)
   ON COMMIT PRESERVE ROWS NOT LOGGED
```

大量のデータをグローバル一時表に挿入し、そのデータに照会を実行する場合には、XML データに索引を作成し、パフォーマンスを向上させることができます。例えば、以下の CREATE INDEX ステートメントは、XML データの 2 つの索引を作成します。最初の索引は、XML 文書内の従業員 ID のものです。2 番目の索引は、従業員の姓のものです。

```
CREATE INDEX SESSION.TEMP_IDX ON SESSION.INSTMPTB (DOC)
  GENERATE KEY USING XMLPATTERN '/company/emp/@id'
  AS SQL INTEGER

CREATE UNIQUE INDEX SESSION.TEMP_IDX2 ON SESSION.INSTMPTB (DOC)
  GENERATE KEY USING XMLPATTERN '/company/name/last'
  AS SQL VARCHAR(100)
```

パーティション・データベース環境に宣言済みのグローバル一時表を作成して、データベース・パーティション化の利点を活用できます。以下の DECLARE ステートメントは、DOCID を分散キーとして使用する一時表を作成します。一時表に XML データを追加してから、XML データで照会や他の操作を実行すると、パーティション・データベース環境を活用できます。

```
DECLARE GLOBAL TEMPORARY TABLE INSTMPTB (ID INT, DOC XML)
   ON COMMIT DELETE ROWS
   IN USR_TBSP
   DISTRIBUTE BY HASH (ID)
```

以下の XQuery 式は、グローバル一時表と COMPANYINFO 表を使用します。一時表には、COMPANYINFO 表の文書のサブセットが含まれています。XQuery 式は、一時表に入っている財務部に所属する従業員に関して、COMPANYINFO 表から従業員情報を返します。

```
XQUERY
  for $i in db2-fn:xmlcolumn("SESSION.INSTMPTB.DOC")/company/emp
  for $j in db2-fn:xmlcolumn("COMPANYINFO.DOC ")[/company/emp/@id = $i/@id ]
  where $i/dept = "Finance"
  return $j ;
```

以下の INSERT ステートメントは、一時表のデータを COMPANYINFO 表に挿入します。

```
INSERT INTO COMPANYINFO FROM
 (SELECT ID, DOC FROM SESSION.INSTMPTB)
```

使用上の注意

以下の項目は、宣言済みのグローバル一時表を使用する場合に当てはまります。

- データ行圧縮が、宣言済み一時表に対して有効です。データベース・マネージャーがパフォーマンスの向上に役立つと判断する場合には、基本表オブジェクトにインラインで格納される XML 文書を含む表の行データが圧縮されます。ただし、宣言済み一時表の XML ストレージ・オブジェクトのデータ圧縮はサポートされません。
- パーティション・データベース環境では、DISTRIBUTE BY 節を使用して、パーティション・キーのある宣言済み一時表を作成します。LIKE 節を使用して宣言済み一時表を作成し、ソース表にパーティション・キーがある場合、一時表にはパーティション・キーがありません。
- XML データに関するユーザー作成索引を含め、宣言済み一時表に作成される索引で、索引圧縮は有効です。
- XML データに関するユーザー作成索引には、一時表以外の表に作成される索引と同じ制約事項があります。例えば、パーティション・データベース環境で、XML データのユニークな XML 索引はサポートされません。

XML データと XQuery 式における最適化ガイドラインの使用

最適化プロファイルで使用される DB2 最適化ガイドラインは XML データをサポートします。パーティション・データベース環境における XML データの移動方式、XML データ・タイプの結合順序、XML データのユーザー定義索引の使用法を制御するために、ガイドラインを使用するプロファイルを作成できます。

XML データにアクセスする照会、または XML データの索引を使用する照会のために、最適化ガイドラインで以下のタイプの最適化を指定できます。

- DPFXMLMOVEMENT 汎用要求要素を使用して、パーティション・データベース環境内のパーティション間で XML データを移動する方法を制御します。
- プラン最適化ガイドラインで、XML データ・タイプにおける結合の結合順序を制御します。そのためには、アクセス要求要素で属性 FIRST を TRUE に設定するか、結合要求要素を使用します。
- 以下のアクセス要求要素を用いて XML データの索引の使用法を制御します。

- XISCAN アクセス要求要素を用いて、表にアクセスするために単一の XML 索引スキャンを使用するように指定します。
- XANDOR アクセス要求要素を用いて、表にアクセスするために複数の XANDOR された XML 索引スキャンを使用するように指定します。
- TYPE 属性値が XMLINDEX に設定された IXAND アクセス要求要素を使用して、複数のリレーショナルおよび XML 索引スキャンを使用するように指定します。
- ACCESS アクセス要求要素を使用し、属性 TYPE を XMLINDEX に設定して、DB2 オプティマイザーがコスト・ベースの分析を実行し、表にアクセスするために一部の XML 索引アクセスを選択するように指定します。例えば、オプティマイザーは、XML 索引スキャン、XANDOR、または少なくとも 1 つの XML 索引を使用する索引 ANDing などの XML 索引アクセスを使用できます。
- ACCESS アクセス要求要素を使用し、属性 TYPE を XMLINDEX に、ALLINDEXES を TRUE にそれぞれ設定して、コストに関係なく、指定の表にアクセスするために適用可能なすべてのリレーショナル索引および XML データの索引を使用するように指定します。
- IXAND アクセス要求要素を使用し、属性 TYPE を XMLINDEX に、ALLINDEXES を TRUE にそれぞれ設定して、コストに関係なく、指定の表にアクセスするために適用可能なすべてのリレーショナル索引および XML データの索引を IXAND プラン内で使用するように指定します。

XML データを使用した最適化のガイドラインの例

最適化プロファイル例には、XML データにアクセスする照会における最適化の実行方法を制御するための汎用要求、アクセス方式、および結合順序に関するガイドラインが含まれています。

参照としての XML 文書を移動するためのガイドライン

以下の最適化プロファイルでは、ガイドライン内の DPFXMLMOVEMENT 汎用要求要素によって、XML 文書への参照がアクセス・プラン内の TQ 演算子を介して移動するように指定します。

```
<?xml version="1.0" encoding="UTF-8"?>
<OPTPROFILE VERSION="9.7.0.0">
<STMTPROFILE ID="Security Tables">
  <STMTKEY SCHEMA="ST">
    SELECT *
    FROM security
    WHERE XMLEXISTS('$SDOC/Security/SecurityInfo
      /StockInfo[Industry= "OfficeSupplies"]')
  </STMTKEY>
  <OPTGUIDELINES>
    <DPFXMLMOVEMENT VALUE="REFERENCE"/>
  </OPTGUIDELINES>
</STMTPROFILE>
</OPTPROFILE>
```

XML データの特定の索引を使用するガイドライン

以下の最適化プロファイルではガイドライン内の XISCAN アクセス要素によって、表 SECURITY が索引 SEC_INDUSTRY を使用してアクセスしなければならないこ

とを指定します。XISCAN 要素が INDEX 属性を使用して索引名を指定しなかった場合、オプティマイザーは最小のコストで XML データの索引を使用して、表 SECURITY にアクセスします。

```
<?xml version="1.0" encoding="UTF-8"?>
<OPTPROFILE VERSION="9.7.0.0">
<STMTPROFILE ID="Security Tables">
  <STMTKEY SCHEMA="ST">
    SELECT *
    FROM security
    WHERE XMLEXISTS('$SDOC/Security/SecurityInfo
      /StockInfo[Industry= "OfficeSupplies"]')
  </STMTKEY>
  <OPTGUIDELINES>
    <XISCAN TABLE='SECURITY' INDEX='SEC_INDUSTRY'/>
  </OPTGUIDELINES>
</STMTPROFILE>
</OPTPROFILE>
```

XANDOR アクセスで XML データの索引を使用するためのガイドライン

以下の最適化プロファイルのガイドラインでは、XANDOR 要素は、XML データのすべての適用可能な索引の XANDOR プランを使用して表 SECURITY にアクセスすることを指定します。XANDOR プランはリレーショナル索引を使用できないので、リレーショナル索引は使用されません。

```
<?xml version="1.0" encoding="UTF-8"?>
<OPTPROFILE VERSION="9.7.0.0">
<STMTPROFILE ID="Security Tables by Date">
  <STMTKEY SCHEMA="STBD">
    SELECT *
    FROM security
    WHERE trans_date = CURRENT DATE
      AND XMLEXISTS('$SDOC/Security/SecurityInfo
           /StockInfo[Industry= "Software"]')
      AND XMLEXISTS('$SDOC/Security/Symbol[.="IBM"]')
  </STMTKEY>
  <OPTGUIDELINES>
    <XANDOR TABLE='SECURITY' />
  </OPTGUIDELINES>
</STMTPROFILE>
</OPTPROFILE>
```

IXAND で XML データの指定された複数の索引を使用するためのガイドライン

以下の最適化プロファイルでは、最適化ガイドラインの IXAND 要素は、XML データの 2 つの索引、SEC_INDUSTRY と SEC_SYMBOL を使用して表 SECURITY にアクセスすることを指定します。オプティマイザーは IXAND プランを生成します。このプランでは、リストされている順序で IXAND プランのレグとして 2 つの索引が使用されます。

```
<?xml version="1.0" encoding="UTF-8"?>
<OPTPROFILE VERSION="9.7.0.0">
<STMTPROFILE ID="Security Tables by Date">
  <STMTKEY SCHEMA="STBD">
    SELECT *
    FROM security
    WHERE trans_date = CURRENT DATE
      AND XMLEXISTS('$SDOC/Security/SecurityInfo
          /StockInfo[Industry= "Software"]')
```

```
      AND XMLEXISTS('$SDOC/Security/Symbol[.="IBM"]')
  </STMTKEY>
  <OPTGUIDELINES>
    <IXAND TABLE='SECURITY' TYPE='XMLINDEX'>
      <INDEX IXNAME='SEC_INDUSTRY'/>
      <INDEX IXNAME='SEC_SYMBOL'/>
    </IXAND>
  </OPTGUIDELINES>
</STMTPROFILE>
</OPTPROFILE>
```

IXAND で XML データのすべての索引を使用するためのガイドライン

以下の最適化プロファイルでは、最適化ガイドラインの IXAND 要素は、すべての適用可能なリレーショナル索引と XML データの索引を使用して表 SECURITY にアクセスすることを指定します。TRANS_DATE にリレーショナル索引があり、SEC_INDUSTRY と SEC_SYMBOL には XML データの索引があるとして、オプティマイザーが選択した順序でこれら 3 つの索引すべてはともに ANDing されます。

```
<?xml version="1.0" encoding="UTF-8"?>
<OPTPROFILE VERSION="9.7.0.0">
<STMTPROFILE ID="Security Tables by Date">
  <STMTKEY SCHEMA="STBD">
    SELECT *
    FROM security
    WHERE trans_date = CURRENT DATE
    AND XMLEXISTS('$SDOC/Security/SecurityInfo
         /StockInfo[Industry= "Software"]')
    AND XMLEXISTS('$SDOC/Security/Symbol[.="IBM"]')
  </STMTKEY>
  <OPTGUIDELINES>
    <IXAND TABLE='SECURITY' TYPE='XMLINDEX' ALLINDEXES='TRUE' />
  </OPTGUIDELINES>
</STMTPROFILE>
</OPTPROFILE>
```

IXAND で XML データの特定の先行索引を使用するガイドライン

以下の最適化プロファイルでは、最適化ガイドラインの IXAND 要素は、IXAND プランを使用して表 SECURITY にアクセスしなければならず、XML データの索引 SEC_INDUSTRY は IXAND 内の最初の索引でなければならないことを指定します。オプティマイザーは、コスト・ベースの方式で、IXAND プラン用の追加の索引を選出します。リレーショナル索引が列 TRANS_DATE で使用可能で、XML データの索引がパス SYMBOL で使用できる場合には、オプティマイザーによって有用であると判断されると、これらの索引の片方または両方が IXAND プランの追加レグとして表示されます。

```
<?xml version="1.0" encoding="UTF-8"?>
<OPTPROFILE VERSION="9.7.0.0">
<STMTPROFILE ID="Security Tables by Date">
  <STMTKEY SCHEMA="STBD">
    SELECT *
    FROM security
    WHERE trans_date = CURRENT DATE
      AND XMLEXISTS('$SDOC/Security/SecurityInfo
            /StockInfo[Industry= "Software"]')
      AND XMLEXISTS('$SDOC/Security/Symbol[.="IBM"]')
  </STMTKEY>
  <OPTGUIDELINES>
```

```
    <IXAND TABLE='SECURITY' TYPE='XMLINDEX' INDEX='SEC_INDUSTRY' />
  </OPTGUIDELINES>
</STMTPROFILE>
</OPTPROFILE>
```

ある XML 索引アクセスを使用するためのガイドライン

以下の最適化プロファイルのガイドラインは、表 SECURITY にアクセスするためにある XML データの索引を使用することだけを指定します。オプティマイザーは、コスト・ベースの分析を使用して、XISCAN、IXAND、XANDOR、または IXOR プランを使用します。

```
<?xml version="1.0" encoding="UTF-8"?>
<OPTPROFILE VERSION="9.7.0.0">
<STMTPROFILE ID="Security Tables">
  <STMTKEY SCHEMA="ST">
     SELECT *
     FROM security
     WHERE XMLEXISTS('$SDOC/Security/SecurityInfo
            /StockInfo[Industry= "OfficeSupplies"]')
  </STMTKEY>
  <OPTGUIDELINES>
     <ACCESS TABLE='SECURITY' TYPE='XMLINDEX' />
  </OPTGUIDELINES>
</STMTPROFILE>
</OPTPROFILE>
```

すべての適用可能な XML データの索引アクセスを使用するためのガイドライン

以下の最適化プロファイルのガイドラインは、SECURITY 表のすべての適用可能な索引を使用することを指定します。方式の選択はオプティマイザーが行います。XMLEXISTS 内の 2 つの述部が一致する、XML データの 2 つの索引 SEC_INDUSTRY と SEC_SYMBOL が作成されたと想定します。オプティマイザーは、XANDOR プランか IXAND プランのどちらかを使用することを選択します。オプティマイザーは、コスト・ベースの分析を使用して 2 つのアクセス・プランのいずれかを選択します。

```
<?xml version="1.0" encoding="UTF-8"?>
<OPTPROFILE VERSION="9.7.0.0">
<STMTPROFILE ID="Security Tables 2">
  <STMTKEY SCHEMA="ST2">
    SELECT *
    FROM security
    WHERE XMLEXISTS('$SDOC/Security/SecurityInfo
             /StockInfo[Industry= "Software"]')
    AND XMLEXISTS('$SDOC/Security/Symbol[.="IBM"]')
  </STMTKEY>
  <OPTGUIDELINES>
    <ACCESS TABLE='SECURITY' TYPE='XMLINDEX' ALLINDEXES='TRUE' />
  </OPTGUIDELINES>
</STMTPROFILE>
</OPTPROFILE>
```

特定の索引を使用する、XML データに対する索引アクセスを指定する場合のガイドライン

以下の最適化プロファイルのガイドラインは、少なくとも SEC_INDUSTRY 索引を使用して表 SECURITY にアクセスすることを指定します。オプティマイザーは、コスト・ベースの分析を使用して、以下のいずれかのアクセス・プランを選出します。

1. SEC_INDUSTRY 索引を使用する XISCAN プラン。
2. IXAND プランの最初のレグとして指定された索引を使用する IXAND プラン。オプティマイザーは、コスト・ベースの分析に基づいて、IXAND プランでさらに索引を使用する場合があります。この例では、リレーショナル索引が列 TRANS_DATE で使用可能な場合、オプティマイザーによって有用であると判断されると、その索引が IXAND プランの追加レグとして表示されます。
3. 指定の索引と、他の適用可能な XML データの索引を使用する XANDOR プラン。

```
<?xml version="1.0" encoding="UTF-8"?>
<OPTPROFILE VERSION="9.7.0.0">
<STMTPROFILE ID="Security Tables by Date">
  <STMTKEY SCHEMA="STBD">
    SELECT *
    FROM security
    WHERE trans_date = CURRENT DATE
      AND XMLEXISTS('$SDOC/Security/SecurityInfo
             /StockInfo[Industry= "Software"]')
      AND XMLEXISTS('$SDOC/Security/Symbol[.="IBM"]')
  </STMTKEY>
  <OPTGUIDELINES>
    <ACCESS TABLE='SECURITY' TYPE='XMLINDEX' INDEX='SEC_INDUSTRY' />
  </OPTGUIDELINES>
</STMTPROFILE>
</OPTPROFILE>
```

以下の最適化ガイドラインは、コスト・ベースの分析を使用して、以下のいずれかのアクセス・プランを使用するように指定します。

- SEC_INDUSTRY 索引と SEC_SYMBOL 索引を指定の順序で用いる IXAND プランを使用します。
- すべての適用可能な XML 索引を用いる XANDOR プランを使用します。

```
<OPTGUIDELINES>
  <ACCESS TABLE='SECURITY' TYPE='XMLINDEX'>
    <INDEX IXNAME='SEC_INDUSTRY'/>
    <INDEX IXNAME='SEC_SYMBOL'/>
  </ACCESS>
</OPTGUIDELINES>
```

結合順序を制御し、XML データの索引アクセスを指定するためのガイドライン

以下の最適化プロファイルの最適化ガイドラインには、2 つの要素が含まれています。最初のガイドライン要素は、FROM 節内の表が結合される際に表 CUSTACC が最外部の表になり、ある XML データの索引アクセスを使用して表 CUSTACC にアクセスしなければならないことを指定します。2 番目のガイドライン要素は、XANDOR プランを使用して表 ORDER にアクセスすることを指定します。オプティマイザーは、XANDOR プラン内ですべての適用可能な XML データの索引を使用します。索引の順序はオプティマイザーによって選択されます。

```
<?xml version="1.0" encoding="UTF-8"?>
<OPTPROFILE VERSION="9.7.0.0">
<STMTPROFILE ID="Order and Security Tables">
  <STMTKEY SCHEMA="OST">
     SELECT ordqty, orddate, ordid, security, lasttrade
     FROM order, security, custacc,
       XMLTABLE('$ODOC/FIXML/Order'
          COLUMNS ordid VARCHAR(10) PATH '@ID',
                  orddate date PATH '@TrdDt',
                  ordqty float PATH 'OrdQty/@Qty') AS T1,
       XMLTABLE(' $SDOC/Security'
          COLUMNS security varchar(50) PATH 'Name',
                  lasttrade float PATH 'Price/LastTrade') AS T2
       WHERE XMLEXISTS('
         $SDOC/Security[Symbol/fn:string(.)
             = $ODOC/FIXML/Order/Instrmt/@Sym/fn:string(.)]')
       and XMLEXISTS(
         '$ODOC/FIXML/Order[@Acct/fn:string(.)
              = $CADOC/Customer/Accounts/Account/@id/fn:string(.)]')
        and XMLEXISTS('$CADOC/Customer[@id = 1011]')
       ORDER BY ordqty desc
  </STMTKEY>
  <OPTGUIDELINES>
      <ACCESS TABLE='CUSTACC' TYPE='XMLINDEX' FIRST='TRUE' />
      <XANDOR TABLE='ORDER' />
  </OPTGUIDELINES>
</STMTPROFILE>
</OPTPROFILE>
```

XQuery 式のガイドライン

以下の最適化プロファイルには、表 SECURITY1 からソフトウェア (Software) を扱う会社の在庫情報を戻す XQuery 式が含まれます。このガイドラインは、1 つの XML 索引 XI1 を使用して表にアクセスすることを指定します。

```
<?xml version="1.0" encoding="UTF-8"?>
<OPTPROFILE VERSION="9.7.0.0">
<STMTPROFILE ID="Security Tables by Industry">
  <STMTKEY SCHEMA="STBD">
    xquery
    for $sinfo1 in db2-fn:xmlcolumn("SECURITY1.SDOC")/Security/SecurityInfo
        /StockInfo[Industry="Software"]
    return $sinfo1
  </STMTKEY>
  <OPTGUIDELINES>
    <XISCAN TABLE='SECURITY1' INDEX='XI1'/>
  </OPTGUIDELINES>
</STMTPROFILE>
</OPTPROFILE>
```

以下の最適化プロファイルには、ソフトウェア (Software) と電子機器 (Electronics) を扱う会社の在庫情報を戻す XQuery 式が含まれます。

このガイドラインは、表 SECURITY2 は結合の外部表で、XML データの索引 XI2 を使用して SECURITY2 表にアクセスしなければならないことを指定します。またこのガイドラインは、表 SECURITY1 は結合の内部表で、XML データの索引 XI1 を使用して SECURITY1 表にアクセスしなければならないことも指定します。

```
<?xml version="1.0" encoding="UTF-8"?>
<OPTPROFILE VERSION="9.7.0.0">
<STMTPROFILE ID="Security Tables by Industry">
  <STMTKEY SCHEMA="STBD">
    <![CDATA[ xquery
     for $sinfo1 in db2-fn:xmlcolumn("SECURITY1.SDOC")/Security
```

```
          /SecurityInfo/StockInfo[Industry="Software"]
      for $sinfo2 in db2-fn:xmlcolumn("SECURITY2.SDOC")/Security
          /SecurityInfo/StockInfo[Industry="Electronics"]
      where $sinfo1 = $sinfo2
      return <stock> {$sinfo1} </stock> ]]>
  </STMTKEY>
  <OPTGUIDELINES>
    <JOIN>
      <ACCESS TABLE='SECURITY2' TYPE='XMLINDEX' INDEX='XI2'/>
      <ACCESS TABLE='SECURITY1' TYPE='XMLINDEX' INDEX='XI1'/>
    </JOIN>
  </OPTGUIDELINES>
</STMTPROFILE>
</OPTPROFILE>
```

<![CDATA[で始まり、]]> で終わる CDATA セクションでは、ステートメント・キーを STMTKEY 要素で囲みます。ステートメント・キーには特殊な XML 文字 < と > が含まれるからです。プロファイル・パーサーは CDATA セクション内の XML タグを無視しますが、オプティマイザーはそれでもステートメント・プロファイルをアプリケーション内の対応するステートメントと突き合わせるためにステートメント・キー全体を使用します。

pureXML データ・ストアのパフォーマンスのための DMS 表スペースの設定

パフォーマンス重視のアプリケーション、特に多くの INSERT アクティビティーを行うアプリケーションでは、データベース管理スペース (DMS) 表スペースを使用することを強くお勧めします。

pureXML データ・ストアでの照会パフォーマンスの低下が発生し、システム管理スペース (SMS) を使用している場合、DMS への切り替えを考慮すると良いでしょう。

さらに DMS 機能を使用すると、DB2 におけるオートノミック機能も活用できます。

第 12 章 XML データ・エンコード方式

XML データのエンコードは、データ自体から (*内部的にエンコードされた* と呼びます) か、または外部ソースから (*外部的にエンコードされた* と呼びます) 導出されます。

アプリケーションと XML 列の間での XML データの交換に使用されるアプリケーション・データ・タイプによって、エンコード方式の導出方法が決まります。

- アプリケーション・データ・タイプが文字またはグラフィックの XML データは、外部的にエンコードされると見なされます。これらのデータ・タイプの XML データは、文字データやグラフィック・データと同様に、アプリケーション・コード・ページでエンコードされると見なされます。
- バイナリー・アプリケーション・データ・タイプの XML データまたは文字データ・タイプのバイナリー・データは、内部的にエンコードされると見なされます。

文字データ・タイプの XML 文書がエンコード方式の宣言を含む場合のように、外部的にエンコードされた XML データが内部エンコード方式を含む場合があります。外部的にエンコードされたデータを DB2 データベースに送信する際には、データベース・マネージャーが内部エンコード方式を検査します。

外部エンコード方式および内部エンコード方式が Unicode エンコード方式でない場合、内部エンコード方式に関連した有効な CCSID が、外部エンコード方式と一致していなければなりません。一致していない場合は、エラーが発生します。外部エンコード方式および内部エンコード方式が Unicode エンコード方式であり、コード化スキームが一致していない場合、DB2 データベース・サーバーは内部エンコード方式を無視します。

内部的にエンコードされた XML データ

バイナリー・アプリケーション・データ・タイプの XML データは、内部エンコード方式になります。内部エンコード方式では、データの内容によってエンコード方式が判別されます。DB2 データベース・システムは、XML 規格に従って文書の内容から内部エンコード方式を導出します。

内部エンコード方式は、以下の 3 つのコンポーネントから導出されます。

Unicode バイト・オーダー・マーク (BOM)
XML データの先頭の Unicode 文字コードを構成するバイト・シーケンス。BOM は後続のテキストのバイト・オーダーを示します。DB2 データベース・マネージャーは XML データの BOM のみ認識します。非 XML 列に保管されている XML データの場合、データベース・マネージャーは BOM 値を他の文字やバイナリー値と同様に扱います。

XML 宣言
XML 文書の先頭にある処理命令。この宣言は、XML の残りの部分に関する具体的な詳細情報を提供します。

エンコード方式の宣言
文書中の文字に関するエンコード方式を指定する XML 宣言の任意指定の部分。

DB2 データベース・マネージャーは、以下の手順を使用してエンコード方式を判別します。

1. データに Unicode BOM が含まれている場合は、BOM でエンコード方式が判別されます。以下の表には、BOM タイプとその結果のデータ・エンコードがリストされています。

表 38. バイト・オーダー・マークとその結果の文書エンコード方式

BOM タイプ	BOM 値	エンコード方式
UTF-8	X'EFBBBF'	UTF-8
UTF-16 ビッグ・エンディアン	X'FEFF'	UTF-16
UTF-16 リトル・エンディアン	X'FFFE'	UTF-16
UTF-32 ビッグ・エンディアン	X'0000FEFF'	UTF-32
UTF-32 リトル・エンディアン	X'FFFE0000'	UTF-32

2. データに XML 宣言が含まれている場合は、エンコード方式の宣言があるかどうかに応じてエンコード方式は異なります。
   - エンコード方式の宣言がある場合は、エンコード方式はエンコード方式の属性の値になります。例えば、以下の XML 宣言がある XML データの場合、エンコード方式は EUC-JP です。

     ```
     <?xml version="1.0" encoding="EUC-JP"?>
     ```

   - エンコード方式の宣言と BOM がある場合は、エンコード方式の宣言と BOM からのエンコード方式が一致していなければなりません。一致していない場合は、エラーが発生します。
   - エンコード方式の宣言と BOM がない場合は、データベース・マネージャーは XML 宣言のエンコード方式からエンコード方式を判別します。
     - XML 宣言が 1 バイトの ASCII 文字の場合は、文書のエンコード方式は UTF-8 です。
     - XML 宣言が 2 バイトの ASCII 文字の場合は、文書のエンコード方式は UTF-16 です。
3. XML 宣言と BOM がない場合は、文書のエンコード方式は UTF-8 です。

XML データの保管または引き渡しを行うときのエンコード方式の考慮事項

XML データは、DB2 表に保管するには正しくエンコードされている必要があります。データが表から検索され、DB2 ストアード・プロシージャーまたはユーザー定義関数とともに使用される場合、あるいは外部 Java アプリケーションとともに使用される場合、エンコード方式について考慮する必要があります。

XML データをデータベースに入力する際のエンコード方式に関する考慮事項

XML データを DB2 表に保管する場合には、内部および外部エンコード方式を考慮する必要があります。

以下の規則を守る必要があります。

- 内部エンコード方式および外部エンコード方式が Unicode エンコード方式でない場合、外部的にエンコードされた XML データ (文字データ・タイプを使用してデータベース・サーバーに送信されるデータ) については、内部的にエンコードされた宣言が外部エンコード方式と一致していなければなりません。一致していない場合、エラーが発生し、データベース・マネージャーはその文書を拒否します。

  外部エンコード方式および内部エンコード方式が Unicode エンコード方式であり、コード化スキームが一致していない場合、DB2 データベース・サーバーは内部エンコード方式を無視します。
- 内部的にエンコードされた XML データ (バイナリー・データ・タイプを使用してデータベース・サーバーに送信されるデータ) の場合、データに正確なエンコード方式の情報が含まれていることを*アプリケーションが確実にしなければなりません。*

XML データをデータベースから取り出す際のエンコード方式に関する考慮事項

XML データを DB2 表から取り出す際には、データが損失したり切り捨てられたりしないようにする必要があります。データ損失は、ソース・データの文字をターゲット・データのエンコード方式で表せない場合に生じることがあります。切り捨ては、ターゲット・データ・タイプに変換した結果、データが拡張された場合に生じます。

Java および .NET アプリケーションの方が、他のタイプのアプリケーションよりデータ損失の問題が少なくなります。その理由は、Java および .NET ストリング・データ・タイプは Unicode UTF-16 または UCS-2 エンコード方式を使用しているからです。UTF-8 文字が UTF-16 または UCS-2 エンコード方式に変換される際に拡張が起こることがあるので、切り捨てが生じる可能性があります。

ルーチン・パラメーター内の XML データの引き渡しに関するエンコード方式の考慮事項

DB2 データベース・システムでは、ストアード・プロシージャーまたはユーザー定義関数の定義内のパラメーターに複数の XML データ・タイプを使用できます。

以下の XML データ・タイプが使用できます。

XML　SQL プロシージャーの場合。

XML AS CLOB
外部 SQL プロシージャーおよび外部ユーザー定義関数の場合。

アプリケーションのエンコード方式が UTF-8 でない場合、XML AS CLOB パラメーター内のデータは文字変換の対象になります。外部のユーザー定義関数またはストアード・プロシージャーでの文字変換のオーバーヘッドを避ける必要があります。呼び出し側アプリケーションのパラメーターとして、アプリケーションの文字またはグラフィック型のデータ・タイプはどれでも使用できますが、ソース・データにエンコード方式の宣言が含まれていてはなりません。追加のコード・ページ変換が起こる可能性があり、エンコード方式の情報が不正確になる場合があります。アプリケーション内でデータがさらに構文解析されると、結果としてデータ破壊が起こる可能性があります。

JDBC、SQLJ、および .NET アプリケーション中の XML データのエンコード方式に関する考慮事項

通常、Java アプリケーションの場合の XML エンコード方式に関する考慮事項は、CLI または組み込み SQL アプリケーションの場合より少なくなります。内部的にエンコードされた XML データのエンコード方式に関する考慮事項はすべてのアプリケーションで同じですが、Java アプリケーションにおいて外部的にコード化されたデータの場合はより単純です。その理由は、このアプリケーション・コード・ページは常に Unicode だからです。

Java アプリケーション中の XML データの入力に関する一般的な推奨事項

- 入力データがファイル内にある場合は、データをバイナリー・ストリームとして読み取り (setBinaryStream)、データベース・マネージャーがそのデータを内部的にエンコードされたデータとして処理できるようにします。
- 入力データが Java アプリケーション変数内にある場合は、アプリケーション変数タイプの選択内容によって、DB2 データベース・マネージャーが内部エンコード方式を使用するかどうかが決まります。データを文字タイプとして入力する場合は (setString など)、データベース・マネージャーはデータを保管する前に UTF-16 (アプリケーション・コード・ページ) から UTF-8 に変換します。

Java アプリケーション中の XML データの出力に関する一般的な推奨事項

- XML データを非バイナリー・データとしてファイルに出力する場合は、XML 内部エンコード方式を出力データに追加する必要があります。

  ファイル・システムのエンコード方式は Unicode でない可能性もあるので、ストリング・データはファイル中に保管される際に変換される可能性があります。データをバイナリー・データとしてファイルに書き込む場合には、変換は起こりません。

  Java アプリケーションの場合、データベース・サーバーは、暗黙的な XML シリアライズ操作に関する明示宣言を追加しません。出力データを com.ibm.db2.jcc.DB2Xml タイプとしてキャストし、getDB2Xml*xxx* メソッドの 1 つを呼び出す場合は、以下の表のように、JDBC ドライバーはエンコード方式の宣言を追加します。

getDB2Xml*xxx*	**宣言内のエンコード指定**
getDB2XmlString	ISO-10646-UCS-2

getDB2Xml*xxx*	**宣言内のエンコード指定**
getDB2XmlBytes(String *targetEncoding*)	*targetEncoding* によって指定されるエンコード方式
getDB2XmlAsciiStream	US-ASCII
getDB2XmlCharacterStream	ISO-10646-UCS-2
getDB2XmlBinaryStream(String *targetEncoding*)	*targetEncoding* によって指定されるエンコード方式

INCLUDING XMLDECLARATION を指定した明示的 XMLSERIALIZE 関数の場合、データベース・サーバーはエンコード方式を追加し、JDBC ドライバーはそのエンコード方式を変更しません。データベース・サーバーが追加する明示エンコード方式は UTF-8 エンコード方式です。アプリケーションが値を取り出す方法によっては、データの実際のエンコード方式が明示的な内部エンコード方式と一致しない場合があります。

- アプリケーションが出力データを XML パーサーに送信する場合は、UTF-8、UCS-2、または UTF-16 エンコード方式で、バイナリー・アプリケーション変数中のデータを取り出す必要があります。

データ変換に対する XML エンコード方式およびシリアライゼーションの影響

内部または外部のいずれかで XML データのエンコード方式を指定する方法、および XML シリアライゼーションの方法は、データベースとアプリケーションとの間でデータを渡すときの XML データの変換に影響します。

内部的にエンコードされた XML データをデータベースに入力する場合のエンコード方式のシナリオ

以下の例は、XML データの XML 列への入力中に、内部エンコード方式がデータ変換や切り捨てに影響する様子を示しています。

一般に、バイナリー・アプリケーション・データ・タイプを使用すると、データベースへの入力中のコード・ページ変換の問題を最小限にすることができます。

シナリオ 1

エンコードのソース	**値**
データ・エンコード	UTF-8 Unicode 入力データ (UTF-8 BOM (Byte Order Mark: バイト・オーダー・マーク) または XML エンコードの宣言が含まれる場合も含まれない場合もある)
アプリケーション・データ・タイプ	バイナリー
アプリケーション・コード・ページ	N/A

入力ステートメントの例:

```
INSERT INTO T1 (XMLCOL) VALUES (?)
INSERT INTO T1 (XMLCOL) VALUES
  (XMLPARSE(DOCUMENT CAST(? AS BLOB) PRESERVE WHITESPACE))
```

文字変換: なし。

データ損失: なし。

切り捨て: なし。

シナリオ 2

エンコードのソース	値
データ・エンコード	UTF-16 BOM または XML エンコードの宣言を含む UTF-16 Unicode 入力データ
アプリケーション・データ・タイプ	バイナリー
アプリケーション・コード・ページ	N/A

入力ステートメントの例:

```
INSERT INTO T1 (XMLCOL) VALUES (?)
INSERT INTO T1 (XMLCOL) VALUES
  (XMLPARSE(DOCUMENT CAST(? AS BLOB) PRESERVE WHITESPACE))
```

文字変換: DB2 データベース・サーバーは、XML 列のストレージに関する XML 構文解析を実行する際に、データを UTF-16 から UTF-8 に変換します。

データ損失: なし。

切り捨て: なし。

シナリオ 3

エンコードのソース	値
データ・エンコード	XML エンコードの宣言を含む ISO-8859-1 入力データ
アプリケーション・データ・タイプ	バイナリー
アプリケーション・コード・ページ	N/A

入力ステートメントの例:

```
INSERT INTO T1 (XMLCOL) VALUES (?)
INSERT INTO T1 (XMLCOL) VALUES
  (XMLPARSE(DOCUMENT CAST(? AS BLOB) PRESERVE WHITESPACE))
```

文字変換: DB2 データベース・システムは、XML 列のストレージに関する XML 構文解析を実行する際に、データを CCSID 819 から UTF-8 に変換します。

データ損失: なし。

切り捨て: なし。

シナリオ 4

エンコードのソース	値
データ・エンコード	XML エンコードの宣言を含む Shift_JIS 入力データ
アプリケーション・データ・タイプ	バイナリー
アプリケーション・コード・ページ	N/A

入力ステートメントの例:

```
INSERT INTO T1 (XMLCOL) VALUES (?)
INSERT INTO T1 (XMLCOL) VALUES
  (XMLPARSE(DOCUMENT CAST(? AS BLOB) PRESERVE WHITESPACE))
```

文字変換: DB2 データベース・システムは、XML 列のストレージに関する XML 構文解析を実行する際に、データを CCSID 943 から UTF-8 に変換します。

データ損失: なし。

切り捨て: なし。

外部的にエンコードされた XML データをデータベースに入力する場合のエンコード方式のシナリオ

以下の例は、XML データの XML 列への入力中に、外部エンコード方式がデータ変換や切り捨てに影響する様子を示しています。

一般に、文字アプリケーション・データ・タイプを使用する際には、データベースへの入力中にコード・ページ変換に関する問題は生じません。

Java および .NET アプリケーションのアプリケーション・コード・ページは常に Unicode なので、Java および .NET アプリケーションにはシナリオ 1 とシナリオ 2 のみが適用されます。

シナリオ 1

エンコードのソース	値
データ・エンコード	UTF-8 Unicode 入力データ (該当するエンコード方式の宣言または BOM が含まれる場合も含まれない場合もある)
アプリケーション・データ・タイプ	文字
アプリケーション・コード・ページ	1208 (UTF-8)

入力ステートメントの例:

```
INSERT INTO T1 (XMLCOL) VALUES (?)
INSERT INTO T1 (XMLCOL) VALUES
  (XMLPARSE(DOCUMENT CAST(? AS CLOB) PRESERVE WHITESPACE))
```

文字変換: なし。

データ損失: なし。

切り捨て: なし。

シナリオ 2

エンコードのソース	値
データ・エンコード	UTF-16 Unicode 入力データ (該当するエンコード方式の宣言または BOM が含まれる場合も含まれない場合もある)
アプリケーション・データ・タイプ	グラフィック
アプリケーション・コード・ページ	任意の SBCS コード・ページまたは CCSID 1208

入力ステートメントの例:

```
INSERT INTO T1 (XMLCOL) VALUES (?)
INSERT INTO T1 (XMLCOL) VALUES
  (XMLPARSE(DOCUMENT CAST(? AS DBCLOB) PRESERVE WHITESPACE))
```

文字変換: DB2 データベース・システムは、XML 列のストレージに関する XML 構文解析を実行する際に、データを UTF-16 から UTF-8 に変換します。

データ損失: なし。

切り捨て: 切り捨ては、UTF-16 から UTF-8 への変換中に、拡張のために起こることがあります。

シナリオ 3

エンコードのソース	値
データ・エンコード	ISO-8859-1 入力データ (該当するエンコード方式の宣言が含まれる場合も含まれない場合もある)
アプリケーション・データ・タイプ	文字
アプリケーション・コード・ページ	819

入力ステートメントの例:

```
INSERT INTO T1 (XMLCOL) VALUES (?)
INSERT INTO T1 (XMLCOL) VALUES
  (XMLPARSE(DOCUMENT CAST(? AS CLOB) PRESERVE WHITESPACE))
```

文字変換: DB2 データベース・システムは、XML 列のストレージに関する XML 構文解析を実行する際に、データを CCSID 819 から UTF-8 に変換します。

データ損失: なし。

切り捨て: なし。

シナリオ 4

エンコードのソース	値
データ・エンコード	Shift_JIS 入力データ (該当するエンコード方式の宣言が含まれる場合も含まれない場合もある)
アプリケーション・データ・タイプ	グラフィック
アプリケーション・コード・ページ	943

入力ステートメントの例:

```
INSERT INTO T1 VALUES (?)
INSERT INTO T1 VALUES
  (XMLPARSE(DOCUMENT CAST(? AS DBCLOB)))
```

文字変換: DB2 データベース・システムは、XML 列のストレージに関する XML 構文解析を実行する際に、データを CCSID 943 から UTF-8 に変換します。

データ損失: なし。

切り捨て: なし。

暗黙のシリアライゼーションによって XML データを取り出す際のエンコード方式のシナリオ

以下の例は、暗黙のシリアライゼーションによる XML データの取り出し中に、ターゲットのエンコード方式とアプリケーション・コード・ページがデータ変換、切り捨て、および内部エンコード方式に影響する様子を示しています。

シナリオ 1 とシナリオ 2 は、Java と .NET アプリケーションにのみ適用されます。これは、Java アプリケーションのアプリケーション・コード・ページは常に Unicode であるためです。一般に、Java および .NET アプリケーションでコード・ページ変換が問題になることはありません。

シナリオ 1

エンコードのソース	値
ターゲットのデータ・エンコード	UTF-8 Unicode
ターゲットのアプリケーション・データ・タイプ	バイナリー
アプリケーション・コード・ページ	N/A

出力ステートメントの例:

```
SELECT XMLCOL FROM T1
```

文字変換: なし。

データ損失: なし。

切り捨て: なし。

シリアライズされたデータの内部エンコード方式: Java または .NET アプリケーション以外のアプリケーションの場合、以下の XML 宣言がデータの接頭部になります。

```
<?xml version="1.0" encoding="UTF-8" ?>
```

Java アプリケーションの場合、データを `com.ibm.db2.jcc.DB2Xml` タイプとしてキャストし、`getDB2Xml`*xxx* メソッドを使用してデータを取り出すのでない限り、エンコード宣言は追加されません。追加される宣言は、使用する `getDB2Xml` *xxx* に応じて異なります。

.NET アプリケーションの場合は、エンコード宣言が追加されることも削除されることもありません。

シナリオ 2

エンコードのソース	値
ターゲットのデータ・エンコード	UTF-16 Unicode
ターゲットのアプリケーション・データ・タイプ	グラフィック
アプリケーション・コード・ページ	任意の SBCS コード・ページまたは CCSID 1208

出力ステートメントの例:

```
SELECT XMLCOL FROM T1
```

文字変換: データは UTF-8 から UTF-16 に変換されます。

データ損失: なし。

切り捨て: 切り捨ては、UTF-8 から UTF-16 への変換中に、拡張のために起こることがあります。

シリアライズされたデータの内部エンコード方式: Java または .NET アプリケーション以外のアプリケーションの場合、UTF-16 バイト・オーダー・マーク (BOM) と以下の XML 宣言がデータの接頭部になります。

```
<?xml version="1.0" encoding="UTF-16" ?>
```

Java アプリケーションの場合、データを `com.ibm.db2.jcc.DB2Xml` タイプとしてキャストし、`getDB2Xml`*xxx* メソッドを使用してデータを取り出すのでない限り、エンコード宣言は追加されません。追加される宣言は、使用する `getDB2Xml` *xxx* に応じて異なります。

.NET アプリケーションの場合は、エンコード宣言が追加されることも削除されることもありません。

シナリオ 3

エンコードのソース	値
ターゲットのデータ・エンコード	ISO-8859-1 データ
ターゲットのアプリケーション・データ・タイプ	文字

エンコードのソース	値
アプリケーション・コード・ページ	819

出力ステートメントの例:

```
SELECT XMLCOL FROM T1
```

文字変換: データは UTF-8 から CCSID 819 に変換されます。

データ損失: データ損失は起こる可能性があります。CCSID 819 で表すことができない UTF-8 文字があります。DB2 データベース・システムはエラーを生成します。

切り捨て: なし。

シリアライズされたデータの内部エンコード方式: 以下の XML 宣言がデータの接頭部になります。

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
```

シナリオ 4

エンコードのソース	値
ターゲットのデータ・エンコード	Windows-31J データ (Shift_JIS のスーパーセット)
ターゲットのアプリケーション・データ・タイプ	グラフィック
アプリケーション・コード・ページ	943

出力ステートメントの例:

```
SELECT XMLCOL FROM T1
```

文字変換: データは UTF-8 から CCSID 943 に変換されます。

データ損失: データ損失は起こる可能性があります。CCSID 943 で表すことができない UTF-8 文字があります。DB2 データベース・システムはエラーを生成します。

切り捨て: 切り捨ては、UTF-8 から CCSID 943 への変換中に、拡張のために起こることがあります。

シリアライズされたデータの内部エンコード方式: 以下の XML 宣言がデータの接頭部になります。

```
<?xml version="1.0" encoding="Windows-31J" ?>
```

明示的 XMLSERIALIZE によって XML データを取り出す際のエンコード方式のシナリオ

以下の例は、明示的に XMLSERIALIZE 呼び出しを使用して XML データを取り出すときに、ターゲットのエンコード方式とアプリケーション・コード・ページがデータ変換、切り捨て、および内部エンコード方式に影響する様子を示しています。

シナリオ 1 とシナリオ 2 は、Java と .NET アプリケーションにのみ適用されます。これは、Java アプリケーションのアプリケーション・コード・ページは常に Unicode であるためです。

シナリオ 1

エンコードのソース	値
ターゲットのデータ・エンコード	UTF-8 Unicode
ターゲットのアプリケーション・データ・タイプ	バイナリー
アプリケーション・コード・ページ	N/A

出力ステートメントの例:

```
SELECT XMLSERIALIZE(XMLCOL AS BLOB(1M) INCLUDING XMLDECLARATION) FROM T1
```

文字変換: なし。

データ損失: なし。

切り捨て: なし。

シリアライズされたデータの内部エンコード方式: 以下の XML 宣言がデータの接頭部になります。

```
<?xml version="1.0" encoding="UTF-8" ?>
```

シナリオ 2

エンコードのソース	値
ターゲットのデータ・エンコード	UTF-16 Unicode

エンコードのソース	値
ターゲットのアプリケーション・データ・タイプ	グラフィック
アプリケーション・コード・ページ	任意の SBCS コード・ページまたは CCSID 1208

出力ステートメントの例:

```
SELECT XMLSERIALIZE(XMLCOL AS CLOB(1M) EXCLUDING XMLDECLARATION) FROM T1
```

文字変換: データは UTF-8 から UTF-16 に変換されます。

データ損失: なし。

切り捨て: 切り捨ては、UTF-8 から UTF-16 への変換中に、拡張のために起こることがあります。

シリアライズされたデータの内部エンコード方式: EXCLUDING XMLDECLARATION が指定されているので、なし。INCLUDING XMLDECLARATION が指定されている場合は、内部エンコード方式は UTF-16 の代わりに UTF-8 を示します。この場合、エンコード方式の名前に依存しているアプリケーション・プロセスで XML データを構文解析できなくなる可能性があります。

シナリオ 3

エンコードのソース	値
ターゲットのデータ・エンコード	ISO-8859-1 データ
ターゲットのアプリケーション・データ・タイプ	文字
アプリケーション・コード・ページ	819

出力ステートメントの例:

```
SELECT XMLSERIALIZE(XMLCOL AS CLOB(1M) EXCLUDING XMLDECLARATION) FROM T1
```

文字変換: データは UTF-8 から CCSID 819 に変換されます。

データ損失: データ損失は起こる可能性があります。CCSID 819 で表すことができない UTF-8 文字があります。文字を CCSID 819 で表すことができない場合、DB2 データベース・マネージャーは出力に置換文字を挿入して、警告を発行します。

切り捨て: なし。

シリアライズされたデータの内部エンコード方式: EXCLUDING XMLDECLARATION が指定されているので、なし。INCLUDING XMLDECLARATION が指定されている場合は、データベース・マネージャーは ISO-8859-1 の代わりに UTF-8 の内部エンコード方式を追加します。この場合、エンコード方式の名前に依存しているアプリケーション・プロセスで XML データを構文解析できなくなる可能性があります。

シナリオ 4

エンコードのソース	値
ターゲットのデータ・エンコード	Windows-31J データ (Shift_JIS のスーパーセット)
ターゲットのアプリケーション・データ・タイプ	グラフィック
アプリケーション・コード・ページ	943

出力ステートメントの例:

```
SELECT XMLSERIALIZE(XMLCOL AS CLOB(1M) EXCLUDING XMLDECLARATION) FROM T1
```

文字変換: データは UTF-8 から CCSID 943 に変換されます。

データ損失: データ損失は起こる可能性があります。CCSID 943 で表すことができない UTF-8 文字があります。文字を CCSID 943 で表すことができない場合、データベース・マネージャーは出力に置換文字を挿入して、警告を発行します。

切り捨て: 切り捨ては、UTF-8 から CCSID 943 への変換中に、拡張のために起こることがあります。

シリアライズされたデータの内部エンコード方式: EXCLUDING XMLDECLARATION が指定されているので、なし。INCLUDING XMLDECLARATION が指定されている場合は、内部エンコード方式は Windows-31J の代わりに UTF-8 を示します。この場合、エンコード方式の名前に依存しているアプリケーション・プロセスで XML データを構文解析できなくなる可能性があります。

内部的にエンコードされた XML データと CCSID のマッピング

DB2 データベース・マネージャーは、データを XML データから別のデータ・タイプに変換するときには XML 内部エンコードに基づいて CCSID を判別し、データを XML データ・タイプに変換するときには CCSID に基づいて XML 内部エンコード名を判別します。

エンコード名から保管済み XML データ用の有効な CCSID へのマッピング

XML 列に保管するデータがバイナリー・アプリケーション変数である場合、DB2 データベース・マネージャーはエンコードを判別するためにデータを調べます。データにエンコード宣言がある場合、データベース・マネージャーはエンコード名を CCSID にマップします。

表 39 はこれらのマッピングをリストしています。 表 39 にエンコード名がない場合、データベース・マネージャーはエラーを戻します。

表 39 の最初の列にある正規化されたエンコード名は、エンコード名を大文字に変換し、ハイフン、正符号、下線、コロン、ピリオド、およびスペースをすべて除去した結果です。例えば、ISO88591 は ISO 8859-1、ISO-8859-1、および iso-8859-1 を正規化したエンコード名です。

表 39. エンコード名および有効な CCSID

正規化されたエンコード名	CCSID
437	437
646	367
813	813
819	819
850	850
852	852
855	855
857	857
862	862
863	863
866	866
869	869
885913	901
885915	923
88591	819
88592	912
88595	915
88597	813
88598	62210
88599	920
904	904

表 39. エンコード名および有効な CCSID (続き)

正規化されたエンコード名	CCSID
912	912
915	915
916	916
920	920
923	923
ANSI1251	1251
ANSIX341968	367
ANSIX341986	367
ARABIC	1089
ASCII7	367
ASCII	367
ASMO708	1089
BIG5	950
CCSID00858	858
CCSID00924	924
CCSID01140	1140
CCSID01141	1141
CCSID01142	1142
CCSID01143	1143
CCSID01144	1144
CCSID01145	1145
CCSID01146	1146
CCSID01147	1147
CCSID01148	1148
CCSID01149	1149
CP00858	858
CP00924	924
CP01140	1140
CP01141	1141
CP01142	1142
CP01143	1143
CP01144	1144
CP01145	1145
CP01146	1146
CP01147	1147
CP01148	1148
CP01149	1149
CP037	37
CP1026	1026
CP1140	1140

表 39. エンコード名および有効な CCSID (続き)

正規化されたエンコード名	CCSID
CP1141	1141
CP1142	1142
CP1143	1143
CP1144	1144
CP1145	1145
CP1146	1146
CP1147	1147
CP1148	1148
CP1149	1149
CP1250	1250
CP1251	1251
CP1252	1252
CP1253	1253
CP1254	1254
CP1255	1255
CP1256	1256
CP1257	1257
CP1258	1258
CP1363	1363
CP1383	1383
CP1386	1386
CP273	273
CP277	277
CP278	278
CP280	280
CP284	284
CP285	285
CP297	297
CP33722	954
CP33722C	954
CP367	367
CP420	420
CP423	423
CP424	424
CP437	437
CP500	500
CP5346	5346
CP5347	5347
CP5348	5348
CP5349	5349

表 39. エンコード名および有効な CCSID (続き)

正規化されたエンコード名	CCSID
CP5350	5350
CP5353	5353
CP813	813
CP819	819
CP838	838
CP850	850
CP852	852
CP855	855
CP857	857
CP858	858
CP862	862
CP863	863
CP864	864
CP866	866
CP869	869
CP870	870
CP871	871
CP874	874
CP904	904
CP912	912
CP915	915
CP916	916
CP920	920
CP921	921
CP922	922
CP923	923
CP936	1386
CP943	943
CP943C	943
CP949	970
CP950	950
CP964	964
CP970	970
CPGR	869
CSASCII	367
CSBIG5	950
CSEBCDICCAFR	500
CSEBCDICDKNO	277
CSEBCDICES	284
CSEBCDICFISE	278

表 39. エンコード名および有効な CCSID (続き)

正規化されたエンコード名	CCSID
CSEBCDICFR	297
CSEBCDICIT	280
CSEBCDICPT	37
CSEBCDICUK	285
CSEBCDICUS	37
CSEUCKR	970
CSEUCPKDFMTJAPANESE	954
CSGB2312	1383
CSHPROMAN8	1051
CSIBM037	37
CSIBM1026	1026
CSIBM273	273
CSIBM277	277
CSIBM278	278
CSIBM280	280
CSIBM284	284
CSIBM285	285
CSIBM297	297
CSIBM420	420
CSIBM423	423
CSIBM424	424
CSIBM500	500
CSIBM855	855
CSIBM857	857
CSIBM863	863
CSIBM864	864
CSIBM866	866
CSIBM869	869
CSIBM870	870
CSIBM871	871
CSIBM904	904
CSIBMEBCDICATDE	273
CSIBMTHAI	838
CSISO128T101G2	920
CSISO146SERBIAN	915
CSISO147MACEDONIAN	915
CSISO2INTLREFVERSION	367
CSISO646BASIC1983	367
CSISO88596I	1089
CSISO88598I	916

表 39. エンコード名および有効な CCSID (続き)

正規化されたエンコード名	CCSID
CSISOLATIN0	923
CSISOLATIN1	819
CSISOLATIN2	912
CSISOLATIN5	920
CSISOLATIN9	923
CSISOLATINARABIC	1089
CSISOLATINCYRILLIC	915
CSISOLATINGREEK	813
CSISOLATINHEBREW	62210
CSKOI8R	878
CSKSC56011987	970
CSMACINTOSH	1275
CSMICROSOFTPUBLISHING	1004
CSPC850MULTILINGUAL	850
CSPC862LATINHEBREW	862
CSPC8CODEPAGE437	437
CSPCP852	852
CSSHIFTJIS	943
CSUCS4	1236
CSUNICODE11	1204
CSUNICODE	1204
CSUNICODEASCII	1204
CSUNICODELATIN1	1204
CSVISCII	1129
CSWINDOWS31J	943
CYRILLIC	915
DEFAULT	367
EBCDICATDE	273
EBCDICCAFR	500
EBCDICCPAR1	420
EBCDICCPBE	500
EBCDICCPCA	37
EBCDICCPCH	500
EBCDICCPDK	277
EBCDICCPES	284
EBCDICCPFI	278
EBCDICCPFR	297
EBCDICCPGB	285
EBCDICCPGR	423
EBCDICCPHE	424

表 39. エンコード名および有効な CCSID (続き)

正規化されたエンコード名	CCSID
EBCDICCPIS	871
EBCDICCPIT	280
EBCDICCPNL	37
EBCDICCPNO	277
EBCDICCPROECE	870
EBCDICCPSE	278
EBCDICCPUS	37
EBCDICCPWT	37
EBCDICCPYU	870
EBCDICDE273EURO	1141
EBCDICDK277EURO	1142
EBCDICDKNO	277
EBCDICES284EURO	1145
EBCDICES	284
EBCDICFI278EURO	1143
EBCDICFISE	278
EBCDICFR297EURO	1147
EBCDICFR	297
EBCDICGB285EURO	1146
EBCDICINTERNATIONAL500EURO	1148
EBCDICIS871EURO	1149
EBCDICIT280EURO	1144
EBCDICIT	280
EBCDICLATIN9EURO	924
EBCDICNO277EURO	1142
EBCDICPT	37
EBCDICSE278EURO	1143
EBCDICUK	285
EBCDICUS37EURO	1140
EBCDICUS	37
ECMA114	1089
ECMA118	813
ELOT928	813
EUCCN	1383
EUCJP	954
EUCKR	970
EUCTW	964
EXTENDEDUNIXCODEPACKEDFORMATFORJAPANESE	954
GB18030	1392
GB2312	1383

表 39. エンコード名および有効な CCSID (続き)

正規化されたエンコード名	CCSID
GBK	1386
GREEK8	813
GREEK	813
HEBREW	62210
HPROMAN8	1051
IBM00858	858
IBM00924	924
IBM01140	1140
IBM01141	1141
IBM01142	1142
IBM01143	1143
IBM01144	1144
IBM01145	1145
IBM01146	1146
IBM01147	1147
IBM01148	1148
IBM01149	1149
IBM01153	1153
IBM01155	1155
IBM01160	1160
IBM037	37
IBM1026	1026
IBM1043	1043
IBM1047	1047
IBM1252	1252
IBM273	273
IBM277	277
IBM278	278
IBM280	280
IBM284	284
IBM285	285
IBM297	297
IBM367	367
IBM420	420
IBM423	423
IBM424	424
IBM437	437
IBM500	500
IBM808	808
IBM813	813

表 39. エンコード名および有効な CCSID (続き)

正規化されたエンコード名	CCSID
IBM819	819
IBM850	850
IBM852	852
IBM855	855
IBM857	857
IBM862	862
IBM863	863
IBM864	864
IBM866	866
IBM867	867
IBM869	869
IBM870	870
IBM871	871
IBM872	872
IBM902	902
IBM904	904
IBM912	912
IBM915	915
IBM916	916
IBM920	920
IBM921	921
IBM922	922
IBM923	923
IBMTHAI	838
IRV	367
ISO10646	1204
ISO10646UCS2	1200
ISO10646UCS4	1232
ISO10646UCSBASIC	1204
ISO10646UNICODELATIN1	1204
ISO646BASIC1983	367
ISO646IRV1983	367
ISO646IRV1991	367
ISO646US	367
ISO885911987	819
ISO885913	901
ISO885915	923
ISO885915FDIS	923
ISO88591	819
ISO885921987	912

表 39. エンコード名および有効な CCSID (続き)

正規化されたエンコード名	CCSID
ISO88592	912
ISO885951988	915
ISO88595	915
ISO885961987	1089
ISO88596	1089
ISO88596I	1089
ISO885971987	813
ISO88597	813
ISO885981988	62210
ISO88598	62210
ISO88598I	916
ISO885991989	920
ISO88599	920
ISOIR100	819
ISOIR101	912
ISOIR126	813
ISOIR127	1089
ISOIR128	920
ISOIR138	62210
ISOIR144	915
ISOIR146	915
ISOIR147	915
ISOIR148	920
ISOIR149	970
ISOIR2	367
ISOIR6	367
JUSIB1003MAC	915
JUSIB1003SERB	915
KOI8	878
KOI8R	878
KOI8U	1168
KOREAN	970
KSC56011987	970
KSC56011989	970
KSC5601	970
L1	819
L2	912
L5	920
L9	923
LATIN0	923

表 39. エンコード名および有効な CCSID (続き)

正規化されたエンコード名	CCSID
LATIN1	819
LATIN2	912
LATIN5	920
LATIN9	923
MAC	1275
MACEDONIAN	915
MACINTOSH	1275
MICROSOFTPUBLISHING	1004
MS1386	1386
MS932	943
MS936	1386
MS949	970
MSKANJI	943
PCMULTILINGUAL850EURO	858
R8	1051
REF	367
ROMAN8	1051
SERBIAN	915
SHIFTJIS	943
SJIS	943
SUNEUGREEK	813
T101G2	920
TIS20	874
TIS620	874
UNICODE11	1204
UNICODE11UTF8	1208
UNICODEBIGUNMARKED	1200
UNICODELITTLEUNMARKED	1202
US	367
USASCII	367
UTF16	1204
UTF16BE	1200
UTF16LE	1202
UTF32	1236
UTF32BE	1232
UTF32LE	1234
UTF8	1208
VISCII	1129
WINDOWS1250	1250
WINDOWS1251	1251

表 39. エンコード名および有効な CCSID (続き)

正規化されたエンコード名	CCSID
WINDOWS1252	1252
WINDOWS1253	1253
WINDOWS1254	1254
WINDOWS1255	1255
WINDOWS1256	1256
WINDOWS1257	1257
WINDOWS1258	1258
WINDOWS28598	62210
WINDOWS31J	943
WINDOWS936	1386
XEUCTW	964
XMSWIN936	1386
XUTF16BE	1200
XUTF16LE	1202
XWINDOWS949	970

CCSID とシリアライズされた XML 出力データのエンコード名とのマップ

暗黙的または明示的 XMLSERIALIZE 操作の一部として、DB2 データベース・マネージャーはシリアライズされた XML 出力データの先頭にエンコード宣言を追加する場合があります。

宣言の形式は次のとおりです。

```
<?xml version="1.0" encoding="encoding-name"?>
```

一般に、エンコード宣言の文字セット ID は、文字のエンコードを出力ストリングに記述します。例えば、ターゲット・アプリケーションのデータ・タイプに一致する CCSID に XML データがシリアライズされると、エンコード宣言はターゲット・アプリケーション変数 CCSID を記述します。例外として、アプリケーションが INCLUDING XMLDECLARATION を指定して明示的な XMLSERIALIZE 関数を実行する場合があります。 INCLUDING XMLDECLARATION を指定すると、データベース・マネージャーは UTF-8 用のエンコード宣言を生成します。ターゲットのデータ・タイプが CLOB または DBCLOB タイプの場合、さらにコード・ページの変換が行われることがありますが、これよってエンコード情報が不正確になる可能性があります。アプリケーション内でデータがさらに構文解析されると、結果としてデータ破壊が起こる可能性があります。

可能な場合には、DB2 データベース・マネージャーは、 XML 規格の規定に従って、CCSID に対応する IANA レジストリー名を選択します。

表 40. CCSID とそれに対応するエンコード名

CCSID	エンコード名
37	IBM037

表 40. CCSID とそれに対応するエンコード名 (続き)

CCSID	エンコード名
273	IBM273
277	IBM277
278	IBM278
280	IBM280
284	IBM284
285	IBM285
297	IBM297
367	US-ASCII
420	IBM420
423	IBM423
424	IBM424
437	IBM437
500	IBM500
808	IBM808
813	ISO-8859-7
819	ISO-8859-1
838	IBM-Thai
850	IBM850
852	IBM852
855	IBM855
857	IBM857
858	IBM00858
862	IBM862
863	IBM863
864	IBM864
866	IBM866
867	IBM867
869	IBM869
870	IBM870
871	IBM871
872	IBM872
874	TIS-620
878	KOI8-R
901	ISO-8859-13
902	IBM902
904	IBM904
912	ISO-8859-2
915	ISO-8859-5
916	ISO-8859-8-I
920	ISO-8859-9

表 40. CCSID とそれに対応するエンコード名 (続き)

CCSID	エンコード名
921	IBM921
922	IBM922
923	ISO-8859-15
924	IBM00924
932	Shift_JIS
943	Windows-31J
949	EUC-KR
950	Big5
954	EUC-JP
964	EUC-TW
970	EUC-KR
1004	Microsoft-Publish
1026	IBM1026
1043	IBM1043
1047	IBM1047
1051	hp-roman8
1089	ISO-8859-6
1129	VISCII
1140	IBM01140
1141	IBM01141
1142	IBM01142
1143	IBM01143
1144	IBM01144
1145	IBM01145
1146	IBM01146
1147	IBM01147
1148	IBM01148
1149	IBM01149
1153	IBM01153
1155	IBM01155
1160	IBM-Thai
1161	TIS-620
1162	TIS-620
1163	VISCII
1168	KOI8-U
1200	UTF-16BE
1202	UTF-16LE
1204	UTF-16
1208	UTF-8
1232	UTF-32BE

表 40. CCSID とそれに対応するエンコード名 (続き)

CCSID	エンコード名
1234	UTF-32LE
1236	UTF-32
1250	windows-1250
1251	windows-1251
1252	windows-1252
1253	windows-1253
1254	windows-1254
1255	windows-1255
1256	windows-1256
1257	windows-1257
1258	windows-1258
1275	MACINTOSH
1363	KSC_5601
1370	Big5
1381	GB2312
1383	GB2312
1386	GBK
1392	GB18030
4909	ISO-8859-7
5039	Shift_JIS
5346	windows-1250
5347	windows-1251
5348	windows-1252
5349	windows-1253
5350	windows-1254
5351	windows-1255
5352	windows-1256
5353	windows-1257
5354	windows-1258
5488	GB18030
8612	IBM420
8616	IBM424
9005	ISO-8859-7
12712	IBM424
13488	UTF-16BE
13490	UTF-16LE
16840	IBM420
17248	IBM864
17584	UTF-16BE
17586	UTF-16LE

表 40. CCSID とそれに対応するエンコード名 (続き)

CCSID	エンコード名
62209	IBM862
62210	ISO-8859-8
62211	IBM424
62213	IBM862
62215	ISO-8859-8
62218	IBM864
62221	IBM862
62222	ISO-8859-8
62223	windows-1255
62224	IBM420
62225	IBM864
62227	ISO-8859-6
62228	windows-1256
62229	IBM424
62231	IBM862
62232	ISO-8859-8
62233	IBM420
62234	IBM420
62235	IBM424
62237	windows-1255
62238	ISO-8859-8-I
62239	windows-1255
62240	IBM424
62242	IBM862
62243	ISO-8859-8-I
62244	windows-1255
62245	IBM424
62250	IBM420

第 13 章 アノテーション付き XML スキーマ分解

アノテーション付き XML スキーマ分解 (または「断片化」) は、XML 文書からの内容をリレーショナル表の列内に保管するプロセスのことです。分解は、XML スキーマに指定されたアノテーションに基づいて実行されます。XML 文書の分解後、挿入されたデータは、挿入先の列の SQL データ・タイプになります。

XML スキーマは 1 つ以上の XML スキーマ文書で構成されています。アノテーション付き XML スキーマ分解、つまりスキーマ・ベースの分解では、文書の XML スキーマに分解アノテーションを付けることにより、分解を制御します。これらのアノテーションは、XML データの保管先にするターゲットの表と列の名前、ターゲット表の SQL スキーマが指定されていない場合のデフォルトの SQL スキーマ、XML データをターゲット表に挿入する順序、および保管する前に行うべき内容の変換方法などの詳細情報を指定します。これらのアノテーションを使用して指定できるものに関する、これら以外の例については、分解アノテーションの要約を参照してください。

アノテーション付きスキーマ文書は、XML スキーマ・リポジトリー (XSR) に保管して登録しなければなりません。その後、スキーマを分解に使用できるようにしなければなりません。

アノテーション付きのスキーマを正常に登録し終えたら、分解ストアード・プロシージャーの 1 つを呼び出すか DECOMPOSE XML DOCUMENT コマンドを実行して、単一の XML 文書の分解を実行できます。列に保管されている複数の XML 文書を分解するには、XDB_DECOMP_XML_FROM_QUERY ストアード・プロシージャーまたは DECOMPOSE XML DOCUMENTS コマンドを使用します。

スキーマ・ベースの分解を使用できないようにしたり、作動不能にしたりできることに注意してください。詳しくは、分解を使用不可にすることについて説明したトピックを参照してください。

アノテーション付き XML スキーマ分解の利点

アノテーション付き XML スキーマ分解は、XML スキーマに適合しているが、XML スキーマが文書の保管先の表の定義に容易に一致しないような XML 文書を保管するための解決策となる可能性があります。

XML スキーマが表構造に明確に一致しない場合、文書が表構造に適合するように、XML スキーマ、リレーショナル・スキーマ、またはその両方を調整しなければならないことがあります。しかし、XML またはリレーショナル・スキーマに対する変更はいつでも可能とは限らず、非常に費用のかかる場合もあります。これは特に既存のアプリケーションが、リレーショナル・スキーマが特定の構造に従うことを予期している場合にそう言えます。

アノテーション付き XML スキーマ分解は、新規または既存の XML スキーマに基づく文書を分解して、新規または既存の表に入れられるようにすることにより、この問題に対処します。これが可能なのは、アノテーション付き XML スキーマ分解

にさまざまな機能が用意されているためです。これらの機能は、XML スキーマ文書に追加されるアノテーションの形で表現され、XML スキーマ構造をリレーショナル表構造に柔軟にマッピングできるようにします。

アノテーション付き XML スキーマを使用した XML 文書の分解

1 つ以上の表の列に XML 文書の一部を保管するときには、アノテーション付き XML スキーマ分解を使用できます。この種の分解は、表に保管するために、登録済みのアノテーション付き XML スキーマに指定されるアノテーションに基づいて XML 文書を分解します。

このタスクについて

アノテーション付き XML スキーマを使って XML 文書を分解するには、次のようにします。

手順

1. 以前のバージョンの DB2 データベース製品から作成されたデータベースを使用している場合は、`sqllib/bnd` ディレクトリーにあるリスト・ファイル `xdb.lst` を使用して、BIND コマンドを実行する。
2. XML 分解アノテーションでスキーマ文書にアノテーションを付ける。
3. スキーマ文書を登録し、スキーマの分解ができるようにする。
4. XML スキーマに属する登録済みのスキーマ文書のいずれかが変更された場合、この XML スキーマの文書すべてを再び登録し、XML スキーマの分解ができるようにする必要がある。
5. 分解する XML 文書の場所に応じて、XML スキーマに XSR オブジェクト名を指定し、示されている方法のいずれかを使用する。
   - 単一 XML 文書がファイル・システム上に存在する場合、以下のいずれかの方法を使用します。
     - 分解されている文書のサイズに合うだけ XDBDECOMPXML ストアード・プロシージャーのいずれかを呼び出す。[3]
     - DECOMPOSE XML DOCUMENT コマンドを発行する。
   - 1 つ以上の文書がバイナリーまたは XML 列に保管されている場合、以下のいずれかの方法を使用します。
     - DECOMPOSE XML DOCUMENTS コマンドを発行する。
     - XDB_DECOMP_XML_FROM_QUERY ストアード・プロシージャーを呼び出す。

タスクの結果

XML スキーマを登録し、分解を可能にする

アノテーション付きスキーマが正常に登録され、分解が可能になった後は、XML 文書の分解に使用できます。

3. サイズが不明な文書を複数分解するためにスクリプトまたはアプリケーションを使用している場合は、XDBDECOMPXML ストアード・プロシージャーよりもむしろ DECOMPOSE XML DOCUMENT コマンドの使用による分解を検討してください。そのコマンドを使用すると、文書のサイズに適したストアード・プロシージャーが自動的に呼び出されるからです。

始める前に

- XML スキーマ内の少なくとも 1 つの要素または属性の宣言が、XML 分解アノテーションでアノテーションを付けられているようにします。このアノテーションを付けられた要素または属性は、複合タイプのグローバル要素の子孫、またはそのものでなければなりません。
- XML スキーマを構成するアノテーション付きスキーマ文書のセットで参照される表と列が、すべてデータベースに存在していることを確認します。スキーマで参照される各表と、このスキーマに対応する XSR オブジェクトの間に従属関係が作成されます。
- **applheapsz** 構成パラメーターは、少なくとも 1024 に設定してください。

手順

以下の方法のいずれかを選択して XML スキーマを登録し、分解を可能にします。[4]

- ストアード・プロシージャー:
  1. XSR_REGISTER ストアード・プロシージャーを呼び出し、1 次スキーマ文書に渡す。
  2. XML スキーマが複数のスキーマ文書で構成されている場合は、まだ登録されていないスキーマ文書ごとに XSR_ADDSCHEMADOC ストアード・プロシージャーを呼び出す。
  3. **isusedfordecomposition** パラメーターを 1 に設定して、XSR_COMPLETE ストアード・プロシージャーを呼び出す。
- コマンド行:
  - XML スキーマが 1 つのスキーマ文書によってのみ構成されている場合、**COMPLETE** および **ENABLE DECOMPOSITION** オプションを指定して **REGISTER XML SCHEMA** コマンドを発行する。
  - XML スキーマが複数のスキーマ文書によって構成されている場合には次のようにします。
    1. 最後のスキーマ文書以外のスキーマ文書ごとに、**REGISTER XML SCHEMA** コマンドを発行する。
    2. まだ登録されていない最後のスキーマ文書に対して、**COMPLETE** および **ENABLE DECOMPOSITION** オプションを指定して **REGISTER XML SCHEMA** コマンドを発行する。
- JDBC インターフェース:
  1. DB2Connection.registerDB2XMLSchema メソッドを呼び出し、**isUsedForDecomposition** boolean パラメーターを true に設定して、分解を可能にする。[5]

---

4. これらいずれかのメソッドを使って XML スキーマが以前に登録されたものの、分解が可能になっていない場合は、ENABLE DECOMPOSITION オプションを指定して ALTER XSROBJECT SQL ステートメントを発行することによってスキーマの分解を可能にします。

5. このメソッドは 2 つの形式で存在します。1 つは XML スキーマ文書を `InputStream` オブジェクトから入力する形式で、もう 1 つは XML スキーマ文書を `String` で入力する形式です。

次のタスク

XML スキーマで分解が可能になると、スキーマで参照される各表と、このスキーマに対応する XSR オブジェクトの間に従属関係が作成されます。この従属関係によって、スキーマで参照される表の名前が変更されることを防ぎます。参照される表の名前を変更するには、XML スキーマの XSR オブジェクトの分解を不可にする必要があります。 XSR オブジェクトによって参照される表は、SYSCAT.XSROBJECTDEP カタログ・ビューにあります。

複数の XML 文書の分解の例

DECOMPOSE XML DOCUMENTS コマンドは、単一の XML スキーマを使用して、バイナリーまたは XML 列に保管されている XML 文書のセットを分解します。 XML 文書からのデータは、スキーマで指定されたアノテーションに基づいて、リレーショナル表の列に保管されます。

例

以下の例では、リレーショナル表 ABC.SALESTAB に 2 つの列 SALESDOC および DOCID が含まれると想定します。列 SALESDOC には XML 文書が含まれ、DOCID には SALESDOC に保管されている XML 文書の文書 ID が含まれます。すべての文書は、XML スキーマ・リポジトリー (XSR) に ABC.SALES として登録されている XML スキーマに対応し、スキーマには分解情報でアノテーションが付けられ、分解が可能にされています。スキーマ ABC.SALES を使用して ABC.SALESTAB.SALESDOC に保管されているすべての文書を分解するには、以下の DECOMPOSE XML DOCUMENTS コマンドを呼び出します。

```
DECOMPOSE XML DOCUMENTS IN 'SELECT DOCID, SALESDOC FROM ABC.SALESTAB'
  XMLSCHEMA ABC.SALES
  MESSAGES /home/myid/errors/errorreport.xml
```

あるいは、XDB_DECOMP_XML_FROM_QUERY ストアード・プロシージャーを使用して、XML 文書を分解します。以下のストアード・プロシージャーは、前のコマンドと同じ分解を実行します。

```
XDB_DECOMP_XML_FROM_QUERY ('ABC', 'SALES', 'SELECT DOCID, SALESDOC FROM SALESTAB',
  0, 0, 0, NULL, NULL, 1, numInput, numDecomposed,
  errorreportBuf);
```

以下のコマンドは、スキーマ CUST_SHRED を使用して、CUSTID が 1003 より大きい CUSTOMER 表の INFO の顧客情報を分解します。この例は、スキーマ CUST_SHRED が XSR で登録されていると想定します。

```
DECOMPOSE XML DOCUMENTS IN 'SELECT CUSTID, INFO FROM CUSTOMER WHERE CUSTID > 1003'
  XMLSCHEMA CUST_SHRED
  MESSAGES /home/myid/errors/errorreport.xml
```

アノテーション付き XML スキーマ分解と再帰的 XML 文書

再帰が含まれる XML スキーマは、XML スキーマ・リポジトリー (XSR) に登録して分解を可能にすることができます。ただし、再帰的関係自体はスカラー値としてターゲット表へ分解できないという制限があります。適切なスキーマ・アノテーションを使用することにより、再帰的セクションを保管し、後でシリアライズされたマークアップとして取り出すことができます。

再帰のタイプ

XML スキーマの中のタイプの定義で、同じ名前とタイプの要素が要素自体の定義に出現することが許可される場合、その XML スキーマは再帰的であると言われます。再帰は明示的である場合もありますし、暗黙的である場合もあります。

明示的再帰

明示的再帰は、要素がそれ自体の項で定義される場合に発生します。これを以下の例で示します。この中で、要素 <root> は、ref 要素宣言属性を使用してそれ自体の定義で明示的に参照されています。

```
<xs:element name="root">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="a" type="xs:string"/>
      <xs:element name="b">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="c" type="xs:string"/>
            <xs:element ref="root" minOccurs="0"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

明示的再帰では、再帰的分岐は以下のように区切られます。

- 再帰的分岐の開始は、要素 Y の宣言のうち、その上位に別の Y の要素宣言が含まれないものです。再帰的分岐の開始には複数の下位の分岐が含まれる場合があります。ある下位分岐において、分岐に Y の別の要素宣言が含まれている場合、その分岐は再帰的分岐であると見なされます。
- 再帰的分岐の末尾は、分岐の開始の下位にある Y の最上位の要素宣言です。分岐の末尾は、具体的には要素参照である点に注目してください。

再帰的分岐の開始であるノードは、複数の再帰的分岐の開始ノードとなることができます。以下の例では、明示的な再帰的分岐が 2 つあります。

1. <root> (*), <b>, <root> (**)
2. <root> (*), <b>, <root> (***)

```
<xs:element name="root"> <!-- * -->
  <xs:complexType>
  <xs:sequence>
    <xs:element name="a" type="xs:string"/>
    <xs:element name="b">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="c" type="xs:string"/>
        <xs:element ref="root" minOccurs="1"/>  <!-- ** -->
        <xs:element ref="root" minOccurs="1"/>  <!-- *** -->
      </xs:sequence>
    </xs:complexType>
    </xs:element>
  </xs:sequence>
  </xs:complexType>
</xs:element>
```

再帰的分岐は、そのメンバー要素が分解される方法を示します。インスタンス文書では、再帰的分岐の開始およびその下位に対応する要素 Y が現れる位置から、その分岐の末尾に対応する Y が現れる位置までを、スカラー値

として分解できます。再帰的分岐の末尾に対応するインスタンス文書の中の Y が現れる位置は、再帰的領域を表します。再帰的領域は、この Y が現れる位置の開始要素タグで始まり、その出現位置の終了要素タグで終わります。この再帰的領域にあるインスタンス文書内のすべての要素および属性は、マークアップまたはストリング値として分解できます。どちらになるかは、db2-xdb:contentHandling 分解アノテーションで指定された値によって異なります。

暗黙的再帰

暗黙的再帰は、複合タイプ定義を持つ要素に複合タイプとして定義された別の要素が含まれ、後者の要素のタイプ属性として、その要素を含んでいる複合タイプ定義の名前を持つ場合に発生します。これを以下の例で示します。この中で、要素 <beginRecursion> は、タイプ「rootType」を参照し、要素 <beginRecursion> はそれ自体が、定義されているタイプ「rootType」の一部です。

```
<xs:element name="root" type="rootType"/>
<xs:complexType name="rootType">
  <xs:sequence>
  <xs:element name="a" type="xs:string"/>
    <xs:element name="b">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="c" type="xs:string"/>
          <xs:element name="beginRecursion" type="rootType" minOccurs="0"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:complexType>
```

暗黙的再帰では、再帰的分岐は以下のように区切られます。

- 再帰的分岐の開始は、complexType タイプ CT の要素 Y の宣言のうち、その上位にタイプ CT の別の要素宣言が含まれないものです。再帰的分岐の開始には複数の下位の分岐が含まれる場合があります。ある下位の分岐において、分岐にタイプ CT の Z の別の要素宣言が含まれている場合、その分岐は再帰的分岐であると見なされます。
- 再帰的分岐の末尾は、分岐の開始の下位にあるタイプ CT の最上位の要素宣言です。

再帰的分岐の開始であるノードは、複数の再帰的分岐の開始ノードとなることができます。以下の例では、暗黙的な再帰的分岐が 2 つあります。

1. <root>, <b>, <beginRecursion>
2. <root>, <b>, <anotherRecursion>

```
<xs:element name="root" type="rootType"/>
<xs:complexType name="rootType">
  <xs:sequence>
  <xs:element name="a" type="xs:string"/>
  <xs:element name="b">
    <xs:complexType>
    <xs:sequence>
      <xs:element name="c" type="xs:string"/>
      <xs:element name="beginRecursion" type="rootType" minOccurs="2"/>
      <xs:element name="anotherRecursion" type="rootType" minOccurs="0"/>
    </xs:sequence>
```

```
    </xs:complexType>
  </xs:element>
  </xs:sequence>
</xs:complexType>
```

明示的再帰と比較すると、この 2 番目の暗黙的な再帰のタイプの分解方法には、多少の相違があります。インスタンス文書では、再帰的分岐の開始およびその下位に対応する要素 Y が現れる位置から、その分岐の末尾に対応する Z が現れる位置までを、スカラー値として分解できます。インスタンス文書におけるこの Z が現れる位置は、再帰的領域を表します。再帰的領域は、Z の開始要素タグの*後で* 始まり、Z の終了要素タグのすぐ*前で* 終わります。この Z が現れる位置のすべての下位要素は、この再帰的領域に含まれます。ただし、この出現位置の属性は再帰的領域の外側にあるため、スカラー値として分解できます。

再帰的分岐の分解動作

両方のタイプの再帰において、再帰的分岐はインスタンス文書の対応する部分における非再帰的領域と再帰的領域の区別をします。 XML インスタンス文書の非再帰的領域のみを、ターゲット・データベース表へスカラー値として分解できます。この制限として、再帰的領域の囲みの中の一部となる、いくつかの非再帰的領域が含まれます。つまり、再帰的分岐 RB2 が再帰的分岐 RB1 で完全に囲まれている場合、そのインスタンス XML 文書の RB2 の一部のインスタンスにおいては、その非再帰的領域が RB1 のインスタンスの再帰的領域の内側に含まれる場合もあります。この場合、この非再帰的領域はスカラー値として分解できず、代わりに RB1 の分解結果であるマークアップの一部となります。RB2 のすべてのインスタンスにおいて、他のどの再帰的領域の内側にもないインスタンスの非再帰的領域のみをスカラー値として分解できます。

例えば、以下の XML スキーマは 2 つの再帰的分岐を含みます。

1. RB1 (<root> (identified with *), <b>, <root> (identified with **))
2. RB2 (<d>, <d>)

```
<xs:element name="d">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="d">
    </xs:sequence>
    <xs:attribute name="id" type="xs:int"/>
  </xs:complexType>
</xs:element>
<xs:element name="root"> <!-- * -->
  <xs:complexType>
  <xs:sequence>
    <xs:element name="a" type="xs:string"/>
       <xs:element ref="d"/>
    <xs:element name="b">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="c" type="xs:string"/>
        <xs:element ref="root" minOccurs="1"/>  <!-- ** -->
      </xs:sequence>
    </xs:complexType>
    </xs:element>
  </xs:sequence>
  </xs:complexType>
</xs:element>
```

関連したインスタンス文書の再帰的領域は、次の例の中で強調表示されています。このインスタンス文書には RB2 の 2 つのインスタンス (<d>、<d>) がありますが、RB2 の最初のインスタンスの非再帰的領域 (# で識別される <d>) のみをスカラー値として分解できます。つまり、属性 id="1" は分解できます。 RB2 の 2 番目のインスタンスの非再帰的領域は、RB1 のインスタンスの再帰的領域である、2 番目の強調表示された領域の中に完全に入ります。そのため、属性 id="2" は分解できません。

```
<root>
  <a>a str1</a>
  <d id="1"> <d id="11"> </d> </d>
  <b>
    <c>c str1</c>
    <root>
      <a>a str11</a>
      <d id="2"> <d id="22"> </d> </d>
      <b>
      <c>c str11</c>
      </b>
    </root>
  </b>
</root>
```

例: 再帰の両方のタイプでの db2-xdb:contentHandling 分解アノテーションの使用

この例は、明示的と暗黙的の両方のタイプの再帰における分解動作と、db2-xdb:contentHandling アノテーションに異なる値を設定した結果を示しています。以下の 2 つの XML インスタンス文書の中で、再帰的領域は強調表示されています。

文書 1 では、<root> 要素がそれ自体の下位に現れたときに再帰が始まります。

```
<root>
  <a>a str1</a>
  <b>
    <c>c str1</c>
    <root>
      <a>a str11</a>
      <b>
        <c>c str11</c>
      </b>
    </root>
  </b>
</root>
```

文書 2 では、再帰は、要素 <beginRecursion> の下位の要素で始まります。

```
<root>
  <a>a str2</a>
  <b>
    <c>c str2</c>
    <beginRecursion>
      <a>a str22</a>
      <b>
        <c>c str22</c>
      </b>
    </beginRecursion>
  </b>
</root>
```

インスタンス文書において、再帰の先頭と再帰の終了の間に出現するすべての要素または属性、およびそれらの内容は、スカラー値として表と列のペアに分解することはできません。ただし、再帰の先頭と再帰の終了の間の項目のシリアライズされたマークアップ・バージョンは、再帰的分岐の (complexType である) 要素に、db2-xdb:contentHandling 属性を「serializeSubtree」に設定してアノテーションを付けることにより取得できます。この部分のすべての文字データをテキストへシリアライズしたものは、db2-xdb:contentHandling を「stringValue」に設定することによっても取得できます。一般的に、再帰的パスの内容またはマークアップは、再帰的分岐のいずれかの complexType の要素または再帰的分岐の要素の上位にある要素において、db2-xdb:contentHandling 属性を適切に設定することにより取得できます。

例えば、以下の XML スキーマにおいて要素 <b> にアノテーションを付ける場合を考えます。

```
<xs:element name="root">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="a" type="xs:string"/>
      <xs:element name="b"
             db2-xdb:rowSet="TABLEx"
             db2-xdb:column="COLx"
             db2-xdb:contentHandling="serializeSubtree">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="c" type="xs:string"/>
            <xs:element ref="root" minOccurs="0"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

このようにすると、文書 1 が分解されるときにこの XML フラグメントが TABLEx、COLx の行に挿入されます。

```
<b>
  <c>c str1</c>
  <root>
    <a>a str11</a>
    <b>
      <c>c str11</c>
    </b>
  </root>
</b>
```

同様に、以下の XML スキーマにおいて要素「beginRecursion」にアノテーションを付ける場合を考えます。

```
<xs:element name="root" type="rootType"/>
<xs:complexType name="rootType">
  <xs:sequence>
    <xs:element name="a" type="xs:string"/>
    <xs:element name="b">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="c" type="xs:string"/>
          <xs:element name="beginRecursion"
            type="rootType" minOccurs="0"
            db2-xdb:rowSet="TABLEx"
            db2-xdb:column="COLx"
            db2-xdb:contentHandling="serializeSubtree"/>
```

```
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:complexType>
```

このようにすると、文書 2 が分解されるときにこの XML フラグメントが TABLEx、COLx の行に挿入されます。

```
<beginRecursion>
  <a>a str22</a>
  <b>
    <c>c str22</c>
  </b>
</beginRecursion>
```

アノテーション付き XML スキーマ分解の使用不可化

アノテーション付き XML スキーマ分解は、特定の条件のもとで DB2 によって作動不能にされることがあります。また、ユーザーによって明示的に使用できないようにすることもできます。

分解が作動不能になる場合の条件

以前に登録され、分解に使用できるようにしたアノテーション付きスキーマの場合、以下のいずれかの条件が満たされると、スキーマ・ベースの分解は自動的に作動不能になります。(分解が作動不能になった XML スキーマは、XMLVALIDATE SQL/XML 関数を使用する場合など、分解のコンテキスト以外で実行される妥当性検査には引き続き使用できることに注意してください。)再び分解に使用できるようにするのに必要な修正アクションが、条件ごとにリストされています。

表 41. 分解を作動不能にする条件と、対応する修正アクション

条件	再び分解に使用できるようにするためのアクション
アノテーションで参照されている表がドロップされる	ドロップされた表に対する参照をスキーマ文書から除去し、アノテーション付きスキーマ全体を再登録して、スキーマを分解に使用できるようにする
アノテーションで参照されている列のデータ・タイプが、XML スキーマ・タイプと互換性のあるタイプに変更される	ENABLE DECOMPOSITION オプションを指定して ALTER XSROBJECT SQL ステートメントを実行し、再びスキーマを分解に使用できるようにする
アノテーションで参照されている列のデータ・タイプが、XML スキーマ・タイプと互換性のないタイプに変更される	必要に応じてアノテーションを調整し、アノテーション付きスキーマ全体を再登録して、スキーマを分解に使用できるようにする
アノテーション付きのスキーマに属する文書が変更される	このスキーマを構成するすべての文書を再登録し、スキーマを分解に使用できるようにする

詳しくは、アノテーション付きスキーマの登録と分解の使用可能化に関するタスク文書を参照してください。

明示的な使用不可化

使用不可としたいアノテーション付きスキーマに対応する XSR オブジェクトを指定して、以下のいずれかの SQL ステートメントを実行することにより、スキーマ・ベースの分解を明示的に使用できないようにすることができます。

- DISABLE DECOMPOSITION オプションを指定した ALTER XSROBJECT

  注: 分解に使用できないようにした XML スキーマでも、妥当性検査には引き続き使用できます。

- XSROBJECT オプションを指定した DROP

注: どちらの方法を選択するかは、XML スキーマを何のために必要としているかに応じて変わります。妥当性検査にスキーマが必要な場合は、ドロップするのではなく、分解に使用できないようにする必要があります。スキーマが分解専用で、再び分解に使用することを期待していない場合は、XSR オブジェクトをドロップできます。

アノテーション付きスキーマ分解のための xdbDecompXML プロシージャー

単一 XML 文書を分解する次の 10 の組み込みプロシージャーのいずれかを呼び出すことによって、アノテーション付き XML スキーマ分解が呼び出されます。

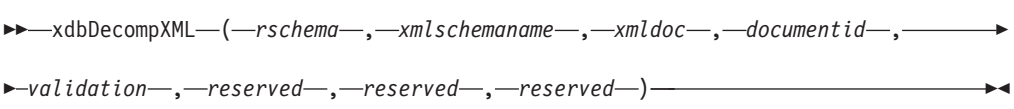
アノテーション付き XML スキーマ分解のプロシージャー

- xdbDecompXML
- xdbDecompXML10MB
- xdbDecompXML25MB
- xdbDecompXML50MB
- xdbDecompXML75MB
- xdbDecompXML100MB
- xdbDecompXML500MB
- xdbDecompXML1GB
- xdbDecompXML1_5GB
- xdbDecompXML2GB

これらのプロシージャーの違いは *xmldoc* 引数のサイズだけです。この引数は、分解される入力文書のサイズを指定します。システム・メモリーの使用を最小限に抑えるため、分解する文書のサイズに合うだけのプロシージャーを呼び出してください。例えば、1 MB の文書を分解するには、xdbDecompXML プロシージャーを使用します。

以下のセクションに xdbDecompXML の構文が示されています。*xmldoc* 引数の仕様については、
xdbDecompXML10MB、xdbDecompXML25MB、xdbDecompXML50MB、xdbDecompXML75MB、xdbDecompXML100MB、xdbDecompXML500MB、xdbDecompXML1GB、xdbDecompXML1_5GB、および xdbDecompXML2GB プロシージャーの *xmldoc* 引数の説明を参照してください。

構文



プロシージャーのスキーマは SYSPROC です。

許可

このプロシージャーを呼び出すステートメントに属する許可 ID には、以下のいずれかの特権または権限が必要です。

- 以下のすべての特権:
  - アノテーション付きスキーマで参照されるターゲット表すべてに対する INSERT 特権
  - db2-xdb:expression または db2-xdb:condition アノテーションによって参照される表すべてに対する SELECT、INSERT、UPDATE、または DELETE 特権 (適用可能な場合)
- アノテーション付きスキーマ文書のセットで参照されるターゲット表すべてに対する CONTROL 特権
- DATAACCESS 権限

validation の値が 1 である場合、このプロシージャーを呼び出すステートメントに属する許可 ID には XML スキーマに対する USAGE 特権が必要です。

プロシージャーのパラメーター

rschema
: XML スキーマ・リポジトリーに登録される 2 つの部分から成る XSR オブジェクト名の SQL スキーマの部分を指定する、VARCHAR(128) タイプの入力引数。この値が NULL の場合、SQL スキーマの部分は CURRENT SCHEMA 特殊レジスターの現行値と見なされます。

xmlschemaname
: XML スキーマ・リポジトリーに登録される 2 つの部分から成る XSR オブジェクト名のスキーマ名を指定する、VARCHAR(128) タイプの入力引数。この値を NULL にすることはできません。

xmldoc
: 分解される XML 文書を含むバッファーを指定する、BLOB(1M) タイプの入力引数。

  注:

  - xdbDecompXML10MB プロシージャーの場合、この引数のタイプは BLOB(10M) です。
  - xdbDecompXML25MB プロシージャーの場合、この引数のタイプは BLOB(25M) です。
  - xdbDecompXML50MB プロシージャーの場合、この引数のタイプは BLOB(50M) です。

- xdbDecompXML75MB プロシージャーの場合、この引数のタイプは BLOB(75M) です。
- xdbDecompXML100MB プロシージャーの場合、この引数のタイプは BLOB(100M) です。
- xdbDecompXML500MB プロシージャーの場合、この引数のタイプは BLOB(500M) です。
- xdbDecompXML1GB プロシージャーの場合、この引数のタイプは BLOB(1G) です。
- xdbDecompXML1_5GB プロシージャーの場合、この引数のタイプは BLOB(1.5G) です。
- xdbDecompXML2GB プロシージャーの場合、この引数のタイプは BLOB(2G) です。

documentid
分解される入力 XML 文書の ID を指定する、VARCHAR(1024) タイプの入力引数。対応する XML スキーマの db2-xdb:expression または db2-xdb:condition アノテーションで指定される $DECOMP_DOCUMENTID が使用される場合には常に、この引数に指定される値に置き換えられます。

validation
文書を分解する前に妥当性検査を実行するかどうかを指定する、INTEGER タイプの入力引数。使用できる値は次のとおりです。

0 入力文書を分解する前に妥当性検査は実行されません。

1 以前 XML スキーマ・リポジトリーに登録された DTD または XML スキーマ文書に対する入力文書に対して妥当性検査が実行されます。入力 XML 文書の分解は、妥当性検査が成功する場合にのみ行われます。

reserved
reserved 引数は、将来の利用のために予約されている入力引数です。これらの引数に渡される値は `NULL` でなければなりません。

出力

このプロシージャーには明示的な出力引数はありません。 SQLCA 構造の sqlcode フィールドを調べ、分解中に発生した可能性のあるエラーがないか調べます。分解の完了後に起こりうる sqlcode 値は次のとおりです。

0 文書は正常に分解されました。

正の整数
文書は正常に分解されましたが、警告条件付きです。警告は **db2diag** ログ・ファイルに記録されます。このファイルは、First Occurrence Data Capture (FODC) 保管ディレクトリーにあります。

負の整数
文書を分解することができませんでした。 sqlcode は失敗の理由を示します。失敗の詳細については、**db2diag** ログ・ファイルを調べてください。

使用上の注意

プロシージャーは読み取り固定分離レベルで実行されます。

プロシージャーはアトミックに実行されます。つまり、実行中にプロシージャーが失敗すると、プロシージャーによって実行される操作はすべてロールバックされます。プロシージャーによって行われる変更をコミットするには、呼び出し元は COMMIT SQL ステートメントを実行する必要があります。プロシージャーそのものは COMMIT を実行しないからです。

サイズが不明な文書を複数分解するためにスクリプトまたはアプリケーションを使用している場合は、xdbDecompXML プロシージャーよりもむしろ **DECOMPOSE XML DOCUMENT** コマンドの使用による分解を検討してください。そのコマンドを使用すると、文書のサイズに適したプロシージャーが自動的に呼び出されるからです。

例

XML 値の分解先である表と列を示す、アノテーション付き XML スキーマが、ABC.TEST として XML スキーマ・リポジトリーに登録されています。XML 文書をリレーショナル表に分解するには、以下に示すように適切なプロシージャーを呼び出します。

```
CALL xdbDecompXML ('ABC', 'TEST', BLOB('<Element1>Hello world</Element1>'),
     'DOCID', 0 , NULL, NULL, NULL)
```

DECOMPOSE XML DOCUMENT

ストアード・プロシージャーを呼び出し、登録済みの分解可能 XML スキーマを使用して、単一の XML 文書を分解します。

許可

以下のグループの特権または権限のいずれかが必要です。

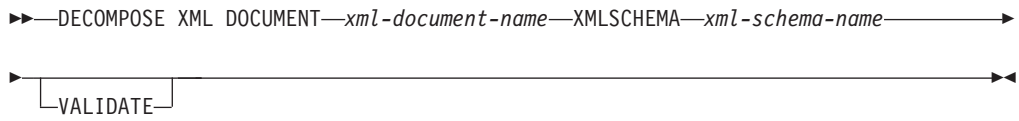
- 以下のいずれかの権限が必要です。
  - アノテーション付きスキーマ文書のセットで参照されるターゲット表すべてに対する CONTROL 特権
  - DATAACCESS 権限
- 以下の特権すべて:
  - ターゲット表に対する INSERT 特権 (アクション・ファイルで指定される操作に必要)
  - db2-xdb:expression または db2-xdb:condition アノテーションによって参照される表すべてに対する SELECT、INSERT、UPDATE、または DELETE 特権 (適用可能な場合)

VALIDATE オプションを指定すると、XML スキーマにおける USAGE 特権も必要になります。

必要な接続

データベース

コマンド構文



コマンド・パラメーター

DECOMPOSE XML DOCUMENT *xml-document-name*

xml-document-name は、分解される入力 XML 文書のファイル・パスおよびファイル名です。

XMLSCHEMA *xml-schema-name*

xml-schema-name は、文書の分解に使用される、XML スキーマ・リポジトリーに登録された既存の XML スキーマの名前です。 *xml-schema-name* は修飾 SQL ID で、オプションの SQL スキーマ名の後にピリオドと XML スキーマ名が続く形で構成されます。 SQL スキーマ名が指定されない場合、DB2 特殊レジスター CURRENT SCHEMA の値であると想定されます。

VALIDATE

このパラメーターは、入力 XML 文書が最初に妥当性検査され、その文書が有効な場合に限り、分解されることを示します。 **VALIDATE** が指定されない場合、入力 XML 文書は分解前に妥当性検査されません。

例

以下の例は、XML 文書 `./gb/document1.xml` が、登録済み XML スキーマ DB2INST1.GENBANKSCHEMA を使用して妥当性検査され、分解されることを指定します。

```
DECOMPOSE XML DOCUMENT ./gb/document1.xml
              XMLSCHEMA DB2INST1.GENBANKSCHEMA
              VALIDATE
```

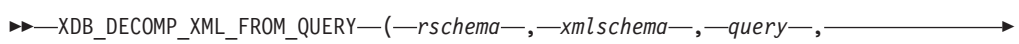
以下の例は、XML 文書 `./gb/document2.xml` が、妥当性検査されずに、登録済み XML スキーマ DB2INST2."GENBANK SCHEMA1" を使用して分解されることを指定します。このとき、DB2 特殊レジスター CURRENT SCHEMA の値が DB2INST2 に設定されていることを想定しています。

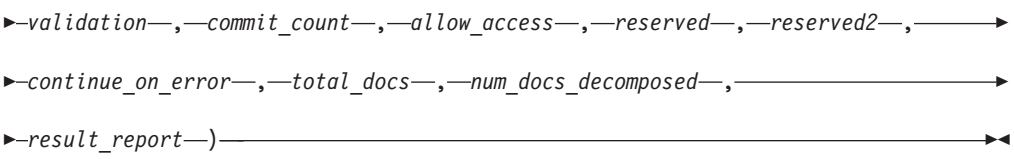
```
DECOMPOSE XML DOCUMENT ./gb/document2.xml
              XMLSCHEMA "GENBANK SCHEMA1"
```

アノテーション付きスキーマ分解に関する XDB_DECOMP_XML_FROM_QUERY ストアード・プロシージャー

ストアード・プロシージャーは、バイナリーまたは XML 列から 1 つ以上の XML 文書を分解します。 XML 文書からのデータは、XML スキーマで指定されたアノテーションに基づいて、リレーショナル表の列に保管されます。

構文





ストアード・プロシージャーのスキーマは SYSPROC です。

プロシージャーは読み取り固定分離レベルで実行されます。

許可

以下のいずれかの権限または特権が必要です。

- 以下のすべての特権:
  - アノテーション付きスキーマで参照されるターゲット表すべてに対する INSERT 特権
  - 入力文書を保持する列を含む表、別名、またはビューに対する SELECT 特権
  - db2-xdb:expression または db2-xdb:condition アノテーションによって参照される表すべてに対する SELECT、INSERT、UPDATE、または DELETE 特権 (適用可能な場合)
- アノテーション付きスキーマ文書のセットで参照される表すべてに対する CONTROL 特権、および入力文書を保持する列を含む表、別名、またはビューに対する CONTROL 特権
- DATAACCESS 権限

validation の値が 1 である場合、XML スキーマに対する USAGE 特権も必要です。

プロシージャーのパラメーター

rschema
: XML スキーマ・リポジトリーに登録される 2 つの部分から成る XML スキーマ・リポジトリー (XSR) オブジェクト名の SQL スキーマの部分を指定する、VARCHAR(128) タイプの入力引数。この値は NULL にすることができます。この値が NULL の場合、SQL スキーマの部分は CURRENT SCHEMA 特殊レジスターの現行値と見なされます。

xmlschema
: XSR に登録される 2 つの部分から成る XSR オブジェクト名の名を指定する、VARCHAR(128) タイプの入力引数。この値を NULL にすることはできません。

query
: タイプ CLOB(1MB) の入力引数。この値を NULL にすることはできません。*query* は SQL SELECT ステートメントの規則に準拠しており、2 つの列が含まれる結果セットを戻す必要があります。最初の列は文書 ID です。各文書 ID は、分解される XML 文書を一意的に識別します。列は、文字タイプであるか、あるいは文字タイプにキャスト可能でなければなりません。 2 番目の列には、分解される XML 文書が含まれます。文書列でサポートされるタイプは、XML、BLOB、VARCHAR FOR BIT DATA、および LONG VARCHAR FOR

BIT DATA です。 XML 文書を含む列は、基礎となる基本表の列に解決される必要があります。その列は生成された列であってはなりません。

例えば、以下の SELECT ステートメントの DOCID 列には、SALESDOC 列に保管されている XML 文書のユニーク ID が含まれます。

```
SELECT DOCID, SALESDOC FROM SALESTAB
```

validation
文書を分解する前に妥当性検査を実行するかどうかを指定する、INTEGER タイプの入力引数。使用できる値は次のとおりです。

0 入力文書を分解する前に妥当性検査は実行されません。

値 0 が渡され、妥当性検査が実行されない場合、ストアード・プロシージャーを呼び出す前の文書の妥当性検査はユーザーの責任で行います。例えば、ユーザーは、XML 文書を列に挿入する場合に XMLVALIDATE を使用することや、文書を挿入する前に XML プロセッサーを使用することができます。入力 XML 文書が無効であり、このパラメーターに 0 が指定されている場合、分解結果は未定義です。

1 以前に XML スキーマ・リポジトリーに登録された DTD または XML スキーマ文書に対する入力文書に照らして、妥当性検査が実行されます。入力 XML 文書の分解は、妥当性検査が成功した場合にのみ行われます。

commit_count
タイプ INTEGER の入力引数。使用できる値は次のとおりです。

0 ストアード・プロシージャーによって COMMIT が実行されることはありません。

***n* (正整数)**
文書が正常に *n* 回分解されるごとに、COMMIT が実行されます。

allow_access
タイプ INTEGER の入力引数。使用できる値は次のとおりです。

0 ストアード・プロシージャーは、XML スキーマでマッピングを使用して、すべての表に対する排他的ロック (X) を取得します。各文書の分解時に必ずしもすべての表が関与するわけではありませんが、長い作業単位におけるデッドロックの可能性を減らすためにすべてのターゲット表がロックされます。

1 ロックを取得する場合、ストアード・プロシージャーは待機し、タイムアウトになる場合もあります。

reserved
reserved 引数は、将来の利用のために予約されている入力引数です。この引数に渡される値は NULL でなければなりません。

reserved2
reserved2 引数は、将来の利用のために予約されている入力引数です。この引数に渡される値は NULL でなければなりません。

continue_on_error
タイプ INTEGER の入力引数。使用できる値は次のとおりです。

0 ストアード・プロシージャーは、正常に分解できなかった最初の文書で停止

します。文書の分解時にエラーが発生した場合、文書の分解時に行われたデータベースへの変更は取り消されます。

1 ストアード・プロシージャーは、文書固有のエラーでは停止せず、*query* で指定されたすべての文書の分解を試行します。文書の分解時にエラーが発生した場合、文書の分解時に行われたデータベースへの変更は取り消され、ストアード・プロシージャーは次の文書の分解を試行します。正常に分解されなかった文書に関する情報は *result_report* に書き込まれます。

ストアード・プロシージャーは、*continue_on_error* の値にかかわらず、致命的エラーおよび文書固有でないエラーが起きると実行を継続しません。

total_docs
XDB_DECOMP_XML_FROM_QUERY ストアード・プロシージャーが分解を試行した入力文書の総数を指定する、タイプ INTEGER の出力引数。

num_docs_decomposed
正常に分解された文書の数を指定する、タイプ INTEGER の出力引数。

result_report
タイプ BLOB(100MB) の出力引数。正常に分解されなかった各入力ファイルの名前を診断メッセージとともにリストする UTF-8 XML 文書を含むバッファー。このレポートは、正常に分解できなかった XML 文書が少なくとも 1 つある場合にのみ生成されます。

result_report の XML 文書の形式は以下のとおりです。

```
<?xml version='1.0'?>
<xdb:errorReport xmlns:xdb="http://www.ibm.com/xmlns/prod/db2/xdb1">
   <xdb:document>
      <xdb:documentId>sssss</xdb:documentId>
      <xdb:errorMsg>qqqqq</xdb:errorMsg>
   </xdb:document>
   <xdb:document>
      . . .
   </xdb:document>
   . . .
</xdb:errorReport>
```

documentId 値 *sssss* は、*query* で指定された最初の列からの値です。この値は、正常に分解されなかった XML 文書を識別します。 errorMsg 値 *qqqqq* は、文書を分解しようとしたときに検出されたエラーです。

出力

SQLCA 構造は、XML 文書を分解しようとした後のプロシージャーの戻り状況を示します。プロシージャーは、以下のいずれかの SQLCODE 値を戻すことができます。

0 *query* で指定されたすべての文書は正常に分解されました。

16278
1 つ以上の文書の分解が失敗しました。正常に分解された文書の数は、ストアード・プロシージャーへの *num_docs_decomposed* 出力パラメーターとして指定されています。失敗した各文書の個別のエラー・メッセージは、*result_report* に記録されます。それぞれの失敗に関する診断の詳細は、**db2diag** ログ・ファイルに記録されます。

負の整数
文書は分解されませんでした。SQLCODE は失敗の理由を示します。失敗の詳細については、**db2diag** ログ・ファイルを調べてください。

注

ストアード・プロシージャーは次の特性を持つように宣言されます。

```
DYNAMIC RESULT SETS 0
NOT DETERMINISTIC
UNFENCED
THREADSAFE
MODIFIES SQL DATA
PARAMETERSTYLE SQL
CALLED ON NULL INPUT
NEW SAVEPOINT LEVEL
DBINFO
```

例

以下の例では、表 ABC.SALESTAB に 2 つの列 SALESDOC および DOCID が含まれると想定しています。列 SALESDOC には XML 文書が含まれ、列 DOCID には SALESDOC にある XML 文書のユニーク ID が含まれます。すべての XML 文書は、ABC.SALES として登録されている XML スキーマに対応し、スキーマには分解情報でアノテーションが付けられ、分解が可能にされています。以下の例では、スキーマ ABC.SALES を使用して、SALESDOC に保管されている文書を分解するストアード・プロシージャーを呼び出します。

```
XDB_DECOMP_XML_FROM_QUERY ('ABC', 'SALES',
    'SELECT DOCID, SALESDOC FROM ABC.SALESTAB', 0, 0, 0,
    NULL, NULL, 1, numInput, numDecomposed, errorreportBuf);
```

XML 分解アノテーション

アノテーション付き XML スキーマ分解は、XML スキーマ文書に追加したアノテーションに依存します。これらの分解アノテーションは、XML 文書の要素または属性と、データベース中のターゲットの表および列との間のマッピングとして機能します。分解プロセスは、これらのアノテーションを参照して XML 文書の分解方法を判別します。

XML 分解アノテーションは http://www.ibm.com/xmlns/prod/db2/xdb1 名前空間に属し、文書全体で "db2-xdb" 接頭部によって識別されます。独自の接頭部を選択できますが、その場合には、ご使用の接頭部を名前空間 http://www.ibm.com/xmlns/prod/db2/xdb1 にバインドしなければなりません。XML スキーマを分解に使用できるようにした時点で、分解プロセスはこの名前空間の下のアノテーションのみ認識します。

スキーマ文書中で、分解アノテーションが要素や属性の宣言に追加されたり、グローバル・アノテーションとして追加されたりする場合にのみ、その分解アノテーションは分解プロセスに認識されます。分解アノテーションは、属性として指定するか、要素または属性の宣言の <xs:annotation> 子要素の一部として指定します。複合タイプ、参照、またはその他の XML スキーマ構成に追加されたアノテーションは無視されます。

これらのアノテーションが XML スキーマ文書中にあっても、スキーマ文書の元の構造に影響を与えたり、XML 文書の妥当性検査に関係したりしません。XML 分解プロセスによってのみ参照されます。

db2-xdb:rowSet と db2-xdb:column は、分解処理の中核機能である 2 つのアノテーションです。これらのアノテーションは分解される値のターゲット表と列をそれぞれ指定します。分解プロセスが正常に完了するために、これら 2 つのアノテーションは指定されなければなりません。他のアノテーションはオプションですが、分解処理の操作方法を詳細に制御するのに使用できます。

XML 分解アノテーション - 指定と有効範囲

分解アノテーションは、XML スキーマ文書で要素または属性宣言として指定することができます。

アノテーションは、以下のいずれかとして指定することができます。

- 要素または属性宣言における単純な属性、または
- 要素または属性宣言の構造化された (複合型の) 子要素 (複数の単純な要素や属性で構成される)

属性としてのアノテーション

要素または属性宣言で単純な属性として指定されたアノテーションは、そのアノテーションが指定された要素や属性にのみ適用されます。

例えば、db2-xdb:rowSet および db2-xdb:column 分解アノテーションを属性として指定できます。これらのアノテーションは、次のように指定します。

```
<xs:element name="isbn" type="xs:string"
               db2-xdb:rowSet="TEXTBOOKS" db2-xdb:column="ISBN"/>
```

db2-xdb:rowSet および db2-xdb:column アノテーションは、isbn という名前を持つこの要素だけに適用されます。

構造化された子要素としてのアノテーション

要素または属性宣言の構造化された子要素として指定するアノテーションは、スキーマ文書において、XML Schema で定義されている <xs:annotation><xs:appinfo></xs:appinfo></xs:annotation> 階層内で指定されている必要があります。

例えば、db2-xdb:rowSet および db2-xdb:column アノテーションは、次のように子要素として指定することができます (<db2-xdb:rowSetMapping> アノテーションの子になります)。

```
<xs:element name="isbn" type="xs:string">
  <xs:annotation>
    <xs:appinfo>
      <db2-xdb:rowSetMapping>
        <db2-xdb:rowSet>TEXTBOOKS</db2-xdb:rowSet>
        <db2-xdb:column>ISBN</db2-xdb:column>
      </db2-xdb:rowSetMapping>
    </xs:appinfo>
  </xs:annotation>
</xs:element>
```

db2-xdb:rowSet および db2-xdb:column アノテーションを子要素として指定することは、これらのアノテーションを属性として指定することと同じです (前述)。アノテーションを子要素として指定する方法は、アノテーションを属性として指定する方法よりも冗長ですが、1 つの要素や属性について複数の <db2-xdb:rowSetMapping> を指定しなければならない場合 (つまり、同じ要素または属性宣言で複数のマッピングを指定しなければならない場合) に必要となります。

グローバル・アノテーション

アノテーションを <xs:schema> 要素の子として指定すると、そのアノテーションは XML スキーマを構成する XML スキーマ文書全体に適用されるグローバル・アノテーションになります。

例えば、<db2-xdb:defaultSQLSchema> アノテーションは、XML スキーマで参照されるすべての未修飾表に対するデフォルトの SQL スキーマを示します。<db2-xdb:defaultSQLSchema> は <xs:schema> の子要素として指定する必要があります。

```
<xs:schema>
  <xs:annotation>
    <xs:appinfo>
      <db2-xdb:defaultSQLSchema>admin</db2-xdb:defaultSQLSchema>
    </xs:appinfo>
  </xs:annotation>
  ...
</xs:schema>
```

この宣言は、この XML スキーマを形成するすべてのスキーマ文書にまたがるすべての未修飾表が、「admin」という SQL スキーマを使用することを指定します。

特定のアノテーションの指定方法を判別するには、具体的なアノテーションの資料を参照してください。

XML 分解アノテーション - 要約

DB2 は、XML 文書の要素および属性をターゲット・データベース表にマップするためにアノテーション付き XML スキーマ分解プロセスによって使用されるアノテーションのセットをサポートします。以下のいくつかの XML 分解アノテーションの要約は、アノテーションを使用して実行するタスクおよびアクションごとにグループ化されています。

特定のアノテーションについて詳しくは、該当する詳細資料を参照してください。

表 42. SQL スキーマの指定

アクション	XML 分解アノテーション
独自の SQL スキーマを指定しないすべての表に対してデフォルトの SQL スキーマを指定する	db2-xdb:defaultSQLSchema
デフォルト以外の SQL スキーマを特定の表に指定する	db2-xdb:table (<db2-xdb:SQLSchema> 子要素)

表 43. XML 要素または属性からターゲット基本表へのマップ

アクション	XML 分解アノテーション
1 つの要素または属性を 1 つの列と表の対にマップする	属性アノテーションとして db2-xdb:column を持つ db2-xdb:rowSet、または db2-xdb:rowSetMapping
1 つの要素または属性を、異なる 1 つ以上の列と表の対にマップする	db2-xdb:rowSetMapping
複数の要素または属性を 1 つの列または表の対にマップする	db2-xdb:table
ターゲット表間の順序付けの従属関係を指定する	db2-xdb:rowSetOperationOrder, db2-xdb:rowSet, db2-xdb:order

表 44. XML データの分解を指定する

アクション	XML 分解アノテーション
複合タイプの要素に挿入される内容のタイプを指定する (テキスト、ストリング、またはマークアップ)	db2-xdb:contentHandling
適用される内容のトランスフォーメーションがある場合、挿入前にそれを指定する	db2-xdb:normalization, db2-xdb:expression, db2-xdb:truncate
項目の内容またはそれが現れるコンテキストに基づいて分解されるデータをフィルタリングする	db2-xdb:condition db2-xdb:locationPath

db2-xdb:defaultSQLSchema 分解アノテーション

db2-xdb:defaultSQLSchema アノテーションは、db2-xdb:table アノテーションを使用して明示的に修飾されていない XML スキーマ内で参照されるすべての表名のデフォルト SQL スキーマを指定します。

<db2-xdb:defaultSQLSchema> は、XML 文書中の要素と属性を DB2 の基本表へマッピングする方法を記述するために、XML スキーマ文書に追加できる分解アノテーションのセットに属します。分解プロセスでは、XML 文書の要素と属性を分解して DB2 表に入れる方法を決定するために、アノテーション付き XML スキーマを使用します。

アノテーションの種類

グローバル <xs:annotation> 要素の子である <xs:appinfo> の子要素。

指定方法

db2-xdb:defaultSQLSchema は次のように指定されます (*value* はアノテーションの有効な値を表します)。

```
<xs:schema>
  <xs:annotation>
    <xs:appinfo>
      <db2-xdb:defaultSQLSchema>value</db2-xdb:defaultSQLSchema>
```

```
    </xs:appinfo>
  </xs:annotation>
  ...
</xs:schema>
```

名前空間

http://www.ibm.com/xmlns/prod/db2/xdb1

有効な値

通常の、または区切り文字付き SQL スキーマ名のいずれか。通常の (区切り文字なしの) SQL スキーマ名は大/小文字を区別しません。区切り文字付き SQL スキーマを指定するには、引用符 (通常は SQL ID を区切るために使用される) を使用します。特殊文字「<」および「&」を含む SQL スキーマ名は XML スキーマ文書ではエスケープされなければなりません。

詳細情報

アノテーション付きのスキーマで参照される表はすべてその SQL スキーマで修飾されなければなりません。表の修飾には 2 とおりの方法があります。1 つは <db2-xdb:table> アノテーションの <db2-xdb:SQLSchema> 子要素を指定する方法、もう 1 つは <db2-xdb:defaultSQLSchema> グローバル・アノテーションを使用する方法です。非修飾表名では、<db2-xdb:defaultSQLSchema> で指定される値がその SQL スキーマ名として使用されます。アノテーション付きスキーマ内の複数のスキーマ文書がこのアノテーションを指定する場合は、すべての値を同じにしなければなりません。

例

次の例は、通常の (区切り文字なしの) SQL ID admin を、アノテーション付きスキーマ内のすべての非修飾表で SQL スキーマとして定義する方法を示しています。

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
                xmlns:db2-xdb="http://www.ibm.com/xmlns/prod/db2/xdb1">
  <xs:annotation>
    <xs:appinfo>
      <db2-xdb:defaultSQLSchema>admin</db2-xdb:defaultSQLSchema>
    </xs:appinfo>
  </xs:annotation>
  ...
</xs:schema>
```

次の例は、アノテーション付きスキーマ内のすべての非修飾表で区切り文字付き SQL ID admin schema を SQL スキーマとして定義する方法を示しています。admin schema は引用符で区切られていなければなりません。

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:db2-xdb="http://www.ibm.com/xmlns/prod/db2/xdb1">
  <xs:annotation>
    <xs:appinfo>
      <db2-xdb:defaultSQLSchema>"admin schema"</db2-xdb:defaultSQLSchema>
    </xs:appinfo>
  </xs:annotation>
  ...
</xs:schema>
```

db2-xdb:rowSet 分解アノテーション

db2-xdb:rowSet アノテーションは、XML 要素または属性からターゲットの基本表へのマッピングを指定します。

db2-xdb:rowSet は、XML 文書中の要素と属性を DB2 の基本表へマッピングする方法を記述するために、XML スキーマ文書に追加できる分解アノテーションのセットに属します。分解プロセスでは、XML 文書の要素と属性を分解して DB2 表に入れる方法を決定するために、アノテーション付き XML スキーマを使用します。

アノテーションの種類

<xs:element> または <xs:attribute> の属性、あるいは <db2-xdb:rowSetMapping> または <db2-xdb:order> の子要素

指定方法

db2-xdb:rowSet は次のいずれかの方法によって指定されます (*value* はアノテーションの有効な値を表します)。

- `<xs:element db2-xdb:rowSet="value" />`
- `<xs:attribute db2-xdb:rowSet="value" />`
- ```
  <db2-xdb:rowSetMapping>
    <db2-xdb:rowSet>value</db2-xdb:rowSet>
    ...
  </db2-xdb:rowSetMapping>
  ```
- ```
  <db2-xdb:order>
    <db2-xdb:rowSet>value</db2-xdb:rowSet>
    ...
  </db2-xdb:order>
  ```

名前空間

http://www.ibm.com/xmlns/prod/db2/xdb1

有効な値

SQL ID の規則に従う ID すべて。詳しくは、ID の資料を参照してください。

詳細情報

db2-xdb:rowSet アノテーションは、XML 要素または属性からターゲットの基本表へのマップを指定します。このアノテーションは、表名を直接識別することもできますし、より複雑なマッピングの rowSet 名を識別することもできます (後者の場合には rowSet は db2-xdb:table アノテーションを介して表名に関連付けられます)。単純なマッピングではこのアノテーションは、値の分解先となる表の名前を指定します。複数の rowSet (それぞれが固有の名前を持つ) が同じ表にマップするような、より複雑なマッピングでは、このアノテーションは表名ではなく、rowSet を指定します。

この XML 要素または属性の値の分解先となるターゲットの基本表は、アノテーション付きスキーマを形成するスキーマ文書のセットに他のアノテーションがあるかどうかによって決まります。

- db2-xdb:rowSet の値が <db2-xdb:table> グローバル・アノテーションの子要素 <db2-xdb:rowSet> と一致しない場合、ターゲット表の名前は、このアノテーションによって指定される値になり、その値は <db2-xdb:defaultSQLSchema> グローバル・アノテーションによって定義される SQL スキーマによって修飾されます。特定の表に関してその表にマップする要素または属性のセットが 1 つしかない場合に、db2-xdb:rowSet のこのような用い方をします。
- db2-xdb:rowSet の値が <db2-xdb:table> グローバル・アノテーションの子要素 <db2-xdb:rowSet> と一致する場合、ターゲット表の名前は、<db2-xdb:table> の子 <db2-xdb:name> の中で指定された表になります。特定の表でその表にマップする要素または属性の複数の (おそらく重複する) セットがあるようなより複雑なケースで、db2-xdb:rowSet のこのような用い方をします。

重要: XML スキーマが XML スキーマ・リポジトリーに登録されるときに、このアノテーションが参照する表がデータベースに存在していなければなりません (XML スキーマの登録時には db2-xdb:column アノテーションで指定される列も存在していなければなりません)。 XML スキーマで分解が可能な場合に表が存在しない場合には、エラーが戻されます。 <db2-xdb:table> が表以外のオブジェクトを指定する場合にも、エラーが戻されます。

db2-xdb:rowSet アノテーションが使用される場合には、db2-xdb:column アノテーションか db2-xdb:condition アノテーションのいずれかを指定する必要があります。 db2-xdb:rowSet と db2-xdb:column の組み合わせは、この要素または属性の分解先となる表および列を記述します。 db2-xdb:rowSet と db2-xdb:condition の組み合わせは、表に挿入されるその rowSet の行すべてで真にならなければならない条件を指定します (これは直接、または <db2-xdb:table> アノテーションを介して間接的に参照されます)。

例

上記にリストされた db2-xdb:rowSet の 2 とおりの使い方が次に説明されます。

表にマップされる要素または属性の 1 つのセット

次のようなアノテーション付きスキーマのセクションがあるとします。この中で、BOOKCONTENTS 表は <db2-xdb:defaultSQLSchema> によって指定される SQL スキーマに属し、「BOOKCONTENTS」と一致する子要素 <db2-xdb:rowSet> を持つグローバル要素 <db2-xdb:table> がありません。

```
<xs:element name="book">
   <xs:complexType>
     <xs:sequence>
       <xs:element name="authorID" type="xs:integer" />
       <xs:element name="chapter" type="chapterType" maxOccurs="unbounded" />
     </xs:sequence>
     <xs:attribute name="isbn" type="xs:string"
                   db2-xdb:rowSet="BOOKCONTENTS" db2-xdb:column="ISBN" />
     <xs:attribute name="title" type="xs:string" />
   </xs:complexType>
 </xs:element>

 <xs:complexType name="chapterType">
   <xs:sequence>
     <xs:element name="paragraph" type="paragraphType" maxOccurs="unbounded"
                 db2-xdb:rowSet="BOOKCONTENTS" db2-xdb:column="CHPTCONTENT" />
   </xs:sequence>
```

```
    <xs:attribute name="number" type="xs:integer"
                  db2-xdb:rowSet="BOOKCONTENTS" db2-xdb:column="CHPTNUM" />
    <xs:attribute name="title" type="xs:string"
                  db2-xdb:rowSet="BOOKCONTENTS" db2-xdb:column="CHPTTITLE" />
  </xs:complexType>

  <xs:simpleType name="paragraphType">
    <xs:restriction base="xs:string"/>
  </xs:simpleType>
```

XML 文書にある次の要素を考えてみましょう。

```
  <book isbn="1-11-111111-1" title="My First XML Book">
    <authorID>22</authorID>
    <!-- this book does not have a preface -->
    <chapter number="1" title="Introduction to XML">
      <paragraph>XML is fun...</paragraph>
      ...
    </chapter>
    <chapter number="2" title="XML and Databases">
      <paragraph>XML can be used with...</paragraph>
    </chapter>
    ...
    <chapter number="10" title="Further Reading">
      <paragraph>Recommended tutorials...</paragraph>
    </chapter>
    ...
  </book>
```

次のように BOOKCONTENTS 表にデータが取り込まれます。

表 45. BOOKCONTENTS

ISBN	CHPTNUM	CHPTTITLE	CHPTCONTENT
1-11-111111-1	1	Introduction to XML	XML is fun...
1-11-111111-1	2	XML and Databases	XML can be used with...
...	...	...	...
1-11-111111-1	10	Further Reading	Recommended tutorials...

同じ表にマップされる要素または属性の複数セット

db2-xdb:rowSet アノテーションで指定された値に一致するグローバル・アノテーション <db2-xdb:table> の子要素 <db2-xdb:rowSet> が存在する場合、要素または属性は、<db2-xdb:table> アノテーションを介して表にマップされます。 ALLBOOKS 表が <db2-xdb:defaultSQLSchema> によって指定される SQL スキーマに属する次のようなアノテーション付きスキーマのセクションがあるとします。

```
<!-- global annotation -->
<xs:annotation>
  <xs:appinfo>
    <db2-xdb:table>
       <db2-xdb:name>ALLBOOKS</db2-xdb:name>
       <db2-xdb:rowSet>book</db2-xdb:rowSet>
       <db2-xdb:rowSet>textbook</db2-xdb:rowSet>
    </db2-xdb:table>
  </xs:appinfo>
</xs:annotation>

<xs:element name="book">
```

```
  <xs:complexType>
    <xs:sequence>
      <xs:element name="authorID" type="xs:integer"
                  db2-xdb:rowSet="book" db2-xdb:column="AUTHORID" />
      <xs:element name="chapter" type="chapterType" maxOccurs="unbounded" />
    </xs:sequence>
    <xs:attribute name="isbn" type="xs:string"
                  db2-xdb:rowSet="book" db2-xdb:column="ISBN" />
    <xs:attribute name="title" type="xs:string"
                  db2-xdb:rowSet="book" db2-xdb:column="TITLE" />
  </xs:complexType>
</xs:element>
<xs:element name="textbook">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="isbn" type="xs:string"
                  db2-xdb:rowSet="textbook" db2-xdb:column="ISBN" />
      <xs:element name="title" type="xs:string"
                  db2-xdb:rowSet="textbook" db2-xdb:column="TITLE" />
      <xs:element name="primaryauthorID" type="xs:integer"
                  db2-xdb:rowSet="textbook" db2-xdb:column="AUTHORID" />
      <xs:element name="coauthorID" type="xs:integer"
                  minOccurs="0" maxOccurs="unbounded" />
      <xs:element name="subject" type="xs:string" />
      <xs:element name="edition" type="xs:integer" />
      <xs:element name="chapter" type="chapterType" maxOccurs="unbounded" />
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:complexType name="chapterType">
  <xs:sequence>
    <xs:element name="paragraph" type="paragraphType" maxOccurs="unbounded" />
  </xs:sequence>
  <xs:attribute name="number" type="xs:integer" />
  <xs:attribute name="title" type="xs:string" />
</xs:complexType>

<xs:simpleType name="paragraphType">
  <xs:restriction base="xs:string"/>
</xs:simpleType>
```

XML 文書にある次の要素を考えてみましょう。

```
<book isbn="1-11-111111-1" title="My First XML Book">
  <authorID>22</authorID>
  <!-- this book does not have a preface -->
  <chapter number="1" title="Introduction to XML">
    <paragraph>XML is fun...</paragraph>
  </chapter>
  <chapter number="2" title="XML and Databases">
    <paragraph>XML can be used with...</paragraph>
  </chapter>
  <chapter number="10" title="Further Reading">
    <paragraph>Recommended tutorials...</paragraph>
  </chapter>
</book>

<textbook>
  <isbn>0-11-011111-0</isbn>
  <title>Programming with XML</title>
  <primaryauthorID>435</primaryauthorID>
  <subject>Programming</subject>
  <edition>4</edition>
  <chapter number="1" title="Programming Basics">
    <paragraph>Before you being programming...</paragraph>
  </chapter>
```

```
  <chapter number="2" title="Writing a Program">
    <paragraph>Now that you have learned the basics...</paragraph>
  </chapter>
  ...
  <chapter number="10" title="Advanced techniques">
    <paragraph>You can apply advanced techniques...</paragraph>
  </chapter>
</textbook>
```

この例には、表 ALLBOOKS にマップする要素または属性の 2 つのセットがあります。

- /book/@isbn, /book/@authorID, /book/title
- /textbook/isbn, /textbook/primaryauthorID, /textbook/title

そのセットは、異なる rowSet 名を相互に関連付けることによって見分けられます。

表 46. ALLBOOKS

ISBN	TITLE	AUTHORID
1-11-111111-1	My First XML Book	22
0-11-011111-0	Programming with XML	435

db2-xdb:table 分解アノテーション

<db2-xdb:table> アノテーションは、複数の XML 要素または属性を同じターゲット列にマップします。また、<db2-xdb:defaultSQLSchema> によって指定されるデフォルトの SQL スキーマとは別の SQL スキーマを持つターゲット表を指定できるようにします。

<db2-xdb:table> は、XML 文書中の要素と属性を DB2 の基本表へマッピングする方法を記述するために、XML スキーマ文書に追加できる分解アノテーションのセットに属します。分解プロセスでは、XML 文書の要素と属性を分解して DB2 表に入れる方法を決定するために、アノテーション付き XML スキーマを使用します。

アノテーションの種類

<xs:appinfo> (<xs:annotation> の子要素) のグローバル子要素

名前空間

http://www.ibm.com/xmlns/prod/db2/xdb1

有効な構造

以下は、サポートされる <db2-xdb:table> の子要素です。指定される場合、その子要素が現れる順にリストされています。

<db2-xdb:SQLSchema>
(任意指定) 表の SQL スキーマ。

<db2-xdb:name>
基本表の名前。この表名がその先頭に <db2-xdb:SQLSchema> アノテーションまたは <db2-xdb:defaultSQLSchema> アノテーションのいずれかを付ける

ことによって修飾されるときは、アノテーション付きスキーマを形成する XML スキーマ文書のセットすべての <db2-xdb:table> アノテーションの中で固有でなければなりません。

<db2-xdb:rowSet>

<db2-xdb:rowSet> に対して同じ値を指定する要素と属性はすべて 1 つの行を形成します。 <db2-xdb:name> の同じ値には複数の <db2-xdb:rowSet> 要素を指定できるので、1 つの表には複数のマッピングのセットを関連付けることができます。 <db2-xdb:rowSet> 値と db2-xdb:column アノテーションで指定される列を組み合わせると、1 つの XML 文書からの複数の要素または属性のセットを同じ表の列にマップすることができます。

アノテーションを有効にするために少なくとも 1 つの <db2-xdb:rowSet> 要素を指定し、それぞれの <db2-xdb:rowSet> 要素を、アノテーション付きスキーマを形成する XML スキーマ文書のセットすべての <db2-xdb:table> アノテーションの中で固有にする必要があります。

<db2-xdb:table> の子要素の文字内容の中の空白には意味があり、正規化されません。これらの要素の内容は、SQL ID のスペル規則に従う必要があります。区切り文字なしの値に大/小文字の区別はありません。区切り文字付きの値の場合には区切り文字に引用符が使用されます。特殊文字「<」および「&」を含む SQL ID はエスケープされなければなりません。

詳細情報

次のいずれかの場合には <db2-xdb:table> アノテーションを使用する必要があります。

- 複数の祖先系列が表の同じ列にマップされる場合 (単一のロケーション・パスを含むマッピングは、表に対して列マッピングのセットが 1 つだけであることを意味し、このアノテーションを使用する必要はありません。代わりに db2-xdb:rowSet アノテーションを使用できます。)
- 分解されたデータを保持する表が、<db2-xdb:defaultSQLSchema> アノテーションによって定義されたのと同じ SQL スキーマのものではない場合

指定できるのは基本表のみです。それ以外の種類の表、例えば型付き表、サマリー表、一時表、マテリアライズ照会表などはこのマッピングではサポートされていません。ニックネームを指定できるのは、DB2 Database for Linux, UNIX, and Windows データ・ソース・オブジェクトについてだけです。ビューおよび表の別名は、今のところこのアノテーションでは許可されていません。

例

以下の例は、複数のロケーション・パスが同じ列にマップされている場合に、<db2-xdb:table> アノテーションを使用して、関連する要素と属性をグループ化し、行を形成する方法を示しています。 XML 文書にある次の要素をまず考えます (別のアノテーションで使用されている例に若干の変更が加えられたものです)。

```
<root>
  ...
  <book isbn="1-11-111111-1" title="My First XML Book">
    <authorID>22</authorID>
    <email>author22@anyemail.com</email>
    <!-- this book does not have a preface -->
```

```
    <chapter number="1" title="Introduction to XML">
      <paragraph>XML is fun...</paragraph>
      ...
    </chapter>
    <chapter number="2" title="XML and Databases">
      <paragraph>XML can be used with...</paragraph>
    </chapter>
    ...
    <chapter number="10" title="Further Reading">
      <paragraph>Recommended tutorials...</paragraph>
    </chapter>
  </book>
  ...
  <author ID="0800" email="author800@email.com">
    <firstname>Alexander</firstname>
    <lastname>Smith</lastname>
    <activeStatus>0</activeStatus>
  </author>
  ...
<root>
```

作成者 ID とそれに対応する E メール・アドレスで構成される行を同じ表の AUTHORSCONTACT に挿入することがこの分解マッピングの目的であるとします。作成者 ID と E メール・アドレスが <book> 要素と <author> 要素の両方に現れています。したがって、同じ表の同じ列に複数のロケーション・パスをマップすることが必要となります。よって、<db2-xdb:table> アノテーションを使わなければなりません。アノテーション付きスキーマのセクションが次に示されますが、ここでは <db2-xdb:table> を使って複数のパスを同じ表に関連付ける方法が示されています。

```
<!-- global annotation -->
  <xs:annotation>
    <xs:appinfo>
      <db2-xdb:defaultSQLSchema>adminSchema</db2-xdb:defaultSQLSchema>
      <db2-xdb:table>
        <db2-xdb:SQLSchema>user1</db2-xdb:SQLSchema>
        <db2-xdb:name>AUTHORSCONTACT</db2-xdb:name>
        <db2-xdb:rowSet>bookRowSet</db2-xdb:rowSet>
        <db2-xdb:rowSet>authorRowSet</db2-xdb:rowSet>
      </db2-xdb:table>
    </xs:appinfo>
  </xs:annotation>

  <xs:element name="book">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="authorID" type="xs:integer"
                    db2-xdb:rowSet="bookRowSet" db2-xdb:column="AUTHID" />
        <xs:element name="email" type="xs:string"
                    db2-xdb:rowSet="bookRowSet" db2-xdb:column="EMAILADDR" />
        <xs:element name="chapter" type="chapterType" maxOccurs="unbounded" />
      </xs:sequence>
      <xs:attribute name="isbn" type="xs:string" />
      <xs:attribute name="title" type="xs:string" />
    </xs:complexType>
  </xs:element>

  <xs:element name="author">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="firstname" type="xs:string" />
        <xs:element name="lastname" type="xs:string" />
        <xs:element name="activeStatus" type="xs:boolean" />
      </xs:sequence>
```

```
      <xs:attribute name="ID" type="xs:integer"
                    db2-xdb:rowSet="authorRowSet" db2-xdb:column="AUTHID" />
      <xs:attribute name="email" type="xs:string"
                    db2-xdb:rowSet="authorRowSet" db2-xdb:column="EMAILADDR" />
    </xs:complexType>
  </xs:element>
```

<db2-xdb:table> アノテーションは、db2-xdb:name 子要素を持つマッピングのターゲット表の名前を識別します。この例では AUTHORSCONTACT がターゲット表です。 <book> 要素の ID と E メール・アドレスを <author> 要素のものと分けておくために (すなわち、各行に論理的に関連する値が含まれることになる)、<db2-xdb:rowSet> 要素を使用して関連する項目を関連付けます。この例では <book> 要素と <author> 要素はそれぞれ別個のエンティティーですが、マップされるエンティティーが分けられず、論理的な分離が必要になる場合もあります。この論理的な分離は、rowSet を使用して行えます。

AUTHORSCONTACT 表は、デフォルトの SQL スキーマとは別の SQL スキーマにあり、<db2-xdb:SQLSchema>要素はこれを指定するために使用されます。結果の AUTHORSCONTACT 表を、表 1 に示します。

表 47. AUTHORSCONTACT

AUTHID	EMAILADDR
22	author22@anyemail.com
0800	author800@email.com

この例は、rowSet による値の論理グループ化によって、関係のない値が意図せず同じ表と列の対にマップされてしまうのをどのように防ぐかを示しています。この例の /root/book/authorID と /root/author/@ID は同じ表と列の対にマップされます。同じように、/root/book/email と /root/author/@email も同じ表と列の対にマップされます。 rowSet を使用できないケースを考えてみましょう。例えば <author> 要素のインスタンスに /root/book/email 要素が存在せず、rowSet を使用できないとしたら、<author> 要素の email を /root/book/authorID に関連付けるか、または /root/author/@ID に関連付けるか、あるいはその両方に関連付けるかを判別することは不可能でしょう。このように、<db2-xdb:table> アノテーションの中の 1 つの表に rowSet が関連付けられていると、さまざまな行セットの中での論理的な区別を行う助けとなります。

db2-xdb:column 分解アノテーション

db2-xdb:column アノテーションは、XML 要素または属性がマップされた表の列名を指定します。

db2-xdb:column は、XML 文書中の要素と属性を DB2 の基本表へマッピングする方法を記述するために、XML スキーマ文書に追加できる分解アノテーションのセットに属します。分解プロセスでは、XML 文書の要素と属性を分解して DB2 表に入れる方法を決定するために、アノテーション付き XML スキーマを使用します。

アノテーションの種類

<xs:element> または <xs:attribute> の属性、あるいは <db2-xdb:rowSetMapping> の子要素

指定方法

db2-xdb:column は次のいずれかの方法によって指定されます (*value* はアノテーションの有効な値を表します)。

- `<xs:element db2-xdb:rowSet="value" db2-xdb:column="value" />`
- `<xs:attribute db2-xdb:rowSet="value" db2-xdb:column="value" />`
-
```
<db2-xdb:rowSetMapping>
  <db2-xdb:rowSet>value</db2-xdb:rowSet>
  <db2-xdb:column>value</db2-xdb:column>
  ...
</db2-xdb:rowSetMapping>
```

名前空間

http://www.ibm.com/xmlns/prod/db2/xdb1

有効な値

以下の規則に従う基本表の列名すべて。

- 区切り文字なしの列名は大/小文字を区別しません。区切り文字付きの列名の場合、区切り文字は `&quot;` でエスケープします。例えば、2 語で成る列名「col one」を指定するには、db2-xdb:column を次のように設定します。

  ```
  db2-xdb:column="&quot;col one&quot;"
  ```

  (これらの条件はこの注釈に固有の要件です。)
- このアノテーションには、次のデータ・タイプの列のみを指定できます。すなわち、ユーザー定義の構造化タイプを除く、CREATE TABLE SQL ステートメントによってサポートされるすべてのデータ・タイプです。

詳細情報

db2-xdb:column アノテーションは XML 要素または属性の宣言内で属性として、または <db2-xdb:rowSetMapping> の子要素として指定され、XML 要素または属性をターゲット表の列名にマップします。このアノテーションが使用される場合には、db2-xdb:rowSet アノテーションも指定しなければなりません。それらが一緒になって、この要素または属性の分解された値を保持する表および列を記述します。

例

以下の例は、db2-xdb:column アノテーションを使用して、<book> 要素の内容を BOOKCONTENTS という表の列に挿入する方法を示しています。アノテーション付きスキーマのセクションがまず示されています。

```
<xs:element name="book">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="authorID" type="xs:integer" />
      <xs:element name="chapter" type="chapterType" maxOccurs="unbounded" />
    </xs:sequence>
    <xs:attribute name="isbn" type="xs:string"
                  db2-xdb:rowSet="BOOKCONTENTS" db2-xdb:column="ISBN" />
    <xs:attribute name="title" type="xs:string" />
  </xs:complexType>
```

```
</xs:element>

<xs:complexType name="chapterType">
  <xs:sequence>
    <xs:element name="paragraph" type="paragraphType" maxOccurs="unbounded"
                db2-xdb:rowSet="BOOKCONTENTS"
                db2-xdb:column="CHPTCONTENT" />
  </xs:sequence>
  <xs:attribute name="number" type="xs:integer"
                db2-xdb:rowSet="BOOKCONTENTS"
                db2-xdb:column="CHPTNUM" />
  <xs:attribute name="title" type="xs:string"
                db2-xdb:rowSet="BOOKCONTENTS"
                db2-xdb:column="CHPTTITLE" />
</xs:complexType>

<xs:simpleType name="paragraphType">
  <xs:restriction base="xs:string"/>
</xs:simpleType>
```

マップされている <book> 要素が次に示され、その後に分解完了後の BOOKCONTENTS 表が示されています。

```
<book isbn="1-11-111111-1" title="My First XML Book">
  <authorID>22</authorID>
  <!-- this book does not have a preface -->
  <chapter number="1" title="Introduction to XML">
    <paragraph>XML is fun...</paragraph>
    ...
  </chapter>
  <chapter number="2" title="XML and Databases">
    <paragraph>XML can be used with...</paragraph>
  </chapter>
  ...
  <chapter number="10" title="Further Reading">
    <paragraph>Recommended tutorials...</paragraph>
  </chapter>
</book>
```

表 48. BOOKCONTENTS

ISBN	CHPTNUM	CHPTTITLE	CHPTCONTENT
1-11-111111-1	1	Introduction to XML	XML is fun...
1-11-111111-1	2	XML and Databases	XML can be used with...
...	...	...	...
1-11-111111-1	10	Further Reading	Recommended tutorials...

db2-xdb:locationPath 分解アノテーション

db2-xdb:locationPath アノテーションは、グローバルに宣言されるか、再使用可能グループの一部として宣言される XML 要素または属性を、その祖先に応じてさまざまな表と列の対にマップします。再使用可能グループはグローバルに宣言される、名前付き複合タイプ、名前付きモデル・グループ、および名前付き属性グループです。

db2-xdb:locationPath は、XML 文書中の要素と属性を DB2 の基本表へマッピングする方法を記述するために、XML スキーマ文書に追加できる分解アノテーションのセ

ットに属します。分解プロセスでは、XML 文書の要素と属性を分解して DB2 表に入れる方法を決定するために、アノテーション付き XML スキーマを使用します。

アノテーションの種類

<xs:element> または <xs:attribute> の属性、あるいは <db2-xdb:rowSetMapping> の属性

指定方法

db2-xdb:locationPath は次のいずれかの方法によって指定されます (*value* はアノテーションの有効な値を表します)。

- `<xs:element db2-xdb:locationPath="value" />`
- `<xs:attribute db2-xdb:locationPath="value" />`
-
```
<db2-xdb:rowSetMapping db2-xdb:locationPath="value">
  <db2-xdb:rowSet>value</db2-xdb:rowSet>
  ...
</db2-xdb:rowSetMapping>
```

名前空間

http://www.ibm.com/xmlns/prod/db2/xdb1

有効な値

db2-xdb:locationPath の値には次の構文が必要です。

```
location path := '/' (locationstep '/')* lastlocationstep
locationstep := (prefix':')? name
lastlocationstep := locationstep | '@' (prefix:)? name
```

`name` は要素または属性名で、`prefix` は名前空間の接頭部です。

注:

- ロケーション・パスで使用される名前空間接頭部はすべて、このロケーション・パスを指定する注釈を含むスキーマ文書内の名前空間と関連付けておく必要があります。
- 名前空間接頭部のバインディングは、名前空間宣言をスキーマ文書の <xs:schema> 要素に追加することによって作成できます。
- `prefix` が空の場合、`name` はどの名前空間中にもないと見なされます。スキーマ文書中でデフォルト名前空間が宣言されており、`locationstep` 中の名前がこの名前空間に属している場合、デフォルト名前空間の名前空間接頭部を宣言して、名前の修飾に使用しなければなりません。db2-xdb:locationPath の中では、空の接頭部はデフォルト名前空間を指しません。

詳細情報

db2-xdb:locationPath アノテーションは、グローバルに宣言されるか、次のいずれかの一部として宣言される要素または属性のマッピングを記述するために使用されます。

- 名前付きモデル・グループ

- 名前付き属性グループ
- グローバル複合タイプ宣言
- 単純または複合タイプのグローバル要素または属性

再利用できない要素または属性の宣言 (名前付き複合タイプ定義の一部ではなく、名前付きモデルまたは属性グループの一部でもないローカル宣言) の場合、db2-xdb:locationPath アノテーションに効果はありません。

グローバル要素または属性の宣言がさまざまな祖先系列からの参照として使用される場合には db2-xdb:locationPathを使用すべきです (例えば <xs:element ref="abc">)。アノテーションは参照に直接指定することができないため、代わりに、対応するグローバル要素または属性の宣言に対して指定しなければなりません。対応する要素または属性の宣言はグローバルなので、多くの異なる XML スキーマ内のコンテキストから要素または属性を参照することができます。一般に、これらの異なるコンテキスト内のマッピングを見分けるには、db2-xdb:locationPath を使用する必要があります。名前付き複合タイプ、モデル・グループ、および属性グループの場合、分解のためにマップされる各コンテキストごとの要素および属性の宣言にアノテーションを付ける必要があります。各 locationPath のターゲットの rowSet と column の対を指定するには、db2-xdb:locationPath アノテーションを使用する必要があります。異なる rowSet と column の対にも同じ db2-xdb:locationPath 値を使用できます。

例

以下の例は、この属性が現れるコンテキストに応じて、同じ属性を異なる表にマップする方法を示しています。アノテーション付きスキーマのセクションがまず示されています。

```
<!-- global attribute -->
<xs:attribute name="title" type="xs:string"
              db2-xdb:rowSet="BOOKS"
              db2-xdb:column="TITLE"
              db2-xdb:locationPath="/books/book/@title">
  <xs:annotation>
    <xs:appinfo>
      <db2-xdb:rowSetMapping db2-xdb:locationPath="/books/book/chapter/@title">
        <db2-xdb:rowSet>BOOKCONTENTS</db2-xdb:rowSet>
        <db2-xdb:column>CHPTTITLE</db2-xdb:column>
      </db2-xdb:rowSetMapping>
    </xs:appinfo>
  </xs:annotation>
</xs:attribute>

<xs:element name="books">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="book">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="authorID" type="xs:integer" />
            <xs:element name="chapter" type="chapterType" maxOccurs="unbounded" />
          </xs:sequence>
          <xs:attribute name="isbn" type="xs:string" />
          <xs:attribute ref="title" />
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
```

```
  </xs:element>

  <xs:complexType name="chapterType">
    <xs:sequence>
      <xs:element name="paragraph" type="paragraphType" maxOccurs="unbounded" />
    </xs:sequence>
    <xs:attribute name="number" type="xs:integer" />
    <xs:attribute ref="title" />
  </xs:complexType>

  <xs:simpleType name="paragraphType">
    <xs:restriction base="xs:string"/>
  </xs:simpleType>
```

"title" という属性宣言が 1 つだけしかないのに対し、この属性への参照は異なるコンテキストで 2 つあります。 1 つは <book> 要素からの参照で、もう 1 つは <chapter> 要素からの参照です。 "title" 属性の値は、そのコンテキストに応じて、異なる表に分解する必要があります。このアノテーション付きスキーマは、"title" 値が本のタイトルである場合は BOOKS 表に分解され、章のタイトルである場合は BOOKCONTENTS 表に分解されることを指定します。

マップされている <books> 要素が次に示され、その後に分解完了後の BOOKS 表が示されています。

```
<books>
  <book isbn="1-11-111111-1" title="My First XML Book">
    <authorID>22</authorID>
    <!-- this book does not have a preface -->
    <chapter number="1" title="Introduction to XML">
      <paragraph>XML is fun...</paragraph>
      ...
    </chapter>
    <chapter number="2" title="XML and Databases">
      <paragraph>XML can be used with...</paragraph>
    </chapter>
    ...
    <chapter number="10" title="Further Reading">
      <paragraph>Recommended tutorials...</paragraph>
    </chapter>
  </book>
...
</books>
```

表 49. BOOKS

ISBN	TITLE	CONTENT
NULL	My First XML Book	NULL

表 50. BOOKCONTENTS

ISBN	CHPTNUM	CHPTTITLE	CHPTCONTENT
NULL	NULL	Introduction to XML	NULL
NULL	NULL	XML and Databases	NULL
...	...	...	...
NULL	NULL	Further Reading	NULL

db2-xdb:expression 分解アノテーション

db2-xdb:expression アノテーションはカスタマイズされた式を指定します。その結果は、この要素のマップ先の表に挿入されます。

db2-xdb:expression は、XML 文書中の要素と属性を DB2 の基本表へマッピングする方法を記述するために、XML スキーマ文書に追加できる分解アノテーションのセットに属します。分解プロセスでは、XML 文書の要素と属性を分解して DB2 表に入れる方法を決定するために、アノテーション付き XML スキーマを使用します。

アノテーションの種類

<xs:element> または <xs:attribute> の属性、あるいは <db2-xdb:rowSetMapping> の任意指定の子要素 (列マッピングを含むアノテーションでのみ有効)。

指定方法

db2-xdb:expression は次のいずれかの方法によって指定されます (*value* はアノテーションの有効な値を表します)。

- `<xs:element db2-xdb:expression="value" db2-xdb:column="value" />`
- `<xs:attribute db2-xdb:expression="value" db2-xdb:column="value" />`
-
```
<db2-xdb:rowSetMapping>
  <db2-xdb:rowSet>value</db2-xdb:rowSet>
  <db2-xdb:column>value</db2-xdb:column>
  <db2-xdb:expression>value</db2-xdb:expression>
  ...
</db2-xdb:rowSetMapping>
```

名前空間

http://www.ibm.com/xmlns/prod/db2/xdb1

有効な値

db2-xdb:expression の値には次の構文が必要です。この構文は SQL 式のサブセットを構成します。

```
expression := function (arglist) | constant | $DECOMP_CONTENT | $DECOMP_ELEMENTID |
   $DECOMP_DOCUMENTID | (scalar-fullselect) | expression operator expression |
   (expression) | special-register | CAST (expression AS data-type) |
   XMLCAST (expression AS data-type) | XML-function

operator := + | - | * | / | CONCAT

arglist := expression | arglist, expression
```

詳細情報

db2-xdb:expression アノテーションを使用すると、カスタマイズされた式を指定できるようになります。これは $DECOMP_CONTENT の使用時にアノテーションが付けられている XML 要素または属性の内容に適用されます。この式の評価結果が、分解時に指定された列に挿入されます。

このアノテーションは、定数値 (例えば要素の名前) や、文書に現れない生成値を挿入するような場合にも便利です。

db2-xdb:expression は有効な SQL 式を使用して指定しなければならず、評価された式の種類は静的に決定でき、値が挿入されるターゲット列の種類と互換性のあるものでなければなりません。サポートされない SQL 式のサブセットに関しては、以下のリストで説明されていない他の SQL 式はすべてサポートされず、このアノテーションのコンテキストでは未定義の動作になります。

function (arglist)
組み込み、またはユーザー定義のスカラー SQL 関数。スカラー関数の引数は個々のスカラー値です。スカラー関数は単一値を戻します (NULL の可能性あり)。詳しくは、関数の資料を参照してください。

constant
ストリング定数または数値定数の値 (リテラルと呼ばれることもある)。詳しくは、定数の資料を参照してください。

$DECOMP_CONTENT
db2-xdb:contentHandling アノテーションの設定に従って構成される、文書にあるマップ済み XML 要素または属性の値。詳しくは、分解のキーワードの資料を参照してください。

$DECOMP_ELEMENTID
XML 文書内でこのアノテーションが記述する要素または属性を一意的に識別するシステム生成の整数 ID。詳しくは、分解のキーワードの資料を参照してください。

$DECOMP_DOCUMENTID
xdbDecompXML ストアード・プロシージャーの *documentid* 入力パラメーターに指定されるストリング値。分解される XML 文書を識別します。詳しくは、分解のキーワードの資料を参照してください。

(scalar-fullselect)
単一列値から成る単一行を戻す、括弧で囲まれた全選択。全選択が行を戻さない場合、式の結果は NULL 値になります。

expression operator expression
前述の、サポートされる値のリストで定義されている、サポートされる 2 つの式オペランドの結果。式の操作について詳しくは、式の資料を参照してください。

(expression)
上記で定義されている、サポートされる式のリストと一致する、括弧で囲まれた式。

special-register
サポートされる特殊レジスターの名前。この設定は、現行サーバーの特殊レジスターの値に評価されます。サポートされる特殊レジスターの完全なリストについて詳しくは、特殊レジスターの資料を参照してください。

CAST (expression AS data-type)
式が NULL でない場合、指定された SQL データ・タイプへの式のキャスト。式が NULL の場合、結果は指定された SQL データ・タイプの NULL 値になります。 NULL 値を列に挿入する際には、式は NULL を互換性のある列タイプ (例えば整数列の場合は CAST (NULL AS INTEGER)) にキャストしなければなりません。

XMLCAST (expression AS data-type)
式が NULL でない場合、指定されたデータ・タイプへの式のキャスト。式またはターゲット・データ・タイプは、XML タイプでなければなりません。式が NULL の場合、ターゲット・タイプは XML でなければならず、結果は NULL の XML 値になります。

XML-function
サポートされる任意の SQL/XML 関数。

例

以下の例は、db2-xdb:expression アノテーションを使用して、XML 文書にある値をユーザー定義関数に適用する方法を示しています。文書そのものからの値ではなく、UDF から戻された結果がデータベースに挿入されます。アノテーション付きスキーマのセクションがまず示されています。

```
<xs:element name="author">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="firstname" type="xs:string" />
      <xs:element name="lastname" type="xs:string" />
      <xs:element name="activeStatus" type="xs:boolean" />
      <xs:attribute name="ID" type="xs:integer"
                    db2-xdb:rowSet="AUTHORS" db2-xdb:column="NUMBOOKS"
                    db2-xdb:expression="AuthNumBooks (INTEGER ($DECOMP_CONTENT))" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

整数パラメーターを取る AuthNumBooks というユーザー定義関数があるとします。これは作成者の ID を表し、その作成者がシステム内に持つ本の合計数を戻します。

マップされている <author> 要素が次に示されています。

```
<author ID="22">
  <firstname>Ann</firstname>
  <lastname>Brown</lastname>
  <activeStatus>1</activeStatus>
</author>
```

$DECOMP_CONTENT は、ID 属性のインスタンスからの値「22」で置き換えられます。 $DECOMP_CONTENT は常に文字タイプで置き換えられ、AuthNumBooks UDF は整数パラメーターを取るため、db2-xdb:expression アノテーションは $DECOMP_CONTENT を整数にキャストしなければなりません。 ID が 22 のこの作成者に整数 8 を UDF が戻すとします。すると、次に示すように、AUTHORS 表の NUMBOOKS 列に 8 が挿入されます。

表 51. AUTHORS

AUTHID	FIRSTNAME	SURNAME	ACTIVE	NUMBOOKS
NULL	NULL	NULL	NULL	8

db2-xdb:condition 分解アノテーション

db2-xdb:condition アノテーションは、行を表に挿入するかどうかを判別する条件を指定します。この条件を満たす行を挿入できます (rowSet に関する他の条件があれば、それらの条件に依存します)。この条件を満たさない行は挿入されません。

db2-xdb:condition は、XML 文書中の要素と属性を DB2 の基本表へマッピングする方法を記述するために、XML スキーマ文書に追加できる分解アノテーションのセットに属します。分解プロセスでは、XML 文書の要素と属性を分解して DB2 表に入れる方法を決定するために、アノテーション付き XML スキーマを使用します。

アノテーションの種類

<xs:element> または <xs:attribute> の属性、あるいは <db2-xdb:rowSetMapping> のオプションの子要素。条件が属するアノテーションに列マッピングがあるかどうかにかかわらず、その条件は適用されます。

指定方法

db2-xdb:condition は次のいずれかの方法によって指定されます (*value* はアノテーションの有効な値を表します)。

- `<xs:element db2-xdb:condition="`*value*`" />`
- `<xs:attribute db2-xdb:condition="`*value*`" />`
- ```
  <db2-xdb:rowSetMapping>
    <db2-xdb:rowSet>value</db2-xdb:rowSet>
    <db2-xdb:condition>value</db2-xdb:condition>
    ...
  </db2-xdb:rowSetMapping>
  ```

名前空間

http://www.ibm.com/xmlns/prod/db2/xdb1

有効な値

基本、比較、BETWEEN、EXISTS、IN、IS VALIDATED、LIKE、NULL、および XMLEXISTS タイプの SQL 述部。また、これらの述部は db2-xdb:expression アノテーションまたは列名 (あるいはその両方) によってサポートされる式で構成されていなければなりません。

詳細情報

db2-xdb:condition アノテーションを使用すると、分解時に値がデータベースに挿入される条件を指定できます。このアノテーションは、ユーザー指定の条件を適用して、行をフィルターに掛けます。指定された条件を満たす行はデータベースに挿入されます。この条件を満たさない行は分解時に挿入されません。

同じ rowSet の複数の要素または属性の宣言に関する db2-xdb:condition アノテーションが指定された場合は、すべての条件の論理 AND が true と評価された場合のみ挿入されます。
  ```

db2-xdb:condition 内の列名

db2-xdb:condition は SQL 述部で構成されるので、このアノテーションに列名を指定できます。rowSet を含む db2-xdb:condition アノテーションに修飾されていない列名がある場合は、その rowSet に関係するすべてのマッピングにその列に対するマッピングが存在していなければなりません。SELECT ステートメントを含む述部で他の列名を使用する際には、それらの列名を修飾しなければなりません。
db2-xdb:condition で修飾されていない列名が指定されているが、db2-xdb:condition の指定対象の要素または属性に列マッピングが指定されていない場合は、条件が評価される際には、参照されている列名にマップしている要素または属性の内容が評価される値になります。

次の例を考慮してください。

```
<xs:element name="a" type="xs:string"
            db2-xdb:rowSet="rowSetA" db2-xdb:condition="columnX='abc'" />
<xs:element name="b" type="xs:string"
            db2-xdb:rowSet="rowSetB" db2-xdb:column="columnX" />
```

<a> には列マッピングが指定されていないものの、条件は列 "columnX" を参照していることに注意してください。条件が評価される際に、条件の "columnX" は <b> からの値に置き換えられます。その理由は、<b> には "columnX" の列マッピングが指定されており、<a> には列マッピングがないからです。XML 文書に以下の内容が含まれているとします。

```
<a>abc</a>
<b>def</b>
```

この場合は条件は false に評価されます。なぜなら、条件の <b> からの値 "def" が評価されるからです。

列名の代わりに $DECOMP_CONTENT (マップされた要素または属性の値を文字データとして指定する分解キーワード) を、要素 <a> 宣言に付加された db2-xdb:condition で使用する場合、<b> ではなく <a> の値を使用して条件が評価されます。

```
<xs:element name="a" type="xs:string"
            db2-xdb:rowSet="rowSetA" db2-xdb:condition="$DECOMP_CONTENT='abc'" />
<xs:element name="b" type="xs:string"
            db2-xdb:rowSet="rowSetB" db2-xdb:column="columnX" />
```

XML 文書に以下の内容が含まれているとします。

```
<a>abc</a>
<b>def</b>
```

この場合は条件は true に評価されます。なぜなら、<a> からの値 "abc" が評価に使用されるからです。

列名と $DECOMP_CONTENT を使用するこの条件付き処理は、データベースに挿入されない別の要素または属性の値に基づいた値のみ分解する際に便利な場合があります。

文書内にないマップされた要素または属性に関する条件が指定されている場合

要素または属性に関する条件が指定されているものの、その要素または属性が XML 文書内にない場合、その条件は依然として適用されます。例えば、アノテーション付きスキーマ文書からの以下の要素マッピングについて考慮してください。

```
<xs:element name="intElem" type="xs:integer"
            db2-xdb:rowSet="rowSetA" db2-xdb:column="colInt"
            db2-xdb:condition="colInt > 100" default="0" />
```

XML 文書内に <intElem> 要素がない場合でも、条件 "colInt > 100" は依然として評価されます。<intElem> がないので、"colInt" の条件評価でデフォルト値 0 が使用されます。続いて条件は 0 > 100 として評価されるので、false に評価されます。したがって、対応する行は分解時に挿入されません。

例

XML 文書からの以下の <author> 要素について考慮してください。

```
<author ID="0800">
  <firstname>Alexander</firstname>
  <lastname>Smith</lastname>
  <activeStatus>1</activeStatus>
</author>
```

db2-xdb:condition によって指定された条件に応じて、分解時にこの <author> 要素からターゲット表に値を挿入したりしなかったりします。次に、2 つのケースについて考慮します。

すべての条件が満たされる場合

前述の例で説明した <author> 要素に対応するアノテーション付きスキーマ中の以下のセクションは、作成者の ID が 1 から 999 までの間にあり、<firstname> および <lastname> 要素が NULL でなく、<activeStatus> 要素の値が 1 の場合のみこの要素を分解する必要があることを指定します。

```
<xs:element name="author">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="firstname" type="xs:string"
                  db2-xdb:rowSet="AUTHORS" db2-xdb:column="GIVENNAME"
                  db2-xdb:condition="$DECOMP_CONTENT IS NOT NULL" />
      <xs:element name="lastname" type="xs:string"
                  db2-xdb:rowSet="AUTHORS" db2-xdb:column="SURNAME"
                  db2-xdb:condition="$DECOMP_CONTENT IS NOT NULL" />
      <xs:element name="activeStatus" type="xs:integer"
                  db2-xdb:rowSet="AUTHORS" db2-xdb:column="statusCode"
                  db2-xdb:condition="$DECOMP_CONTENT=1" />
      <xs:attribute name="ID" type="xs:integer"
                  db2-xdb:rowSet="AUTHORS" db2-xdb:column="AUTHID"
                  db2-xdb:condition="$DECOMP_CONTENT BETWEEN 1 and 999 />
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

前述の <author> 要素の例の値により、db2-xdb:condition によって指定されたすべての条件が満たされているので、<author> 要素から AUTHORS 表にデータが追加されます。

表 52. AUTHORS

AUTHID	GIVENNAME	SURNAME	STATUSCODE	NUMBOOKS
0800	Alexander	Smith	1	NULL

1 つの条件が満たされない場合

以下のアノテーション付きスキーマは、作成者の ID が 1 から 100 までの間にあり、<firstname> および <lastname> 要素が NULL でない場合のみ <author> 要素を分解する必要があることを指定します。

```
<xs:element name="author">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="firstname" type="xs:string"
                  db2-xdb:rowSet="AUTHORS" db2-xdb:column="GIVENNAME"
                  db2-xdb:condition="$DECOMP_CONTENT IS NOT NULL"/>
      <xs:element name="lastname" type="xs:string"
                  db2-xdb:rowSet="AUTHORS" db2-xdb:column="SURNAME"
                  db2-xdb:condition="$DECOMP_CONTENT IS NOT NULL"/>
      <xs:element name="activeStatus" type="xs:integer" />
      <xs:attribute name="ID" type="xs:integer"
                    db2-xdb:rowSet="AUTHORS" db2-xdb:column="AUTHID"
                    db2-xdb:condition="$DECOMP_CONTENT BETWEEN 1 and 100 />
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

例の <author> 要素の <firstname> および <lastname> 要素は指定された条件を満たしていますが、ID 属性の値は満たしていないので、分解時に行全体が挿入されません。その理由は、AUTHORS 表に関して指定されている 3 つの条件すべての論理 AND が評価されるからです。この場合、条件の 1 つが false なので、論理 AND は false に評価されるため、行は挿入されません。

db2-xdb:contentHandling 分解アノテーション

db2-xdb:contentHandling アノテーションは、表に分解する、複合タイプまたは単純タイプの要素の内容のタイプを指定します。

db2-xdb:contentHandling は、XML 文書中の要素と属性を DB2 の基本表へマッピングする方法を記述するために、XML スキーマ文書に追加できる分解アノテーションのセットに属します。分解プロセスでは、XML 文書の要素と属性を分解して DB2 表に入れる方法を決定するために、アノテーション付き XML スキーマを使用します。

アノテーションの種類

複合タイプまたは単純タイプの要素宣言に適用される <xs:element> の属性または <db2-xdb:rowSetMapping> の属性。

指定方法

db2-xdb:contentHandling は次のいずれかの方法によって指定されます (*value* はアノテーションの有効な値を表します)。

- `<xs:element db2-xdb:contentHandling="value" />`

- <db2-xdb:rowSetMapping db2-xdb:contentHandling="*value*">
    <db2-xdb:rowSet>*value*</db2-xdb:rowSet>
    ...
  </db2-xdb:rowSetMapping>

名前空間

http://www.ibm.com/xmlns/prod/db2/xdb1

有効な値

大/小文字の区別がある以下のトークンのいずれか。

- text
- stringValue
- serializeSubtree

詳細情報

XML 要素の宣言に属性として指定された db2-xdb:contentHandling アノテーションは、分解時に、db2-xdb:rowSet によって指定された表や db2-xdb:column によって指定された列に挿入する値を示します。以下の 3 つの値が db2-xdb:contentHandling にとって有効です。

text

- この要素内の文字データの連結 (CDATA セクションの文字内容を含む) が挿入されます。
- この要素のコメントと処理命令、CDATA セクションの区切り文字 ("<![CDATA[" "]]>")、およびこの要素の子孫 (タグと内容を含む) が除外されます。

stringValue

- この要素の文字データの連結 (CDATA セクションの文字内容を含む) と、この要素の子孫の文字データが、文書順に挿入されます。
- コメント、処理命令、CDATA セクションの区切り文字 ("<![CDATA[" "]]>")、およびこの要素の子孫の開始タグと終了タグが除外されます。

serializeSubtree

- この要素の開始タグと終了タグの間にあるすべてのもののマークアップ (この要素の開始タグと終了タグを含む) が挿入されます。コメント、処理命令、および CDATA セクションの区切り文字 ("<![CDATA[" "]]>") が含まれます。
- 除外されるものはありません。
-

  注: 挿入されるシリアライズ化ストリングは、XML 文書内の対応するセクションと同一でない可能性があります。その要因には、XML スキーマに指定されているデフォルト値、エンティティーの拡張、属性の順序、属性の空白文字の正規化、CDATA セクションの処理などがあります。

  この設定の結果のシリアライズ化ストリングは XML エンティティーなので、コード・ページの問題を考慮する必要があります。ターゲット列の

タイプが文字またはグラフィックの場合は、XML フラグメントがデータベースのコード・ページで挿入されます。アプリケーションがこの種のエンティティーを XML プロセッサーに渡す際には、そのアプリケーションはエンティティーのエンコード方式をこのプロセッサーに明示的に通知しなければなりません。その理由は、このプロセッサーは UTF-8 以外のエンコード方式を自動的に検出しないからです。しかし、ターゲット列のタイプが BLOB の場合は、XML エンティティーが UTF-8 エンコード方式で挿入されます。この場合、エンコード方式を指定しなくても XML エンティティーを XML プロセッサーに渡すことができます。

分解のアノテーションが付けられている XML 要素宣言が、複合タイプであり、複合内容を含むが、db2-xdb:contentHandling が指定されていない場合、デフォルトの動作は「serializeSubtree」設定に従います。他のすべてのアノテーションが付けられている要素宣言の場合、db2-xdb:contentHandling が指定されていない場合のデフォルトの動作は、「stringValue」設定に従います。

要素が複合タイプと宣言されており、要素のみ、または空の内容モデルを持つ (つまり要素宣言の「混合」属性が true または 1 に設定されていない) 場合、db2-xdb:contentHandling は "text" に設定できません。

要素に関する db2-xdb:contentHandling アノテーションを指定しても、その要素の子孫の分解には影響しません。

db2-xdb:contentHandling の設定は、db2-xdb:expression または db2-xdb:condition アノテーションにおいて $DECOMP_CONTENT と置換される値に影響します。置換値は、まず db2-xdb:contentHandling 設定に従って処理されてから、次に評価のために渡されます。

分解前か分解中に妥当性検査が実行されている場合は、db2-xdb:contentHandling によって処理される内容のエンティティーはすでに解決済みになることに注意してください。

例

以下の例は、db2-xdb:contentHandling アノテーションのさまざまな設定を使用して、ターゲット表にさまざまな結果を生じさせられる方法を示しています。最初にアノテーション付きスキーマを掲げ、db2-xdb:contentHandling を使用して <paragraph> 要素にアノテーションを付ける方法を示します。 (アノテーション付きスキーマとして、db2-xdb:contentHandling が "text" に設定された例のみ掲示しています。この節の 2 つ目以降の例では、同じアノテーション付きスキーマで、db2-xdb:contentHandling の設定値だけが違っていることを前提にしています。)

```
<xs:schema>
  <xs:element name="books">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="book">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="authorID" type="xs:integer" />
              <xs:element name="chapter" type="chapterType" maxOccurs="unbounded" />
            </xs:sequence>
            <xs:attribute name="isbn" type="xs:string"
                          db2-xdb:rowSet="BOOKCONTENTS" db2-xdb:column="ISBN" />
```

```
          <xs:attribute name="title" type="xs:string" />
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:complexType name="chapterType">
  <xs:sequence>
    <xs:element name="paragraph" type="paragraphType" maxOccurs="unbounded"
                db2-xdb:rowSet="BOOKCONTENTS" db2-xdb:column="CHPTCONTENT"
                db2-xdb:contentHandling="text" />
  </xs:sequence>
  <xs:attribute name="number" type="xs:integer"
                db2-xdb:rowSet="BOOKCONTENTS" db2-xdb:column="CHPTNUM" />
  <xs:attribute name="title" type="xs:string"
                db2-xdb:rowSet="BOOKCONTENTS" db2-xdb:column="CHPTTITLE" />
</xs:complexType>

<xs:complexType name="paragraphType" mixed="1">
  <xs:choice>
    <xs:element name="b" type="xs:string" minOccurs="0" maxOccurs="unbounded" />
  </xs:choice>
</xs:complexType>
</xs:schema>
```

次に、マップされる <books> 要素を示します。

```
<books>
  <book isbn="1-11-111111-1" title="My First XML Book">
    <authorID>22</authorID>
    <chapter number="1" title="Introduction to XML">
      <paragraph>XML is <b>lots of</b> fun...</paragraph>
    </chapter>
    <chapter number="2" title="XML and Databases">
      <paragraph><!-- Start of chapter -->XML can be used with...</paragraph>
      <paragraph><?processInstr example?>
        Escape characters such as <![CDATA[ <, >, and & ]]>...</paragraph>
    </chapter>
    ...
    <chapter number="10" title="Further Reading">
      <paragraph>Recommended tutorials...</paragraph>
    </chapter>
  </book>
  ...
<books>
```

次の 3 つの表は、db2-xdb:contentHandling の値が異なる以外は同じ XML 要素を分解した結果を示しています。

注: 以下の表の CHPTTITLE および CHPTCONTENT 列には値の前後に引用符が付いています。これらの引用符は列内にはありませんが、挿入されるストリングの境界と空白を示すためだけに付けられています。

db2-xdb:contentHandling="text"

表 53. BOOKCONTENTS

ISBN	CHPTNUM	CHPTTITLE	CHPTCONTENT
1-11-111111-1	1	"Introduction to XML"	"XML is fun..."
1-11-111111-1	2	"XML and Databases"	"XML can be used with..."

表 53. BOOKCONTENTS (続き)

ISBN	CHPTNUM	CHPTTITLE	CHPTCONTENT
1-11-111111-1	2	"XML and Databases"	" Escape characters such as <, >, and & ..."
...	...	...	...
1-11-111111-1	10	"Further Reading"	"Recommended tutorials..."

"text" 設定の使用時には、第 1 章の最初の段落の 要素の内容が挿入されていないことに注意してください。その理由は、"text" 設定は子孫からの内容を除外するからです。"text" 設定の使用時には、第 2 章の最初の段落からコメントと処理命令が除外されることにも注意してください。<paragraph> 要素からの文字データの連結の空白文字は保存されます。

db2-xdb:contentHandling="stringValue"

表 54. BOOKCONTENTS

ISBN	CHPTNUM	CHPTTITLE	CHPTCONTENT
1-11-111111-1	1	"Introduction to XML"	"XML is lots of fun..."
1-11-111111-1	2	"XML and Databases"	"XML can be used with..."
1-11-111111-1	2	"XML and Databases"	" Escape characters such as <, >, and & ..."
...	...	...	...
1-11-111111-1	10	"Further Reading"	"Recommended tutorials..."

この表と前の表との間の違いは、最初の行の CHPTCONTENT 列にあります。ストリング "lots of" (<paragraph> 要素の 子孫に由来する) が挿入されている様子に注意してください。db2-xdb:contentHandling が "text" に設定されていた場合には、"text" 設定は子孫の内容を除外するので、このストリングは除外されていました。しかし、"stringValue" 設定は、子孫からの内容を組み込みます。"text" 設定と同様に、コメントと処理命令は挿入されませんが、空白文字は保存されます。

db2-xdb:contentHandling="serializeSubtree"

表 55. BOOKCONTENTS

ISBN	CHPTNUM	CHPTTITLE	CHPTCONTENT
1-11-111111-1	1	"Introduction to XML"	"<paragraph>XML is lots of fun...</paragraph>"
1-11-111111-1	2	"XML and Databases"	"<paragraph><!-- Start of chapter -->XML can be used with...</paragraph>"

表 55. BOOKCONTENTS (続き)

ISBN	CHPTNUM	CHPTTITLE	CHPTCONTENT
1-11-111111-1	2	"XML and Databases"	"<paragraph><?processInstr example?> Escape characters such as <![CDATA[<, >, and &]]>...</paragraph>"
...	...	...	...
1-11-111111-1	10	"Further Reading"	"<paragraph>Recommended tutorials...</paragraph>"

この表と前の 2 つの表の違いは、<paragraph> 要素の子孫からのマークアップがすべて挿入されることです (<paragraph> の開始タグと終了タグを含む)。この表では、最初の行の CHPTCONTENT 列に <b> の開始タグと終了タグが含まれており、2 行目にコメントが含まれており、3 行目に処理命令が含まれています。前の 2 つの例と同様に、XML 文書からの空白文字は保存されています。

db2-xdb:normalization 分解アノテーション

db2-xdb:normalization アノテーションは、挿入または $DECOMP_CONTENT に置換される (db2-xdb:expression とともに使用される場合) XML データ内の空白の正規化を指定します。

db2-xdb:normalization は、XML 文書中の要素と属性を DB2 の基本表へマッピングする方法を記述するために、XML スキーマ文書に追加できる分解アノテーションのセットに属します。分解プロセスでは、XML 文書の要素と属性を分解して DB2 表に入れる方法を決定するために、アノテーション付き XML スキーマを使用します。

アノテーションの種類

<xs:element> または <xs:attribute> の属性、あるいは <db2-xdb:rowSetMapping> の属性

指定方法

db2-xdb:normalization は次のいずれかの方法によって指定されます (*value* はアノテーションの有効な値を表します)。

- `<xs:element db2-xdb:normalization="value" />`
- `<xs:attribute db2-xdb:normalization="value" />`
- ```
  <db2-xdb:rowSetMapping db2-xdb:normalization="value">
    <db2-xdb:rowSet>value</db2-xdb:rowSet>
    ...
  </db2-xdb:rowSetMapping>
  ```

名前空間

http://www.ibm.com/xmlns/prod/db2/xdb1
  ```

有効な値

大/小文字の区別がある以下のトークンのいずれか。

- canonical
- original (デフォルト)
- whitespaceStrip

注: db2-xdb:normalization 属性は、特定の XML スキーマ・タイプと SQL 文字タイプの間のマッピングでのみ有効です。 SQL 文字列に関して正規化できるサポート XML スキーマ・タイプのリストについては、「詳細情報」セクションを参照してください。

詳細情報

XML 値を文字タイプのターゲット列 (CHAR、VARCHAR、LONG VARCHAR、CLOB、DBCLOB、GRAPHIC、VARGRAPHIC、LONG VARGRAPHIC) に挿入する場合は、挿入されるデータを正規化する必要があるかもしれません。
db2-xdb:normalization アノテーションを使用すると、異なるタイプの正規化を指定することができます。有効な値 (大/小文字の区別あり) は次のとおりです。

canonical
XML 値は、XML スキーマ・タイプに応じてその正規形に変換されます。その後ターゲット列に挿入されるか、または db2-xdb:normalization アノテーションと同じマッピング内に $DECOMP_CONTENT が指定されている場合は、その指定で置き換えられます。

original
XML 値は、XML パーサー処理の後、(このマッピング対象が XML 要素かまたは XML 属性かにより) 要素内容または属性値の元の文字データのまま扱われます。その後ターゲット列に挿入されるか、または db2-xdb:normalization アノテーションと同じマッピング内に $DECOMP_CONTENT が指定されている場合は、その指定で置き換えられます。このアノテーションが関連するマッピングの db2-xdb:normalization 属性が指定されない場合、分解プロセスは「original」設定に従ってデータを正規化します。

whitespaceStrip
XML 値の前後の空白は除去され、連続する空白は 1 つの空白文字に縮小されます。その後ターゲット列に挿入されるか、または db2-xdb:normalization アノテーションと同じマッピング内に $DECOMP_CONTENT が指定されている場合は、その指定で置き換えられます。

次のアトミック XML スキーマ・タイプのうちのいずれかの要素または属性が文字タイプ (CHAR、VARCHAR、LONG VARCHAR、CLOB、DBCLOB、GRAPHIC、VARGRAPHIC、および LONG VARGRAPHIC) の列にマップされる (または派生する) ときに、db2-xdb:normalization が適用可能になります。

- byte、unsigned byte
- integer、positiveInteger、 negativeInteger、 nonPositiveInteger、 nonNegativeInteger
- int、unsignedInt

- long、unsignedLong
- short、unsignedShort
- decimal
- float
- double
- boolean
- time
- date
- dateTime

これ以外のタイプに対して指定される場合、db2-xdb:normalization は無視されます。これらは、W3C 勧告 XML Schema Part 2: Datatypes Second Edition が正規表現を持つ XML スキーマ・タイプです。

db2-xdb:normalization アノテーションは特定の XML スキーマから SQL 文字タイプへのマッピングでのみ有効であるため、サポートされないマッピングでアノテーションが指定されると無視されます。

例

以下の例は、db2-xdb:normalization アノテーションを使って空白の正規化を制御する方法を示しています。アノテーション付きのスキーマがまず示されています。

```
      <xs:element name="author">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="firstname" type="xs:string"
                        db2-xdb:rowSet="AUTHORS" db2-xdb:column="FIRSTNAME" />
            <xs:element name="lastname" type="xs:string"
                        db2-xdb:rowSet="AUTHORS" db2-xdb:column="SURNAME"
                        db2-xdb:normalization="whitespaceStrip" />
            <xs:element name="activeStatus" type="xs:boolean"
                        db2-xdb:rowSet="AUTHORS" db2-xdb:column="ACTIVE"
                        db2-xdb:normalization="canonical" />
            <xs:attribute name="ID" type="xs:integer"
                        db2-xdb:rowSet="AUTHORS" db2-xdb:column="AUTHID"
                        db2-xdb:normalization="whitespaceStrip" />
   </xs:sequence>
        </xs:complexType>
      </xs:element>
```

次に、マップされている <author> 要素を示します (次の例の中で、注目する空白文字は明示するために下線文字 '_' によって表されています)。続いて分解完了後に生成される AUTHORS 表を示します。

```
 <author ID="__22">
    <firstname>Ann</firstname>
    <lastname>__Brown_</lastname>
    <activeStatus>1</activeStatus>
  </author>
```

表 56. AUTHORS

AUTHID	FIRSTNAME	SURNAME	ACTIVE	NUMBOOKS
22	Ann	__Brown_	true	NULL

「whitespaceStrip」に設定すると、値をターゲット表に挿入する前に、「ID」属性から先頭の空白が除去されます。ただし、「whitespaceStrip」設定が指定されたとしても、<lastname> 要素前後の空白は除去されません。これは <lastname> 要素が XML スキーマ・タイプのストリングを持つためです。このタイプは db2-xdb:normalization には適用できません。 <author> の子要素 <activeStatus> は boolean タイプとして定義され、 boolean タイプの正規表現はリテラル「true」または「false」のいずれかです。 <activeStatus> 要素の「canonical」設定では正規形の「1」になり、これは「true」で、AUTHORS 表の ACTIVE 列に挿入されます。

上記で提示した XML スキーマの「ID」属性が代わりに db2-xdb:normalization="original" を使ってアノテーションを付けられていたとしたら、文書からの元の値 "__22" (下線文字は空白を表す) は AUTHID 列に挿入されていたはすです。

db2-xdb:order 分解アノテーション

db2-xdb:order アノテーションは、異なる表の間で行を挿入する順序を指定します。

db2-xdb:order は、XML 文書中の要素と属性を DB2 の基本表へマッピングする方法を記述するために、XML スキーマ文書に追加できる分解アノテーションのセットに属します。分解プロセスでは、XML 文書の要素と属性を分解して DB2 表に入れる方法を決定するために、アノテーション付き XML スキーマを使用します。

アノテーションの種類

<db2-xdb:rowSetOperationOrder> の子要素

指定方法

db2-xdb:order は次のように指定されます (*value* はアノテーションの有効な値を表します)。

```
<xs:schema>
  <xs:annotation>
    <xs:appinfo>
      <db2-xdb:rowSetOperationOrder>
        <db2-xdb:order>
          <db2-xdb:rowSet>value</db2-xdb:rowSet>
          ...
        </db2-xdb:order>
      </db2-xdb:rowSetOperationOrder>
    </xs:appinfo>
  </xs:annotation>
  ...
</xs:schema>
```

名前空間

http://www.ibm.com/xmlns/prod/db2/xdb1

有効な構造

以下の <db2-xdb:order> の子要素がサポートされています。

db2-xdb:rowSet

ターゲットの基本表への XML 要素または属性のマッピングを指定します。

詳細情報

db2-xdb:order アノテーションは、特定の rowSet に属する行の挿入について、別の rowSet に属する行の挿入と比較したときの順序を定義するために使用されます。これを使用することにより、ターゲット表に対してリレーショナル・スキーマの一部として定義されたすべての参照整合性制約と整合するように XML データをターゲット表に挿入できます。

特定の rowSet RS1 のすべての行は、db2-xdb:order の中で RS1 が RS2 の前にリストされている場合には、別の rowSet RS2 に属するすべての行の前に挿入されます。複数の挿入順序階層を定義するために、この要素のインスタンスを複数指定できます。どの要素にも出現しない rowSet については、それらの行は、他のいずれかの rowSet の行との比較において任意の順序で挿入できます。また、各 <db2-xdb:rowSet> 要素の内容は、明示的に定義された rowSet か、または明示的な rowSet 宣言が行われていない既存の表の名前のいずれかである必要があります。

rowSet 挿入階層は複数定義できます。ただし、rowSet は <db2-xdb:order>要素の 1 つのインスタンスにしか出現できず、その要素の中で一度しか出現できません。

子要素の中で指定される区切り付き SQL ID の場合、引用符の区切り文字はエスケープする必要がなく、文字内容の中に含める必要があります。しかし、SQL ID で使用される「&」文字と「<」文字はエスケープする必要があります。

例

以下の例は、db2-xdb:order アノテーションの使用を示しています。

```
<xs:schema>
  <xs:annotation>
    <xs:appinfo>
      <db2-xdb:rowSetOperationOrder>

        <db2-xdb:order>
          <db2-xdb:rowSet>CUSTOMER</db2-xdb:rowSet>
          <db2-xdb:rowSet>PURCHASE_ORDER</db2-xdb:rowSet>
        </db2-xdb:order>

        <db2-xdb:order>
          <db2-xdb:rowSet>ITEMS_MASTER</db2-xdb:rowSet>
          <db2-xdb:rowSet>PO_ITEMS</db2-xdb:rowSet>
        </db2-xdb:order>

      </db2-xdb:rowSetOperationOrder>
    </xs:appinfo>
  </xs:annotation>
</xs:schema>
```

この例では、2 つの背反する挿入順序の階層が指定されています。最初の階層は、CUSTOMER rowSet または表のすべての内容が PURCHASE_ORDER のために収集されたどの内容よりも前に挿入されることを指定し、2 番目の階層は、ITEMS_MASTER rowSet または表のすべての内容が、なんらかの内容が PO_ITEMS に挿入される前に挿入されることを指定しています。ここで 2 つの階層間の順序は

未定義である点に注目してください。例えば、なんらかの内容が ITEMS_MASTER に挿入される前であっても後であっても、PURCHASE_ORDER rowSet または表の任意の内容を挿入できます。

制約事項

rowSet 挿入の順序の指定は、以下の制約事項に従います。

- 32 ビット・システムでは、挿入順序要件がある大規模な文書の分解時に、システムがメモリー不足になる場合があります。
- 64 ビット・システムでは、処理のために許可する仮想メモリー・スペースを管理者が制限した場合には、メモリー不足状態が発生する場合があります。処理のために十分に大きい、または無制限の仮想メモリー設定値を指定すると、メモリー不足状態を回避する助けになる反面、システムの全体的なパフォーマンスには悪影響を及ぼす場合があります。

db2-xdb:truncate 分解アノテーション

db2-xdb:truncate アノテーションは、XML 値を文字ターゲット列に挿入するときに切り捨てを許可するかどうかを指定します。

db2-xdb:truncate は、XML 文書中の要素と属性を DB2 の基本表へマッピングする方法を記述するために、XML スキーマ文書に追加できる分解アノテーションのセットに属します。分解プロセスでは、XML 文書の要素と属性を分解して DB2 表に入れる方法を決定するために、アノテーション付き XML スキーマを使用します。

アノテーションの種類

<xs:element> または <xs:attribute> の属性、あるいは <db2-xdb:rowSetMapping> の属性

指定方法

db2-xdb:truncate は次のいずれかの方法によって指定されます (*value* はアノテーションの有効な値を表します)。

- `<xs:element db2-xdb:truncate="value" />`
- `<xs:attribute db2-xdb:truncate="value" />`
- ```
  <db2-xdb:rowSetMapping db2-xdb:truncate="value">
    <db2-xdb:rowSet>value</db2-xdb:rowSet>
    ...
  </db2-xdb:rowSetMapping>
  ```

名前空間

http://www.ibm.com/xmlns/prod/db2/xdb1

有効な値

以下のうちのいずれかのトークンです。

- 0 (false に相当。デフォルト)
- 1 (true に相当)
- false (大/小文字の区別あり。デフォルト)
  ```

- true (大/小文字の区別あり)

詳細情報

ターゲット文字列に挿入されている XML 値が列サイズよりも大きい場合があるかもしれません。その場合、分解を正常に行うために値を切り捨てる必要があります。 db2-xdb:truncate 属性は、値がターゲット列に対して大きすぎる場合に切り捨てが許可されるかどうかを示します。この属性が「false」または「0」に設定されて切り捨てが許可されないことを示し、挿入されている XML 値がターゲット列に対して大きすぎる場合は、XML 文書の分解中にエラーが発生し、値は挿入されません。「true」または「1」の設定は、挿入中にデータ切り捨てが許可されることを示します。

db2-xdb:truncate はターゲット列が次のいずれかのタイプの場合にのみ適用できます。

- 文字タイプ。
- DATE、TIME、または TIMESTAMP タイプで、XML 値のタイプがそれぞれ xs:date、xs:time、または xs:dateTime。

db2-xdb:truncate と同じ要素または属性の宣言に db2-xdb:expression アノテーションが指定される場合、切り捨て可能として式が定義されていれば切り捨てを実行できるので、db2-xdb:truncate の値は無視されます。

時間帯を指定し、XML スキーマ・タイプが date、time、または timestamp である XML 値を SQL datetime 列に分解するときは、db2-xdb:truncate を「true」または「1」に設定する必要があります。これは、SQL datetime タイプの構造が時間帯指定を提供しないためです。

例

以下の例は、<author> 要素に対してどのように切り捨てを適用できるかを示しています。アノテーション付きスキーマのセクションがまず示されています。

```
<xs:element name="author">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="firstname" type="xs:string"
                    db2-xdb:rowSet="AUTHORS" db2-xdb:column="FIRSTNAME"
                    db2-xdb:truncate="true" />
      <xs:element name="lastname" type="xs:string" />
      <xs:element name="activeStatus" type="xs:boolean" />
      <xs:element name="activated" type="xs:date"
                    db2-xdb:truncate="true" />
      <xs:attribute name="ID" type="xs:integer" />
    <xs:sequence>
  </xs:complexType>
</xs:element>
```

マップされている <author> 要素が次に示されています。

```
<author ID="0800">
  <firstname>Alexander</firstname>
  <lastname>Smith</lastname>
  <activeStatus>0</activeStatus>
  <activated>2001-10-31Z</activated>
</author>
```

FIRSTNAME 列のタイプが CHAR SQL、サイズが 7、ACTIVEDATE 列のタイプが DATE SQL タイプに定義されているとします。分解完了後に生成される AUTHORS 表が次に示されています。

表 57. AUTHORS

AUTHID	FIRSTNAME	SURNAME	ACTIVE	ACTIVEDATE	NUMBOOKS
NULL	Alexand	NULL	NULL	2001-10-31	NULL

<firstname> 値「Alexander」は SQL 列サイズより大きいため、値を挿入するために切り捨てを行う必要があります。 XML 文書の <activated> 要素に時間帯が含まれているため、分解中に日付が確実に挿入されるよう、db2-xdb:truncate が「true」に設定されている点にも注目できます。

<firstname> 要素または <activated> 要素からの値を挿入するには切り捨てが必要なため、db2-xdb:truncate が指定されないと db2-xdb:truncate のデフォルト値が想定され (切り捨ては許可されない)、行が挿入されなかったことを示すエラーが生成されることになります。

db2-xdb:rowSetMapping 分解アノテーション

<db2-xdb:rowSetMapping> アノテーションは、単一の XML 要素または属性を、1 つ以上の列と表の対にマップします。

<db2-xdb:rowSetMapping> は、XML 文書中の要素と属性を DB2 の基本表へマッピングする方法を記述するために、XML スキーマ文書に追加できる分解アノテーションのセットに属します。分解プロセスでは、XML 文書の要素と属性を分解して DB2 表に入れる方法を決定するために、アノテーション付き XML スキーマを使用します。

アノテーションの種類

<xs:element> または <xs:attribute> の子要素である <xs:appinfo> (<xs:annotation> の子要素) の子要素

指定方法

db2-xdb:rowSetMapping は次のいずれかの方法によって指定されます (*value* はアノテーションの有効な値を表します)。

- ```
  <xs:element>
    <xs:annotation>
      <xs:appinfo>
        <db2-xdb:rowSetMapping>
          <db2-xdb:rowSet>value</db2-xdb:rowSet>
          ...
        </db2-xdb:rowSetMapping>
      </xs:appinfo>
    </xs:annotation>
    ...
  </xs:element>
  ```
- ```
  <xs:attribute>
    <xs:annotation>
      <xs:appinfo>
        <db2-xdb:rowSetMapping>
          <db2-xdb:rowSet>value</db2-xdb:rowSet>
          ...
  ```

```
        </db2-xdb:rowSetMapping>
      </xs:appinfo>
    </xs:annotation>
    ...
  </xs:attribute>
```

名前空間

http://www.ibm.com/xmlns/prod/db2/xdb1

有効な構造

以下の <db2-xdb:rowSetMapping> の属性がサポートされています。

db2-xdb:contentHandling
複合タイプの要素の表に分解される内容のタイプを指定できるようにします。

db2-xdb:locationPath
再利用可能グループの一部として宣言される XML 要素または属性を、その祖先に応じてさまざまな表と列の対にマップできるようにします。

db2-xdb:normalization
文字ターゲット列にマップされる XML 要素または属性の内容を挿入する前に、その内容の正規化動作を指定できるようにします。

db2-xdb:truncate
XML 値を文字ターゲット列に挿入するときに切り捨てを許可するかどうかを指定できるようにします。

<db2-xdb:rowSetMapping> のこれらの属性は、XML 要素または属性の宣言の属性としても使用できます。これらの属性が、<db2-xdb:rowSetMapping> の属性か、<xs:element> または <xs:attribute> の属性かにかかわらず、設定対象に対し同じ動作と用件が当てはまります。これらのアノテーションについて詳しくは、対応する個々の文書を参照してください。

以下は、サポートされる <db2-xdb:rowSetMapping> の子要素です。指定される場合、その子要素が現れる順にリストされています。

<db2-xdb:rowSet>
XML 要素または属性をターゲットの基本表にマップします。

<db2-xdb:column>
(任意指定) XML 要素または属性を基本表の列にマップします。この要素は、db2-xdb:expression が db2-xdb:rowSetMapping のアノテーションに存在する場合には必須です。

<db2-xdb:column> が任意指定となり得るのは、値が表に挿入されないものの、それが条件付き処理でのみ使用されるような場合です。例えば、1 つの要素が別の要素の値に基づいて分解される場合には、その他方の要素は列のマッピングを必要としません。その値は挿入されないからです。

<db2-xdb:expression>
(任意指定) カスタマイズされた式を指定します。その結果は db2-xdb:rowSet 属性によって指定される表に挿入されます。

db2-xdb:expression が $DECOMP_CONTENT を指定し、その同じマッピングで db2-xdb:normalization が指定される場合、db2-xdb:expression の $DECOMP_CONTENT 値は評価のために式に渡される前に正規化されます (適用可能な場合)。

<db2-xdb:condition>
(任意指定) 評価の条件を指定します。

<db2-xdb:rowSetMapping> のこれらの子要素は、引用符をエスケープする必要がないという点を除いて、それに対応する属性注釈と同じセマンティクスと構文を持ちます。

詳しくは、これらのアノテーションの属性バージョンの対応する資料を参照してください。

詳細情報

<db2-xdb:rowSetMapping> を使用することにより、XML 要素または属性を 1 つのターゲット表または列、同じ表の複数のターゲット列、あるいは複数の表および列にマップすることができます。次の 2 つのメソッドは、1 つの表または列に同じようにマッピングします。 1 つは db2-xdb:rowSet アノテーションと db2-xdb:column アノテーション (マップされている要素または属性の属性) を組み合わせるメソッドで、もう 1 つは <db2-xdb:rowSetMapping> (マップされている要素または属性の子要素) を指定するメソッドです。どちらのメソッドも同じ結果を生成します。違うのはその表記の仕方だけです。

<db2-xdb:rowSetMapping> の子要素の文字内容の中の空白にはすべて意味があります。空白の正規化は実行されません。子要素の中で指定される区切り付き SQL ID の場合、引用符の区切り文字はエスケープするのではなく、文字内容の中に含める必要があります。しかし、SQL ID で使用される「&」文字と「<」文字はエスケープする必要があります。

例

以下の例は、<db2-xdb:rowSetMapping> アノテーションを使用して、「isbn」という 1 つの属性を複数の表にマップする方法を示しています。アノテーション付きスキーマのセクションがまず示されています。 isbn 値を BOOKS 表と BOOKCONTENTS 表の両方にマップする方法を示しています。

```
<xs:element name="book">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="authorID" type="xs:integer"/>
      <xs:element name="chapter" type="chapterType" maxOccurs="unbounded" />
    </xs:sequence>
    <xs:attribute name="isbn" type="xs:string">
      <xs:annotation>
        <xs:appinfo>
          <db2-xdb:rowSetMapping>
            <db2-xdb:rowSet>BOOKS</db2-xdb:rowSet>
            <db2-xdb:column>ISBN</db2-xdb:column>
          </db2-xdb:rowSetMapping>
          <db2-xdb:rowSetMapping>
            <db2-xdb:rowSet>BOOKCONTENTS</db2-xdb:rowSet>
            <db2-xdb:column>ISBN</db2-xdb:column>
          </db2-xdb:rowSetMapping>
```

```
        </xs:appinfo>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="title" type="xs:string" />
  </xs:complexType>
</xs:element>
```

マップされている <book> 要素が次に示され、その後に分解完了後の BOOKS 表および BOOKCONTENTS 表が示されています。

```
<book isbn="1-11-111111-1" title="My First XML Book">
  <authorID>22</authorID>
  <!-- this book does not have a preface -->
  <chapter number="1" title="Introduction to XML">
    <paragraph>XML is fun...</paragraph>
    ...
  </chapter>
  ...
</book>
```

表 58. BOOKS

ISBN	TITLE	CONTENT
1-11-111111-1	NULL	NULL

表 59. BOOKCONTENTS

ISBN	CHPTNUM	CHPTTITLE	CHPTCONTENT
1-11-111111-1	NULL	NULL	NULL

<db2-xdb:rowSetMapping>、db2-xdb:rowSet、および db2-xdb:column の組み合わせを使った代替マッピング

以下のアノテーション付きスキーマのセクションは上記で示されている XML スキーマのフラグメントと同等のもので、同じ分解の結果を生成します。 2 つのスキーマの違いは、以下のスキーマが 1 つのマッピングを、 <db2-xdb:rowSetMapping> アノテーションの使用のみで置換を行うのではなく、 db2-xdb:rowSet と db2-xdb:column の組み合わせで置き換えるという点です。

```
<xs:element name="book">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="authorID" type="xs:integer"/>
      <xs:element name="chapter" type="chapterType" maxOccurs="unbounded" />
    </xs:sequence>
    <xs:attribute name="isbn" type="xs:string"
                  db2-xdb:rowSet="BOOKS" db2-xdb:column="ISBN" >
      <xs:annotation>
        <xs:appinfo>
          <db2-xdb:rowSetMapping>
            <db2-xdb:rowSet>BOOKCONTENTS</db2-xdb:rowSet>
            <db2-xdb:column>ISBN</db2-xdb:column>
          </db2-xdb:rowSetMapping>
        </xs:appinfo>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="title" type="xs:string" />
  </xs:complexType>
</xs:element>
```

db2-xdb:rowSetOperationOrder 分解アノテーション

db2-xdb:rowSetOperationOrder アノテーションは、1 つ以上の db2-xdb:order 要素の親です。異なる表の間で行を挿入する順序を定義する場合の使用法の詳細については、db2-xdb:order のセクションを参照してください。

db2-xdb:rowSetOperationOrder は、XML 文書中の要素と属性を DB2 の基本表へマッピングする方法を記述するために、XML スキーマ文書に追加できる分解アノテーションのセットに属します。分解プロセスでは、XML 文書の要素と属性を分解して DB2 表に入れる方法を決定するために、アノテーション付き XML スキーマを使用します。

アノテーションの種類

グローバル <xs:annotation> 要素の子である <xs:appinfo> の子要素。

指定方法

db2-xdb:rowSetOperationOrder は次のように指定されます (*value* はアノテーションの有効な値を表します)。

```
<xs:schema>
  <xs:annotation>
    <xs:appinfo>
      <db2-xdb:rowSetOperationOrder>
        <db2-xdb:order>
          <db2-xdb:rowSet>value</db2-xdb:rowSet>
          ...
        </db2-xdb:order>
      </db2-xdb:rowSetOperationOrder>
    </xs:appinfo>
  </xs:annotation>
  ...
</xs:schema>
```

名前空間

http://www.ibm.com/xmlns/prod/db2/xdb1

有効な構造

以下の <db2-xdb:rowSetOperationOrder > の子要素がサポートされています。

db2-xdb:order

詳細情報

<db2-xdb:rowSetOperationOrder> は、<db2-xdb:order> 要素をグループ化します。<db2-xdb:order> 子要素のインスタンスを複数置いて、挿入の階層を複数定義することができます。

XML 文書の内容が挿入される順序を制御できるようにすることにより、db2-xdb:rowSetOperationOrder および db2-xdb:order アノテーションを一緒に使用すると、XML スキーマの分解プロセスにおいて、ターゲット表に対するすべての参照整合性制約だけでなく、ある表の行を別の表の行の前に挿入するというような他のアプリケーションの要件も確実に受け入れることができます。

db2-xdb:rowSetOperationOrder アノテーションは、XML スキーマで一度だけ使用できます。

例

rowSet 挿入の順序を指定する例については、db2-xdb:order アノテーションのセクションを参照してください。

アノテーション付き XML スキーマ分解のキーワード

アノテーション付き XML スキーマ分解は、db2-xdb:condition および db2-xdb:expression アノテーションで使用する分解キーワードを提供します。

$DECOMP_CONTENT

db2-xdb:contentHandling アノテーションの設定に従って構成される、文書にあるマップ済み XML 要素または属性の値。式で $DECOMP_CONTENT に置換される値は常に文字タイプと見なされる必要があります。サポートされる $DECOMP_CONTENT インスタンスの最大ストリング長および最大数については、制限および制約事項の資料を参照してください。
db2-xdb:expression が $DECOMP_CONTENT を指定し、その同じマッピングで db2-xdb:normalization が指定される場合、db2-xdb:expression の $DECOMP_CONTENT 値は評価のために式に渡される前に正規化されます (適用可能な場合)。

値を直接挿入するのではなく、カスタマイズされた式を使って、マップされた要素または属性の値を処理するために、$DECOMP_CONTENT を使用することができます。

$DECOMP_DOCUMENTID

xdbDecompXML ストアード・プロシージャーの *documentid* 入力パラメーターに指定されるストリング値。分解される XML 文書を識別します。文書が分解されると、xdbDecompXML ストアード・プロシージャーに提供される入力値が $DECOMP_DOCUMENTID に置換される値として使用されます。

アプリケーションは一意的に生成される文書 ID を xdbDecompXML に渡すことができます。これにより、これらの ID をデータベースに直接挿入することができます。要素または属性の固有 ID を生成する式にも ID を渡すことができます。したがって、$DECOMP_DOCUMENTID を使用すると、XML 文書に存在しない固有 ID を挿入することができます。

$DECOMP_ELEMENTID

XML 文書内でこのアノテーションが記述する要素または属性を一意的に識別するシステム生成の整数 ID。この値は、要素の追加、要素の削除、または文書順序内の要素の位置の変更によって文書に変更が加えられない限り、同じ XML 文書の分解操作の間では値は変更されません。これらの操作によって文書に変更が加えられ、再び分解される場合、要素の ID は前の分解後のものと同じにならない可能性があります。属性に対して指定される $DECOMP_ELEMENTID は、この属性が属する要素の $DECOMP_ELEMENTID の値として定義されます。

$DECOMP_ELEMENTID によって生成される値は、元の文書の要素の順序を示すためにも使用できます。XML 文書をリレーショナル表から再構成する必要がある場合にはこれが役に立つかもしれません。

アノテーション付き XML スキーマ分解で分解結果が形成される方法

典型的な分解プロセスでは XML 要素または属性内容の分解だけが行われますが、アノテーション付き XML スキーマ分解は、XML 文書に存在しない値の挿入もサポートしています。

分解された内容は以下のいずれかです。

- XML 文書の属性の値。
- XML 文書の要素の値。正確な内容は <db2-xdb:contentHandling> アノテーションの設定によって異なります。
  - text - この要素だけ (子孫ではない) から取り出された文字データ。
  - stringValue - この要素とその子孫から取り出された文字データ。
  - serializedSubtree - この要素の開始および終了タグの間にあるすべての内容のマークアップ。

  詳しくは、<db2-xdb:contentHandling> の資料を参照してください。
- XML 文書にあるマップ済み属性または要素の内容に基づく値。
- XML 文書内のあらゆる値から独立した、生成された値。

最後の 2 つの値は、db2-xdb:expression アノテーションによって得ることができます。このアノテーションでは、式を指定することができます。その結果は、分解の間に挿入されます。

XML 文書の値を式に適用して結果を生成することにより、データをターゲット列に挿入する前に変形することができます。式を使用して、マップされた要素または属性に基づく値 (要素の名前など) を生成することもできます。 db2-xdb:expression では、定数を指定することもできます。この定数は、XML 文書のマップされた値に関連したものであってもなくてもかまいません。db2-xdb:expression では、これらの任意の技法を組み合わせて、挿入用の値を生成できます。

この式は、関連付けられている要素や属性が XML 文書内で見つかるたびに呼び出されることに注意してください。

XML 分解結果の妥当性検査の効果

アノテーション付き XML スキーマ分解では、入力文書を妥当性検査することは必須ではありませんが、このことにはいくつかの利点があるため、分解前または分解中のどちらかに妥当性検査することが推奨されています。

妥当性検査は、(XMLVALIDATE SQL/XML 関数を使用して) 分解前に実行することもできますし、xdbDecompXML ストアード・プロシージャーまたは DECOMPOSE XML DOCUMENT コマンドに対する呼び出しの一部として分解中に実行することもできます。分解される XML 文書を妥当性検査すると、以下のことが確認されます。

- 指定された XML スキーマに従って文書全体が有効な場合のみ、値が表に分解される (有効な値のみデータベースに保管されることが確認される)
- 要素または属性の定義済みデフォルト値がデータベースに挿入されます。(要素または属性が XML 文書中になく、xdbDecompXML 分解ストアード・プロシージャーの 1 つを使用して妥当性検査が実行される場合)
- 妥当性検査が分解中に実行される場合、XML 文書中のすべてのエンティティーが解決されます。(分解前に XML 文書中のエンティティーが登録されていない場合は、エラーが戻される)
- スキーマの指定どおりに非デフォルトの空白文字の正規化が行われます。(xdbDecompXML 分解ストアード・プロシージャーの 1 つを使用して妥当性検査が実行される場合)

登録済みの XML スキーマに対して入力文書を妥当性検査することが推奨されています。その理由は、分解プロセスは対応するアノテーション付きスキーマに従って入力文書が有効であると想定するからです。妥当性検査を実行せず、しかも入力文書が無効な場合は、分解時に行われるエンティティーの解決やデフォルトの属性の追加などのために、(妥当性検査が実行された場合とは) 異なる行が挿入されることがあります。また、分解が予期しない結果になることもあります。無効な文書を分解した結果、および既存のデータに関する影響は定義されていません。

分解中に妥当性検査を実行する際には、非決定的な内容モデルなどのスキーマ内でのエラーや、誤ったタイプの派生により、分解プロセスが失敗する可能性があることに注意してください。分解を再試行する前に、アノテーション付きスキーマが正しいか検査し、そのスキーマを再登録してください。

アノテーション付き XML スキーマ分解での CDATA セクションの処理

分解アノテーションを付けた要素については、CDATA セクションの内容がデータベースに挿入されます。CDATA セクションの区切り文字 ("<![CDATA[" および "]]>") も挿入することができます。CDATA の内容は、XML パーサーによる行終了の正規化の対象になります。

属性 db2-xdb:contentHandling を使用しない場合、分解アノテーションを付けた要素については、CDATA セクションの内容がデータベースに挿入されます。CDATA セクションの区切り文字は挿入されません。

属性 db2-xdb:contentHandling="serializeSubtree" によって XML スキーマ中の XML 要素宣言にアノテーションを付けると、構文解析されていない XML 文書を分解するときに CDATA 区切り文字を含む CDATA セクションが挿入されます。

構文解析されていない XML と構文解析されている XML を分解した場合の相違点

属性 db2-xdb:contentHandling="serializeSubtree" によって対応する要素宣言にアノテーションが付けられている場合、CDATA セクションを分解するときに違いが生じます。結果は、入力 XML 文書が構文解析されている XML かどうかによって異な

ります。例えば、XML 文書が構文解析されていない XML として CLOB 列に保管されており、構文解析されている XML として XML 列に保管されているとします。

入力 XML 文書が非 XML タイプの列からのものである場合、分解結果には、要素の CDATA セクションの境界および元の内容が保持されます。入力文書が XML タイプの列からのものである場合、CDATA セクションは分解結果には保持されません。例えば、以下のフラグメントが含まれる XML 文書について考えてみましょう。

```
<a> before cdata <![CDATA[ in cdata & <>]]> after cdata</a>
```

文書が CLOB 列に保管されており、要素 a のマッピング内で db2-xdb:contentHandling="serializeSubtree" が指定されている場合、分解を行うことにより、要素 a にマップされる列に対して以下の結果が生成されます。

```
<a>before cdata <![CDATA[ in cdata & <>]]> after cdata</a>
```

XML 文書が XML タイプの列に保管されている場合、この XML フラグメントの分解結果は以下のようになります。

```
<a> before cdata in cdata &amp; < &gt; after cdata</a>
```

分解時の入力文書が XML タイプの列からのものである場合は常に、元の CDATA セクションは保持されません。元の CDATA セクションが保持されなくても、分解前と後のセクションは論理的に同等であるため、正確さに影響はありませんが、予想された出力とは異なる場合があります。

アノテーション付き XML スキーマ分解の NULL 値と空ストリング

アノテーション付き XML スキーマ分解では、特定の条件下で NULL 値か空ストリングが挿入されます。

XML 要素

以下の表は、XML 文書中の要素について、空ストリングまたは NULL 値がいつデータベースに挿入されるかを示しています。

表 60. マップされた要素に関する NULL の処理

条件	空ストリング	NULL 値
文書から要素が欠落している		X
要素が以下の条件をすべて満たしている: • 文書内にある • 開始タグに xsi:nil="true" または xsi:nil="1" 属性が含まれている		X

表 60. マップされた要素に関する NULL の処理 (続き)

条件	空ストリング	NULL 値
要素が以下の条件をすべて満たしている: • 文書内にあり、空である • 開始タグに `xsi:nil="true"` または `xsi:nil="1"` 属性が含まれていない • リスト・タイプ、ユニオン・タイプ、混合内容の複合タイプ、またはアトミック・ビルトイン・タイプ xsd:string、xsd:normalizedString、xsd:token、xsd:hexBinary、xsd:base64Binary、xsd:anyURI、xsd:anySimpleType から派生しているか、またはこれらのタイプになるように宣言されている (他のタイプの場合はエラーになる)	X	
注: 1. マッピングに db2-xdb:condition アノテーションまたは db2-xdb:expression アノテーションが関係している場合、空ストリングまたは NULL 値 (この表に示されているとおり) が式評価の引数として渡されます。 2. ターゲット列のタイプが CHAR または GRAPHIC の場合、空ストリングがブランク文字のストリングとして挿入されます。		

XML 属性

以下の表は、文書内の分解アノテーションが付けられた XML 属性が NULL 値を含む場合や欠落している場合に、空ストリングまたは NULL 値がいつデータベースに挿入されるかを示しています。

表 61. マップされた属性に関する NULL 処理

条件	空ストリング	NULL 値
文書から属性が欠落している (妥当性検査が実行されなかったか、または妥当性検査によってデフォルト値が提供されなかった)		X
属性が以下の条件をすべて満たしている: • 文書内にあり、空である • リスト・タイプ、ユニオン・タイプ、またはアトミック・ビルトイン・タイプ xsd:string、xsd:normalizedString、xsd:token、xsd:hexBinary、xsd:base64Binary、xsd:anyURI、xsd:anySimpleType から派生しているか、またはこれらのタイプになるように宣言されている (他のタイプの場合はエラーになる)	X	
注: マッピングに db2-xdb:condition アノテーションまたは db2-xdb:expression アノテーションが関係している場合、空ストリングまたは NULL 値 (この表に示されているとおり) が式評価の引数として渡されます。		

アノテーション付き XML スキーマ分解のチェックリスト

アノテーション付き XML スキーマ分解は複雑になってしまう可能性があります。この作業を管理しやすくするには、いくつかの事柄を考慮に入れる必要があります。

アノテーション付き XML スキーマ分解では、おそらく複数の XML 要素や属性をデータベース中の複数の列と表にマップする必要があります。このマッピングには、XML データを挿入する前に変換することや、挿入の条件を適用することが含まれる場合もあります。

XML スキーマにアノテーションを付ける際に考慮する項目と、関連資料を指すポインターを以下に示します。

- 使用できる分解アノテーションについて理解します。
- マッピング中に、列のタイプが、マップされる要素または属性の XML スキーマ・タイプと互換性があることを確認します。
- システム・メモリー・リソースに関する要求が最小限になるように、XML スキーマを構造化します。
- 制限または拡張によって派生した複合タイプのアノテーションが適切に付けられていることを確認します。
- 分解の制限と制約事項に違反していないことを確認します。
- スキーマを XSR に登録する時点で、アノテーション中で参照される表や列が存在することを確認します。

アノテーション付き XML スキーマ分解の場合の派生した複合タイプのアノテーション

分解に関する制限や拡張によって派生した複合タイプにアノテーションを付ける際には、追加のマッピングを適用する必要があります。

制限による派生

制限によって派生した複合タイプの場合、基本タイプの共通要素と属性が、派生タイプの定義で繰り返される必要があります。したがって、基本タイプにある分解アノテーションを、派生タイプの中にも含めなければなりません。

拡張による派生

拡張によって派生した複合タイプの定義では、基本タイプに追加される要素と属性のみが指定されます。派生タイプの分解のマッピングが基本タイプのマッピングと異なる場合、分解アノテーションを基本タイプに追加して、基本タイプと派生タイプのマッピングを明確に区別しなければなりません。

以下の例は、拡張によって派生したタイプ `outOfPrintBookType` を、その基本タイプ `bookType` とは異なる表にマップできる方法を示しています。`bookType` 基本タイプに db2-xdb:locationPath アノテーションを指定して、基本タイプに適用するマッピングと派生タイプに適用するマッピングを明確に区別する方法に注意してくださ

い。この例では、派生タイプ outOfPrintType の <lastPublished> および <publisher> 要素は、単一のマッピングのみに関係するので、db2-xdb:locationPath アノテーションは必要ありません。

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns:db2-xdb="http://www.ibm.com/xmlns/prod/db2/xdb1">
  <xs:annotation>
    <xs:appinfo>
      <db2-xdb:table>
        <db2-xdb:name>BOOKS</db2-xdb:name>
        <db2-xdb:rowSet>inPrintRowSet</db2-xdb:rowSet>
      </db2-xdb:table>
      <db2-xdb:table>
        <db2-xdb:name>OUTOFPRINT</db2-xdb:name>
        <db2-xdb:rowSet>outOfPrintRowSet</db2-xdb:rowSet>
      </db2-xdb:table>
    </xs:appinfo>
  </xs:annotation>
  <xs:element name="books">
    <xs:complexType>
      <xs:choice>
        <xs:element name="book" type="bookType"
                    minOccurs="0" maxOccurs="unbounded"/>
        <xs:element name="outOfPrintBook" type="outOfPrintBookType"
                    minOccurs="0" maxOccurs="unbounded"/>
      </xs:choice>
    </xs:complexType>
  </xs:element>
  <xs:complexType name="bookType">
    <xs:sequence>
      <xs:element name="authorID" type="xs:integer"/>
      <xs:element name="chapter" type="chapterType" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="title" type="xs:string"
                  db2-xdb:locationPath="/books/book/@title"
                  db2-xdb:rowSet="inPrintRowSet" db2-xdb:column="TITLE">
      <xs:annotation>
        <xs:appinfo>
         <db2-xdb:rowSetMapping db2-xdb:locationPath="/books/outOfPrintBook/@title">
           <db2-xdb:rowSet>outOfPrintRowSet</db2-xdb:rowSet>
           <db2-xdb:column>TITLE</db2-xdb:column>
         </db2-xdb:rowSetMapping>
        </xs:appinfo>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="isbn" type="xs:string"
                  db2-xdb:locationPath="/books/book/@isbn"
                  db2-xdb:rowSet="inPrintRowSet" db2-xdb:column="ISBN">
      <xs:annotation>
        <xs:appinfo>
         <db2-xdb:rowSetMapping db2-xdb:locationPath="/books/outOfPrintBook/@isbn">
           <db2-xdb:rowSet>outOfPrintRowSet</db2-xdb:rowSet>
           <db2-xdb:column>ISBN</db2-xdb:column>
         </db2-xdb:rowSetMapping>
        </xs:appinfo>
      </xs:annotation>
    </xs:attribute>
  </xs:complexType>
  <xs:complexType name="outOfPrintBookType">
    <xs:complexContent>
      <xs:extension base="bookType">
        <xs:sequence>
          <xs:element name="lastPublished" type="xs:date"
                    db2-xdb:rowSet="outOfPrintRowSet" db2-xdb:column="LASTPUBDATE"/>
          <xs:element name="publisher" type="xs:string"
                    db2-xdb:rowSet="outOfPrintRowSet" db2-xdb:column="PUBLISHER"/>
        </xs:sequence>
```

```
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:simpleType name="paragraphType">
    <xs:restriction base="xs:string"/>
  </xs:simpleType>
  <xs:complexType name="chapterType">
    <xs:sequence>
      <xs:element name="paragraph" type="paragraphType" maxOccurs="unbounded"
                  db2-xdb:locationPath="/books/book/chapter/paragraph"
                  db2-xdb:rowSet="inPrintRowSet" db2-xdb:column="CONTENT">
        <xs:annotation>
          <xs:appinfo>
            <db2-xdb:rowSetMapping
              db2-xdb:locationPath="/books/outOfPrintBook/chapter/paragraph">
              <db2-xdb:rowSet>outOfPrintBook</db2-xdb:rowSet>
              <db2-xdb:column>CONTENT</db2-xdb:column>
            </db2-xdb:rowSetMapping>
          </xs:appinfo>
        </xs:annotation>
      </xs:element>
    </xs:sequence>
    <xs:attribute name="number" type="xs:integer"/>
    <xs:attribute name="title" type="xs:string"/>
  </xs:complexType>
</xs:schema>
```

<book> 要素からの値は BOOKS 表に分解され、<outOfPrintBook> 要素からの値は OUTOFPRINT 表に分解されることを、アノテーションは示します。

XML 文書にある次の要素を考えてみましょう。

```
<books>
  <book isbn="1-11-111111-1" title="My First XML Book">
    <authorID>22</authorID>
    <chapter number="1" title="Introduction to XML">
      <paragraph>XML is fun...</paragraph>
    </chapter>
    <chapter number="2" title="XML and Databases">
      <paragraph>XML can be used with...</paragraph>
    </chapter>
  </book>
  <outOfPrintBook isbn="7-77-777777-7" title="Early XML Book">
    <authorID>41</authorID>
    <chapter number="1" title="Introductory XML">
      <paragraph>Early XML...</paragraph>
    </chapter>
    <chapter number="2" title="What is XML">
      <paragraph>XML is an emerging technology...</paragraph>
    </chapter>
    <lastPublished>2000-01-31</lastPublished>
    <publisher>Early Publishers Group</publisher>
  </outOfPrintBook>
</books>
```

以下の表は、上記のアノテーション付きスキーマを使用して、この要素が属する文書を分解した結果です。

表 62. BOOKS

ISBN	TITLE	CONTENT
1-11-111111-1	My First XML Book	XML is fun...
1-11-111111-1	My First XML Book	XML can be used with...

表 63. OUTOFPRINT

ISBN	TITLE	CONTENT	LASTPUBDATE	PUBLISHER
7-77-777777-7	Early XML Book	Early XML...	2000-01-31	Early Publishers Group
7-77-777777-7	Early XML Book	XML is an emerging technology...	2000-01-31	Early Publishers Group

分解に関する XML スキーマの構造化の推奨

アノテーション付き XML スキーマ内の要素の順序を調整することにより、アノテーション付きスキーマ分解によって必要になるシステム・メモリー・リソースを最小限にすることができます。

非常に大きな文書の場合、この推奨に従うと、DB2 データベース・サーバーで使用できるメモリーの量を増やさずに文書を分解できる可能性があります。分解アノテーションが付けられた兄弟要素の場合、アノテーション付きスキーマ中の複合タイプの兄弟要素の前に、単純なタイプの要素を挿入するべきです。同様に、maxOccurs 属性が 1 より大きい兄弟要素の前に、maxOccurs を 1 に設定した兄弟要素を挿入するようにしてください。

アノテーション付きスキーマ分解に必要なメモリー使用量は XML スキーマの構造によって左右されます。その理由は、行を構成するすべての項目が処理されるまで、その行を構成する個々の項目をメモリー内に保持しなければならないからです。これらのスキーマの構造化に関する推奨に従うと、行の項目が、メモリー内に保持しなければならない項目数を最小限にするような仕方で編成されます。

以下の例は、マップされた兄弟要素に関する、推奨されている XML スキーマの構造化を示し、最適でない構造化と対比されています。最適でない方の例で、複合タイプの <complexElem> が単純タイプの <status> の前に挿入されていることに注目してください。<id> および <status> 要素の後に <complexElem> を挿入すると、実行時の分解の効率が高くなります。

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns:db2-xdb="http://www.ibm.com/xmlns/prod/db2/xdb1">
  <-- Recommended structuring with simple types placed before
      the recurring element <wrapper>, which is of complex type -->
  <xs:complexType name="typeA">
    <xs:sequence>
      <xs:element name="id" type="xs:integer"
                  db2-xdb:rowSet="relA" db2-xdb:column="ID" />
      <xs:element name="status" type="xs:string"
                  db2-xdb:rowSet="relA" db2-xdb:column="status" />
      <xs:element name="wrapper" type="typeX" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>

  <-- Less optimal structuring with recurring complex type element
      appearing before the simple type element -->
  <--
  <xs:complexType name="typeA">
    <xs:sequence>
      <xs:element name="id" type="xs:integer"
                  db2-xdb:rowSet="relA" db2-xdb:column="ID" />
      <xs:element name="wrapper" type="typeX" maxOccurs="unbounded"/>
      <xs:element name="status" type="xs:string"
```

```
                db2-xdb:rowSet="relA" db2-xdb:column="status" />
    </xs:sequence>
  </xs:complexType> -->

  <xs:complexType name="typeX">
    <xs:sequence>
      <xs:element name="elem1" type="xs:string"
                  db2-xdb:rowSet="relA" db2-xdb:column="elem1" />
      <xs:element name="elem2" type="xs:long"
                  db2-xdb:rowSet="relA" db2-xdb:column="elem2" />
    </xs:sequence>
  </xs:complexType>

  <xs:element name="A" type="typeA" />

</xs:schema>
```

<id>、<status>、<elem1>、および <elem2> は同じ rowSet にマップされている、つまりこれらが一緒になって行を構成していることに注意してください。行に関連したメモリーは、行が完了すると解放されます。上記の最適でない方の例では、文書中の <status> 要素に達するまで、rowSet relA に関連した行は完了していると見なすことができません。しかし、<wrapper> 要素が <status> 要素の前にあるので、この要素を最初に処理しなければなりません。したがって、<status> 要素に達する (または、<status> が文書中にない場合は <A> の終わりに達する) まで、<wrapper> のすべてのインスタンスがメモリー内のバッファーに入れられなければなりません。

要素のインスタンス数が多いと、この構造の影響は大きくなります。例えば、<wrapper> 要素のインスタンスが 10 000 個あったとすれば、rowSet が完了するまで 10 000 個のインスタンスすべてがメモリー内に保持される必要があったでしょう。しかし、上記の最適な方の例では、<elem2> に達した時点で、rowset relA の行に関連したメモリーを解放できます。

アノテーション付き XML スキーマ分解のマッピング例

アノテーション付き XML スキーマ分解では、XML 文書が表に分解される方法を、マッピングに基づいて決定します。マッピングは XML スキーマ文書に追加されたアノテーションとして表現されます。これらのマッピングは、XML 文書を表に分解する方法を示します。以下の例では、一般的なマッピングのシナリオを示します。

一般的なマッピングのシナリオ:

アノテーション付きの XML スキーマ分解における rowSet

db2-xdb:rowSet は、値が分解されるターゲット表を識別します。このアノテーションは表の名前か rowSet 名に設定することができます。

rowSet は db2-xdb:rowSet アノテーションによって指定されます。このアノテーションは、要素または属性宣言の属性、または <db2-xdb:rowSetMapping> アノテーションの子として XML スキーマ文書に追加されます。

XML スキーマを形成するすべてのスキーマ文書にまたがるマッピングの集合は、要素または属性のインスタンスについて同じ db2-xdb:rowSet 値を持ち、1 つの行を定義します。

例えば、次の XML 文書を考えてみます。

```
<publications>
  <textbook title="Programming with XML">
    <isbn>0-11-011111-0</isbn>
    <author>Mary Brown</author>
    <author>Alex Page</author>
    <publicationDate>2002</publicationDate>
    <university>University of London</university>
  </textbook>
  <childrensbook title="Children's Fables">
    <isbn>5-55-555555-5</isbn>
    <author>Bob Carter</author>
    <author>Melaine Snowe</author>
    <publicationDate>1999</publicationDate>
  </childrensbook>
</publications>
```

この文書を分解して、それぞれの本の isbn と表題が (教科書か児童書かにかかわらず) 同じ表 ALLPUBLICATIONS に挿入されるようにするには、複数の rowSet を定義する必要があります。 1 つの rowSet は教科書に関連した値をグループ化し、別の rowSet は児童書に関連した値をグループ化します。

この事例において、rowSet は、意味が関連している値だけがグループ化されて 1 つの行を形成することを保証します。つまり、rowSet の使用により、教科書の isbn 値が表題とともにグループ化され、児童書の isbn 値が表題とともにグループ化されます。これにより、教科書の isbn 値とともに児童書の表題が行に含まれるようなことのないことが保証されます。

rowSet がなければ、どの値をグループ化すれば意味構造が正しい行が形成されるかを判別することは不可能です。

次に、XML スキーマ文書における rowSet の適用について説明します。 textbk_rowSet および childrens_rowSet という 2 つの rowSet が、 <textbook> および <childrensbook> 要素の isbn 要素宣言でそれぞれ指定されています。次いで、これらの rowSet は <db2-xdb:table> アノテーションにより、ALLPUBLICATIONS 表に関連付けられます。

rowSet 注釈を表 ID でなく rowSet ID として使用することにより、 XML スキーマで参照される表の名前を容易に変更できることに注意してください。これは、db2-xdb:rowSet の値が表の名前でなく ID を表す場合、実際に表の名前を指定するのに <db2-xdb:table><db2-xdb:name></db2-xdb:name></db2-xdb:table> アノテーションを使用しなければならないからです。この方式では、必要に応じて 1 箇所だけの表の名前の更新で済みます。

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns:db2-xdb="http://www.ibm.com/xmlns/prod/db2/xdb1"
           elementFormDefault="qualified" attributeFormDefault="unqualified">
  <xs:annotation>
    <xs:appinfo>
      <db2-xdb:defaultSQLSchema>admin</db2-xdb:defaultSQLSchema>
      <db2-xdb:table>
        <db2-xdb:name>ALLPUBLICATIONS</db2-xdb:name>
        <db2-xdb:rowSet>textbk_rowSet</db2-xdb:rowSet>
        <db2-xdb:rowSet>childrens_rowSet</db2-xdb:rowSet>
      </db2-xdb:table>
    </xs:appinfo>
  </xs:annotation>
    <xs:element name="publications">
```

```
      <xs:complexType>
        <xs:sequence>
          <xs:element name="textbook" maxOccurs="unbounded">
            <xs:complexType>
              <xs:sequence>
                <xs:element name="isbn" type="xs:string"
                       db2-xdb:rowSet="textbk_rowSet" db2-xdb:column="PUBS_ISBN"/>
                <xs:element name="author" type="xs:string" maxOccurs="unbounded"/>
                <xs:element name="publicationDate" type="xs:gYear"/>
                <xs:element name="university" type="xs:string"
                            maxOccurs="unbounded"/>
              </xs:sequence>
              <xs:attribute name="title" type="xs:string" use="required"
                      db2-xdb:rowSet="textbk_rowSet" db2-xdb:column="PUBS_TITLE"/>
            </xs:complexType>
          </xs:element>
          <xs:element name="childrensbook" maxOccurs="unbounded">
            <xs:complexType>
              <xs:sequence>
                <xs:element name="isbn" type="xs:string"
                    db2-xdb:rowSet="childrens_rowSet" db2-xdb:column="PUBS_ISBN"/>
                <xs:element name="author" type="xs:string" maxOccurs="unbounded"/>
                <xs:element name="publicationDate" type="xs:gYear"/>
              </xs:sequence>
              <xs:attribute name="title" type="xs:string" use="required"
                   db2-xdb:rowSet="childrens_rowSet" db2-xdb:column="PUBS_TITLE"/>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
</xs:schema>
```

このアノテーション付き XML スキーマを使用して分解した結果として得られる表は次のとおりです。

表 64. ALLPUBLICATIONS

ISBN	PUBS_TITLE
0-11-011111-0	Programming with XML
5-55-555555-5	Children's Fables

上記の例は rowSet を使用した分解の単純な事例を示していますが、 rowSet をより複雑なマッピングで使用すれば、XML スキーマのさまざまな部分にある複数の項目をグループ化して、同じ表と列の対で行を形成することができます。

条件付きトランスフォーメーション

rowSet を使用すれば、分解対象の値に対して、値そのものに応じたさまざまなトランスフォーメーションを適用することができます。

例えば、以下の「temperature」という名前の要素の 2 つのインスタンスを考えてみてください。

```
<temperature unit="Celsius">49</temperature>
<temperature unit="Fahrenheit">49</temperature>
```

これらの要素の値を同じ表に挿入し、その表に一貫性のある値 (例えば Celsius の値すべて) が含まれるようにするには、属性 unit="Fahrenheit" を持つ値を挿入前に Celsius に変換する必要があります。これを行うには、属性 unit="Celsius" を持つす

べての要素を 1 つの rowSet にマッピングし、属性 unit="Fahrenheit" を持つすべての要素を別の rowSet にマッピングします。次いで、Fahrenheit 値の rowSet に、挿入前に変換公式を適用することができます。

"unit" の属性宣言に関するマッピングに db2-xdb:column の指定が含まれていないことに注意してください。したがって、項目の値は条件評価のみに使用され、db2-xdb:rowSet 仕様で指定されている表への保管には使用されません。

以下の XML スキーマ文書を使用すると、Celsius 値と変換済みの Fahrenheit 値を同じ表に挿入できます。

```
....
<!-- Global annotation -->
<db2-xdb:table>
  <db2-xdb:name>TEMPERATURE_DATA</db2-xdb:name>
  <db2-xdb:rowSet>temp_celsius</db2-xdb:rowSet>
  <db2-xdb:rowSet>temp_fahrenheit</db2-xdb:rowSet>
</db2-xdb:table>
...
<xs:element name="temperature">
  <xs:annotation>
    <xs:appinfo>
      <db2-xdb:rowSetMapping>
        <db2-xdb:rowSet>temp_celsius</db2-xdb:rowSet>
        <db2-xdb:column>col1</db2-xdb:column>
      </db2-xdb:rowSetMapping>
      <db2-xdb:rowSetMapping>
        <db2-xdb:rowSet>temp_fahrenheit</db2-xdb:rowSet>
        <db2-xdb:column>col1</db2-xdb:column>
        <db2-xdb:expression>
   myudf_convertTocelsius($DECOMP_CONTENT)
 </db2-xdb:expression>
      </db2-xdb:rowSetMapping>
    </xs:appinfo>
  </xs:annotation>
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:int">
        <xs:attribute name="unit" type="xs:string">
          <xs:annotation>
            <xs:appinfo>
              <db2-xdb:rowSetMapping>
                <db2-xdb:rowSet>temp_celsius</db2-xdb:rowSet>
                <db2-xdb:condition>
    $DECOMP_CONTENT = 'Celsius'
  </db2-xdb:condition>
              </db2-xdb:rowSetMapping>
              <db2-xdb:rowSetMapping>
                <db2-xdb:rowSet>temp_fahrenheit</db2-xdb:rowSet>
                <db2-xdb:condition>
    $DECOMP_CONTENT = 'fahrenheit'
  </db2-xdb:condition>
              </db2-xdb:rowSetMapping>
            </xs:appinfo>
          </xs:annotation>
        </xs:attribute>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

分解アノテーション例: XML 列へのマッピング

アノテーションが付けられた XML スキーマ分解では、XML フラグメントを、XML データ・タイプを使用して定義された列にマップできます。

次の XML 文書を考えてみます。

```
<publications>
  <textbook title="Programming with XML">
    <isbn>0-11-011111-0</isbn>
    <author>Mary Brown</author>
    <author>Alex Page</author>
    <publicationDate>2002</publicationDate>
    <university>University of London</university>
  </textbook>
</publications>
```

次のように、<textbook> の XML 要素およびブック・タイトルを保管する場合、対応する XML スキーマ文書の <textbook> 要素およびタイトル属性の宣言にアノテーションを追加します。アノテーションは、DETAILS および TITLE 列を指定する必要があります。ここで DETAILS 列は、TEXTBOOKS 表と同様、XML タイプで定義されています。

表 65. TEXTBOOKS

TITLE	DETAILS
Programming with XML	<textbook title="Programming with XML"> <isbn>0-11-011111-0</isbn> <author>Mary Brown</author> <author>Alex Page</author> <publicationDate>2002</publicationDate> <university>University of London</university> </textbook>

アノテーションによっては、属性または要素としてスキーマ文書に指定できます。一部のアノテーションはどちらか一方に指定できます。特定のアノテーションをどのように指定するかを決定する場合、それぞれのアノテーションに関する文書を参照してください。

<xs:element> または <xs:attribute> の属性として db2-xdb:rowSet および db2-xdb:column を使用するか、あるいは <db2-xdb:rowSetMapping> の子要素である <db2-xdb:rowSet> および <db2-xdb:column> を使用することで、ターゲット表および列を指定します。これらのマッピングを要素または属性として指定することは同等です。

次の XML スキーマ文書の一部は、アノテーションを属性として指定して、2 つのマッピングを <textbook> 要素およびタイトル属性に追加する方法を示しています。

```
<xs:element name="publications">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="textbook" maxOccurs="unbounded"
                  db2-xdb:rowSet="TEXTBOOKS" db2-xdb:column="DETAILS">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="isbn" type="xs:string"/>
            <xs:element name="author" type="xs:string" maxOccurs="unbounded"/>
            <xs:element name="publicationDate" type="xs:gYear"/>
            <xs:element name="university" type="xs:string" maxOccurs="unbounded"/>
```

```
          </xs:sequence>
          <xs:attribute name="title" type="xs:string" use="required"
                        db2-xdb:rowSet="TEXTBOOKS" db2-xdb:column="TITLE"/>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

db2-xdb:rowSet アノテーションはターゲット表の名前を指定し、db2-xdb:column アノテーションはターゲット列の名前を指定します。 <textbook> 要素は複合タイプであり、複合内容を含み、db2-xdb:contentHandling アノテーションが指定されていないので、デフォルトでは、要素内のすべてのマークアップ (その開始および終了タグを含む) が、db2-xdb:contentHandling の serializeSubtree 設定に応じて XML 列に挿入されます。 XML 文書内の空白文字は保存されます。詳しくは、db2-xdb:contentHandling の資料を参照してください。

分解アノテーション例: 単一行を生成する単一表に値をマップする

値を XML 文書から単一表および列のペアにマップすることは、アノテーション付き XML スキーマ分解のマッピングの簡易的な形式です。この例では、rowSet の値の 1 対 1 の関係の簡易的なケースを示します。

このマッピングの結果は、同一の rowSet にマップされた項目間のリレーションシップに依存します。単一の rowSet に一緒にマップされる値に 1 対 1 の関係がある場合、その要素の maxOccurs 属性の値、または含まれているモデル・グループ宣言の値によって決定されるように、単一行が XML 文書中のマップされた項目のインスタンスごとに一行が形成されます。単一の rowSet の値に 1 対多の関係があり、ある値が別の項目の複数のインスタンスの文書で一度のみ現れる場合は、maxOccurs 属性の値によって示されるように、XML 文書の分解時に複数の行が形成されます。

次の XML 文書を考えてみます。

```
<publications>
  <textbook title="Programming with XML">
    <isbn>0-11-011111-0</isbn>
    <author>Mary Brown</author>
    <author>Alex Page</author>
    <publicationDate>2002</publicationDate>
    <university>University of London</university>
  </textbook>
</publications>
```

次のように、<isbn> 要素および <publicationDate> 要素の値を、タイトル属性と同様に TEXTBOOKS 表に分解する場合、対応する XML スキーマ文書の要素および属性の宣言にアノテーションを追加する必要があります。アノテーションは、各項目がマップされる表および列の名前を指定します。

表 66. TEXTBOOKS

ISBN	TITLE	DATE
0-11-011111-0	Programming with XML	2002

アノテーションによっては、属性または要素としてスキーマ文書に指定できます。一部のアノテーションはどちらか一方に指定できます。特定のアノテーションをどのように指定するかを決定する場合、それぞれのアノテーションに関する文書を参照してください。

値を単一の表および列のペアにマップする場合は、マップされる値に表および列を指定する必要があります。これは、<xs:element> または <xs:attribute> の属性として db2-xdb:rowSet および db2-xdb:column を使用するか、あるいは <db2-xdb:rowSetMapping> の子要素である <db2-xdb:rowSet> および <db2-xdb:column> を使用することで実行できます。これらのマッピングを要素または属性として指定することは同等です。

次の例では、アノテーションを属性に指定して、<textbook> 要素から TEXTBOOKS 表に要素および属性をマップする方法を示しています。

```
<xs:element name="publications">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="textbook" maxOccurs="unbounded">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="isbn" type="xs:string"
                        db2-xdb:rowSet="TEXTBOOKS" db2-xdb:column="ISBN"/>
            <xs:element name="author" type="xs:string" maxOccurs="unbounded"/>
            <xs:element name="publicationDate" type="xs:gYear"
                        db2-xdb:rowSet="TEXTBOOKS" db2-xdb:column="DATE"/>
            <xs:element name="university" type="xs:string" maxOccurs="unbounded"/>
          </xs:sequence>
         <xs:attribute name="title" type="xs:string" use="required"
                       db2-xdb:rowSet="TEXTBOOKS" db2-xdb:column="TITLE"/>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

maxOccurs XML スキーマ属性にはデフォルト値 1 があり、したがって、TEXTBOOKS rowSet にマップされる各項目には互いに 1 対 1 の関係があります。この 1 対 1 の関係のため、単一行が <textbook> 要素の各インスタンスに形成されます。

分解アノテーション例: 複数行を生成する単一表に値をマップする

値を XML 文書から単一表および列のペアにマップすることは、アノテーション付き XML スキーマ分解のマッピングの簡易的な形式です。この例では、rowSet の値の 1 対多の関係のより複雑なケースを示しています。

このマッピングの結果は、同一の rowSet にマップされた項目間のリレーションシップに依存します。単一の rowSet に一緒にマップされる値に 1 対 1 の関係がある場合、その要素の maxOccurs 属性の値、または含まれているモデル・グループ宣言の値によって決定されるように、単一行が XML 文書中のマップされた項目のインスタンスごとに一行が形成されます。単一の rowSet の値に 1 対多の関係があり、ある値が別の項目の複数のインスタンスの文書で一度のみ現れる場合は、maxOccurs 属性の値によって示されるように、XML 文書の分解時に複数の行が形成されます。

次の XML 文書を考えてみます。

```
<textbook title="Programming with XML">
  <isbn>0-11-011111-0</isbn>
  <author>Mary Brown</author>
  <author>Alex Page</author>
  <publicationDate>2002</publicationDate>
  <university>University of London</university>
</textbook>
```

次のように、テキスト・ブックの ISBN および著者を保存する場合、対応する XML スキーマ文書の <isbn> 要素および <author> 要素の宣言にアノテーションを追加します。アノテーションは、TEXTBOOK_AUTH 表に加えて、ISBN 列および AUTHNAME 列も指定する必要があります。

表 67. TEXTBOOKS_AUTH

ISBN	AUTHNAME
0-11-011111-0	Mary Brown
0-11-011111-0	Alex Page

アノテーションによっては、属性または要素としてスキーマ文書に指定できます。一部のアノテーションはどちらか一方に指定できます。特定のアノテーションをどのように指定するかを決定する場合、それぞれのアノテーションに関する文書を参照してください。

値を単一の表および列のペアにマップする場合は、マップされる値に表および列を指定する必要があります。これは、<xs:element> または <xs:attribute> の属性として db2-xdb:rowSet および db2-xdb:column を使用するか、あるいは <db2-xdb:rowSetMapping> の子要素である <db2-xdb:rowSet> および <db2-xdb:column> を使用することで実行できます。

これらのマッピングを要素または属性として指定することは同等です。マッピングは、次で示される XML スキーマ文書に要素として指定されます。

```
<xs:element name="textbook" maxOccurs="unbounded">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="isbn" type="xs:string">
        <xs:annotation>
          <xs:appinfo>
            <db2-xdb:rowSetMapping>
              <db2-xdb:rowSet>TEXTBOOKS_AUTH</db2-xdb:rowSet>
              <db2-xdb:column>ISBN</db2-xdb:column>
            </db2-xdb:rowSetMapping>
          </xs:appinfo>
        </xs:annotation>
      </xs:element>
      <xs:element name="author" type="xs:string" maxOccurs="unbounded">
        <xs:annotation>
          <xs:appinfo>
            <db2-xdb:rowSetMapping>
              <db2-xdb:rowSet>TEXTBOOKS_AUTH</db2-xdb:rowSet>
              <db2-xdb:column>AUTHNAME</db2-xdb:column>
            </db2-xdb:rowSetMapping>
          </xs:appinfo>
        </xs:annotation>
      </xs:element>
      <xs:element name="publicationDate" type="xs:gYear"/>
      <xs:element name="university" type="xs:string" maxOccurs="unbounded"/>
```

```
    </xs:sequence>
    <xs:attribute name="title" type="xs:string" use="required"/>
  </xs:complexType>
</xs:element>
```

どのように <isbn> 要素が ISBN 列に一度のみマップされ、さらに表の 2 つの行に現れるかに注目してください。ISBN の値ごとに複数の著者が存在するため、分解処理時に自動的に処理されます。<isbn> の値が著者ごとに各行で複製されます。

<author> の maxOccurs 属性が 1 を超え、1 対多の関係が <isbn> 要素および <author> 要素で検出されるため、この動作が発生します。

1 対多の関係には 3 つ以上の項目および項目のセットを含めることができるという点に留意してください。また、1 対多の関係は深くネストでき、すでに 1 対多の関係に関連している項目は別の 1 対多の関係にも関連できます。

分解アノテーション例: 複数表に値をマップする

XML 文書の単一値を複数表にマップできます。この例では、XML スキーマ文書にアノテーションを付け、単一値を 2 つの表にマップする方法を示しています。

次の XML 文書を考えてみます。

```
<textbook title="Programming with XML">
  <isbn>0-11-011111-0</isbn>
  <author>Mary Brown</author>
  <author>Alex Page</author>
  <publicationDate>2002</publicationDate>
  <university>University of London</university>
</textbook>
```

次の 2 つの表にテキスト・ブックの ISBN をマップするには、<isbn> 要素に 2 つのマッピングを作成する必要があります。これは、XML スキーマ文書で複数の <db2-xdb:rowSetMapping> 要素を <isbn> 要素宣言に追加することによって実行できます。

表 68. TEXTBOOKS

ISBN	TITLE
0-11-011111-0	Programming with XML

表 69. SCHOOLPUBS

ISBN	SCHOOL
0-11-011111-0	University of London

次の XML スキーマ文書の一部は、2 つのマッピングを <isbn> 要素宣言に追加して 2 つの表にマッピングを指定する方法を示しています。また、タイトル属性および <university> 要素の値も、マッピングに含まれています。

```
<xs:element name="textbook" maxOccurs="unbounded">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="isbn" type="xs:string">
        <xs:annotation>
          <xs:appinfo>
            <db2-xdb:rowSetMapping>
              <db2-xdb:rowSet>TEXTBOOKS</db2-xdb:rowSet>
```

```
            <db2-xdb:column>ISBN</db2-xdb:column>
          </db2-xdb:rowSetMapping>
          <db2-xdb:rowSetMapping>
            <db2-xdb:rowSet>SCHOOLPUBS</db2-xdb:rowSet>
            <db2-xdb:column>ISBN</db2-xdb:column>
          </db2-xdb:rowSetMapping>
        </xs:appinfo>
      </xs:annotation>
    </xs:element>
    <xs:element name="author" type="xs:string" maxOccurs="unbounded"/>
    <xs:element name="publicationDate" type="xs:gYear"/>
    <xs:element name="university" type="xs:string" maxOccurs="unbounded">
      <xs:annotation>
        <xs:appinfo>
          <db2-xdb:rowSetMapping>
            <db2-xdb:rowSet>SCHOOLPUBS</db2-xdb:rowSet>
            <db2-xdb:column>SCHOOL</db2-xdb:column>
          </db2-xdb:rowSetMapping>
        </xs:appinfo>
      </xs:annotation>
    </xs:element>
  </xs:sequence>
  <xs:attribute name="title" type="xs:string" use="required">
    <xs:annotation>
      <xs:appinfo>
        <db2-xdb:rowSetMapping>
          <db2-xdb:rowSet>TEXTBOOKS</db2-xdb:rowSet>
          <db2-xdb:column>TITLE</db2-xdb:column>
        </db2-xdb:rowSetMapping>
      </xs:appinfo>
    </xs:annotation>
  </xs:attribute>
 </xs:complexType>
</xs:element>
```

複数回現れる複合型

複合型が XML スキーマの複数の場所で参照される場合、スキーマ内の場所によっては db2-xdb:locationPath アノテーションを使用して、異なる表および列にマップできます。

この場合、複合型の要素または属性の宣言は、複数の <db2-xdb:rowSetMapping> アノテーションを使用してアノテーションを付けられる必要があり (各マッピングに 1 つのアノテーション)、各マッピングは db2-xdb:locationPath 属性によって区別されます。

分解アノテーション例: 単一表にマップされる複数値をグループ化する

アノテーション付き XML スキーマ分解では、論理的に関連する値のリレーションシップを保持しながら、関連しない要素の複数の値を同一の表にマップできます。以下の例に示されているように、これは複数の rowSet を宣言することで可能で、関連する項目をグループ化して行を作成する場合に使用されます。

例えば、次の XML 文書を考えてみます。

```
<publications>
  <textbook title="Programming with XML">
    <isbn>0-11-011111-0</isbn>
    <author>Mary Brown</author>
    <author>Alex Page</author>
```

```
    <publicationDate>2002</publicationDate>
    <university>University of London</university>
  </textbook>
  <childrensbook title="Children's Fables">
    <isbn>5-55-555555-5</isbn>
    <author>Bob Carter</author>
    <author>Melaine Snowe</author>
    <publicationDate>1999</publicationDate>
  </childrensbook>
</publications>
```

分解後に次の表を生成するには、テキスト・ブックに関連する値が、児童書に関連する値と同一の行にグループ化されないようにする必要があります。複数の rowSet を使用して関連した値をグループ化し、論理的に意味のある行を生成します。

表 70. ALLPUBLICATIONS

PUBS_ISBN	PUBS_TITLE
0-11-011111-0	Programming with XML
5-55-555555-5	Children's Fables

単一のマッピング・シナリオでは、単一表および列のペアに単一の値をマップする場合、値をマップする表および列のみを指定すればよい場合もあります。

この例ではより複雑なケースを示しています。ただし、複数値は同一の表にマップされ、論理的にグループ化される必要があります。rowSet を使用しないで、各 ISBN およびタイトルを PUBS_ISBN 列および PUBS_TITLE 列にマップするのみの場合は、分解処理では ISBN 値がどのタイトル値に属するかを決定することはできません。rowSet を使用することで、論理的に関連した値をグループ化し、意味のある行を作成できます。

次の XML スキーマ文書では、2 つの rowSet が定義され、<textbook> 要素の値を <childrensbook> 要素の値と区別する方法を示しています。

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns:db2-xdb="http://www.ibm.com/xmlns/prod/db2/xdb1"
           elementFormDefault="qualified" attributeFormDefault="unqualified">
  <xs:annotation>
    <xs:appinfo>
      <db2-xdb:table>
        <db2-xdb:name>ALLPUBLICATIONS</db2-xdb:name>
        <db2-xdb:rowSet>textbk_rowSet</db2-xdb:rowSet>
        <db2-xdb:rowSet>childrens_rowSet</db2-xdb:rowSet>
      </db2-xdb:table>
    </xs:appinfo>
  </xs:annotation>
  <xs:element name="publications">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="textbook" maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="isbn" type="xs:string"
                          db2-xdb:rowSet="textbk_rowSet" db2-xdb:column="PUBS_ISBN"/>
              <xs:element name="author" type="xs:string" maxOccurs="unbounded"/>
              <xs:element name="publicationDate" type="xs:gYear"/>
              <xs:element name="university" type="xs:string" maxOccurs="unbounded"/>
            </xs:sequence>
            <xs:attribute name="title" type="xs:string" use="required"
                          db2-xdb:rowSet="textbk_rowSet" db2-xdb:column="PUBS_TITLE"/>
```

```
          </xs:complexType>
        </xs:element>
        <xs:element name="childrensbook" maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="isbn" type="xs:string"
                     db2-xdb:rowSet="childrens_rowSet" db2-xdb:column="PUBS_ISBN"/>
              <xs:element name="author" type="xs:string" maxOccurs="unbounded"/>
              <xs:element name="publicationDate" type="xs:gYear"/>
            </xs:sequence>
            <xs:attribute name="title" type="xs:string" use="required"
                    db2-xdb:rowSet="childrens_rowSet" db2-xdb:column="PUBS_TITLE"/>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

要素および属性の各宣言の db2-xdb:rowSet マッピングが、表の名前ではなく rowSet の名前を指定する方法に留意してください。rowSet は <db2-xdb:table> アノテーションの ALLPUBLICATIONS 表に関連付けられ、<xs:schema> の子として指定される必要があります。

同一の表にマップする複数の rowSet を指定することにより、論理的に関連した値は確実にその表に行を生成できます。

分解アノテーション例: コンテキストの異なる複数の値を単一表にマップする

アノテーション付き XML スキーマ分解では、複数の値を同一の表および列にマップし、文書のさまざまな部分に由来する値を単一の列に入れることができます。以下の例に示されているように、これは、複数の rowSet を宣言することによって可能です。

例えば、次の XML 文書を考えてみます。

```
<publications>
  <textbook title="Principles of Mathematics">
    <isbn>1-11-111111-1</isbn>
    <author>Alice Braun</author>
    <publisher>Math Pubs</publisher>
    <publicationDate>2002</publicationDate>
    <university>University of London</university>
  </textbook>
</publications>
```

ある特定の書籍の連絡先を含む同一の表に、著者と出版社の両方をマップできます。

表 71. BOOKCONTACTS

ISBN	CONTACT
1-11-111111-1	Alice Braun
1-11-111111-1	Math Pubs

結果として生じる表の CONTACT 列の値は、XML 文書のさまざまな部分から取得されます。ある行には著者名が (<author> 要素から) 入り、別の行には出版社名が (<publisher> 要素から) 入ることがあります。

次の XML スキーマ文書では、複数の rowSet を使用してこの表を生成する方法を示しています。

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns:db2-xdb="http://www.ibm.com/xmlns/prod/db2/xdb1"
           elementFormDefault="qualified" attributeFormDefault="unqualified">
  <xs:annotation>
    <xs:appinfo>
      <db2-xdb:table>
        <db2-xdb:name>BOOKCONTACTS</db2-xdb:name>
        <db2-xdb:rowSet>author_rowSet</db2-xdb:rowSet>
        <db2-xdb:rowSet>publisher_rowSet</db2-xdb:rowSet>
      </db2-xdb:table>
    </xs:appinfo>
  </xs:annotation>
  <xs:element name="publications">
     <xs:complexType>
       <xs:sequence>
         <xs:element name="textbook" maxOccurs="unbounded">
           <xs:complexType>
             <xs:sequence>
               <xs:element name="isbn" type="xs:string">
                 <xs:annotation>
                   <xs:appinfo>
                     <db2-xdb:rowSetMapping>
                       <db2-xdb:rowSet>author_rowSet</db2-xdb:rowSet>
                       <db2-xdb:column>ISBN</db2-xdb:column>
                     </db2-xdb:rowSetMapping>
                     <db2-xdb:rowSetMapping>
                       <db2-xdb:rowSet>publisher_rowSet</db2-xdb:rowSet>
                       <db2-xdb:column>ISBN</db2-xdb:column>
                     </db2-xdb:rowSetMapping>
                   </xs:appinfo>
                 </xs:annotation>
               </xs:element>
               <xs:element name="author" type="xs:string" maxOccurs="unbounded">
                 <xs:annotation>
                   <xs:appinfo>
                     <db2-xdb:rowSetMapping>
                       <db2-xdb:rowSet>author_rowSet</db2-xdb:rowSet>
                       <db2-xdb:column>CONTACT</db2-xdb:column>
                     </db2-xdb:rowSetMapping>
                   </xs:appinfo>
                 </xs:annotation>
               </xs:element>
               <xs:element name="publisher" type="xs:string">
                 <xs:annotation>
                   <xs:appinfo>
                     <db2-xdb:rowSetMapping>
                       <db2-xdb:rowSet>publisher_rowSet</db2-xdb:rowSet>
                       <db2-xdb:column>CONTACT</db2-xdb:column>
                     </db2-xdb:rowSetMapping>
                   </xs:appinfo>
                 </xs:annotation>
               </xs:element>
               <xs:element name="publicationDate" type="xs:gYear"/>
               <xs:element name="university" type="xs:string"
                           maxOccurs="unbounded"/>
             </xs:sequence>
             <xs:attribute name="title" type="xs:string" use="required"/>
           </xs:complexType>
         </xs:element>
```

```
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

各要素宣言の db2-xdb:rowSet マッピングが、表の名前ではなく rowSet の名前を指定する方法に留意してください。rowSet は <db2-xdb:table> アノテーションの BOOKCONTACTS 表に関連付けられ、これは <xs:schema> の子として指定される必要があります。

アノテーション付きスキーマ分解に関する XML スキーマと SQL タイプとの互換性

アノテーション付き XML スキーマ分解により、XML 値を表の列に保管できるようになります。 XML 値は、互換性のある SQL 列のみに分解できます。以下の表では、XML スキーマ・タイプと SQL 列タイプの互換性がリストされています。

表 72. 互換性のある XML スキーマと SQL データ・タイプ

XML スキーマ・タイプ	SQL SMALLINT	SQL INTEGER	SQL BIGINT	SQL REAL	SQL DECFLOAT(16)	SQL DOUBLE	SQL DECFLOAT(34)	SQL VARCHAR	SQL CLOB	SQL TIMESTAMP	SQL DATE	SQL TIME	SQL CHAR	SQL GRAPHIC	SQL VARGRAPHIC	SQL DBCLOB	SQL BLOB	SQL XML	SQL VARCHAR FOR BIT DATA	SQL CHAR FOR BIT DATA	SQL BINARY	SQL VARBINARY
string、normalizedString、token	1	1	1	1	1	1	1	1	2	3	4	6	5	5	5	6a	5a	5a	5a	7a	7	7	7
base64Binary、hexBinary	-	-	-	-	-	-	-	-	-	-	-	8a	8	8	8	-	-	-	-	8c	8b	8b	8b
byte、unsigned byte	0a	0a	0a	0a	0a	0a	0a	0a	-	-	-	9a*	9*	9*	9*	-	-	-	-	-	-	-	-
integer、positiveInteger、negativeInteger、nonNegativeInteger、nonPositiveInteger	10	10	10	11	11	11	10	10	-	-	-	9a*	9*	9*	9*	-	-	-	-	-	-	-	-
int	10	0a	0a	11	11	0a	0a	0a	-	-	-	9a*	9*	9*	9*	-	-	-	-	-	-	-	-
unsignedInt	10	10	0a	11	11	0a	0a	0a	-	-	-	9a*	9*	9*	9*	-	-	-	-	-	-	-	-
long	10	10	0a	11	11	11	10	0a	-	-	-	9a*	9*	9*	9*	-	-	-	-	-	-	-	-
unsignedLong	10	10	10	11	11	11	10	0a	-	-	-	9a*	9*	9*	9*	-	-	-	-	-	-	-	-
short	0a	0a	0a	0a	0a	0a	0a	0a	-	-	-	9a*	9*	9*	9*	-	-	-	-	-	-	-	-
unsignedShort	10	0a	0a	0a	0a	0a	0a	0a	-	-	-	9a*	9*	9*	9*	-	-	-	-	-	-	-	-
decimal	21	21	21	11	11	11	11	11	-	-	-	9a*	9*	9*	9*	-	-	-	-	-	-	-	-
float	22	22	22	17	16	17	0a	0a	-	-	-	9a*	9*	9*	9*	-	-	-	-	-	-	-	-
double	22	22	22	16	16	17	11	11	-	-	-	9a*	9*	9*	9*	-	-	-	-	-	-	-	-
boolean	12	12	12	12	12	12	12	12	-	-	-	9a*	9*	9*	9*	-	-	-	-	-	-	-	-
time	-	-	-	-	-	-	-	-	-	14	-	13a*	13*	13*	13*	-	-	-	-	-	-	-	-
dateTime	-	-	-	-	-	-	-	-	15	15	19	13a*	13*	13*	13*	-	-	-	-	-	-	-	-

表 72. 互換性のある XML スキーマと SQL データ・タイプ (続き)

XML スキーマ・タイプ	SQLSMALLINT	SQLINTEGER	SQLBIGINT	SQLREAL	SQLDOUBLE	SQLDECIMAL	SQLDECFLOAT(16)	SQLDECFLOAT(34)	SQLCHAR	SQLWCHAR	SQLTIME	SQLDATE	SQLTIMESTAMP(0)	SQLTIMESTAMP(1)	SQLTIMESTAMP(2)	SQLTIMESTAMP(3)	SQLTIMESTAMP(4)	SQLTIMESTAMP(5)	SQLTIMESTAMP(6)	SQLVARCHAR	SQLWVARCHAR	SQLCLOB	SQLWCLOB	SQLBLOB	SQLLONG	SQLVARCHAR FOR BIT DATA	SQLLONG VARCHAR FOR BIT DATA
duration、gMonth、gYear、gDay、gMonthDay、gYearMonth	-	-	-	-	-	-	-	-	-	-	-	13a	13	13	13	-	-	-	-	-	-	-	-	-	-	-	-
date	-	-	-	-	-	-	-	-	20	-	-	13a*	13*	13*	13*	-	-	-	-	-	-	-	-	-	-	-	-
Name、NCName、NOTATION、ID、IDREF、QName、NMTOKEN、ENTITY	-	-	-	-	-	-	-	-	-	-	-	6	5	5	5	6a	5a	5a	5a	7a	7	7	7	7	7	7	7
ENTITIES、NMTOKENS、IDREFS、list types	-	-	-	-	-	-	-	-	-	-	-	6b	5b	5b	5b	6c	5c	5c	5c	7c	7b	7b	7b	7b	7b	7b	7b
anyURI	-	-	-	-	-	-	-	-	-	-	-	18a	18	18	18	-	-	-	-	7a	7	7	7	7	7	7	7
language	-	-	-	-	-	-	-	-	-	-	-	6	5	5	5	-	-	-	-	7a	7	7	7	7	7	7	7
anySimpleType、union types	-	-	-	-	-	-	-	-	-	-	-	6d	5d	5d	5d	6e	5e	5e	5e	7e	7d	7d	7d	7d	7d	7d	7d
anyType	-	-	-	-	-	-	-	-	-	-	-	6d	5d	5d	5d	6e	5e	5e	5e	7e	7d	7d	7d	7d	7d	7d	7d

凡例

† FOR BIT DATA

***** db2-xdb:normalization アノテーションは、データベースに挿入されるストリングの形式を決定するために使用されます。

- データ・タイプにはアノテーション付き XML スキーマ分解のための互換性がありません。

0 データ・タイプには互換性があります。

0a 互換性があります。-0 が XML タイプの値スペースにある場合、-0 はデータベース内では 0 として保管されます。

1 ストリングがターゲットの SQL タイプにとって受け入れ可能な字句形式であり、その SQL タイプの範囲内で数値に変換可能である場合は、互換性があります。有効数字の消失が起きる可能性があります。

2 ストリングの日付が有効な形式の場合 (*yyyy-mm-dd*、*mm/dd/yyyy*、または *dd.mm.yyyy*)、互換性があります。

3 ストリングの時刻が有効な形式の場合 (*hh.mm.ss*、*hh:mm AM* または *PM*、または *hh:mm:ss*)、互換性があります。

4 ストリングのタイム・スタンプが有効な形式の場合 (*yyyy-mm-dd-hh.mm.ss.nnnnnn* または *yyyy-mm-dd hh.mm.ss.nnnnnn*)、互換性があります。

5 XML 入力ストリングの長さ (バイト数) がターゲット列の長さ (バイト数) 以下の場合、互換性があります。入力ストリングがターゲット列より長い場合は、db2-xdb:truncate の列マッピングが「true」または「1」に設定される場合にのみ互換性があります。ストリングの長さは正規化の後に計算されます。この正規化では、XML スキーマ・タイプの空白ファセットに応じて入力ストリングが正規化されます。

5a 5 で記述された条件に応じて、互換性があります。さらに、入力ストリングは 2 バイト文字で構成されている必要があります。

5b 5 で記述された条件に応じて、互換性があります。さらに、ターゲット列に挿入される値は連結されたリスト項目のストリングで、それぞれシングル・スペースによって区切られます (リストの「縮小」空白ファセットに従う)。

5c 5a で記述された条件に応じて、互換性があります。さらに、ターゲット列に挿入される値は連結されたリスト項目のストリングで、それぞれシングル・スペースによって区切られます (リストの「縮小」空白ファセットに従う)。

5d XML 入力ストリングの長さ (バイト数) がターゲット列の長さ (バイト数) 以下の場合、互換性があります。入力ストリングがターゲット列より長い場合は、db2-xdb:truncate の列マッピングが「true」または「1」に設定される場合にのみ互換性があります。いずれかの場合のターゲット列に挿入される値は、要素または属性の文字内容です。

5e 5d で記述された条件に応じて、互換性があります。さらに、入力ストリングは 2 バイト文字で構成しなければなりません。

6 XML 入力ストリングの長さ (バイト数) がターゲット列の長さ (バイト数) 以下の場合、互換性があります。入力ストリングがターゲット列より長い場合は、db2-xdb:truncate の列マッピングが「true」または「1」に設定される場合にのみ互換性があります。ストリングの長さは正規化の後に計算されます。この正規化では、XML スキーマ・タイプの空白ファセットに応じて入力ストリングが正規化されます。入力 XML ストリングの長さが定義済みのターゲット列の長さより短い場合、ストリングの挿入時に右側にブランクが埋められます。

6a 6 で記述された条件に応じて、互換性があります。さらに、入力ストリングは 2 バイト文字で構成されている必要があります。

6b 6 で記述された条件に応じて、互換性があります。さらに、ターゲット列に挿入される値は連結されたリスト項目のストリングで、それぞれシングル・スペースによって区切られます (リストの「縮小」空白ファセットに従う)。

6c 6a で記述された条件に応じて、互換性があります。さらに、ターゲット列に挿入される値は連結されたリスト項目のストリングで、それぞれシングル・スペースによって区切られます (リストの「縮小」空白ファセットに従う)。

6d XML 入力ストリングの長さ (バイト数) がターゲット列の長さ (バイト数) 以下の場合、互換性があります。入力ストリングがターゲット列より長い場合は、db2-xdb:truncate の列マッピングが「true」または「1」に設定される場合にのみ互換性があります。いずれかの場合のターゲット列に挿入される

値は、要素または属性の文字内容です。入力 XML ストリングの長さが定義済みのターゲット列の長さより短い場合、ストリングの挿入時に右側にブランクが埋められます。

6e 6d で記述された条件に応じて、互換性があります。さらに、入力ストリングは 2 バイト文字で構成しなければなりません。

7 XML 入力ストリングの長さ (バイト数) がターゲット列の長さ (バイト数) 以下の場合、互換性があります。入力ストリングがターゲット列より長い場合は、db2-xdb:truncate の列マッピングが「true」または「1」に設定される場合にのみ互換性があります。ストリングの長さは正規化の後に計算されます。この正規化では、XML スキーマ・タイプの空白ファセットに応じて入力ストリングが正規化されます。

7a 7 で記述された条件に応じて、互換性があります。さらに、入力 XML ストリングの長さが定義済みのターゲット列の長さより短い場合、ストリングの挿入時に右側にブランクが埋められます。

7b 7 で記述された条件に応じて、互換性があります。さらに、ターゲット列に挿入される値は連結されたリスト項目のストリングで、それぞれシングル・スペースによって区切られます (リストの「縮小」空白ファセットに従う)。

7c 7b で記述された条件に応じて、互換性があります。さらに、入力 XML ストリングの長さが定義済みのターゲット列の長さより短い場合、ストリングの挿入時に右側にブランクが埋められます。

7d XML 入力ストリングの長さ (バイト数) がターゲット列の長さ (バイト数) 以下の場合、互換性があります。入力ストリングがターゲット列より長い場合は、db2-xdb:truncate の列マッピングが「true」または「1」に設定される場合にのみ互換性があります。いずれかの場合のターゲット列に挿入される値は、要素または属性の文字内容です。

7e 7d で記述された条件に応じて、互換性があります。さらに、入力 XML ストリングの長さが定義済みのターゲット列の長さより短い場合、ストリングの挿入時に右側にブランクが埋められます。

8 XML 入力ストリングの長さ (バイト数) がターゲット列の長さ (バイト数) 以下の場合、互換性があります。入力ストリングがターゲット列より長い場合は、db2-xdb:truncate の列マッピングが「true」または「1」に設定される場合にのみ互換性があります。エンコードされた (オリジナルの) ストリングが挿入されます。

8a 8 で記述された条件に応じて、互換性があります。さらに、入力 XML ストリングの長さが定義済みのターゲット列の長さより短い場合、ストリングの挿入時に右側にブランクが埋められます。

8b XML 入力ストリングの長さ (バイト数) がターゲット列の長さ (バイト数) 以下の場合、互換性があります。入力ストリングがターゲット列より長い場合は、db2-xdb:truncate の列マッピングが「true」または「1」に設定される場合にのみ互換性があります。ターゲット列に挿入される値は、デコードされたストリングです。

8c 8b で記述された条件に応じて、互換性があります。さらに、入力 XML ストリングの長さが定義済みのターゲット列の長さより短い場合、ストリングの挿入時に右側にブランクが埋められます。

9 db2-xdb:normalization 設定に応じた処理の後に計算される XML 入力ストリングの長さがターゲット列の長さ以下の場合、互換性があります。db2-xdb:truncate の列マッピングが「true」または「1」に設定される場合にも互換性があります。

9a 9 で記述された条件に応じて、互換性があります。さらに、入力 XML ストリングの長さが定義済みのターゲット列の長さより短い場合、ストリングの挿入時に右側にブランクが埋められます。

10 XML タイプが SQL タイプの範囲内にある場合、互換性があります。 -0 が XML タイプの値スペースにある場合、-0 はデータベース内では 0 として保管されます。

11 XML 値が SQL タイプの範囲内にある場合、互換性があります。有効数字の消失が起きる可能性があります。 -0 が XML タイプの値スペースにある場合、-0 はデータベース内では 0 として保管されます。

12 互換性があり、挿入される値は「0」(false の場合)、または「1」(true の場合) です。

13 db2-xdb:normalization 設定に応じた処理の後に計算される XML 入力ストリングの長さがターゲット列の長さ以下の場合、互換性があります。db2-xdb:truncate の列マッピングが「true」または「1」に設定される場合にも互換性があります。

13a 13 で記述された条件に応じて、互換性があります。さらに、入力 XML ストリングの長さが定義済みのターゲット列の長さより短い場合、ストリングの挿入時に右側にブランクが埋められます。

14 サブ秒を含む XML 値では、分解アノテーションが db2-xdb:truncate を「true」または「1」として指定する場合にのみ互換性があります。時間帯標識を持つ XML 値の場合、db2-xdb:truncate が「true」または「1」に設定されると互換性があります。値は時間帯なしで挿入されます。

15 年が 4 桁の数字で構成され、その前に「-」符号が付かない場合、互換性があります。 XML 値が時間帯標識を持たない場合、互換性があります。XML 値が時間帯標識を持つ場合、db2-xdb:truncate が「true」または「1」に設定されると互換性があります。

16 値が SQL タイプの範囲内にあり、「INF」、「-INF」、または「NaN」ではない場合に互換性があります。 -0 が XML タイプの値スペースにある場合、-0 はデータベース内では 0 として保管されます。有効数字の消失が起きる可能性があります。

17 値が「INF」、「-INF」、または「NaN」ではない場合に互換性があります。 -0 が XML タイプの値スペースにある場合、-0 はデータベース内では 0 として保管されます。

18 URI のストリングの長さ (バイト数) がターゲット列の長さ (バイト数) 以下の場合、互換性があります。入力ストリングがターゲット列より長い場合は、db2-xdb:truncate の列マッピングが「true」または「1」に設定される場合にのみ互換性があります。 URI が指すリソースではなく、URI そのものが挿入されることにご注意ください。

18a 18 で記述された条件に応じて、互換性があります。さらに、入力 XML ス

トリングの長さが定義済みのターゲット列の長さより短い場合、ストリングの挿入時に右側にブランクが埋められます。

19 年が 4 桁の数字で構成され、その前に「-」符号が付かない場合、互換性があります。時間帯標識を持つ XML 値の場合、db2-xdb:truncate が「true」または「1」に設定されると互換性があります。 (この場合は時間帯を持たない値が挿入されます。) 6 桁を超えるサブ秒が指定される場合、db2-xdb:truncate が「true」または「1」に設定されると互換性があります。

20 年が 4 桁の数字で構成され、その前に「-」符号が付かない場合、互換性があります。時間帯標識を持つ XML 値の場合、db2-xdb:truncate が「true」または「1」に設定されると互換性があります。 (この場合は時間帯を持たない日付値が挿入されます。)

21 数値の小数部分は切り捨てられます。整数部分が SQL タイプの範囲内にある場合、互換性があります。 -0 が XML タイプの値スペースにある場合、-0 はデータベース内では 0 として保管されます。

22 数値の小数部分は切り捨てられます。整数部分が SQL タイプの範囲内にあり、値が「INF」、「-INF」、または「NaN」ではない場合に互換性があります。 -0 が XML タイプの値スペースにある場合、-0 はデータベース内では 0 として保管されます。

アノテーション付き XML スキーマ分解の制限と制約事項

アノテーション付き XML スキーマ分解には特定の制限と制約事項が適用されます。

制限

表 73. アノテーション付き XML スキーマ分解の制限

条件	しきい値
分解する文書の最大サイズ	2 GB
1 つのアノテーション付き XML スキーマで参照される表の最大数	100
db2-xdb:expression アノテーションにおける $DECOMP_CONTENT または $DECOMP_ELEMENTID インスタンスの最大数	10
db2-xdb:locationPath 内のステップの最大数	100
<xs:any> または <xs:anyAttribute> の「namespace」属性に明示的にリストされる名前空間の最大数 (リストに特殊値 `##targetNamespace` または `##local` が含まれる場合、それらも制限に対してプラスされます)	25
db2-xdb:name (表名)、db2-xdb:column、db2-xdb:defaultSQLSchema、または db2-xdb:SQLSchema の値の最大ストリング長	対応する DB2 オブジェクトの制限と同じ
db2-xdb:rowSet の値の最大ストリング長	db2-xdb:name の制限と同じ

表 73. アノテーション付き XML スキーマ分解の制限 (続き)

条件	しきい値
$DECOMP_CONTENT の値の最大ストリング長	4096 バイト

制約事項

- アノテーション付き XML スキーマ分解は以下をサポートしていません。
  - 要素または属性ワイルドカードの分解: XML スキーマ内の <xs:any> または <xs:anyAttribute> 宣言に対応する XML 文書内の要素または属性は分解されません。

    しかし、これらの要素または属性が、「serializeSubtree」または「stringValue」に設定されている db2-xdb:contentHandling によって分解される要素の子である場合、ワイルドカードの要素または属性の内容はシリアライズされたサブツリーまたはストリング値の一部として分解されます。ただし、シリアライゼーションの一部となるためには、これらのワイルドカード要素または属性が、対応する <xs:any> または <xs:anyAttribute> 宣言で指定される名前空間制約を満たさなければなりません。
  - 置換グループ: 置換グループのメンバーが XML 文書内に出現し、グループのヘッドが XML スキーマに出現する場合、その置換グループのメンバーが文書のルート要素としてのみ使用されているのでなければ、エラーが生成されます。

    対処策として、置換グループのヘッドおよびメンバーの要素宣言を、代わりにタイプ xs:choice の名前付きモデル・グループに変更することができます。例えば、以下の置換グループの宣言があります。

    ```
    <xs:element name="head" type="BaseType" />
    <xs:element name="member1" type="derived1FromBaseType" substitutionGroup="head"/>
    <xs:element name="member2" type="derived2FromBaseType" substitutionGroup="head"/>
    <xs:element name="member3" type="derived3FromBaseType" substitutionGroup="head"/>
    ```

    これらは以下のように同等の名前付きモデル・グループに変更できます。

    ```
    <xs:group name="mysubstitutiongrp">
      <xs:choice>
        <xs:element name="head" type="BaseType"/>
        <xs:element name="member1" type="derived1FromBaseType"/>
        <xs:element name="member2" type="derived2FromBaseType"/>
         <xs:element name="member3" type="derived3FromBaseType"/>
      </xs:choice>
    </xs:group>
    ```

    <head> 要素のオカレンスは、XML 文書内の新しく定義された名前付きモデル・グループと置き換えることができます。
  - xsi:type を使用した実行時置換: 要素はスキーマの要素名に関連したスキーマ・タイプのマッピングに従って分解されます。 xsi:type を使用して文書内の要素に別のタイプを指定すると、分解時にエラーが戻されます。

    XML 文書内の xsi:type で指定された要素のタイプが、コンテキスト内のその要素に指定されたタイプと一致することを確認してください。要素の内容また

はその子孫が個別に分解される必要がない場合、要素のタイプは XML スキーマ内の xs:anyType に変更できます。この変更により、XML 文書は変更する必要がなくなります。

- 再帰的要素: 再帰が含まれる XML スキーマは、XML スキーマ・リポジトリー (XSR) に登録して分解を可能にすることができます。ただし、関連した XML インスタンス文書の再帰的セクションはスカラー値としてターゲット表へ分解できません。適切なスキーマ・アノテーションを使用することにより、再帰的セクションを保管し、後でシリアライズされたマークアップとして取り出すことができます。
- ターゲット表の既存の行の更新または削除: 分解でサポートされるのは新規行の挿入だけです。 (XML 分解プロセスの外側で行を更新または削除することは可能です。)
- NOTATION から派生する単純タイプの属性: 分解では表記名だけが挿入されます。
- ENTITY タイプの属性: 分解ではエンティティー名だけが挿入されます。
- db2-xdb:expression および db2-xdb:condition を使用した同じ rowSet および column への複数マッピング: マッピング規則に従って複数の項目を同じ rowSet と column にマップできる場合、マッピングに db2-xdb:expression または db2-xdb:condition アノテーションが含まれていてはなりません。

- パーティション・データベース環境では、アノテーション付き XML スキーマ分解は、データベース・カタログ表を含むデータベース・パーティション (IBMCATGROUP データベース・パーティション) でのみサポートされます。

アノテーション付き XML スキーマ分解のトラブルシューティングに関する考慮事項

期待される結果にならない分解が見つかった場合は、以下の事柄について考慮してください。

全般的な考慮事項

- XML スキーマに対応する XSR オブジェクトが、SYSCAT.XSROBJECTS カタログ・ビューの DECOMPOSITION 列で使用可能として表示されていることを確認します。XSR オブジェクトが使用可能でない場合、使用不可化についての文書で説明されている修正処置を取ることを考慮してください。
- XML 分解に関する制限と制約事項に違反していないことを確認します。
- XML 文書がその XML スキーマに従って検査済みであることを確認します。妥当性検査は分解のための要件ではありませんが、文字エンティティーの拡張などの特定の動作を期待している場合には、妥当性検査とともに分解を実行してください。

XML スキーマの問題

- 非決定的な内容モデルなどのエラーが XML スキーマに含まれていないことを確認します。なぜなら、これらのタイプのエラーにより妥当性検査の実行時に分解が失敗したり、妥当性検査が実行されなかった場合には未定義の分解が実行される可能性があるからです。

- グローバルではないアノテーションが要素または属性の宣言のみに宣言されていて、複合タイプ、要素または属性の参照、モデル・グループ、または他の XML スキーマ構成には宣言されていないことを確認します。また、サポートされている形式 (属性、要素、またはグローバル・アノテーション) でアノテーションが宣言されていることも確認します。(アノテーションの指定方法について詳しくは、アノテーションごとの資料を参照してください。)
- 拡張または制限によって派生する複合タイプのアノテーションが適切に付けられていることを確認します。

特定のエラー

データベース構成パラメーターを調整すると、次のエラーが解決できます。

- アノテーション付き XML スキーマに多数の rowSet が含まれている場合に受け取る SQL0954C。 applheapsz 構成パラメーターを使用してアプリケーション・ヒープ・サイズを増やします。
- アノテーション付き XML スキーマの各 rowSet に複雑または多数の式が含まれている場合に受け取る SQL0954C。 applheapsz 構成パラメーターを使用してアプリケーション・ヒープ・サイズを増やします。
- 分解により多数の行が生成される場合に受け取る SQL0964C。 **logprimary** および **logsecond** 構成パラメーターを使用して、使用可能な 1 次または 2 次ログ・ファイルの数を増やします。**logfilsiz** 構成パラメーターで 1 次および 2 次ログ・ファイルのサイズを増やすこともできます。

ロックおよび並行性

文書の分解中にロック・エスカレーションまたはデッドロックが起こる場合、ご利用のアプリケーションから並行性制御を調整します。アプリケーションが xdbDecompXML ストアード・プロシージャーのいずれかを同時に複数呼び出していて、その同じ表の多くが複数の分解操作に関係している場合、そのアプリケーションは、ロック・エスカレーションおよびデッドロックを回避するためにこれらの表への同時アクセスを管理する必要があります。

並行性制御を調整する 1 つの方法は、xdbDecompXML ストアード・プロシージャーを呼び出す前に、分解に関係するすべての表を明示的にロックするというものです。その後、そのストアード・プロシージャーが戻った後に適宜 COMMIT または ROLLBACK ステートメントを実行します。大きな文書の分解により多数の行が挿入される場合があるため、また、各行は挿入操作中にデフォルトでロックされるため、多くの行を挿入しているアプリケーションは多くの行ロックを保持しロック・エスカレーションになる場合があります。代わりに表ロックを取得することにより、行ロック取得のオーバーヘッドとロック・エスカレーションのオーバーヘッドを回避することができます。

表ロックの取得と関連した並行性をこのように削減することがご使用のアプリケーションにおいて適切でない場合には、maxlocks と locklist データベース構成パラメーターのいずれかまたは両方を増やし、ロック・エスカレーションの可能性を減らすことができます。

locktimeout データベース構成パラメーターを設定すると、アプリケーションがロックを取得するために無期限に待機することを防げます。

カタログ・ビューでのマッピング検査

上記にリストされている条件を検証した後でも引き続き分解に関する問題が起こる場合は、SYSCAT.XDBMAPSHREDTREES カタログ・ビューの MAPPINGDESCRIPTION 列が、意図したマッピングと一致しているか検査してください。MAPPINGDESCRIPTION 列には、以下の内容を含む、rowSet 中の個々の項目がマップされた方法に関する詳細情報が含まれています。

- ターゲットの列名
- ターゲットの列タイプ
- 項目の XML スキーマ・タイプ
- db2-xdb:contentHandling、db2-xdb:normalization、db2-xdb:truncate、db2-xdb:expression、および db2-xdb:condition の指定値

MAPPINGDESCRIPTION 以外の SYSCAT.XDBMAPSHREDTREES の列は DB2 お客様サポート部門用の内部情報です。

XML 分解アノテーションのスキーマ

アノテーション付き XML スキーマ分解は、XML 文書を分解してデータベース表に挿入する方法を指定するための分解アノテーションのセットをサポートしています。このトピックでは、XML 分解によって定義されるアノテーション付きスキーマのための XML スキーマを示します。

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns="http://www.ibm.com/xmlns/prod/db2/xdb1"
           targetNamespace="http://www.ibm.com/xmlns/prod/db2/xdb1"
           elementFormDefault="qualified" >
  <xs:element name="defaultSQLSchema" type="xs:string"/>
  <xs:attribute name="rowSet" type="xs:string"/>
  <xs:attribute name="column" type="xs:string"/>
  <xs:attribute name="locationPath" type="xs:string"/>
  <xs:attribute name="truncate" type="xs:boolean"/>
  <xs:attribute name="contentHandling">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="text"/>
        <xs:enumeration value="serializeSubtree"/>
        <xs:enumeration value="stringValue"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
  <xs:attribute name="normalization" >
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="original"/>
        <xs:enumeration value="whitespaceStrip"/>
        <xs:enumeration value="canonical"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
  <xs:attribute name="expression" type="xs:string"/>
  <xs:attribute name="condition" type="xs:string"/>
  <xs:element name="table">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="SQLSchema" type="xs:string" minOccurs="0"/>
        <xs:element name="name" type="xs:string"/>
        <xs:element name="rowSet" type="xs:string"
                    maxOccurs="unbounded" form="qualified"/>
```

```
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="rowSetMapping">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="rowSet" type="xs:string" />
        <xs:element name="column" type="xs:string" minOccurs="0"/>
        <xs:element name="expression" type="xs:string" minOccurs="0" />
        <xs:element name="condition" type="xs:string" minOccurs="0"/>
      </xs:sequence>
      <xs:attribute ref="truncate" />
      <xs:attribute ref="locationPath" />
      <xs:attribute ref="normalization" />
      <xs:attribute ref="contentHandling" />
    </xs:complexType>
  </xs:element>
  <xs:element name='rowSetOperationOrder'>
    <xs:complexType>
      <xs:choice minOccurs='1'  maxOccurs='1'>
        <xs:element name='order' type='orderType' minOccurs='1'
 maxOccurs='unbounded'/>
      </xs:choice>
    </xs:complexType>
  </xs:element>
  <xs:complexType name='orderType'>
    <xs:sequence>
      <xs:element name='rowSet' type='xsd:string' minOccurs='2'
      maxOccurs='unbounded'/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

第 14 章 pureXML に対する制約事項

pureXML に対する制約事項

pureXML は特定の制約事項の影響を受けます。こうした制約事項には、XML 列定義の制約事項、表をパーティション表にアタッチする際の制約事項、およびパーティション・データベース環境における制約事項があります。

XML 列定義に対する制約事項

XML 列は、以下の制約事項に従います。

- XML 列で索引を作成する場合、GENERATE KEY USING XMLPATTERN 節を使用する必要があり、その索引を複合索引の一部とすることはできません。複数の索引を 1 つの XML 列に対して作成できます。
- XML 列を CHECK 制約で参照できるのは VALIDATED 述部と組み合わせた場合のみです。
- XML 列は WITH DEFAULT 節で指定されるデフォルト値を持つことはできません。列が NULL 可能である場合、列のデフォルトは NULL です。
- XML 列は、範囲がクラスター化された表 (RCT) では使用できません。
- XML 列はキーの列として組み込むことはできません。これには、主キー、外部キー、ユニーク・キー、マルチディメンション・クラスタリング (MDC) 表のディメンション・キー (ORGANIZE BY 節内)、範囲がクラスター化された表の順序付けキー、パーティション表の表パーティション化キー、およびパーティション・データベース環境内の表の分散キーが含まれます。
- XML 列が含まれるパーティション表には、表パーティション化キー列として使用できるデータ・タイプの XML 以外の列が少なくとも 1 列含まれていなければなりません。
- XML 列は、型付き表および型付きビューに組み込むことはできません。
- XML 列は、生成列では参照できません。
- XML 列は、両方向スクロール・カーソルの選択リストでは指定できません。
- XML 列は、XML データの取得時にカーソル・ブロッキングが無効になる原因となります。
- ALTER TABLE ステートメントを使用して XML 列をドロップする場合、1 つの ALTER TABLE ステートメントで、表内のすべての XML 列をドロップしなければなりません。
- DB2 V9.7 フィックスパック 1 以降のリリースの場合、XML 列で定義された XML データに対する索引の分散統計を収集できます。以下の制限事項は、XML 列上の分散統計の収集に適用されます。
  - 分散統計は、XML 列で指定された XML データに対する索引ごとに収集されます。索引に指定するデータ・タイプは、VARCHAR、DOUBLE、INTEGER、DECIMAL、 TIMESTAMP、または DATE でなければなりません。分散統計は、タイプ VARCHAR HASHED の XML データの索引に対しては、収集されません。

- XML データに対する索引ごとの分散統計は、デフォルトで最大 250 の変位値を使用します。 **RUNSTATS** コマンドを発行するときに、**ON COLUMNS** または **DEFAULT** 節の **NUM_QUANTILES** パラメーターで値を指定することにより、デフォルト値を変更することができます。 XML 分散統計の収集中、**num_quantiles** データベース構成パラメーターは無視されます。
- **STATISTICS** オプションでデータをロードするとき、XML 分散統計は作成されません。
- XML 分散統計は、パーティション表で定義された XML データのパーティション索引に対しては、収集されません。

トリガーに関する制約事項

BEFORE または AFTER トリガーのトリガー本体では、影響を受ける行内にある XML タイプの列を参照する遷移変数を使用できるのは、XMLVALIDATE 関数を使用して妥当性検査する場合だけであり、XML 列値は NULL に設定されるか未変更のままになります。

パーティション表にパーティションをアタッチする際の制約事項

ALTER ATTACH を使用して XML 列が含まれるパーティション表にパーティションをアタッチする場合、アタッチ元となる表 (ソース表) のそれぞれの XML 列の INLINE LENGTH は、アタッチ先となる表 (ターゲット表) の対応する XML 列の INLINE LENGTH と一致しなければなりません。

バージョン 9.5 以前の XML レコード・フォーマットを使用する XML 列が含まれる表は、バージョン 9.7 以降のレコード・フォーマットを使用する XML 列が含まれるパーティション表にアタッチすることはできません。表をアタッチする前に、ターゲットのパーティション表の XML レコード・フォーマットと一致するように、表のレコード・フォーマットを更新する必要があります。以下の 2 つの方式のどちらかを使用して、表の XML レコード・フォーマットを更新します。

- ADMIN_MOVE_TABLE プロシージャーを使用して、オンラインで表の移動をします。
- 以下のステップを実行します。
  1. **EXPORT** コマンドを使用して表データのコピーを作成します。
  2. TRUNCATE ステートメントを使用して、表からすべての行を削除し、その表に割り振られているストレージを解放します。
  3. **LOAD** コマンドを使用して、表にデータを追加します。

表の XML レコード・フォーマットを更新した後で、その表をターゲットのパーティション表にアタッチします。

パーティション・データベース環境における制約事項

パーティション・データベース環境で pureXML を使用する場合、以下の規則が適用されます。

- XML 列を分散キーとして使用することはできません。そのため、以下の制約事項が当てはまります。
  - XML 列だけが含まれる表を分散することはできません。

  - 分散キーのある表は、主キー、ユニーク制約、またはユニーク索引を XML 列に定義できません。
  - XML 列が含まれる分散キーを持つ表には、分散キーとして使用できるデータ・タイプの XML 以外の列が少なくとも 1 列含まれていなければなりません。
- XML データを XML データ・ファイルからパーティション表に並列にロードする場合、並列でロードが実行されているすべてのパーティションに対して XML データ・ファイルが読み取り可能でなければなりません。
- XML データを複数パーティションのデータベースにロードするために **LOAD** コマンドで CURSOR ファイル・タイプを使用する場合、PARTITION_ONLY モードと LOAD_ONLY モードはサポートされません。
- **REDISTRIBUTE DATABASE PARTITION GROUP** コマンドで **NOT ROLLFORWARD RECOVERABLE** オプションを指定する場合、この再配分操作では、XML 列が含まれる表に対して **INDEXING MODE DEFERRED** オプションが使用されます。表に XML 列が含まれていないと、再配分操作では、このコマンドの発行時に指定された索引モードが使用されます。
- バージョン 9.5 以前の XML レコード・フォーマットを使用する XML 列が含まれる表では、再配分は行えません。表を新しいフォーマットにマイグレーションするには、ADMIN_MOVE_TABLE ストアード・プロシージャーを使用します。

データ行圧縮に関する制約事項

ALTER TABLE または CREATE TABLE ステートメントの COMPRESS YES オプションを使用すると、表のデータ行圧縮が有効になります。表の XML ストレージ・オブジェクト内にあるデータの圧縮は、バージョン 9.5 以前の XML レコード・フォーマットを使用する XML 列が含まれる表ではサポートされません。データ行の圧縮でこうした表を対象にした場合、表オブジェクトの表の行データのみが圧縮されます。

表でデータ行圧縮が有効なものの、表の XML ストレージ・オブジェクトを挿入、ロード、また reorg 操作の際に圧縮できない場合、メッセージが **db2diag** ログ・ファイルに書き込まれます。

表の XML ストレージ・オブジェクトのデータを圧縮に適したものにするには、ADMIN_MOVE_TABLE ストアード・プロシージャーを使用して表をマイグレーションしてから、データ行の圧縮をマイグレーション済みの表で使用可能にします。

追加の制約事項

シリアライズされた XML データの使用: データベース内に保管される XML 値のサイズに対するアーキテクチャー上の制限はありませんが、データベースで交換されるシリアライズされた XML データは事実上 2 GB に制限されます。

RUNSTATS コマンドの使用: XML データを使用する **ALLOW READ ACCESS** ロード操作が完了し、表が SET INTEGRITY ペンディング状態になっている場合に、その表に対して **RUNSTATS** コマンドを実行することができます。このようなシナリオの場合、 **RUNSTATS** 操作では、前のロード操作による不可視の XML 索引キーを認識することができないため、エラーが返されます。回避策として、**RUNSTATS** コマンドの前に SET INTEGRITY ステートメントを実行してください。

LOAD コマンドの使用 : 以下の場合、**LOAD** コマンドを使用して XML データをロードする際に、 **FOR EXCEPTION** 節を使用してロード例外表を指定することはサポートされません。

- ラベル・ベースのアクセス制御 (LBAC) を使用する場合。
- パーティション表にデータをロードする場合。

XML 列に対する索引の作成: XML 列の索引の作成と XSLT スタイルシートを使用した変換には、追加の制約事項があります。

5000 を超える maxOccurs 属性値の使用: DB2 バージョン 9.7 フィックスパック 1 以降では、 DB2 XSR に登録された XML スキーマで 5000 を超える値の maxOccurs 属性を使用する場合、その maxOccurs 属性値は「unbounded」を指定された場合と同様に扱われます。文書要素の maxOccurs 属性値が 5000 を超える値の場合は「unbounded」が指定された場合と同様に処理されるので、ある要素の出現回数が XML 文書の妥当性検査に使用される XML スキーマで指定された最大数を超えるとしても、XMLVALIDATE 関数を使用した XML 文書に対する妥当性検査を通過する可能性があります。追加情報および推奨される回避策については、XMLVALIDATE 関数の情報を確認してください。

RESTORE DATABASE コマンドの使用: SQL スキーマ内のいずれかの表に XML 列が含まれる場合、**TRANSPORT** オプションを使用して表スペースと SQL スキーマの転送を行うことはできません。

付録 A. エンコード・マッピング

エンコード名から保管済み XML データ用の有効な CCSID へのマッピング

XML 列に保管するデータがバイナリー・アプリケーション変数である場合、DB2 データベース・マネージャーはエンコードを判別するためにデータを調べます。データにエンコード宣言がある場合、データベース・マネージャーはエンコード名を CCSID にマップします。

352 ページの表 39 はこれらのマッピングをリストしています。 352 ページの表 39 にエンコード名がない場合、データベース・マネージャーはエラーを戻します。

352 ページの表 39 の最初の列にある正規化されたエンコード名は、エンコード名を大文字に変換し、ハイフン、正符号、下線、コロン、ピリオド、およびスペースをすべて除去した結果です。例えば、ISO88591 は ISO 8859-1、ISO-8859-1、および iso-8859-1 を正規化したエンコード名です。

表 74. エンコード名および有効な CCSID

正規化されたエンコード名	CCSID
437	437
646	367
813	813
819	819
850	850
852	852
855	855
857	857
862	862
863	863
866	866
869	869
885913	901
885915	923
88591	819
88592	912
88595	915
88597	813
88598	62210
88599	920
904	904
912	912
915	915

表 74. エンコード名および有効な CCSID (続き)

正規化されたエンコード名	CCSID
916	916
920	920
923	923
ANSI1251	1251
ANSIX341968	367
ANSIX341986	367
ARABIC	1089
ASCII7	367
ASCII	367
ASMO708	1089
BIG5	950
CCSID00858	858
CCSID00924	924
CCSID01140	1140
CCSID01141	1141
CCSID01142	1142
CCSID01143	1143
CCSID01144	1144
CCSID01145	1145
CCSID01146	1146
CCSID01147	1147
CCSID01148	1148
CCSID01149	1149
CP00858	858
CP00924	924
CP01140	1140
CP01141	1141
CP01142	1142
CP01143	1143
CP01144	1144
CP01145	1145
CP01146	1146
CP01147	1147
CP01148	1148
CP01149	1149
CP037	37
CP1026	1026
CP1140	1140
CP1141	1141
CP1142	1142

表 74. エンコード名および有効な CCSID (続き)

正規化されたエンコード名	CCSID
CP1143	1143
CP1144	1144
CP1145	1145
CP1146	1146
CP1147	1147
CP1148	1148
CP1149	1149
CP1250	1250
CP1251	1251
CP1252	1252
CP1253	1253
CP1254	1254
CP1255	1255
CP1256	1256
CP1257	1257
CP1258	1258
CP1363	1363
CP1383	1383
CP1386	1386
CP273	273
CP277	277
CP278	278
CP280	280
CP284	284
CP285	285
CP297	297
CP33722	954
CP33722C	954
CP367	367
CP420	420
CP423	423
CP424	424
CP437	437
CP500	500
CP5346	5346
CP5347	5347
CP5348	5348
CP5349	5349
CP5350	5350
CP5353	5353

表 74. エンコード名および有効な CCSID (続き)

正規化されたエンコード名	CCSID
CP813	813
CP819	819
CP838	838
CP850	850
CP852	852
CP855	855
CP857	857
CP858	858
CP862	862
CP863	863
CP864	864
CP866	866
CP869	869
CP870	870
CP871	871
CP874	874
CP904	904
CP912	912
CP915	915
CP916	916
CP920	920
CP921	921
CP922	922
CP923	923
CP936	1386
CP943	943
CP943C	943
CP949	970
CP950	950
CP964	964
CP970	970
CPGR	869
CSASCII	367
CSBIG5	950
CSEBCDICCAFR	500
CSEBCDICDKNO	277
CSEBCDICES	284
CSEBCDICFISE	278
CSEBCDICFR	297
CSEBCDICIT	280

表 74. エンコード名および有効な CCSID (続き)

正規化されたエンコード名	CCSID
CSEBCDICPT	37
CSEBCDICUK	285
CSEBCDICUS	37
CSEUCKR	970
CSEUCPKDFMTJAPANESE	954
CSGB2312	1383
CSHPROMAN8	1051
CSIBM037	37
CSIBM1026	1026
CSIBM273	273
CSIBM277	277
CSIBM278	278
CSIBM280	280
CSIBM284	284
CSIBM285	285
CSIBM297	297
CSIBM420	420
CSIBM423	423
CSIBM424	424
CSIBM500	500
CSIBM855	855
CSIBM857	857
CSIBM863	863
CSIBM864	864
CSIBM866	866
CSIBM869	869
CSIBM870	870
CSIBM871	871
CSIBM904	904
CSIBMEBCDICATDE	273
CSIBMTHAI	838
CSISO128T101G2	920
CSISO146SERBIAN	915
CSISO147MACEDONIAN	915
CSISO2INTLREFVERSION	367
CSISO646BASIC1983	367
CSISO88596I	1089
CSISO88598I	916
CSISOLATIN0	923
CSISOLATIN1	819

表 74. エンコード名および有効な CCSID (続き)

正規化されたエンコード名	CCSID
CSISOLATIN2	912
CSISOLATIN5	920
CSISOLATIN9	923
CSISOLATINARABIC	1089
CSISOLATINCYRILLIC	915
CSISOLATINGREEK	813
CSISOLATINHEBREW	62210
CSKOI8R	878
CSKSC56011987	970
CSMACINTOSH	1275
CSMICROSOFTPUBLISHING	1004
CSPC850MULTILINGUAL	850
CSPC862LATINHEBREW	862
CSPC8CODEPAGE437	437
CSPCP852	852
CSSHIFTJIS	943
CSUCS4	1236
CSUNICODE11	1204
CSUNICODE	1204
CSUNICODEASCII	1204
CSUNICODELATIN1	1204
CSVISCII	1129
CSWINDOWS31J	943
CYRILLIC	915
DEFAULT	367
EBCDICATDE	273
EBCDICCAFR	500
EBCDICCPAR1	420
EBCDICCPBE	500
EBCDICCPCA	37
EBCDICCPCH	500
EBCDICCPDK	277
EBCDICCPES	284
EBCDICCPFI	278
EBCDICCPFR	297
EBCDICCPGB	285
EBCDICCPGR	423
EBCDICCPHE	424
EBCDICCPIS	871
EBCDICCPIT	280

表 74. エンコード名および有効な CCSID (続き)

正規化されたエンコード名	CCSID
EBCDICCPNL	37
EBCDICCPNO	277
EBCDICCPROECE	870
EBCDICCPSE	278
EBCDICCPUS	37
EBCDICCPWT	37
EBCDICCPYU	870
EBCDICDE273EURO	1141
EBCDICDK277EURO	1142
EBCDICDKNO	277
EBCDICES284EURO	1145
EBCDICES	284
EBCDICFI278EURO	1143
EBCDICFISE	278
EBCDICFR297EURO	1147
EBCDICFR	297
EBCDICGB285EURO	1146
EBCDICINTERNATIONAL500EURO	1148
EBCDICIS871EURO	1149
EBCDICIT280EURO	1144
EBCDICIT	280
EBCDICLATIN9EURO	924
EBCDICNO277EURO	1142
EBCDICPT	37
EBCDICSE278EURO	1143
EBCDICUK	285
EBCDICUS37EURO	1140
EBCDICUS	37
ECMA114	1089
ECMA118	813
ELOT928	813
EUCCN	1383
EUCJP	954
EUCKR	970
EUCTW	964
EXTENDEDUNIXCODEPACKEDFORMATFORJAPANESE	954
GB18030	1392
GB2312	1383
GBK	1386
GREEK8	813

表 74. エンコード名および有効な CCSID (続き)

正規化されたエンコード名	CCSID
GREEK	813
HEBREW	62210
HPROMAN8	1051
IBM00858	858
IBM00924	924
IBM01140	1140
IBM01141	1141
IBM01142	1142
IBM01143	1143
IBM01144	1144
IBM01145	1145
IBM01146	1146
IBM01147	1147
IBM01148	1148
IBM01149	1149
IBM01153	1153
IBM01155	1155
IBM01160	1160
IBM037	37
IBM1026	1026
IBM1043	1043
IBM1047	1047
IBM1252	1252
IBM273	273
IBM277	277
IBM278	278
IBM280	280
IBM284	284
IBM285	285
IBM297	297
IBM367	367
IBM420	420
IBM423	423
IBM424	424
IBM437	437
IBM500	500
IBM808	808
IBM813	813
IBM819	819
IBM850	850

表 74. エンコード名および有効な CCSID (続き)

正規化されたエンコード名	CCSID
IBM852	852
IBM855	855
IBM857	857
IBM862	862
IBM863	863
IBM864	864
IBM866	866
IBM867	867
IBM869	869
IBM870	870
IBM871	871
IBM872	872
IBM902	902
IBM904	904
IBM912	912
IBM915	915
IBM916	916
IBM920	920
IBM921	921
IBM922	922
IBM923	923
IBMTHAI	838
IRV	367
ISO10646	1204
ISO10646UCS2	1200
ISO10646UCS4	1232
ISO10646UCSBASIC	1204
ISO10646UNICODELATIN1	1204
ISO646BASIC1983	367
ISO646IRV1983	367
ISO646IRV1991	367
ISO646US	367
ISO885911987	819
ISO885913	901
ISO885915	923
ISO885915FDIS	923
ISO88591	819
ISO885921987	912
ISO88592	912
ISO885951988	915

表 74. エンコード名および有効な CCSID (続き)

正規化されたエンコード名	CCSID
ISO88595	915
ISO885961987	1089
ISO88596	1089
ISO88596I	1089
ISO885971987	813
ISO88597	813
ISO885981988	62210
ISO88598	62210
ISO88598I	916
ISO885991989	920
ISO88599	920
ISOIR100	819
ISOIR101	912
ISOIR126	813
ISOIR127	1089
ISOIR128	920
ISOIR138	62210
ISOIR144	915
ISOIR146	915
ISOIR147	915
ISOIR148	920
ISOIR149	970
ISOIR2	367
ISOIR6	367
JUSIB1003MAC	915
JUSIB1003SERB	915
KOI8	878
KOI8R	878
KOI8U	1168
KOREAN	970
KSC56011987	970
KSC56011989	970
KSC5601	970
L1	819
L2	912
L5	920
L9	923
LATIN0	923
LATIN1	819
LATIN2	912

表 74. エンコード名および有効な CCSID (続き)

正規化されたエンコード名	CCSID
LATIN5	920
LATIN9	923
MAC	1275
MACEDONIAN	915
MACINTOSH	1275
MICROSOFTPUBLISHING	1004
MS1386	1386
MS932	943
MS936	1386
MS949	970
MSKANJI	943
PCMULTILINGUAL850EURO	858
R8	1051
REF	367
ROMAN8	1051
SERBIAN	915
SHIFTJIS	943
SJIS	943
SUNEUGREEK	813
T101G2	920
TIS20	874
TIS620	874
UNICODE11	1204
UNICODE11UTF8	1208
UNICODEBIGUNMARKED	1200
UNICODELITTLEUNMARKED	1202
US	367
USASCII	367
UTF16	1204
UTF16BE	1200
UTF16LE	1202
UTF32	1236
UTF32BE	1232
UTF32LE	1234
UTF8	1208
VISCII	1129
WINDOWS1250	1250
WINDOWS1251	1251
WINDOWS1252	1252
WINDOWS1253	1253

表 74. エンコード名および有効な CCSID (続き)

正規化されたエンコード名	CCSID
WINDOWS1254	1254
WINDOWS1255	1255
WINDOWS1256	1256
WINDOWS1257	1257
WINDOWS1258	1258
WINDOWS28598	62210
WINDOWS31J	943
WINDOWS936	1386
XEUCTW	964
XMSWIN936	1386
XUTF16BE	1200
XUTF16LE	1202
XWINDOWS949	970

CCSID とシリアライズされた XML 出力データのエンコード名とのマップ

暗黙的または明示的 XMLSERIALIZE 操作の一部として、DB2 データベース・マネージャーはシリアライズされた XML 出力データの先頭にエンコード宣言を追加する場合があります。

宣言の形式は次のとおりです。

```
<?xml version="1.0" encoding="encoding-name"?>
```

一般に、エンコード宣言の文字セット ID は、文字のエンコードを出力ストリングに記述します。例えば、ターゲット・アプリケーションのデータ・タイプに一致する CCSID に XML データがシリアライズされると、エンコード宣言はターゲット・アプリケーション変数 CCSID を記述します。例外として、アプリケーションが INCLUDING XMLDECLARATION を指定して明示的な XMLSERIALIZE 関数を実行する場合があります。 INCLUDING XMLDECLARATION を指定すると、データベース・マネージャーは UTF-8 用のエンコード宣言を生成します。ターゲットのデータ・タイプが CLOB または DBCLOB タイプの場合、さらにコード・ページの変換が行われることがありますが、これよってエンコード情報が不正確になる可能性があります。アプリケーション内でデータがさらに構文解析されると、結果としてデータ破壊が起こる可能性があります。

可能な場合には、DB2 データベース・マネージャーは、 XML 規格の規定に従って、CCSID に対応する IANA レジストリー名を選択します。

表 75. CCSID とそれに対応するエンコード名

CCSID	エンコード名
37	IBM037
273	IBM273
277	IBM277

表 75. CCSID とそれに対応するエンコード名 (続き)

CCSID	エンコード名
278	IBM278
280	IBM280
284	IBM284
285	IBM285
297	IBM297
367	US-ASCII
420	IBM420
423	IBM423
424	IBM424
437	IBM437
500	IBM500
808	IBM808
813	ISO-8859-7
819	ISO-8859-1
838	IBM-Thai
850	IBM850
852	IBM852
855	IBM855
857	IBM857
858	IBM00858
862	IBM862
863	IBM863
864	IBM864
866	IBM866
867	IBM867
869	IBM869
870	IBM870
871	IBM871
872	IBM872
874	TIS-620
878	KOI8-R
901	ISO-8859-13
902	IBM902
904	IBM904
912	ISO-8859-2
915	ISO-8859-5
916	ISO-8859-8-I
920	ISO-8859-9
921	IBM921
922	IBM922

表 75. CCSID とそれに対応するエンコード名 (続き)

CCSID	エンコード名
923	ISO-8859-15
924	IBM00924
932	Shift_JIS
943	Windows-31J
949	EUC-KR
950	Big5
954	EUC-JP
964	EUC-TW
970	EUC-KR
1004	Microsoft-Publish
1026	IBM1026
1043	IBM1043
1047	IBM1047
1051	hp-roman8
1089	ISO-8859-6
1129	VISCII
1140	IBM01140
1141	IBM01141
1142	IBM01142
1143	IBM01143
1144	IBM01144
1145	IBM01145
1146	IBM01146
1147	IBM01147
1148	IBM01148
1149	IBM01149
1153	IBM01153
1155	IBM01155
1160	IBM-Thai
1161	TIS-620
1162	TIS-620
1163	VISCII
1168	KOI8-U
1200	UTF-16BE
1202	UTF-16LE
1204	UTF-16
1208	UTF-8
1232	UTF-32BE
1234	UTF-32LE
1236	UTF-32

表 75. CCSID とそれに対応するエンコード名 (続き)

CCSID	エンコード名
1250	windows-1250
1251	windows-1251
1252	windows-1252
1253	windows-1253
1254	windows-1254
1255	windows-1255
1256	windows-1256
1257	windows-1257
1258	windows-1258
1275	MACINTOSH
1363	KSC_5601
1370	Big5
1381	GB2312
1383	GB2312
1386	GBK
1392	GB18030
4909	ISO-8859-7
5039	Shift_JIS
5346	windows-1250
5347	windows-1251
5348	windows-1252
5349	windows-1253
5350	windows-1254
5351	windows-1255
5352	windows-1256
5353	windows-1257
5354	windows-1258
5488	GB18030
8612	IBM420
8616	IBM424
9005	ISO-8859-7
12712	IBM424
13488	UTF-16BE
13490	UTF-16LE
16840	IBM420
17248	IBM864
17584	UTF-16BE
17586	UTF-16LE
62209	IBM862
62210	ISO-8859-8

表 75. CCSID とそれに対応するエンコード名 (続き)

CCSID	エンコード名
62211	IBM424
62213	IBM862
62215	ISO-8859-8
62218	IBM864
62221	IBM862
62222	ISO-8859-8
62223	windows-1255
62224	IBM420
62225	IBM864
62227	ISO-8859-6
62228	windows-1256
62229	IBM424
62231	IBM862
62232	ISO-8859-8
62233	IBM420
62234	IBM420
62235	IBM424
62237	windows-1255
62238	ISO-8859-8-I
62239	windows-1255
62240	IBM424
62242	IBM862
62243	ISO-8859-8-I
62244	windows-1255
62245	IBM424
62250	IBM420

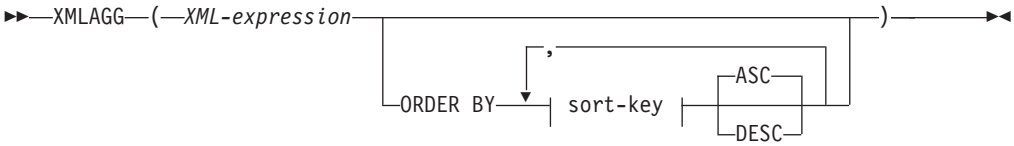
付録 B. SQL/XML 発行関数

以下のセクションでは、DB2 SQL/XML パブリッシング関数の構文について説明しています。

これらの関数の使用については、 127 ページの『XML 値を構成するための SQL/XML 発行関数』を参照してください。

XMLAGG

XMLAGG 関数は、それぞれの非 NULL 値の項目を XML 値のセットに含む XML シーケンスを戻します。



スキーマは SYSIBM です。関数名を修飾名で指定することはできません。

XML-expression
データ・タイプ XML の式を指定します。

ORDER BY
集合内の処理対象の、同じグループ化集合に属する行の順序を指定します。ORDER BY 節を省略した場合や、ORDER BY が列データの ORDER BY を特定できない場合、同一のグループ化集合内の行は任意に順序付けられます。

sort-key
ソート・キーは、列名または *sort-key-expression* のどちらでも構いません。ソート・キーが定数の場合、ソート・キーは出力列の位置を (通常の ORDER BY 節におけるように) 参照しませんが、これは単なる定数でしかなく、ソート・キーではないことを意味することに注意してください。

結果のデータ・タイプは XML です。

この関数は、引数の値から NULL 値を除いて求めた値の集合に対して適用されます。

XML-expression 引数が NULL 値になる可能性がある場合、結果も NULL 値になる可能性があります。値のセットが空の場合、結果は NULL 値になります。それ以外の場合、結果は各値の項目をセットに含む XML シーケンスになります。

SELECT 節に ARRAY_AGG 関数が組み込まれている場合、同じ SELECT 節にある ARRAY_AGG、LISTAGG、XMLAGG、および XMLGROUP 関数のすべての呼び出しにおいて、同じ順序を指定するか、または順序を指定しないかのいずれかにする必要があります (SQLSTATE 428GZ)。

注

- **OLAP 式でのサポート**: XMLAGG を、OLAP 集約関数の列関数として使用することはできません (SQLSTATE 42601)。

例

姓別にソートされた従業員のリストを含む、各部門の部門エレメントを構成します。

```
SELECT XMLSERIALIZE(
  CONTENT XMLELEMENT(
    NAME "Department", XMLATTRIBUTES(
      E.WORKDEPT AS "name"
    ),
    XMLAGG(
      XMLELEMENT(
        NAME "emp", E.LASTNAME
      )
      ORDER BY E.LASTNAME
    )
  )
  AS CLOB(110)
)
AS "dept_list"
FROM EMPLOYEE E
WHERE E.WORKDEPT IN ('C01','E21')
GROUP BY WORKDEPT
```

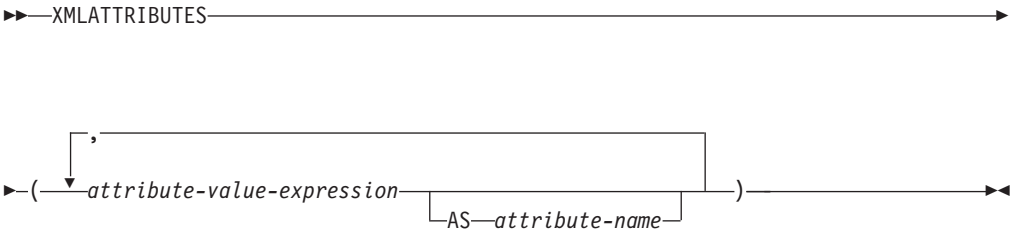
この照会は、次のような結果を生成します。

```
dept_list
----------------------...
<Department name="C01">
  <emp>KWAN</emp>
  <emp>NICHOLLS</emp>
  <emp>QUINTANA</emp>
</Department>
<Department name="E21">
  <emp>GOUNOT</emp>
  <emp>LEE</emp>
  <emp>MEHTA</emp>
  <emp>SPENSER</emp>
</Department>
```

注: XMLAGG は、出力の中にブランク・スペースまたは改行文字を挿入しません。例の出力はすべて、読みやすくするために書式を整えています。

XMLATTRIBUTES

XMLATTRIBUTES 関数は、引数から XML 属性を構成します。



スキーマは SYSIBM です。関数名を修飾名で指定することはできません。

この関数は、XMLELEMENT 関数の引数としてのみ使用できます。結果は、それぞれの非 NULL 入力値の XQuery 属性ノードを含む XML シーケンスになります。

attribute-value-expression
結果が属性値になる式。データ・タイプ *attribute-value-expression*を、構造化タイプとすることはできません (SQLSTATE 42884)。式には任意の SQL 式を指定できます。式が単純な列参照でない場合、属性名を指定する必要があります。

attribute-name
属性名を指定します。この名前は SQL ID であり、XML 修飾名の形式かまたは QName でなければなりません (SQLSTATE 42634)。有効な名前の詳細は、「W3C の XML 名前空間仕様」を参照してください。属性名を `xmlns` にしたり、その前に `xmlns:` を付けたりすることはできません。名前空間は、関数 XMLNAMESPACES を使って宣言します。重複した属性名を (暗黙的、明示的にかかわらず) 使用することはできません (SQLSTATE 42713)。

attribute-name が指定されない場合、*attribute-value-expression* は列名でなければなりません (SQLSTATE 42703)。属性名は、列名から XML 属性名への完全にエスケープしたマッピングを使用する列名から作成されます。

結果のデータ・タイプは XML です。 *attribute-value-expression* の結果が NULL になる可能性がある場合は、結果も NULL になる可能性があります。
attribute-value-expression の結果がすべて NULL であれば、結果も NULL 値になります。

例

注: XMLATTRIBUTES は、出力の中にブランク・スペースまたは改行文字を挿入しません。例の出力はすべて、読みやすくするために書式を整えています。

- *例 1*: エレメント、およびその属性を生成します。

```
SELECT E.EMPNO, XMLELEMENT(
  NAME "Emp",
  XMLATTRIBUTES(
    E.EMPNO, E.FIRSTNME ||' '|| E.LASTNAME AS "name"
  )
)
AS "Result"
FROM EMPLOYEE E WHERE E.EDLEVEL = 12
```

この照会は、次のような結果を生成します。

```
EMPNO  Result
000290 <Emp EMPNO="000290" name="JOHN PARKER"></Emp>
000310 <Emp EMPNO="000310" name="MAUDE SETRIGHT"></Emp>
200310 <Emp EMPNO="200310" name="MICHELLE SPRINGER"></Emp>
```

- *例 2*: エレメント、および QName で使用されない名前空間宣言を生成します。属性値には接頭部が使用されます。

```
VALUES XMLELEMENT(
  NAME "size",
  XMLNAMESPACES(
    'http://www.w3.org/2001/XMLSchema-instance' AS "xsi",
    'http://www.w3.org/2001/XMLSchema' AS "xsd"
  ),
```

```
    XMLATTRIBUTES(
      'xsd:string' AS "xsi:type"
    ), '1'
  )
```

この照会は、次のような結果を生成します。

```
<size xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xsi:type="xsd:string">1</size>
```

XMLCOMMENT

XMLCOMMENT 関数は、XQuery コメント・ノードを 1 つ持つ XML 値を戻します。その内容は入力引数です。

スキーマは SYSIBM です。関数名を修飾名で指定することはできません。

string-expression
値が文字ストリング・タイプ CHAR、VARCHAR、または CLOB を持つ式。 *string-expression* の結果は構文解析され、XML 1.0 規則で指定されている XML コメントの要件との適合性が検査されます。 *string-expression* の結果は次の正規表現に従っていなければなりません。

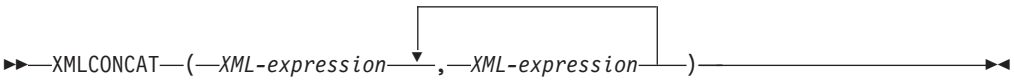
```
((Char - '-') | ('-' (Char - '-')))*
```

Char は、任意の Unicode 文字として定義されます。ただしサロゲート・ブロック X'FFFE' および X'FFFF' は除きます。基本的に、XML コメントに隣接する 2 つのハイフンを含めることはできません。また、XML コメントをハイフンで終了することもできません (SQLSTATE 2200S)。

結果のデータ・タイプは XML です。 *string-expression* の結果が NULL になる可能性がある場合、結果も NULL になる可能性があります。入力値が NULL であれば、結果も NULL 値になります。

XMLCONCAT

XMLCONCAT 関数は、可変数の XML 入力引数の連結を含むシーケンスを戻します。



スキーマは SYSIBM です。関数名を修飾名で指定することはできません。

XML-expression
データ・タイプ XML の式を指定します。

結果のデータ・タイプは XML です。結果は、非 NULL の入力 XML 値の連結を含む XML シーケンスになります。入力内の NULL 値は無視されます。いずれか

の *XML-expression* の結果が NULL になる可能性がある場合は、結果も NULL になる可能性があります。各入力値の結果が NULL であれば、結果も NULL 値になります。

例

注: XMLCONCAT は、出力の中にブランク・スペースまたは改行文字を挿入しません。例の出力はすべて、読みやすくするために書式を整えています。

姓別にソートされた従業員のリストを含む、部門 A00 および B01 の部門エレメントを構成します。部門エレメントの直前に紹介コメントを含めます。

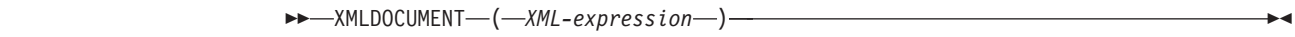
```
SELECT XMLCONCAT(
  XMLCOMMENT(
    'Confirm these employees are on track for their product schedule'
  ),
  XMLELEMENT(
    NAME "Department",
    XMLATTRIBUTES(
      E.WORKDEPT AS "name"
    ),
    XMLAGG(
      XMLELEMENT(
        NAME "emp", E.FIRSTNME
      )
    ORDER BY E.FIRSTNME
    )
  )
)
FROM EMPLOYEE E
WHERE E.WORKDEPT IN ('A00', 'B01')
GROUP BY E.WORKDEPT
```

この照会は、次のような結果を生成します。

```
<!--Confirm these employees are on track for their product schedule-->
<Department name="A00">
<emp>CHRISTINE</emp>
<emp>DIAN</emp>
<emp>GREG</emp>
<emp>SEAN</emp>
<emp>VINCENZO</emp>
</Department>
<!--Confirm these employees are on track for their product schedule-->
<Department name="B01">
<emp>MICHAEL</emp>
</Department>
```

XMLDOCUMENT

XMLDOCUMENT 関数は、XQuery 文書ノードを 1 つ持つ XML 値を戻します。これにはゼロ個以上の子ノードが含まれます。



スキーマは SYSIBM です。関数名を修飾名で指定することはできません。

XML-expression
XML 値を戻す式。 XML 値のシーケンス項目が属性ノードであってはなりません (SQLSTATE 10507)。

結果のデータ・タイプは XML です。 *XML-expression* の結果が NULL になる可能性がある場合、結果も NULL になる可能性があります。入力値が NULL であれば、結果も NULL 値になります。

その結果生成される文書ノードの子は、以下のステップの説明に従って構成されます。入力式はノードのシーケンスまたは原子値です。これはこのステップで内容シーケンスとして参照されます。

1. 内容シーケンスに文書ノードが含まれる場合、内容シーケンスの文書ノードはその文書ノードの子によって置き換えられます。
2. 内容シーケンス内の 1 つ以上の原子値の各隣接シーケンスは、各原子値をストリング (隣接する値の間にブランク文字を 1 つ挿入したもの) にキャストした結果を含むテキスト・ノードで置き換えられます。
3. 内容シーケンスの各ノードごとにノードの新規ディープ・コピーが構成されます。ノードのディープ・コピーとは、そのノードをルートとするサブツリー全体のコピーのことです (これにはノードそのものとその子孫が含まれます)。コピーされた各ノードは、新しいノード ID を持ちます。
4. 内容シーケンス内のノードは、新規文書ノードの子になります。

XMLDOCUMENT 関数は、XQuery が計算する文書コンストラクターを効果的に実行します。以下の関数があるとします。

```
XMLQUERY('document {$E}' PASSING BY REF XML-expression AS "E")
```

これは、以下と同じ意味になります。

```
XMLDOCUMENT( XML-expression )
```

ただし、*XML-expression* が NULL であり、XMLQUERY が空シーケンスを戻す場合 (XMLDOCUMENT の場合は NULL 値) は例外です。

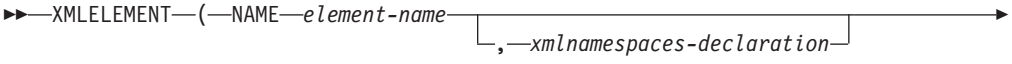
例

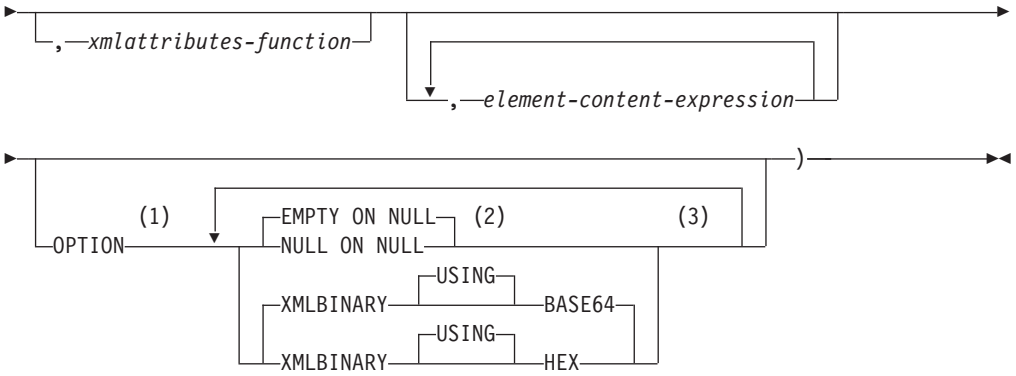
構成した文書を XML 列に挿入します。

```
INSERT INTO T1 VALUES(
  123, (
    SELECT XMLDOCUMENT(
      XMLELEMENT(
        NAME "Emp", E.FIRSTNME || ' ' || E.LASTNAME, XMLCOMMENT(
          'This is just a simple example'
        )
      )
    )
    FROM EMPLOYEE E
    WHERE E.EMPNO = '000120'
  )
)
```

XMLELEMENT

XMLELEMENT 関数は、XQuery エレメント・ノードである XML 値を戻します。





注:

1 OPTION 節は、少なくとも 1 つの *xmlattributes-function* または *element-content-expression* が指定されている場合にのみ指定できます。

2 NULL ON NULL または EMPTY ON NULL は、少なくとも 1 つの *element-content-expression* が指定されている場合にのみ指定できます。

3 同じ節を複数回指定することはできません。

スキーマは SYSIBM です。関数名を修飾名で指定することはできません。

NAME *element-name*
XML エレメントの名前を指定します。この名前は SQL ID であり、XML 修飾名の形式かまたは QName でなければなりません (SQLSTATE 42634)。有効な名前の詳細は、「W3C の XML 名前空間仕様」を参照してください。名前が修飾される場合は、名前空間の接頭部をその有効範囲内で宣言する必要があります (SQLSTATE 42635)。

xmlnamespaces-declaration
XMLNAMESPACES 宣言の結果である XML 名前空間宣言を指定します。宣言される名前空間は、XMLELEMENT 関数の有効範囲内です。この名前空間は、それらが別の副選択内に現れるかどうかに関係なく、XMLELEMENT 関数内のネストされた XML 関数に適用されます。

xmlnamespaces-declaration が指定されない場合、名前空間宣言は構成されたエレメントとは関連付けられません。

xmlattributes-function
エレメントの XML 属性を指定します。この属性は、XMLATTRIBUTES 関数の結果です。

element-content-expression
生成される XML エレメント・ノードの内容を、式によって、または式リストによって指定します。データ・タイプ *element-content-expression*を、構造化タイプとすることはできません (SQLSTATE 42884)。式には任意の SQL 式を指定できます。

element-content-expression が指定されない場合、空ストリングがエレメントの内容として使用され、OPTION NULL ON NULL または EMPTY ON NULL を指定することはできません。

OPTION

XML エレメントを構成するための追加オプションを指定します。 OPTION 節を指定しない場合、デフォルトは EMPTY ON NULL XMLBINARY USING BASE64 です。この節は、*element-content-expression* で指定するネストされた XMLELEMENT の呼び出しに影響を与えません。

EMPTY ON NULL or NULL ON NULL

各 *element-content-expression* の値が NULL 値の場合に NULL 値を戻すか、あるいは空エレメントを戻すかを指定します。このオプションはエレメントの内容の NULL 処理にのみ影響し、属性値の NULL 処理には影響を及ぼしません。デフォルトは EMPTY ON NULL です。

EMPTY ON NULL

それぞれの *element-content-expression* の値が NULL であれば、空のエレメントが戻されます。

NULL ON NULL

それぞれの *element-content-expression* の値が NULL であれば、NULL 値が戻されます。

XMLBINARY USING BASE64 or XMLBINARY USING HEX

バイナリー入力データ、FOR BIT DATA 属性を持つ文字ストリング・データ、またはこれらのタイプのいずれかに基づく特殊タイプの想定エンコードを指定します。エンコードはエレメントの内容または属性値に適用されます。デフォルトは XMLBINARY USING BASE64 です。

XMLBINARY USING BASE64

想定エンコードが Base64 文字である (XML スキーマ・タイプ xs:base64Binary のエンコードに対して定義される) ことを指定します。Base64 エンコードは、US-ASCII の 65 文字のサブセット (10 個の数字、26 個の小文字、26 個の大文字、'+' および '/') を使用して、サブセット内の 1 つの印刷可能文字により、バイナリーまたはビット・データのすべての 6 ビットを表します。文字を普遍的に表せるようにこれらの文字が選ばれています。この方法を使うと、エンコード・データのサイズが元のバイナリーまたはビット・データより 33 % 大きくなります。

XMLBINARY USING HEX

想定エンコードが 16 進文字である (XML スキーマ・タイプ xs:hexBinary のエンコードに対して定義される) ことを指定します。 16 進数エンコードは、各バイト (8 ビット) を 2 つの 16 進文字で表します。この方法を使うと、エンコード・データが元のバイナリーまたはビット・データの 2 倍のサイズになります。

この関数は、エレメント名、オプションの名前空間宣言の集合、オプションの属性の集合、および XML エレメントの内容を構成するゼロ個以上の引数をとります。この結果は、XML エレメント・ノードを含む XML シーケンスまたは NULL 値です。

結果のデータ・タイプは XML です。いずれかの *element-content-expression* 引数が NULL の可能性がある場合、結果も NULL になる可能性があります。すべての

element-content-expression 引数値が NULL で NULL ON NULL オプションが有効になっている場合、結果は NULL 値になります。

注

- デフォルトの名前空間を定義する別のエレメントの内容としてコピーされるエレメントを構成する場合、デフォルトの名前空間はコピーされたエレメント内で明示的に宣言解除する必要があります。これは、新規の親エレメントからデフォルトの名前空間を継承した結果として生じる可能性のあるエラーを避けるためです。事前定義名前空間接頭部 (「xs」、「xsi」、「xml」、および「sqlxml」) をその使用時に明示的に宣言する必要もあります。
- **エレメント・ノードの構成**: 結果のエレメント・ノードは以下のように構成されます。
  1. *xmlnamespaces-declaration* は、構成されたエレメントの有効範囲内の名前空間のセットを追加します。それぞれの有効範囲内の名前空間は、名前空間接頭部 (またはデフォルトの名前空間) を、名前空間 URI と関連付けます。有効範囲内の名前空間は、エレメントの有効範囲内の QNames の解釈に使用できる名前空間接頭部のセットを定義します。
  2. xmlattributes-function を指定した場合、それが評価され、結果は属性ノードのシーケンスになります。
  3. それぞれの *element-content-expression* は評価され、結果は以下のようにノードのシーケンスに変換されます。
     - 結果のタイプが XML でない場合は、XML にマッピングされる *element-content-expression* の結果が内容である、XML テキスト・ノードに変換されます。このマッピングは、SQL データ値から XML データ値へのマッピングの規則に従います (『データ・タイプ間のキャスト』の非 XML 値から XML 値へのサポートされるキャストを説明する表を参照)。
     - 結果タイプが XML である場合、一般に結果は項目のシーケンスになります。そのシーケンス内のいくつかの項目は、文書ノードである場合があります。シーケンス内の各文書ノードは、その最上位の子のシーケンスによって置き換えられます。次いで結果のシーケンス内の各ノードに対して、その子と属性を組み込んだノードの新しいディープ・コピーが構成されます。コピーされた各ノードは、新しいノード ID を持ちます。コピーされたエレメントおよび属性ノードは、それぞれのタイプのアノテーションを保持します。シーケンス内で戻される 1 つ以上の原子値の隣接する各シーケンスごとに、新規テキスト・ノードが構成され、隣接値の間に単一ブランク文字が挿入された、ストリングに対する各原子値のキャストの結果が含まれます。内容のシーケンス内の隣接するテキスト・ノードは、ブランクを間に挟まずにその内容を連結して単一のテキスト・ノードにマージされます。連結後に、内容がゼロ長ストリングであるテキスト・ノードは、内容のシーケンスから削除されます。
  4. XML 属性の結果のシーケンス、およびすべての *element-content-expression* 指定の結果のシーケンスは、内容シーケンスと呼ばれる 1 つのシーケンスに連結されます。内容シーケンス内の隣接するテキスト・ノードのすべてのシーケンスは、単一のテキスト・ノードにマージされます。すべての *element-content-expression* 引数が空ストリングである場合、または *element-content-expression* 引数が指定されていない場合、空のエレメントが戻されます。

5. 内容シーケンスには、属性ノードではないノードに続けて属性ノードを含めることはできません (SQLSTATE 10507)。内容シーケンス内の属性ノードは、新規エレメント・ノードの属性になります。これらの属性ノードの複数が同じ名前を持つことはできません (SQLSTATE 10503)。名前空間 URI が、構成されたエレメントの有効範囲内名前空間にない場合、名前空間宣言は、属性ノードの名前で使用されるすべての名前空間に対応して作成されます。
6. 内容シーケンス内のエレメント、テキスト、コメント、および処理命令ノードは、構成されたエレメント・ノードの子になります。
7. 構成されたエレメント・ノードには xs:anyType のタイプ・アノテーションが与えられ、その各属性には xdt:untypedAtomic のタイプ・アノテーションが与えられます。構成されたエレメント・ノードのノード名は、NAME キーワードの後に指定された element-name です。

- **XMLELEMENT 内での名前空間の使用規則**: 名前空間の範囲に関する以下の規則を考慮してください。
  - XMLNAMESPACES declaration で宣言された名前空間は、XMLELEMENT 関数で構成されるエレメント・ノードの有効範囲内名前空間です。エレメント・ノードが直列化される場合、そのそれぞれの有効範囲内名前空間は名前空間属性として直列化されます。ただしこれは、エレメント・ノードの親の有効範囲内名前空間であったり、親エレメントも直列化されていたりするのではない場合に限ります。
  - XMLQUERY または XMLEXISTS が *element-content-expression* にある場合、名前空間は XMLQUERY または XMLEXISTS の XQuery 式の静的に既知の名前空間になります。静的に既知の名前空間は、XQuery 式内の QNames を解決するために使用されます。 XQuery プロローグが、XQuery 式の有効範囲内で、同じ接頭部を持つ名前空間を宣言する場合、プロローグ内で宣言される名前空間は、XMLNAMESPACES 宣言内で宣言された名前空間をオーバーライドします。
  - 構成されたエレメントの属性が *element-content-expression* に由来するものである場合、その名前空間は構成されたエレメントの有効範囲内名前空間としてまだ宣言されていない可能性があり、その場合には、それに対して新規名前空間が作成されます。これが結果として競合になる場合、属性名の接頭部が既に異なる URI に有効範囲内名前空間によりバインド済みであるということです。DB2 はそのような競合の原因にならない接頭部を生成し、属性名で使用されている接頭部は新規接頭部に変更され、名前空間がその新規接頭部に対して作成されます。生成される新規接頭部は、「db2ns-xx」というパターンに従います。ここで x は、A から Z、a から z、0 から 9 の範囲から選択された文字です。以下に例を示します。

```
VALUES XMLELEMENT(
  NAME "c", XMLQUERY(
    'declare namespace ipo="www.ipo.com"; $m/ipo:a/@ipo:b'
      PASSING XMLPARSE(
        DOCUMENT '<tst:a xmlns:tst="www.ipo.com" tst:b="2"/>'
      ) AS "m"
  )
)
```

これは、以下のものを戻します。

```
<c xmlns:tst="www.ipo.com" tst:b="2"/>
```

2 番目の例は、以下のようなものです。

```
VALUES XMLELEMENT(
  NAME "tst:c", XMLNAMESPACES(
    'www.tst.com' AS "tst"
  ),
  XMLQUERY(
    'declare namespace ipo="www.ipo.com"; $m/ipo:a/@ipo:b'
      PASSING XMLPARSE(
        DOCUMENT '<tst:a xmlns:tst="www.ipo.com" tst:b="2"/>'
      ) AS "m"
  )
)
```

これは、以下のものを戻します。

```
<tst:c xmlns:tst="www.tst.com" xmlns:db2ns-a1="www.ipo.com"
  db2ns-a1:b="2"/>
```

例

注: XMLELEMENT は、出力の中にブランク・スペースまたは改行文字を挿入しません。例の出力はすべて、読みやすくするために書式を整えています。

- *例 1*: エレメントを NULL ON NULL オプションを使用して構成します。

```
SELECT E.FIRSTNME, E.LASTNAME, XMLELEMENT(
  NAME "Emp", XMLELEMENT(
    NAME "firstname", E.FIRSTNME
  ),
  XMLELEMENT(
    NAME "lastname", E.LASTNAME
  )
  OPTION NULL ON NULL
)
AS "Result"
FROM EMPLOYEE E
WHERE E.EDLEVEL = 12
```

この照会は、次のような結果を生成します。

```
FIRSTNME     LASTNAME        Emp
JOHN         PARKER          <Emp><firstname>JOHN</firstname>
                               <lastname>PARKER</lastname></Emp>
MAUDE        SETRIGHT        <Emp><firstname>MAUDE</firstname>
                               <lastname>SETRIGHT</lastname></Emp>
MICHELLE     SPRINGER        <Emp><firstname>MICHELLE</firstname>
                               <lastname>SPRINGER</lastname></Emp>
```

- *例 2*: 子エレメントとしてネストしたエレメントのリストを使用してエレメントを生成します。

```
SELECT XMLELEMENT(
  NAME "Department", XMLATTRIBUTES(
    E.WORKDEPT AS "name"
  ),
  XMLAGG(
    XMLELEMENT(
      NAME "emp", E.FIRSTNME
    )
    ORDER BY E.FIRSTNME
  )
)
AS "dept_list"
FROM EMPLOYEE E
WHERE E.WORKDEPT IN ('A00', 'B01')
GROUP BY WORKDEPT
```

この照会は、次のような結果を生成します。

```
dept_list
<Department name="A00">
<emp>CHRISTINE</emp>
<emp>SEAN</emp>
<emp>VINCENZO</emp>
</Department>
<Department name="B01">
<emp>MICHAEL</emp>
</Department>
```

- *例 3*: デフォルトの XML エレメント名前空間を指定し、副選択を使用して、ネストされた XML エレメントを作成します。

```
SELECT XMLELEMENT(
         NAME "root",
         XMLNAMESPACES(DEFAULT 'http://mytest.uri'),
         XMLATTRIBUTES(cid),
            (SELECT
               XMLAGG(
                 XMLELEMENT(
                   NAME "poid", poid
                 )
               )
             FROM purchaseorder
             WHERE purchaseorder.custid = customer.cid
            )
       )
FROM customer
WHERE cid = '1002'
```

このステートメントは、以下のような、ルート・エレメント内でデフォルト・エレメント名前空間が宣言された XML 文書を戻します。

```
<root xmlns="http://mytest.uri" CID="1002">
  <poid>5000</poid>
  <poid>5003</poid>
  <poid>5006</poid>
</root>
```

- *例 4*: XML 名前空間を指定した共通表式を使用します。

共通表式を使用して XML エレメントを構成する際に、このエレメントを同じ SQL ステートメント内の他の場所で使用する場合は、エレメントの構成の一部として名前空間宣言を指定する必要があります。以下のステートメントは、PURCHASEORDER 表を使用して poid エレメントを作成する共通表式と、CUSTOMER 表を使用してルート・エレメントを作成する SELECT ステートメントの両方でデフォルトの XML 名前空間を指定します。

```
WITH tempid(id, elem) AS
  (SELECT custid, XMLELEMENT(NAME "poid",
     XMLNAMESPACES(DEFAULT 'http://mytest.uri'),
       poid)
   FROM purchaseorder )
SELECT XMLELEMENT(NAME "root",
         XMLNAMESPACES(DEFAULT 'http://mytest.uri'),
            XMLATTRIBUTES(cid),
            (SELECT XMLAGG(elem)
             FROM tempid
             WHERE tempid.id = customer.cid )
        )
FROM customer
WHERE cid = '1002'
```

このステートメントは、以下のような、ルート・エレメント内でデフォルト・エレメント名前空間が宣言された XML 文書を戻します。

```
<root xmlns="http://mytest.uri" CID="1002">
  <poid>5000</poid>
  <poid>5003</poid>
  <poid>5006</poid>
</root>
```

以下のステートメントでは、CUSTOMER 表を使用してルート・エレメントを作成する SELECT ステートメントのみでデフォルト・エレメント名前空間が宣言されています。

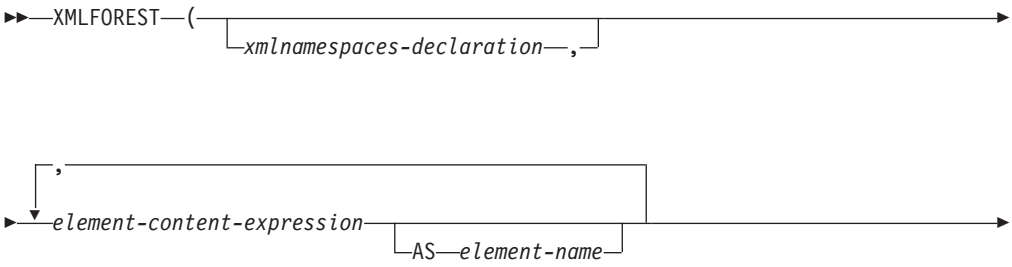
```
WITH tempid(id, elem) AS
    (SELECT custid, XMLELEMENT(NAME "poid", poid)
     FROM purchaseorder )
SELECT XMLELEMENT(NAME "root",
         XMLNAMESPACES(DEFAULT 'http://mytest.uri'),
            XMLATTRIBUTES(cid),
            (SELECT XMLAGG(elem)
             FROM tempid
             WHERE tempid.id = customer.cid )
       )
FROM customer
WHERE cid = '1002'
```

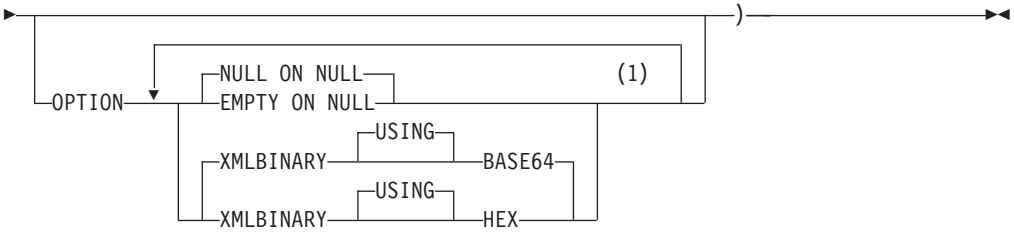
このステートメントは、以下のような、ルート・エレメント内でデフォルト・エレメント名前空間が宣言された XML 文書を戻します。poid エレメントはデフォルト・エレメント名前空間宣言のない共通表式内で作成されるので、poid エレメントのデフォルト・エレメント名前空間は定義されません。XML 文書内で、poid エレメントのデフォルト・エレメント名前空間は空ストリング "" に設定されます。その理由は、poid エレメントのデフォルト・エレメント名前空間は定義されず、poid エレメントはルート・エレメント xmlns="http://mytest.uri" のデフォルト・エレメント名前空間に所属しないからです。

```
<root xmlns="http://mytest.uri" CID="1002">
  <poid xmlns="">5000</poid>
  <poid xmlns="">5003</poid>
  <poid xmlns="">5006</poid>
</root>
```

XMLFOREST

XMLFOREST 関数は、一連の XQuery エレメント・ノードである XML 値を戻します。





注:

1 同じ節を複数回指定することはできません。

スキーマは SYSIBM です。関数名を修飾名で指定することはできません。

xmlnamespaces-declaration
XMLNAMESPACES 宣言の結果である XML 名前空間宣言を指定します。宣言される名前空間は、XMLFOREST 関数の有効範囲内にあります。名前空間は、別の副選択の中で現れるかどうかにかかわらず、XMLFOREST 関数内でネストされているすべての XML 関数に対して適用されます。

xmlnamespaces-declaration を指定しないと、名前空間宣言と構成されたエレメントとの関連付けが行われません。

element-content-expression
生成される XML エレメント・ノードの内容を式によって指定します。データ・タイプ *element-content-expression*を、構造化タイプとすることはできません (SQLSTATE 42884)。式には任意の SQL 式を指定できます。式が単純な列参照でない場合、エレメント名を指定する必要があります。

AS *element-name*
SQL ID として XML エレメント名を指定します。エレメント名は、 XML 修飾名の形式であるかまたは QName (SQLSTATE 42634) でなければなりません。有効な名前の詳細は、「W3C の XML 名前空間仕様」を参照してください。名前が修飾される場合は、名前空間の接頭部をその有効範囲内で宣言する必要があります (SQLSTATE 42635)。 *element-name* が指定されない場合、*element-content-expression* は列名でなければなりません (SQLSTATE 42703)。エレメント名は、列名から QName への完全にエスケープしたマッピングを使用する列名から作成されます。

OPTION
XML エレメントを構成するための追加オプションを指定します。 OPTION 節を指定しない場合、デフォルトは NULL ON NULL XMLBINARY USING BASE64 です。この節は、*element-content-expression* で指定するネストされた XMLELEMENT の呼び出しに影響を与えません。

EMPTY ON NULL or NULL ON NULL
各 *element-content-expression* の値が NULL 値の場合に NULL 値を戻すか、あるいは空エレメントを戻すかを指定します。このオプションはエレメントの内容の NULL 処理にのみ影響し、属性値の NULL 処理には影響を及ぼしません。デフォルトは NULL ON NULL です。

EMPTY ON NULL
それぞれの *element-content-expression* の値が NULL であれば、空のエレメントが戻されます。

NULL ON NULL
それぞれの *element-content-expression* の値が NULL であれば、NULL 値が戻されます。

XMLBINARY USING BASE64 or XMLBINARY USING HEX
バイナリー入力データ、FOR BIT DATA 属性を持つ文字ストリング・データ、またはこれらのタイプのいずれかに基づく特殊タイプの想定エンコードを指定します。エンコードはエレメントの内容または属性値に適用されます。デフォルトは XMLBINARY USING BASE64 です。

XMLBINARY USING BASE64
想定エンコードが Base64 文字である (XML スキーマ・タイプ xs:base64Binary のエンコードに対して定義される) ことを指定します。Base64 エンコードは、US-ASCII の 65 文字のサブセット (10 個の数字、26 個の小文字、26 個の大文字、'+' および '/') を使用して、サブセット内の 1 つの印刷可能文字により、バイナリーまたはビット・データのすべての 6 ビットを表します。文字を普遍的に表せるようにこれらの文字が選ばれています。この方法を使うと、エンコード・データのサイズが元のバイナリーまたはビット・データより 33 % 大きくなります。

XMLBINARY USING HEX
想定エンコードが 16 進文字である (XML スキーマ・タイプ xs:hexBinary のエンコードに対して定義される) ことを指定します。 16 進数エンコードは、各バイト (8 ビット) を 2 つの 16 進文字で表します。この方法を使うと、エンコード・データが元のバイナリーまたはビット・データの 2 倍のサイズになります。

この関数は、任意指定の名前空間宣言のセット、および名前とエレメントの内容を構成する 1 つ以上の引数 (エレメント・ノードが 1 つ以上ある場合) を取ります。結果は、一連の XQuery エレメント・ノードまたは NULL 値を含む XML シーケンスとなります。

結果のデータ・タイプは XML です。いずれかの *element-content-expression* 引数が NULL の可能性がある場合、結果も NULL になる可能性があります。すべての *element-content-expression* 引数値が NULL で NULL ON NULL オプションが有効になっている場合、結果は NULL 値になります。

XMLFOREST 関数は XMLCONCAT および XMLELEMENT を使って表現できます。例えば、以下の 2 つの式は意味的には同じです。

```
XMLFOREST(xmlnamespaces-declaration, arg1 AS name1, arg2 AS name2 ...)
```

```
XMLCONCAT(
  XMLELEMENT(
    NAME name1, xmlnamespaces-declaration, arg1
  ),
  XMLELEMENT(
```

```
    NAME name2, xmlnamespaces-declaration, arg2
  )
  ...
)
```

注

- デフォルトの名前空間を定義する別のエレメントの内容としてコピーされるエレメントを構成する場合、デフォルトの名前空間はコピーされたエレメント内で明示的に宣言解除する必要があります。これは、新規の親エレメントからデフォルトの名前空間を継承した結果として生じる可能性のあるエラーを避けるためです。事前定義名前空間接頭部 (「xs」、「xsi」、「xml」、および「sqlxml」) をその使用時に明示的に宣言する必要もあります。

例

注: XMLFOREST は、出力の中にブランク・スペースまたは改行文字を挿入しません。例の出力はすべて、読みやすくするために書式を整えています。
デフォルトの名前空間を持つエレメントのフォレストを構成します。

```
SELECT EMPNO,
  XMLFOREST(
    XMLNAMESPACES(
      DEFAULT 'http://hr.org', 'http://fed.gov' AS "d"
    ),
    LASTNAME, JOB AS "d:job"
  )
AS "Result"
FROM EMPLOYEE
WHERE EDLEVEL = 12
```

この照会は、次のような結果を生成します。

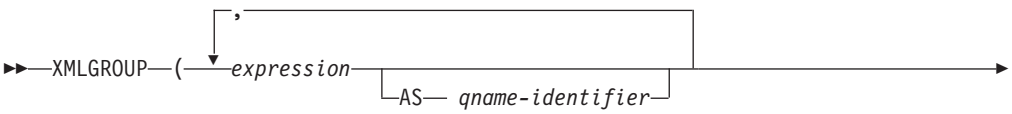
```
EMPNO  Result
000290 <LASTNAME xmlns="http://hr.org" xmlns:d="http://fed.gov">PARKER
  </LASTNAME>
<d:job xmlns="http://hr.org" xmlns:d="http://fed.gov">OPERATOR</d:job>

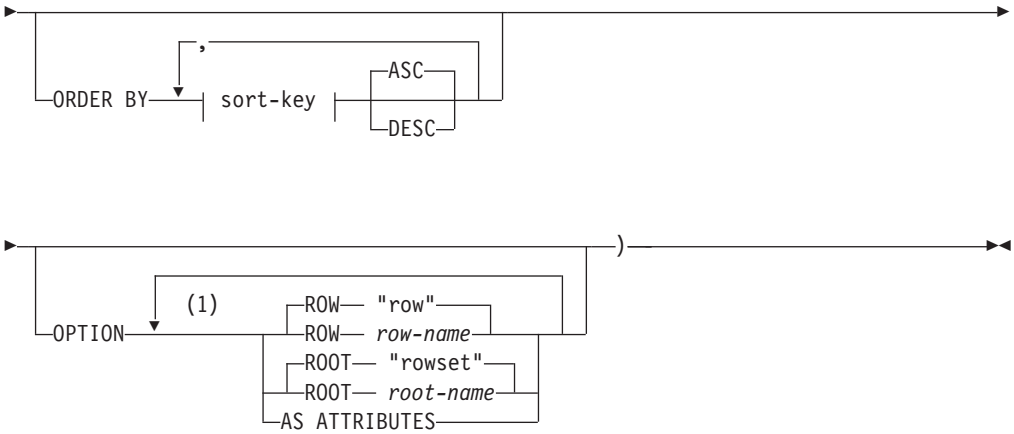
000310 <LASTNAME xmlns="http://hr.org" xmlns:d="http://fed.gov">SETRIGHT
  </LASTNAME>
<d:job xmlns="http://hr.org" xmlns:d="http://fed.gov">OPERATOR</d:job>

200310 <LASTNAME xmlns="http://hr.org" xmlns:d="http://fed.gov">SPRINGER
  </LASTNAME>
<d:job xmlns="http://hr.org" xmlns:d="http://fed.gov">OPERATOR</d:job>
```

XMLGROUP

XMLGROUP 関数は、XQuery 文書ノードを 1 つ持つ XML 値を戻します。これには最上位エレメント・ノードが 1 つ含まれています。これは、各行が行のサブエレメントにマップされる行のグループから単一ルートを持つ XML 文書を戻す集約式です。





注:

1 同じ節を複数回指定することはできません。

スキーマは SYSIBM です。関数名を修飾名で指定することはできません。

expression
生成される各 XML エレメント・ノード (または生成される各属性の値) の内容を式によって指定します。データ・タイプ *expression*を、構造化タイプとすることはできません (SQLSTATE 42884)。式には任意の SQL 式を指定できます。式が単純な列参照でない場合、*qname-identifier* を指定する必要があります。

AS *qname-identifier*
SQL ID として XML エレメント名または属性名を指定します。*qname-identifier* は、XML 修飾名の形式であるかまたは QName でなければなりません (SQLSTATE 42634)。有効な名前の詳細は、「W3C の XML 名前空間仕様」を参照してください。名前が修飾される場合は、名前空間の接頭部をその有効範囲内で宣言する必要があります (SQLSTATE 42635)。 *qname-identifier* が指定されない場合、 *expression* は列名でなければなりません (SQLSTATE 42703)。エレメント名または属性名は、列名から QName への完全にエスケープしたマッピングを使用する列名から作成されます。

OPTION
XML 値を構成するための追加オプションを指定します。 OPTION 節を指定しない場合、デフォルトの動作が適用されます。

ROW *row-name*
各行のマップ先のエレメントの名前を指定します。オプションが指定されない場合のデフォルトのエレメント名は "row" です。

ROOT *root-name*
ルート・エレメント・ノードの名前を指定します。オプションが指定されない場合のデフォルトのルート・エレメント名は "rowset" です。

AS ATTRIBUTES
各式を列名または *qname-identifier* (属性名としての役割を果たす) を使用して属性値にマップすることを指定します。

ORDER BY
集合内の処理対象の、同じグループ化集合に属する行の順序を指定します。ORDER BY 節を省略した場合や、ORDER BY が列データの ORDER BY を特定できない場合、同一のグループ化集合内の行は任意に順序付けられます。

sort-key
ソート・キーは、列名または *sort-key-expression* のどちらでも構いません。ソート・キーが定数の場合、ソート・キーは出力列の位置を (通常の ORDER BY 節におけるように) 参照しませんが、これは単なる定数でしかなく、ソート・キーではないことを意味することに注意してください。

規則

- SELECT 節に ARRAY_AGG 関数が組み込まれている場合、同じ SELECT 節にある ARRAY_AGG、LISTAGG、XMLAGG、および XMLGROUP 関数のすべての呼び出しにおいて、同じ順序を指定するか、または順序を指定しないかのいずれかにする必要があります (SQLSTATE 428GZ)。

注

デフォルトの動作は、結果セットと XML 値の間の単純なマッピングを定義します。以下は、関数の動作に関して当てはまるいくつかの追加注意事項です。

- デフォルトで、各行は "row" という名前の XML エレメントに変換され、各列はネストされたエレメントに変換されます。その際、エレメント名として列名が使用されます。
- ヌル処理の動作は NULL ON NULL です。列の値が NULL の場合、そのマップ先のサブエレメントは空になります。すべての列の値が NULL の場合、行エレメントは生成されません。
- BLOB および FOR BIT DATA データ・タイプのバイナリー・コード化スキームは base64Binary エンコードです。
- デフォルトで、グループの行に対応するエレメントは、"rowset" という名前のルート・エレメントの子です。
- ルート・エレメントの行サブエレメントの順序は、照会結果セットに行が戻される順序と同じです。
- XML の結果を、単一ルートを持つ整形式 XML 文書とするために、文書ノードが暗黙的にルート・エレメントに追加されます。

例

この例は、次の表 T1 に基づいています。そこにはリレーショナル形式で格納された数値データが入っている整数列 C1 および C2 があります。

```
C1          C2
----------- -----------
          1           2
          -           2
          1           -
          -           -

  4 record(s) selected.
```

- *例 1*: 以下の例は、XMLGroup 照会とデフォルトの動作による出力断片が示されています。表を表すために 1 つの最上位エレメントがその中で使用されています。 :

```
SELECT XMLGROUP(C1, C2)FROM T1
```

```
<rowset>
   <row>
      <C1>1</C1>
      <C2>2</C2>
   </row>
   <row>
      <C2>2</C2>
   </row>
   <row>
      <C1>1</C1>
   </row>
</rowset>

  1 record(s) selected.
```

- *例 2*: 以下の例は、XMLGroup 照会と属性を中心としたマッピングによる出力断片を示しています。リレーショナル・データは前例のようにネストされたエレメントとして現れておらず、エレメント属性にマップされています。

```
SELECT XMLGROUP(C1, C2 OPTION AS ATTRIBUTES) FROM T1
```

```
<rowset>
   <row C1="1" C2="2"/>
   <row C2="2"/>
   <row C1="1"/>
</rowset>

  1 record(s) selected.
```

- *例 3*: 以下の例は、XMLGroup 照会とデフォルトの <rowset> ルート・エレメントが <document> によって置き換えられ、デフォルトの <row> エレメントが <entry> によって置き換えられた出力断片を示しています。列 C1 と C2 が <column1> と <column2> エレメントで返され、戻りセットは列 C1 で順序付けられます。

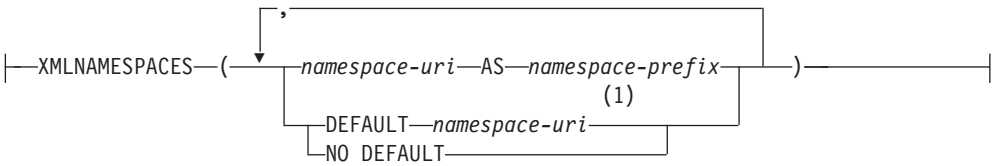
```
SELECT XMLGROUP(
   C1 AS "column1", C2 AS "column2"
   ORDER BY C1 OPTION ROW "entry" ROOT "document")
FROM T1
```

```
<document>
   <entry>
      <column1>1</column1>
      <column2>2</column2>
   </entry>
   <entry>
      <column1>1</column1>
   </entry>
   <entry>
      <column2>2</column2>
    </entry>
</document>
```

XMLNAMESPACES

XMLNAMESPACES 宣言は、引数からネーム・スペース宣言を構成します。

xmlnamespaces-declaration:



注:

1 DEFAULT または NO DEFAULT は、XMLNAMESPACES の引数内で一度しか指定できません。

スキーマは SYSIBM です。宣言名を修飾名で指定することはできません。

この宣言は、XMLELEMENT、XMLFOREST、XMLTABLE などの特定の関数の引数としてのみ使用できます。結果は、NULL 以外の入力値のそれぞれの有効範囲内ネーム・スペースが入っている、1 つ以上の XML ネーム・スペース宣言です。

namespace-uri
SQL 文字ストリング定数に、名前空間の URI を指定します。 *namespace-prefix* と併せて使用する場合には、この文字ストリング定数を空にしてはなりません (SQLSTATE 42815)。

namespace-prefix
名前空間の接頭部を指定します。この接頭部は SQL ID であり、XML NCName の形式でなければなりません (SQLSTATE 42634)。有効な名前の詳細は、「W3C の XML 名前空間仕様」を参照してください。接頭部を `xml` または `xmlns` にすることはできず、名前空間宣言リストの中で固有でなければなりません (SQLSTATE 42635)。

DEFAULT *namespace-uri*
使用するデフォルトの名前空間を、この名前空間宣言の有効範囲内で指定します。 *namespace-uri* はネストされる有効範囲内の別の DEFAULT 宣言または NO DEFAULT 宣言によってオーバーライドされなければ、有効範囲内の非修飾名に適用されます。

NO DEFAULT
この名前空間宣言の有効範囲内のデフォルトの名前空間が使用されないことを指定します。ネストされる有効範囲内の DEFAULT 宣言によってオーバーライドされなければ、有効範囲内のデフォルトの名前空間はありません。

結果のデータ・タイプは XML です。結果は、各指定名前空間の XML 名前空間宣言となります。結果が NULL 値になることはありません。

例

注: XMLNAMESPACES は、出力の中にブランク・スペースまたは改行文字を挿入しません。例の出力はすべて、読みやすくするために書式を整えています。

- *例 1*: adm:employee という名前の XML エレメントと、 XML 属性 adm:department を生成します。このどちらにも、 adm を接頭部として持つ名前空間が関連付けられます。

```
SELECT EMPNO, XMLELEMENT(
  NAME "adm:employee", XMLNAMESPACES(
    'http://www.adm.com' AS "adm"
  ),
  XMLATTRIBUTES(
    WORKDEPT AS "adm:department"
  ),
LASTNAME
)
FROM EMPLOYEE
WHERE JOB = 'ANALYST'
```

この照会は、次のような結果を生成します。

```
000130 <adm:employee xmlns:adm="http://www.adm.com" adm:department="C01">
  QUINTANA</adm:employee>
000140 <adm:employee xmlns:adm="http://www.adm.com" adm:department="C01">
  NICHOLLS</adm:employee>
200140 <adm:employee xmlns:adm="http://www.adm.com" adm:department="C01">
  NATZ</adm:employee>
```

- *例 2*:デフォルトの名前空間に関連付けられた「employee」という名前の XML エレメントと、デフォルトの名前空間を使用するサブエレメント「department」を持ったデフォルトの名前空間を使用しない「job」という名前のサブエレメントを作成します。

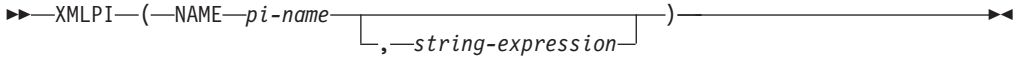
```
SELECT EMP.EMPNO, XMLELEMENT(
  NAME "employee", XMLNAMESPACES(
    DEFAULT 'http://hr.org'
  ),
  EMP.LASTNAME, XMLELEMENT(
    NAME "job", XMLNAMESPACES(
      NO DEFAULT
    ),
    EMP.JOB, XMLELEMENT(
      NAME "department", XMLNAMESPACES(
        DEFAULT 'http://adm.org'
      ),
      EMP.WORKDEPT
    )
  )
)
FROM EMPLOYEE EMP
WHERE EMP.EDLEVEL = 12
```

この照会は、次のような結果を生成します。

```
000290 <employee xmlns="http://hr.org">PARKER<job xmlns="">OPERATOR
  <department xmlns="http://adm.org">E11</department></job></employee>
000310 <employee xmlns="http://hr.org">SETRIGHT<job xmlns="">OPERATOR
  <department xmlns="http://adm.org">E11</department></job></employee>
200310 <employee xmlns="http://hr.org">SPRINGER<job xmlns="">OPERATOR
  <department xmlns="http://adm.org">E11</department></job></employee>
```

XMLPI

XMLPI 関数は、XQuery 処理命令ノードを 1 つ持つ XML 値を戻します。



スキーマは SYSIBM です。関数名を修飾名で指定することはできません。

NAME *pi-name*
処理命令の名前を指定します。この名前は SQL ID であり、XML NCName の形式でなければなりません (SQLSTATE 42634)。有効な名前の詳細は、「W3C の XML 名前空間仕様」を参照してください。どのような大/小文字の組み合わせにしても、この名前を「xml」にすることはできません (SQLSTATE 42634)。

string-expression
文字ストリングである値を戻す式。結果のストリングは UTF-8 に変換されます。これは、XML 1.0 規則で指定されている XML 処理命令の内容に従っていなければなりません (SQLSTATE 2200T)。

- ストリングにサブストリング「?>」を含めてはなりません。このサブストリングは処理命令を終了させるからです。
- ストリングの各文字には任意の Unicode 文字を使用できます。ただし、サロゲート・ブロック X'FFFE' および X'FFFF' は除きます。

結果として生成されるストリングは、構成される処理命令ノードの内容になります。

結果のデータ・タイプは XML です。 *string-expression* の結果が NULL になる可能性がある場合、結果も NULL になる可能性があります。 *string-expression* の結果が NULL であれば、結果も NULL 値になります。 *string-expression* が空ストリングであるかまたは指定されない場合、空の処理命令ノードが戻されます。

例

- *例 1*: XML 処理命令ノードを生成します。

```
SELECT XMLPI(
  NAME "Instruction", 'Push the red button'
)
FROM SYSIBM.SYSDUMMY1
```

この照会は、次のような結果を生成します。

```
<?Instruction Push the red button?>
```

- *例 2*: 空の XML 処理命令ノードを生成します。

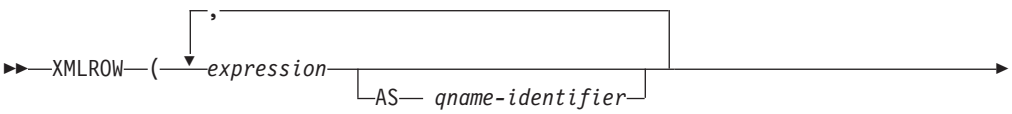
```
SELECT XMLPI(
  NAME "Warning"
)
FROM SYSIBM.SYSDUMMY1
```

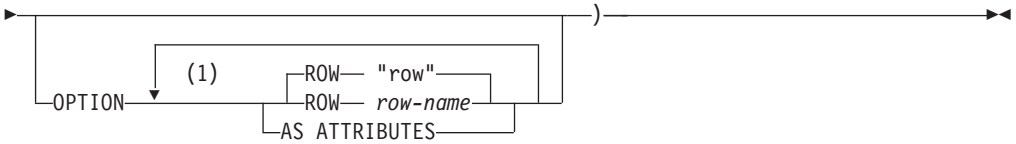
この照会は、次のような結果を生成します。

```
<?Warning ?>
```

XMLROW

XMLROW 関数は、XQuery 文書ノードを 1 つ持つ XML 値を戻します。これには最上位エレメント・ノードが 1 つ含まれています。





注:

1 同じ節を複数回指定することはできません。

スキーマは SYSIBM です。関数名を修飾名で指定することはできません。

expression
生成される各 XML エレメント・ノードの内容を式によって指定します。式のデータ・タイプを、構造化タイプとすることはできません (SQLSTATE 42884)。式には任意の SQL 式を指定できます。式が単純な列参照でない場合、エレメント名を指定する必要があります。

AS *qname-identifier*
SQL ID として XML エレメント名または属性名を指定します。 *qname-identifier* は、XML 修飾名の形式であるかまたは QName でなければなりません (SQLSTATE 42634)。有効な名前の詳細は、「W3C の XML 名前空間仕様」を参照してください。名前が修飾される場合は、名前空間の接頭部をその有効範囲内で宣言する必要があります (SQLSTATE 42635)。 *qname-identifier* が指定されない場合、 *expression* は列名でなければなりません (SQLSTATE 42703)。エレメント名または属性名は、列名から QName への完全にエスケープしたマッピングを使用する列名から作成されます。

OPTION
XML 値を構成するための追加オプションを指定します。 OPTION 節を指定しない場合、デフォルトの動作が適用されます。

AS ATTRIBUTES
各式を列名または *qname-identifier* (属性名としての役割を果たす) を使用して属性値にマップすることを指定します。

ROW *row-name*
各行のマップ先のエレメントの名前を指定します。オプションが指定されない場合のデフォルトのエレメント名は "row" です。

注

デフォルトで、結果セットの各行は次のように XML 値にマップされます。

- 各行は "row" という名前を持つ XML エレメントに変換され、各列はネストされたエレメントに変換されます。その際、エレメント名として列名が使用されます。
- ヌル処理の動作は NULL ON NULL です。列の値が NULL の場合、そのマップ先のサブエレメントは空になります。すべての列の値が NULL の場合、関数によって NULL 値が戻されます。
- BLOB および FOR BIT DATA データ・タイプのバイナリー・コード化スキームは base64Binary エンコードです。

- XML の結果を、単一ルートを持つ整形式 XML 文書とするために、文書ノードが暗黙的に行エレメントに追加されます。

例

列 C1 および C2 を持つ次のような表 T1 があるとします。列にはリレーショナル形式で格納された数値データが入っています。

```
C1          C2
----------- -----------
          1           2
          -           2
          1           -
          -           -

  4 record(s) selected.
```

- *例 1*: 以下の例は、XMLRow 照会とデフォルト動作による出力断片が示されています。表を表すために一連の行エレメントがその中で使用されています。

```
SELECT XMLROW(C1, C2) FROM T1
```

```
<row><C1>1</C1><C2>2</C2></row>
<row><C2>2</C2></row>
<row><C1>1</C1></row>

  4 record(s) selected.
```

- *例 2*: 以下の例は、XMLRow 照会と属性を中心としたマッピングによる出力断片を示しています。リレーショナル・データは前例のようにネストされたエレメントとして現れておらず、エレメント属性にマップされています。

```
SELECT XMLROW(C1, C2 OPTION AS ATTRIBUTES) FROM T1
```

```
<row C1="1" C2="2"/>
<row C2="2"/>
<row C1="1"/>

  4 record(s) selected.
```

- *例 3*: 以下の例は、XMLRow 照会とデフォルトの <row> エレメントが <entry> によって置き換えられた出力断片を示しています。列 C1 と C2 が <column1> と <column2> エレメントで返され、C1 と C2 の合計が <total> エレメント内に返されます。

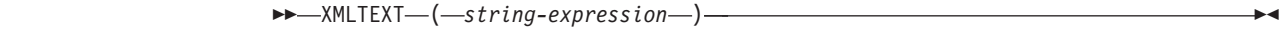
```
SELECT XMLROW(
   C1 AS "column1", C2 AS "column2",
   C1+C2 AS "total" OPTION ROW "entry")
FROM T1
```

```
<entry><column1>1</column1><column2>2</column2><total>3</total></entry>
<entry><column2>2</column2></entry>
<entry><column1>1</column1></entry>

  4 record(s) selected.
```

XMLTEXT

XMLTEXT 関数は、入力引数を内容として持つ、単一の XQuery テキスト・ノードがある XML 値を戻します。



スキーマは SYSIBM です。関数名を修飾名で指定することはできません。

string-expression
値が文字ストリング・タイプ CHAR、VARCHAR、または CLOB を持つ式。

結果のデータ・タイプは XML です。 *string-expression* の結果が NULL になる可能性がある場合、結果も NULL になる可能性があります。入力値が NULL であれば、結果も NULL 値になります。 *string-expression* の結果が空ストリングであれば、結果の値は空テキスト・ノードです。

例

- *例 1*: 単純な XMLTEXT 照会を作成します。

```
VALUES(
  XMLTEXT(
    'The stock symbol for Johnson&Johnson is JNJ.'
  )
)
```

この照会は、以下のシリアライズされた結果を生成します。

```
1
--------------------------------------------------
The stock symbol for Johnson&amp;Johnson is JNJ.
```

「&」記号は、テキスト・ノードがシリアライズされるときには「&」にマップされることに注意してください。

- *例 2*: XMLTEXT を XMLAGG と共に使用して、混合の内容を構成します。表 T の内容が以下のようであるとします。

```
seqno  plaintext
                                                    emphtext
------ ------------------------------------------------------------
-------------
1      This query shows how to construct
 mixed content
2       using XMLAGG and XMLTEXT. Without
 XMLTEXT
3      XMLAGG will not have text nodes to group with other nodes,
 mixed content
       therefore, cannot generate
```

```
   SELECT
XMLELEMENT(
     NAME "para", XMLAGG(
       XMLCONCAT(
         XMLTEXT(
           PLAINTEXT
         ),
         XMLELEMENT(
           NAME "emphasis", EMPHTEXT
         )
       )
       ORDER BY SEQNO
     ), '.'
   ) AS "result"
   FROM T
```

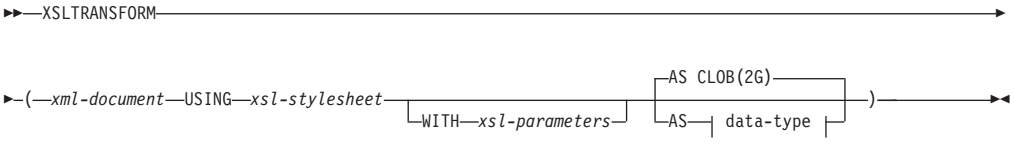
この照会は、次のような結果を生成します。

```
結果
--------------------------------------------------------------------------------
<para>This query shows how to construct <emphasis>mixed content</emphasis>
```

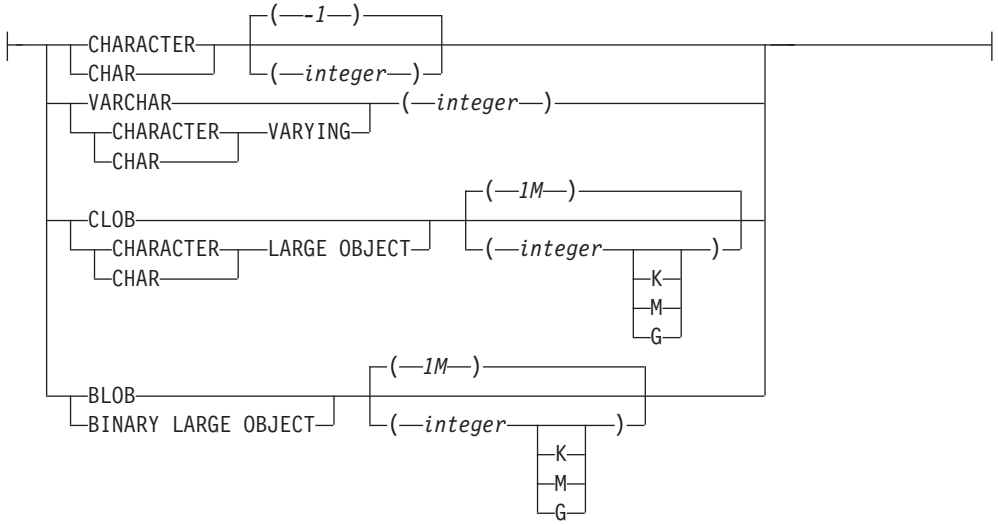
```
using XMLAGG and XMLTEXT. Without <emphasis>XMLTEXT</emphasis>
, XMLAGG
will not have text nodes to group with other nodes, therefore, cannot
generate
<emphasis>mixed content</emphasis>.</para>
```

XSLTRANSFORM

XSLTRANSFORM を使用して XML データを他の形式に変換します。これには、1 つの XML スキーマに準拠する XML 文書を別のスキーマに準拠する文書に変換することも含まれます。



data-type:



スキーマは SYSIBM です。この関数を修飾名で指定することはできません。

XSLTRANSFORM 関数は XML 文書を別のデータ形式に変換します。データは XSLT プロセッサーで可能なあらゆる形式、例えば、XML、HTML、またはプレーン・テキスト (ただし必ずしもこれらに限定されない) などに変換できます。

XSLTRANSFORM で使用されるパスはすべて、データベース・システム内部のパスです。現在のところ、このコマンドを外部ファイル・システムにあるファイルやスタイルシートで直接使用することができません。

xml-document
: データ・タイプ XML、CHAR、VARCHAR、CLOB、または BLOB の整形式 XML 文書を戻す式。これは、*xsl-stylesheet* で指定された XSL スタイルシートを使用して変換される文書です。

  XML 文書は、少なくとも 1 つのルートを持つ整形式文書でなければなりません。

xsl-stylesheet
データ・タイプ XML、CHAR、VARCHAR、CLOB、または BLOB の整形式 XML 文書を戻す式。文書は W3C XSLT バージョン 1.0 勧告に準拠した XSL スタイルシートです。 XQUERY ステートメントまたは xsl:include 宣言を取り込むスタイルシートはサポートされていません。 このスタイルシートは、*xml-document* で指定された値を変換するために適用されます。

xsl-parameters
データ・タイプ XML、CHAR、VARCHAR、CLOB、または BLOB の整形式 XML 文書またはヌルを戻す式。これは、*xsl-stylesheet* で指定された XSL スタイルシートにパラメーター値を提供する文書です。パラメーターの値は、属性またはテキスト・ノードとして指定できます。

パラメーター文書の構文は次のとおりです。

```
<params xmlns="http://www.ibm.com/XSLTransformParameters">
<param name="..." value="..."/>
<param name="...">enter value here</param>
  ...
</params>
```

スタイルシート文書には xsl:param エレメントが含まれている必要があり、パラメーター文書で指定されたものと一致する名前属性値がなければなりません。

AS *data-type*
結果のデータ・タイプを指定します。指定された結果のデータ・タイプの暗黙的または明示的な長さ属性には、変換された出力を収める十分な大きさがなければなりません (SQLSTATE 22001)。デフォルトの結果のデータ・タイプは CLOB(2G) です。

xml-document 引数または *xsl-stylesheet* 引数のいずれかがヌルの場合、結果はヌルになります。

上記文書のいずれかを CHAR、VARCHAR、または CLOB 列に格納する際にコード・ページ変換が発生することがあります。その場合、結果として文字が失われることがあります。

例

この例では、XSLT をフォーマット・エンジンとして使用する方法を示します。セットアップとして、まず以下の 2 つのサンプル文書をデータベースに挿入します。

```
INSERT INTO XML_TAB VALUES
(1,
          '<?xml version="1.0"?>
<students xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation = "/home/steffen/xsd/xslt.xsd">
<student studentID="1" firstName="Steffen" lastName="Siegmund"
    age=â€23â€ university=â€Rostockâ€/>
</students>',
    '<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
            xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:param name="headline"/>
<xsl:param name="showUniversity"/>
<xsl:template match="students">
                 <html>
                    <head/>
                       <body>
                              <h1><xsl:value-of select="$headline"/></h1>
```

```
                              <table border="1">
                                      <th>
                                          <tr>
                                <td width="80">StudentID</td>
                                <td width="200">First Name</td>
                                <td width="200">Last Name</td>
                                <td width="50">Age</td>
                                <xsl:choose>
       <xsl:when test="$showUniversity ='true'">
                                                  <td width="200">University</td>
                                    </xsl:when>
       </xsl:choose>
                             </tr>
                                </th>
                                <xsl:apply-templates/>
                              </table>
                     </body>
               </html>
              </xsl:template>
                   <xsl:template match="student">
                   <tr>
                  <td><xsl:value-of select="@studentID"/></td>
                  <td><xsl:value-of select="@firstName"/></td>
                  <td><xsl:value-of select="@lastName"/></td>
                  <td><xsl:value-of select="@age"/></td>
                  <xsl:choose>
                      <xsl:when test="$showUniversity = 'true' ">
                        <td><xsl:value-of select="@university"/></td>
                      </xsl:when>
                  </xsl:choose>
                         </tr>
                    </xsl:template>
</xsl:stylesheet>'
);
```

次に、XSLTRANSFORM 関数を呼び出して XML データを HTML に変換して表示します。

```
SELECT XSLTRANSFORM (XML_DOC USING XSL_DOC AS CLOB(1M)) FROM XML_TAB;
```

以下が、その結果の文書になります。

```
<html>
<head>
<META http-equiv="Content-Type" content="text/html; charset=UTF-8">
</head>
<body>
<h1></h1>
<table border="1">
<th>
<tr>
<td width="80">StudentID</td>
<td width="200">First Name</td>
<td width="200">Last Name</td>
<td width="50">Age</td>
</tr>
</th>
 <tr>
<td>1</td>
<td>Steffen</td><td>Siegmund</td>
<td>23</td>
</tr>
              </table>
</body>
</html>
```

この例では、出力は HTML で、各パラメーターによってどのような HTML が生成されるか、およびどのようなデータがパラメーターにもたらされるかのみが影響を受けます。このため、ここでは XSLT のエンド・ユーザーの出力用のフォーマット・エンジンとしての使用例を示しています。

使用上の注意

XML 文書の変換は、XSLTRANSFORM 関数、XQuery の更新式、および外部アプリケーション・サーバーによる XSLT 処理など、多数の方式を使用して行えます。DB2 の XML 列に格納された文書の場合、多数の変換を実行するには XSLT よりも XQuery 更新式を使う方がより効率的です。これは、XSLT を使用する場合は常に変換対象の XML 文書の構文解析が必要になるからです。 XSLT を使用して XML 文書の変換を行うことに決めた場合は、その文書の変換をデータベース内で行うか、あるいはアプリケーション・サーバー内で行うかについて、注意深く決定する必要があります。

付録 C. XSR ストアード・プロシージャーおよび XSR コマンド

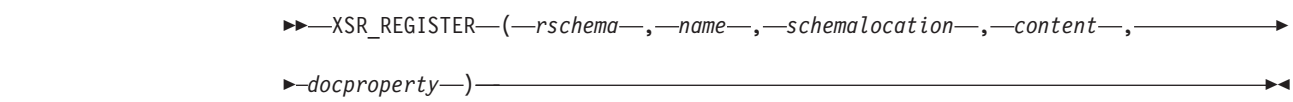
以下のセクションでは、DB2 XSR ストアード・プロシージャーおよび XSR コマンドの構文について説明しています。

これらのストアード・プロシージャーおよびコマンドの使用については、 217 ページの『XSR オブジェクト』を参照してください。

XSR ストアード・プロシージャー

XSR_REGISTER

XSR_REGISTER プロシージャーは、XML スキーマ登録プロセスの一部として呼び出される最初のプロシージャーです。このプロシージャーは XML スキーマ・リポジトリー (XSR) で XML スキーマを登録します。



スキーマは SYSPROC です。

許可

このプロシージャーの呼び出し元の許可 ID には、少なくとも次のいずれかの権限が必要です。

- DBADM 権限。
- IMPLICIT_SCHEMA データベース権限 (SQL スキーマが存在しない場合)。
- CREATEIN 特権 (SQL スキーマが存在する場合)。

rschema

XML スキーマのための SQL スキーマを指定するタイプ VARCHAR (128) の入出力引数。 SQL スキーマは XSR 内でこの XML スキーマの識別に使用される SQL ID の一部です。 (SQL ID のもう 1 つの部分は name 引数によって与えられます。) この引数には、NULL 値を入れることができます。このことは、CURRENT SCHEMA 特殊レジスターで定義されるように、デフォルトの SQL スキーマが使用されていることを示しています。すべての SQL ID に適用される有効な文字と区切り文字の規則は、この引数にも適用されます。ストリング SYS で始まるリレーショナル・スキーマをこの値に使用しないでください。XSR オブジェクトは、XSR の外に存在するデータベース・オブジェクトとの間で名前の衝突を起こすことはありません。XSR オブジェクトは、XML スキーマ・リポジトリー の外にあるオブジェクトとは違う名前空間で発生するからです。

name

XML スキーマの名前を指定する、タイプ VARCHAR (128) の入力および出力引数。 XML スキーマの完全 SQL ID は、*rschema.name* で、XSR にあるすべ

てのオブジェクト間で固有でなければなりません。この引数は NULL 値を受け入れます。この引数に NULL 値が提供される場合、固有な値が生成され、XSR 内に保存されます。すべての SQL ID に適用される有効な文字と区切り文字の規則は、この引数にも適用されます。

schemalocation
タイプ VARCHAR (1000) の入力引数 (NULL 値を入れることができる)。1 次 XML スキーマ文書のスキーマ位置を示します。この引数は XML スキーマの外部名です。この 1 次文書は XML インスタンス文書内で xsi:schemaLocation 属性を指定して識別することができます。

content
1 次 XML スキーマ文書の内容を含むタイプ BLOB (30M) の入力パラメーター。この引数に NULL 値を入れることはできません。XML スキーマ文書を提供する必要があります。

docproperty
1 次 XML スキーマ文書のプロパティーを示すタイプ BLOB (5M) の入力パラメーター。このパラメーターには NULL 値を入れることができます。そうでない場合、この値は XML 文書です。

例

- *例 1*: 次の例は、コマンド行から XSR_REGISTER プロシージャーを呼び出す方法を示しています。

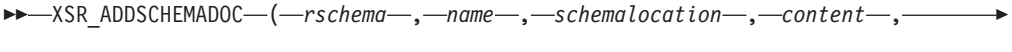
```
CALL SYSPROC.XSR_REGISTER(
  'user1',
  'POschema',
  'http://myPOschema/PO.xsd',
  :content_host_var,
  :docproperty_host_var)
```

- *例 2*: 次の例は、Java アプリケーション・プログラムから XSR_REGISTER プロシージャーを呼び出す方法を示しています。

```
stmt = con.prepareCall("CALL SYSPROC.XSR_REGISTER (?, ?, ?, ?, ?)");
String xsrObjectName = "myschema1";
String xmlSchemaLocation = "po.xsd";
stmt.setNull(1, java.sql.Types.VARCHAR);
stmt.setString(2, xsrObjectName);
stmt.setString(3, xmlSchemaLocation);
stmt.setBinaryStream(4, buffer, (int)length);
stmt.setNull(5, java.sql.Types.BLOB);
stmt.registerOutParameter(1, java.sql.Types.VARCHAR);
stmt.registerOutParameter(2, java.sql.Types.VARCHAR);
stmt.execute();
```

XSR_ADDSCHEMADOC

XML スキーマ・リポジトリー (XSR) 内の各 XML スキーマは 1 つ以上の XML スキーマ文書で構成可能です。 XML スキーマが複数の文書で構成されている場合、XSR_ADDSCHEMADOC プロシージャーを使用して、1 次 XML スキーマ文書以外のすべての XML スキーマを追加します。



スキーマは SYSPROC です。

許可

このプロシージャーの呼び出し元の許可 ID は、カタログ・ビュー SYSCAT.XSROBJECTS に記録されているような XSR オブジェクトの所有者でなければなりません。

rschema
XML スキーマのための SQL スキーマを指定する、タイプ VARCHAR (128) の入力引数。 SQL スキーマは XSR 内でこの XML スキーマの識別に使用される SQL ID の一部です。これは、完了状態に移されます。 (SQL ID のもう 1 つの部分は name 引数によって与えられます。) この引数には、NULL 値を入れることができます。このことは、CURRENT SCHEMA 特殊レジスターで定義されるように、デフォルトの SQL スキーマが使用されていることを示しています。すべての SQL ID に適用される有効な文字と区切り文字の規則は、この引数にも適用されます。 XSR オブジェクトは、XSR の外に存在するデータベース・オブジェクトとの間で名前の衝突を起こすことはありません。XSR オブジェクトは、XML スキーマ・リポジトリー の外にあるオブジェクトとは違う名前空間で発生するからです。

name
XML スキーマの名前を指定する、タイプ VARCHAR (128) の入力引数。 XML スキーマの完全 SQL ID は、*rschema.name* です。 XML スキーマ名は XSR_REGISTER プロシージャーの呼び出しの結果として既に存在していなければなりません。また、XML スキーマ登録はまだ完了することができません。この引数に NULL 値を入れることはできません。すべての SQL ID に適用される有効な文字と区切り文字の規則は、この引数にも適用されます。

schemalocation
タイプ VARCHAR (1000) の入力引数 (NULL 値を入れることができる)。1 次 XML スキーマ文書を追加するこの XML スキーマ文書のスキーマ位置を示します。この引数は XML スキーマの外部名です。この 1 次文書は XML インスタンス文書内で xsi:schemaLocation 属性を指定して識別することができます。

content
追加する XML スキーマ文書の内容を含むタイプ BLOB (30M) の入力パラメーター。この引数に NULL 値を入れることはできません。XML スキーマ文書を提供する必要があります。

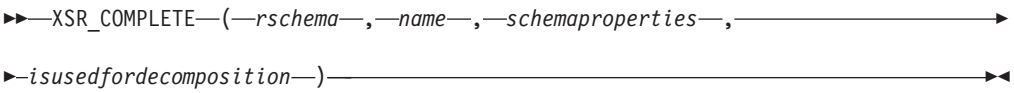
docproperty
追加する XML スキーマ文書のプロパティーを示すタイプ BLOB (5M) の入力パラメーター。このパラメーターには NULL 値を入れることができます。そうでない場合、この値は XML 文書です。

例

```
CALL SYSPROC.XSR_ADDSCHEMADOC(
  'user1',
  'POschema',
  'http://myPOschema/address.xsd',
  :content_host_var,
  0)
```

XSR_COMPLETE

XSR_COMPLETE プロシージャーは、XML スキーマ登録プロセスの一部として呼び出される最終プロシージャーです。このプロシージャーは XML スキーマ・リポジトリー (XSR) で XML スキーマを登録します。 XML スキーマは、このプロシージャーの呼び出しを介してスキーマ登録が完了するまで妥当性検査には使用できません。



スキーマは SYSPROC です。

許可:

このプロシージャーの呼び出し元の許可 ID は、カタログ・ビュー SYSCAT.XSROBJECTS に記録されているような XSR オブジェクトの所有者でなければなりません。

rschema
XML スキーマのための SQL スキーマを指定する、タイプ VARCHAR (128) の入力引数。 SQL スキーマは XSR 内でこの XML スキーマの識別に使用される SQL ID の一部です。これは、完了状態に移されます。 (SQL ID のもう 1 つの部分は name 引数によって与えられます。) この引数には、NULL 値を入れることができます。このことは、CURRENT SCHEMA 特殊レジスターで定義されるように、デフォルトの SQL スキーマが使用されていることを示しています。すべての SQL ID に適用される有効な文字と区切り文字の規則は、この引数にも適用されます。 XSR オブジェクトは、XSR の外に存在するデータベース・オブジェクトとの間で名前の衝突を起こすことはありません。XSR オブジェクトは、XML スキーマ・リポジトリー の外にあるオブジェクトとは違う名前空間で発生するからです。

name
XML スキーマの名前を指定する、タイプ VARCHAR (128) の入力引数。 XML スキーマの完全 SQL ID は、*rschema.name* で、この ID に対して完了チェックが実行されます。 XML スキーマ名は XSR_REGISTER プロシージャーの呼び出しの結果として既に存在していなければなりません。また、XML スキーマ登録はまだ完了することができません。この引数に NULL 値を入れることはできません。すべての SQL ID に適用される有効な文字と区切り文字の規則は、この引数にも適用されます。

schemaproperties
XML スキーマに関連している場合、プロパティーを指定する、タイプ BLOB

(5M) の入力引数。この引数の値は、NULL 値 (関連プロパティーがない場合)、または XML 文書 (XML スキーマのプロパティーを表す) のいずれかです。

isusedfordecomposition
XML スキーマが分解に使用されるかどうかを示す integer タイプの入力パラメーター。 XML スキーマが分解に使用される場合、この値は 1 に設定してください。それ以外の場合はゼロに設定してください。

例

```
CALL SYSPROC.XSR_COMPLETE(
  'user1',
  'POschema',
  :schemaproperty_host_var,
  0)
```

XSR_DTD

XSR_DTD プロシージャーは、文書タイプ宣言 (DTD) を XML スキーマ・リポジトリー (XSR) に登録します。

スキーマは SYSPROC です。

許可

このプロシージャーの呼び出し元の許可 ID には、少なくとも次のいずれかが必要です。

- DBADM 権限。
- IMPLICIT_SCHEMA データベース権限 (SQL スキーマが存在しない場合)。
- CREATEIN 特権 (SQL スキーマが存在する場合)。

rschema
DTD のための SQL スキーマを指定するタイプ VARCHAR (128) の入出力引数。 SQL スキーマは XSR 内でこの DTD の識別に使用される SQL ID の一部です。 (SQL ID のもう 1 つの部分は *name* 引数によって与えられます。) この引数には、NULL 値を入れることができます。このことは、CURRENT SCHEMA 特殊レジスターで定義されるように、デフォルトの SQL スキーマが使用されていることを示しています。すべての SQL ID に適用される有効な文字と区切り文字の規則は、この引数にも適用されます。ストリング SYS で始まるリレーショナル・スキーマをこの値に使用しないでください。 XSR オブジェクトは、XSR の外に存在するデータベース・オブジェクトとの間で名前の衝突を起こすことはありません。XSR オブジェクトは、XML スキーマ・リポジトリー の外にあるオブジェクトとは違う名前空間で発生するからです。

name
DTD の名前を指定する、タイプ VARCHAR (128) の入力および出力引数。 DTD の完全 SQL ID は、*rschema.name* で、XSR にあるすべてのオブジェクト間で固有でなければなりません。この引数は NULL 値を受け入れます。この引

数に NULL 値が提供される場合、固有な値が生成され、XSR 内に保存されます。すべての SQL ID に適用される有効な文字と区切り文字の規則は、この引数にも適用されます。

systemid
DTD のシステム ID を指定する、タイプ VARCHAR (1000) の入力パラメーター。 DTD のシステム ID は、XML インスタンス文書の DOCTYPE 宣言または ENTITY 宣言 (使用されている場合は SYSTEM キーワードが接頭部になる) 中の DTD の URI と一致している必要があります。この引数に NULL 値を入れることはできません。システム ID と公開 ID を一緒に指定できます。

publicid
DTD の公開 ID を指定する、タイプ VARCHAR (1000) の入力パラメーター。DTD の公開 ID は、XML インスタンス文書の DOCTYPE 宣言または ENTITY 宣言 (使用されている場合は PUBLIC キーワードが接頭部になる) 中の DTD の URI と一致している必要があります。この引数は、NULL 値を受け入れ、XML インスタンス文書の DOCTYPE 宣言または ENTITY 宣言中でも指定されている場合のみ使用する必要があります。

content
DTD 文書の内容を含むタイプ BLOB (30M) の入力パラメーター。この引数に NULL 値を入れることはできません。

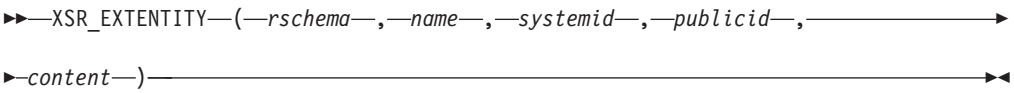
例

システム ID *http://www.test.com/person.dtd* および公開 ID *http://www.test.com/person* によって識別される DTD を登録します。

```
CALL SYSPROC.XSR_DTD ( 'MYDEPT' ,
   'PERSONDTD' ,
   'http://www.test.com/person.dtd' ,
   'http://www.test.com/person',
   :content_host_variable
   )
```

XSR_EXTENTITY

XSR_EXTENTITY プロシージャーは、外部エンティティーを XML スキーマ・リポジトリー (XSR) に登録します。



スキーマは SYSPROC です。

許可

このプロシージャーの呼び出し元の許可 ID には、少なくとも次のいずれかが必要です。

- DBADM 権限。
- IMPLICIT_SCHEMA データベース権限 (SQL スキーマが存在しない場合)。
- CREATEIN 特権 (SQL スキーマが存在する場合)。

rschema
外部エンティティーのための SQL スキーマを指定するタイプ VARCHAR (128) の入出力引数。 SQL スキーマは XSR 内でこの外部エンティティーの識別に使用される SQL ID の一部です。 (SQL ID のもう 1 つの部分は *name* 引数によって与えられます。) この引数には、NULL 値を入れることができます。このことは、CURRENT SCHEMA 特殊レジスターで定義されるように、デフォルトの SQL スキーマが使用されていることを示しています。すべての SQL ID に適用される有効な文字と区切り文字の規則は、この引数にも適用されます。ストリング SYS で始まるリレーショナル・スキーマをこの値に使用しないでください。XSR オブジェクトは、XSR の外に存在するデータベース・オブジェクトとの間で名前の衝突を起こすことはありません。XSR オブジェクトは、XML スキーマ・リポジトリー の外にあるオブジェクトとは違う名前空間で発生するからです。

name
外部エンティティーの名前を指定する、タイプ VARCHAR (128) の入出力引数。外部エンティティーの完全 SQL ID は、*rschema.name* で、XSR にあるすべてのオブジェクト間で固有でなければなりません。この引数は NULL 値を受け入れます。この引数に NULL 値が提供される場合、固有な値が生成され、XSR 内に保存されます。すべての SQL ID に適用される有効な文字と区切り文字の規則は、この引数にも適用されます。

systemid
外部エンティティーのためのシステム ID を指定する、タイプ VARCHAR (1000) の入力パラメーター。外部エンティティーのシステム ID は、ENTITY 宣言 (使用されている場合は SYSTEM キーワードが接頭部になる) 中の外部エンティティーの URI と一致している必要があります。この引数に NULL 値を入れることはできません。システム ID と公開 ID を一緒に指定できます。

publicid
外部エンティティーのための公開 ID を指定する、タイプ VARCHAR (1000) の入力パラメーター。外部エンティティーの公開 ID は、ENTITY 宣言 (使用されている場合は PUBLIC キーワードが接頭部になる) 中の外部エンティティーの URI と一致している必要があります。この引数は、NULL 値を受け入れ、XML インスタンス文書の DOCTYPE 宣言または ENTITY 宣言中でも指定されている場合のみ使用する必要があります。

content
外部エンティティー文書の内容を含むタイプ BLOB (30M) の入力パラメーター。この引数に NULL 値を入れることはできません。

例

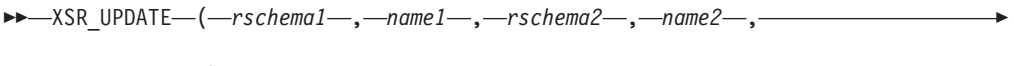
システム ID *http://www.test.com/food/chocolate.txt* および *http://www.test.com/food/cookie.txt* で識別される外部エンティティーを登録します。

```
CALL SYSPROC.XSR_EXTENTITY ( 'FOOD' ,
   'CHOCLATE' ,
   'http://www.test.com/food/chocolate.txt' ,
   NULL ,
   :content_of_chocolate.txt_as_a_host_variable
   )
```

```
CALL SYSPROC.XSR_EXTENTITY ( 'FOOD' ,
   'COOKIE' ,
   'http://www.test.com/food/cookie.txt' ,
   NULL ,
   :content_of_cookie.txt_as_a_host_variable
   )
```

XSR_UPDATE

XSR_UPDATE プロシージャーは、XML スキーマ・リポジトリー (XSR) 内の既存の XML スキーマを発展させるために使用されます。これを使用すると、既存の XML 文書と新しく挿入された XML 文書の両方を妥当性検査できるように、既存の XML スキーマを変更または拡張できます。



スキーマは SYSPROC です。

XSR_UPDATE の引数として指定された元の XML スキーマと新しい XML スキーマは、プロシージャーが呼び出される前に XSR に登録され、かつ完成している必要があります。これらの XML スキーマは互換性がなければなりません。互換性要件の詳細については、『*XML スキーマを発展させるための互換性要件*』を参照してください。

許可

プロシージャーの呼び出し元の許可 ID が持つ特権には、少なくとも以下のいずれかが含まれていなければなりません。

- DBADM 権限。
- カタログ・ビュー SYSCAT.XSROBJECTS および SYSCAT.XSROBJECTCOMPONENTS に対する SELECT 特権、および次の特権セットの 1 つ:
  - SQL スキーマ *rschema1* およびオブジェクト名 *name1* によって指定された XML スキーマの OWNER
  - *rschema1* 引数で指定された SQL スキーマに対する ALTERIN 特権に加え、*dropnewschema* 引数がゼロに等しくない場合には、*rschema2* 引数で指定された SQL スキーマに対する DROPIN 特権。

rschema1
更新対象となる元の XML スキーマのための SQL スキーマを指定する、タイプ VARCHAR (128) の入力引数。 SQL スキーマは XSR 内でこの XML スキーマの識別に使用される SQL ID の一部です。 (SQL ID のもう 1 つの部分は *name1* 引数によって与えられます。) この引数に NULL 値を入れることはできません。すべての SQL ID に適用される有効な文字と区切り文字の規則は、この引数にも適用されます。

name1
更新対象となる元の XML スキーマの名前を指定する、タイプ VARCHAR (128) の入力引数。 XML スキーマの完全 SQL ID は、*rschema1.name1* です。この XML スキーマは、XSR に既に登録され、かつ完成している必要がありま

す。この引数に NULL 値を入れることはできません。すべての SQL ID に適用される有効な文字と区切り文字の規則は、この引数にも適用されます。

rschema2
元の XML スキーマを更新するために使用される新しい XML スキーマのための SQL スキーマを指定する、タイプ VARCHAR (128) の入力引数。 SQL スキーマは XSR 内でこの XML スキーマの識別に使用される SQL ID の一部です。 (SQL ID のもう 1 つの部分は *name2* 引数によって与えられます。) この引数に NULL 値を入れることはできません。すべての SQL ID に適用される有効な文字と区切り文字の規則は、この引数にも適用されます。

name2
元の XML スキーマを更新するために使用される新しい XML スキーマの名前を指定する、タイプ VARCHAR (128) の入力引数。 XML スキーマの完全 SQL ID は、*rschema2.name2* です。この XML スキーマは、XSR に既に登録され、かつ完成している必要があります。この引数に NULL 値を入れることはできません。すべての SQL ID に適用される有効な文字と区切り文字の規則は、この引数にも適用されます。

dropnewschema
元の XML スキーマを更新するために新しい XML スキーマを使用した後にそのスキーマがドロップされるかどうかを示す integer タイプの入力パラメーター。このパラメーターをゼロ以外のいずれかの値に設定した場合、新しい XML スキーマはドロップされます。この引数に NULL 値を入れることはできません。

例

```
CALL SYSPROC.XSR_UPDATE(
  'STORE',
  'PROD',
  'STORE',
  'NEWPROD',
  1)
```

XML スキーマ STORE.PROD の内容は STORE.NEWPROD の内容で更新され、XML スキーマ STORE.NEWPROD はドロップされます。

XSR コマンド

REGISTER XMLSCHEMA

XML スキーマを XML スキーマ・リポジトリー (XSR) に登録します。

許可

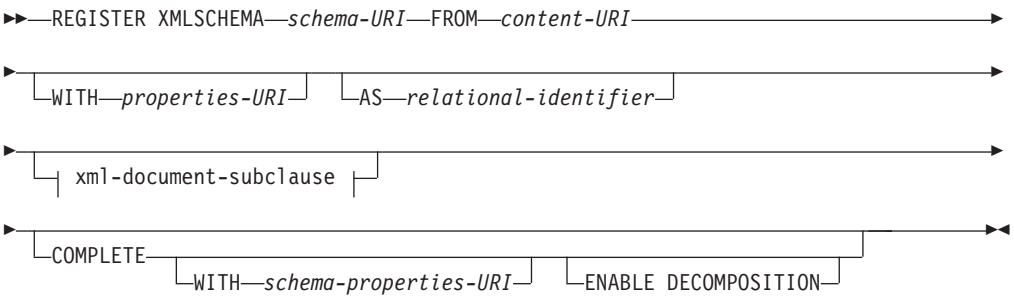
以下の権限のいずれか。

- DBADM
- IMPLICIT_SCHEMA データベース権限 (SQL スキーマが存在しない場合)。
- CREATEIN 特権 (SQL スキーマが存在する場合)。

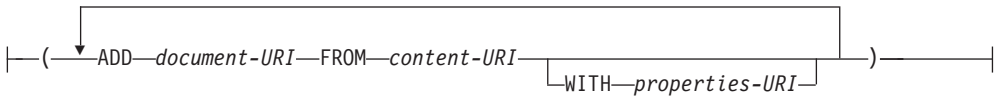
必要な接続

データベース

コマンド構文



xml-document-subclause:



コマンド・パラメーター

schema-URI
登録される XML スキーマの URI を、XML インスタンス文書で参照されるとおりに指定します。

FROM *content-URI*
XML スキーマ文書が置かれている URI を指定します。ファイル・スキーム URI で指定されたローカル・ファイルだけがサポートされます。

WITH *properties-URI*
XML スキーマのプロパティー文書の URI を指定します。ファイル・スキーム URI で指定されたローカル・ファイルだけがサポートされます。

AS *relational-identifier*
登録される XML スキーマを参照するために使用できる名前を指定します。リレーショナル名は 2 つの部分の SQL ID として指定することができます。これは、SQL スキーマと XML スキーマ名から成り、SQLschema.name というフォーマットを持ちます。スキーマが指定されない場合、CURRENT SCHEMA 特殊レジスターで定義されたとおりに、デフォルトのリレーショナル・スキーマが使用されます。名前が提供されない場合、固有値が生成されます。

COMPLETE
これ以上の XML スキーマ文書は追加されないことを示します。これが指定される場合、スキーマの妥当性検査が行われ、エラーが見つからなければ使用できるものとしてマークされます。

WITH *schema-properties-URI*
XML スキーマのプロパティー文書の URI を指定します。ファイル・スキーム URI で指定されたローカル・ファイルだけがサポートされます。

ENABLE DECOMPOSITION
このスキーマが XML 文書の分解のために使用されることを指定します。

ADD *document-URI*
このスキーマに追加される XML スキーマ文書の URI を指定します。この文書は別の XML 文書から参照されることがあるからです。

FROM *content-URI*
XML スキーマ文書が置かれている URI を指定します。ファイル・スキーム URI で指定されたローカル・ファイルだけがサポートされます。

WITH *properties-URI*
XML スキーマのプロパティー文書の URI を指定します。ファイル・スキーム URI で指定されたローカル・ファイルだけがサポートされます。

例

```
REGISTER XMLSCHEMA 'http://myPOschema/PO.xsd'
FROM 'file:///c:/TEMP/PO.xsd'
WITH 'file:///c:/TEMP/schemaProp.xml'
AS user1.POschema
```

使用上の注意

- XML スキーマ文書を参照し、妥当性検査およびアノテーションのために使用できるようになるには、その前にまず XSR に登録する必要があります。このコマンドは、基本 XML スキーマ文書を登録することにより、XML スキーマ登録プロセスの最初のステップを実行します。 XML スキーマ登録プロセスの最終ステップでは、**COMPLETE XMLSCHEMA** コマンドが XML スキーマに対して正常に実行される必要があります。あるいは、その他の XML スキーマ文書が組み込まれない場合、**COMPLETE** キーワードを指定して **REGISTER XMLSCHEMA** コマンドを発行し、登録を 1 ステップで完了してください。
- データベースで XML スキーマを登録する際に、XML スキーマのサイズによっては、より大きなアプリケーション・ヒープ (**applheapsz**) が必要になる場合があります。 推奨されるサイズは 1024 ですが、スキーマが大きくなると追加メモリーが必要になります。

ADD XMLSCHEMA DOCUMENT

登録が完了する前に、1 つ以上の XML スキーマ文書を、既存の未完成の XML スキーマに追加します。

許可

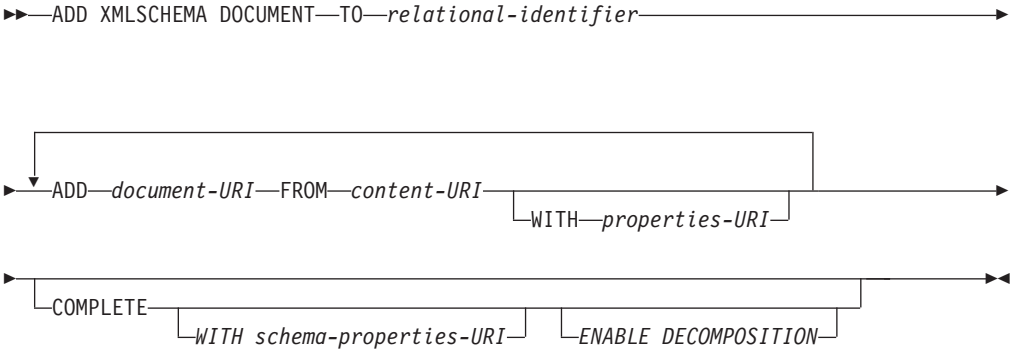
以下の権限が必要です。

- ユーザー ID は、カタログ・ビュー SYSCAT.XSROBJECTS で記録されたとおりに XSR オブジェクトの所有者でなければなりません。

必要な接続

データベース

コマンド構文



説明

TO *relational-identifier*
追加のスキーマ文書が追加される、登録済みであっても未完成の XML スキーマのリレーショナル名を指定します。

ADD *document-URI*
このスキーマに追加される XML スキーマ文書の Uniform Resource Identifier (URI) を指定します。この文書は別の XML 文書から参照されることがあるからです。

FROM *content-URI*
XML スキーマ文書が置かれている URI を指定します。ファイル・スキーム URI だけがサポートされています。

WITH *properties-URI*
XML スキーマのプロパティー文書の URI を指定します。ファイル・スキーム URI だけがサポートされています。

COMPLETE
これ以上の XML スキーマ文書は追加されないことを示します。これが指定される場合、スキーマの妥当性検査が行われ、エラーが見つからなければ使用できるものとしてマークされます。

WITH *schema-properties-URI*
XML スキーマのプロパティー文書の URI を指定します。ファイル・スキーム URI だけがサポートされています。

ENABLE DECOMPOSITION
このスキーマが XML 文書の分解のために使用されることを指定します。

例

```
ADD XMLSCHEMA DOCUMENT TO JOHNDOE.PRODSCHEMA
   ADD 'http://myPOschema/address.xsd'
   FROM 'file:///c:/TEMP/address.xsd'
```

COMPLETE XMLSCHEMA

XML スキーマを XML スキーマ・リポジトリー (XSR) に登録するプロセスを完了します。

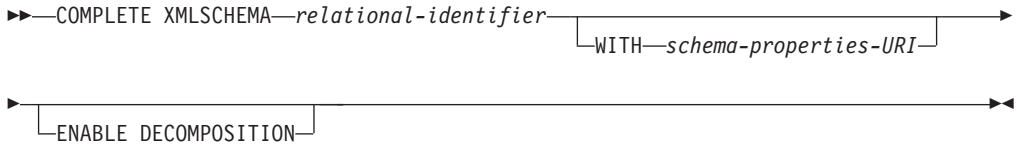
許可

- ユーザー ID は、カタログ・ビュー SYSCAT.XSROBJECTS で記録されたとおりに XSR オブジェクトの所有者でなければなりません。

必要な接続

データベース

コマンド構文



説明

relational-identifier
以前に **REGISTER XMLSCHEMA** コマンドで登録された XML スキーマのリレーショナル名を指定します。リレーショナル名は 2 つの部分の SQL ID として指定することができます。これは、SQL スキーマと XML スキーマ名から成り、*SQLschema.name* というフォーマットを持ちます。スキーマが指定されない場合、CURRENT SCHEMA 特殊レジスターで定義されたとおりに、デフォルト SQL スキーマが使用されます。

WITH *schema-properties-URI*
XML スキーマのプロパティー文書の Uniform Resource Identifier (URI) を指定します。ファイル・スキーム URI で指定されたローカル・ファイルだけがサポートされます。スキーマ・プロパティー文書は、XML スキーマ登録の完了段階でのみ指定できます。

ENABLE DECOMPOSITION
スキーマを XML インスタンス文書の分解に使用できることを示します。

例

```
COMPLETE XMLSCHEMA user1.POschema WITH 'file:///c:/TEMP/schemaProp.xml'
```

使用上の注意

XML スキーマ登録プロセスが完了するまで、XML スキーマを参照したり、妥当性検査またはアノテーションに使用することはできません。このコマンドは、**REGISTER XMLSCHEMA** コマンドで開始された XML スキーマの XML スキーマ登録プロセスを完了します。

REGISTER XSROBJECT

データベース・カタログに XML オブジェクトを登録します。サポートされるオブジェクトは、DTD および外部エンティティーです。

許可

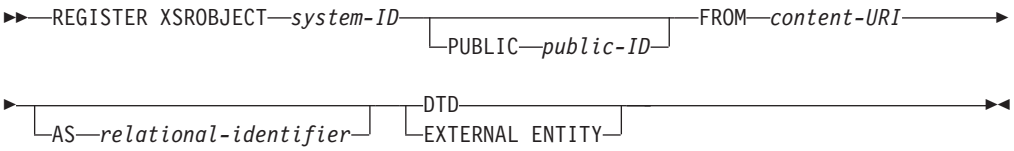
以下の権限のいずれか。

- DBADM
- IMPLICIT_SCHEMA データベース権限 (SQL スキーマが存在しない場合)。
- CREATEIN 特権 (SQL スキーマが存在する場合)。

必要な接続

データベース

コマンド構文



コマンド・パラメーター

system-ID
XML オブジェクト宣言で指定されているシステム ID を指定します。

PUBLIC *public-ID*
XML オブジェクト宣言内のオプションの PUBLIC ID を指定します。

FROM *content-URI*
XML スキーマ文書の内容が置かれている URI を指定します。ファイル・スキーム URI で指定されたローカル・ファイルだけがサポートされます。

AS *relational-identifier*
登録される XML オブジェクトを参照するために使用できる名前を指定します。リレーショナル名は 2 つの部分の SQL ID として指定することができます。これは、ピリオドで区切られたリレーショナル・スキーマと名前から成ります (例えば、"JOHNDOE.EMPLOYEEDTD")。リレーショナル・スキーマが指定されない場合、特殊レジスター CURRENT SCHEMA で定義されているデフォルトのリレーショナル・スキーマが使用されます。名前を指定しない場合、自動的に生成されます。

DTD 登録されるオブジェクトがデータ・タイプ定義文書 (DTD) であることを指定します。

EXTERNAL ENTITY
登録されるオブジェクトが外部エンティティーであることを指定します。

例

1. 以下のサンプル XML 文書は外部エンティティーを参照します。

```
<?xml version="1.0" standalone="no" ?>
<!DOCTYPE copyright [
  <!ELEMENT copyright (#PCDATA)>

]>
<copyright>c</copyright>
```

この文書を正常に XML 列に挿入するには、その前に外部エンティティーを登録する必要があります。以下のコマンドは、エンティティー・コンテンツがローカルの C:¥TEMP に保管されているエンティティーを登録しています。

```
REGISTER XSROBJECT 'http://www.xmlwriter.net/copyright.xml'
   FROM 'c:¥temp¥copyright.xml' EXTERNAL ENTITY
```

2. 以下の XML 文書フラグメントは DTD を参照します。

```
<!--inform the XML processor
  that an external DTD is referenced-->
<?xml version="1.0" standalone="no" ?>

<!--define the location of the
  external DTD using a relative URL address-->
<!DOCTYPE document SYSTEM "http://www.xmlwriter.net/subjects.dtd">

<document>
  <title>Subjects available in Mechanical Engineering.</title>
  <subjectID>2.303</subjectID>
    <subjectname>Fluid Mechanics</subjectname>
 ...
```

この文書を正常に XML 列に挿入するには、その前に DTD を登録する必要があります。以下のコマンドは、DTD 定義がローカルの C:¥TEMP に保管されており、DTD に関連付けるリレーショナル ID が "TEST.SUBJECTS" である DTD を登録します。

```
REGISTER XSROBJECT 'http://www.xmlwriter.net/subjects.dtd'
   FROM 'file:///c:/temp/subjects.dtd' AS TEST.SUBJECTS DTD
```

3. 以下のサンプル XML 文書は public 外部エンティティーを参照します。

```
<?xml version="1.0" standalone="no" ?>
<!DOCTYPE copyright [
  <!ELEMENT copyright (#PCDATA)>

]>
<copyright>c</copyright>
```

この文書を正常に XML 列に挿入するには、その前に public 外部エンティティーを登録する必要があります。以下のコマンドは、エンティティー・コンテンツがローカルの C:¥TEMP に保管されているエンティティーを登録しています。

```
REGISTER XSROBJECT 'http://www.w3.org/xmlspec/copyright.xml'
   PUBLIC '-//W3C//TEXT copyright//EN' FROM 'file:///c:/temp/copyright.xml'
   EXTERNAL ENTITY
```

UPDATE XMLSCHEMA

任意の XML スキーマを、XML スキーマ・リポジトリー (XSR) にある別のスキーマに更新します。

許可

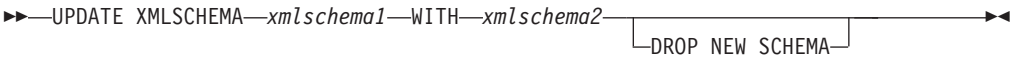
以下の権限のいずれか。

- DBADM
- カタログ・ビュー SYSCAT.XSROBJECTS および SYSCAT.XSROBJECTCOMPONENTS に対する SELECT 特権、および次の特権セットの 1 つ:
  - 更新される XML スキーマに対する ALTERIN 特権、および新しい XML スキーマに対する DROPIN 特権 (**DROP NEW SCHEMA** オプションが指定されている場合)。
  - xmlschema1 によって指定される XML スキーマの OWNER。

必要な接続

データベース

コマンド構文



コマンド・パラメーター

UPDATE XMLSCHEMA *xmlschema1*
更新されるオリジナルの XML スキーマの SQL ID を指定します。

WITH *xmlschema2*
オリジナルの XML スキーマを更新するのに使用される新 XML スキーマの SQL ID を指定します。

DROP NEW SCHEMA
新しい XML スキーマがオリジナルの XML スキーマを更新するために使用された後、ドロップされるよう指定します。

例

```
UPDATE XMLSCHEMA JOHNDOE.OLDPROD
WITH JOHNDOE.NEWPROD
DROP NEW SCHEMA
```

XML スキーマ JOHNDOE.OLDPROD の内容が JOHNDOE.NEWPROD の内容に更新され、XML スキーマ JOHNDOE.NEWPROD はドロップされます。

使用上の注意

- オリジナルの XML スキーマと新しい XML スキーマには互換性がなければなりません。この互換性の要件について詳しくは、『XML スキーマの展開のための互換性要件』を参照してください。
- XML スキーマを更新するためには、その前にオリジナルのスキーマと新しいスキーマの両方が XML スキーマ・リポジトリー (XSR) に登録されている必要があります。

付録 D. DB2 技術情報の概説

DB2 技術情報は、さまざまな方法でアクセスすることが可能な、各種形式で入手できます。

DB2 技術情報は、以下のツールと方法を介して利用できます。

- DB2インフォメーション・センター
  - トピック (タスク、概念、およびリファレンス・トピック)
  - サンプル・プログラム
  - チュートリアル
- DB2 資料
  - PDF ファイル (ダウンロード可能)
  - PDF ファイル (DB2 PDF DVD に含まれる)
  - 印刷資料
- コマンド行ヘルプ
  - コマンド・ヘルプ
  - メッセージ・ヘルプ

注: DB2 インフォメーション・センターのトピックは、PDF やハードコピー資料よりも頻繁に更新されます。最新の情報を入手するには、資料の更新が発行されたときにそれをインストールするか、ibm.com にある DB2 インフォメーション・センターを参照してください。

技術資料、ホワイト・ペーパー、IBM Redbooks® 資料などのその他の DB2 技術情報には、オンライン (ibm.com) でアクセスできます。 DB2 Information Management ソフトウェア・ライブラリー・サイト (http://www.ibm.com/software/data/sw-library/) にアクセスしてください。

資料についてのフィードバック

DB2 の資料についてのお客様からの貴重なご意見をお待ちしています。DB2 の資料を改善するための提案については、db2docs@ca.ibm.com まで E メールを送信してください。DB2 の資料チームは、お客様からのフィードバックすべてに目を通しますが、直接お客様に返答することはありません。お客様が関心をお持ちの内容について、可能な限り具体的な例を提供してください。特定のトピックまたはヘルプ・ファイルについてのフィードバックを提供する場合は、そのトピック・タイトルおよび URL を含めてください。

DB2 お客様サポートに連絡する場合には、この E メール・アドレスを使用しないでください。資料を参照しても、DB2 の技術的な問題が解決しない場合は、お近くの IBM サービス・センターにお問い合わせください。

DB2 テクニカル・ライブラリー (ハードコピーまたは PDF 形式)

以下の表は、IBM Publications Center (www.ibm.com/e-business/linkweb/publications/servlet/pbi.wss) から利用できる DB2 ライブラリーについて説明しています。英語および翻訳された DB2 バージョン 10.1 のマニュアル (PDF 形式) は、www.ibm.com/support/docview.wss?rs=71&uid=swg2700947 からダウンロードできます。

この表には印刷資料が入手可能かどうかが示されていますが、国または地域によっては入手できない場合があります。

資料番号は、資料が更新される度に大きくなります。資料を参照する際は、以下にリストされている最新版であることを確認してください。

注: *DB2 インフォメーション・センター*は、PDF やハードコピー資料よりも頻繁に更新されます。

表 76. DB2 の技術情報

資料名	資料番号	印刷資料が入手可能かどうか	最終更新
管理 API リファレンス	SA88-4671-00	入手可能	2012 年 4 月
管理ルーチンおよびビュー	SA88-4672-00	入手不可	2012 年 4 月
コール・レベル・インターフェース ガイドおよびリファレンス 第 1 巻	SA88-4676-00	入手可能	2012 年 4 月
コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻	SA88-4677-00	入手可能	2012 年 4 月
コマンド・リファレンス	SA88-4673-00	入手可能	2012 年 4 月
データベース: 管理の概念および構成リファレンス	SA88-4662-00	入手可能	2012 年 4 月
データ移動ユーティリティー ガイドおよびリファレンス	SA88-4693-00	入手可能	2012 年 4 月
データベースのモニタリング ガイドおよびリファレンス	SA88-4663-00	入手可能	2012 年 4 月
データ・リカバリーと高可用性 ガイドおよびリファレンス	SA88-4694-00	入手可能	2012 年 4 月
データベース・セキュリティー・ガイド	SA88-4695-00	入手可能	2012 年 4 月

表 76. DB2 の技術情報 (続き)

資料名	資料番号	印刷資料が入手可能かどうか	最終更新
DB2 ワークロード管理ガイドおよびリファレンス	SA88-4685-00	入手可能	2012 年 4 月
ADO.NET および OLE DB アプリケーションの開発	SA88-4665-00	入手可能	2012 年 4 月
組み込み SQL アプリケーションの開発	SA88-4666-00	入手可能	2012 年 4 月
Java アプリケーションの開発	SA88-4669-00	入手可能	2012 年 4 月
Perl、PHP、Python および Ruby on Rails アプリケーションの開発	SA88-4670-00	入手不可	2012 年 4 月
SQL および外部ルーチンの開発	SA88-4667-00	入手可能	2012 年 4 月
データベース・アプリケーション開発の基礎	GI88-4279-00	入手可能	2012 年 4 月
DB2 インストールおよび管理 概説 (Linux および Windows 版)	GI88-4280-00	入手可能	2012 年 4 月
グローバリゼーション・ガイド	SA88-4696-00	入手可能	2012 年 4 月
DB2 サーバー機能 インストール	GA88-4679-00	入手可能	2012 年 4 月
IBM データ・サーバー・クライアント機能 インストール	GA88-4680-00	入手不可	2012 年 4 月
メッセージ・リファレンス 第 1 巻	SA88-4688-00	入手不可	2012 年 4 月
メッセージ・リファレンス 第 2 巻	SA88-4689-00	入手不可	2012 年 4 月
Net Search Extender 管理およびユーザーズ・ガイド	SA88-4691-00	入手不可	2012 年 4 月
パーティションおよびクラスタリングのガイド	SA88-4697-00	入手可能	2012 年 4 月
pureXML ガイド	SA88-4686-00	入手可能	2012 年 4 月
Spatial Extender ユーザーズ・ガイドおよびリファレンス	SA88-4690-00	入手不可	2012 年 4 月

表 76. DB2 の技術情報 (続き)

資料名	資料番号	印刷資料が入手可能かどうか	最終更新
SQL プロシージャー言語: アプリケーションのイネーブルメントおよびサポート	SA88-4668-00	入手可能	2012 年 4 月
SQL リファレンス 第 1 巻	SA88-4674-00	入手可能	2012 年 4 月
SQL リファレンス 第 2 巻	SA88-4675-00	入手可能	2012 年 4 月
Text Search ガイド	SA88-4692-00	入手可能	2012 年 4 月
問題判別およびデータベース・パフォーマンスのチューニング	SA88-4664-00	入手可能	2012 年 4 月
DB2 バージョン 10.1 へのアップグレード	SA88-4678-00	入手可能	2012 年 4 月
DB2 バージョン 10.1 の新機能	SA88-4684-00	入手可能	2012 年 4 月
XQuery リファレンス	SA88-4687-00	入手不可	2012 年 4 月

表 77. DB2 Connect 固有の技術情報

資料名	資料番号	印刷資料が入手可能かどうか	最終更新
DB2 Connect Personal Edition インストールおよび構成	SA88-4681-00	入手可能	2012 年 4 月
DB2 Connect サーバー機能 インストールおよび構成	SA88-4682-00	入手可能	2012 年 4 月
DB2 Connect ユーザーズ・ガイド	SA88-4683-00	入手可能	2012 年 4 月

コマンド行プロセッサーから SQL 状態ヘルプを表示する

DB2 製品は、SQL ステートメントの結果の原因になったと考えられる条件の SQLSTATE 値を戻します。 SQLSTATE ヘルプは、SQL 状態および SQL 状態クラス・コードの意味を説明します。

手順

SQL 状態ヘルプを開始するには、コマンド行プロセッサーを開いて以下のように入力します。

```
? sqlstate または ? class code
```

ここで、*sqlstate* は有効な 5 桁の SQL 状態を、*class code* は SQL 状態の最初の 2 桁を表します。

例えば、? 08003 を指定すると SQL 状態 08003 のヘルプが表示され、 ? 08 を指定するとクラス・コード 08 のヘルプが表示されます。

異なるバージョンの DB2 インフォメーション・センターへのアクセス

他のバージョンの DB2 製品の資料は、ibm.com® のそれぞれのインフォメーション・センターにあります。

このタスクについて

DB2 バージョン 10.1 のトピックを扱っている *DB2 インフォメーション・センター* の URL は、http://publib.boulder.ibm.com/infocenter/db2luw/v10r1 です。

DB2 バージョン 9.8 のトピックを扱っている *DB2 インフォメーション・センター* の URL は、http://publib.boulder.ibm.com/infocenter/db2luw/v9r8/ です。

DB2 バージョン 9.7 のトピックを扱っている *DB2 インフォメーション・センター* の URL は、http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/ です。

DB2 バージョン 9.5 のトピックを扱っている *DB2 インフォメーション・センター* の URL は、http://publib.boulder.ibm.com/infocenter/db2luw/v9r5 です。

DB2 バージョン 9.1 のトピックを扱っている *DB2 インフォメーション・センター* の URL は、http://publib.boulder.ibm.com/infocenter/db2luw/v9/ です。

DB2 バージョン 8 のトピックについては、*DB2 インフォメーション・センター* の URL (http://publib.boulder.ibm.com/infocenter/db2luw/v8/) を参照してください。

コンピューターまたはイントラネット・サーバーにインストールされた DB2 インフォメーション・センターの更新

ローカルにインストールした DB2 インフォメーション・センターは、定期的に更新する必要があります。

始める前に

DB2 バージョン 10.1 インフォメーション・センターが既にインストール済みである必要があります。詳しくは、「*DB2 サーバー機能 インストール*」の『DB2 セットアップ・ウィザードによる DB2 インフォメーション・センターのインストール』のトピックを参照してください。インフォメーション・センターのインストールに適用されるすべての前提条件と制約事項は、インフォメーション・センターの更新にも適用されます。

このタスクについて

既存の DB2 インフォメーション・センターは、自動で更新することも手動で更新することもできます。

- 自動更新は、既存のインフォメーション・センターのフィーチャーと言語を更新します。自動更新を使用すると、手動更新と比べて、更新中にインフォメーショ

ン・センターが使用できなくなる時間が短くなるというメリットがあります。さらに、自動更新は、定期的に実行する他のバッチ・ジョブの一部として実行されるように設定することができます。

- 手動更新は、既存のインフォメーション・センターのフィーチャーと言語の更新に使用できます。自動更新は更新処理中のダウン時間を減らすことができますが、フィーチャーまたは言語を追加する場合は手動処理を使用する必要があります。例えば、ローカルのインフォメーション・センターが最初は英語とフランス語でインストールされており、その後ドイツ語もインストールすることにした場合、手動更新でドイツ語をインストールし、同時に、既存のインフォメーション・センターのフィーチャーおよび言語を更新できます。しかし、手動更新ではインフォメーション・センターを手動で停止、更新、再始動する必要があります。更新処理の間はずっと、インフォメーション・センターは使用できなくなります。自動更新処理では、インフォメーション・センターは、更新を行った後に、インフォメーション・センターを再始動するための停止が発生するだけで済みます。

このトピックでは、自動更新のプロセスを詳しく説明しています。手動更新の手順については、『コンピューターまたはイントラネット・サーバーにインストールされた DB2 インフォメーション・センターの手動更新』のトピックを参照してください。

手順

コンピューターまたはイントラネット・サーバーにインストールされている DB2 インフォメーション・センターを自動更新する手順を以下に示します。

1. Linux オペレーティング・システムの場合、次のようにします。
   a. インフォメーション・センターがインストールされているパスにナビゲートします。デフォルトでは、DB2 インフォメーション・センターは、`/opt/ibm/db2ic/V10.1` ディレクトリーにインストールされています。
   b. インストール・ディレクトリーから `doc/bin` ディレクトリーにナビゲートします。
   c. 次のように `update-ic` スクリプトを実行します。

      ```
      update-ic
      ```
2. Windows オペレーティング・システムの場合、次のようにします。
   a. コマンド・ウィンドウを開きます。
   b. インフォメーション・センターがインストールされているパスにナビゲートします。デフォルトでは、DB2 インフォメーション・センターは、`<Program Files>¥IBM¥DB2 Information Center¥バージョン 10.1` ディレクトリーにインストールされています (`<Program Files>` は「Program Files」ディレクトリーのロケーション)。
   c. インストール・ディレクトリーから `doc¥bin` ディレクトリーにナビゲートします。
   d. 次のように `update-ic.bat` ファイルを実行します。

      ```
      update-ic.bat
      ```

タスクの結果

DB2 インフォメーション・センターが自動的に再始動します。更新が入手可能な場合、インフォメーション・センターに、更新された新しいトピックが表示されます。インフォメーション・センターの更新が入手可能でなかった場合、メッセージがログに追加されます。ログ・ファイルは、doc¥eclipse¥configuration ディレクトリーにあります。ログ・ファイル名はランダムに生成された名前です。例えば、1239053440785.log のようになります。

コンピューターまたはイントラネット・サーバーにインストールされた DB2 インフォメーション・センターの手動更新

DB2 インフォメーション・センターをローカルにインストールしている場合は、IBM から資料の更新を入手してインストールすることができます。

このタスクについて

ローカルにインストールされた *DB2 インフォメーション・センター* を手動で更新するには、以下のことを行う必要があります。

1. コンピューター上の *DB2 インフォメーション・センター* を停止し、インフォメーション・センターをスタンドアロン・モードで再始動します。インフォメーション・センターをスタンドアロン・モードで実行すると、ネットワーク上の他のユーザーがそのインフォメーション・センターにアクセスできなくなります。これで、更新を適用できるようになります。DB2 インフォメーション・センターのワークステーション・バージョンは、常にスタンドアロン・モードで実行されます。を参照してください。
2. 「更新」機能を使用することにより、どんな更新が利用できるかを確認します。インストールしなければならない更新がある場合は、「更新」機能を使用してそれを入手およびインストールできます。

   注: ご使用の環境において、インターネットに接続されていないマシンに *DB2 インフォメーション・センター* の更新をインストールする必要がある場合、インターネットに接続されていて *DB2 インフォメーション・センター* がインストールされているマシンを使用して、更新サイトをローカル・ファイル・システムにミラーリングしてください。ネットワーク上の多数のユーザーが資料の更新をインストールする場合にも、更新サイトをローカルにミラーリングして、更新サイト用のプロキシーを作成することにより、個々のユーザーが更新を実行するのに要する時間を短縮できます。
   更新パッケージが入手可能な場合、「更新」機能を使用してパッケージを入手します。ただし、「更新」機能は、スタンドアロン・モードでのみ使用できます。
3. スタンドアロンのインフォメーション・センターを停止し、コンピューター上の *DB2 インフォメーション・センター* を再開します。

注: Windows 2008、Windows Vista (およびそれ以上) では、このセクションの後の部分でリストされているコマンドは管理者として実行する必要があります。完全な管理者特権でコマンド・プロンプトまたはグラフィカル・ツールを開くには、ショートカットを右クリックしてから、**「管理者として実行」**を選択します。

手順

コンピューターまたはイントラネット・サーバーにインストール済みの *DB2* インフォメーション・センター を更新するには、以下のようにします。

1. *DB2* インフォメーション・センター を停止します。
   - Windows では、「**スタート**」 > 「**コントロール パネル**」 > 「**管理ツール**」 > 「**サービス**」をクリックします。次に、「**DB2 インフォメーション・センター**」サービスを右クリックして「**停止**」を選択します。
   - Linux では、以下のコマンドを入力します。

     ```
     /etc/init.d/db2icdv10 stop
     ```
2. インフォメーション・センターをスタンドアロン・モードで開始します。
   - Windows の場合:
     a. コマンド・ウィンドウを開きます。
     b. インフォメーション・センターがインストールされているパスにナビゲートします。デフォルトでは、*DB2* インフォメーション・センター は、*Program_Files*¥IBM¥DB2 Information Center¥バージョン 10.1 ディレクトリーにインストールされています (*Program_Files* は Program Files ディレクトリーのロケーション)。
     c. インストール・ディレクトリーから doc¥bin ディレクトリーにナビゲートします。
     d. 次のように help_start.bat ファイルを実行します。

        ```
        help_start.bat
        ```
   - Linux の場合:
     a. インフォメーション・センターがインストールされているパスにナビゲートします。デフォルトでは、*DB2* インフォメーション・センター は、/opt/ibm/db2ic/V10.1 ディレクトリーにインストールされています。
     b. インストール・ディレクトリーから doc/bin ディレクトリーにナビゲートします。
     c. 次のように help_start スクリプトを実行します。

        ```
        help_start
        ```

   システムのデフォルト Web ブラウザーが開き、スタンドアロンのインフォメーション・センターが表示されます。
3. 「**更新**」ボタン () をクリックします。(ブラウザーで JavaScript が有効になっている必要があります。) インフォメーション・センターの右側のパネルで、「**更新の検索**」をクリックします。 既存の文書に対する更新のリストが表示されます。
4. インストール・プロセスを開始するには、インストールする更新をチェックして選択し、「**更新のインストール**」をクリックします。
5. インストール・プロセスが完了したら、「**完了**」をクリックします。
6. 次のようにして、スタンドアロンのインフォメーション・センターを停止します。
   - Windows の場合は、インストール・ディレクトリーの doc¥bin ディレクトリーにナビゲートしてから、次のように help_end.bat ファイルを実行します。

```
help_end.bat
```

注: help_end バッチ・ファイルには、help_start バッチ・ファイルを使用して開始したプロセスを安全に停止するのに必要なコマンドが含まれています。help_start.bat は、Ctrl-C や他の方法を使用して停止しないでください。

- Linux の場合は、インストール・ディレクトリーの doc/bin ディレクトリーにナビゲートしてから、次のように help_end スクリプトを実行します。

```
help_end
```

注: help_end スクリプトには、help_start スクリプトを使用して開始したプロセスを安全に停止するのに必要なコマンドが含まれています。他の方法を使用して、help_start スクリプトを停止しないでください。

7. *DB2 インフォメーション・センター* を再開します。
   - Windows では、「**スタート**」 > 「**コントロール パネル**」 > 「**管理ツール**」 > 「**サービス**」をクリックします。次に、「**DB2 インフォメーション・センター**」サービスを右クリックして「**開始**」を選択します。
   - Linux では、以下のコマンドを入力します。

```
/etc/init.d/db2icdv10 start
```

タスクの結果

更新された *DB2 インフォメーション・センター* に、更新された新しいトピックが表示されます。

DB2 チュートリアル

DB2 チュートリアルは、DB2 データベース製品のさまざまな機能について学習するための支援となります。この演習をとおして段階的に学習することができます。

はじめに

インフォメーション・センター (http://publib.boulder.ibm.com/infocenter/db2luw/v10r1/) から、このチュートリアルの XHTML 版を表示できます。

演習の中で、サンプル・データまたはサンプル・コードを使用する場合があります。個々のタスクの前提条件については、チュートリアルを参照してください。

DB2 チュートリアル

チュートリアルを表示するには、タイトルをクリックします。

「*pureXML ガイド*」の『**pureXML**』
: XML データを保管し、ネイティブ XML データ・ストアに対して基本的な操作を実行できるように、DB2 データベースをセットアップします。

DB2 トラブルシューティング情報

DB2 データベース製品を使用する際に役立つ、トラブルシューティングおよび問題判別に関する広範囲な情報を利用できます。

DB2 の資料

トラブルシューティング情報は、「*問題判別およびデータベース・パフォーマンスのチューニング*」または *DB2 インフォメーション・センター* の『データベースの基本』セクションにあります。ここには、以下の情報が記載されています。

- DB2 診断ツールおよびユーティリティーを使用した、問題の切り分け方法および識別方法に関する情報。
- 最も一般的な問題のうち、いくつかの解決方法。
- DB2 データベース製品で発生する可能性のある、その他の問題の解決に役立つアドバイス。

IBM サポート・ポータル

現在問題が発生していて、考えられる原因とソリューションを見つけるには、IBM サポート・ポータルを参照してください。 Technical Support サイトには、最新の DB2 資料、TechNotes、プログラム診断依頼書 (APAR またはバグ修正)、フィックスパック、およびその他のリソースへのリンクが用意されています。この知識ベースを活用して、問題に対する有効なソリューションを探し出すことができます。

IBM サポート・ポータル (http://www.ibm.com/support/entry/portal/Overview/Software/Information_Management/DB2_for_Linux,_UNIX_and_Windows) にアクセスしてください。

ご利用条件

これらの資料は、以下の条件に同意していただける場合に限りご使用いただけます。

適用度: これらのご利用条件は、IBM Web サイトのあらゆるご利用条件に追加で適用されるものです。

個人使用: これらの資料は、すべての著作権表示その他の所有権表示をしていただくことを条件に、非商業的な個人による使用目的に限り複製することができます。ただし、IBM の明示的な承諾をえずに、これらの資料またはその一部について、二次的著作物を作成したり、配布 (頒布、送信を含む) または表示 (上映を含む) することはできません。

商業的使用: これらの資料は、すべての著作権表示その他の所有権表示をしていただくことを条件に、お客様の企業内に限り、複製、配布、および表示することができます。 ただし、IBM の明示的な承諾をえずにこれらの資料の二次的著作物を作成したり、お客様の企業外で資料またはその一部を複製、配布、または表示することはできません。

権利: ここで明示的に許可されているもの以外に、資料や資料内に含まれる情報、データ、ソフトウェア、またはその他の知的所有権に対するいかなる許可、ライセンス、または権利を明示的にも黙示的にも付与するものではありません。

資料の使用が IBM の利益を損なうと判断された場合や、上記の条件が適切に守られていないと判断された場合、IBM はいつでも自らの判断により、ここで与えた許可を撤回できるものとさせていただきます。

お客様がこの情報をダウンロード、輸出、または再輸出する際には、米国のすべての輸出入関連法規を含む、すべての関連法規を遵守するものとします。

IBM は、これらの資料の内容についていかなる保証もしません。これらの資料は、特定物として現存するままの状態で提供され、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なしで提供されます。

IBM の商標: IBM、IBM ロゴおよび ibm.com は、世界の多くの国で登録された International Business Machines Corporation の商標です。他の製品名およびサービス名等は、それぞれ IBM または各社の商標である場合があります。現時点での IBM の商標リストについては、http://www.ibm.com/legal/copytrade.shtml をご覧ください。

付録 E. 特記事項

本書は米国 IBM が提供する製品およびサービスについて作成したものです。IBM 以外の製品に関する情報は、本書の最初の発行時点で入手可能な情報に基づいており、変更される場合があります。

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。

〒103-8510
東京都中央区日本橋箱崎町19番21号
日本アイ・ビー・エム株式会社
法務・知的財産
知的財産権ライセンス渉外

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
U59/3600
3600 Steeles Avenue East
Markham, Ontario L3R 9Z7
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。サンプル・プログラムは、現存するままの状態で提供されるものであり、いかなる種類の保証も提供されません。 IBM は、これらのサンプル・プログラムの使用から生ずるいかなる損害に対しても責任を負いません。

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。

© (*お客様の会社名*) (*西暦年*). このコードの一部は、IBM Corp. のサンプル・プログラムから取られています。 © Copyright IBM Corp. _*年を入れる*_. All rights reserved.

商標

IBM、IBM ロゴおよび ibm.com は、世界の多くの国で登録された International Business Machines Corporation の商標です。他の製品名およびサービス名等は、それぞれ IBM または各社の商標である場合があります。現時点での IBM の商標リストについては、http://www.ibm.com/legal/copytrade.shtml をご覧ください。

以下は、それぞれ各社の商標または登録商標です。

- Linux は、Linus Torvalds 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 Oracle やその関連会社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。
- インテル、Intel、Intel ロゴ、Intel Inside、Intel Inside ロゴ、Celeron、Intel SpeedStep、Itanium、Pentium は、Intel Corporation または子会社の米国およびその他の国における商標または登録商標です。
- Microsoft、 Windows、 Windows NT、および Windows ロゴは、Microsoft Corporation の米国およびその他の国における商標です。

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

I

J

N

P

R

S

U

V

X

[特殊文字]

Printed in Japan

SA88-4686-00

日本アイ・ビー・エム株式会社
〒103-8510 東京都中央区日本橋箱崎町19-21

Spine information:

IBM DB2 10.1 for Linux, UNIX, and Windows

pureXML ガイド