IBM i
バージョン 7.2

Database
SQL XML プログラミング

IBM i
バージョン 7.2

Database
SQL XML プログラミング

お願い

本書および本書で紹介する製品をご使用になる前に、 237 ページの『特記事項』に記載されている情報をお読みください。

本製品およびオプションに付属の電源コードは、他の電気機器で使用しないでください。

本書は、IBM i 7.1 (製品番号 5770-SS1) に適用されます。また、改訂版で断りがない限り、それ以降のすべてのリリースおよびモディフィケーションに適用されます。このバージョンは、すべての RISC モデルで稼働するとは限りません。また CISC モデルでは稼働しません。

本書にはライセンス内部コードについての参照が含まれている場合があります。ライセンス内部コードは機械コードであり、IBM 機械コードのご使用条件に基づいて使用権を許諾するものです。

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： IBM i
Version 7.2
Database
SQL XML Programming

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 2014.4

目次

SQL XML プログラミング

DB2® for IBM® i は、構造化照会言語 (SQL) を使用した XML データの保管および検索をサポートしています。表、関数、プロシージャーなど、SQL を使用して定義されたオブジェクトは、列、パラメーター、および変数の定義に XML データ・タイプを使用できます。 XML データ・タイプに加えて、XML 文書を生成して、XML 文書のすべてまたは一部を取得するために使用できる組み込みの関数およびプロシージャーがあります。

注: コード例を使用する場合は、 235 ページの『コードに関するライセンス情報および特記事項』のご使用条件に同意する必要があります。

IBM i 7.2 の新機能

SQL XML プログラミングのトピック集に関する新情報や重要な変更情報についてお読みください。

SQL XML プログラミングのトピック集は新規のものです。ここには、SQL プログラミングのトピック集に以前あった XML 情報がすべて含まれています。

新規情報または変更情報の見分け方

技術上の変更が加えられた場所を見分けるのに役立つように、Information Center では以下のイメージを使用しています。

- » イメージにより、新規または変更された情報の開始点を示します。
- « イメージにより、新規または変更された情報の終了点を示します。

PDF ファイルでは、左マージンに新規および変更情報のリビジョン・バー (|) があります。

このリリースでの新機能または変更点についてのその他の情報は、「プログラム資料説明書」を参照してください。

構文図の見方

本書で使用される構文図には、以下の規則が適用されます。

- 構文図は、線に沿って左から右へ、上から下へ読んでください。

  記号 ▸▸── は、構文図の始まりを示します。

  記号 ──▸ は、構文が次の行に続くことを示します。

  記号 ▸── は、構文が前の行から続いていることを示します。

  記号 ──▸◂ は、構文図の終わりを示します。

  構文単位を示す図は、記号 ├── で始まり、記号 ──┤ で終わります。
- 必須項目は、次のように水平方向の線 (メインパス) 上に示します。

- オプション項目は、メインパスより下に表示されます。

メインパスより上に示される項目は、単に読みやすさのために使用されるオプション項目であり、ステートメントの実行には影響を与えません。

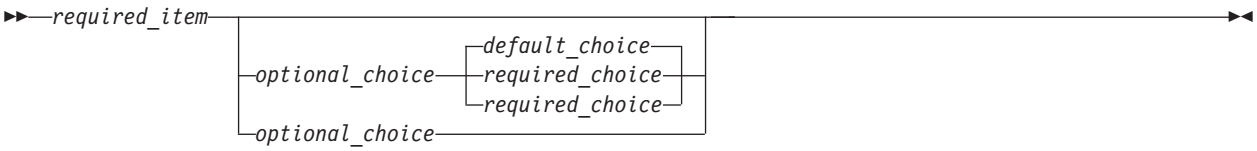
- 複数の項目からユーザーが選択できる場合は、それらの項目を縦方向に並べて示します。

項目の中から必ずどれか 1 つを選択しなければならない場合は、縦方向に並んでいる選択項目のうちの 1 つをメインパス上に示します。

項目を選択しても選択しなくてもよい場合は、縦方向に並んでいる選択項目をすべてメインパスの下に示します。

項目の中にデフォルトの選択項目がある場合は、その選択項目をメインパスの上に示し、残りの選択項目をメインパスの下に示します。

指定されていないオプション項目にデフォルトがある場合、デフォルトはメインパスの上に示します。



- メインパスの上を通って左に戻る矢印は、反復可能な項目を示します。

反復可能を示す矢印にコンマが入っている場合は、反復可能な項目を指定するときに、項目相互間をコンマで区切る必要があります。

項目群の上にある反復可能を示す矢印は、その項目群にある項目を反復して指定できることを示します。

- キーワードは大文字で示されます (例: FROM)。これは表記どおり正確につづらなければなりません。変数は全体が小文字で示されます (例えば、*column-name*)。これらはユーザーが指定する名前または値を表しています。
- 構文図に句読記号、括弧、算術演算子、またはその他の記号が示されている場合には、それらを構文の一部として入力しなければなりません。

SQL プログラミングの PDF ファイル

この情報の PDF ファイルを表示または印刷できます。

本書の PDF 版を表示またはダウンロードするには、「SQL XML プログラミング」を選択します。

PDF ファイルの保存

表示または印刷のために PDF をワークステーションに保存するには、以下のようにします。

1. ご使用のブラウザーで PDF リンクを右クリックする。
2. PDF をローカルに保存するオプションをクリックする。
3. PDF を保存したいディレクトリーに進む。
4. 「**保存**」をクリックする。

Adobe Reader のダウンロード

これらの PDF を表示または印刷するには、Adobe Reader がご使用のシステムにインストールされている必要があります。このアプリケーションは、 Adobe Web サイト (http://get.adobe.com/reader/) から無償でダウンロードできます。

SQL ステートメントおよび SQL/XML 関数

多くの SQL ステートメントは、XML データ・タイプをサポートしています。これにより、XML データに関連した多くの一般的なデータベース操作を実行できます。これには、XML 列を含んだ表の作成、既存の表への XML 列の追加、XML 列を含んだ表に対するトリガーの作成、XML 文書の挿入、更新、または削除などが含まれます。 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します。

XML データ・タイプ

XML データ・タイプは、2 GB までの XML 値を保管できます。 XML データ・タイプには、CCSID を指定できます。 CCSID が指定されない場合、SQL_XML_DATA_CCSID QAQQINI オプションの値が使用されます。このオプションのデフォルトは、1208 (UTF-8) です。 XML データ・タイプは、1 バイト文字および Unicode の 2 バイト文字を保管できます。

表内の 1 つ以上の XML 値または LOB 値が入っている単一行は、3.5 GB を超えることはできません。XML データ・タイプは、パーティション表で指定できます。

XML ホスト変数および XML ロケーターは、アプリケーション・プログラムで宣言できます。

XML ロケーターを使用して、XML 値を参照できます。 XML 値は、XML ロケーター内に取り出すことができます。 XML ロケーターは、プロシージャーまたは関数に渡すことができます。ロケーターは、INSERT または UPDATE ステートメントで値として指定できます。

XML 列のジャーナル項目は、LOB の場合と同じです。 SQL プログラミングのトピック集の『LOB 列のジャーナル・エントリーのレイアウト』を参照してください。

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ます。SQL および外部の関数と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

XML 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 データとリレーショナル・データのアクセスと保管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、SQL で XML データ・タイプをサポートします。

- ILE RPG
- ILE COBOL
- C または C++ (組み込み SQL または DB2 CLI)
- Java™ (JDBC または SQLJ)

SQL および外部の関数とプロシージャー

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。また、CREATE FUNCTION 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 関数および外部関数に渡すこともできます。 SQL ルーチンの既存のフィーチャーは、変数への XML データ値の一時格納に加えて、XML 値を生成または利用する SQL ステートメントの前後に置かれるプロシージャー型ロジック・フローの実装をサポートします。

管理

XML フィーチャーにより、XML 文書の URI 従属関係の管理を行うためのリポジトリーが提供されます。

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なすべての 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 スキーマを格納します。これを使用して、XML インスタンス文書の妥当性検査または分解を行うことができます。

ツール

XML データ・タイプのサポートは、IBM i ナビゲーターで使用できます。

アノテーション付き XML スキーマ分解

XML フィーチャーにより、 XML データを XML 文書として保管およびアクセスできます。 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分解します。

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、XML 文書形式で保持されます。

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んだ表で定義できます。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

XML データをデータベース・システムに入れるために使用する方法は、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、SQL INSERT 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。挿入操作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。 SQL UPDATE ステートメントを使用して、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。分解の際には、XML スキーマ文書に追加されたアノテーションに従って、データが列に保管されます。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。

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、DB2 によって提供されるストアード・プロシージャーを使用して、XSR に登録されます。

出力

XML データは、SQL を使用してデータベース・システムから取り出します。

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。 XMLTABLE 組み込み表関数は、SQL 照会で XML 文書の一部を検索する場合に使用できます。

また DB2 データベース・サーバーに保管されているリレーショナル・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いくつかあります。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。

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 モデルとリレーショナル・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。また、設計に際しては、DB2 データベースの特徴である、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能力も活用できます。

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肢を見極めるための要素は多数あります。この解説は、特定の実装環境に影響を与える可能性のあるさまざまな要素を評価するための指針として使用してください。

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あるのに対し、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、階層の形式で入っています。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他の方法はありません。リレーショナル・データでは、全選択の中の 1 つ以上の列に対して ORDER BY 文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を与える要因

保管するデータの種類によって、そのデータを保管する方法が決まることがあります。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素も影響することがあります。

柔軟性が最大限に必要かどうか
リレーショナル表は、かなり固定的なモデルに従います。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 つの表にしたり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 スキーマは時間と共に発展することがあります。

データ検索の最大限のパフォーマンスが必要かどうか
XML データのシリアライズおよび解釈には、ある程度のコストが関連しています。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。

データが後でリレーショナル・データとして処理されるかどうか

後続のデータ処理が、リレーショナル・データベースに保管されているデータに依存する場合、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のが適切である場合があります。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。または、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のが適切な方法である場合があります。

データ・コンポーネントが階層の外部で意味を持つかどうか

データによっては、元々階層的な性質を持ってはいても、子コンポーネントは親から値を供給される必要がない場合もあります。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・ナンバーが含まれる購入注文は、XML 文書として表現するのが最適かもしれません。しかし、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のか、またはデータの小さなサブセットだけに適用されるのか

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。カタログ内のどの品目にせよ、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・ソーには意味がありますが、コートには意味がありません。この種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。

参照整合性が必要かどうか

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で使用す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かどうか

XML 列内の XML データを更新するには文書全体を置き換える必要があります。非常に大きい文書の小さな断片を多数の行で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 として保管しても効率は同じになる場合があります。

XML のチュートリアル

XML データ・タイプを使用すると、各行に 1 つの整形式 XML 文書を保管する表列を定義できます。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て、XML データの保管と、XML フィーチャーでの基本操作を実行する方法を示します。

このチュートリアルを完了すると、以下の作業を実行できるようになります。

- XML データを格納できる表を作成する
- XML 型付き列に XML 文書を挿入する
- XML 列に格納されている XML 文書を更新する
- XML スキーマに照らして XML 文書を妥当性検査する
- XSLT スタイルシートを使用した変換

準備

演習の中の例を、IBM i Navigator Run SQL スクリプト・ツールに入力するか、またはコピーして貼り付ける必要があります。対話式 SQL を使用すると、XML 結果データが直列化されたデータとして表示されません。

演習 1: XML データを格納できる表を作成する

この演習では、XML 列を含んだ表の作成方法を示します。

次のようにして、XML 列を含んだ Customer という名前の表を作成します。

```
CREATE SCHEMA POSAMPLE;

SET CURRENT SCHEMA POSAMPLE;

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

主キーの指定はオプションであり、XML を格納するために必須ではないことに注意してください。

また、ALTER TABLE SQL ステートメントを使用して、既存の表に 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 型付き列に XML 文書を挿入する

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

一般的には、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アプリケーションで挿入できますが、多数のソースに由来する XML 文書が異なるエンコード方式で処理される場合には、XML またはバイナリー形式を使用することをお勧めします。

この演習では、XML 文書が常に文字リテラルである「SQL スクリプトの実行」で、XML 文書を XML 型付き列に手動で挿入する方法を示します。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ができません。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 または UPDATE 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 2 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 の構文解析に関する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>');
```

次のようにすると、レコードが正常に挿入されたことを確認できます。

```
SELECT * from Customer;
```

この演習を対話式 SQL で実行している場合、XML 値は自動的には直列化されません。挿入されたデータを表示するには、XMLSERIALIZE 関数を明示的に使用する必要があります。

チュートリアルに戻る

演習 3: XML 列に格納されている XML 文書を更新する

この演習では、SQL ステートメントで XML 文書を更新する方法を示します。

SQL による更新

XML 列に格納されている XML 文書を SQL を使用して更新するには、 SQL UPDATE ステートメントを使用して、全文更新を実行する必要があります。

次のようにして、演習 2 で挿入した文書の 1 つを更新します (<street>、<city>、および <pcode-zip> の各要素の値が変更されています)。

```
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 Cid = 1002;
```

次のようにして、XML 文書が更新されたことを確認します。

```
SELECT * from Customer;
```

この演習を対話式 SQL で実行している場合、XML 値は自動的には直列化されません。更新されたデータを表示するには、XMLSERIALIZE 関数を明示的に使用する必要があります。

Cid="1002" の行には、変更された <street>、<city>、および <pcode-zip> の値が格納されます。

XML 文書は、同じ表の XML 以外の列の値によって識別できます。

チュートリアルに戻る

演習 4: XML スキーマに照らして XML 文書を妥当性検査する

この演習では、XML 文書を妥当性検査する方法を示します。 XML 文書は、XML スキーマに照らしてのみ妥当性検査できます。DTD 妥当性検査はサポートされていません。 (DTD に照らして妥当性検査することはできませんが、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、既存の表、XML 文書などの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があります。

妥当性検査する前に、組み込み XML スキーマ・リポジトリー (XSR) に 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 スキーマ文書の登録が含まれます。すべての XML スキーマ文書が正常に登録された後に、登録を完了する必要があります。

次のようにして、posample.customer XML スキーマを登録してから登録を完了します。

```
CREATE PROCEDURE SAMPLE_REGISTER
LANGUAGE SQL
BEGIN
  DECLARE CONTENT BLOB(1M);
  VALUES BLOB('<?xml version="1.0"?>
<xs:schema targetNamespace="http://posample.org"
  xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
  <xs:element name="customerinfo">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="name" type="xs:string" minOccurs="1" />
        <xs:element name="addr" minOccurs="1" maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="street" type="xs:string" minOccurs="1" />
              <xs:element name="city" type="xs:string" minOccurs="1" />
              <xs:element name="prov-state" type="xs:string" minOccurs="1" />
              <xs:element name="pcode-zip" type="xs:string" minOccurs="1" />
            </xs:sequence>
            <xs:attribute name="country" type="xs:string" />
          </xs:complexType>
        </xs:element>
        <xs:element name="phone" nillable="true" minOccurs="0" maxOccurs="unbounded">
          <xs:complexType>
            <xs:simpleContent>
              <xs:extension base="xs:string">
                <xs:attribute name="type" form="unqualified" type="xs:string" />
              </xs:extension>
            </xs:simpleContent>
          </xs:complexType>
        </xs:element>
        <xs:element name="assistant" minOccurs="0" maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="name" type="xs:string" minOccurs="0" />
              <xs:element name="phone" nillable="true" minOccurs="0" maxOccurs="unbounded">
                <xs:complexType>
                  <xs:simpleContent  >
                    <xs:extension base="xs:string">
                      <xs:attribute name="type" type="xs:string" />
                    </xs:extension>
                  </xs:simpleContent>
```

```
              </xs:complexType>
            </xs:element>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
    <xs:attribute name="Cid" type="xs:integer" />
  </xs:complexType>
 </xs:element>
</xs:schema>') INTO CONTENT;

 CALL SYSPROC.XSR_REGISTER('POSAMPLE', 'CUSTOMER', 'http://posample.org', CONTENT, null);
END;

SET PATH POSAMPLE;

CALL SAMPLE_REGISTER;

CALL SYSPROC.XSR_COMPLETE('POSAMPLE', 'CUSTOMER', null, 0);
```

XSR に格納されているオブジェクトについての情報が入っている QSYS2.XSROBJECTS カタログ・ビューを照会することにより、 XML スキーマが正常に登録されたことを検証できます。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 XSROBJECTSCHEMA, XSROBJECTNAME FROM QSYS2.XSROBJECTS
  WHERE XSROBJECTSCHEMA = 'POSAMPLE';

XSROBJECTSCHEMA        XSROBJECTNAME
--------------------   --------------------
POSAMPLE               CUSTOMER
```

これで、この XML スキーマは妥当性検査で使用できます。通常、妥当性検査は INSERT または UPDATE 操作の際に実行されます。 XMLVALIDATE 関数を使用して妥当性検査を行います。XMLVALIDATE が指定された 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。

次の INSERT ステートメントは、前に登録された posample.customer XML スキーマに従って文書が有効である場合にのみ、 Customer 表の Info 列に新しい XML 文書を挿入します。

```
INSERT INTO Customer(Cid, Info) VALUES (1004, XMLVALIDATE (XMLPARSE (DOCUMENT
'<customerinfo xmlns="http://posample.org" Cid="1004">
  <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 ));
```

XMLVALIDATE は XML データを操作します。この例では、XML 文書が文字データとして渡されるので、XMLVALIDATE を XMLPARSE 関数と一緒に使用する必要があります。文字データを XML に直接割り当てることができるのは、INSERT、UPDATE、または MERGE ステートメントのみであることに注意してください。ここでは、INSERT ステートメントが使用されます。 XMLPARSE 関数は、その引数を XML 文書として構文解析してから、XML 値を返します。

妥当性検査および挿入が正常に行われたことを検証するには、以下のようにして Info 列を照会します。

```
SELECT Info FROM Customer;
```

この照会では、3 つの XML 文書が返されるはずです。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 5: XSLT スタイルシートを使用した変換

XSLTRANSFORM 関数を使用して、データベース内の XML データを他のフォーマットに変換できます。

この例では、XSLTRANSFORM 組み込み関数を使用して、データベースに保管されている XML 文書を変換する方法を示します。この事例では、XML 文書に任意の数の大学生レコードが含まれています。各 student 要素には、以下のように、生徒の ID、ファーストネーム、ラストネーム、年齢、および在籍する大学が含まれています。

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

この XSLT 変換の目的は、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 ページを作成することです。そのために、以下の XSLT スタイルシートを使用します。このスタイルシートもデータベースに保管されています。

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

このスタイルシートは標準の XSLT 変換でも作動しますし、実行時の動作を制御するために提供されているパラメーター・ファイルを使用しても作動します。

1. XML 文書およびスタイルシート文書を保管することができる表を作成します。

   ```
   SET CURRENT SCHEMA USER;

   CREATE TABLE XML_TAB (DOCID INTEGER, XML_DOC XML, XSL_DOC CLOB(1M));
   ```

2. その表に文書を挿入します。この例では、XML 文書および XSLT スタイルシートを別々の列値として同じ表にロードできます。 INSERT ステートメントは、3 番目の値として XSLT スタイルシートの切り捨てられたバージョンを使用します。以下の INSERT ステートメントを使用するには、切り捨てられたスタイルシートの値を、この演習で前にリストされた XSLT スタイルシートに置き換えます。

   ```
   INSERT INTO XML_TAB VALUES
        (1,
         '<?xml version="1.0"?>
         <students xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
             <student studentID="1" firstName="Steffen" lastName="Siegmund"
      age="23" university="Rostock"/>
          </students>',

         '<?xml version="1.0" encoding="UTF-8"?>
         <xsl:stylesheet version="1.0"
              ...
         </xsl:stylesheet>'
   );
   ```

3. XSLTRANSFORM 組み込み関数を呼び出して、XML 文書を変換します。

   ```
   SELECT XSLTRANSFORM (XML_DOC USING XSL_DOC AS CLOB(1M)) FROM XML_TAB;
   ```

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></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>
```

これは簡単な方法ではありますが、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 (例えば標準 HTML ではなく XHTML に変更するなど) したりするために、実行時に XSLT スタイルシートの動作を変更する場合もあります。別のパラメーター・ファイルを使用して、実行時に 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・ファイル内の類似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。

例えば、上記のスタイルシートには次の 2 つのパラメーターが定義されています。

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

これらのパラメーターは、前述のように最初の変換では使用されていません。パラメーターの受け渡しが行われる方法を知るには、以下のようにパラメーター・ファイルを作成します。

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

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>'
     );
```

次の照会結果を調べます。

```
SELECT XSLTRANSFORM (
   XML_DOC USING XSL_DOC WITH PARAM AS CLOB(1M)) FROM XML_TAB X, PARAM_TAB P
WHERE X.DOCID=P.DOCID;
```

上記の照会により、次の HTML が生成されます。

```
<html>
<head>
<META http-equiv="Content-Type" content="text/html; charset=UTF-8">
</head>
<body>
<h1>The student's list ...</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>
<td width="200">University</td>
</tr>
</th>
<tr>
<td>1</td>
<td>Steffen</td>
<td>Siegmund</td><td>23</td><td>Rostock</td>
</tr>
</table>
</body>
</html>
```

チュートリアルに戻る

XML データの挿入

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

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。

```
SET CURRENT SCHEMA USER;

CREATE TABLE MyCustomer LIKE Customer;
ALTER TABLE MyCustomer ADD COLUMN Preferences 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、またはバイナリーのいずれかのタイプとなります。

DB2 データベース・サーバーがホスト変数のデータ・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ようにしてください。

アプリケーション内の XML データは、直列化されたストリング・フォーマットになります。そのデータを XML 列に挿入する際、保管された XML 文書のフォーマットにデータを変換する必要があります。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、データをその直列化されたストリング・フォーマットから XML 文書フォーマットに変換できます。

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 xmlns="http://posample.org" 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: 組み込み静的 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、直列化されたストリング・フォーマットから XML 文書フォーマット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、以下の場合に実行されます。

- 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・サーバーは、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、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。暗黙的な構文解析は、ステートメントを実行するときに実行されます。

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、バイナリー、または Unicode グラフィックのいずれかのデータ・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。例えば、次の通りです。

```
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 のいずれかのタイプ) 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の境界空白文字の保持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、読みやすく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) のことをいいます。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、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 の間のスペース、および と の間のスペースは、境界空白です。

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

明示的に XMLPARSE を呼び出すとき、 STRIP WHITESPACE または PRESERVE WHITESPACE オプションを使用して境界空白の保持を制御します。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、次のようになります。

- 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ない場合、 DB2 データベース・サーバーは常に空白文字を除去します。
- 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持を制御できます。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、妥当性検査の規則だけを使用して空白の除去または保持を決めます。

```
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 標準は、XML データ内の空白文字の除去または保持を制御する xml:space 属性を規定しています。xml:space 属性は、暗黙的または明示的な XML 構文解析における空白文字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持されます。それらのスペースが、属性 xml:space="preserve" で定義された要素に含まれているためです。

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

ただし次の文書では、 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で制御できます。それらのスペースが、属性 xml:space="default" で定義された要素に含まれているためです。

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

XML 値を構成するための SQL/XML 発行関数

結果として生成される 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。結果における出現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 として返されます。 XML 値の使用目的によっては、その値を明示的に直列化して別の 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 の直列化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。

XMLAGG 集約関数
一連の XML 値において、NULL 以外の各値に対して、1 つの項目が入った XML シーケンスを返します。

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
入力引数を内容とする XML 値を返します。

XMLCONCAT スカラー関数
不特定数の XML 入力引数を連結したものが入ったシーケンスを返します。

XMLDOCUMENT スカラー関数
整形式 XML 文書である XML 値を返します。 DB2 表に保管されるすべての XML 値は文書でなければなりません。この関数は、XML 値から XML 文書を形成します。

XMLELEMENT スカラー関数
XML 要素である XML 値を返します。 DB2 表に保管されるすべての XML 値は文書でなければなりません。 XMLELEMENT 関数が作成するのは要素だけで、文書は作成しません。保管される XML 値は、XMLDOCUMENT 関数によって形成される文書でなければなりません。

XMLFOREST スカラー関数
XML 要素のシーケンスである XML 値を返します。

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返します。デフォルトでは、結果セット内の各行は行サブエレメントにマップされ、それぞれの入力式は行サブエレメントのサブエレメントにマップされます。オプションとして、結果内の各行を行サブエレメントにマップできます。さらに、それぞれの入力式を行サブエレメント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 関数および XMLFOREST 関数の引数としてのみ使用できます。

XMLPI スカラー関数
単一の処理命令を持った XML 値を返します。

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返します。デフォルトでは、各入力式は行要素のサブエレメントに変換されます。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
入力引数の値が入った XML 値を返します。

XSLTRANSFORM スカラー関数
XML データを、他の XML スキーマなどの他の形式に変換します。

例: 単一の表からの値による XML 文書の作成

この例は、発行に適した XML 値を、SQL/XML 発行関数を使用して単一の表から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。次の条件で照会を作成します。

- 各 <item> 要素は、XMLELEMENT 関数を使用して PRODUCT 表の *NAME* 列の値を使って構成されます。
- その後、XMLAGG を使用してすべての <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。
- XMLNAMESPACES 関数によって、名前空間が <allProducts> 要素に追加されます。

```
SELECT XMLELEMENT (NAME "allProducts",
                   XMLNAMESPACES (DEFAULT 'http://posample.org'),
                   XMLAGG(XMLELEMENT (NAME "item", p.name)))
FROM Product p
```

この照会は次の XML 値を返します。ここでは整形して読みやすくしています。

```
<allProducts xmlns="http://posample.org">
  <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>
```

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、一連の行要素を含んだ同様の XML 文書を構成できます。以下のようにすると、項目要素に名前空間接頭部が付与されます。

```
SELECT XMLELEMENT (NAME "products",
                   XMLNAMESPACES ('http://posample.org' AS "po"),
                   XMLROW(NAME AS "po:item"))
FROM Product
```

結果出力は以下のようになります。

```
<products xmlns:po="http://posample.org">
   <row>
      <po:item>Snow Shovel, Basic 22 inch</po:item>
   </row>
</products>
<products xmlns:po="http://posample.org">
   <row>
      <po:item>Snow Shovel, Deluxe 24 inch</po:item>
   </row>
</products>
<products xmlns:po="http://posample.org">
      <row><po:item>Snow Shovel, Super Deluxe 26 inch</po:item>
   </row>
</products>
<products xmlns:po="http://posample.org">
      <row><po:item>Ice Scraper, Windshield 4 inch</po:item>
   </row>
</products>
```

例: 複数の表からの値による XML 文書の作成

この例は、発行に適した XML 値を、SQL/XML 発行関数を使用して複数の表から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。次の条件で照会を作成します。

- XMLFOREST 関数を使用して、<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 表の *NAME* 列および INVENTORY 表の *QUANTITY* 列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> 要素内に集約されます。

```
SELECT XMLELEMENT (NAME "saleProducts",
                   XMLNAMESPACES (DEFAULT 'http://posample.org'),
                   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
```

直前の照会によって、次の XML 文書が生成されます。

```
<saleProducts xmlns="http://posample.org">
  <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 要素を持つ表の行からの値による XML 文書の作成

この例は、発行に適した XML 値を、NULL 要素を持つ表の行から SQL/XML 発行関数を使用して構成する方法を示しています。

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。 XMLELEMENT 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。

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> 要素を返しません。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
```

生成される値には、NULL 値が含まれる行に対する <loc> 要素はありません。

```
<newElem prodID="100-100-01">
  <quantity>5</quantity>
</newElem>
```

同じ照会で EMPTY ON NULL オプションを指定すると、空の <loc> 要素が返されます。

```
SELECT XMLELEMENT (NAME "newElem",
                   XMLATTRIBUTES (PID AS "prodID"),
                   XMLFOREST (QUANTITY AS "quantity",
                              LOCATION AS "loc" OPTION EMPTY ON NULL))
FROM INVENTORY
```

生成される値には、空の <loc> 要素があります。

```
<newElem prodID="100-100-01">
  <quantity>5</quantity>
  <loc/>
</newElem>
```

例: XSLT スタイルシートを使用した変換

XML データを他のフォーマットに変換するための標準的な方法は、XSLT (Extensible Stylesheet Language Transformations) を使用する方法です。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、または異なる 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・フォーマットに変換します。 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ためにも使用できます。また XSLT は XML データを関連のないフォーマット (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、主に次のような 2 つの適用可能な分野があります。

- フォーマット設定 (XML から HTML への変換)
- データ交換 (1 つの XML スキーマから別の XML スキーマへの、または SOAP などのデータ交換フォーマットへのデータの照会、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の変換が必要になる場合があります。 XSLT には XPath 仕様が組み込まれているので、ソース XML 文書にある任意のデータの照会および取得が可能です。また XSLT テンプレートには、出力ファイルに追加されるファイル・ヘッダーや命令ブロックなどの追加情報も含まれています。あるいはそれらがテンプレートで作成される場合もあります。

XSLT の動作方法

XSLT スタイルシートは、XSL (Extensible Stylesheet Language) で作成される XML スキーマです。 XSL は、C や 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・ファイル内で検出した場合に実行するアクションを記述するものです。標準的な XSLT テンプレート要素では、最初に適用先の要素の指定をします。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 ファイル内で検出される <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。この事例では、アイス・スクレーパーを記述するこのレコードのような、在庫情報が入った 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・ガラス用アイス・スクレーパーのパーツ・ナンバー、説明、価格などの情報が含まれています。この情報の一部は、<name> などの要素内に入っています。パーツ・ナンバーなどのように、属性内に含まれているものもあります (この事例では <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、上記のテンプレートとターゲット文書の両方を入力として受け取ると、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、指定の条件に照らして着信 XML 文書をテストします (通常は 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そのテンプレートの内容が出力に挿入され、偽の場合にはそのテンプレートはプロセッサーで無視されます。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 を使用すると、テンプレート条件の定義 (例: `<xsl:template match="product">`) と、XML ストリーム内の任意の場所にあるデータの選択と挿入 (例: `<h1><xsl:value-of select="/product/description/name"/></h1>`) の両方を行うことができます。

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 スタイルシートを指定すると、この関数によってその文書にスタイルシートが適用されて、結果が返されます。

例: フォーマット設定エンジンとしての XSLT の使用

以下の例では、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ます。

セットアップするには、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="23" university="Rostock"/>
</students>',

    '<?xml version="1.0" ?>
<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、それを表示します。

```
SELECT XSLTRANSFORM (XML_DOC USING XSL_DOC AS CLOB(1M)) FROM XML_TAB;
```

結果は以下の文書になります。

```
<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、そこに渡されるデータにのみ影響を与えます。そのため、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 の使用方法について示しています。

例: データ交換での XSLT の使用

この例では、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ます。

この例は、スタイルシートでパラメーターを使用して、実行時にさまざまなデータ交換フォーマットを生成します。

ここでは、xsl:param 要素が組み込まれた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収集します。

```
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>'
);
```

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>'
);
```

その後、以下のコマンドを使用して、実行時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、その内容は、このパラメーター・ファイルと、XML 文書の内容に対して行われた検査によって決定されます。

```
<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 の出力が HTML ではなく、別の XML 文書または別のデータ・フォーマットを使用したファイル (EDI ファイルなど) となる場合があります。

データ交換アプリケーションでは、パラメーター・ファイルに 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 (E メール・アドレス、ポート・アドレス、特定のトランザクションに対して固有の他の重要なデータなどの情報)。上記の例で使用される XML は在庫レコードであるため、クライアントの購買システムとのやり取りのために XSLT を使用してこのレコードを再パッケージ化するという使い方については、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 文書および複数の XSLT スタイルシートを XML 列に保管し、XSLTRANSFORM 関数を使用していずれかの XSLT スタイルシートで 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 文書が入り、XMLTRANS には XSLT スタイルシートが入ります。

```
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、すべての名前空間情報を除去して以下の XML を返します。

```
<?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 要素のみで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。要素の名前は、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、XSLID = 2 が指定されているので、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 要素のみの名前空間とともに以下の XML を返します。

```
<?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>
```

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、Unicode ではない文字の列に文字が挿入されると、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 1.10 勧告だけがサポートされます。
- すべてのパラメーターおよび結果タイプは、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 宣言を使用) は、サポートされていません。

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 1. 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されたエンティティーに置換されます。

SQL ID と QName

SQL 値から XML 値を発行または構成する際、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」について考慮します。 @ 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 節で名前として指定される 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 直列化

XML 直列化は、XML データを、DB2 データベース内でのフォーマットから、アプリケーション内での直列化されたストリング・フォーマット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直列化を行うこともできますし、 XMLSERIALIZE 関数を呼び出して XML 直列化を明示的に要求することもできます。 XML 直列化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信されるときです。

暗黙的な直列化の方がコーディングが容易であり、XML データをクライアントに送信すると、DB2 クライアント側で XML データを適切に処理できるため、暗黙的な直列化はほとんどの場合に好ましい方法となります。以下に示すように、明示的な直列化では追加の処理が必要となります。この処理は、暗黙的な直列化ではクライアントによって自動的に処理されます。

一般的には、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ため、暗黙的な直列化がより好ましい方法といえます。ただし、特定の状況 (後述) 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・データを取得するときの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 直列化

DB2 CLI および組み込み SQL アプリケーションの暗黙的な直列化では、適切なエンコード方式を指定した XML 宣言が DB2 データベース・サーバーによってデータに追加されます。 .NET アプリケーションでも、DB2 データベース・サーバーは XML 宣言を追加します。 Java アプリケーションでは、 SQLXML オブジェクトからデータを取り出すために呼び出される SQLXML オブジェクト・メソッドに応じて、DB2 データベース・サーバーによって XML 宣言が追加されたデータが返されます。

例: C プログラムで、カスタマー ID が「1000」の customerinfo 文書を暗黙的に直列化し、直列化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。取り出されるデータは 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 直列化

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 スカラー関数では、以下を指定できます。

- データが直列化されるときの、変換後の SQL データ・タイプ

  データ・タイプは、文字、グラフィック、またはバイナリーのいずれかのデータ・タイプです。
- 出力データに明示的なエンコード指定 (EXCLUDING XMLDECLARATION または INCLUDING XMLDECLARATION) を含めるかどうか

XMLSERIALIZE からの出力は、Unicode、文字、またはグラフィックのデータとなります。

直列化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、エンコード名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。

一般的には、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ため、暗黙的な直列化がより好ましい方法といえます。ただし、クライアントが XML データをサポートしないときは、明示的な XMLSERIALIZE を行う方が優れています。

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 直列化を使用するときには、 DB2 データベース・サーバーはデータを CLOB または DBCLOB に変換してからクライアントに送信します。

取り出したデータを別のデータ・タイプにする場合、XMLSERIALIZE を使用できます。

例: サンプル表 `Customer` の XML 列 `Info` に、次のデータと階層的に同等のものが含まれる文書が入っています。

```
<customerinfo xml:space="default" 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>
```

XMLSERIALIZE を呼び出してデータを直列化し、それを BLOB タイプに変換した後に、データを取り出してホスト変数に入れます。

```
SELECT XMLSERIALIZE(Info as BLOB(1M)) into :hostvar from Customer
  WHERE CID=1000
```

保管と取り出しを行った後の XML 文書の相違点

XML 文書を DB2 データベースに保管してからそのコピーをデータベースから取り出すと、取り出し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ます。

文書を保管するとき、文書が多少変更されます。それらの変更内容は、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。
  - 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
  - 文書内のすべての復帰および改行の対 (U+000D および U+000A) または復帰 (U+000D) を改行 (U+000A) に置換する
  - 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、さらに変更が発生します。それらの変更内容は、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。

  DB2 CLI および組み込み SQL アプリケーションの暗黙的な直列化では、適切なエンコード方式を指定した XML 宣言が DB2 データベース・サーバーによってデータに追加されます。 .NET アプリケーションでも、DB2 データベース・サーバーは XML 宣言を追加します。 Java アプリケーションでは、SQLXML オブジェクトからデータを取り出すために呼び出される SQLXML オブジェクト・メソッドに応じて、 DB2 データベース・サーバーによって XML 宣言が追加されたデータが返されます。

  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、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	エンティティー表記
& 記号	U+0026	&amp;
より小記号	U+003C	&lt;
より大記号	U+003E	&gt;

- 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 に基づく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、これらのセクションは出力の中に保持されません。

XML 文書をアーカイブする場合のデータ・タイプ

XML を直列化したストリング・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。文字の列を使用してアーカイブすると、CCSID 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。

XMLTABLE を使用して XML コンテンツをリレーショナル表として参照

XMLTABLE 組み込み表関数は、XML 文書の内容を SQL で参照できる結果セットとして取得する場合に使用できます。

XML 列が以下のように定義された EMP と呼ばれる表があると想定します。

```
CREATE TABLE EMP (DOC XML)
```

この表には、以下のような 2 行が含まれています。

```
<dept bldg="101">
  <employee id="901">
    <name>
      <first>John</first>
      <last>Doe</last>
    </name>
    <office>344</office>
    <salary currency="USD">55000</salary>
  </employee>
  <employee id="902">
    <name>
      <first>Peter</first>
      <last>Pan</last>
    </name>
    <office>216</office>
    <phone>905-416-5004</phone>
  </employee>
</dept>

<dept bldg="114">
  <employee id="903">
    <name>
      <first>Mary</first>
      <last>Jones</last>
    </name>
    <office>415</office>
    <phone>905-403-6112</phone>
    <phone>647-504-4546</phone>
    <salary currency="USD">64000</salary>
  </employee>
</dept>
```

XMLTABLE 関数呼び出しで、行生成 XPath 式を指定し、COLUMNS 文節で 1 つ以上の列生成式を指定します。この例では、行生成式は XPath 式 $d/dept/employee です。パッシング文節は、変数 $d が表 emp の XML 列 doc を参照していることを示します。

```
SELECT X.*
  FROM emp,
       XMLTABLE ('$d/dept/employee' PASSING emp.doc AS "d"
```

```
            COLUMNS
                 empID     INTEGER     PATH '@id',
                 firstname VARCHAR(20) PATH 'name/first',
                 lastname  VARCHAR(25) PATH 'name/last') AS X
```

行生成式は、XML 列の各 XML 文書に適用され、文書ごとに 1 つ以上の employee 要素 (サブツリー)を作成します。 XMLTABLE 関数の出力には、employee 要素ごとに 1 行が含まれます。したがって、行生成 XPath 式によって作成された出力は、SELECT ステートメントの結果セットのカーディナリティーを決定します。

COLUMNS 文節は、XML データをリレーショナル・データに変換するために使用されます。この文節内の各項目は、列名と SQL データ・タイプで列を定義します。上の例では、戻された行には empID、firstname、lastname という名前の 3 つの列があり、データ・タイプはそれぞれ Integer、Varchar(20)、Varchar(25) です。各列の値は、行生成 XPath 式によって作成された employee 要素から抽出され、SQL データ・タイプにキャストされます。例えば、パス name/first は、列 firstname の値を取得するために各 employee 要素に適用されます。行生成式は、列生成式のコンテキストを提供します。つまり、一般的には列生成式を行生成式に追加すると、指定の XMLTABLE 関数が列に何を戻すかを確認できます。

前の照会の結果は、以下のようになります。

```
EMPID       FIRSTNAME           LASTNAME
----------- ------------------- -------------------------
        901 John                Doe
        902 Peter               Pan
        903 Mary                Jones
```

COLUMNS 文節のパス式は、1 行に複数の項目を戻してはならないということに注意してください。パス式が 2 つ以上の項目のシーケンスを戻す場合、一般的に XMLTABLE の実行は失敗します。XML 値のシーケンスを 1 つのアトミック SQL 値に変換できないためです。

例: XMLTABLE を使用した欠落要素の処理

XML データには、すべての文書に存在するわけではないオプションの要素を含めることができます。

例えば、従業員 Peter Pan には salary 要素がありません。これは必須のデータ・フィールドではないためです。 XMLTABLE 関数は欠落要素に NULL 値を作成するので、これを扱うのは容易です。XMLTABLE 照会は、salary 要素が常に存在するものとして作成できます。

```
SELECT X.*
  FROM emp,
       XMLTABLE ('$d/dept/employee' PASSING doc AS "d"
                 COLUMNS
                      empID     INTEGER     PATH '@id',
                      firstname VARCHAR(20) PATH 'name/first',
                      lastname  VARCHAR(25) PATH 'name/last',
                      salary    INTEGER     PATH 'salary') AS X
```

この照会は、以下の結果を戻します。 XML 文書に salary 値がないため、Peter Pan の salary 列には NULL 値が入っているということに注意してください。

```
EMPID       FIRSTNAME           LASTNAME                  SALARY
----------- ------------------- ------------------------- ----------
        901 John                Doe                       55000
        902 Peter               Pan                       -
        903 Mary                Jones                     64000
```

欠落要素に NULL 以外の値を表示したい場合、予期した要素がないときに使用されるデフォルト値を定義できます。ここで、NULL ではなく 0 を戻すように、salary 結果列を定義します。

```
SELECT X.*
  FROM emp,
       XMLTABLE ('$d/dept/employee' PASSING doc AS "d"
                 COLUMNS
                      empID     INTEGER            PATH '@id',
                      firstname VARCHAR(20)        PATH 'name/first',
                      lastname  VARCHAR(25)        PATH 'name/last',
                      salary    INTEGER DEFAULT 0  PATH 'salary') AS X
```

例: XMLTABLE を使用した結果データのサブセット作成

何らかのフィルタリング述部に基づいて、考えられる行のサブセットを含む結果を作成したいことが多くあります。

行のサブセットを作成するには、いくつかの方法があります。 1 つの解決策は、出力列を使用してフィルター処理を行うために WHERE 文節を照会に追加することです。この場合、すべての行が、生成されると直ちに廃棄されることが必要となります。別の解決策では、XMLTABLE 関数の行生成式にフィルタリング述部を使用します。

ビル 114 の従業員に対してのみ行を生成する必要があると想定します。以下のように対応する条件を XMLTABLE に追加できます。

```
SELECT X.*
  FROM emp,
       XMLTABLE ('$d/dept[@bldg="114"]/employee' PASSING doc AS "d"
                 COLUMNS
                      empID     INTEGER            PATH '@id',
                      firstname VARCHAR(20)        PATH 'name/first',
                      lastname  VARCHAR(25)        PATH 'name/last',
                      salary    INTEGER DEFAULT 0  PATH 'salary') AS X
```

この照会では、ビル 114 の唯一の従業員である Mary Jones についての単一行が戻されます。

例: XMLTABLE を使用した複数の値の処理

パス式が複数の値を持つ項目を参照することがあります。

COLUMNS 文節のパス式は、1 行に複数の項目を作成するものであってはなりません。サンプル文書で、従業員 Mary Jones に 2 つの電話番号があることに注目してください。このデータを照会して、各従業員の名前と電話番号を示すリレーショナル表を戻す必要がある場合、作成する照会は以下のようになります。

```
SELECT X.*
  FROM emp,
       XMLTABLE ('$d/dept/employee' PASSING doc AS "d"
                 COLUMNS
                      firstname VARCHAR(20)  PATH 'name/first',
                      lastname  VARCHAR(25)  PATH 'name/last',
                      phone     VARCHAR(12)  PATH 'phone') AS X
```

サンプル文書に対して実行しても、phone に 2 つの値があるため、この照会は失敗します。別の解決策が必要です。

最初の値のみを戻す

この問題に対処する 1 つの方法として、複数の電話番号のうちの 1 つだけを戻すようにします。各従業員の要約情報が必要な場合、1 つの電話番号だけで十分であると思われます。 phone 要素のオカレンスを 1 つだけ戻すには、XPath 式で列 phone に定位置述部を使用できます。

述部を指定するためには、XPath では大括弧が使用されます。従業員の最初の phone 要素を取得するには、定位置述部を [1] または [fn:position()=1] のいずれかとして指定します。最初の表記 [1] は 2 番目の表記を省略したものです。

```
SELECT X.*
  FROM emp,
       XMLTABLE ('$d/dept/employee' PASSING doc AS "d"
                 COLUMNS
                      firstname VARCHAR(20)  PATH 'name/first',
                      lastname  VARCHAR(25)  PATH 'name/last',
                      phone     VARCHAR(12)  PATH 'phone[1]') AS X
```

複数の値を XML として戻す

単一の従業員の複数の電話番号を戻すもう 1 つのオプションは、phone 要素の XML シーケンスを戻すことです。これを行うには、生成される phone 列がタイプ XML である必要があります。これにより、XML 値を XPath 式の結果として戻すことができます。

```
SELECT X.*
  FROM emp,
       XMLTABLE ('$d/dept/employee' PASSING doc AS "d"
                 COLUMNS
                      firstname VARCHAR(20)  PATH 'name/first',
                      lastname  VARCHAR(25)  PATH 'name/last',
                      phone     XML          PATH 'phone') AS X
```

この照会の結果は以下のとおりです。

```
FIRSTNAME    LASTNAME   PHONE
 ----------- ---------- ------------------
John         Doe        -
Peter        Pan        <phone>905-416-5004</phone>
Mary         Jones      <phone>905-403-6112</phone><phone>647-504-4546</phone>
```

Mary Jones の phone 列に戻される XML 値は、単一のルート要素がないため、正しい形式の XML 文書にはなりません。それでもこの値は、DB2 で処理できますが、XML 列に挿入することも、XML パーサーで構文解析することもできません。複数の電話番号を単一の VARCHAR 値または XML 値に結合するには、アプリケーション内に個別の番号を使用するための追加コードが必要になります。

複数の列を戻す

別の解決策として、固定数の結果の phone 列を作成することで、各電話番号を別個の VARCHAR 値として戻すことができます。この例では、定位置述部を使用して、2 つの列に電話番号を戻します。

```
SELECT X.*
  FROM emp,
       XMLTABLE ('$d/dept/employee' PASSING doc AS "d"
                 COLUMNS
                      firstname VARCHAR(20)  PATH 'name/first',
                      lastname  VARCHAR(25)  PATH 'name/last',
                      phone     VARCHAR(12)  PATH 'phone[1]',
                      phone2    VARCHAR(12)  PATH 'phone[2]') AS X
```

この方法の明らかな欠点は、可変数の項目が固定数の列にマップされるということです。ある従業員が予想を超える電話番号を持っていることが考えられます。また、他の従業員の電話番号は予想より少なく、結果が NULL 値になることも考えられます。すべての従業員が勤務先電話と携帯電話をそれぞれ 1 台だけ持っている場合、対応する名前を持つ 2 つの列を作成すると非常に有効です。

各値に 1 行を戻す

各電話番号を個別の列に戻す代わりに、XMLTABLE を使用して個別の行に戻すこともできます。この場合、各従業員に 1 行ではなく、各電話番号に 1 行を戻す必要があります。この結果、名と姓の列に情報が反復される場合があります。

```
SELECT X.*
  FROM emp,
       XMLTABLE ('$d/dept/employee/phone' PASSING doc AS "d"
                 COLUMNS
                      firstname VARCHAR(20)  PATH '../name/first',
                      lastname  VARCHAR(25)  PATH '../name/last',
                      phone     VARCHAR(12)  PATH '.') AS X
```

この照会の結果は以下のとおりです。

```
FIRSTNAME    LASTNAME   PHONE
 ----------- ---------- ------------------
Peter        Pan        905-416-5004
Mary         Jones      905-403-6112
Mary         Jones      647-504-4546
```

この結果では、電話番号を持たない John Doe に対応する行はありません。

存在しないパス値の処理

前の例では、行の Xquery 式がすべての phone 要素を対象として反復され、従業員 John Doe には phone 要素がないため、従業員 John Doe に対応する行は戻されませんでした。結果として、John Doe の employee 要素は処理されることはありません。

この問題を解決するには、2 つの XMLTABLE 関数の SQL UNION を使用する必要があります。

```
SELECT X.*
  FROM emp,
       XMLTABLE ('$d/dept/employee/phone' PASSING doc AS "d"
                 COLUMNS
                      firstname VARCHAR(20)  PATH '../name/first',
                      lastname  VARCHAR(25)  PATH '../name/last',
                      phone     VARCHAR(12)  PATH '.') AS X
UNION
SELECT Y.*, CAST(NULL AS VARCHAR(12))
  FROM emp,
       XMLTABLE ('$d/dept/employee[fn:not(phone)]' PASSING doc AS "d"
                 COLUMNS
                      firstname VARCHAR(20)  PATH 'name/first',
                      lastname  VARCHAR(25)  PATH 'name/last') AS Y
```

2 番目の XMLTABLE の `$d/dept/employee[fn:not(phone)]` 行式は、電話番号のないすべての従業員を戻し、最初の XMLTABLE で除外された employee 行を追加します。

例: 名前空間での XMLTABLE の使用

XML 名前空間は、XML 文書内で固有の名前を持つ要素および属性を提供するための W3C XML 標準です。 XML 文書には、異なるボキャブラリーに由来するが同じ名前を持つ要素および属性が含まれている場合があります。各ボキャブラリーに名前空間を割り当てることで、同一の要素名または属性名の間のあいまいさが解決されます。

XML 文書では、予約属性 `xmlns` で XML 名前空間を宣言します。その値には Universal Resource Identifier (URI) が含まれている必要があります。 URI は ID として使用されます。この ID は一般的に URL と類似していますが、存在する Web ページを指す必要はありません。名前空間宣言には、要素および属性の識別に使用される接頭部を含めることもできます。以下は、接頭部がある場合とない場合の名前空間宣言の例です。

```
xmlns:ibm = "http://www.ibm.com/xmltable/"
xmlns = "http://www.ibm.com/xmltable/"
```

XMLTABLE での名前空間の使用法を示すために、サンプル文書が前の例に追加されています。そこで、以下の 3 つの行を処理することになります。

```
<dept bldg="101">
  <employee id="901">
    <name>
      <first>John</first>
      <last>Doe</last>
    </name>
    <office>344</office>
    <salary currency="USD">55000</salary>
  </employee>
  <employee id="902">
    <name>
      <first>Peter</first>
      <last>Pan</last>
    </name>
    <office>216</office>
    <phone>905-416-5004</phone>
  </employee>
</dept>

<dept bldg="114">
  <employee id="903">
    <name>
      <first>Mary</first>
      <last>Jones</last>
    </name>
    <office>415</office>
    <phone>905-403-6112</phone>
    <phone>647-504-4546</phone>
    <salary currency="USD">64000</salary>
  </employee>
</dept>

<ibm:dept bldg="123" xmlns:ibm="http://www.ibm.com/xmltable">
  <ibm:employee id="144">
    <ibm:name>
      <ibm:first>James</ibm:first>
      <ibm:last>Bond</ibm:last>
    </ibm:name>
    <ibm:office>007</ibm:office>
    <ibm:phone>905-007-1007</ibm:phone>
    <ibm:salary currency="USD">77007</ibm:salary>
  </ibm:employee>
</ibm:dept>
```

データベース内のすべての従業員を戻すために、パス式内の名前空間接頭部に * ワイルドカードを使用できます。このワイルドカード (*) はすべての名前空間に一致する (名前空間がないものも含む) ため、これにより名前空間に関係なくすべての要素が対象となります。

```
SELECT X.*
  FROM emp,
       XMLTABLE ('$d/*:dept/*:employee' PASSING doc AS "d"
                 COLUMNS
                      empID     INTEGER     PATH '@*:id',
                      firstname VARCHAR(20) PATH '*:name/*:first',
                      lastname  VARCHAR(25) PATH '*:name/*:last') AS X
```

照会の結果は、以下のようになります。

```
EMPID       FIRSTNAME            LASTNAME
----------- -------------------- ------------------------
        901 John                 Doe
        902 Peter                Pan
        903 Mary                 Jones
        144 James                Bond
```

この特定のデータの場合、属性 @id の名前空間ワイルドカードは、厳密には必要ありませんでした。従業員 James Bond の @id 属性に名前空間がないことが、その理由です。属性はその要素から名前空間を継承することはなく、またデフォルトの名前空間を想定することもありません。このため、属性名は、接頭部を持たない限り、どの名前空間にも属しません。

ワイルドカード式を使用するのは、名前空間に関係なくすべての従業員を戻す最も簡単な方法です。

デフォルトの要素名前空間の宣言

照会したいすべての要素が同じ名前空間に属している場合は、デフォルトの要素名前空間を宣言することが照会を作成する最も簡単な方法です。 XPath 式の先頭にデフォルトの名前空間を宣言するだけでかまいません。その後、参照されるすべての未修飾の要素はこの名前空間に結び付けられます。

```
SELECT X.*
  FROM emp,
       XMLTABLE ('declare default element namespace "http://www.ibm.com/xmltable";
                  $d/dept/employee' PASSING doc AS "d"
                  COLUMNS
                       empID     INTEGER     PATH '@id',
                       firstname VARCHAR(20) PATH
                  'declare default element namespace "http://www.ibm.com/xmltable"; name/first',
                       lastname  VARCHAR(25) PATH
                  'declare default element namespace "http://www.ibm.com/xmltable"; name/last') AS X
```

結果は、以下のようになります。

```
EMPID       FIRSTNAME            LASTNAME
----------- -------------------- ------------------------
        144 James                Bond
```

列生成式は、行生成式から名前空間宣言を継承しません。各列生成式は別個の XPath 照会であり、その独自の名前空間宣言を必要とします。例えば、文書に複数の名前空間が含まれる場合、こうした名前空間宣言は、それぞれ異なることがあります。

一般的には名前空間は 1 つしかありません。その場合には、単一の名前空間を、XMLTABLE 関数内のすべての式に対して宣言すると便利です。これは、関数 XMLNAMESPACES() を使用して実行できます。この関数により、デフォルトの要素名前空間を宣言したり、XMLTABLE 関数内で複数の名前空間接頭部を使用したり、あるいはその両方を行うことができます。 XMLNAMESPACES 関数を使用する利点は、宣言された名前空間が XMLTABLE コンテキスト内のすべての式にグローバルなものになり、これによりすべての XPath 式がこの名前空間宣言を認識するので、名前空間宣言を繰り返す必要がないということです。

XMLNAMESPACES 関数によって宣言されたデフォルトの名前空間は、行生成式とすべての列生成式の両方に適用されます。このように、XMLTABLE 関数内のすべての XPath 式には名前空間宣言が 1 つしか必要ありません。以下の照会の結果は、前の例とまったく同じになります。

```
SELECT X.*
  FROM emp,
       XMLTABLE (XMLNAMESPACES(DEFAULT 'http://www.ibm.com/xmltable'),
                  '$d/dept/employee' PASSING doc AS "d"
                  COLUMNS
                       empID     INTEGER     PATH '@id',
                       firstname VARCHAR(20) PATH 'name/first',
                       lastname  VARCHAR(25) PATH 'name/last') AS X
```

XMLNAMESPACES による名前空間接頭部の宣言

複数の特定の名前空間から要素および属性を選択したい場合、名前空間接頭部を使用することが最善の方法であると考えられます。XMLNAMESPACES 関数を使用しない限り、すべての式に名前空間接頭部を宣言する必要があります。しかし、デフォルトの要素名前空間の場合と同様に、XMLNAMESPACES 関数を使用すると、名前空間宣言の繰り返しを避けることができます。

```
SELECT X.*
  FROM emp,
       XMLTABLE (XMLNAMESPACES('http://www.ibm.com/xmltable' AS "ibm"),
                 '$d/ibm:dept/ibm:employee' PASSING doc AS "d"
                 COLUMNS
                      empID     INTEGER     PATH '@id',
                      firstname VARCHAR(20) PATH 'ibm:name/ibm:first',
                      lastname  VARCHAR(25) PATH 'ibm:name/ibm:last') AS X
```

例: XMLTABLE の結果行の番号付け

場合によっては、特定の文書に対して XMLTABLE で生成される行の番号付けを行う列を生成したいことがあります。これは、各文書での値の順序をアプリケーションが記憶する上で役立ちます。

結果行の番号付けを行うには、FOR ORDINALITY 文節を使用します。 XMLTABLE 関数への入力である文書ごとに、番号付けは 1 から始まるということに注意してください。

```
SELECT X.*
  FROM emp,
       XMLTABLE ('$d/dept/employee' PASSING doc AS "d"
                 COLUMNS
                      seqno     FOR ORDINALITY,
                      empID     INTEGER     PATH '@id',
                      firstname VARCHAR(20) PATH 'name/first',
                      lastname  VARCHAR(25) PATH 'name/last') AS X
```

照会の結果は、以下のようになります。

```
SEQNO    EMPID       FIRSTNAME           LASTNAME
-------- ----------- ------------------- -------------------------
       1         901 John                Doe
       2         902 Peter               Pan
       1         903 Mary                Jones
```

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。特定の行を更新するときには、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、またはバイナリーのいずれかのタイプとなります。

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。

以下の例は、XML 列で XML データを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。この例では、カスタマー ID 値が 1004 である行が既に 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、その内容は次のようであると想定します。

```
<customerinfo xmlns="http://posample.org" 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 データをバイナリー・データとしてファイル c7.xml から読み取り、それを使用して XML 列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;
```

これらの例では、<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。 UPDATE ステートメントの 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。

表からの XML データの削除

XML 文書を格納する行を削除するには、SQL DELETE ステートメントを使用します。特定の行を削除するときは、WHERE 節を組み込みます。

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。行を削除せずに 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 ステートメントを使用して、その列を NULL に設定します。

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
```

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 スキーマに関する情報を格納する表集合です。

XML インスタンス文書には、関連付けられた XML スキーマを指す URI (Uniform Resource Identifier) への参照が通常含まれています。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、そうした外部に参照される XML 成果物上の従属関係を管理します。これは XSR を使用して行われるため、URI ロケーション参照を変更する必要はありません。

関連付けられた XML スキーマ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。

XML スキーマは、一連の XML スキーマ文書で構成されています。 XML スキーマを DB2 XSR に追加するには、以下の DB2 提供ストアード・プロシージャーを呼び出すことにより、 XML スキーマ文書を DB2 に登録します。

SYSPROC.XSR_REGISTER
XML スキーマの登録を開始します。 XML スキーマに最初の XML スキーマ文書を追加する際に、このストアード・プロシージャーを呼び出します。

```
CALL SYSPROC.XSR_REGISTER ('user1', 'POschema',
                   'http://myPOschema/PO',
                   :content_host_var, NULL)
```

SYSPROC.XSR_ADDSCHEMADOC
登録処理中の XML スキーマに、さらに XML スキーマ文書を追加します。
SYSPROC.XSR_ADDSCHEMADOC は、まだ完了していない既存の XML スキーマに対してのみ呼び出すことができます。

```
CALL SYSPROC.XSR_ADDSCHEMADOC ('user1', 'POschema',
                   'http://myPOschema/address',
                   :content_host_var, NULL)
```

SYSPROC.XSR_COMPLETE
XML スキーマの登録を完了します。

```
CALL SYSPROC.XSR_COMPLETE ('user1', 'POschema', :schemaproperty_host_var, 0)
```

XML スキーマ登録完了時に、DB2 は、XML スキーマ文書内部にある、他の XML スキーマ文書への参照を解決します。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するときにのみ行われます。

DB2 XML スキーマ・リポジトリーから XML スキーマを除去するには、以下のいずれかの方法で行います。

- SYSPROC.XSR_REMOVE ストアード・プロシージャーを呼び出す。
- DROP XSROBJECT SQL ステートメントを使用する。

XML スキーマ・リポジトリー (XSR) の独立 ASP に関する考慮事項

複数のシステム間で独立補助記憶域プール (ASP) を切り替えることができるので、ASP で XML スキーマを管理する際にはいくつかの追加の考慮事項があります。

XML スキーマを登録した独立 ASP で、それを使用するように設定する必要があります。ジョブが独立 ASP に接続されているとき、別の独立 ASP グループまたはシステム ASP で定義された XML スキーマを参照することはできません。

アプリケーション・プログラミング言語のサポート

XML データを DB2 データベース表に保管したり、表からデータを取得したり、あるいは XML パラメーターを持つストアード・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できます。

次のいずれかの言語を使用して、アプリケーションを作成できます。
- ILE RPG
- ILE COBOL
- C および C++ (組み込み SQL または DB2 CLI)
- Java (JDBC または SQLJ)

アプリケーション・プログラムは、XML 列に保管された文書全体を取り出すことができます。

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 の直列化されたストリング・フォーマットから、指定の CCSID の XML 値に変換します。

アプリケーションが XML 列からデータを取り出すとき、DB2 データベース・サーバーは、データを指定の CCSID の XML 値から XML の直列化されたストリング・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 XML の CCSID からストリングの CCSID に変換する必要がある場合もあります。

XML データを取り出すときには、CCSID 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 XML の CCSID の文字をターゲット・ストリングの CCSID で表せない場合に生じることがあります。

XML 文書をフェッチするときには、直列化された形式の文書をアプリケーション変数として取り出します。

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場合は、入力データは直列化されたストリング・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します。

SQL XML データ・タイプは、次のアプリケーション C 文字およびグラフィックのデータ・タイプにバインドできます。
- SQL_C_CHAR
- SQL_VARCHAR
- SQL_C_WCHAR
- SQL_VARGRAPHIC

次の文字 LOB データ・タイプです。
- SQL_C_CLOB
- SQL_C_CLOB_LOCATOR

そして、次のバイナリー・データ・タイプです。
- SQL_C_BINARY
- SQL_C_BLOB
- SQL_C_BLOB_LOCATOR
- SQL_C_BINARY

バイナリー・データ・タイプの XML データを格納するデータ・バッファーをバインドするとき、 DB2 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。これにより、文字タイプが使用された場合の文字変換の際のオーバーヘッドとデータ損失の可能性を回避できるので、これは好ましい方法です。

注: XML データをコード化スキームの異なる多数のソースから受け取るとき、XML をバイナリー・データ・タイプにバインドする必要があります。

SQL_C_CHAR または SQL_C_WCHAR の XML データを格納するデータ・バッファーをバインドするとき、 DB2 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。 DB2 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 である場合、DB2 CLI はデータが UTF-16 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 である場合、DB2 CLI はデータがアプリケーションの 1 バイトのデフォルト CCSID でエンコードされていると想定します。

データベース・サーバーでデータを暗黙的に構文解析してから XML 列に保管するようにするには、SQLBindParameter() 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。

次の例は、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 = &apos;001&apos;", SQL_NTS);
SQLBindParameter (hStmt, 1, SQL_PARAM_INPUT, SQL_C_BINARY, SQL_XML, 0, 0,
                  xmlBuffer, 10240, &length);
SQLExecute (hStmt);
```

CLI アプリケーションでの XML データの取り出し

表の XML 列からデータを選択するとき、出力データは直列化されたストリング・フォーマットになります。

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 C 文字、グラフィック、文字、グラフィックの LOB、およびバイナリーの各データ・タイプを指定できます。 XML 列から結果セットを取り出すとき、アプリケーション変数をバイナリー形式にバインドすることを考慮してください。文字タイプにバインドすると、CCSID 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 XML の CCSID の文字をターゲット・ストリングの CCSID で表せない場合に生じることがあります。変数をバイナリー形式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返されます。 DB2 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 である場合、DB2 CLI はデータを XML 値のコード化スキームで返します。
- C タイプが SQL_C_CHAR である場合、DB2 CLI はデータをアプリケーション文字のコード化スキームで返します。

- C タイプが SQL_C_WCHAR である場合、DB2 CLI はデータを UTF-16 コード化スキームで返します。

データベース・サーバーは、データを暗黙的に直列化してからアプリケーションに返します。
XMLSERIALIZE 関数を呼び出すことにより、XML データを特定のデータ・タイプに明示的に直列化することもできます。ただし、XMLSERIALIZE で文字タイプに明示的に直列化するとエンコードの問題が生じることがあるので、暗黙的な直列化が推奨されています。

次の例は、XML データを XML 列から取り出してバイナリー・アプリケーション変数に入れ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=&apos;001&apos;", 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
```

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。

XML データは、表の XML データ・タイプ列に保管されます。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 または SQL/XML 拡張を使用して直接アクセスできます。 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。

XML データは本来、文字であり、使用される文字セットを指定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 文書を直列化したストリングで表現したもので構成される基本アプリケーション・タイプから導き出すことにより、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では、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、DBCLOB、BLOB といった、既存の文字、グラフィック、およびバイナリーの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 関数が適用されます。

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、アプリケーションの宣言セクションで、XML ホスト変数を LOB データ・タイプとして宣言します。ここには C の場合の例を示しますが、サポートされる他の言語についても、類似の構文が存在します。

- SQL TYPE IS XML AS CLOB(n) <hostvar_name> は、 SQL_XML_DATA_CCSID QAQQINI ファイル・オプションで指定する CCSID でエンコードされた XML データを格納する CLOB ホスト変数を定義します。
- SQL TYPE IS XML AS DBCLOB(n) <hostvar_name> は、XML データを格納する DBCLOB ホスト変数を定義します。オプションが UCS-2 または UTF-16 の場合、SQL_XML_DATA_CCSID QAQQINI ファイル・オプションで指定した CCSID でエンコードされます。その他の場合、デフォルトの CCSID は UTF-16 です。
- SQL TYPE IS XML AS BLOB(n) <hostvar_name> は、内部エンコード XML データを格納する BLOB ホスト変数を定義します。
- SQL TYPE IS XML AS LOCATOR <hostvar_name> は、XML データを格納するロケーターを定義します。
- SQL TYPE IS XML AS CLOB_FILE <hostvar_name> は、ファイルの CCSID でエンコードされた XML データを格納する CLOB ファイルを定義します。
- SQL TYPE IS XML AS DBCLOB_FILE <hostvar_name> は、アプリケーションのデフォルトの 2 バイト CCSID でエンコードされた XML データを格納する DBCLOB ファイルを定義します。
- SQL TYPE IS XML AS BLOB_FILE <hostvar_name> は、内部エンコード XML データを格納する BLOB ファイルを定義します。

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。

C の組み込み SQL アプリケーション

```
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;
// using XML AS CLOB host variable
// The XML value written to xmlBuf will be prefixed by an XML declaration
// similar to: <?xml version = "1.0" encoding = "UTF-8"?>
// Note: The encoding name will depend upon the SQL_XML_DATA_CCSID QAQQINI setting
EXEC SQL SELECT xmlCol INTO :xmlBuf
  FROM myTable
  WHERE id = '001';
EXEC SQL UPDATE myTable
  SET xmlCol = :xmlBuf
  WHERE id = '001';

// using XML AS BLOB host variable
// 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';

// using CLOB host variable
// The output will be encoded in the application single byte default CCSID,
// 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';
```

COBOL の組み込み SQL アプリケーション

```
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 XML AS BLOB(5K).
EXEC SQL END DECLARE SECTION END-EXEC.

* using XML AS CLOB host variable
EXEC SQL SELECT xmlCol INTO :xmlBuf
  FROM myTable
  WHERE id = '001'.
EXEC SQL UPDATE myTable
  SET xmlCol = :xmlBuf
  WHERE id = '001'.

* using XML AS BLOB host variable
EXEC SQL SELECT xmlCol INTO :xmlblob
  FROM myTable
  WHERE id = '001'.
EXEC SQL UPDATE myTable
  SET xmlCol = :xmlblob
  WHERE id = '001'.

* using CLOB host variable
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'.
```

XML 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 を使用するとき、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、日時データなどを含め、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 データを持つ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・サーバー要求が生成される)。
- 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、ホスト変数の CCSID 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ければならない。
- XML ホスト変数の基本タイプとしては、LOB データ・タイプを指定しなければならない。
- 静的 SQL を使用するとき、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、DBCLOB、および BLOB の各ホスト変数を使用すると、それらは、デフォルトの空白処理特性 ('STRIP WHITESPACE') を伴う XMLPARSE 操作の対象になる。他の非 XML ホスト変数タイプはすべて拒否されます。

SQLDA 内の 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 フィールドを更新する必要があります。

- `sqlname.length` は 8 でなければなりません
- sqlname.data の最初の 2 バイトは X'0000' でなければなりません
- sqlname.data の 3 番目と 4 番目のバイトは X'0000' でなければなりません
- sqlname.data の 5 番目のバイトは X'01' でなければなりません (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ます)
- 残りのバイトは X'000000' でなければなりません

SQLTYPE が非 LOB である SQLVAR で XML サブタイプ標識を設定すると、実行時に SQL0804 エラーが返されます。

注: SQL_TYP_XML は DESCRIBE ステートメントからのみ返せます。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。アプリケーションでは、SQLDA を変更して、有効な文字タイプまたはバイナリー・タイプを格納するようにする必要があります。さらに sqlname フィールドを適切に設定して、データが XML であることを示すようにする必要があります。

Java

Java および XML

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 列からのデータの取り出しが可能です。

データベース表では、XML 組込みデータ型を使用して XML データを列に保管します。

アプリケーションでは、XML データは直列化されたストリング・フォーマットになります。

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。

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` が返されます。

JDBC アプリケーションでの XML 列の更新:

データベース表の XML 列でデータを更新または挿入する場合は、JDBC アプリケーション内の入力データは直列化されたストリング・フォーマットでなければなりません。

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 2. 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

表 2. *XML 列の更新用のメソッドとデータ・タイプ (続き)*

メソッド	入力データ・タイプ
PreparedStatement.setBinaryStream	InputStream
PreparedStatement.setBlob	Blob
PreparedStatement.setBytes	byte[]
PreparedStatement.setCharacterStream	Reader
PreparedStatement.setClob	Clob
PreparedStatement.setObject	byte[]、Blob、Clob、SQLXML、InputStream、Reader、String
PreparedStatement.setString	String

XML データのエンコード方式は、データ自体から導出する (*内部エンコード・データという*) か、または外部ソースから導出する (*外部エンコード・データという*) ことができます。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、内部エンコード・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、外部エンコード・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。

外部エンコード・データには、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。内部エンコード方式と外部エンコード方式との間に互換性がない場合、データ・ソースはエラーを生成して、内部エンコード方式と外部エンコード方式の非互換性を処理します。

XML 列のデータは、XML 列の CCSID 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 XML 列の CCSID へのデータの変換が処理されます。

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 String 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"" xmlns=""http://posample.org"">...</customerinfo>";
   cid.setString(cidData);
                // 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());
   }
}
```

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()
{
  PreparedStatement insertStmt = null;
  String sqls = null;
  int cid = 0;
  ResultSet rs=null;
  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());
  }
}
```

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。

DB2 表の XML 列からデータを取り出す場合、出力データは直列化されたストリング・フォーマットになります。

XML データを取り出すには、以下のいずれかの手法を使用できます。

- 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、互換性のある出力データ・タイプでデータを取り出します。 SQLXML.getBinaryStream メソッドで、エンコード方式を指定した XML 宣言を出力データに追加します。 SQLXML.getString および SQLXML.getCharacterStream メソッドは、XML 宣言を追加しません。
- 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・タイプでデータを取り出します。

次の表には、XML データを取り出すための ResultSet メソッドと、対応する出力データ・タイプがリストされています。

表 3. 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.getSQLXML	SQLXML
ResultSet.getString	String

次の表には、java.sql.SQLXML オブジェクトからデータを取り出すために呼び出すことのできるメソッド、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・タイプがリストされています。

表 4. 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指定

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	XML 列の CCSID エンコード方式
SQLXML.getCharacterStream	Reader	なし
SQLXML.getSource	ソース	なし
SQLXML.getString	String	なし

返される予定の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、関数実行後のデータは、XML データ・タイプではなく XMLSERIALIZE 関数で指定したデータ・タイプになります。そのため、ドライバーは指定タイプで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。

以下の例は、データを SQLXML オブジェクトとして XML 列から取り出してから、SQLXML.getString メソッドを使用してデータをストリングとして取り出す方法を示しています。

```
public void fetchToSQLXML()
  {
    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();
    String 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());
    }
}
```

以下の例は、データをストリング変数として XML 列から取り出す方法を示しています。

```
public void fetchToString()
  {
    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();
      String 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());
   }
}
```

Java アプリケーションでの XML パラメーターを持つルーチンの呼び出し:

SQL ストアード・プロシージャー、外部ストアード・プロシージャー、および外部ユーザー定義関数には、XML パラメーターを組み込むことができます。

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それらのパラメーターは XML タイプです。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 タイプです。 XML パラメーターを持つストアード・プロシージャーまたはユーザー定義関数を呼び出すとき、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。

XML 入力パラメーターを持つルーチンを JDBC プログラムから呼び出すには、java.sql.SQLXML 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 タイプで登録します。

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 JDBC プログラム。この例では JDBC 4.0 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
                               // input/output XML parameter

Connection con;
CallableStatement cstmt;
ResultSet rs;
...
stmt = 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.
```

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 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
                              // input/output 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 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 列からのデータを取り出しが可能です。

DB2 表では、XML 組込みデータ型を使用して XML データを列に保管します。

アプリケーションでは、XML データは直列化されたストリング・フォーマットになります。

SQLJ アプリケーションでは以下を行うことができます。

- INSERT または UPDATE 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` が返されます。

SQLJ アプリケーションでの XML 列の更新:

表の XML 列において SQLJ アプリケーションでデータを更新または挿入する場合は、入力データは直列化されたストリング・フォーマットでなければなりません。

XML 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 (SQLJ バージョン 4.0 以降が必要)
- String
- byte
- Blob
- Clob
- sqlj.runtime.AsciiStream
- sqlj.runtime.BinaryStream
- sqlj.runtime.CharacterStream

ストリーム・タイプの場合、ストリームの長さを JDBC ドライバーに受け渡すことができるように、java.io.typeInputStream ホスト式ではなく、 sqlj.runtime.*type*Stream ホスト式を使用する必要があります。

XML データのエンコード方式は、データ自体から導出する (*内部エンコード・データという*) か、または外部ソースから導出する (*外部エンコード・データという*) ことができます。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、内部エンコード・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、外部エンコード・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。

外部エンコード・データには、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。内部エンコード方式と外部エンコード方式との間に互換性がない場合、データ・ソースはエラーを生成して、内部エンコード方式と外部エンコード方式の非互換性を処理します。

XML 列のデータは、XML 列の CCSID で保管されます。

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 は文字タイプであるため、その外部エンコード方式が使用されます。

```
#sql [ctx] {INSERT INTO CUSTACC VALUES (1, :xmlString)};
```

sqlj.runtime.CharacterStream ホスト式: sqlj.runtime.CharacterStream ホスト式を構成して、データを sqlj.runtime.CharacterStream host ホスト式から表の XML 列に挿入するとします。

```
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 は文字タイプであるため、その外部エンコード方式が使用されます。

文書を XML 列から java.sql.SQLXML ホスト式として取り出して、このデータを表の XML 列に挿入するとします。

```
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もこのデータは UTF-8 エンコード方式のままであるため、このデータを別の XML 列に挿入する場合で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、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 (SQLJ バージョン 4.0)
- 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 タイプまたは com.ibm.db2.jcc.DB2Xml タイプのオブジェクトである場合は、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が決まります。さらに、エンコード方式を指定した XML 宣言が追加されるかどうかも決まります。

次の表には、java.sql.SQLXML または com.ibm.db2.jcc.DB2Xml オブジェクトからデータを取り出すために呼び出すことのできるメソッド、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・タイプがリストされています。

表 5. 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指定

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	XML 列の CCSID エンコード方式
SQLXML.getCharacterStream	Reader	なし
SQLXML.getSource	ソース	なし
SQLXML.getString	String	なし

返される予定の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、関数実行後のデータは、XML データ・タイプではなく XMLSERIALIZE 関数で指定したデータ・タイプになります。そのため、ドライバーは指定タイプで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。

データを XML 列から String ホスト式として取り出すとします。

```
#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、XML 宣言なしで返されます。

データを XML 列から byte[] ホスト式として取り出すとします。

```
#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形式であるため、 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。

文書を XML 列から java.sql.SQLXML ホスト式として取り出しますが、バイナリー・ストリームのデータが必要であるとします。

```
#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 オブジェクトとして取り出されます。 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。

ルーチン

ルーチンおよび XML

SQL プロシージャーでの XML 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。それらは、他のデータ・タイプの変数と同様に、SQL ステートメントで使用できます。

次の例では、SQL プロシージャーにおける 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ます。

```
CREATE TABLE T1(C1 XML) ;

CREATE PROCEDURE proc1(IN parm1 XML, IN parm2 VARCHAR(32000))
LANGUAGE SQL
BEGIN
   DECLARE var1 XML;


     /* insert the value of parm1 into table T1 */
     INSERT INTO T1 VALUES(parm1);

   /* 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 ;
```

上の例には 1 つの XML 列を持つ表 T1 があります。 SQL プロシージャーは、parm1 および parm2 という 2 つのパラメーターを受け入れます。 parm1 は XML データ・タイプです。 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。

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。

SQL プロシージャー内の XML パラメーターおよび XML 変数の値に対するコミットとロールバックの影響:

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中、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使用できなくなります。

コミットまたはロールバックの操作後に、データ・タイプが 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、エラーが出されます。

コミットまたはロールバックの操作発生後に XML パラメーターおよび XML 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 ステートメントおよび 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 XML 変数の値を使用できるかどうかをご検討ください。

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作成された外部プロシージャーおよび外部関数は、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。

これらのプログラミング言語には、以下のものがあります。

- ILE RPG

- ILE COBOL
- C
- C++
- Java

XML データ・タイプの値は、文字、グラフィック、およびバイナリーの各データ・タイプと同じ方法で外部ルーチンのコードに示されます。

データ・タイプが 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・タイプを文字、グラフィック、またはバイナリーのいずれかのデータ・タイプで保管することを指定する必要があります。文字、グラフィック、またはバイナリーの値のサイズは、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。

以下の 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実装される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';
```

以下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'
```

外部ルーチンのコード内部では、XML パラメーターおよび XML 変数の値へのアクセス、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方法で行われます。

例: Java (JDBC) プロシージャーでの XML サポート:

Java プロシージャーの基本である、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

Java 外部コード・ファイル

例では、Java プロシージャーの実装を示します。例は、CREATE PROCEDURE ステートメントと、関連 Java クラスの作成元プロシージャーの外部 Java コード実装という 2 つの部分で構成されています。

以下の例のプロシージャー実装を含んだ 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。

Java 外部コード・ファイルの形式

```
using System;
import java.lang.*;
import java.io.*;
import java.sql.*;
import java.util.*;

  public class stpclass
  {      ...
   // Java procedure implementations
         ...
  }
```

クラス・ファイルの名前、および特定のプロシージャー実装を含んだ JAR の名前をメモしておくことは重要です。これらの名前が重要である理由は、DB2 が実行時にプロシージャーのクラスを見つけられるようにするために、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する必要があるからです。

例: 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、次のものを示します。

- 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
- 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、その値を含んだ行を表に挿入し、 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。

XML パラメーターを使用するパラメーター・スタイル JAVA プロシージャーを作成するためのコード

```
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 'stpclass.xmlProc1';
```

```
public void xmlProc1(int inNum,
                     CLOB  inXML ,
                     CLOB [] out1XML,
                    )
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.getCharacterStream() ;
```

```
     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] = (CLOB) rs.getObject(1); }

     rs.close() ;
     stmt.close();

     return ;
}
```

例: C プロシージャーでの XML サポート:

プロシージャーの基本である、C ルーチンおよび XML の要点を理解したなら、XML フィーチャーを持つ 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。

以下の例では、XML データの更新および照会の方法に加えて、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。

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、次のものを示します。

- 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受け取ります。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。検索された XML 値は out1XML パラメーターに割り当てられます。結果セットは返されません。

```
CREATE PROCEDURE xmlProc1 ( IN inNUM INTEGER,
                            IN inXML XML as CLOB (1K),
                            OUT out1XML XML as CLOB (1K)
                          )
LANGUAGE C
PARAMETER STYLE SQL
DYNAMIC RESULT SETS 0
DETERMINISTIC
MODIFIES SQL DATA
PROGRAM TYPE SUB
EXTERNAL NAME '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 SQL
//****************************************************************************
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <sqlda.h>
#include <sqlca.h>
#include <sqludf.h>
#include <sql.h>
#include <memory.h>

#ifdef __cplusplus
extern "C"
#endif
SQL_API_RC SQL_API_FN testSecA1(sqlint32* inNum,
                                SQLUDF_CLOB* inXML,
                                SQLUDF_CLOB* out1XML,
                                SQLUDF_NULLIND *inNum_ind,
                                SQLUDF_NULLIND *inXML_ind,
                                SQLUDF_NULLIND *out1XML_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;
 EXEC SQL END DECLARE SECTION;

 /* Check null indicators for input parameters */
 if ((*inNum_ind &lt; 0) || (*inXML_ind &lt; 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;

 exit:

 /* Set output return code */
 *outReturnCode = sqlca.sqlcode;
 *outReturnCode_ind = 0;

 return 0;
}
```

XML データ・エンコード方式

XML データのエンコード方式は、データ自体から導出する (*内部エンコード・データという*) か、または外部ソースから導出する (*外部エンコード・データという*) ことができます。

アプリケーションと XML 列の間での XML データの交換にどのアプリケーション・データ・タイプを使用するかにより、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である 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、ホスト変数で宣言した CCSID 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 文書にエンコード方式の宣言が含まれている場合などには、外部的にコード化された XML データに内部エンコード方式が含まれることがあります。外部エンコード・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の有無を検査します。

内部エンコード方式に関連付けられた有効な 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。

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 データを DB2 表に保管するには、それを正しくエンコードする必要があります。データが表から取り出され、DB2 ストアード・プロシージャーまたはユーザー定義関数で使用される場合、あるいは外部 Java アプリケーションで使用される場合、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 表に保管する際には、内部エンコード方式および外部エンコード方式を検討する必要があります。

以下の規則を守る必要があります。

- 外部エンコード 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、内部エンコード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。
- 内部エンコード 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で確認する必要があります。

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・タイプを変換した結果、データが拡張すると生じます。

ルーチン・パラメーターでの 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に含まれるパラメーターにおいて、複数の XML データ・タイプを使用できます。

以下の XML データ・タイプを使用できます。

XML　SQL プロシージャーおよび関数の場合。

XML AS
外部プロシージャーおよび外部ユーザー定義関数の場合。

XML AS パラメーター内のデータは文字変換の対象になります。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 CCSID 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、データ破壊が起こる可能性があります。

JDBC および SQLJ 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。内部エンコード 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エンコード・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の CCSID 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、それをバイナリー・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エンコード・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 (アプリケーションの CCSID) から XML 列の CCSID 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 でない可能性があるため、ストリング・データはファイルに保管されるときに変換される可能性があります。

  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。
- 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 エンコード方式を使って、バイナリー・アプリケーション変数で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直列化の影響

XML データのエンコード方式を指定する方法 (内部または外部のいずれか)、および XML 直列化の方法は、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。

内部エンコード 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 列への XML データの入力中に、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

エンコードするソース	値
データ・エンコード	UTF-8 Unicode 入力データ (UTF-8 BOM または XML エンコード宣言が含まれる場合も含まれない場合もある)
ホスト変数データ・タイプ	2 進数
CCSID を宣言したホスト変数	適用されない

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

エンコードするソース	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んだ UTF-16 Unicode 入力データ
ホスト変数データ・タイプ	2 進数
CCSID を宣言したホスト変数	適用されない

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 構文解析を実行して UTF-8 XML 列に保管するときに、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

エンコードするソース	値
データ・エンコード	XML エンコードの宣言を含んだ ISO-8859-1 入力データ

エンコードするソース	値
ホスト変数データ・タイプ	2 進数
CCSID を宣言したホスト変数	適用されない

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 構文解析を実行して UTF-8 XML 列に保管するときに、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

エンコードするソース	値
データ・エンコード	XML エンコードの宣言を含んだ Shift_JIS 入力データ
ホスト変数データ・タイプ	2 進数
CCSID を宣言したホスト変数	適用されない

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 構文解析を実行して UTF-8 XML 列に保管するときに、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。

外部エンコード 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 列への XML データの入力中に、外部エンコード方式がデータの変換や切り捨てにどのように影響するかを示しています。

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 CCSID 変換に関する問題は生じません。

シナリオ 1 とシナリオ 2 だけが Java に適用されます。これは、Java の文字ストリングが常に Unicode であるために、Java 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

エンコードするソース	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宣言または BOM が含まれる場合も含まれない場合もある)
ホスト変数データ・タイプ	文字
CCSID を宣言したホスト変数	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

エンコードするソース	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宣言または BOM が含まれる場合も含まれない場合もある)
ホスト変数データ・タイプ	グラフィック
CCSID を宣言したホスト変数	1200 または 13488

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 構文解析を実行して UTF-8 XML 列に保管するときに、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: UTF-16 から UTF-8 への変換時に、拡張による切り捨てが発生することがあります。

シナリオ 3

エンコードするソース	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宣言が含まれる場合も含まれない場合もある)

エンコードするソース	値
ホスト変数データ・タイプ	文字
CCSID を宣言したホスト変数	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 構文解析を実行して UTF-8 XML 列に保管するときに、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

エンコードするソース	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宣言が含まれる場合も含まれない場合もある)
ホスト変数データ・タイプ	グラフィック
CCSID を宣言したホスト変数	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 構文解析を実行して UTF-8 XML 列に保管するときに、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。

暗黙の直列化によって XML データを取り出す際のエンコードのシナリオ:

以下の例は、暗黙の直列化による 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 だけが Java アプリケーションに適用されます。これは、Java の文字ストリングが常に Unicode であるために、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

エンコードするソース	値
ターゲットのデータ・エンコード	UTF-8 Unicode
ターゲットのホスト変数データ・タイプ	2 進数
CCSID を宣言したホスト変数	適用されない

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。

直列化されたデータの内部エンコード方式: Java 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、かつ `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、エンコード方式の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。

シナリオ 2

エンコードするソース	値
ターゲットのデータ・エンコード	UTF-16 Unicode
ターゲットのホスト変数データ・タイプ	グラフィック
CCSID を宣言したホスト変数	1200 または 13488

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: UTF-8 から UTF-16 への変換時に、拡張による切り捨てが発生することがあります。

直列化されたデータの内部エンコード方式: Java と .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、かつ 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、エンコード方式の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。

シナリオ 3

エンコードするソース	値
ターゲットのデータ・エンコード	ISO-8859-1 データ
ターゲットのホスト変数データ・タイプ	文字
CCSID を宣言したホスト変数	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: データ損失が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。

直列化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

エンコードするソース	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)
ターゲットのホスト変数データ・タイプ	グラフィック
CCSID を宣言したホスト変数	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: データ損失が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: UTF-8 から CCSID 943 への変換時に、拡張による切り捨てが発生することがあります。

直列化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のシナリオ:

以下の例は、明示的 XMLSERIALIZE 呼び出しによる 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 だけが Java および .NET アプリケーションに適用されます。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

エンコードするソース	値
ターゲットのデータ・エンコード	UTF-8 Unicode
ターゲットのホスト変数データ・タイプ	2 進数
CCSID を宣言したホスト変数	適用されない

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。

直列化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

エンコードするソース	値
ターゲットのデータ・エンコード	UTF-16 Unicode
ターゲットのホスト変数データ・タイプ	グラフィック
CCSID を宣言したホスト変数	1200 または 13488

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: UTF-8 から UTF-16 への変換時に、拡張による切り捨てが発生することがあります。

直列化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、示される内部エンコード方式は UTF-16 ではなく UTF-8 です。この場合、エンコード名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

エンコードするソース	値
ターゲットのデータ・エンコード	ISO-8859-1 データ
ターゲットのホスト変数データ・タイプ	文字
CCSID を宣言したホスト変数	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: データ損失が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 で表せない場合、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、警告を出します。

切り捨て: なし。

直列化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、エンコード名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

エンコードするソース	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)
ターゲットのホスト変数データ・タイプ	グラフィック
CCSID を宣言したホスト変数	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: データ損失が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 で表せない場合、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、警告を出します。

切り捨て: UTF-8 から CCSID 943 への変換時に、拡張による切り捨てが発生することがあります。

直列化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、示される内部エンコード方式は Windows-31J ではなく UTF-8 です。この場合、エンコード名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。

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、または内部エンコード XML タイプである場合は、 DB2 データベース・マネージャーはエンコード方式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。

IANA エンコード名を CCSID にマップするために、QlgCvtTextDescToDesc API が使用されます。

直列化された XML 出力データのエンコード名への CCSID のマッピング

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・マネージャーは直列化された XML 出力データの先頭にエンコード宣言を追加します。

その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、出力ストリングにおける文字のエンコード方式を記述します。例えば、ターゲット・アプリケーションのデータ・タイプに対応する CCSID に XML データが直列化されるとき、エンコード宣言にはターゲット・アプリケーション変数の CCSID を記述します。

DB2 データベース・マネージャーは、可能であれば 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。 CCSID を IANA エンコード名にマップするために、QlgCvtTextDescToDesc API が使用されます。

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 スキーマ分解は「分解」または「断片化」とも呼ばれ、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をいいます。アノテーション付き XML スキーマ分解は、XML スキーマで指定されたアノテーションに基づいて行われます。 XML 文書を分解し終えると、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、以下のような詳細情報を指定します。

- 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
- 内容を保管する前に行う変換

これらのアノテーションを使用して何が指定できるのか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、スキーマを分解可能にする必要があります。

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・プロシージャーを呼び出して、分解を実行できます。

分解の際に、XML 文書からのデータは常に妥当性検査されます。 XML 文書内の情報が XML スキーマ内の仕様に準拠しない場合、そのデータは表に挿入されません。

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 文書の断片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 スキーマで指定したアノテーションに基づいて XML 文書を分解します。

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。

1. 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ます。
2. スキーマ文書を登録し、スキーマを分解可能にします。
3. 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 XML スキーマ文書すべてを再び登録し、XML スキーマを再度分解可能にする必要があります。
4. SYSPROC.XDBDECOMPXML ストアード・プロシージャーを呼び出して、XML 文書を分解します。

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を正常に登録し、分解を可能にすると、XML 文書の分解に使用できます。

- XML スキーマ内の少なくとも 1 つの要素宣言または属性宣言に、XML 分解アノテーションが付いていることを確認します。このアノテーションが付いている要素または属性は、複合タイプのグローバル要素そのもの、またはその子孫でなければなりません。

XML スキーマを登録し、分解を可能にするには、次のようにします。

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します。
2. XML スキーマが複数のスキーマ文書から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します。
3. `issuedfordecomposition` 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します。

アノテーション付き XML スキーマ分解のソース

アノテーション付き XML スキーマ分解は、XML 文書の要素または属性、および db2-xdb:contentHandling または db2-xdb:expression アノテーションの結果に対して実行できます。

アノテーション付き XML スキーマ分解は、以下のタイプの内容の分解をサポートします。

- XML 文書の属性または要素の値。
- XML 文書の要素の値。正確な内容は <db2-xdb:contentHandling> アノテーションの値によって異なります。 <db2-xdb:contentHandling> の値は、以下のとおりです。

  text　要素の文字データ。その子孫の文字データは除外。

stringValue
要素とその子孫の文字データ。

serializedSubtree
要素の開始タグと終了タグの間にあるすべての内容のマークアップ。

- db2-xdb:expression アノテーションにより生成される値。
  - XML 文書のマップ済み属性またはエレメントの内容に基づく値。
  - XML 文書内のどの値からも独立した、生成された値。
  - 定数。

  db2-xdb:expression により指定される式は、関連付けられた要素または属性ごとに一度呼び出されます。

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、この資料全体においては "db2-xdb" 接頭部で識別されます。独自の接頭部を選択できますが、その場合には、接頭部を名前空間 http://www.ibm.com/xmlns/prod/db2/xdb1 にバインドする必要があります。 XML スキーマを分解可能にした時点で、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を認識します。

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で認識されます。分解アノテーションは、属性として指定するか、要素宣言または属性宣言の <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 スキーマ文書中に存在しても、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することはありません。 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を指定する必要があります。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ために使用できます。

XML 分解アノテーションの指定とスコープ

XML スキーマ文書で、分解アノテーションを要素宣言または属性宣言として指定することができます。

以下のいずれかの方法で行うことができます。

- 要素宣言または属性宣言で単純属性として
- 単純要素宣言または単純属性宣言の構造化された (複合) 子要素として

属性としてのアノテーション

要素宣言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または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は属性として指定できます。これらのアノテーションを、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 という名前の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

要素宣言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 スキーマで定義されている <xs:annotation><xs:appinfo></xs:appinfo></xs:annotation> 階層内で指定する必要があります。

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に相当します。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> を指定する必要がある場合 (つまり、同じ要素宣言または属性宣言で複数のマッピングを指定する必要がある場合) 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 スキーマ文書すべて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非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、次のようにして、<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全体において、非修飾表すべてが「admin」という SQL スキーマを使用するように指定します。

XML 分解アノテーション - 要約

DB2 には、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で使用される一連のアノテーションが用意されています。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処置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の詳細は、該当する詳細資料を参照してください。

表 6. SQL スキーマの指定

処置	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 7. XML 要素または XML 属性からターゲット基本表へのマップ

処置	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 つの列と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 8. 分解される XML データの指定

処置	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
挿入前に適用される内容の変換がある場合、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、表名のデフォルト SQL スキーマを指定します。対象となるのは、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るのではない XML スキーマ内で参照されるすべての表名です。

アノテーション・タイプ

<xs:annotation> グローバル要素の子である <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 SQL スキーマ名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する必要があります。

詳細

アノテーション付きスキーマで参照される表はすべてその SQL スキーマで修飾する必要があります。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> 子要素を明示的に指定する方法、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> で指定する値がその 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する必要があります。

例

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において、通常の (区切り文字なしの) SQL ID admin を 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において、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 は引用符で区切る必要があります。

```
<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 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。

アノテーション・タイプ

<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

db2-xdb:rowSet アノテーションは、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。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 XML 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一連のスキーマ文書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> のいずれとも一致しない場合、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 のこのような使い方をします。
- 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> の 1 つと一致する場合、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 のこのような使い方をします。

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 スキーマが分解可能な場合に表が存在しない場合には、エラーが返されます。 <db2-xdb:table> が表以外のオブジェクトを指定する場合にも、エラーが返されます。

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 を組み合わせることによって、この要素または属性の分解先となる表および列が記述されます。 db2-xdb:rowSet と db2-xdb:condition を組み合わせることによって、表に挿入されるその rowSet の行すべてで true にならなければならない条件が指定されます (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 とおりの使い方があります。

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> で指定する 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・エレメント <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 文書の次の要素について考慮します。

```
<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>
```

次のように BOOKCONTENTS 表にデータが取り込まれます。

表 9. 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し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。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、ALLBOOKS 表は <db2-xdb:defaultSQLSchema> で指定する SQL スキーマに属します。

```
<!-- 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 文書の次の要素について考慮します。

```
<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>

<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

それらのセットは、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 10. 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 XML 属性を同じターゲット列にマップします。または、<db2-xdb:defaultSQLSchema> で指定したデフォルトの SQL スキーマとは異なる 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。

アノテーション・タイプ

<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。これらの子要素を使用するときには、このリストの順序で指定する必要があります。

<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の値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一連の XML スキーマ文書のすべての <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すべてで 1 つの行が形成されます。<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する列を組み合わせると、 1 つの XML 文書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一連の XML スキーマ文書のすべての <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文字には意味があり、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 はエスケープする必要があります。

詳細

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。

- 複数のパスを表の同じ列にマップする場合。
- 分解されたデータを保持する表が、<db2-xdb:defaultSQLSchema> アノテーションで定義された SQL スキーマと同じスキーマを持つものではない場合。

指定できるのは基本表のみです。それ以外の種類の表、例えば一時表や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。ビューおよび表の別名は、このアノテーションでは許可されていません。

例

次の例では、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、1 行にまとめる方法を示します。 XML 文書の次の要素についてまず考慮します (他のアノテーションで使用した例に若干の変更を加えています)。

```
<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>
```

著者 ID とそれらに対応する 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。著者 ID と E メール・アドレスが <book> 要素と <author> 要素の両方に存在することに注目してください。そのため、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。それで、<db2-xdb:table> アノテーションを使わなければなりません。次に、アノテーション付きスキーマのセクションを示します。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を示します。

```
<!-- 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、マッピングのターゲット表の名前を db2-xdb:name 子要素で指定します。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、rowSets を使用して行えます。

AUTHORSCONTACT 表は、デフォルトの SQL スキーマとは異なる 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。生成される AUTHORSCONTACT 表を以下に示します。

表 11. AUTHORSCONTACT

AUTHID	EMAILADDR
22	author22@anyemail.com
0800	author800@email.com

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 XML 属性がマップされる表の列名を指定します。

アノテーション・タイプ

<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

任意の基本表の列名。

詳細

- 区切り文字なしの列名は大/小文字を区別しません。
- 二重引用符 (")、アンパーサンド (&)、より小符号 (<) などの特殊文字が SQL ID の一部である場合、これらの特殊文字を同等の XML 表記に置き換える必要があります。例えば、" は " に、& は & に、< は < に置き換えます。
- db2-xdb:locationPath アノテーションがある場合、db2-xdb:column は db2-xdb:rowSetMapping のオプションの子要素です。

例

次の例は、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。最初にアノテーション付きスキーマのセクションを示します。

```
<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>
```

次に、マップされている <book> 要素を示します。続いて分解完了後に生成される BOOKCONTENTS 表を示します。

```
<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 12. 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、XML 要素または属性を、その要素または属性のパスに応じて、さまざまな表と列の対にマップします。

db2-xdb:locationPath アノテーションは、グローバルに宣言されるか、以下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
- 単純タイプまたは複合タイプのグローバル要素またはグローバル属性

アノテーション・タイプ

<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" />`
-

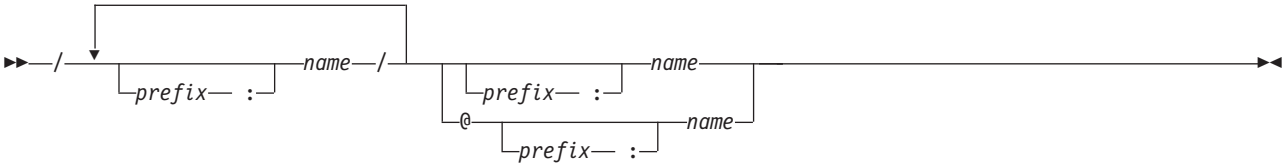
```
<db2-xdb:rowSetMapping> db2-xdb:locationPath="value">
  <db2-xdb:rowSet>value</db2-xdb:rowSet>
  ...
</db2-xdb:rowSetMapping>>
```

名前空間

http://www.ibm.com/xmlns/prod/db2/xdb1

有効な値

db2-xdb:locationPath の値は次の構文でなければなりません。



name 要素または属性の名前。

prefix 名前空間接頭部。

名前空間接頭部は、db2-xdb:locationPath を使用したアノテーションが含まれているスキーマ文書内の名前空間と関連付けられている必要があります。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。

詳細

再利用できない要素宣言または属性宣言 (名前付き複合タイプ定義の一部ではなく、名前付きのモデル・グループや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。

要素または属性のグローバル宣言が他のパスからの参照として使用される場合 (例えば、<xs:element ref="abc">)、db2-xdb:locationPath を使用してください。アノテーションは、参照で直接指定することができないため、対応する要素または属性のグローバル宣言で指定する必要があります。グローバルな要素または属性は、XML スキーマ内の多数のさまざまなコンテキストから参照できます。一般に、さまざまなコンテキストでマッピングを区別するには、db2-xdb:locationPath を使用する必要があります。名前付きの複合タイプ、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コンテキストごとに、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。各パスのターゲットの rowSet と列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 と列の対にも同じ db2-xdb:locationPath 値を使用できます。

デフォルトの名前空間と *attributeFormDefault ="unqualified"* を指定すると、非修飾属性の分解アノテーションは無視されます。これは、アノテーションの処理で、属性がデフォルトの名前空間に属するものであるかのように扱われるためです。しかし *attributeFormDetail = "unqualified"* 設定は、属性が実際にはグローバル名前空間に属していることを示しています。その場合、この属性のマッピングは無視され、値は挿入されません。

例

次の例は、この属性が現れるコンテキストに応じて、同じ属性がどのように異なる表にマップされるかを示しています。最初にアノテーション付きスキーマのセクションを示します。

```
<!-- 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。

次に、マップされている <books> 要素を示します。続いて分解完了後に生成される BOOKS 表を示します。

```
<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 13. BOOKS

ISBN	TITLE	CONTENT
NULL	My First XML Book	NULL

表 14. 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。

アノテーション・タイプ

<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>
```

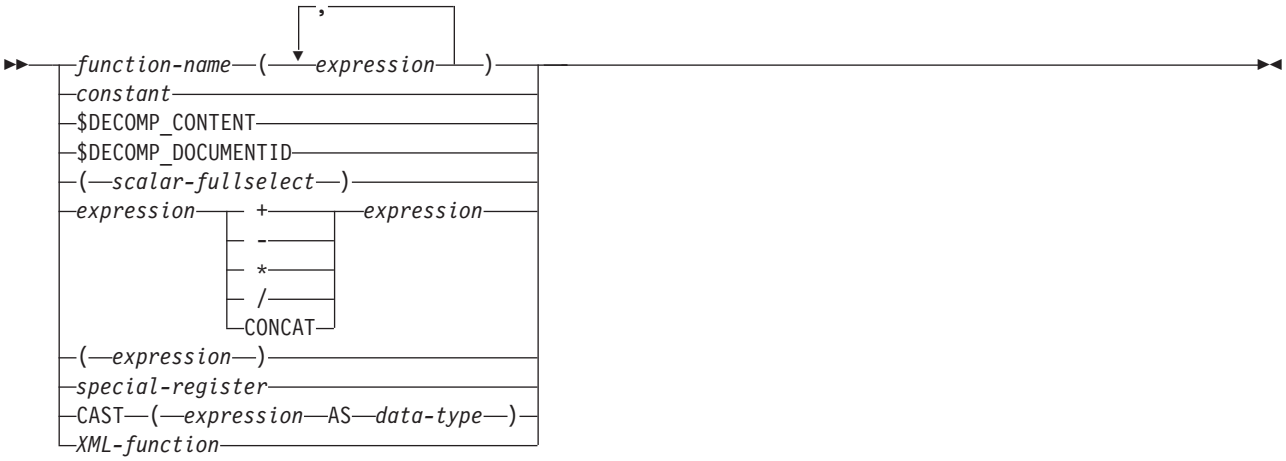
名前空間

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。

式:



詳細

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ます。これは $DECOMP_CONTENT の使用時に、アノテーション付きの 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。

db2-xdb:expression 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便利です。

db2-xdb:expression は有効な SQL 式を使用して指定する必要があり、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。 SQL 式の以下のサブセットがサポートされます。以下に記述されていない他の 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。

db2-xdb:expression アノテーション内のスキーマ名、表名、および列名は、修飾時には SQL 命名方式を使用する必要があり、それらの名前が区切り文字で区切られている場合にのみ、大/小文字が区別されます。

function (expression-list)
組み込み、またはユーザー定義の SQL スカラー関数。スカラー関数は単一値を返します (NULL の可能性あり)。

constant
ストリング定数または数値定数の値。

$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。

$DECOMP_DOCUMENTID
XDBDECOMPXML ストアード・プロシージャーの *documentid* 入力パラメーターで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。

(scalar-fullselect)
単一列値から成る単一行を返す、括弧で囲まれた全選択。全選択が行を返さない場合、式の結果は NULL 値になります。

expression operator expression
上記リストの、サポートされる値で定義されている、サポートされる 2 つの式オペランドの結果。

(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。

CAST (expression AS data-type)
式が NULL でない場合、指定した SQL データ・タイプへの式のキャスト。式が NULL の場合、結果は指定した 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にキャストする必要があります (例えば整数列の場合は CAST (NULL AS INTEGER))。

XML-function
サポートされる任意の SQL/XML 関数。

例

次の例は、db2-xdb:expression アノテーションを使用して行うことができる操作のタイプを示します。この例はまた、XML 入力を SQL タイプにキャストする必要がある事例についても示します。

入力 XML 文書は、次のようになります。

```
<num>
1000.99
</num>
```

アノテーション付き XML スキーマは、以下のように定義された表の行を作成します。

```
CREATE TABLE TAB1
   (ASIS    DECIMAL(9,2),
    ADD     INTEGER,
    TAX     INTEGER,
    STR     VARCHAR(25),
    LITERAL DECIMAL(5,1)
    SELECT  VARCHAR(25))
```

アノテーション付き XML スキーマは、以下のように定義された表を使用します。

```
CREATE TABLE SCH1.TAB2
   (ID   INTEGER,
    COL1 VARCHAR(25))
```

表 SCH1.TAB2 には、(100, 'TAB2COL1VAL') という値を持つ 1 つの行が含まれています。

```
<xs:element name="num" type="xs:double">
  <xs:annotation>
    <xs:appinfo>
      <xdb:rowSetMapping>
        <xdb:rowSet>TAB1</xdb:rowSet>
        <xdb:column>ASIS</xdb:column>
      </xdb:rowSetMapping>
      <xdb:rowSetMapping>
        <xdb:rowSet>TAB1</xdb:rowSet>
        <xdb:column>ADD</xdb:column>
        <xdb:expression>
          CAST(XDB.ADD(1234,CAST($DECOMP_CONTENT AS INTEGER)) AS INTEGER)
        </xdb:expression>
     </xdb:rowSetMapping>
     <xdb:rowSetMapping>
        <xdb:rowSet>TAB1</xdb:rowSet>
        <xdb:column>TAX</xdb:column>
        <xdb:expression>
```

```
        CAST(XDB.TAX(CAST($DECOMP_CONTENT AS DOUBLE)) AS INTEGER)
      </xdb:expression>
    </xdb:rowSetMapping>
    <xdb:rowSetMapping>
      <xdb:rowSet>TAB1</xdb:rowSet>
      <xdb:column>STR</xdb:column>
      <xdb:expression>
        CAST($DECOMP_CONTENT AS VARCHAR(25))
      </xdb:expression>
    </xdb:rowSetMapping>
    <xdb:rowSetMapping>
      <xdb:rowSet>TAB1</xdb:rowSet>
      <xdb:column>LITERAL</xdb:column>
      <xdb:expression>32.3</xdb:expression>
    </xdb:rowSetMapping>
    <xdb:rowSetMapping>
      <xdb:rowSet>TAB1</xdb:rowSet>
      <xdb:column>SELECT</xdb:column>
      <xdb:expression>
        (SELECT "COL1" FROM "SCH1"."TAB2" WHERE "ID" = 100)
      </xdb:expression>
    </xdb:rowSetMapping>
  </xs:appinfo>
 </xs:annotation>
</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。これは著者の ID を表し、その著者がシステム内に持つ本の合計数を返します。

表 15. TAB1

ASIS	ADD	TAX	STR	LITERAL	SELECT
1000.99	2234	300	1000.99	32.3	TAB2COL1VAL

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。

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。

アノテーション・タイプ

<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。

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

以下のタイプの SQL 述部。

- 基本
- 限量化
- BETWEEN
- DISTINCT
- EXISTS
- IN
- LIKE
- NULL

また、これらの述部は db2-xdb:expression アノテーション、列名のいずれかまた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

同じ rowSet の要素または属性の複数の宣言に対して 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行が挿入されます。

db2-xdb:condition 内の列名

db2-xdb:condition は SQL 述部で構成されるため、このアノテーションで列名を指定できます。 rowSet に関する 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の中に、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んだ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。修飾されていない列名を db2-xdb:condition で指定している場合でも、db2-xdb:condition が指定されている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、条件の評価時には、参照され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。

次の例を検討してください。

```
<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:condition="columnX" />
```

<a> には列マッピングが指定されていませんが、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、条件の "columnX" は <b> 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> 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> 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。

文書内に存在しないマップ済み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場合、その要素または属性が XML 文書内にない場合でも、その条件は適用されます。例えば、アノテーション付きスキーマ文書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" は評価されます。 <intElem> がないため、"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 文書の以下の <author> 要素について考慮します。

```
<author ID="0800">
  <firstname>Alexander</firstname>
  <lastname>Smith</lastname>
  <activeStatus>1</activeStatus>
</author>
```

db2-xdb:condition で指定した条件に応じて、分解時にこの <author> 要素からターゲット表に値を挿入するかしないかが決まります。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

上記の <author> 要素に対応するアノテーション付きスキーマ内の以下のセクションは、著者の 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>
```

上記の例の <author> 要素の値により、db2-xdb:condition で指定したすべての条件が満たされているので、<author> 要素のデータが AUTHORS 表に追加されます。

表 16. 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、著者の 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> 要素は指定の条件を満たしていますが、 ID 属性の値は満たしていないため、分解時には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 であるため、論理 AND は false に評価され、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、複合タイプまたは単純タイプの要素について、表に分解する内容のタイプを指定します。

アノテーション・タイプ

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いずれかのトークン。

- text
- stringValue
- serializeSubtree

詳細

XML 要素の宣言で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。

db2-xdb:contentHandling では、次の 3 つの値が有効です。

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: 挿入される直列化ストリングは、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 スキーマで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。

  この設定の結果として生成される直列化ストリングは XML エンティティーであるため、CCSID 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である場合は、XML フラグメントが列の CCSID 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。このプロセッサーが UTF-8 以外のエンコード方式を自動的に検出しないことが、その理由です。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。

分解アノテーション付き XML 要素宣言が複合タイプであり、複合内容は含まれていても db2-xdb:contentHandling が指定されていない場合、デフォルトの動作は「serializeSubtree」の設定に従います。他のすべてのアノテーション付き要素宣言の場合、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」の設定に従います。

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「mixed」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 のいずれかのアノテーションにおいて $DECOMP_CONTENT を置換する値に影響します。置換値は、まず db2-xdb:contentHandling の設定に従って処理されてから、評価のために渡されます。

例

以下の例は、使用する db2-xdb:contentHandling アノテーションの設定が異なると、ターゲット表に生成される結果がどのように異なるかを示しています。最初に示すアノテーション付きスキーマでは、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は、db2-xdb:contentHandling を "text" に設定した例のみ取り上げています。このセクションのそれ以降の例では、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ものとします。)

```
<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、マップされている <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: 以下の結果表の CHPTTITLE 欄と CHPTCONTENT 欄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文字を示すために付いているに過ぎません。

db2-xdb:contentHandling="text"

表 17. 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	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 <b> 要素の内容が挿入されない理由に注目してください。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目してください。 <paragraph> 要素の文字データの連結の空白文字は保持されます。

db2-xdb:contentHandling="stringValue"

表 18. 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..."

この表と前の表の違いが最初の行の CHPTCONTENT 欄にあります。ストリング "lots of" (<paragraph> 要素の 子孫に由来する) が挿入されている理由に注目してください。 db2-xdb:contentHandling を "text" に設定した場合、"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ました。しかし "stringValue" 設定は、子孫の内容を含めます。 "text" 設定と同様に、コメントと処理命令は挿入されず、空白文字は保持されます。

db2-xdb:contentHandling="serializeSubtree"

表 19. 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>"
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)。この違いには、最初の行の CHPTCONTENT 欄に の開始タグと終了タグの有無、さらに 2 行目のコメントと 3 行目の処理命令の有無が含まれます。前の 2 つの例と同様に、XML 文書の空白文字は保持されています。

db2-xdb:normalization 分解アノテーション

db2-xdb:normalization アノテーションは、挿入される XML データまたは $DECOMP_CONTENT を置換する (db2-xdb:expression とともに使用される場合) XML データ内の空白文字の正規化を指定します。

アノテーション・タイプ

<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いずれかのトークン。

canonical
: XML 値がターゲット列に挿入される前、またはこの db2-xdb:normalization アノテーションと同じマッピング内の $DECOMP_CONTENT のオカレンスが XML 値に置換される前に、XML 値はその XML スキーマ・タイプに応じた正規形に変換されます。

original
: XML 値がターゲット列に挿入される前、またはこの db2-xdb:normalization アノテーションと同じマッピング内の $DECOMP_CONTENT のオカレンスが XML 値に置換される前には、XML に対する変更は行われません。ただし、XML パーサーによって処理が行われる可能性はあります。これはデフォルトです。

whitespaceStrip
: XML 値がターゲット列に挿入される前、またはこの db2-xdb:normalization アノテーションと同じマッピング内の $DECOMP_CONTENT のオカレンスが XML 値に置換される前に、次の処理が行われます。
  - XML 値の前後の空白文字は、すべて除去されます。
  - 連続した空白文字は、単一の空白文字に短縮されます。

詳細

db2-xdb:normalization は、次のいずれかのアトミック XML スキーマ・タイプを持つ要素または属性に適用されます。

- byte、unsigned byte
- integer、positiveInteger、negativeInteger、nonPositiveInteger、nonNegativeInteger
- int、unsignedInt
- long、unsignedLong
- short、unsignedShort
- decimal
- float
- double
- boolean
  ```

- time
- date
- dateTime

ターゲット列は、次のいずれかのデータ・タイプでなければなりません。

- CHAR
- VARCHAR
- CLOB
- DBCLOB
- GRAPHIC
- VARGRAPHIC

その他のタイプに対して指定された場合、db2-xdb:normalization は無視されます。

例

以下の例は、db2-xdb:normalization アノテーションを使って空白文字の正規化を制御す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示します。

```
      <xs:element name="author">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="firstname" type="xs:string"
                        db2-xdb:rowSet="AUTHORS" db2-xdb:column="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 (注目する空白文字は明示するために以下で下線文字 '_' 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 20. AUTHORS

AUTHID	FIRSTNAME	SURNAME	ACTIVE	NUMBOOKS
22	Ann	__Brown_	true	NULL

ID 属性の db2-xdb:normalization="whitespaceStrip" アノテーションにより、データは、AUTHORS 表に挿入される前に、前後の空白文字が除去されます。 <activeStatus> 要素の db2-xdb:normalization="canonical" アノテーションにより、要素のブール値は、 AUTHORS 表に挿入される前に、その値の正規表現に置き換えられます。要素はブール・タイプです。ブール・タイプの正規表現は、true または false 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、異なる表間での行の挿入順序を指定します。

アノテーション・タイプ

<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

以下は、<db2-xdb:order> のサポートされている子要素です。

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

db2-xdb:order アノテーションは、特定の rowSet に属する行の挿入順序を、別の rowSet に属する行の挿入と比較して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。db2-xdb:order 要素内で使用できる db2-xdb:rowSet 要素の数は、1 より大きい任意の数とすることができます。

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、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 の場合、その行は、他のいずれかの rowSet の行と比較し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文字内容の中に含める必要があります。これはエスケープする必要はありません。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>
```

上記の例では、共通する部分のない 2 つの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。

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。

アノテーション・タイプ

<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

以下のいずれかの値です。

0 または false
値は挿入前に切り捨てることはできません。値が長過ぎる場合には、エラーが発生し、値は挿入されません。これはデフォルトです。

1 または true
値は挿入前に切り捨てることができます。

詳細

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、切り捨てが許可されないことを示す「false」または「0」に設定され、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、以下のマッピングにのみ適用できます。

XML データ・タイプ	列のデータ・タイプ
任意の互換タイプ	CHAR VARCHAR CLOB GRAPHIC VARGRAPHIC DBCLOB
xs:date	DATE
xs:time	TIME
xs:dateTime	TIMESTAMP

db2-xdb:truncate と同じ要素宣言または属性宣言で db2-xdb:expression アノテーションを指定した場合、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。

XML 日時値を DATE、TIME、または TIMESTAMP 列に分解するアノテーションにおいては、XML データに時間帯がある場合には、db2-xdb:truncate を true または 1 に設定する必要があります。

例

次の例は、<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。最初にアノテーション付きスキーマのセクションを示します。

```
<xs:element name="author">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="firstname" type="xs:string"
                  db2-xdb:rowSet="AUTHORS" db2-xdb:column="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>
```

次に、マップされている <author> 要素を示します。

```
<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 表を次に示します。

表 21. 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 を指定しない場合には db2-xdb:truncate のデフォルト値が取られ (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、XML 要素または XML 属性を、1 つのターゲット表および列、同じターゲット表の複数の列、あるいは複数の表および列にマップします。

アノテーション・タイプ

<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 XML 属性を、その要素または属性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 XML 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。

以下は、サポートされる <db2-xdb:rowSetMapping> の子要素です。これらの子要素を使用するときには、このリストの順序で指定する必要があります。

<db2-xdb:rowSet>
XML 要素または XML 属性をターゲットの基本表にマップします。

<db2-xdb:column>
XML 要素または XML 属性を基本表の列にマップします。 db2-xdb:condition または db2-xdb:locationPath アノテーションがある場合、この要素はオプションです。

<db2-xdb:expression>
カスタマイズされた式を指定します。その結果は db2-xdb:rowSet 属性で指定する表に挿入されます。この要素はオプションです。

<db2-xdb:condition>
評価の条件を指定します。この要素はオプションです。

詳細

db2-xdb:expression と db2-xdb:truncate を一緒に指定すると、db2-xdb:truncate は無視されます。

1 つの表および列にマッピングする場合、db2-xdb:rowSetMapping を指定することは、 db2-xdb:rowSet アノテーションと db2-xdb:column アノテーションを組み合わせて指定することと同じになります。

<db2-xdb:rowSetMapping> の子要素の文字内容の中の空白文字にはすべて意味があります。空白文字の正規化は行われません。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

プせずに文字内容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

次の例は、<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。最初にアノテーション付きスキーマのセクションを示します。 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>
```

次に、マップされている <book> 要素を示します。続いて分解完了後に生成される BOOKS 表および BOOKCONTENTS 表を示します。

```
<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 22. BOOKS

ISBN	TITLE	CONTENT
1-11-111111-1	NULL	NULL

表 23. BOOKCONTENTS

ISBN	CHPTNUM	CHPTTITLE	CHPTCONTENT
1-11-111111-1	NULL	NULL	NULL

<db2-xdb:rowSetMapping>、db2-xdb:rowSet、および db2-xdb:column を組み合わせて使用する代替マッピング

次のアノテーション付きスキーマのセクションは上記の XML スキーマの断片と同等のもので、分解の結果は同じになります。 2 つのスキーマの違いは、次のスキーマの場合、1 つのマッピングの置換を、<db2-xdb:rowSetMapping> アノテーションのみを使用して行うのではなく、db2-xdb:rowSet と db2-xdb:column を組み合わせて行うという点です。

```
<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。

アノテーション・タイプ

<xs:annotation> グローバル要素の子である <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

以下は、<db2-xdb:rowSetOperationOrder> のサポートされている子要素です。

db2-xdb:order

詳細

<db2-xdb:rowSetOperationOrder> は、<db2-xdb:order> 要素をまとめてグループ化します。子 <db2-xdb:order> 要素のインスタンスを複数置くことができ、これにより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なっていることにより、db2-xdb:rowSetOperationOrder アノテーションおよび 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った他のアプリケーションの要件も確実に配慮されるようになります。

db2-xdb:rowSetOperationOrder アノテーションは、XML スキーマで一度しか使用できません。

例

rowSet の挿入順序の指定例については、102 ページの『db2-xdb:order 分解アノテーション』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

文書にあるマップ済み XML 要素または XML 属性の値。値は、db2-xdb:contentHandling アノテーションの設定に従って構成されます。 $DECOMP_CONTENT の値は文字タイプです。

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用して、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 を使用できます。

db2-xdb:expression が $DECOMP_CONTENT を指定し、その同じマッピングで db2-xdb:normalization を指定すると、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。

$DECOMP_DOCUMENTID

XDBDECOMPXML ストアード・プロシージャーの *documentid* 入力パラメーターで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き、ストアード・プロシージャーに提供される入力値が $DECOMP_DOCUMENTID を置換する値として使用されます。

$DECOMP_DOCUMENTID を使用すると、XML 文書に存在しない固有 ID を挿入できます。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。その後、これらの ID をデータベース内の表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。

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

分解のアノテーション付き要素に CDATA セクションがある場合、分解プロセスにより、CDATA セクションの内容は CDATA セクションの区切り文字 ("<![CDATA[" および "]]>") を除いて表に挿入されます。

CDATA セクション内の復帰および改行の対 (U+000D と U+000A) または復帰 (U+000D) は、改行 (U+000A) に置き換えられます。

属性 db2-xdb:contentHandling="serializeSubtree" を使用して、XML スキーマ内の XML 要素宣言にアノテーションを付けると、CDATA セクションの内容は以下のように変更されて、表に挿入されます。

- CDATA セクションの区切り文字 ("<![CDATA[" および "]]>") が除去されます。
- CDATA セクションのアンパーサンド (&) が、それぞれストリング `&amp:` に置き換えられます。
- CDATA セクションの左不等号括弧 (<) が、それぞれストリング `&lt:` に置き換えられます。

分解後の CDATA セクションの内容は、論理的には、CDTATA セクションの元の内容と等価です。

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 24. 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
文書で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
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、和集合タイプ、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付き XML 属性が NULL 値を持つか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 25. 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
文書で属性が欠落している (妥当性検査が実行されなかったか、または妥当性検査で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、和集合タイプ、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: 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それを変換することや、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学習します。
- マッピング中に、列のタイプが、マップされてい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。
- 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記述します。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を以下に示します。

派生した複合タイプのアノテーション

XML スキーマには、制限または拡張により派生する複合タイプ (restriction または extension 要素を含んだ complexType 要素で指定する) を含めることができます。

分解のためにこれらの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。

複合タイプが XML スキーマの複数の場所で参照される場合、db2-xdb:locationPath アノテーションを使用して、スキーマ内の場所に応じて、それぞれ異なる表および列にマップすることができます。

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共通属性が、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が、派生タイプ内にも含まれていなければなりません。

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。

例: 以下の XML スキーマ文書で、outOfPrintBookType は、拡張により派生します。これは、基本タイプとは異なる表 bookType にマップされます。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ために、bookType 基本タイプに db2-xdb:locationPath アノテーションが指定されています。この例では、派生タイプ outOfPrintType の <lastPublished> 要素と <publisher> 要素には db2-xdb:locationPath アノテーションは必要ありません。これらの要素は単一のマッピングにのみ関係するものだからです。

```
<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: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>                                                    注 1
        <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"                      注 2a
                  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"                          注 2b
                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"              注 3
                 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>
```

アノテーションは、<book> 要素の値は BOOKS 表に分解され、 <outOfPrintBook> 要素の値は OUTOFPRINT 表に分解されることを示します。

アノテーション付き XML スキーマ文書に関する注:

1 <books> 文書には、<book> 要素と <outOfPrintBook> 要素の 2 種類の要素を使用できます。これら 2 種類の要素に由来する情報は、それぞれ別の表に分解されます。 <outOfPrintBook> 要素定義には、<book> 要素定義への拡張があります。

2a および 2b

XML スキーマの title 属性および isbn 属性では、<book> 要素または <outOfPrintBook> 要素の定義が同じですが、 in-print および out-of-print の各ブック情報はそれぞれ別の表に送られます。そのため、in-print ブックおよび out-of-print ブックのタイトルと ISBN 番号を区別するために、db2-xdb:locationPath アノテーションが必要です。

3 lastPublished 要素および publisher は、<outOfPrintBook> 要素に固有の要素であるので、これらの要素に db2-xdb:locationPath アノテーションは必要ありません。

この XML スキーマ文書を使用して、以下の XML 文書を分解する場合を考えてみます。

```
<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 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>
```

分解によって以下の表が作成されます。

表 26. 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 27. 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

アノテーション付き XML スキーマ分解における行セット

db2-xdb:rowSet アノテーションは、XML 文書内の値の分解先のターゲット表および行を示します。

db2-xdb:rowSet アノテーションの値として、表名または行セット名を指定します。

db2-xdb:rowSet アノテーションは、要素宣言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指定できます。

XML スキーマでは、1 つの要素または属性を複数回使用可能で、これらすべての要素または属性において、同じターゲット表値を持つ db2-xdb:rowSet アノテーションを指定できます。これらの db2-xdb:rowSet アノテーションは、それぞれターゲット表内の行を定義します。

例: 各本の isbn および title の値が ALLPUBLICATIONS 表に挿入されるように、この文書を分解する場合を考えてみます。

```
<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>Melanie Snowe</author>
    <publicationDate>1999</publicationDate>
  </childrensbook>
</publications>
```

以下の行セットを定義する必要があります。

- テキスト・ブックのタイトルでその isbn 値をグループ化する行セット
- そのタイトルの児童書の isbn 値をグループ化する行セット

アノテーション付き XML スキーマは、以下のようになります。

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns:db2-xdb="http://www.ibm.com/xmlns/prod/db2/xdb1"
           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>                    注 1
        <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"                注 2a
                       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"注 2b
                      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"                注 3a
                    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"注 3b
                db2-xdb:rowSet="childrens_rowSet" db2-xdb:column="PUBS_TITLE"/>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

アノテーション付き XML スキーマ文書に関する注:

1 グローバルな <xs:annotation> で、 textbk_rowSet 行セットおよび childrens_rowSet 行セットは、XML スキーマで後で参照するために宣言されています。 <db2-xdb:name> アノテーションは、行セットが参照する表 (ALLPUBLICATIONS) を指定します。

2a および 2b

textbk_rowSet アノテーションは、<textbook> 要素の isbn 要素宣言および title 属性宣言で指定されています。これは、<textbook> 要素に由来する isbn および title の情報が、ALLPUBLICATIONS 表の行に分解されることを示しています。

3a および 3b

childrens_rowSet アノテーションは、<childrensbook> 要素の isbn 要素宣言および title 属性宣言で指定されています。これは、<childrensbook> 要素に由来する isbn および title の情報が、ALLPUBLICATIONS 表の行に分解されることを示しています。

以下の表は、アノテーション付き XML スキーマで、前述の文書を分解した結果を示しています。

表 28. ALLPUBLICATIONS

ISBN	PUBS TITLE
0-11-011111-0	Programming with XML
5-55-555555-5	Children's Fables

上記の例は、行セットを使用した分解の単純な事例を示しています。行セットをさらに複雑なマッピングで使用し、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、同じ表と列の対に行をまとめることもできます。

条件付き変換

行セットを使用すると、分解対象の値に対して、その値ごとに異なる別々の変換を適用することができます。

例: temperature という名前の要素の次の 2 つのインスタンスは、属性値が異なります。

```
<temperature unit="Celsius">49</temperature>
<temperature unit="Farenheit">49</temperature>
```

これらの値を分解して同じ表に入れるには、属性 unit="Celsius" を持つすべての要素を 1 つの行セットにマップし、属性 unit="Farenheit" を持つすべての要素を別の行セットにマップする必要があります。次に、1 つの行セットの値に変換公式を適用して、値を表に挿入する前にすべての値が同じ温度単位になるようにします。

以下のアノテーション付き XML スキーマは、この手法を示しています。

```
<db2-xdb:name>TEMPERATURE_DATA</db2-xdb:name>
  <db2-xdb:rowSet>temp_celcius</db2-xdb:rowSet>                          注 1
  <db2-xdb:rowSet>temp_farenheit</db2-xdb:rowSet>
</db2-xdb:table>
...

<xs:element name="temperature">
  <xs:annotation>
    <xs:appinfo>
      <db2-xdb:rowSetMapping>
        <db2-xdb:rowSet>temp_celcius</db2-xdb:rowSet>
        <db2-xdb:column>col1</db2-xdb:column>
      </db2-xdb:rowSetMapping>
      <db2-xdb:rowSetMapping>
        <db2-xdb:rowSet>temp_farenheit</db2-xdb:rowSet>
        <db2-xdb:column>col1</db2-xdb:column>
<db2-xdb:expression>CAST(myudf_convertToCelcius(CAST($DECOMP_CONTENT AS FLOAT)) AS FLOAT)
</db2-xdb:expression>                                                    注 2
      </db2-xdb:rowSetMapping>
    </xs:appinfo>
  </xs:annotation>
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute name="unit" type="xs:string">
          <xs:annotation>
            <xs:appinfo>
              <db2-xdb:rowSetMapping>                                    注 3
                <db2-xdb:rowSet>temp_celcius</db2-xdb:rowSet>
                <db2-xdb:condition>$DECOMP_CONTENT = 'Celsius'</db2-xdb:condition>
              </db2-xdb:rowSetMapping>
              <db2-xdb:rowSetMapping>
                <db2-xdb:rowSet>temp_farenheit</db2-xdb:rowSet>
                <db2-xdb:condition>$DECOMP_CONTENT = 'Farenheit'</db2-xdb:condition>
              </db2-xdb:rowSetMapping>
            </xs:appinfo>
          </xs:annotation>
        </xs:attribute>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

アノテーション付き XML スキーマ文書に関する注:

1 グローバルな <xs:annotation>, で、temp_celcius 行セットおよび temp_farenheit 行セットは、XML スキーマで後で参照するために宣言されています。

2 変換公式によって temp_farenheit 行セットの値が摂氏単位に変換され、値を表に挿入する際にすべての値が摂氏単位になるようにします。式アノテーションには、関数の引数と戻りの型を、関数で定義されている対応する SQL データ・タイプにキャストするための CAST 指定が含まれている必要があります。

3 unit="Celsius" 属性を持つすべての要素は、temp_celcius 行セットにマップされ、unit="Farenheit" 属性を持つすべての要素は temp_farenheit 行セットにマップされます。

分解アノテーションの例: XML 列へのマッピング

アノテーション付き 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 文書について考慮します。

```
<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、XML 要素 <textbook> 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 title 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 29. 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>

スキーマ文書の中でアノテーションを属性と要素のどちらで指定できるかは、アノテーションによって異なります。一部のアノテーションはどちらか一方で指定できます。特定のアノテーションをどのように指定するかを判別するには、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として指定することと属性として指定することは同等です。

次の XML スキーマ文書の断片は、アノテーションを属性として指定することによって、 2 つのマッピングを <textbook> 要素および title 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、複合内容が入り、

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、その要素内のすべてのマークアップ (その開始タグおよび終了タグを含む) が、db2-xdb:contentHandling の serializeSubtree 設定に従って XML 列に挿入されます。 XML 文書内の空白文字は保持されます。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の例: 単一行を生成する値を単一の表にマップする

XML 文書から単一の表と列の対への値のマッピングは、アノテーション付き XML スキーマ分解におけるマッピングの単純な形式です。この例では、1 つの rowSet の値間に 1 対 1 の関係があるような単純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関係に依存します。単一の rowSet に属する値が 1 対 1 の関係でマップされる場合、その要素の maxOccurs 属性の値、または含まれているモデル・グループ宣言によって決定されるように、XML 文書中のマップされた項目のインスタンスごとに単一行が形成されます。単一の rowSet に属する値に 1 対多の関係があり、文書内で、ある値が別の項目の複数のインスタンス (maxOccurs 属性の値で示される数) において一度だけ現れる場合には、XML 文書の分解時に複数の行が形成されます。

次の XML 文書について考慮します。

```
<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、title 属性と一緒に TEXTBOOKS 表に分解する場合、対応する XML スキーマ文書で、これら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。アノテーションでは、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 30. TEXTBOOKS

ISBN	TITLE	DATE
0-11-011111-0	Programming with XML	2002

値を単一の表と列の対にマップする場合は、マップする値に表および列を指定する必要があります。これは、次の一連のアノテーションを使用して行います。

- <xs:element> または <xs:attribute> の属性として db2-xdb:rowSet および db2-xdb:column
- <db2-xdb:rowSetMapping> の子要素として <db2-xdb:rowSet> および <db2-xdb:column>

次のアノテーション付き XML スキーマでは、db2-xdb:rowSet および db2-xdb:column アノテーションを属性として指定します。

```
<?xml version="1.0" encoding="UTF-8"?>
<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>"MYSCHEMA"</db2-xdb:defaultSQLSchema>
  </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" maxOccurs="1"
                       db2-xdb:rowSet="TEXTBOOKS"
                       db2-xdb:column="ISBN"/>
           <xs:element name="author" type="xs:string" maxOccurs="unbounded"/>
           <xs:element name="publicationDate" type="xs:gYear" maxOccurs="1"
                       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>
</xs:schema>
```

行セットに入る要素の maxOccurs 属性は 1 であるため、TEXTBOOKS 表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。そのため、<textbook> 要素の各インスタンスにおいて単一行が形成されます。

分解アノテーションの例: 複数行を生成する値を単一の表にマップする

XML 文書から単一の表と列の対への値のマッピングは、アノテーション付き XML スキーマ分解におけるマッピングの単純な形式です。この例では、1 つの rowSet の値間に 1 対多の関係があるような複雑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関係に依存します。単一の rowSet に属する値が 1 対 1 の関係でマップされる場合、その要素の maxOccurs 属性の値、または含まれているモデル・グループ宣言によって決定されるように、XML 文書中のマップされた項目のインスタンスごとに単一行が形成されます。単一の rowSet に属する値に 1 対多の関係があり、文書内で、ある値が別の項目の複数のインスタンス (maxOccurs 属性の値で示される数) において一度だけ現れる場合には、XML 文書の分解時に複数の行が形成されます。

次の XML 文書について考慮します。

```
<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 および著者を格納する場合、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。アノテーションでは、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 31. TEXTBOOKS_AUTH

ISBN	AUTHNAME
0-11-011111-0	Mary Brown

表 31. TEXTBOOKS_AUTH (続き)

ISBN	AUTHNAME
0-11-011111-0	Alex Page

スキーマ文書の中でアノテーションを属性と要素のどちらで指定できるかは、アノテーションによって異なります。一部のアノテーションはどちらか一方で指定できます。特定のアノテーションをどのように指定するかを判別するには、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。

値を単一の表と列の対にマップする場合は、マップする値に表および列を指定する必要があります。これは、次の一連のアノテーションを使用して行います。

- <xs:element> または <xs:attribute> の属性として db2-xdb:rowSet および db2-xdb:column
- <db2-xdb:rowSetMapping> の子要素として <db2-xdb:rowSet> および <db2-xdb:column>

次のアノテーション付き XML スキーマでは、<db2-xdb:rowSet> および <db2-xdb:column> アノテーションを要素として指定します。

```
<?xml version="1.0" encoding="UTF-8"?>
<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>"MYSCHEMA"</db2-xdb:defaultSQLSchema>
  </xs:appinfo>
 </xs:annotation>

 <xs:element name="textbook">
  <xs:complexType>
    <xs:sequence maxOccurs="unbounded">

      <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>
</xs:schema>
```

<isbn> 要素が ISBN 列に一度のみマップされ、表の 2 つの行に現れる様子に注目してください。これは分解処理時に自動的に処理されます。ISBN の値ごとに複数の著者が存在するためです。 <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 を超えるので 1 対多の関係が <isbn> 要素と <author> 要素の間で検出されるため、この動作が発生します。

1 対多の関係は 3 つ以上の項目が関係する場合があり、項目のセットが複数含まれる場合があ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される場合があり、既に 1 対多の関係に属する項目が別の 1 対多の関係に属することもできます。

分解アノテーションの例: 値を複数の表にマップする

XML 文書の単一値を複数の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ます。

次の XML 文書について考慮します。

```
<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 32. TEXTBOOKS

ISBN	TITLE
0-11-011111-0	Programming with XML

表 33. SCHOOLPUBS

ISBN	SCHOOL
0-11-011111-0	University of London

次の XML スキーマ文書の断片は、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、title 属性および <university> 要素の値も、マッピングに含まれています。

```
<?xml version="1.0" encoding="UTF-8"?>
<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>"MYSCHEMA"</db2-xdb:defaultSQLSchema>
  </xs:appinfo>
 </xs:annotation>

 <xs:element name="textbook">
  <xs:complexType>
    <xs:sequence maxOccurs="unbounded">

      <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>
```

分解アノテーションの例: 単一の表にマップされる複数の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関係を保持しながら、関連しない複数の要素から取られた複数の値を同一の表にマップできます。以下の例で示すように、これは複数の rowSet を宣言することによって可能になります。rowSet は、関連する項目をグループ化して 1 行にまとめるために使用されます。

例えば、次の XML 文書について考慮します。

```
<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、児童書に関連付けられた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 34. ALLPUBLICATIONS

PUBS_ISBN	PUBS_TITLE
0-11-011111-0	Programming with XML
5-55-555555-5	Children's Fables

単一の表と列の対に単一値をマップするような単純なマッピング・シナリオでは、値をマップする表および列を指定するだけでよい場合もあります。

しかし、この例では、複数の値が同一の表にマップされ、論理的にグループ化する必要があるような、少し複雑なケースを示しています。 rowSet を使用せず、それぞれの ISBN とタイトルを PUBS_ISBN 列と PUBS_TITLE 列にマップするだけである場合は、分解処理ではどの ISBN の値がどのタイトルの値に属するかを判別できません。 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 1 行にまとめることができます。

次の XML スキーマ文書では、2 つの rowSet を定義して、<textbook> 要素の値を <childrensbook> 要素の値と区別する方法を示します。

```
<?xml version="1.0" encoding="UTF-8"?>
<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>"MYSCHEMA"</db2-xdb:defaultSQLSchema>
    <db2-xdb:table>
      <db2-xdb:name>ALLPUBLICATIONS</db2-xdb:name>
      <db2-xdb:rowSet>testbk_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 maxOccurs="unbounded">

     <xs:element name="textbook">
       <xs:complexType>
         <xs:sequence maxOccurs="unbounded">
           <xs:element name="isbn" type="xs:string"
             db2-xdb:rowSet="testbk_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="testbk_rowSet"
             db2-xdb:column="PUBS_TITLE"/>
       </xs:complexType>
     </xs:element>
```

```
    <xs:element name="childrensbook">
      <xs:complexType>
        <xs:sequence maxOccurs="unbounded">
          <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 の名前を指定している様子に注目してください。 rowSet は <db2-xdb:table>アノテーションの ALLPUBLICATIONS 表に関連付けられます。このアノテーションは、<xs:schema> の子として指定する必要があります。

同一の表にマップする複数の rowSet を指定することにより、表の中で論理的に関連した値を確実に 1 行にまとめることができます。

分解アノテーションの例: コンテキストの異なる複数の値を単一の表にマップする

アノテーション付き XML スキーマ分解では、文書のさまざまな部分から取られた値が単一の列に入るように、複数の値を同一の表と列にマップすることができます。以下の例で示すように、これは複数の rowSet を宣言することによって可能になります。

例えば、次の XML 文書について考慮します。

```
<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格納する同一の表に、著者と出版社の両方をマップできます。

表 35. BOOKCONTACTS

ISBN	CONTACT
1-11-111111-1	Alice Braun
1-11-111111-1	Math Pubs

生成される表の 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) 入るといった具合です。

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ます。

```
<?xml version="1.0" encoding="UTF-8"?>
<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>"MYSCHEMA"</db2-xdb:defaultSQLSchema>
    <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 maxOccurs="unbounded">

        <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>
```

要素の各宣言の db2-xdb:rowSet マッピングが、表の名前ではなく rowSet の名前を指定している様子に注目してください。 rowSet は <db2-xdb:table>アノテーションの BOOKCONTACTS 表に関連付けられます。このアノテーションは、<xs:schema> の子として指定する必要があります。

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 タイプの互換性

アノテーション付き XML スキーマ分解は、XML 値から、互換性のある SQL タイプの列への分解をサポートしています。

次の表は、XML スキーマ・データ・タイプ、および XML スキーマ分解に関して互換性のある SQL 文字、グラフィック、および日付のデータ・タイプのリストです。

表 36. 互換性のある XML スキーマと SQL データ・タイプ

	SQL タイプ				
XML スキーマ・タイプ	**CHAR**	**VARCHAR**	**DATE**	**TIME**	**TIMESTAMP**
string、normalizedString、token	1a*	1a	7	8	9
base64Binary,	2*	2	いいえ	いいえ	いいえ
hexBinary	2*	2	いいえ	いいえ	いいえ
byte、unsigned byte	3*	3	いいえ	いいえ	いいえ
integer、positiveInteger、negativeInteger、nonNegativeInteger、nonPositiveInteger	3*	3	いいえ	いいえ	いいえ
int、unsignedInt	3*	3	いいえ	いいえ	いいえ
long、unsignedLong	3*	3	いいえ	いいえ	いいえ
short、unsignedShort	3*	3	いいえ	いいえ	いいえ
decimal	3*	3	いいえ	いいえ	いいえ
float	3*	3	いいえ	いいえ	いいえ
double	3*	3	いいえ	いいえ	いいえ
boolean	3*	3	いいえ	いいえ	いいえ
time	4*	4	いいえ	10	いいえ
dateTime	4*	4	11	12	13
duration、gMonth、gYear、gDay、gMonthDay、gYearMonth	4*	4	いいえ	いいえ	いいえ
date	4*	4	14	いいえ	いいえ
Name、NCName、NOTATION、ID、IDREF、QName、NMTOKEN、ENTITY	1a*	1a	いいえ	いいえ	いいえ
ENTITIES、NMTOKENS、IDREFS、リスト・タイプ	1b*	1b	いいえ	いいえ	いいえ
anyURI	6*	6	いいえ	いいえ	いいえ
language	1a*	1a	いいえ	いいえ	いいえ
anySimpleType、和集合タイプ	5*	5	いいえ	いいえ	いいえ

注

* 入力 XML ストリングの長さがターゲット列の定義済み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。

いいえ データ・タイプは、アノテーション付き XML スキーマ分解では互換性がありません。

1a XML 入力ストリングの文字の長さが、ターゲット列の文字の長さ以下である場合には、互換性があります。入力ストリングがターゲット列よりも長い場合は、この列のマッピングで db2-xdb:truncate が「true」または「1」に設定されている場合のみ、ストリングは互換性があります。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。

1b 1a で記述された条件に基づく互換性があります。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)。

2 XML 入力ストリングの文字の長さが、ターゲット列の文字の長さ以下である場合には、互換性があります。入力ストリングがターゲット列よりも長い場合は、この列のマッピングで db2-xdb:truncate が「true」または「1」に設定されている場合のみ、ストリングは互換性があります。入力ストリングは XML スキーマ・タイプの空白ファセットに従って正規化されます。エンコードされる (元の) ストリングが挿入されます。

3 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。この列のマッピングで db2-xdb:truncate が「true」または「1」に設定されている場合は、互換性があります。

4 db2-xdb:normalization 設定に応じた処理の後に計算される XML 入力値のストリング表現 (文字数) がターゲット列の長さ (文字数) 以下の場合、互換性があります。この列のマッピングで db2-xdb:truncate が「true」または「1」に設定されている場合は、互換性があります。

5 XML 入力ストリングの文字の長さが、ターゲット列の文字の長さ以下である場合には、互換性があります。入力ストリングがターゲット列よりも長い場合は、この列のマッピングで db2-xdb:truncate が「true」または「1」に設定されている場合のみ、ストリングは互換性があります。いずれかの場合に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。

6 URI のストリングの長さ (文字数) がターゲット列の長さ (文字数) 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も長い場合は、この列のマッピングで db2-xdb:truncate が「true」または「1」に設定されている場合のみ、ストリングは互換性があります。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注意してください。

7 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。

8 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。

9 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*)、互換性があります。

10 サブ秒が入った 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。 XML 値が時間帯標識を持ち、db2-xdb:truncate が「true」または「1」に設定されている場合、時間帯標識は時間帯なしで挿入されます。

11 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。

12 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ち、db2-xdb:truncate が「true」または「1」に設定されている場合、値には互換性があります。

13 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ち、db2-xdb:truncate が「true」または「1」に設定されている場合、互換性があります。 6 桁を超えるサブ秒が指定されている場合、db2-xdb:truncate が「true」または「1」に設定されていると互換性があります。

14 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ち、db2-xdb:truncate が「true」または「1」に設定されている場合、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。)

表 37. 互換性のある XML スキーマと SQL データ・タイプ

	SQL タイプ			
XML スキーマ・タイプ	GRAPHIC	VARGRAPHIC	CLOB	DBCLOB
string、normalizedString、token	1a*	1a	1a	1a*
base64Binary,	いいえ	いいえ	3	いいえ
hexBinary	いいえ	いいえ	3	いいえ
byte、unsigned byte	いいえ	いいえ	4	いいえ
integer、positiveInteger、negativeInteger、nonNegativeInteger、nonPositiveInteger	いいえ	いいえ	4	いいえ
int、unsignedInt	いいえ	いいえ	4	いいえ
long、unsignedLong	いいえ	いいえ	4	いいえ
short、unsignedShort	いいえ	いいえ	4	いいえ
decimal	いいえ	いいえ	4	いいえ
float	いいえ	いいえ	4	いいえ
double	いいえ	いいえ	4	いいえ
boolean	いいえ	いいえ	4	いいえ
time	いいえ	いいえ	5	いいえ
dateTime	いいえ	いいえ	5	いいえ
duration、gMonth、gYear、gDay、gMonthDay、gYearMonth	いいえ	いいえ	5	いいえ
date	いいえ	いいえ	5	いいえ
Name、NCName、NOTATION、ID、IDREF、QName、NMTOKEN、ENTITY	1a*	1a	1a	1a*
ENTITIES、NMTOKENS、IDREFS、リスト・タイプ	1b*	1b	1b	1b*
anyURI	いいえ	いいえ	6	いいえ
language	いいえ	いいえ	1a	いいえ
anySimpleType、和集合タイプ	2a*	2a	2a	2a*

注

* 入力 XML ストリングの長さがターゲット列の定義済み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。

いいえ データ・タイプは、アノテーション付き XML スキーマ分解では互換性がありません。

1a XML 入力ストリングの長さ (2 バイト文字の数) が、ターゲット列の長さ (文字数) 以下である場合には、互換性があります。入力ストリングがターゲット列よりも長い場合は、この列のマッピングで db2-xdb:truncate が「true」または「1」に設定されている場合のみ、ストリングは互換性があります。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。

1b 1a で記述された条件に基づく互換性があります。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)。

2a XML 入力ストリングの文字の長さが、ターゲット列の文字の長さ以下である場合には、互換性があります。入力ストリングがターゲット列よりも長い場合は、この列のマッピングで

db2-xdb:truncate が「true」または「1」に設定されている場合のみ、ストリングは互換性があります。いずれかの場合のターゲットに挿入される値は、要素または属性の文字内容です。

3 XML 入力ストリングの文字の長さが、ターゲット列の文字の長さ以下である場合には、互換性があります。入力ストリングがターゲット列よりも長い場合は、この列のマッピングで db2-xdb:truncate が「true」または「1」に設定されている場合のみ、ストリングは互換性があります。入力ストリングは XML スキーマ・タイプの空白ファセットに従って正規化されます。エンコードされる (元の) ストリングが挿入されます。

4 db2-xdb:normalization 設定に応じた処理の後に計算される XML ストリングの長さがターゲット列の長さ (文字数) 以下の場合、互換性があります。この列のマッピングで db2-xdb:truncate が「true」または「1」に設定されている場合は、互換性があります。

5 db2-xdb:normalization 設定に応じた処理の後に計算される XML 入力値のストリング表現 (文字数) がターゲット列の長さ (文字数) 以下の場合、互換性があります。この列のマッピングで db2-xdb:truncate が「true」または「1」に設定されている場合は、互換性があります。

6 URI のストリングの長さ (文字数) がターゲット列の長さ (文字数) 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も長い場合は、この列のマッピングで db2-xdb:truncate が「true」または「1」に設定されている場合のみ、ストリングは互換性があります。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注意してください。

以下の表は、XML スキーマ・データ・タイプ、および XML スキーマの分解に関して互換性のある SQL バイナリー・データ・タイプのリストです。

表 38. 互換性のある XML スキーマと SQL データ・タイプ

	SQL タイプ		
XML スキーマ・タイプ	**BINARY**	**VARBINARY**	**BLOB**
string、normalizedString、token	1a*	1a	1a
base64Binary、	1a	1a	1a
hexBinary	2*	2	2
byte、unsigned byte	いいえ	いいえ	いいえ
integer、positiveInteger、negativeInteger、nonNegativeInteger、nonPositiveInteger	いいえ	いいえ	いいえ
int、unsignedInt	いいえ	いいえ	いいえ
long、unsignedLong	いいえ	いいえ	いいえ
short、unsignedShort	いいえ	いいえ	いいえ
decimal	いいえ	いいえ	いいえ
float	いいえ	いいえ	いいえ
double	いいえ	いいえ	いいえ
boolean	いいえ	いいえ	いいえ
time	いいえ	いいえ	いいえ
dateTime	いいえ	いいえ	いいえ
duration、gMonth、gYear、gDay、gMonthDay、gYearMonth	いいえ	いいえ	いいえ
date	いいえ	いいえ	いいえ
Name、NCName、NOTATION、ID、IDREF、QName、NMTOKEN、ENTITY	1a*	1a	1a
ENTITIES、NMTOKENS、IDREFS、リスト・タイプ	1b*	1b	1b
anyURI	1a	1a	1a

表 38. 互換性のある XML スキーマと SQL データ・タイプ (続き)

	SQL タイプ		
XML スキーマ・タイプ	BINARY	VARBINARY	BLOB
language	1a*	1a	1a
anySimpleType、和集合タイプ	3*	3	3

注

* 入力 XML ストリングの長さがターゲット列の定義済み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。

いいえ データ・タイプは、アノテーション付き XML スキーマ分解では互換性がありません。

1a XML 入力ストリングの文字の長さが、ターゲット列の文字の長さ以下である場合には、互換性があります。入力ストリングがターゲット列よりも長い場合は、この列のマッピングで db2-xdb:truncate が「true」または「1」に設定されている場合のみ、ストリングは互換性があります。いずれかの場合のターゲットに挿入される値は、要素または属性の文字内容です。

1b 1a で記述された条件に基づく互換性があります。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)。

2 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も長い場合は、この列のマッピングで db2-xdb:truncate が「true」または「1」に設定されている場合のみ、ストリングは互換性があります。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。エンコードされる (元の) ストリングが挿入されます。

3 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も長い場合は、この列のマッピングで db2-xdb:truncate が「true」または「1」に設定されている場合のみ、ストリングは互換性があります。いずれかの場合に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。

以下の表は、XML スキーマ・データ・タイプ、および XML スキーマの分解に関して互換性のある SQL 数値データ・タイプのリストです。

表 39. 互換性のある XML スキーマと SQL データ・タイプ

	SQL タイプ					
XML スキーマ・タイプ	SMALLINT	INTEGER	BIGINT	REAL	DOUBLE	DECFLOAT
string、normalizedString、token	1	1	1	1	1	1
base64Binary,	いいえ	いいえ	いいえ	いいえ	いいえ	いいえ

表 39. 互換性のある XML スキーマと SQL データ・タイプ (続き)

	SQL タイプ					
XML スキーマ・タイプ	SMALLINT	INTEGER	BIGINT	REAL	DOUBLE	DECFLOAT
hexBinary	いいえ	いいえ	いいえ	いいえ	いいえ	いいえ
byte、unsigned byte	2	2	2	2	2	2
integer、positiveInteger、negativeInteger、nonNegativeInteger、nonPositiveInteger	3	3	3	4	4	3
int、unsignedInt	3	2	2	4	2	2
long、unsignedLong	3	3	2	4	2	2
short、unsignedShort	2	2	2	2	2	2
decimal	4	4	4	4	4	4
float	5	5	5	6	6	6
double	5	5	5	5	6	6
boolean	7	7	7	7	7	7
time	いいえ	いいえ	いいえ	いいえ	いいえ	いいえ
dateTime	いいえ	いいえ	いいえ	いいえ	いいえ	いいえ
duration、gMonth、gYear、gDay、gMonthDay、gYearMonth	いいえ	いいえ	いいえ	いいえ	いいえ	いいえ
date	いいえ	いいえ	いいえ	いいえ	いいえ	いいえ
Name、NCName、NOTATION、ID、IDREF、QName、NMTOKEN、ENTITY	いいえ	いいえ	いいえ	いいえ	いいえ	いいえ
ENTITIES、NMTOKENS、IDREFS、リスト・タイプ	いいえ	いいえ	いいえ	いいえ	いいえ	いいえ
anyURI	いいえ	いいえ	いいえ	いいえ	いいえ	いいえ
language	いいえ	いいえ	いいえ	いいえ	いいえ	いいえ
anySimpleType、和集合タイプ	いいえ	いいえ	いいえ	いいえ	いいえ	いいえ

注

いいえ データ・タイプは、アノテーション付き XML スキーマ分解では互換性がありません。

1 以下の条件が真である場合に互換性があります。

- ストリングがターゲット・タイプの XML スキーマ字句表記規則に従っている。
- 切り捨てや有効数字の喪失が発生することなく、ストリングを数値に変換できる。

2 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 がデータベースに保管されます。

3 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 がデータベースに保管されます。

4 切り捨てや有効数字の喪失が発生することなく値を数値に変換できる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 がデータベースに保管されます。

5 切り捨てや有効数字の喪失が発生することなく値を数値に変換でき、値が「INF」、「-INF」、または「NaN」のいずれでもない場合に互換性があります。 -0 は 0 としてデータベースに保管されます。

6 値が「INF」、「-INF」、または「NaN」のいずれでもない場合に互換性があります。 -0 は 0 としてデータベースに保管されます。

7 互換性があります。挿入される値は「0」(false の場合)、または「1」(true の場合) です。

表 40. 互換性のある XML スキーマと SQL データ・タイプ

	SQL タイプ	
XML スキーマ・タイプ	DECIMAL	NUMERIC
string、normalizedString、token	1	1
base64Binary,	いいえ	いいえ
hexBinary	いいえ	いいえ
byte、unsigned byte	2	2
integer、positiveInteger、negativeInteger、nonNegativeInteger、nonPositiveInteger	4	4
int、unsignedInt	4	4
long、unsignedLong	4	4
short、unsignedShort	2	2
decimal	4	4
float	5	5
double	5	5
boolean	7	7
time	いいえ	いいえ
dateTime	いいえ	いいえ
duration、gMonth、gYear、gDay、gMonthDay、gYearMonth	いいえ	いいえ
date	いいえ	いいえ
Name、NCName、NOTATION、ID、IDREF、QName、NMTOKEN、ENTITY	いいえ	いいえ
ENTITIES、NMTOKENS、IDREFS、リスト・タイプ	いいえ	いいえ
anyURI	いいえ	いいえ
language	いいえ	いいえ
anySimpleType、和集合タイプ	いいえ	いいえ

注

いいえ データ・タイプは、アノテーション付き XML スキーマ分解では互換性がありません。

1 以下の条件が真である場合に互換性があります。

- ストリングがターゲット・タイプの XML スキーマ字句表記規則に従っている。
- 切り捨てや有効数字の喪失が発生することなく、ストリングを数値に変換できる。

2 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 がデータベースに保管されます。

3 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 がデータベースに保管されます。

4 切り捨てや有効数字の喪失が発生することなく値を数値に変換できる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 がデータベースに保管されます。

5 切り捨てや有効数字の喪失が発生することなく値を数値に変換でき、値が「INF」、「-INF」、または「NaN」のいずれでもない場合に互換性があります。 -0 は 0 としてデータベースに保管されます。

6 値が「INF」、「-INF」、または「NaN」のいずれでもない場合に互換性があります。 -0 は 0 としてデータベースに保管されます。

7 互換性があります。挿入される値は「0」(false の場合)、または「1」(true の場合) です。

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 41. 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 のストリング値の最大長	1024 バイト
db2-xdb:condition のストリング値の最大長	1024 バイト
db2-xdb:locationPath 内の最大ステップ数	128
XML スキーマ内の固有のアノテーションの最大数	64 KB
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

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。

- 要素または属性の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、ワイルドカード要素またはワイルドカード属性の内容は直列化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。これらのワイルドカード要素またはワイルドカード属性は、対応する <xs:any> 宣言または <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

  グループ・ヘッドが XML スキーマに存在する XML 文書に、その置換グループのメンバーが存在する場合、エラーが生成されます。
- xsi:type を使用した実行時置換

  要素は、XML スキーマの要素名に関連付けられたスキーマ・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、分解時にエラーが出されます。
- 再帰的要素 (自分自身を参照する要素)

  再帰的要素宣言が含まれているアノテーション付き XML スキーマを分解可能にすることはできません。
- ターゲット表内の既存の行の更新または削除

  現時点で、分解では、ターゲット表に新しい行しか挿入できません。
- 分解によって更新される表間の参照制約

この制限を回避するには、同じインスタンス文書における複数の分解操作に、それぞれ別のアノテーション付きスキーマを使用して、親または子の表が更新される順序を制御します。

- 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 および列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 と列に正式にマップできる場合、マッピングに db2-xdb:expression や db2-xdb:condition アノテーションが含まれていてはなりません。

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できるようにするための一連の分解アノテーション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>
```

XML データ・モデル

XML データ・モデルは、XPath 2.0 および XQuery 1.0 データ・モデルに従います。このデータ・モデルでは、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・モデルの目的は、中間計算時に使用される値を含め、XPath の式ですべての暗黙的値を定義することです。すべての XPath 式は、データ・モデルのインスタンスを入力とし、データ・モデルのインスタンスを戻します。XML データ・モデルは、シーケンスと項目、アトミック値、およびノードに関して記述されます。

シーケンスおよび項目

XPATH データ・モデルは、シーケンスの概念に基づいています。 XPath 式の値は、常にシーケンスです。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、アトミック値かノードのいずれかです。

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、以下の各値は、それぞれ単一のシーケンスとして表すことができます。

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

ある 1 つのノードが複数のシーケンスにあってもかまいません。また、シーケンス内に複数の重複した項目があってもかまいません。シーケンスは、別のシーケンスのメンバーになることはできません。言い換えると、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になります。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。

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してモデル化されます。例えば、シーケンス (2) とアトミック値 2 との間に区別はありません。

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・タイプのいずれか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、およびその他のアトミック・タイプが含まれます。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」と呼ばれています。アトミック・タイプには、リテラル値を持つものがあります。例えば、以下のリテラルは、アトミック値です。

- "this is a string"
- 45
- 1.44

また、ストリングからアトミック値を作成するコンストラクター関数をもつアトミック・タイプもあります。例えば、以下のコンストラクター関数は、ストリング "12.34" から xs:decimal タイプの値を作成します。

```
xs:decimal("12.34")
```

ノード

ノード は、XPath に定義されているノードのタイプのいずれかに準拠します。これらのノード・タイプには、文書、エレメント、属性、テキスト、処理命令、コメント、およびネーム・スペースがあります。

シーケンスのノードは、1 つの文書ノードと、その文書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 つ以上のツリーを形成します。すべてのノードは、必ず 1 つのツリーに属し、すべてのツリーには必ず 1 つの文書ノードがあります。ルート・ノードが文書ノードであるツリーは、*文書* といいます。ルート・ノードが文書ノードでないツリーは、*フラグメント* といいます。

以下の XML 文書には、`product` という文書エレメントが含まれており、その文書エレメントには `description` エレメントが含まれています。`product` エレメントには、`pid` (購入注文 ID) という属性があります。`description` エレメントには、`name`、`details`、`price`、および `weight` というエレメントが含まれています。

```
<product xmlns="http://posample.org" pid="100-101-01">
  <description>
    <name>Snow Shovel, Deluxe 24"</name>
    <details>A Deluxe Snow Shovel, 24 inches wide, ergonomic
            curved handle with D-Grip</details>
    <price>19.99</price>
    <weight>2 kg</weight>
  </description>
</product>
```

ノード ID

各ノードには固有の ID があります。このことは、 2 つのノードの名前や値が同じでも区別できることを意味します。一方、アトミック値には ID はありません。アトミック値 (整数 7 など) のすべてのインスタンスは、その値の他のすべてのインスタンスと同一です。

文書順序

階層内のすべてのノード間には、文書順序と呼ばれる全体的な順序付けがあり、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 XML フォーマットで表される際にノードが表示される順序に対応します。

- 文書ノードが 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。
- ネーム・スペース・ノードは、関連付けられているエレメント・ノードの直後に表示されます。
- 属性ノードは、ネーム・スペース・ノードの後に表示され、ネーム・スペース・ノードが無い場合は関連するエレメント・ノードの後に表示されます。

  属性ノードとネーム・スペース・ノードは、エレメント・ノードの子ではありませんが、関連するエレメント・ノードはそれらの親ノードです。

  属性ノードの相対順序は任意ですが、この順序は XPath 式の処理時も変わりません。
- エレメント・ノード、テキスト・ノード、処理命令ノード、およびコメント・ノードは、エレメント・ノードまたは文書ノードの子になることができます。
- 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。

ノード名
ノードの名前 (QName として表されます)。

parent 現行ノードの親であるノード。

タイプ名
ノードの動的 (ランタイム) タイプ。

children
現行ノードの子 であるノードのシーケンス。

属性 現行ノードに属する属性ノードのセット。

ストリング値
ノードから抽出可能なストリング値。

型付きの値
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。

target 処理命令の送信先であるアプリケーションを示します。ターゲットは NCName (コロンが無いローカル名) です。

content
処理命令、テキスト・ノード、またはコメント・ノードの内容。

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、親ノードを持つことはできませんが、下位ノード (子) を持つことはできます (下位ノードを持たない場合もあります)。下位ノード (子) には、エレメント・ノード、テキスト・ノード、処理命令ノード、コメント・ノードがあります。よく整理された文書にするには、文書ノードが、子エレメント・ノードを 1 つだけ持っており、子テキスト・ノードを持たないようにします。

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
- ストリング値
- 型付きの値

文書ノードでは、ストリング値は、そのすべての下層テキスト・ノードのストリング値を文書順序に従って連結した値であり、型付きの値は、ストリング値と同じ値で、タイプは xs:untypedAtomic です。

例えば、以下のようなテキスト表記を持つ文書があるとします。

```
<product xmlns="http://posample.org" pid="100-101-01">
  <description>
    <name>Snow Shovel, Deluxe 24"</name>
    <details>A Deluxe Snow Shovel, 24 inches wide, ergonomic
            curved handle with D-Grip</details>
    <price>19.99</price>
    <weight>2 kg</weight>
  </description>
```

文書ノードには、以下のプロパティー値があります。

表 42. 文書ノードのプロパティー

ノード・プロパティー	値	値のタイプ
children	製品ノード	
ストリング値	"Snow Shovel, Deluxe 24"A Deluxe Snow Shovel, 24 inches wide, ergonomic curved handle with D-Grip19.992 kg"	xs:string
型付きの値	"Snow Shovel, Deluxe 24"A Deluxe Snow Shovel, 24 inches wide, ergonomic curved handle with D-Grip19.992 kg"	xs:untypedAtomic

要素ノード

要素ノードは、XML 要素をカプセル化します。

エレメントは、親を持たないか 1 個の親、および子を持たないか 1 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。 エレメント・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。

- ノード名
- parent
- タイプ名 (DB2 におけるエレメント・ノードのタイプ名は、常に xs:untyped です。)

- children
- 属性
- ストリング値
- 型付きの値
- 範囲内の名前空間

エレメント・ノードでは、ストリング値は、そのすべての下層テキスト・ノードのストリング値を文書順序に従って連結した値です。エレメントが空の場合、ストリング値は空ストリング "" になります。エレメントの型付きの値は、以下のいずれかの値になります。

- エレメントが NULL 可能である場合、型付きの値は () です。
- エレメントが空の場合、型付きの値は空のシーケンス () です。
- それ以外の場合は、型付きの値はそのストリング値になり、タイプ xs:untypedAtomic です。

例えば、以下のようなテキスト表記を持つ文書があるとします。

```
<product xmlns="http://posample.org" pid="100-101-01">
  <description>
    <name>Snow Shovel, Deluxe 24"</name>
    <details>A Deluxe Snow Shovel, 24 inches wide, ergonomic
            curved handle with D-Grip</details>
    <price>19.99</price>
    <weight>2 kg</weight>
  </description>
```

製品エレメント・ノードには、以下のプロパティー値があります。

表 43. 製品ノードのプロパティー

ノード・プロパティー	値	値のタイプ
ノード名	製品	
parent	文書ノード	
タイプ名	xs:untyped	
children	記述ノード	
属性	pid	
ストリング値	"Snow Shovel, Deluxe 24"A Deluxe Snow Shovel, 24 inches wide, ergonomic curved handle with D-Grip19.992 kg"	xs:string
型付きの値	"Snow Shovel, Deluxe 24"A Deluxe Snow Shovel, 24 inches wide, ergonomic curved handle with D-Grip19.992 kg"	xs:untypedAtomic
in-scope namespaces	(default, http://posample.org)	

名前エレメント・ノードには、以下のプロパティー値があります。

表 44. 名前ノードのプロパティー

ノード・プロパティー	値	値のタイプ
ノード名	name	
parent	記述ノード	
タイプ名	xs:untyped	
children	テキスト・ノード "Snow Shovel, Deluxe 24" "	

表 44. 名前ノードのプロパティー (続き)

ノード・プロパティー	値	値のタイプ
属性	なし	
ストリング値	"Snow Shovel, Deluxe 24" "	xs:string
型付きの値	"Snow Shovel, Deluxe 24" "	xs:untypedAtomic
in-scope namespaces	(default, http://posample.org)	

属性ノード

属性ノード は、XML 属性を表します。

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。

- ノード名
- parent
- タイプ名 (DB2 における属性ノードのタイプ名は、常に xs:untypedAtomic です。)
- ストリング値
- 型付きの値

属性ノードの場合、ストリング値は、属性の正規化値であるか、または属性がスキーマにより妥当性検査されている場合は属性のスキーマ正規化値になります。型付きの値は、xs:untypedAtomic タイプのストリング値と同じです。

```
<product xmlns="http://posample.org" pid="100-101-01">
  <description>
    <name>Snow Shovel, Deluxe 24"</name>
    <details>A Deluxe Snow Shovel, 24 inches wide, ergonomic
            curved handle with D-Grip</details>
    <price>19.99</price>
    <weight>2 kg</weight>
  </description>
```

pid 属性には、以下のプロパティー値があります。

表 45. *pid* 属性ノードのプロパティー

ノード・プロパティー	値	値のタイプ
ノード名	pid	
parent	製品ノード	
タイプ名	xs:untypedAtomic	
ストリング値	"100-101-01"	xs:string
型付きの値	100-101-01"	xs:untypedAtomic

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。 テキスト・ノードの内容は、空であってもかまいません。ただし、テキスト・ノードの親が空でない限り、テキスト・ノードの内容を空ストリングにすることはできません。文書ノードまたはエレメント・ノードの子であるテキスト・ノードが、隣接する

兄弟として表示されることはありません。文書ノードまたはエレメント・ノードの構成時に、隣接する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
- parent

例えば、以下のようなテキスト表記を持つ文書があるとします。

```
<product xmlns="http://posample.org" pid="100-101-01">
  <description>
    <name>Snow Shovel, Deluxe 24"</name>
    <details>A Deluxe Snow Shovel, 24 inches wide, ergonomic
            curved handle with D-Grip</details>
    <price>19.99</price>
    <weight>2 kg</weight>
  </description>
```

名前エレメント・ノードの下のテキスト・ノードには、以下のプロパティー値があります。

表 46. 名前テキスト・ノードのプロパティー

ノード・プロパティー	値
content	Snow Shovel, Deluxe 24"
parent	name

テキスト・ノードのストリング値は、ノードの内容です。上記の例では、「Snow Shovel, Deluxe 24"」になります。テキスト・ノードの型付きの値は、ストリング値と同じ値で、タイプ xs:untypedAtomic です。

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。 処理命令のターゲットは、 NCName (コロンが付いていないローカル名) にする必要があります。

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

例えば、以下の処理命令を考えてみます。

```
<?xml-stylesheet href="book.css" type="text/css"?>
```

この処理命令には、以下のプロパティー値があります。

表 47. 処理命令ノードのプロパティー

ノード・プロパティー	値
target	xml-stylesheet
content	href="book.css" type="text/css"
parent	文書ノード

処理命令ノードのストリング値は、ノードの内容です。上記の場合では、href="book.css" type="text/css" になります。型付きの値はストリング値と同じ値で、タイプ xs:string でもあります。

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、親を持たないか 1 個の親を持つことができます。

コメント・ノードは、以下のプロパティーを持ちます。

- content
- parent

例えば、以下のコンテンツを考えてください。

```
 <ID>
<!-- This element contains an ID number. -->
101
</ID>
```

このコメントには、以下のプロパティー値があります。

表 48. コメント・ノードのプロパティー

ノード・プロパティー	値
content	This element contains an ID number.
parent	ID ノード

コメント・ノードのストリング値は、ノードの内容です。上記の例では、 This element contains an ID number. になります。型付きの値はストリング値と同じ値で、タイプ xs:string でもあります。

データ・モデルの生成

XPath 式を処理するには、その前に入力文書を XML データ・モデルで表す必要があります。

入力 XML 文書は、*XML 構文解析* と呼ばれる処理を通じて XML データ・モデルのインスタンスに変換されます。あるいは、XMLELEMENT や XMLATTRIBUTES などの SQL XML コンストラクターを使用して XML データ・モデルのインスタンスを生成することができます。これらの組み込み関数を使用して、リレーショナル・データから XML データを生成することができます。同様に、*XML シリアライゼーション* と呼ばれる処理を通じて XPath 式の結果を XML 表記に変換することができます。

- *XML 構文解析* 時に、XML 文書のストリング表記は、XPath モデルのインスタンスに変換されます。オプションで、特定のスキーマに対して XML 文書の妥当性検査を行うことができます。構文解析されたデータは、ノードとアトミック値の階層で表されます。XPATH データ・モデルのそれぞれのアトミック値、エレメント・ノード、および属性ノードには、動的タイプのアノテーションが付いています。動的タイプは、値の範囲を指定します。例えば、 version という属性の動的タイプが xs:decimal である場合は、属性に 10 進数値が含まれていることを示しています。

  制約事項: スキーマに対して XML 文書の妥当性検査を行う場合、DB2 はノードごとのタイプ・アノテーションを維持しません。データは型なしとして保管されます。

  属性の値は、属性ノード内で直接表されます。不明なタイプの属性ノードには、動的タイプ xs:untypedAtomic のアノテーションが付けられます。

エレメントの値は、エレメント・ノードの子によって表されます。これには、テキスト・ノードおよびその他のエレメント・ノードが該当します。エレメント・ノードの動的タイプは、子テキスト・ノードの値の解釈方法を示します。すべてのエレメント・ノードには、タイプ xs:untyped があります。

不明なタイプのアトミック値には、タイプ xs:untypedAtomic のアノテーションが付けられます。

入力文書にスキーマがない場合、その文書は妥当性検査されません。ノードおよびアトミック値は DB2 により、型なし (xs:untyped または xs:untypedAtomic) であると指定されます。

- シリアライゼーション 時に、ノードおよびアトミック値 (XPATH データ・モデルのインスタンス) のシーケンスは、そのストリング表記に変換されます。シリアライゼーションの結果が、必ずしも完全な文書表記になるとは限りません。実際、シリアライゼーションの結果が単一のアトミック値 (17 など) や、共通の親を持たないエレメントのシーケンスとなることがあります。

SQLでの XML 値

XQuery/XPath 2.0 データ・モデルという観点からは、DB2 for i SQL は、文書の内容を含むサブツリーを使用して、XML 値を、単一の文書ノードを含むシーケンスとして定義します。 XML 値を文書ノードとして表すと、確実に、XML 値を正確に表す文字表現となるように値をシリアライズできるようになります。この定義は、2008 Database Language SQL Part 14 - XML-Related Specifications (ISO 9075-14) で XML(CONTENT) として言及されています。

XML 値がタイプ XML(CONTENT) となるように、SQL での XML 値の作成またはコピー時に文書ノードが構成されます。

文書ノードの構成時に以下の処理が行われます。

1. 内容シーケンスに文書ノードが含まれる場合、その文書ノードはその子に置き換えられます。
2. 内容シーケンス内のアトミック値は、すべてストリングに変換されてテキスト・ノードに保管されます。これらは、構成された文書の子になります。
3. 内容シーケンス内の隣接するテキスト・ノードは、単一のテキスト・ノードにマージされます。
4. 内容シーケンスに属性ノードが含まれる場合、エラーとなります。

文書ノードに対しては妥当性検査は実行されません。 XML 値を構成するときには、XML 文書の構造を扱う XML 1.0 規則 (例えば、文書ノードは要素ノードである子を 1 つだけ持つ必要があるという規則) が強制されません。

例えば、次のとおりです。

```
XMLCONCAT(XMLTEXT('text node one '), XMLTEXT('text node two'))
```

結果の XML 値は、単一の子テキスト・ノード「text node one text node two」を持つ単一文書ノードのシーケンスとして表されます。この表現は、シリアライズされた XML 文書と整合性があります。

この仕様のその他の実装 (DB2 for z/OS® および DB2 for LUW を含む) では、SQL XML タイプを、タイプに関係なくノードまたはアトミック値の項目をいくつでも含むことができる XQuery/XPath 2.0 シーケンスとして定義する場合があります。この定義は、シーケンスを正確に表す文字表現となるように値をシリアライズできることを保証するものではありません。このタイプは、仕様では XML(SEQUENCE) として参照され、XML(CONTENT) のスーパーセットです。

タイプ XML(SEQUENCE) の値を作成する SQL 実装では、前の XMLCONCAT 式の結果が、親文書ノードを持たない隣接する 2 つのテキスト・ノードのシーケンスとして表現されます ('text node one', 'text node two')。

ほとんどの場合、表現でのこの差異は重要ではありません。 XML(SEQUENCE) という値がシリアライズされるときには、まず XML(CONTENT) 値の作成時と同じプロセスで XML(CONTENT) に変換されます。したがって、XML 値をシリアライズすると、実装で使用された XML タイプに関係なく、同じ結果が生じます。

さらに、表内の列、ホスト変数、パラメーター・マーカーから取得したか、または XMLPARSE 組み込み関数を使用して取得した正しい形式の XML 文書には、ルート文書ノードが含まれます。これにより、より一般的なタイプの XML(SEQUENCE) を実装する環境であっても、XML 値は既に、より限定的なタイプの XML(CONTENT) になっています。

XML 値が SQL で構成され、XPath 式で評価されるケースは多くありませんが、このようなケースでは異なる結果が生じることがあります。

例:

```
select XMLSERIALIZE(OUTPUT_COL AS VARCHAR(100)) from
 XMLTABLE('$d_or_e/root/child'
           passing XMLELEMENT(NAME "root",
                              XMLELEMENT(NAME "child",
                                         XMLTEXT('hello world'))) as "d_or_e"
          ) X(OUTPUT_COL);
```

DB2 for i は、XML 値を表わす文書ノードに `$d_or_e` を割り当てます。このステップ式が評価され、予想される出力 `<child>hello world</child>` が戻されます。`"root"` は、文書ノードの子要素です。

しかし、DB2 for LUW および DB2 for z/OS は `$d_or_e` を要素 `"root"` に割り当てます。要素 `"root"` は `"root"` という名前の子を持たないため、この照会で行は戻されません。

プラットフォームにまったく依存しないソリューションを提供する正しい方法は、文書ノードが明示的に構成されるように照会を作成することです。これにより、仕様に準拠したすべてのプラットフォームで確実に表現が同じになります。

```
select XMLSERIALIZE(OUTPUT_COL AS VARCHAR(100)) from
 XMLTABLE('$d_or_e/root/child' passing
                    XMLDOCUMENT(
                               XMLELEMENT(NAME "root",
                                          XMLELEMENT(NAME "child",
                                                     XMLTEXT('hello world')))
                               ) as "d_or_e"
 ) X(OUTPUT_COL);
```

XML(CONTENT) となることが保証された XML 値の場合、文書ノードの明示的な構成は必要ありません。例えば以下のような場合です。

- 値がホスト変数またはパラメーター・マーカーを介してアプリケーションから得られる場合。
- 値が DB2 表の列から得られる場合。
- XML 値が XMLPARSE 組み込み関数を使用して明示的にまたは暗黙的に作成される場合。

XPath の概要

XPath は、XQuery 1.0 および XPath 2.0 データ・モデルに準拠した XML データを処理できるように、World Wide Web Consortium (W3C) によって設計された式言語です。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。 DB2 XPath は、XPath 2.0 勧告の言語構造体のサブセットをサポートします。

XPath 言語は、キーワード、記号、およびオペランドから構成できるいくつかの種類の式を提供します。ほとんどの場合、さまざまな式、演算子、および関数のオペランドを期待されるタイプに合致させる必要があります。DB2 は、特定の状態において、タイプ・エラーを無視します。

DB2 XPath は、XML 値をリレーショナル結果セットに変換するために使用される XMLTABLE SQL 組み込み表関数に対する引数として使用できます。

XPath 式

XPath の基本的な構成単位は式です。 DB2 XPath は、XML データを処理するためのいくつかの種類の式を提供します。

- リテラル、変数参照、関数呼び出しなど、言語の基本的なプリミティブが含まれる 1 次式。
- 文書ツリー内のノードを位置付けるためのパス式。
- 加算、減算、乗算、除算、およびモジュラスの算術式。
- 2 つの値を比較する比較式。
- ブール・ロジックを使用する論理式。

XPath 式は、完全な一般性をもって構成できます。つまり、式が予期されているところであれば、どの種類の式でも使用できます。一般的に、式のオペランドは他の式になっています。以下の例で、論理式のオペランドは、比較式 `1 = 1` および `2 = 2` です。

```
1 = 1 and 2 = 2
```

XPath 処理

XPath 式は、処理環境を設定するオプションの*プロローグ* と結果を生成する式 から構成されます。 XPath 処理は、静的分析フェーズと動的評価フェーズという 2 フェーズで実行されます。

静的分析フェーズで、式は、プロローグに定義されている情報に基づいて構文解析され、増大されます。静的コンテキストは、式で使用されるタイプ名、関数名、変数名を解決するために使用されます。*静的コンテキスト* には、式を評価する前に使用可能なすべての情報が含まれています。静的フェーズは、式が最初に評価されるときに実行されます。必要な名前が静的コンテキストで見つからない場合は、エラーが発生します。

動的評価フェーズは、静的分析フェーズでエラーが検出されなかった場合に発生します。動的評価フェーズでは、式の値が計算されます。動的タイプは、値が計算されるときに各値に関連付けられます。式のオペランドに、期待されるタイプと一致しない動的タイプがある場合は、タイプ・エラーになります。評価でエラーが生成されなかった場合は、結果が戻されます。*動的コンテキスト* には、式が評価されるときに使用可能な情報が含まれています。

XPath 式の結果は、一般的に XML ノードとアトミック値の異種混合のシーケンスです。さらに具体的には、XPath 式の結果は、XPath データ・モデルのインスタンスです。

XPath 2.0 および XQuery 1.0 データ・モデル

XPath 2.0 および XQuery 1.0 データ・モデルは、XML 文書を、XML エレメントおよび属性を表すノードの階層 (ツリー) として表します。データ・モデルの各値は、項目がゼロ、1 つ、または複数含まれているシーケンスです。項目は、アトミック値またはノードです。すべての XPath 式は、XPath 2.0 および XQuery 1.0 データ・モデルのインスタンスを入力とし、XPath 2.0 および XQuery 1.0 データ・モデルのインスタンスを戻します。

DB2 XPath データ・タイプ

DB2 XPath は、以下のデータ・タイプをサポートします。

- xs:integer
- xs:decimal
- xs:double
- xs:string
- xs:boolean
- xs:untypedAtomic
- xs:date
- xs:dateTime
- xs:time
- xs:duration
- xs:yearMonthDuration
- xs:dayTimeDuration

DB2 は、動的評価フェーズおよび静的分析フェーズでデータ・タイプを検査します。式において不適切なタイプが検出されると、タイプ・エラーが発生します。例えば、正演算子 (+) を使用して 2 つのストリングを結合する XPath 式は、タイプ・エラーになります。これは、正演算子が数値、yearMonthDuration 値、および dayTimeDuration 値を加算するための算術式のみで使用されるためです。式で期待されるタイプを提供するために、可能であれば、暗黙的なタイプ変換およびタイプ置換が行われます。

組み込み関数ライブラリー

DB2 XPath は、XML データを処理するための組み込み関数のライブラリーを提供します。ライブラリーには、以下のタイプの関数が含まれています。

- ストリング関数
- 数字関数
- 日付/時刻関数
- ブール値を操作する関数
- シーケンスを操作する関数

これらの組み込み関数は、デフォルトで接頭部 `fn` と関連付けられている、URI `http://www.w3.org/2005/xpath-functions` を持つネーム・スペースにあります。デフォルト関数ネーム・スペースは、デフォルトで `fn` に設定されています。これは、接頭部を指定しなくても、このネーム・スペース内の関数を呼び出せることを意味します。

関数呼び出しは、XPath 式内で式が期待されるどの場所でも使用できます。

DB2 XPath における大/小文字の区別

XPath は大/小文字を区別する言語です。

XPath のキーワードは小文字を使用し、予約はされていません。 XPath 式での名前は、言語キーワードと同じにすることができます。

DB2 XPath での空白文字

空白文字は式の構文の一部でない場合であっても、読みやすさを向上させるためにほとんどの XPath 式で使用できます。空白文字は、スペース文字 (U+0020)、復帰 (U+000D)、改行 (U+000A)、またはタブ (U+0009) から成ります。

一般に、空白文字は、空白文字が保持される以下のような状況を除いては、XPath 式では重要ではありません。

- 空白がストリング・リテラル内にある。
- 空白が、2 つの隣接するトークンが誤って 1 つのトークンとして認識されることを防ぐことにより、式を明確にしている。

以下に、明確にするために空白文字が必要な式の例を示します。

- `one- two` は、構文エラーになります。パーサーは、 one- を単一の QName (修飾名) と認識し、演算子が見つからない時点でエラーを戻します。
- `one -two` は、構文エラーになりません。パーサーは、`one` を QName として、負符号 (-) を演算子として、そして `two` を別の QName として認識します。
- `one-two` は、構文エラーになりません。ただし、QName においてハイフン (-) は有効な文字であるため、式は単一の QName として構文解析されます。
- 以下の式は、すべて構文エラーになります。
  - `5 div2`
  - `5div2`

  これらの式では、パーサーが各トークンを別個に認識できるようにするために、空白が必要です。`5div 2` は、構文エラーにならないことに注意してください。

DB2 XPath でのコメント

コメントは、XPath 式内で不要な空白文字が許可されるいずれの場所でも使用できます。コメントは、式の処理には影響しません。

コメントは、記号 `(:` および `:)` で区切られたストリングです。以下は、XPath のコメントの例です。

```
(: This is a comment. It makes code easier to understand. :)
```

DB2 XPath でのコメントの使用には、以下の汎用規則が適用されます。

- コメントは、不要な空白文字が許可されるいずれの場所でも使用できます。*不要な空白文字* とは、XPath 式の構文の一部でない空白文字のことです。
- コメント同士はネスト可能ですが、ネストされたコメントは、左右を区切り文字 `(:` および `:)` で囲む必要があります。

以下は、正しいコメントの例、およびエラーになるコメントの例です。

- `(: is this a comment? ::)` は、正しいコメントです。
- `(: is this a comment? ::) or an error? :)` は、エラーになります。原因は、記号 `(:` および `:)` のネストが対になっていないためです。
- `(: commenting out a (: comment :) may be confusing, but often helpful :)` は、正しいコメントです。コメントのネストが正しく対になっていれば、コメントはネストできます。
- `"this is just a string :)"` は正しい式です。

- `(: "this is just a string :)" :)` は、構文エラーになります。同様に、`"this is another string (:"` は正しい式ですが、 `(: "this is another string (:" :)` は構文エラーになります。リテラル・コンテンツは、コメントのネストが対でなくても構いません。

文字セット

DB2 XPath モデルは、UTF-8 エンコード方式を使用します。DB2 では、文字、グラフィック、および XML 入力パラメーターがすべて UTF-8 に変換されます。また出力データは、UTF-8 から、結果列に指定された CCSID にキャストされます。

デフォルトの照合

DB2 XPath は、XPath 式のデフォルトの照合を、XMLTABLE を含む SQL ステートメントに使用される照合シーケンスから決定します。DB2 XPath は、ICU (UNICODE 国際コンポーネント) を使用して、バイナリー (*HEX) 照合シーケンスと Unicode 照合シーケンスをサポートします。

DB2 XPath での XML 名前空間と修飾名

DB2 XPath は、XML 名前空間を使用して、ネーミングの衝突を回避します。*XML 名前空間* は、名前空間 URI によって識別される名前のコレクションです。 名前空間は、XPath でエレメント、属性、データ・タイプ、および関数に使用される名前を修飾する手段となります。

XPath での名前は、QName (修飾名) と呼ばれ、W3C 勧告「*Namespaces in XML*」で定義されている構文に準拠します。*QName* は、オプションの名前空間接頭部およびローカル名で構成されます。 ネーム・スペース接頭部を指定する場合、これは URI にバインドされ、 URI の短縮形を提供します。照会の処理時に、DB2 XPath は名前空間接頭部にバインドされている URI を解決して、QName を展開します。拡張 QName には、名前空間 URI およびローカル名が含まれます。 2 つの QName は、その名前空間 URI とローカル名が同じである場合に等しくなります。 これは、2 つの QName は、異なる接頭部を持っていても、それらが同じ名前空間 URI にバインドされていれば一致する可能性があるということを意味します。

XPath で QName を使用すると、同じローカル名を持つが、異なる DTD または XML スキーマに関連している可能性のあるエレメント・タイプまたは属性名を式で参照できます。以下の XML データにおいて、`pfx1` は、ある URI にバインドされる接頭部です。 `pfx2` は、別の URI にバインドされる接頭部です。 `c` は、3 つのエレメントすべてのローカル名です。

```
<a xmlns:pfx1="uri1" xmlns:pfx2="uri2">
<b>
 <pfx1:c>C</pfx1:c>
 <pfx2:c>B</pfx2:c>
 <c>A</c>
</b>
</a>
```

この例のエレメントは、同じローカル名 `c` を共用しますが、これらのエレメントが異なるネーム・スペースに存在するため、ネーミングの競合は発生しません。式の処理時に、名前 `pfx1:c` は、`pfx1 (uri1)` にバインドされた URI を含む名前とローカル名 `c` に展開されます。同様に、名前 `pfx2:c` は、 `pfx2 (uri2)` にバインドされた URI を含む名前とローカル名 `c` に展開されます。空の接頭部を持つエレメント `c` は、接頭部が指定されていないため、デフォルトのエレメント・ネーム・スペースにバインドされます。 URI にバインドされていない接頭部が名前に使用されていると、エラーが発生します。

ネーム・スペースの接頭部は NCName (コロンが付いていない名前) でなければなりません。XML NCName は、 NCName にコロンを含めることができないことを除いて、XML Name と似ています。

一部のネーム・スペースは事前に宣言されています。それ以外のネーム・スペースは XPath 式プロローグ内の宣言によって追加できます。DB2 XPath には以下の事前宣言されたネーム・スペース接頭部が組み込まれています。

接頭部	URI	説明
xs	http://www.w3.org/2001/XMLSchema	XML スキーマ名前空間
xsi	http://www.w3.org/2001/XMLSchema-instance	XML スキーマ・インスタンス・ネーム・スペース
fn	http://www.w3.org/2005/xpath-functions	デフォルトの関数名前空間
xdt	http://www.w3.org/2005/xpath-datatypes	XPath タイプのネーム・スペース
db2-fn	http://www.ibm.com/xmlns/prod/db2/functions	DB2 関数名前空間

事前宣言されたネーム・スペースに加えて、以下の方法でネーム・スペースを提供することができます。

- 静的なコンテキストでは、以下のネーム・スペース情報を使用できます。
  - *範囲内ネーム・スペース* とは、接頭部と URI ペアの集合体です。範囲内ネーム・スペースは、XPath 式の QName で使用されている接頭部を解決するために使用されます。範囲内ネーム・スペースは、以下のソースから派生します。
    - XPath 式でのネーム・スペース宣言
    - XMLELEMENT、XMLFOREST、または XMLTABLE DB2 組み込み関数内の XMLNAMESPACES DB2 組み込み関数
  - *デフォルトのエレメントまたはタイプ・ネーム・スペース* は、エレメント名またはタイプ名が予測される場合に現れる、接頭部が付いていない QName に使用されるネーム・スペースです。初期のデフォルトのエレメントまたはタイプのネーム・スペースは、XPath 式のプロローグ内の `declare default element namespace` 文節で提供されるデフォルトのネーム・スペースです。
  - *デフォルトの関数ネーム・スペース* は、組み込み関数 http://www.w3.org/2003/11/xpath-functions に関連付けられたネーム・スペースです。XPath にはユーザー定義関数はありません。

XPath タイプ・システム

DB2 XPath は、強く型付けされた言語であり、さまざまな式、演算子、および関数のオペランドが、期待されるタイプに準拠します。

DB2 XPath のタイプ・システムには、XML スキーマの組み込みタイプのサブセット、および XPath の事前定義タイプが含まれています。

XML スキーマの組み込みタイプは、ネーム・スペース `http://www.w3.org/2001/XMLSchema` にあり、事前宣言されたネーム・スペース接頭部 `xs` を持ちます。組み込みスキーマ・タイプの例としては、`xs:integer` および `xs:string` などがあります。

タイプ・システムの概要

DB2 XPath のタイプ・システムには、単純アトミック・タイプと複合タイプがあります。*単純アトミック・タイプ* は、エレメントや属性が含まれていない基本的または派生したアトミック・タイプです。*複合タイプ* には、混合された内容またはエレメントのみの内容が含まれます。

組み込みデータ・タイプのコンストラクター関数

XML スキーマ定義に定義されているすべての組み込みアトミック・タイプには、関連したコンストラクター関数があります。

構文

prefix
データ・タイプのネーム・スペースに結合される接頭部。これは、デフォルト関数ネーム・スペースに結合される接頭部ではありません。

タイプ
ターゲット・データ・タイプの非修飾名。

value
構成される値。この値が空のシーケンスである場合は、空のシーケンスが戻されます。

value がターゲット・データ・タイプに対して不適切な場合、コンストラクター関数はエラーを戻します。

戻り値

value が空のシーケンスでない場合、戻り値は、*prefix:type* のインスタンスです。

value が空のシーケンスの場合、コンストラクター関数は空のシーケンスを戻します。

例:

コンストラクター関数 `xs:integer(100)` またはコンストラクター関数 `xs:integer("100")` は、`xs:integer` 値 100 を戻します。引数が型付きの値 100 のノードであるコンストラクター関数も、型付きの値 100 を戻します。

汎用データ・タイプ

汎用データ・タイプは、タイプが明確に指定されていないデータをサポートします。

xs:anyType:

データ・タイプ `xs:anyType` は、XML スキーマ定義言語に定義されているすべてのデータ・タイプの基本タイプです。

xs:anySimpleType:

データ・タイプ `xs:anySimpleType` は、XML スキーマ定義言語に定義されているすべてのプリミティブ・タイプの基本タイプです。

`xs:anySimpleType` は、任意の単純タイプを使用できることを示すために必要なタイプを定義する際に (例えば、関数シグニチャーなどで) 使用されます。 `xs:anySimpleType` の基本タイプは `xs:anyType` です。

データ・タイプ `xs:anySimpleType` からのキャストも、`xs:anySimpleType` へのキャストもサポートされません。

字句形式

xs:anySimpleType では、任意の字句形成を使用できます。

xs:anyAtomicType:

データ・タイプ xs:anyAtomicType は、XML スキーマ定義言語に定義されているすべてのプリミティブなアトミック・タイプの基本タイプです。

字句形式

データ・タイプ xs:anyAtomicType は、任意のプリミティブなアトミック・タイプまたは xs:untypedAtomic を使用できることを示すために必要なタイプを定義する際に (例えば、関数シグニチャーなどで) 使用されます。 xs:anyAtomicType の基本タイプは xs:anySimpleType です。

xs:anyAtomicType では、任意の字句形成を使用できます。

型のないデータのデータ・タイプ

xs:untyped および xs:untypedAtomic データ・タイプは、型のないデータをサポートします。

xs:untyped:

データ・タイプ xs:untyped は、XML スキーマによって検証されていないタイプを示すための特殊なタイプ・アノテーションとして機能します。データ・タイプ xs:untyped は、(例えば、関数シグニチャーなどで) 型のない値のみを使用できることを示すために必要なタイプを定義する際に使用できます。 xs:untyped の基本タイプは xs:anyType です。

要素ノードに xs:untyped のアノテーションが付けられる場合、そのすべての子孫要素ノードにも xs:untyped のアノテーションが付けられます。 DB2 for i は、スキーマ妥当性検査からのタイプ・アノテーションを保持しません。 XML 文書内のすべての要素のタイプは xs:untyped です。

xs:untypedAtomic:

データ・タイプ xs:untypedAtomic は、XML スキーマによって妥当性検査されていないアトミック値を示すための特殊なタイプ・アノテーションとして機能します。

不明なタイプの属性は、データ・モデルにおいて、 xs:untypedAtomic タイプの属性ノードによって表されます。データ・タイプ xs:untypedAtomic は、(例えば、関数シグニチャーなどで) 型のないアトミック値のみを使用できることを示すために必要なタイプを定義する際に使用できます。 xs:untypedAtomic の基本タイプは xs:anyAtomicType です。このタイプにはコンストラクターがありません。DB2 for i は、XML 文書が XML スキーマにより検証済みである場合にも、XML 文書内にタイプ・アノテーションを保持しません。すべての属性のタイプは、データ・モデル内のタイプ xs:untypedAtomic となります。

字句形式

xs:untypedAtomic では、任意の字句形成を使用できます。

xs:string

データ・タイプ xs:string は、XML の文字ストリングを表します。xs:string は単純タイプなので、子を持つことはできません。

字句形式

xs:string の字句形式は、 XML で使用できる文字の範囲内である任意の文字が含まれる文字のシーケンスです。

コンストラクター

以下の構文を使用して、 xs:string のインスタンスを構成します。

value
構成される値。この値が空のシーケンスである場合は、空のシーケンスが戻されます。

value がターゲット・データ・タイプに対して不適切な場合、コンストラクター関数はエラーを戻します。

数値データ・タイプ

xs:decimal、xs:double、および xs:integer データ・タイプは、数値データをサポートします。

xs:decimal:

データ・タイプ xs:decimal は、10 進数表示で表せる実数のサブセットを表します。

字句形式

xs:decimal の字句形式は、小数部標識としてのピリオドによって区切られた 10 進数字 (0 から 9) の有限長シーケンスです。状況に応じて符号を前に置くことが許可されています。符号を省略した場合は、正符号 (+) が想定されます。 前後にゼロを置くことは任意です。小数部分がゼロの場合、ピリオドおよびその後に置くゼロは省略できます。以下の数値は、すべて 10 進数の有効な例です。

- -1.23
- 12678967.543233
- +100000.00
- 210.

コンストラクター

以下の構文を使用して、 xs:decimal のインスタンスを構成します。

value
構成される値。この値が空のシーケンスである場合は、空のシーケンスが戻されます。

value がターゲット・データ・タイプに対して不適切な場合、コンストラクター関数はエラーを戻します。

xs:double:

データ・タイプ xs:double は、DB2 XPath で、IEEE 64 ビット 10 進浮動小数点によってサポートされます。

字句形式

xs:double の字句形式は、小数部ですが、状況に応じて後ろに文字 E または e、その後に指数が続きます。指数は整数でなければなりません。 小数部は、10 進数でなければなりません。指数および小数部の表記は、 xs:integer および xs:decimal の字句規則に従う必要があります。続く E または e および指数が省略されると、指数値は 0 と想定されます。

特殊値 (正の無限大、負の無限大、および非数値) の字句表記は、それぞれ INF、-INF、および NaN です。ゼロの字句表記には、正符号または負符号を使用できます。以下のリテラルは、すべて double の有効な例です。

- -1E4
- 1267.43233E12
- 12.78e-2
- 12
- -0
- 0
- INF

コンストラクター

以下の構文を使用して、 xs:double のインスタンスを構成します。

value
構成される値。この値が空のシーケンスである場合は、空のシーケンスが戻されます。

value がターゲット・データ・タイプに対して不適切な場合、コンストラクター関数はエラーを戻します。

xs:integer:

データ・タイプ xs:integer は、後書き小数点が含まれていない 10 進数を表します。xs:integer の基本タイプは xs:decimal です。

字句形式

xs:integer の字句形式は、前に符号が付いた (符号はオプション) 10 進数字 (0 から 9) の有限長シーケンスです。符号を省略した場合は、正符号 (+) が想定されます。以下の数値は、すべて整数の有効な例です。

- -1
- 0
- 12678967543233
- +100000

コンストラクター

以下の構文を使用して、 xs:integer のインスタンスを構成します。

value
構成される値。この値が空のシーケンスである場合は、空のシーケンスが戻されます。

value がターゲット・データ・タイプに対して不適切な場合、コンストラクター関数はエラーを戻します。

数値タイプの範囲制限:

DB2 XPath では、数値データ・タイプに範囲制限があります。

以下の表に、各 XPath 数値データ・タイプの範囲制限とその同等の SQL タイプをリストしています。

表 49. 数値タイプの範囲制限

XML タイプ	DB2 XML の範囲	SQL タイプ・マッピング
xs:double	精度は 34 桁で指数範囲は 10**-6143 から 10**+6144	DECFLOAT
xs:decimal	精度は 34 桁までで範囲は 1-10**34 から 10**34 -1	DECIMAL 34 桁を超える精度では切り捨てが起こる場合があるということに注意してください。
xs:integer	-9223372036854775808 から 9223372036854775807	BIGINT

xs:boolean

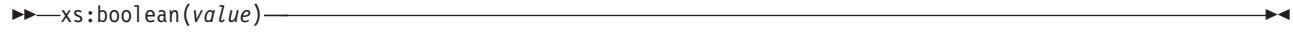
データ・タイプ xs:boolean は、数学上の 2 値論理 (true または false) の概念をサポートします。

字句形式

データ・タイプ xs:boolean の字句形式は、 true、false、1、および 0 のいずれかのリテラル値です。

コンストラクター

以下の構文を使用して、 xs:boolean のインスタンスを構成します。



value
構成される値。この値が空のシーケンスである場合は、空のシーケンスが戻されます。

value がターゲット・データ・タイプに対して不適切な場合、コンストラクター関数はエラーを戻します。

日時データ・タイプ

xs:date、xs:time、および xs:dateTime データ・タイプは、日時データをサポートします。

xs:date:

日付タイプ xs:date は、指定された日の最初の瞬間から始まって、ちょうど 1 日分の間隔を表します。

字句形式

xs:date の字句形式は、*yyyy-mm-ddzzzzzz* 形式の有限長シーケンスです。以下の省略形はこの形式を説明しています。

yyyy
年を表す 4 桁の数表示。

値は、負の符号 (-) で始めることも正の符号 (+) で始めることもできません。

0001 は、西暦紀元 (AD と呼ばれています) 1 年の字句表記です。

値を 0000 にすることはできません。

- 日付部分間の区切り記号。

mm 月を表す 2 桁の数表示。

dd 日を表す 2 桁の数表示。

zzzzzz
オプション。 存在する場合、時間帯を示します。

時間帯標識

時間帯標識の字句形式は、以下のいずれかの形式を含むストリングです。

- hh:mm の前にある正符号 (+) または負符号 (-) は、次の省略形が使用されます。
  hh 時間を表す 2 桁の数表示 (必要に応じて先行ゼロが付きます)。値は、-14 以上、+14 以下である必要があります。
  mm 分を表す 2 桁の数表示。時間プロパティーが 24 の場合、分のプロパティーの値はゼロでなければなりません。
  + 指定された時刻インスタンスが、*hh* 時 *mm* 分、UTC 時刻より前の時間帯にあることを示します。
  - 指定された時刻インスタンスが *hh* 時 *mm* 分、UTC 時刻より後の時間帯にあることを示します。
- リテラル Z は、UTC の時刻を示します (Z は、ズールー時を示しますが、これは UTC と同じです)。時間帯に Z を指定することは、+00:00 または -00:00 を指定することと同じです。

例

以下の形式は、米国の東部標準時 (EST) の 2009 年 10 月 10 日を示します。

```
2009-10-10-05:00
```

この日付は 2009-10-10T05:00:00Z という UTC 日付を表します。

xs:time:

データ・タイプ xs:time は、毎日定期的に繰り返される瞬間を表します。

字句形式

データ・タイプ xs:time の字句形式は、 *hh*:*mm*:*ss*.*sssssssssssszzzzzz* です。

データ・タイプ xs:time の字句形式は、 *hh*:*mm*:*ss*.*ssssssszzzzzz* です。

以下の省略形はこの形式を説明しています。

hh 時間を表す 2 桁の数表示 (必要に応じて先行ゼロが付きます)。

: 時刻部の各部分間の区切り記号。

mm 分を表す 2 桁の数表示。

ss 秒を表す 2 桁の数表示。

.ssssssssssssssss
オプション。 存在する場合、小数点以下の秒数を表す 1 から 12 桁の数表示。

zzzzzz
オプション。 存在する場合、時間帯を示します。

例

オプションの時間帯標識が含まれている以下の形式は、東部標準時午後 1 時 20 分 (協定世界時 (UTC) より 5 時間遅れ) を表します。

```
13:20:00-05:00
```

xs:dateTime:

データ・タイプ xs:dateTime は、時間における瞬間を表します。

xs:dateTime データ・タイプには以下のプロパティーがあります。
- year
- month
- day
- hour
- minute
- second
- time zone (オプション)

year、month、day、hour、および minute のプロパティーは、整数値で表されます。seconds プロパティーは、10 進値で表されます。 time zone プロパティーは、時間帯標識で表現されます。

字句形式

xs:dateTime の字句形式は、*yyyy*-*mm*-*dd*T*hh*:*mm*:*ss*.*sssssssssssszzzzzz* 形式の有限長シーケンスです。以下の省略形はこの形式を説明しています。

yyyy
年を表す 4 桁の数表示。

値は、負の符号 (-) で始めることも正の符号 (+) で始めることもできません。

0001 は、西暦紀元 (AD と呼ばれています) 1 年の字句表記です。

値を 0000 にすることはできません。

- 日付部の部分間の区切り記号です。

mm 月を表す 2 桁の数表示。

dd 日を表す 2 桁の数表示。

T 後に時刻が続くことを示す区切り記号。

hh 時間を表す 2 桁の数表示 (必要に応じて先行ゼロが付きます)。値は、-14 以上、+14 以下である必要があります。

: 時刻部の各部分間の区切り記号。

mm 分を表す 2 桁の数表示。

ss 秒を表す 2 桁の数表示。

.sssssssssss
オプション。 存在する場合、小数点以下の秒数を表す 1 から 12 桁の数表示。

zzzzzz
オプション。 存在する場合、時間帯を示します。時間帯を指定しない場合、dateTime に時間帯はありません。ただし、UTC (協定世界時、グリニッジ標準時とも呼ばれる) の暗黙的時間帯が、比較および算術演算に使用されます。

数値として表される日時値の各部分は、日時値の 1 レベル上の部分によって設定された間隔の最大値に制限されます。例えば、日付値を 32 にすることはできません。また、月が 02 で年が 2002 (2002 年 2 月) の場合は、日付値を 29 にすることはできません。

時間帯標識

時間帯標識の字句形式は、以下のいずれかの形式を含むストリングです。

- hh:mm の前にある正符号 (+) または負符号 (-) は、次の省略形が使用されます。
  hh 時間を表す 2 桁の数表示 (必要に応じて先行ゼロが付きます)。値は、-14 以上、+14 以下である必要があります。
  mm 分を表す 2 桁の数表示。時間プロパティーが 24 の場合、分のプロパティーの値はゼロでなければなりません。
  + 指定された時刻インスタンスが、*hh* 時 *mm* 分、UTC 時刻より前の時間帯にあることを示します。
  - 指定された時刻インスタンスが *hh* 時 *mm* 分、UTC 時刻より後の時間帯にあることを示します。
- リテラル Z は、UTC の時刻を示します (Z は、ズールー時を示しますが、これは UTC と同じです)。時間帯に Z を指定することは、+00:00 または -00:00 を指定することと同じです。

例

以下の形式は、米国の東部標準時 (EST) の 2009 年 10 月 10 日正午を示します。

```
2009-10-10T12:00:00-05:00
```

この時刻は、UTC では 2009-10-10T17:00:00Z と表記されます。

xs:duration:

データ・タイプ xs:duration は、グレゴリオの年、月、日、時間、分、および秒のコンポーネントによって表現される期間を表します。 xs:duration は、データ・タイプ xs:anyAtomicType から派生しています。

このデータ・タイプによって表現可能な範囲は
-P83333333333333Y3M11574074074DT1H46M39.999999999999S から
P83333333333333Y3M11574074074DT1H46M39.999999999999S まで (つまり -999999999999999 カ月と 999999999999999.999999999999 秒から 999999999999999 カ月と 999999999999999.999999999999 秒まで) です。

xs:duration の字句形式は ISO 8601 拡張形式で PnYnMnDTnHnMnSです。以下の省略形は、拡張フォーマットを説明しています。

P 期間指定子。

*n***Y** *n* は年数を示す符号なしの整数です。

*n***M** *n* は月数を示す符号なしの整数です。

*n***D** *n* は日数を示す符号なしの整数です。

T 日時の分離文字。

*n***H** *n* は時間数を示す符号なしの整数です。

*n***M** *n* は分数を示す符号なしの整数です。

*n***S** *n* は秒数を示す符号なしの 10 進数です。 小数点が表示される場合、小数秒を示す 1 から 12 桁の数字が小数点の後に必要です。

例えば、次の形式は、1 年と 2 カ月 3 日 10 時間 30 分の期間を示します。

```
P1Y2M3DT10H30M
```

以下の形式は、マイナス 120 日間を示します。

```
-P120D
```

先行するオプションの負符号 (-) は、負の期間を示します。 符号を省略すると、正の期間と認識されます。

この形式の精度を低減し、表記を切り捨てることが可能ですが、以下の要件に準拠する必要があります。

- 式の年数、月数、日数、時間数、分数、または秒数がゼロである場合、数値とそれに対応する指定子は省略できます。 ただし、最低 1 個の数値および指定子が存在する必要があります。
- 2 番目の部分には、小数部を使用できます。
- 時間項目がなければ指定子 T を省略する必要があり、T を省略する必要があるのはその場合に限ります。
- 指定子 P は常に必要です。

例えば、以下の形式は使用可能です。

```
P1347Y
P1347M
P1Y2MT2H
P0Y1347M
P0Y1347M0D
```

P1Y2MT の形式は、時間項目がないため、使用できません。 形式 P-1347M は使用できませんが、形式 -P1347M は使用できます。

DB2 は、xs:duration 値を正規化された形式で保管します。正規化された形式では、構成要素の秒と分は 60 より小さく、時間は 24 より小さく、月は 12 より小さくなります。 DB2 では、60 秒の倍数ごとを 1 分

に変換し、60 分の倍数ごとを 1 時間に変換し、24 時間の倍数ごとを 1 日に変換し、12 カ月の倍数ごとを 1 年に変換します。例えば、以下の XPath 式は、2 カ月と 63 日と 55 時間 91 分の期間を指定するコンストラクター関数を呼び出します。

```
xs:duration("P2M63DT55H91M")
```

DB2 は 55 時間を 2 日と 7 時間に変換し、91 分を 1 時間と 31 分に変換します。この式は、正規化された duration 値 P2M65DT8H31M を戻します。

xs:dayTimeDuration:

データ・タイプ xs:dayTimeDuration は、日、時間、分、および秒のコンポーネントによって表現される期間を表します。 xs:dayTimeDuration は、データ・タイプ xs:duration から派生しています。

このデータ・タイプによって表現可能な範囲は -P11574074073DT23H163M219.999999999999S から P11574074073DT23H163M219.999999999999S まで (つまり -999999999999999.999999999999 秒から 999999999999999.999999999999 秒まで) です。

xs:dayTimeDuration の字句形式は P*n*DT*n*H*n*M*n*Sであり、これは ISO 8601 形式の還元型です。以下の省略形はこの形式を説明しています。

P 期間指定子。

***n*D** *n* は日数を示す符号なしの整数です。

T 日時の分離文字。

***n*H** *n* は時間数を示す符号なしの整数です。

***n*M** *n* は分数を示す符号なしの整数です。

***n*S** *n* は秒数を示す符号なしの 10 進数です。 小数点が表示される場合、小数秒を示す 1 から 12 桁の数字が小数点の後に必要です。

例えば、以下の形式は、3 日間、10 時間と 30 分間を示します。

```
P3DT10H30M
```

以下の形式は、マイナス 120 日間を示します。

```
-P120D
```

先行するオプションの負符号 (-) は、負の期間を示します。 符号を省略すると、正の期間と認識されます。

この形式の精度を低減し、表記を切り捨てることが可能ですが、以下の要件に準拠する必要があります。

- 任意の式の日数、時間数、分数、秒数がゼロである場合、その数値と対応する指定子は省略できます。ただし、最低 1 個の数値および指定子が存在する必要があります。
- 2 番目の部分には、小数部を使用できます。
- 時間項目がなければ指定子 T を省略する必要があり、T を省略する必要があるのはその場合に限ります。 指定子 P は常に必要です。

例えば、以下の形式は使用可能です。

```
P13D
PT47H
P3DT2H
-PT35.89S
P4DT251M
```

P-134D の形式は使用できませんが、-P1347D の形式は使用できます。

DB2 は xs:dayTimeDuration 値を正規化された形式で保管します。正規化された形式では、構成要素の秒と分は 60 より小さく、時間は 24 より小さくなります。 DB2 では、60 秒の倍数ごとを 1 分に変換し、60 分の倍数ごとを 1 時間に変換し、24 時間の倍数ごとを 1 日に変換します。例えば、以下の XPath 式は、63 日 と 55 時間 81 秒の dayTimeDuration を指定するコンストラクター関数を呼び出します。

```
xs:dayTimeDuration("P63DT55H81S")
```

DB2 は 55 時間を 2 日と 7 時間に変換し、81 秒を 1 分と 21 秒に変換します。この式は、正規化された dayTimeDuration 値 P65DT7H1M21S を戻します。

xs:yearMonthDuration:

データ・タイプ xs:yearMonthDuration は、グレゴリオの年および月のコンポーネントによって表現される期間を表します。 xs:yearMonthDuration は、データ・タイプ xs:duration から派生しています。

このデータ・タイプで表すことができる範囲は、 -P83333333333333Y3M から P83333333333333Y3M です (または -999999999999999 カ月から 999999999999999 カ月)。

xs:yearMonthDuration の字句形式は、P*n*Y*n*M で、これは ISO 8601 形式の還元型です。以下の省略形はこの形式を説明しています。

P 期間指定子。

***n*Y** *n* は年数を示す符号なしの整数です。

***n*M** *n* は月数を示す符号なしの整数です。

先行するオプションの負符号 (-) は、負の期間を示します。 符号を省略すると、正の期間と認識されます。

例えば、以下の形式は、1 年 2 カ月の期間を示します。

```
P1Y2M
```

以下の形式は、負の 13 カ月の期間を示します。

```
-P13M
```

この形式の精度を低減し、表記を切り捨てることが可能ですが、以下の要件に準拠する必要があります。

- 指定子 P は常に必要です。
- 式の中の年または月の数が 0 の場合、その数と対応する指定子は省略できます。 ただし、少なくとも 1 つの数およびその指定子 (Y または M) が必要です。

例えば、以下の形式は使用可能です。

```
P1347Y
P1347M
```

形式 P-1347M は使用できませんが、形式 -P1347M は使用できます。 形式 P24YM は、許可されません。これは、M に先行する数字が 1 つ必要であるためです。PY43M は、許可されません。これは、Y に先行する数字が 1 つ以上必要であるためです。

DB2 は、正規形で xs:yearMonthDuration 値を保管します。正規化された形式では、月コンポーネントは 12 未満です。 DB2 は、12 カ月の倍数ごとを 1 年に変換します。例えば、以下のXPath 式は、20 年と 30 カ月の yearMonthDuration を指定するコンストラクター関数を呼び出します。

```
xs:yearMonthDuration("P20Y30M")
```

DB2 は 30 カ月を 2 年と 6 カ月に変換します。この式は、正規化された yearMonthDuration 値 P22Y6M を戻します。

XML スキーマ・データ・タイプ間のキャスト

データ・タイプ・コンストラクター関数を使用して、値を特定のデータ・タイプにキャストすることができます。キャストする値とキャストするタイプを指定します。

以下の表に、キャストの互換タイプをリストします。リストされている入力タイプの値のみを各ターゲット・タイプにキャストすることができます。

表 50. キャストの互換タイプ

ターゲット・タイプ	ソース・タイプ	注釈
xs:string	すべてのタイプ	• ソース・タイプが xs:boolean の場合、結果は「true」または「false」です。 • ソース・タイプが xs:integer の場合、結果は、XML スキーマ仕様で定義されている、値の正規字句表記です。 • ソース・タイプが xs:decimal の場合: – 値の小数点以降に有効数字が無い場合、小数点および小数点以降のゼロは削除されます。 xs:integer からのキャストの規則が適用されます。 – それ以外の場合、結果は、XML スキーマ仕様で定義されている、値の正規字句表記です。 • ソース・タイプが xs:double の場合: – .000001<=*value*<=1000000 の場合、値は xs:decimal に変換され、 xs:decimal からのキャストの規則が適用されます。 – *value*=+0 または *value*=-0 の場合、結果は '0' です。 – それ以外の場合、結果は、XML スキーマ仕様で定義されている、値の正規字句表記です。 • ソース・タイプが xs:duration、xs:yearMonthDuration、または xs:dayTimeDuration の場合、結果は、値の正規字句表記です。 • ソース・タイプが xs:date、xs:dateTime、または xs:time の場合、結果は、時間帯の調整を含まない値の正規字句表記です。値に時間帯がない場合、結果にも時間帯はありません。時間帯が +00:00 または -00:00 の場合、結果は UTC 時間帯「Z」を含みます。

表 50. キャストの互換タイプ (続き)

ターゲット・タイプ	ソース・タイプ	注釈
xs:boolean	xs:untypedAtomic、 xs:string、 xs:boolean、 xs:double、 xs:decimal、 xs:integer	• ソース・タイプが numeric の場合、値 0 または NaN は、タイプ xs:boolean にキャストされ、値は `false` になります。他のすべての数値はタイプ xs:boolean にキャストされ、値は `true` になります。 • ソース・タイプが xs:string または xs:untypedAtomic の場合、値 "true" および値 "1" は xs:boolean 値 `true` にキャストされます。値 "false" および値 "0" は、xs:boolean 値 `false` にキャストされます。それ以外の値はすべて無効で、結果はエラーになります。
xs:decimal	数値タイプ、xs:untypedAtomic、 xs:string、 xs:boolean	数値タイプの値は、タイプ xs:decimal に有効な範囲の値で数値的に最もソースに近い値に変換されます。 2 つの値が同距離の近似値である場合は、ゼロに近い方の値になります。ソース値は、+INF、-INF、NaN、またはタイプ xs:decimal の範囲外であってはなりません。タイプ xs:boolean の値の場合、`true` は 1.0 に変換され、`false` は 0.0 に変換されます。
xs:double	数値タイプ、xs:untypedAtomic、 xs:string、 xs:boolean	ソースがタイプ xs:decimal または xs:integer の場合、キャストは xs:double(*SV* cast as xs:string) として行われます。ここで、*SV* はソース値です。ソースがタイプ xs:boolean の場合、`true` は値 1.0E0 に、`false` は値 0.0E0 にキャストされます。
xs:integer	数値タイプ、xs:untypedAtomic、 xs:string、 xs:boolean	ソース・タイプが整数以外の数値タイプである場合、結果はソース値から小数部分が廃棄された値になります。ソースは、タイプ xs:integer の範囲外であってはなりません。タイプ xs:boolean の値の場合、`true` は 1 に変換され、`false` は 0 に変換されます。
xs:date	xs:dateTime、xs:untypedAtomic、 xs:string	ソース値の時刻部は変換に使用されません。
xs:time	xs:dateTime、xs:untypedAtomic、 xs:string	ソース値の日付部は変換に使用されません。
xs:dateTime	xs:date、xs:untypedAtomic、 xs:string	ソース・タイプが xs:date の場合、ターゲット値の時刻部分は、その日の最初の瞬間です。時間帯による値の調整はありません。

表 50. キャストの互換タイプ (続き)

ターゲット・タイプ	ソース・タイプ	注釈
xs:duration	xs:dayTimeDuration、xs:yearMonthDuration、xs:untypedAtomic、xs:string	• ソース・タイプが xs:dayTimeDuration の場合、ターゲット値は、ソース値と同じ日、時間、分、および秒のコンポーネントを持ちます。ターゲット値の年コンポーネントおよび月コンポーネントは 0 です。 • ソース・タイプが xs:yearMonthDuration の場合、ターゲット値は、ソース値と同じ年および月のコンポーネントを持ちます。日、時間、分、および秒のコンポーネントは 0 です。
xs:dayTimeDuration	xs:duration、xs:untypedAtomic、xs:string	xs:duration から xs:dayTimeDuration にキャストすると情報が失われます。情報が失われないようにするには、xs:duration 値を xs:yearMonthDuration 値と xs:dayTimeDuration 値にキャストし、両方の値を処理してください。
xs:yearMonthDuration	xs:duration、xs:untypedAtomic、xs:string	xs:duration から xs:yearMonthDuration にキャストすると情報が失われます。情報が失われないようにするには、xs:duration 値を xs:yearMonthDuration 値と xs:dayTimeDuration 値にキャストし、両方の値を処理してください。

例

以下の XPath 式は、複数の項目が含まれている購入注文を戻します。xs:integer コンストラクター関数は、数量エレメントの値を整数にキャストします。次に、その整数が整数 1 と比較されます。

```
declare namespace ipo="http://www.example.com/IPO";
/ipo:purchaseOrder[items/item/quantity/xs:integer(.) > 1]
```

xs:untypedAtomic が整数と比較されるときには、DB2 は、両方のオペランドをタイプ xs:double に変換して数値比較を行います。キャストにより、値が確実にタイプ xs:integer の値として比較されます。

XPath のプロローグと式

DB2 XPath では、XPath 式はオプションのプロローグとそれに続く式から構成されます。*プロローグ* には、式の処理環境を定義する一連の宣言が含まれます。*式* は、XPath 式の結果を定義する式から構成されます。

構文

以下のダイアグラムは、XPath 式の一般形式を示しています。

XPath 式

式:

プロローグ:

例

以下の例は、DB2 XPath における典型的な式の構造を示しています。この例では、プロローグには、接頭部 ipo を URI にバインドする名前空間宣言が含まれています。式本体には、XML 文書が XMLPO 列に保管されている ipo:purchaseOrder 要素ごとに 1 行を戻す式が含まれています。述部は、shipTo ノードの名前属性が「Jane」で、billTo ノードの名前属性が「Jason」であることを指定するために使用されます。

```
SELECT X.* FROM T1, XMLTABLE ('declare namespace ipo="http://www.example.com/IPO";
     /ipo:purchaseOrder[shipTo/@name = "Jane" and billTo/@name = "Jason"]'
     PASSING T1.XMLPO) X;
```

図 1. DB2 XPath における典型的な式の構造

プロローグ

プロローグ は、XPath 式の処理環境を定義する宣言から構成されます。プロローグ内の宣言の後にはセミコロン (;) が付きます。プロローグは、XPath 式のオプション部分です。

プロローグには、ゼロ個以上の名前空間宣言およびゼロ個または 1 つのデフォルト名前空間宣言を含めることができます。

ネーム・スペース宣言:

ネーム・スペース宣言は、XPath 式プロローグ内にあり、ネーム・スペース接頭部を宣言して、その接頭部をネーム・スペース URI と関連付けるオプションの宣言です。

宣言は、接頭部と URI のペアを式の静的に既知のネーム・スペースのセットに追加します。*静的に既知の*ネーム・スペース には、式の静的処理中に認識されるすべてのネーム・スペースが含まれます。ネーム・スペース宣言は、宣言されている XPath 式全体で有効です。照会プロローグにおいて同じネーム・スペース接頭部を複数回宣言すると、結果はエラーになります。

制約事項: 接頭部 xmlns および xml は予約済みであり、ネーム・スペース宣言で接頭部として指定することはできません。

構文

namespace-declaration

prefix
URI にバインドするネーム・スペース接頭部を指定します。ネーム・スペース接頭部は、エレメント、属性、データ・タイプ、または関数のネーム・スペースを識別するために、修飾名 (QName) で使用されます。

stringLiteral
接頭部をバインドする URI を表すストリング・リテラル指定します。ストリング・リテラル値は、有効な URI でなければならず、ゼロ長ストリングにすることはできません。

これらの接頭部のネーム・スペース宣言を指定することにより、事前宣言されたネーム・スペース接頭部を指定変更できます。ただし、接頭部 `xml` に関連付けられた URI は指定変更できません。

ストリング・リテラルは、http://www.w3.org/XML/1998/namespace または http://www.w3.org/2000/xmlns/ であってはなりません。

例

以下のネーム・スペース宣言は、ネーム・スペース接頭部 `ns1` を宣言し、これをネーム・スペース URI `http://posample.org` に関連付けます。

```
declare namespace ns1 = "http://posample.org";
/ns1:purchaseOrder[shipTo/name = "Jane" and billTo/name = "Jason"]
```

この例の式が実行されると、ネーム・スペース接頭部 ns1 はネーム・スペース URI http://posample.org に関連付けられます。式が参照する purchaseOrder 文書のインスタンスは、ネーム・スペース `http://posample.org` が付いたインスタンスです。

デフォルト・ネーム・スペース宣言:

デフォルト・ネーム・スペース宣言 は、XPath 式プロローグ内で、接頭部なしの QName (修飾名) に使用するネーム・スペースを指定するオプションの宣言です。

XPath 式プロローグには、デフォルトのエレメント・ネーム・スペース宣言を含めることができます。

デフォルト・エレメント・ネーム・スペース宣言は、接頭部が付いていないエレメント名に使用されるネーム・スペース URI を指定します。XPath 式プロローグに含めることができるデフォルトのエレメント・ネーム・スペース宣言は 1 つだけです。この宣言は、宣言されている式全体で有効です。デフォルト・エレメント・ネーム・スペースが宣言されていない場合、未修飾エレメント名はネーム・スペースに属しません。

構文

default-namespace-declaration

namespace
名前空間の URI を示すストリング・リテラルを指定します。ストリング・リテラルは、有効な URI またはゼロ長ストリングでなければなりません。

namespace がゼロ長ストリングである場合、接頭部が付いていないエレメント名はネーム・スペースに属しません。

ストリング・リテラルは、http://www.w3.org/XML/1998/namespace または http://www.w3.org/2000/xmlns/ であってはなりません。

例

以下の宣言は、エレメント名のデフォルト・ネーム・スペースが URI `http://posample.org` と関連付けられているネーム・スペースであることを指定します。

```
declare default element namespace "http://posample.org";
```

この例の照会が実行されると、この式のすべてのエレメント・ノード (purchaseOrder、 shipTo、billTo、および name) は、ネーム・スペース URI `http://posample.org` に関連付けられます。

```
declare default element namespace "http://posample.org";
/purchaseOrder[shipTo/name = "Jane" and billTo/name = "Jason"]
```

この例の照会が実行されると、ネーム・スペース URI http://posample.org は、式において接頭部が付いていないすべてのエレメント名に関連付けられます。

式の評価および処理

いくつかの操作は、式の処理によく含められます。 これらの操作には、ノードからのアトミック値の抽出や、タイプ・プロモーションとサブタイプ置換を使用した期待されるタイプの値の取得などがあります。

原子化:

原子化 とは、項目のシーケンスをアトミック値のシーケンスに変換するプロセスです。 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。

シーケンス内の各項目は、以下の規則を適用することによりアトミック値に変換されます。

- 項目がアトミック値の場合、アトミック値が戻されます。
- 項目がノードである場合、その型付きの値が戻されます。*ノードの型付き値* は、ノードから抽出できるゼロ個以上のアトミック値のシーケンスです。 ノードに型付きの値がない場合、エラーが戻されます。XML 文書がスキーマによって妥当性検査される場合、DB2 for i は、ノードごとのタイプ・アノテーションを維持しません。データは常に型なしとして保管されます。型のない要素または属性の型付きの値は、xs:untypedAtomic のインスタンスとしてのストリング値です。

シーケンスの暗黙的な原子化により、シーケンスに対する明示的な fn:data 関数の呼び出しと同じ結果が作成されます。

例えば、以下のシーケンスには、ノードとアトミック値の組み合わせが含まれます。

```
("Some text", <anElement>More text</anElement>, 1001)
```

このシーケンスに原子化を適用すると、以下のアトミック値のシーケンスになります。

```
("Some text", "More text", 1001)
```

以下の XPath 式は、原子化を使用して、項目をアトミック値に変換します。

- 算術式

- 比較式
- 期待されるタイプがアトミック・タイプである引数を使用した関数の呼び出し

タイプのプロモーション:

タイプのプロモーション は、アトミック値を、そのオリジナルのタイプから式によって期待されるタイプに変換する処理です。 XPath は、数値オペランドまたはストリング・オペランドを受け入れる関数呼び出しおよび演算子の評価中に、タイプ・プロモーションを使用します。

XPath は、タイプ・プロモーションおよびサブタイプ置換を許可します。タイプ・プロモーションとサブタイプ置換は、以下のように異なります。

- タイプのプロモーションの場合、アトミック値は、そのオリジナルのタイプから式によって期待されるタイプに実際に変換されます。
- サブタイプ置換の場合、特定のタイプを期待する式を、そのタイプから派生した値を使用して呼び出すことができます。 ただし、値はそのオリジナルのタイプを保持します。

数値タイプのプロモーション:

タイプ xs:decimal (xs:decimal から制限により派生するタイプ) は、xs:double にプロモートできます。このプロモーションの結果は、オリジナル値を必要な値にキャストすることにより作成されます。

下の例では、xs:double 値 13.54e-2 が xs:decimal 値 100 に加算されます。 xs:decimal は演算を実行するために xs:double にプロモートされ、タイプ xs:double の結果が戻されます。

```
xs:double(13.54e-2) + xs:decimal(100)
```

サブタイプ置換:

サブタイプ置換 は、そのタイプが期待されるタイプから派生する値の使用です。

サブタイプ置換により、値の実際のタイプは変更されません。例えば、xs:decimal 値が期待される場所で xs:integer 値が使用された場合、値はそのタイプを xs:integer として保持します。

サブタイプ置換は、期待されるタイプから派生する値が式に渡されると必ず使用されます。

以下の例では、xs:dayTimeDuration は、fn:hours-from-duration 関数が必要とする xs:duration 値に代入されます。

```
fn:hours-from-duration(xs:dayTimeDuration("PT2H"))
```

基本式

基本式にはリテラル、変数参照、括弧で囲んだ式、コンテキスト項目式、関数呼び出しのいずれかのタイプの項目が含まれています。

構文

primary-expression:

リテラル:

DB2 XPath は、数値リテラルとストリング・リテラルの 2 種類のリテラルをサポートします。

数値リテラル は、xs:integer タイプ、xs:decimal タイプ、または xs:double タイプのアトミック値です。小数点 (.) および e または E の文字を含まない数値リテラルは、xs:integer タイプのアトミック値です。小数点 (.) を含み、e または E の文字を含まない数値リテラルは、 xs:decimal タイプのアトミック値です。e または E の文字を含む数値リテラルは、 xs:double タイプのアトミック値です。数値リテラルの値は、XML スキーマの規則に従って解釈されます。

ストリング・リテラル は、区切り文字のアポストロフィまたは引用符に囲まれた xs:string タイプのアトミック値です。ストリング・リテラルには、事前定義されたエンティティー参照および文字参照を含めることができます。

アポストロフィで区切られたストリング・リテラル内にアポストロフィを含めるには、2 つの連続するアポストロフィを指定します。同様に、引用符で区切られたストリング・リテラル内に引用符を含めるには、2 つの連続する引用符を指定します。

XML 属性の値内の XPath 式でストリング・リテラルを使用する場合、リテラルを区切るために使用される文字は、属性を区切るために使用される文字とは異なっていなければなりません。

例

数値リテラルを使用する XPath 式の例:

```
'7635'
'8735.98834'
'93948.87E+77'
```

埋め込み二重引用符があるストリング・リテラルを含む XPath 式の例:

```
SELECT X.* FROM X1,
       XMLTABLE('$inp/purchaseOrder[contains(., "string literal double-quote "" in the middle")]'
                        PASSING X1.XMLPO as "inp") X;
```

定義済みエンティティー参照:

定義済みエンティティー参照 は、DB2 XPath においてある種の構文的重要性を持つ文字を表す、文字の短いシーケンスです。

定義済みエンティティー参照は、アンパーサンド (&) で開始され、セミコロン (;) で終了します。ストリング・リテラルの処理時に、各定義済みエンティティー参照が、それが表す文字に置き換えられます。以下の表には、DB2 XPath が認識する定義済みエンティティー参照がリストされています。

表 51. DB2 XPath での定義済みエンティティー参照

エンティティー参照	表される文字
&lt;	<
&gt;	>
&amp;	&
&quot;	"
&apos;	'

文字参照:

文字参照 は、10 進数コード・ポイントまたは 16 進数コード・ポイントで識別される Unicode 文字の XML 形式のリファレンスです。

文字参照は、&#x または &# のいずれかで開始し、セミコロン (;) で終了します。文字参照が &#x で開始されている場合、終了のセミコロン (;) までの数字と文字は、ISO/IEC 10646 の文字のコード・ポイントの 16 進表記を示します。文字参照が &# で開始されている場合、終了のセミコロン (;) までの数字は、文字のコード・ポイントの 10 進表記を示します。

例

文字参照 € は、ユーロ記号を示します。

DB2 XPath での変数参照:

変数参照は、ドル記号 ($) が先頭に付加された QName です。XPath 式の評価時に、各変数参照は、変数にバインドされている式の値に解決されます。

すべての変数参照は、XPath 式の範囲内変数にある名前に一致する必要があります。範囲内変数は、XPath 式を呼び出す SQL コンテキストからバインドされます。例えば、XMLTABLE の行引数式に定義された変数がこれに該当します。

2 つの変数参照は、ローカル名が同じで、ネーム・スペースの接頭部が範囲内ネーム・スペースの同じネーム・スペース URI にバインドされている場合、同じなります。接頭部がない変数参照は、ネーム・スペースにありません。DB2 for i では、ネーム・スペース接頭部を変数名に指定することはできません。

例

以下の例では、XMLTABLE 関数は、ホスト変数 :IHV の値を $PARTNUMBER にバインドし、列 C1 の値を $QTY にバインドします。

```
SELECT X.* FROM T1, XMLTABLE('//item[@partNum = $PARTNUMBER and quantity=$QTY]'
  PASSING T1.XMLPO, :IHV AS PARTNUMBER, T1.C1 AS QTY) X;
```

括弧で囲んだ式:

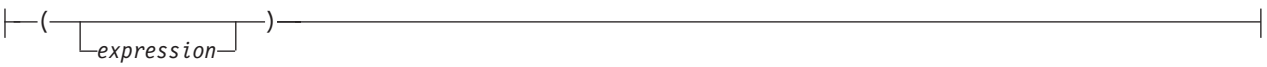
複数の演算子を含む式において評価を特定の順序で実行するために、括弧を使用できます。

括弧で囲んだ式を使用して、複雑な算術式の演算順序を明示的に指定します。

空の括弧は、空のシーケンスを表す際に使用されます。

構文

parenthesized-expression:



例

以下の例では、括弧で囲んだ式 5+5 と 6+4 が最初に評価されます。

```
((5+5) * (6+4)) div 5
```

結果は 20 になります。

コンテキスト・アイテム式:

コンテキスト・アイテム式は、1 つのピリオド (.) で構成されます。. コンテキスト・アイテム式は、現在処理中の項目 (コンテキスト・アイテム) に評価されます。コンテキスト・アイテムは、ノードまたはアトミック値のどちらでも構いません。

例

以下の例には、products 文書内の Nylon pants を識別するコンテキスト・アイテム式が含まれています。

```
declare namespace ipo="http://www.example.com/IPO";
 /ipo:products/product/name[. = "Nylon pants"]
```

関数呼び出し:

DB2 XPath は、組み込み XPath 関数の呼び出しをサポートします。

組み込み XPath 関数はネーム・スペース http://www.w3.org/2003/11/xpath-functions にあります。関数呼び出し内の関数名にネーム・スペース接頭部がない場合、関数は、デフォルトの関数ネーム・スペース内にあるとみなされます。

DB2 XPath は、以下のプロセスで関数を評価します。

1. DB2 XPath は、関数呼び出しで引数として渡される各式を評価し、各式の値を戻します。
2. 各引数について戻される値が、その引数に期待されるデータ・タイプに変換されます。 期待されるタイプが 0 個以上のアトミック・タイプのシーケンスである場合、DB2 XPath は、以下の規則を使用して、値をその期待されるタイプに変換します。
   a. 指定した値が、アトミック値のシーケンスにアトム化されます。
   b. アトミック・シーケンス内の `xs:untypedAtomic` タイプの各項目は、期待されるアトミック・タイプにキャストされます。期待されるタイプとして `numeric` が指定された組み込み関数の場合、`xs:untypedAtomic` タイプの引数は `xs:double` にキャストされます。
   c. 数値タイプのプロモーションにおいて期待されるアトミック・タイプにプロモート可能な、アトミック・シーケンス内のすべての数値項目に、数値タイプ・プロモーションが適用されます。数値項目は、`xs:integer`、`xs:decimal`、または `xs:double` タイプの項目です。
3. 関数は、その引数の変換済みの値を使用して評価されます。関数呼び出しの結果は、関数の宣言済みの戻り値のタイプのインスタンスまたはエラーのいずれかです。

例

以下の例では、製品文書の pid 属性の最初の 3 文字を取得します。

```
declare namespace pos="http://posample.org";
  fn:substring(/pos:product/@pid, 1, 3)
```

パス式

パス式 は、XML ツリー内のノードを探索します。 DB2 XPath でのパス式は、XPath 2.0 の構文に基づいています。

パス式は、スラッシュ文字 (/) または 2 つのスラッシュ文字 (//) で区切られた一連の 1 つ以上のステップから構成されます。パス式の先頭には、スラッシュ文字 (/)、2 つのスラッシュ文字 (//)、またはステップを置くことができます。

スラッシュ文字 (/) は、個別のステップを区切るために使用されます。パス式における 2 つのスラッシュ文字 (//) は、/descendant-or-self::node()/ として展開されます。これは、スラッシュ文字 (/) によって区切られているステップのシーケンスをそのままにします。各ステップは、項目のシーケンスを生成します。

パス式のステップは、左から右に評価されます。ステップが生成する項目のシーケンスは、後続のステップのコンテキスト・ノードとして使用されます。例えば、式 description/name で、最初のステップは、すべての description エレメントを含むノードのシーケンスを生成します。最後のステップは、シーケンス内の各 description 項目の name ステップを一度評価します。すべての記述項目が評価されるまで、name ステップが評価のたびに異なるフォーカスで評価されます。ステップの各評価結果のシーケンスが 1 つの文書順序にまとめてマージされ、ノード ID に基づいて、重複ノードが除去されます。

XPath ステップ式の結果は、ステップおよび述部を左から右に評価することによって判別されますが、DB2 は、より効率のよい順序で評価を実行する場合があります。そのため、場合によっては、通知されるエラーが変わることがあります。

パス式の最初のスラッシュ文字 (/) は、パスが、コンテキスト・ノードを含むツリーのルート・ノードから開始されることを意味します。そのルート・ノードは、文書ノードでなければなりません。

推奨事項: スラッシュ文字は演算子とオペランドの両方に使用できるので、演算子の最初の文字として使用する場合には、括弧を使用して、スラッシュ文字の意味を明確にしてください。例えば、乗算の左オペランドとして空のパス式を指定するには、 /*5 ではなく (/)*5 と指定します。括弧がない前者の式は、エラーになります。パス式の優先順位が高いため、 DB2 は、この式を、名前テストのワイルドカード (/*) にトークン 5 が続くパス式として解釈します。

パス式の最初の 2 つのスラッシュ文字 (//) は、コンテキスト・ノードがあるツリーのルートを含む最初のノード・シーケンス、およびそのルートの下位にあるすべてのノードを規定します。このノード・シーケンスは、パス式の後続のステップへの入力として使用されます。そのルート・ノードは、文書ノードでなければなりません。

パス式の値は、パスの最終ステップの結果である、結合された項目のシーケンスです。 この値は、一連のノードまたはアトミック値です。ノードとアトミック値が混合したものを戻すパス式の結果はエラーになります。

ステップは、軸ステップまたはフィルター式で構成されます。

構文

path-expression:

step:

例

パス式を使用して、誰かがその見積もりにつけた値段よりも高い値段の入札が少なくとも 1 つはある見積もりを判別します。

```
//stock[bid/xs:double(price) > offer/price]/@stock_id
```

軸ステップ:

*軸*ステップ は、オプションの軸、ノード・テスト、ゼロ以上の述部の 3 つの部分から構成されます。

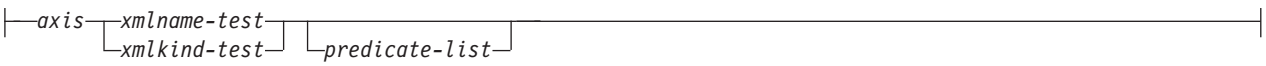
ノード・テスト により、ノードの選択に使用される基準を指定します。*述部* により、軸ステップが戻すシーケンスをフィルタリングします。

軸ステップの結果は、常にゼロ以上のノードのシーケンスになります。これらのノードは、文書の順序で戻されます。軸ステップは、フォワード・ステップまたはリバース・ステップのいずれかです。フォワード・ステップ は、コンテキスト・ノードから開始して XML ツリーを下方に移動します。リバース・ステップは、コンテキスト・ノードから開始して XML ツリーを上方に移動します。コンテキストの項目がノードでない場合、式の結果はタイプ・エラーになります。

軸ステップの非省略構文は、軸の名前とノード・テストで構成され、二重コロンで分離されています。軸式の構文は、軸を省略し、省略表現を使用することで短縮できます。

構文

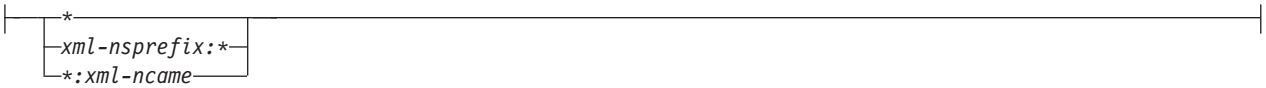
axis-step:



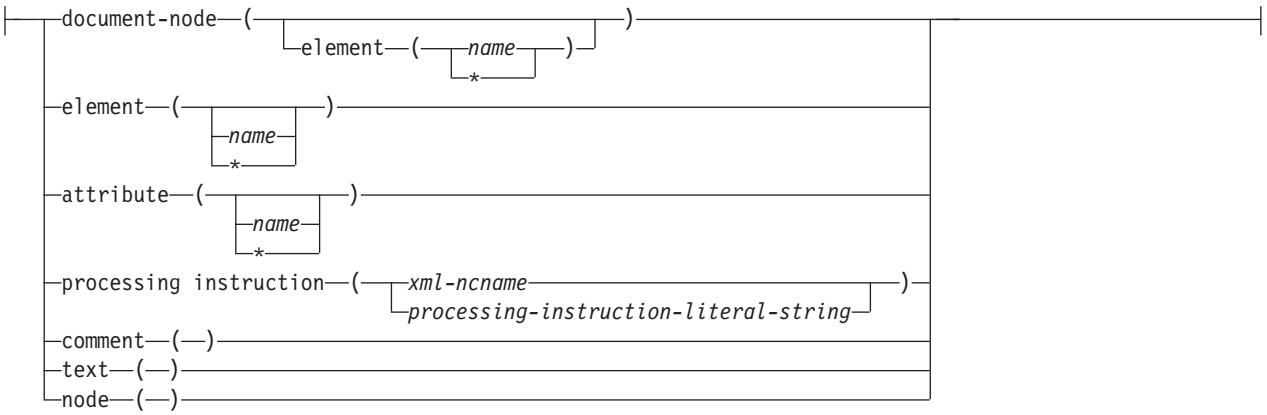
軸:

xmlname-test:

xml-wildcard:



xmlkind-test:



predicate-list:

例

以下の例では、child が軸の名前で、price がこの軸上で選択されるエレメント・ノードの名前です。

```
child::price
```

この例の軸ステップでは、コンテキスト・ノードの子であるすべての price エレメントを選択します。

軸:

軸 とは、XML 文書内を移動する方向を指定する軸ステップのオプション部分です。

表 52 では、DB2 XPath でサポートされる軸について説明します。

表 52. DB2 XPath でサポートされる軸

軸	説明	注
child	コンテキスト・ノードの子を戻します。この軸はデフォルトです。	子を持つノードは、文書ノードおよびエレメント・ノードのみです。コンテキスト・ノードがそれ以外の種類のノードである場合、またはコンテキスト・ノードが空の文書ノードまたはエレメント・ノードである場合、child 軸は空のシーケンスです。エレメント・ノード、処理命令ノード、コメント・ノード、またはテキスト・ノードは、文書ノードまたはエレメント・ノードの子である場合があります。属性ノードおよび文書ノードは子として表示されることはありません。
descendant	コンテキスト・ノードの子孫 (子、子の子など) を戻します。	
attribute	コンテキスト・ノードの属性を戻します。	この軸は、コンテキスト・ノードがエレメント・ノードでない場合、空です。

表 52. *DB2 XPath* でサポートされる軸 *(続き)*

軸	説明	注
self	コンテキスト・ノードのみを戻します。	
descendant-or-self	コンテキスト・ノードおよびコンテキスト・ノードの子孫を戻します。	
parent	コンテキスト・ノードの親を戻します。コンテキスト・ノードが親を持たない場合、空のシーケンスを戻します。	属性ノードがエレメント・ノードの子になることはありませんが、エレメント・ノードは属性ノードの親となることができます。

DB2 XPath は、W3 標準の全軸機能によって定義された追加の軸をサポートしません。

軸は、フォワード軸またはリバース軸のいずれかに分類できます。*フォワード軸* には、コンテキスト・ノード、および文書の順序においてコンテキスト・ノードの後にあるノードが含まれます。*リバース軸* には、コンテキスト・ノード、および文書の順序においてコンテキスト・ノードの前にあるノードが含まれます。DB2 XPath では、フォワード軸には child、descendant、attribute、self、および descendant-or-self があります。サポートされているリバース軸は parent 軸のみです。

軸ステップがノードのシーケンスを選択すると、各ノードには、そのシーケンスの位置に対応するコンテキストの位置が割り当てられます。 軸がフォワード軸の場合、コンテキストの位置は、文書の順序で 1 からノードに割り当てられます。軸がリバース軸の場合、文書の逆順で 1 からノードに割り当てられます。

ノード・テスト:

ノード・テスト は、軸ステップにより選択される各ノードについて true でなければならない条件です。ノード・テストは、名前テストまたは種類テストとして表されます。*名前テスト* は、ノードの名前に基づいてノードを選択します。 *種類テスト* は、ノードの種類に基づいてノードを選択します。

名前テスト

名前テストは、QName またはワイルドカードで構成されます。 軸ステップで名前テストが指定されると、ステップは、QName またはワイルドカードに一致する指定された軸上のノードを選択します。 名前テストが attribute 軸で指定される場合、ステップは名前テストに一致する任意の属性を選択します。それ以外の場合、ステップは、他のすべての軸において、名前テストに一致する任意のエレメントを選択します。 QName が一致するためには、ノードの拡張 QName が、名前テストで指定される拡張 QName に (コード・ポイント・ベースで) 等しくなければなりません。2 つの拡張 QName は、そのネーム・スペース URI が等しく、ローカル名も等しい場合に (ネーム・スペース接頭部が等しくなくても) 等しくなります。

重要: 名前テストで指定される接頭部は、式の静的に既知の名前空間の 1 つに対応する必要があります。属性軸について実行される名前テストの場合、接頭部なしの QName には名前空間 URI はありません。他のすべての軸について実行される名前テストの場合、接頭部なしの QName にはデフォルト・エレメント・ネーム・スペースのネーム・スペース URI があります。

176 ページの表 53 では、DB2 XPath でサポートされる名前テストを説明しています。

表 53. DB2 XPath でサポートされる名前テスト

テスト	説明	例
QName	その QName が、指定された QName に等しい (指定された軸上の) 任意のノードに一致します。 軸が attribute 軸の場合、このテストは指定された QName に等しい属性ノードに一致します。他のすべての軸の場合、このテストは指定された QName に等しいエレメント・ノードに一致します。	式 `child::para` において、名前テスト `para` は、子軸上のすべての `para` 要素を選択します。
*	指定された軸上のすべてのノードに一致します。 軸が属性軸の場合、このテストはすべての属性ノードに一致します。 他のすべての軸の場合、このテストはすべての要素ノードに一致します。	式 `child::*` において、名前テスト * は、child 軸上のすべてのエレメントに一致します。

種類テスト

軸ステップで種類テストが指定されると、ステップは、種類テストに一致する指定された軸上のノードのみを選択します。 表 54 では、DB2 XPath でサポートされる種類テストを説明しています。

表 54. DB2 XPath でサポートされる種類テスト

テスト	説明	例
node()	指定された軸上の任意のノードに一致します。	式 `child::node()` において、種類テスト `node()` は、子軸上の任意のノードを選択します。
text()	指定された軸上の任意のテキスト・ノードに一致します。	式 `child::text()` において、種類テスト `text()` は、子軸上の任意のテキスト・ノードを選択します。
comment()	指定された軸上の任意のコメント・ノードに一致します。	式 `child::comment()` において、種類テスト `comment()` は、子軸上の任意のコメント・ノードを選択します。
processing-instruction(*NCName*)	この名前テストで指定されている NCName に一致する名前 (XML では "PITarget"と呼ばれています) の任意の処理命令ノード (指定された軸上にある) に一致します。	式 `child::processing-instruction (xml-stylesheet)` において、種類テスト `processing-instruction(xml-stylesheet)` は、 PITarget が xml-stylesheet である child 軸上の任意の処理命令ノードを選択します。
processing-instruction(*StringLiteral*)	このテストで指定されているストリング・リテラルに一致する名前の任意の処理命令ノード (指定された軸上にある) に一致します。 このノード・テストには、XPath 1.0 との後方の互換性があります。	式 `child::processing-instruction ("xml-stylesheet")` において、種類テスト `processing-instruction("xml-stylesheet")` は、PITarget が xml-stylesheet である child 軸上の任意の処理命令ノードを選択します。
element()	指定された軸上の任意の要素ノードに一致します。	式 `child::element()` において、種類テスト `element()` は、child 軸上の任意のエレメント・ノードを選択します。

表 54. DB2 XPath でサポートされる種類テスト (続き)

テスト	説明	例
element(*QName*)	このテストで指定されている修飾名に一致する名前の任意のエレメント・ノード (指定された軸上にある) に一致します。	式 `child::element("price")` において、種類テスト `element("price")` は、price という名前の、child 軸上の任意のエレメント・ノードを選択します。
element(*)	指定された軸上の任意の要素ノードに一致します。	式 `child::element(*)` において、種類テスト `element(*)` は、子軸上の任意の要素ノードを選択します。
attribute()	指定された軸上の任意の属性ノードに一致します。	式 `child::attribute()` において、種類テスト `attribute()` は、child 軸上の任意の属性ノードを選択します。
attribute(*QName*)	このテストで指定されている修飾名に一致する名前の任意の属性ノード (指定された軸上にある) に一致します。	式 `child::attribute("price")` において、種類テスト `attribute("price")` は、price という名前の、child 軸上の任意の属性ノードを選択します。
attribute(*)	指定された軸上の任意の属性ノードに一致します。	式 `child::attribute(*)` において、種類テスト `attribute(*)` は、子軸上の任意の属性ノードを選択します。
document-node()	指定された軸上の任意の文書ノードに一致します。	式 `self::document-node()` では、種類テスト `document-node()` で self 軸上のいずれかの文書ノードが選択されます。
document-node(element(*QName*))	指定された軸上で、エレメント・ノードが 1 つしかない文書ノードと一致します。	式 `self::document-node(element("price"))` では、種類テスト `document-node(element("price"))` によって self 軸上で price という名前の単一のルート・エレメントを持ついずれかの文書ノードが選択されます。
document-node(element(*))	エレメント・ノードを持つ、指定された軸上の任意の文書ノードに一致します。	式 `self::document-node(element(*))` では、種類テスト `document-node(element(*))` によって self 軸上でエレメント・ノードを持ついずれかの文書ノードが選択されます。

述部:

述部 は、大括弧 ([]) で囲まれた式 (述部式と呼びます) で構成されます。述部は、一部の項目を保持し、それ以外の項目を破棄することにより、シーケンスをフィルタリングします。

述部式は、シーケンス内の各項目について 1 回評価されます。述部式の結果は、*述部真理値* と呼ばれる `xs:boolean` 値です。述部真理値が `true` である項目は保持され、述部真理値が `false` である項目は破棄されます。

述部式の値は、その静的タイプが数値 singleton である限り、数値とすることができます。述部式の静的タイプが数値 singleton である場合、評価対象の項目のシーケンスにおけるコンテキスト項目の位置が数値 singleton と一致すれば、述部真理値は `true` となります。つまり、`child::employee/child::address[2]`

は child::employee/child::address[fn:position() = 2] と同等であり、employee の下の 2 番目のアドレスを戻します。述部式が式の構文解析時に singleton と判断できない数値である場合、非サポートのエラーが通知されます。

その他すべてのデータ・タイプの場合、述部真理値は、述部式の有効なブール値です。有効なブール値は、述部式の評価が空のシーケンスまたは false である場合には false です。それ以外の場合、有効なブール値は true です。

述部がアトミック値または関数呼び出しのフィルター処理に使用される場合、非サポートのエラーが通知されることがあります。

例

以下の例は、述部が含まれている軸ステップです。

- descendant::phone[attribute::type = "work"] は、 phone という名前で、タイプ属性値が「work」になっているエレメントであるコンテキスト・ノードのすべての子孫を選択します。
- child::address[prov-state][pcode-zip] は、 prov-state 子エレメントと pcode-zip 子エレメントを持つコンテキスト・ノードの address という名前のすべての子を選択します。

パス式の省略構文:

DB2 XPath には、パス式の軸を示すための省略構文があります。

表 55 に、パス式で許可されている省略形を示します。

表 55. パス式の省略構文

省略構文	説明
軸の指定なし	軸ステップでノード・テストに attribute() が指定される場合以外は、child:: の省略形です。軸ステップが属性テストを指定するときは、省略される軸は、attribute:: の省略形です。
@	attribute:: の省略形です。
//	この省略形はパス式の先頭にあるとき以外は、/descendant-or-self::node()/ の省略形です。 この省略表記がパス式の先頭に存在すると、その軸ステップでは、コンテキスト・ノードがあるツリーのルートを含む最初のノード・シーケンス、およびそのルートの下位にあるすべてのノードが選択されます。 この式では、ルート・ノードが文書ノードでない場合、エラーが発生します。
..	parent::node() の省略形です。

省略構文および非省略構文の例

表 56 に、省略構文および非省略構文の例を示します。

表 56. 非省略構文および省略構文の比較

非省略構文	省略構文	結果
child::para	para	コンテキスト・ノードの子である para 要素を選択します。

表 56. 非省略構文および省略構文の比較 (続き)

非省略構文	省略構文	結果
child::*	*	コンテキスト・ノードの子であるすべてのエレメントを選択します。
child::text()	text()	コンテキスト・ノードの子であるすべてのテキスト・ノードを選択します。
child::node()	node()	コンテキスト・ノードのすべての子を選択します。 属性はノードの子でないため、この式では、属性ノードは戻されません。
attribute::name	@name	コンテキスト・ノードの name 属性を選択します。
attribute::*	@*	コンテキスト・ノードのすべての属性を選択します。
child::para[attribute::type="warning"]	para[@type="warning"]	値が warning のタイプ属性を持つコンテキスト・ノードのすべての子 para を選択します。
child::chapter[child::title="Introduction"]	chapter[title="Introduction"]	型付き値がストリング Introduction と同じである 1 つ以上の子 title を持つコンテキスト・ノードの子 chapter を選択します。
child::chapter[child::title]	chapter[title]	1 つ以上の子 title を持つコンテキスト・ノードの子 chapter を選択します。

フィルター式

フィルター式は、基本式と、それに続く 0 個以上の述部で構成されます。 述部が存在する場合、 1 次式の結果をフィルタリングします。

フィルター式の結果は、すべての述部が true である 1 次式によって戻されるすべての項目から構成されます。述部が指定されていない場合、結果は 1 次式の結果になります。この結果には、ノード、アトミック値、またはノードとアトミック値の組み合わせが含まれます。フィルター式によって戻される項目の順序は、1 次式の結果の順序と同じです。コンテキストの位置は、結果シーケンスの位置順序に基づいて項目に割り当てられます。 1 番目のコンテキストの位置は、1 です。

構文

filter-expression

例

以下の例では、$x で指定された文書のどこかに customerinfo 要素がある場合に $x という値を戻すフィルター式を使用します。

```
declare default element namespace "http://posample.org";
  $x[.//customerinfo]
```

算術式

算術式では、加算、減算、乗算、除算、モジュラスなどの演算を実行します。

以下の表では、算術演算子を演算子優先順位の高いものから低いものの順にリストし、それぞれの演算子について説明しています。単項演算子の優先順位は、2 項演算子よりも高くなります。ただし、括弧を使用した場合は、2 項演算子の計算が先に行われます。

表 57. DB2 XPath での算術演算子

演算子	目的	結合順序
- (単項)、+ (単項)	オペランド値を反対の符号の値にする、オペランド値を維持する	右から左
*、div、idiv、mod	乗算、除算、整数除算、係数	左から右
+, -	加算、減算	左から右

注: 減算演算子の前には空白文字が必要です。空白文字がないと、演算子が前のトークンの一部であると解釈されます。 例えば、`a-b` は名前と解釈されますが、`a - b` や `a -b` は、算術演算と解釈されます。

算術式の結果は、次のいずれかの項目です。

- 数値
- 日付値または時刻値
- 期間値
- 空のシーケンス
- エラー

DB2 XPath は、以下のプロセスで算術式を評価します。

1. 各オペランドをアトミック値のシーケンスにアトム化します。
2. 以下の規則を使用して、算術式のオペランドを評価します。
   - アトム化されたオペランドが空のシーケンスである場合、算術式の結果は空のシーケンスです。
   - 原子化されたオペランドが複数の値を含むシーケンスである場合、エラーが戻されます。
   - アトム化されたオペランドが型のないアトミック値 (xs:untypedAtomic) である場合、DB2 XPath はその値を xs:double にキャストします。キャストが失敗すると、DB2 XPath はエラーを戻します。
3. オペランドのタイプがその算術演算子に有効な組み合わせである場合、DB2 XPath は、その演算子をアトム化された値に適用します。この操作の結果は、アトミック値または動的エラーです (例えば、エラーは、ゼロによる xs:integer の除算によって発生します)。
4. オペランドのタイプが算術演算子に有効な組み合わせでない場合、DB2 XPath はタイプ・エラーと見なします。

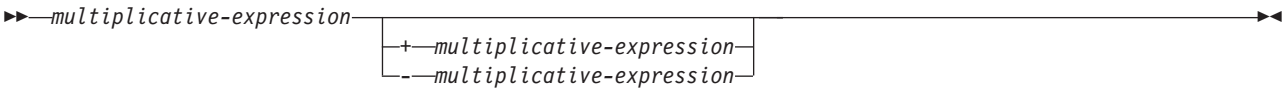
以下の表に、算術演算子に有効な組み合わせのタイプを示します。この表では、文字 A は式の第 1 オペランドを、文字 B は第 2 オペランドを示します。数値という用語は、タイプ xs:integer、xs:decimal、xs:double、またはこのいずれかのタイプから派生された任意のタイプを示します。演算子の結果タイプが数値としてリストされている場合、結果タイプは、順序付けリスト (xs:integer、xs:decimal、xs:double) のタイプのうち、サブタイプ置換およびタイプ・プロモーションによってすべてのオペランドをそのタイプに変換できる、最初のタイプになります。

表 58. 算術式のオペランドに有効なタイプ

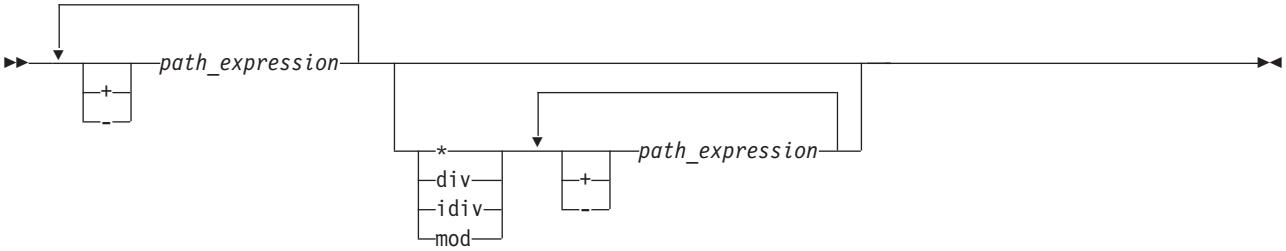
演算子とオペランド	オペランド A タイプ	オペランド B タイプ	結果タイプ
A + B	数値	数値	数値
	xs:date	xs:yearMonthDuration	xs:date
	xs:yearMonthDuration	xs:date	xs:date
	xs:date	xs:dayTimeDuration	xs:date
	xs:dayTimeDuration	xs:date	xs:date
	xs:time	xs:dayTimeDuration	xs:time
	xs:dayTimeDuration	xs:time	xs:time
	xs:dateTime	xs:yearMonthDuration	xs:dateTime
	xs:yearMonthDuration	xs:dateTime	xs:dateTime
	xs:dateTime	xs:dayTimeDuration	xs:dateTime
	xs:dayTimeDuration	xs:dateTime	xs:dateTime
	xs:yearMonthDuration	xs:yearMonthDuration	xs:yearMonthDuration
	xs:dayTimeDuration	xs:dayTimeDuration	xs:dayTimeDuration
A - B	数値	数値	数値
	xs:date	xs:date	xs:dayTimeDuration
	xs:date	xs:yearMonthDuration	xs:date
	xs:date	xs:dayTimeDuration	xs:date
	xs:time	xs:time	xs:dayTimeDuration
	xs:time	xs:dayTimeDuration	xs:time
	xs:dateTime	xs:dateTime	xs:dayTimeDuration
	xs:dateTime	xs:yearMonthDuration	xs:dateTime
	xs:dateTime	xs:dayTimeDuration	xs:dateTime
	xs:yearMonthDuration	xs:yearMonthDuration	xs:yearMonthDuration
	xs:dayTimeDuration	xs:dayTimeDuration	xs:dayTimeDuration
A * B	数値	数値	数値
	xs:yearMonthDuration	数値	xs:yearMonthDuration
	数値	xs:yearMonthDuration	xs:yearMonthDuration
	xs:dayTimeDuration	数値	xs:dayTimeDuration
	数値	xs:dayTimeDuration	xs:dayTimeDuration
A idiv B	数値	数値	xs:integer
A div B	数値	数値	数値 (両方のオペランドが xs:integer の場合は xs:decimal)
	xs:yearMonthDuration	数値	xs:yearMonthDuration
	xs:dayTimeDuration	数値	xs:dayTimeDuration
	xs:yearMonthDuration	xs:yearMonthDuration	xs:decimal
	xs:dayTimeDuration	xs:dayTimeDuration	xs:decimal
A mod B	数値	数値	数値

構文

arithmetic expression



multiplicative expression



例

以下の照会は、算術式を使用して、購入者が支払う税金の額 (8.25% の税率で製品にかかる) を計算し、税金が 1 通貨単位を超える記述エレメントを選択します。

```
SELECT X.* FROM T1, XMLTABLE('declare namespace pos="http://posample.org";
                             /pos:product/description[price * .0825 > 1]'
                             PASSING T1.DESCRIPTION) X;
```

以下の照会は、2 つの xs:date 値を減算します。その結果は xs:yearMonthDuration 値 P8559D になります。

```
SELECT * FROM XMLTABLE('xs:date("2005-10-10") - xs:date("1982-05-05")') X;
```

比較式

比較式では、2 つの値を比較します。 DB2 XPath では、1 種類の比較式 (汎用比較) が用意されています。

一般比較:

一般比較は、任意の長さの 2 つのシーケンスを比較して、両方のシーケンスの少なくとも 1 つの項目について、比較が true であるかどうかを判別します。一般比較演算子には、 =、!=、<、<=、>、および >= があります。

以下の表に、比較演算子とその説明を示します。演算子優先順位の高いものから順にリストしています。

表 59. XPath における一般比較演算子

演算子	目的
=	1 番目のシーケンス内のいずれかの値が 2 番目のシーケンス内のいずれかの値と等しい場合に、true を戻します。
!=	1 番目のシーケンス内のいずれかの値が 2 番目のシーケンス内のいずれかの値と等しくない場合に、true を戻します。

表 59. XPath における一般比較演算子 (続き)

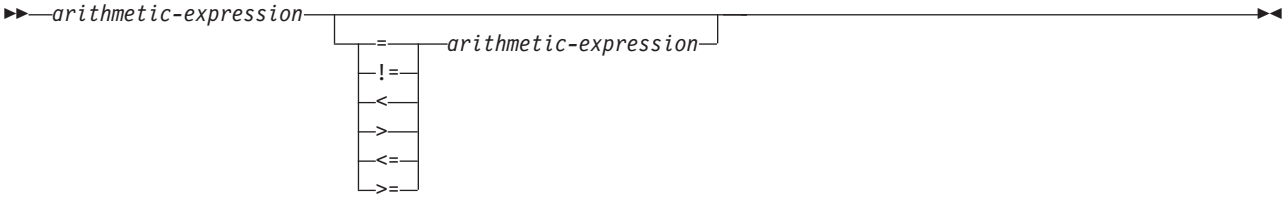
演算子	目的
<	1 番目のシーケンス内のいずれかの値が 2 番目のシーケンス内のいずれかの値より小さい場合に、true を戻します。
<=	1 番目のシーケンス内のいずれかの値が 2 番目のシーケンス内のいずれかの値以下の場合に、true を戻します。
>	1 番目のシーケンス内のいずれかの値が 2 番目のシーケンス内のいずれかの値より大きい場合に、true を戻します。
>=	1 番目のシーケンス内のいずれかの値が 2 番目のシーケンス内のいずれかの値以上の場合に、true を戻します。

一般比較式の結果は、ブール値またはエラーのいずれかです。DB2 XPath は、以下のプロセスで一般比較式を評価します。

1. 各オペランドをアトミック値のシーケンスにアトム化します。
2. 1 番目のシーケンス内の各値を 2 番目のシーケンス内の各値と比較します。ストリング比較の場合、デフォルトの照合が使用されます。それぞれの比較は、以下のようになります。
   - 一方のアトミック値が `xs:untypedAtomic` のインスタンスであり、他方のアトミック値が数値タイプ (xs:integer、xs:decimal、または xs:double) のインスタンスである場合、型のない値は `xs:double` タイプにキャストされます。
   - 一方のアトミック値が `xs:untypedAtomic` のインスタンスであり、他方のアトミック値が `xs:untypedAtomic` または `xs:string` のインスタンスである場合、 `xs:untypedAtomic` 値は `xs:string` タイプにキャストされます。
   - 一方のアトミック値が `xs:untypedAtomic` のインスタンスであり、他方のアトミック値が `xs:string`、`xs:untypedAtomic`、または任意の数値タイプのインスタンスでない場合、`xs:untypedAtomic` 値は他方の値の動的タイプにキャストされます。
3. 1 番目のシーケンス内の少なくとも 1 つの値と 2 番目のシーケンス内の少なくとも 1 つの値が、比較の条件を満たしている場合、一般比較は true になります。

構文

comparison expression



例

以下のステートメントは、一般比較式を使用して、価格が 20 通貨単位未満の製品の記述を検索します。

```
declare namespace pos="http://posample.org";
  /pos:product/description[price < 20]
```

論理式

論理式は、2 つの式が両方とも真 (true) である場合、または 2 つの式の一方あるいは両方が真 (true) である場合に、ブール値 true を戻します。論理式で使用される演算子には、and と or があります。

以下の表に、比較演算子とその説明を示します。演算子優先順位の高いものから順にリストしています。

表 60. XPath の論理式演算子

演算子	目的
and	両方の式が true の場合に true を戻す。
あるいは	一方または両方の式が true の場合に true を戻す。

論理式の結果は、ブール値またはエラーのいずれかです。DB2 XPath は、以下のプロセスで論理式を評価します。

1. 各オペランドの有効なブール値 (EBV) を決定します。
2. 演算子をオペランドの有効なブール値に適用します。結果は、ブール値またはエラーのいずれかです。表 61 に、オペランドの EBV に基づいて論理式から戻される結果と、オペランドの評価中に検出されるエラーを示します。

表 61. オペランドの有効なブール値 (EBV) に基づく論理式の結果

オペランド 1 の EBV	演算子	オペランド 2 の EBV	結果
true	and	true	true
true	and	false	false
false	and	true	false
false	and	false	false
true	and	エラー	エラー
エラー	and	true	エラー
false	and	エラー	false またはエラー
エラー	and	false	false またはエラー
エラー	and	エラー	エラー
true	あるいは	true	true
false	あるいは	false	false
true	あるいは	false	true
false	あるいは	true	true
true	あるいは	エラー	true またはエラー
エラー	あるいは	true	true またはエラー
false	あるいは	エラー	エラー
エラー	あるいは	false	エラー
エラー	あるいは	エラー	エラー

構文

論理式

例

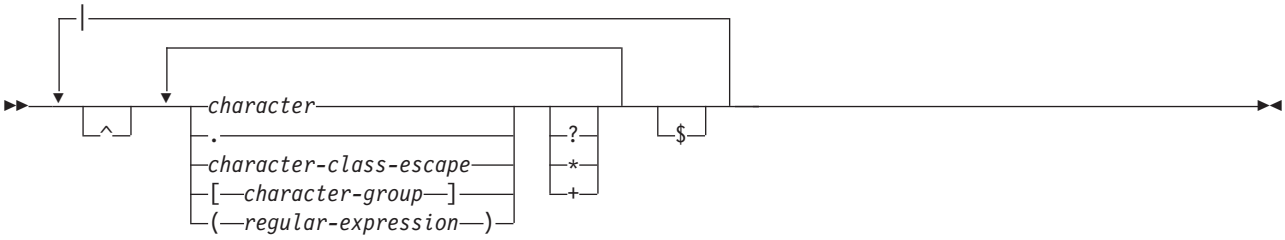
以下の例では、論理式を使用して、22 インチのスノー・ショベルまたは 24 インチのスノー・ショベルのレコードを検索します。

```
declare namespace pos="http://posample.org";
  /pos:product/description[name = "Snow Shovel, Deluxe 24"
                           or name = "Snow Shovel, Basic 22"]
```

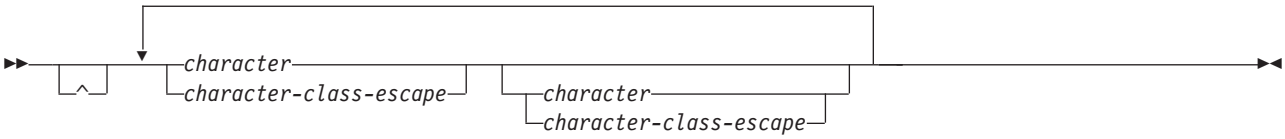
正規表現

正規表現は、ストリングのマッチングおよび操作のためのパターンとして機能する文字のシーケンスです。正規表現は、fn:matches 関数、fn:replace 関数、および fn:tokenize 関数で使用されます。

構文



character-group



文字

正規表現において、*character* はメタキャラクターでない通常の XML 文字です。

メタキャラクター

メタキャラクターは、正規表現における制御文字です。現在サポートされている正規表現メタキャラクターは、以下のとおりです。

バックスラッシュ (\)

文字クラス・エスケープを開始します。文字クラス・エスケープは、それに続くメタキャラクターを、メタキャラクターではなく文字として使用することを示します。

ピリオド (.)

改行文字 (\n) を除く任意の単一文字に一致します。

脱字記号 (^)

文字クラス外に脱字記号文字が現れる場合、その脱字記号に続く文字列は、入力ストリングの先

頭、または複数行入力ストリングの場合は行の先頭に一致します。入力ストリングを使用する関数に m フラグが含まれている場合、その入力ストリングは複数行入力ストリングであるとみなされます。

脱字記号文字が文字クラス内の先頭文字として現れる場合、その脱字記号は否定記号として機能します。正規表現に対して比較されているストリング内に、文字グループ内の文字がどれも現れない場合に、一致します。

ドル記号 ($)
入力ストリングの末尾、または複数行入力ストリングの場合は行の末尾に一致します。入力ストリングを使用する関数に m フラグが含まれている場合、その入力ストリングは複数行入力ストリングであるとみなされます。

疑問符 (?)
正規表現内の前の文字または文字グループに 0 回か 1 回一致します。

アスタリスク (*)
正規表現内の前の文字または文字グループに 0 回以上一致します。

正符号 (+)
正規表現内の前の文字または文字グループに 1 回以上一致します。

パイプ (|)
前の文字 (または文字グループ) または後の文字 (または文字グループ) と一致します。

左大括弧 ([) および右大括弧 (])
左大括弧と右大括弧、および囲まれている文字グループが、文字クラスを定義します。例えば、文字クラス [aeiou] は、任意の単一母音に一致します。文字クラスは、文字の範囲もサポートします。例えば、次の通りです。

- [a-z] は、任意の小文字を意味します。
- [a-p] は、a から p までの任意の小文字を意味します。
- [0-9] は、任意の単一の数字を意味します。

左括弧 (() および右括弧 ())
左括弧と右括弧は、正規表現内の一部の文字のグループ化を示します。その後、反復演算子などの演算子をグループ全体に適用できます。

左中括弧 ({) と右中括弧 (}) もメタキャラクターですが、それらは現在はサポートされていません。

character-class-escape
文字クラス・エスケープは、特定の特殊文字を、何らかの機能を実行させるのではなく、文字として扱うことを指定します。文字クラス・エスケープは、バックスラッシュ (\) と、それに続く単一のメタキャラクター、改行文字、リターン文字、またはタブ文字で構成されます。次の表に、文字クラス・エスケープをリストします。

表 62. 単一文字の文字クラス・エスケープ

文字エスケープ	表される文字	説明
¥n	#x0A	改行
¥r	#x0D	リターン
¥t	#x09	タブ
\\	\	円記号
\\|	|	パイプ
\.	.	ピリオド

表 62. 単一文字の文字クラス・エスケープ (続き)

文字エスケープ	表される文字	説明
\?	?	疑問符 (?)
*	*	アスタリスク
\+	+	正符号
\(	(	左括弧
\)	)	右括弧
\{	{	左中括弧
\}	}	右中括弧
\$	$	ドル記号
\-	-	ダッシュ
\[	[	左大括弧
\]	]	右大括弧
¥^	^	脱字

character-group
文字グループは、文字クラス内の文字のセットです。文字クラスはマッチングに使用されます。これは、文字、文字の範囲、文字クラスのエスケープ、およびオプションの左脱字記号から構成することができます。脱字記号が含まれている場合、残りの文字グループによって定義されている文字のセットの補集合を示します。

例

以下の例は、それぞれのメタキャラクターがどのように正規表現に影響するかを示しています。

- "hello[0-9]world" は、"hello3world" に一致しますが、"hello world" には一致しません。
- "^hello" は次のテキストに一致します。

  ```
  hello world
  ```

  ただし、"^hello" は次のテキストには一致しません。

  ```
  world hello
  ```
- "hello$" は次のテキストに一致します。

  ```
  world hello
  ```

  ただし、"hello$" は次のテキストには一致しません。

  ```
  hello world
  ```
- "(ca)|(bd)" は "arcade" または "abdicate" に一致します。
- "^((ca)|(bd))" は "arcade" または "abdicate" には一致しません。
- "w?s" は "ws" または "s" に一致します。
- "w.*s" は "was" または "waters" に一致します。
- "be+t" は "beet" または "bet" に一致します。
- "\[n\]" は "[n]" に一致します。

XPath 関数の説明

DB2 XPath 関数は、XPath 2.0 および XQuery 1.0 の関数および演算子のサブセットです。

以下のトピックには、DB2 for i によってサポートされる XPath 関数の詳細な参照情報が記載されています。関数は、XPath 式内で、式が期待されるどの場所にも使用できます。サポートされる関数は、以下の表にリストされています。

表 63. ストリング関数

機能	説明	参照
fn:compare	2 つのストリングを比較して、等しい、より小、またはより大のいずれであるかの標識を戻します。	197 ページの『fn:compare 関数』
fn:concat	2 つ以上のストリングを連結して単一のストリングにしたものを戻します。	198 ページの『fn:concat 関数』
fn:contains	ストリングに指定のサブストリングが含まれているかどうかの標識を戻します。	199 ページの『fn:contains 関数』
fn:lower-case	小文字に変換されたストリングを戻します。	211 ページの『fn:lower-case 関数』
fn:matches	ストリングが指定のパターンに一致するかどうかの標識を戻します。	212 ページの『fn:matches 関数』
fn:max	シーケンス内の最大値を戻します。	213 ページの『fn:max 関数』
fn:min	シーケンス内の最小値を戻します。	214 ページの『fn:min 関数』
fn:normalize-space	先行および末尾の空白文字が除去され、内側の空白文字の連続する箇所がそれぞれ単一のブランク文字に置き換えられたストリングを戻します。	220 ページの『fn:normalize-space 関数』
fn:replace	パターンに一致する文字を持つストリングを、別の文字セットに置き換えて戻します。	222 ページの『fn:replace 関数』
fn:starts-with	ストリングが指定のサブストリングで始まっているかどうかの標識を戻します。	226 ページの『fn:starts-with 関数』
fn:string	値のストリング表記を戻します。	226 ページの『fn:string 関数』
fn:string-length	ストリングの長さを戻します。	227 ページの『fn:string-length 関数』
fn:substring	ストリングのサブストリングを戻します。	227 ページの『fn:substring 関数』
fn:tokenize	ストリング内のサブストリングのリストを戻します。	231 ページの『fn:tokenize 関数』
fn:translate	ストリングのうちの選択された文字を別の文字に置き換えて、そのストリングを戻します。	232 ページの『fn:translate 関数』
fn:upper-case	大文字に変換されたストリングを戻します。	233 ページの『fn:upper-case 関数』

表 64. 数値関数

機能	説明	参照
fn:abs	数値の絶対値を戻します。	191 ページの『fn:abs 関数』
fn:max	シーケンス内の最大値を戻します。	213 ページの『fn:max 関数』
fn:min	シーケンス内の最小値を戻します。	214 ページの『fn:min 関数』
fn:round	指定された値に最も近い整数を戻します。	223 ページの『fn:round 関数』
fn:sum	シーケンス内の値の合計を戻します。	228 ページの『fn:sum 関数』

表 65. ブール関数

機能	説明	参照
fn:boolean	シーケンスの有効なブール値を戻します。	197 ページの『fn:boolean 関数』
fn:exists	その引数により結果が空となる場合は、false を戻します。それ以外の場合は、true を戻します。	206 ページの『fn:exists 関数』
fn:not	シーケンス式の有効なブール値が true である場合は、false を戻し、シーケンス式の有効なブール値がfalseである場合は、true を戻します。	220 ページの『fn:not 関数』

表 66. 日付、時間、および期間関数

機能	説明	参照
fn:adjust-date-to-timezone	時間帯が調整または除去された xs:date 値を戻します。	192 ページの『fn:adjust-date-to-timezone 関数』
fn:adjust-dateTime-to-timezone	時間帯が調整または除去された xs:dateTime 値を戻します。	194 ページの『fn:adjust-dateTime-to-timezone 関数』
fn:adjust-time-to-timezone	時間帯が調整または除去された xs:time 値を戻します。	195 ページの『fn:adjust-time-to-timezone 関数』
fn:current-date	UTC の暗黙的な時間帯における現在の日付を戻します。	200 ページの『fn:current-date 関数』
fn:current-dateTime	UTC の暗黙的な時間帯における現在の日時を戻します。	200 ページの『fn:current-dateTime 関数』
db2-fn:current-local-date	ローカル時間帯における現在の日付を戻します。	201 ページの『db2-fn:current-local-date 関数』
db2-fn:current-local-dateTime	ローカル時間帯における現在の日時を戻します。	201 ページの『db2-fn:current-local-dateTime 関数』
db2-fn:current-local-time	ローカル時間帯における現在の時刻を戻します。	201 ページの『db2-fn:current-local-time 関数』
fn:current-time	UTC の暗黙的な時間帯における現在の時刻を戻します。	202 ページの『fn:current-time 関数』
fn:dateTime	xs:date 値および xs:time 値からの xs:dateTime 値を戻します。	203 ページの『fn:dateTime 関数』
fn:day-from-date	xs:date 値の日コンポーネントを戻します。	203 ページの『fn:day-from-date 関数』
fn:day-from-dateTime	xs:dateTime 値の日コンポーネントを戻します。	204 ページの『fn:day-from-dateTime 関数』
fn:days-from-duration	期間の日コンポーネントを戻します。	204 ページの『fn:days-from-duration 関数』

表 66. 日付、時間、および期間関数 (続き)

機能	説明	参照
fn:hours-from-dateTime	xs:dateTime 値の時コンポーネントを戻します。	206 ページの『fn:hours-from-dateTime 関数』
fn:hours-from-duration	期間値の時コンポーネントを戻します。	207 ページの『fn:hours-from-duration 関数』
fn:hours-from-time	xs:time 値の時コンポーネントを戻します。	208 ページの『fn:hours-from-time function』
fn:implicit-timezone	UTC が暗黙的な時間帯であることを示す PT0S の暗黙的時間帯値を戻します。	208 ページの『fn:implicit-timezone 関数』
db2-fn:local-timezone	ローカル・システムの時間帯を戻します。	211 ページの『db2-fn:local-timezone 関数』
fn:max	シーケンス内の最大値を戻します。	213 ページの『fn:max 関数』
fn:min	シーケンス内の最小値を戻します。	214 ページの『fn:min 関数』
fn:minutes-from-dateTime	xs:dateTime 値の分コンポーネントを戻します。	215 ページの『fn:minutes-from-dateTime 関数』
fn:minutes-from-duration	期間の分コンポーネントを戻します。	215 ページの『fn:minutes-from-duration 関数』
fn:minutes-from-time	xs:time 値の分コンポーネントを戻します。	216 ページの『fn:minutes-from-time 関数』
fn:month-from-date	xs:date 値の月コンポーネントを戻します。	217 ページの『fn:month-from-date 関数』
fn:month-from-dateTime	xs:dateTime 値の月コンポーネントを戻します。	217 ページの『fn:month-from-dateTime 関数』
fn:months-from-duration	期間の月コンポーネントを戻します。	218 ページの『fn:months-from-duration 関数』
fn:seconds-from-dateTime	xs:dateTime 値の秒コンポーネントを戻します。	224 ページの『fn:seconds-from-dateTime 関数』
fn:seconds-from-duration	期間の秒コンポーネントを戻します。	224 ページの『fn:seconds-from-duration 関数』
fn:seconds-from-time	xs:time 値の秒コンポーネントを戻します。	225 ページの『fn:seconds-from-time 関数』
fn:sum	シーケンス内の値の合計を戻します。	228 ページの『fn:sum 関数』
fn:timezone-from-date	xs:date 値の時間帯コンポーネントを戻します。	229 ページの『fn:timezone-from-date 関数』
fn:timezone-from-dateTime	xs:dateTime 値の時間帯コンポーネントを戻します。	229 ページの『fn:timezone-from-dateTime 関数』

表 66. 日付、時間、および期間関数 (続き)

機能	説明	参照
fn:timezone-from-time	xs:time 値の時間帯コンポーネントを戻します。	230 ページの『fn:timezone-from-time 関数』
fn:year-from-date	xs:date 値の年コンポーネントを戻します。	234 ページの『fn:year-from-date 関数』
fn:year-from-dateTime	xs:dateTime 値の年コンポーネントを戻します。	234 ページの『fn:year-from-dateTime 関数』
fn:years-from-duration	期間の年コンポーネントを戻します。	235 ページの『fn:years-from-duration 関数』

表 67. シーケンス関数

機能	説明	参照
fn:count	シーケンス内の値の個数を戻します。	199 ページの『fn:count 関数』
fn:data	項目のシーケンスからのアトミック値のシーケンスを戻します。	202 ページの『fn:data 関数』
fn:distinct-values	シーケンス内の特殊値を戻します。	205 ページの『fn:distinct-values 関数』
fn:last	現在処理中の項目のシーケンス内の値の個数を戻します。	209 ページの『fn:last 関数』
fn:position	シーケンス内での、現在処理中のコンテキスト項目の位置を戻します。	221 ページの『fn:position 関数』

表 68. ノード関数

機能	説明	参照
fn:local-name	ノードのローカル名プロパティーを戻します。	210 ページの『fn:local-name 関数』
fn:name	ノード名の接頭部部分およびローカル名部分を戻します。	218 ページの『fn:name 関数』

fn:abs 関数

fn:abs 関数は、数値の絶対値を戻します。

構文

numeric-value

アトミック値または空のシーケンス。

numeric-value がアトミック値である場合、以下のいずれかのタイプを持ちます。

- xs:double
- xs:decimal
- xs:integer
- 上記のいずれかのタイプから派生したタイプ
- xs:untypedAtomic

numeric-value が xs:untypedAtomic タイプである場合、xs:double 値に変換されます。

戻り値

numeric-value が空のシーケンスでない場合、戻り値は、*numeric-value* の絶対値です。

numeric-value が空のシーケンスの場合、fn:abs は空のシーケンスを戻します。

戻り値のデータ・タイプは、*numeric-value* のデータ・タイプによって異なります。

- *numeric-value* が xs:double、xs:decimal、または xs:integer である場合、戻される値は *numeric-value* と同じタイプになります。
- *numeric-value* が xs:double、xs:decimal、または xs:integer から派生したデータ・タイプである場合、戻される値は *numeric-value* の直接の親のデータ・タイプになります。
- *numeric-value* が xs:untypedAtomic タイプである場合、戻される値のタイプは xs:double です。

例

次の関数は、-10.5 の絶対値を戻します。

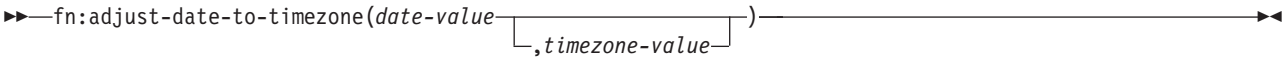
```
fn:abs(-10.5)
```

戻り値は、10.5 です。

fn:adjust-date-to-timezone 関数

fn:adjust-date-to-timezone 関数は、 xs:date 値を特定の時間帯に調整するか、xs:date 値から時間帯コンポーネントを除去します。

構文



date-value

調整対象の日付値。

date-value は xs:date タイプであるか、空のシーケンスです。

timezone-value

date-value が調整されるタイム・ゾーンを表す期間です。

timezone-value は、空シーケンスにするか、 -PT14H から PT14H (両端を含む) の範囲内の xs:dayTimeDuration タイプの単一値にすることができます。この値は分数 (整数) であり、秒のコンポーネントにすることはできません。 *timezone-value* が指定されていない場合、デフォルト値は PT0H で、これは UTC を表します。

戻り値

戻り値は、指定されるパラメーターに応じて、xs:date タイプの値か、空のシーケンスになります。*date-value* が空のシーケンスではない場合、戻り値は xs:date タイプです。 戻される可能性のある戻り値を以下の表に示します。

表 69. *fn:adjust-date-to-timezone* の入力値および戻り値のタイプ

date-value	*timezone-value*	**戻り値**
時間帯コンポーネントを含む *date-value*	明示的な値、または値の指定なし (期間 PT0H)	*timezone-value* によって表されるタイム・ゾーンに調整される *date-value*。
時間帯コンポーネントを含む *date-value*	空のシーケンス	時間帯コンポーネントを含まない *date-value*。
時間帯コンポーネントを含まない *date-value*	明示的な値、または値の指定なし (期間 PT0H)	時間帯コンポーネントを含む *date-value*。時間帯コンポーネントは、*timezone-value* で表される時間帯です。日付コンポーネントは、時間帯に調整されません。
時間帯コンポーネントを含まない *date-value*	空のシーケンス	*date-value*。
空のシーケンス	明示的な値、空のシーケンス、または値の指定なし	空のシーケンス。

date-value を異なるタイム・ゾーンに調整する場合、*date-value* は時間コンポーネント 00:00:00 を持つ dateTime として扱われます。 戻り値には、*timezone-value* で表される時間帯コンポーネントが含まれます。以下の関数は、調整された日付の値を計算します。

```
xs:date(fn:adjust-dateTime-to-timezone(xs:dateTime(date-value),timezone-value))
```

例

以下の例で、変数 `$tz` は、-10 時間の期間であり、`xs:dayTimeDuration("-PT10H")` として定義されます。

以下の関数は、 UTC+1 時間帯の 2009 年 5 月 7 日の日付値を調整します。関数は *timezone-value* -PT10H を指定します。

```
fn:adjust-date-to-timezone(xs:date("2009-05-07+01:00"), $tz)
```

戻される日付値は、2009-05-06-10:00 です。日付は UTC-10 タイム・ゾーンに調整されます。

以下の関数は、時間帯コンポーネントがない 2009 年 3 月 7 日の日付値に時間帯コンポーネントを追加します。関数は *timezone-value* -PT10H を指定します。

```
fn:adjust-date-to-timezone(xs:date("2009-03-07"), $tz)
```

戻り値は 2009-03-07-10:00 です。時間帯コンポーネントが、日付値に追加されます。

以下の関数は、UTC-7 時間帯の 2009 年 2 月 9 日の日付値を調整します。*timezone-value* が指定されていない場合、関数はデフォルトの *timezone-value* PT0H を使用します。

```
fn:adjust-date-to-timezone(xs:date("2009-02-09-07:00"))
```

戻される日付は、 2009-02-09Z です。日付は UTC に調整されます。

以下の関数は、 UTC-7 時間帯の 2009 年 3 月 7 日 の日付値から時間帯コンポーネントを除去します。*timezone-value* は空のシーケンスです。

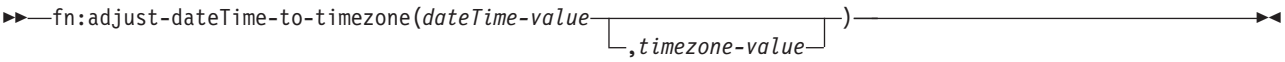
```
fn:adjust-date-to-timezone(xs:date("2009-05-07-07:00"), ())
```

戻り値は 2009-05-07 です。

fn:adjust-dateTime-to-timezone 関数

fn:adjust-dateTime-to-timezone 関数は、 xs:dateTime 値を特定の時間帯に調整するか、xs:dateTime 値から時間帯コンポーネントを除去します。

構文



dateTime-value

調整対象の dateTime 値。

dateTime-value は xs:dateTime タイプであるか、空のシーケンスです。

timezone-value

dateTime-value が調整されるタイム・ゾーンを表す期間です。

timezone-value は、空シーケンスにするか、 -PT14H から PT14H (両端を含む) の範囲内の xs:dayTimeDuration タイプの単一値にすることができます。この値は分数 (整数) であり、秒のコンポーネントにすることはできません。 *timezone-value* が指定されていない場合、デフォルト値は PT0H で、これは UTC を表します。

戻り値

戻り値は、入力値のタイプに応じて xs:dateTime タイプの値か、空のシーケンスになります。 *dateTime-value* が空のシーケンスではない場合、戻り値は xs:dateTime タイプです。 戻される可能性のある戻り値を以下の表に示します。

表 70. fn:adjust-dateTime-to-timezone の入力値および戻り値のタイプ

dateTime-value	*timezone-value*	**戻り値**
時間帯コンポーネントを含む *dateTime-value*	明示的な値、または値の指定なし (期間 PT0H)	*dateTime-value* は、*timezone-value* が表すタイム・ゾーンに調整されます。戻り値には、*timezone-value* で表される時間帯コンポーネントが含まれます。
時間帯コンポーネントを含む *dateTime-value*	空のシーケンス	時間帯コンポーネントを含まない *dateTime-value*。
時間帯コンポーネントを含まない *dateTime-value*	明示的な値、または値の指定なし (期間 PT0H)	時間帯コンポーネントを含む *dateTime-value*。時間帯コンポーネントは、*timezone-value* で表される時間帯です。日付コンポーネントと時間コンポーネントはタイム・ゾーンに調整されません。
時間帯コンポーネントを含まない *dateTime-value*	空のシーケンス	*dateTime-value*。
空のシーケンス	明示的な値、空のシーケンス、または値の指定なし	空のシーケンス。

例

以下の例で、変数 `$tz` は、-10 時間の期間であり、`xs:dayTimeDuration("-PT10H")` として定義されます。

以下の関数は、 UTC-7 時間帯の 2009 年 3 月 7 日 午前 10 時の dateTime 値を -PT10H の *time zone-value* で指定された時間帯に調整します。

```
fn:adjust-dateTime-to-timezone(xs:dateTime("2009-03-07T10:00:00-07:00"), $tz)
```

戻される dateTime 値は、2009-03-07T07:00:00-10:00 です。

以下の関数は、 2009 年 3 月 7 日 午前 10 時の dateTime 値を調整します。*dateTime-value* には時間帯コンポーネントがないため、関数は -PT10H の *timezone-value* を指定します。

```
fn:adjust-dateTime-to-timezone(xs:dateTime("2009-03-07T10:00:00"), $tz)
```

戻される dateTime は、2009-03-07T10:00:00-10:00 です。

以下の関数は、 UTC-7 時間帯の 2009 年 6 月 4 日 午前 10 時 の dateTime 値を調整します。*timezone-value* が指定されない場合、関数は、デフォルトの時間帯値である PT0H を使用します。

```
fn:adjust-dateTime-to-timezone(xs:dateTime("2009-06-04T10:00:00-07:00"))
```

戻される dateTime 値は、2009-06-04T17:00:00Z です。この値は UTC に調整された dateTime 値です。

以下の関数は、 UTC-7 時間帯の 2009 年 3 月 7 日 午前 10 時の dateTime 値から時間帯コンポーネントを除去します。*timezone-value* 値は、空シーケンスです。

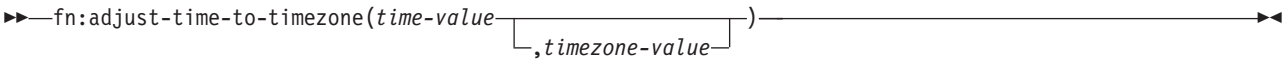
```
fn:adjust-dateTime-to-timezone(xs:dateTime("2009-03-07T10:00:00-07:00"), ())
```

戻される dateTime 値は、2009-03-07T10:00:00 です。

fn:adjust-time-to-timezone 関数

fn:adjust-time-to-timezone 関数は、 xs:time 値を特定の時間帯に調整するか、xs:time 値から時間帯コンポーネントを除去します。

構文



time-value

調整する時間値。

time-value は xs:time タイプであるか、空のシーケンスです。

timezone-value

time-value が調整されるタイム・ゾーンを表す期間です。

timezone-value は、空シーケンスにするか、 -PT14H から PT14H (両端を含む) の範囲内の xs:dayTimeDuration タイプの単一値にすることができます。この値は分数 (整数) であり、秒のコンポーネントにすることはできません。 *timezone-value* が指定されていない場合、デフォルト値は PT0H で、これは UTC を表します。

戻り値

戻り値は、指定されるパラメーターに応じて、xs:time タイプの値か、空のシーケンスになります。*time-value* が空のシーケンスではない場合、戻り値は xs:time タイプです。 戻される可能性のある戻り値を以下の表に示します。

表 71. fn:adjust-time-to-timezone の入力値および戻り値のタイプ

date-value	*timezone-value*	**戻り値**
時間帯コンポーネントを含む *time-value*	明示的な値、または値の指定なし (期間 PT0H)	*timezone-value* によって表されるタイム・ゾーンに調整される *time-value*。戻り値には、*timezone-value* で表される時間帯コンポーネントが含まれます。タイム・ゾーンの調整が夜中の 12 時をまたぐ場合、日付の変更は無視されます。
時間帯コンポーネントを含む *time-value*	空のシーケンス	時間帯コンポーネントを含まない *time-value*。
時間帯コンポーネントを含まない *time-value*	明示的な値、または値の指定なし (期間 PT0H)	時間帯コンポーネントを含む *time-value*。時間帯コンポーネントは、*timezone-value* で表される時間帯です。時刻コンポーネントはタイム・ゾーンに調整されません。
時間帯コンポーネントを含まない *time-value*	空のシーケンス	*time-value*。
空のシーケンス	明示的な値、空のシーケンス、または値の指定なし	空のシーケンス。

例

以下の例で、変数 `$tz` は、-10 時間の期間であり、`xs:dayTimeDuration("-PT10H")` として定義されます。

以下の関数は、 UTC-7 時間帯の午前 10:00 の時刻値を調整し、 -PT10H の *timezone-value* を指定します。

```
fn:adjust-time-to-timezone(xs:time("10:00:00-07:00"), $tz)
```

戻り値は 7:00:00-10:00 です。 時刻は、期間 -PT10H によって表されるタイム・ゾーンに調整されます。

以下の関数は、午後 1:00 の時刻値を調整します。この時刻値には、時間帯コンポーネントは含まれません。

```
fn:adjust-time-to-timezone(xs:time("13:00:00"), $tz)
```

戻り値は 13:00:00-10:00 です。 時間には、-PT10H の期間で表される時間帯コンポーネントが含まれます。

以下の関数は、UTC-7 時間帯の午前 10:00 の時刻値を調整します。関数では、 *timezone-value* を指定せず、デフォルト値の PT0H を使用します。

```
fn:adjust-time-to-timezone(xs:time("10:00:00-07:00"))
```

戻り値は 17:00:00Z であり、時間は UTC に調整されます。

以下の関数は、 UTC-7 時間帯の午前 8:00 の時刻値から時間帯コンポーネントを除去します。*timezone-value* は、空シーケンスです。

```
fn:adjust-time-to-timezone(xs:time("08:00:00-07:00"), ())
```

戻り値は 8:00:00 です。

以下の例は、2 つの時刻を比較しています。 タイム・ゾーンの調整が夜中の 12 時をまたぐ場合、日付の変更が生じます。 しかし、fn:adjust-time-to-timezone では日付の変更は無視します。

```
fn:adjust-time-to-timezone(xs:time("01:00:00+14:00"), $tz)
   = xs:time("01:00:00-10:00")
```

戻り値は true です。

fn:boolean 関数

fn:boolean 関数は、シーケンスの有効なブール値を戻します。

構文

sequence-expression
任意のタイプの項目を含むシーケンス、または空のシーケンス。

戻り値

戻される有効なブール値 (EBV) は、*sequence-expression* の値によって異なります。

表 72. 特定タイプの値に戻される EBV

値の説明	戻される EBV
空のシーケンス	false
1 番目の項目がノードのシーケンス	true
xs:boolean タイプの単一値 (または xs:boolean から派生したタイプの単一値)	false - xs:boolean 値が false の場合
	true - xs:boolean 値が true の場合
タイプ xs:string または xs:untypedAtomic (またはこれらのタイプから派生したタイプ) の単一値	false - 値の長さがゼロの場合
	true - 値の長さがゼロより大きい場合
任意の数値タイプ (または数値タイプから派生したタイプ) の単一値	false - 値が NaN または数的にゼロと等しい場合
	true - 値が数的にゼロと等しくない場合
その他すべての値	エラー
注: 少なくとも 1 個のノードおよび少なくとも 1 個のアトミック値を含むシーケンスの有効なブール値は、順序が予測不能な照会では非決定論的になります。	

例

単一の数値の引数を使用した例: 以下の関数は有効なブール値 0 を戻します。

- `fn:boolean(0)`

戻り値は false です。

fn:compare 関数

fn:compare 関数は、2 つのストリングを比較します。

構文

string-1 および *string-2*
比較される xs:string 値。DB2 は、各文字の数値 Unicode UTF-8 コード値を比較します。この比較は、デフォルトの照合に従って行われます。

戻り値

string-1 および *string-2* が空のシーケンスでない場合、以下のいずれかの xs:integer 値が戻されます。

-1 *string-1* が *string-2* 未満の場合。

0 *string-1* が *string-2* と等しい場合。

1 *string-1* が *string-2* より大きい場合。

2 つのストリングは、各ストリングの対応するバイトを比べることによって比較されます。ストリングの長さが同じでない場合は、短い方のストリングがもう一方のストリングと同じ長さになるように、右側をブランクで埋め込んだ一時コピーが作成され、それを使用して比較されます。

string-1 と *string-2* は、両方とも長さが 0 の場合、または対応するバイトがすべて同じ場合に等しいとみなされます。

string-1 と *string-2* が等しくない場合、その関係 (より大きい値を持つ方) は、ストリングの左端から比較して、等しくないバイトの最初の組の比較によって判別されます。

string-1 が *string-2* より長く、*string-2* のすべてのバイトが *string-1* の先行バイトと等しい場合、*string-1* は *string-2* よりも大きくなります。

string-1 または *string-2* が空のシーケンスである場合、空のシーケンスが戻されます。

例

以下の関数は、デフォルトの照合を使用して 'ABC' と 'ABD' を比較します。

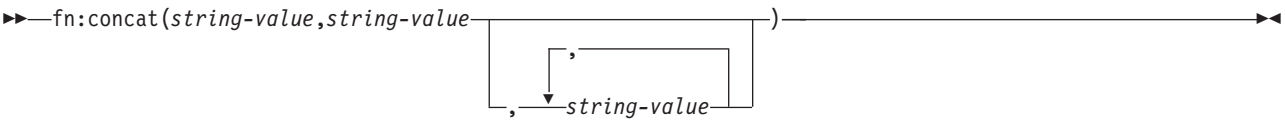
```
fn:compare('ABC', 'ABD')
```

'ABC' は 'ABD' よりも小さいです。 戻り値は -1 です。

fn:concat 関数

fn:concat 関数は、複数のストリングを 1 つのストリングに連結します。

構文



string-value
xs:string 値または空のシーケンス。

戻り値

すべての *string-value* 引数が空のシーケンスである場合、戻り値は空のシーケンスです。それ以外の場合、戻り値は、空のシーケンスではない *string-value* 引数をすべて連結したものである xs:string 値になります。

例

以下の関数は、ストリング 'ABC'、'ABD'、空のシーケンス、および 'ABE' を連結します。

```
fn:concat('ABC', 'ABD', (), 'ABE')
```

戻り値は、'ABCABDABE' です。

fn:contains 関数

fn:contains 関数は、指定されたサブストリングがストリングに含まれているかどうかを判別します。

構文

```
►►—fn:contains(string,substring)———————————————————————————————►◄
```

string *substring* を検索するストリング。

string は、xs:string データ・タイプであるか、空のシーケンスです。 *string* が空のシーケンスの場合、*string* はゼロ長ストリングに設定されます。

substring
string 内を検索するサブストリング。

substring は、xs:string データ・タイプであるか、空のシーケンスです。

戻り値

戻り値は、*string* および *substring* の値によって異なります。

- *string* および *substring* が空のシーケンスでない場合、戻り値は、*substring* が *string* 内にある場合は true になります。 *substring* が *string* に無い場合、戻り値は false になります。
- *string* が空のシーケンスの場合、戻り値は、*substring* が空のシーケンスまたはゼロ長ストリングの場合は true になります。
- *substring* が空のシーケンスまたはゼロ長ストリングである場合、戻り値は true です。

例

以下の関数は、ストリング 'Test literal' にストリング 'lite' が含まれるかどうかを判別します。

```
fn:contains('Test literal','lite')
```

戻り値は true です。

fn:count 関数

fn:count 関数は、シーケンス内の値の個数を戻します。

構文

```
►►—fn:count(sequence-expression)———————————————————————————————►◄
```

sequence-expression
任意のアトミック・タイプの項目を含むシーケンス、または空のシーケンス。

戻り値

sequence-expression が空のシーケンスでない場合、*sequence-expression* 内の値の数である xs:integer 値が戻されます。*sequence-expression* が空のシーケンスの場合、0 が戻されます。

例

以下の関数は 1 を戻します。

```
fn:count(5)
```

以下の関数は、部門 ID が K55 の従業員数を戻します。

```
fn:count(//company/emp[dept/@id="K55"])
```

fn:current-date 関数

fn:current-date 関数は、UTC の暗黙的な時間帯での現在の日付を戻します。

構文

戻り値

戻り値は、現在日付である xs:date 値です。

例

以下の関数は現在日付を戻します。

```
fn:current-date()
```

この関数が 2009 年 12 月 2 日に呼び出された場合、戻り値は 2009-12-02Z です。

fn:current-dateTime 関数

fn:current-dateTime 関数は、UTC の暗黙的な時間帯での現在の日時を戻します。

構文

戻り値

戻り値は、現在の日付と時刻の xs:dateTime 値です。

例

以下の関数は、現在の日付と時刻を戻します。

```
fn:current-dateTime()
```

この関数がトロント (時間帯 -PT5H) で 2009 年 12 月 2 日 6:25 に呼び出された場合、戻り値は 2009-12-02T11:25:30.864001Z になります。

db2-fn:current-local-date 関数

db2-fn:current-local-date 関数は、ローカル時間帯での現在の日付を戻します。

構文

戻り値

戻り値は、現在日付である xs:date 値です。 戻り値には、タイム・ゾーン・コンポーネントは含まれません。

例

以下の関数は現在日付を戻します。

```
db2-fn:current-local-date()
```

この関数がグリニッジ標準時 (GMT) の 2009 年 12 月 2 日 3:00 に呼び出され、ローカル時間帯が東部標準時 (-PT5H) である場合、戻り値は 2009-12-01 になります。

db2-fn:current-local-dateTime 関数

db2-fn:current-local-dateTime 関数は、ローカル時間帯での現在の日時を戻します。

構文

戻り値

戻り値は、現在の日付と時刻の xs:dateTime 値です。 戻り値には、タイム・ゾーン・コンポーネントは含まれません。

例

以下の関数は、現在の日付と時刻を戻します。

```
db2-fn:current-local-dateTime()
```

この関数が 2009 年 12 月 2 日の 6:25 地方時にどこかで呼び出された場合、戻り値は `2009-12-02T06:25:30.864001` になります。

db2-fn:current-local-time 関数

db2-fn:current-local-time 関数は、ローカル時間帯での現在の時刻を戻します。

構文

戻り値

戻り値は、現在時刻の xs:time の値です。 戻り値には、タイム・ゾーン・コンポーネントは含まれません。

例

以下の関数は現在時刻を戻します。

```
db2-fn:current-local-time()
```

この関数がグリニッジ標準時 (GMT) の 6:31 に呼び出され、ローカル時間帯が東部標準時 (-PT5H) である場合、戻り値は `01:31:35.519001` になります。

fn:current-time 関数

fn:current-time 関数は、UTC の暗黙的な時間帯での現在の時刻を戻します。

構文

戻り値

戻り値は、現在時刻の xs:time の値です。

例

以下の関数は現在時刻を戻します。

```
fn:current-time()
```

この関数がグリニッジ標準時 6:31 に呼び出された場合、戻り値は 06:31:35.519001Z です。

fn:data 関数

fn:data 関数は、項目のシーケンスをアトミック値のシーケンスに変換します。

構文

sequence
任意のシーケンス (空のシーケンスを含む)。

戻り値

戻り値は、タイプ xs:anyAtomicType の項目のシーケンスです。シーケンス内の項目によって、以下のようになります。

- 項目がアトミック値の場合、戻り値はその値になります。
- 項目がノードの場合、戻り値はそのノードの型付きの値になります。

例

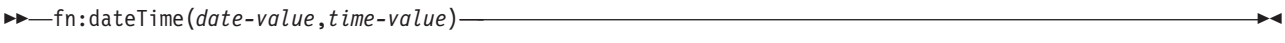
次の関数は、修飾されている名前ノードすべての型付きの値を戻します。修飾されている名前ノードとは、文書内の billTo ノードの子であるすべての名前ノードです。

```
fn:data(//billTo/name)
```

fn:dateTime 関数

fn:dateTime 関数は、 xs:date 値および xs:time 値から xs:dateTime 値を構成します。

構文



date-value
: xs:date 値。

time-value
: xs:time 値。

戻り値

戻り値は *date-value* と等しい日付コンポーネント、および *time-value* と等しい時間コンポーネントを持つ xs:dateTime 値です。 結果の時間帯は、以下のように計算されます。

- どちらの引数にも時間帯がない場合、結果にも時間帯はありません。
- 引数の片方のみに時間帯がある場合、または両方の引数が同じ時間帯の場合、結果はこの時間帯になります。
- 2 つの引数が異なる時間帯の場合、エラーが返されます。

例

以下の関数は、xs:date 値および xs:time 値の xs:dateTime 値を戻します。

```
fn:dateTime((xs:date("2009-04-16")), (xs:time("12:30:59")))
```

戻り値は、 xs:dateTime 値の 2009-04-16T12:30:59 です。

fn:day-from-date 関数

fn:day-from-date 関数は、 xs:date 値の日付コンポーネントをローカル時間帯の形式で戻します。

構文

date-value
: 日付コンポーネントが取り出される日付値。

  date-value は xs:date タイプであるか、空のシーケンスです。

戻り値

date-value は xs:date タイプであり、戻り値は xs:integer タイプであり、値の範囲は 1 から 31 です。 値は *date-value* の日付コンポーネントです。

date-value が空のシーケンスである場合、戻り値は空のシーケンスです。

例

次の関数は、 2009 年 6 月 1 日の日付値の日付コンポーネントを戻します。

```
fn:day-from-date(xs:date("2009-06-01"))
```

戻り値は 1 です。

fn:day-from-dateTime 関数

fn:day-from-dateTime 関数は、ローカル時間帯の形式の xs:dateTime 値の日付コンポーネントを戻します。

構文

dateTime-value
日付コンポーネントが取り出される dateTime 値。

dateTime-value は xs:dateTime タイプであるか、空のシーケンスです。

戻り値

dateTime-value は xs:dateTime タイプであり、戻り値は xs:integer タイプであり、値の範囲は 1 から 31 です。 値は、*dateTime-value* の日付コンポーネントです。

dateTime-value が空のシーケンスである場合、戻り値は空のシーケンスです。

例

次の関数は、 UTC+4 時間帯の 2009 年 1 月 31 日 午後 8:00 の dateTime 値の日付コンポーネントを戻します。

```
fn:day-from-dateTime(xs:dateTime("2009-01-31T20:00:00+04:00"))
```

戻り値は 31 です。

fn:days-from-duration 関数

fn:days-from-duration 関数は、期間の日付コンポーネントを戻します。

構文

duration-value
日付コンポーネントが取り出される期間値。

duration-value は、空のシーケンスか、または xs:dayTimeDuration、xs:duration、xs:yearMonthDuration のいずれかのタイプの値です。

戻り値

戻り値は、*duration-value* のタイプによって異なります。

- *duration-value* が xs:dayTimeDuration タイプまたは xs:duration タイプの場合、戻り値は xs:integer タイプであり、 xs:dayTimeDuration としてキャストされた *duration-value* の日付コンポーネントです。 *duration-value* が負の数である場合、戻り値も負の数です。
- *duration-value* が xs:yearMonthDuration タイプの場合、戻り値は 0 です。
- *duration-value* が空のシーケンスである場合、戻り値は空のシーケンスです。

xs:dayTimeDuration としてキャストされた *duration-value* の日付コンポーネントは、 (*S* idiv 86400) として計算された日数 (整数) です。値 *S* は、年コンポーネントおよび月コンポーネントを除去するために xs:dayTimeDuration としてキャストされた *duration-value* の秒の総数です。

例

この関数では、-10 日と 0 時間の期間の日付コンポーネントを戻します。

```
fn:days-from-duration(xs:dayTimeDuration("-P10DT00H"))
```

戻り値は -10 です。

この関数では、3 日と 55 時間の期間の日付コンポーネントを戻します。

```
fn:days-from-duration(xs:dayTimeDuration("P3DT55H"))
```

戻り値は 5 です。期間の総日数を計算する場合、55 時間は 2 日と 7 時間に変換されます。 この期間は P5D7H と等しく、これは 5 日の日付コンポーネントを持ちます。

fn:distinct-values 関数

fn:distinct-values 関数は、シーケンス内の特殊値を戻します。

構文

sequence-expression
: アトミック値のシーケンスまたは空のシーケンス。シーケンス内の項目は、次の任意のタイプを持つことができます。
  - 数値
  - ストリング
  - 日付、時刻、または期間のタイプ

戻り値

sequence-expression が空のシーケンスでない場合、戻り値は *sequence-expression* 内の特殊値である値を含むシーケンスです。結果シーケンスの項目のタイプは、入力シーケンスのタイプと一致します。2 つの項目は、それらが相互に等しくない場合、区別されます。XPath は、以下の規則を使用して特殊値のシーケンスを取得します。

- 2 つの値を比較できない場合、それらの値は異なるとみなされます。
- タイプ xs:untypedAtomic の値は、xs:string タイプの規則を使用して比較されます。
- 値のシーケンスが戻される順序は、入力順序と同じでない可能性があります。
- 等しいと比較された値のセットの最初の値が戻されます。
- *sequence-expression* が空のシーケンスである場合、空のシーケンスが戻されます。

- xs:double 値の場合、正のゼロは負のゼロに等しくなります。
- *sequence-expression* が複数の NaN 値を含む場合、単一の NaN 値が戻されます。

例

次の例は、ノード b の特殊値を戻します。

```
 <x xmlns="http://posample.org">
      <b>1</b><b>a</b><b>1.0</b><b>A</b><b>1</b>
</x>

declare default element namespace "http://posample.org";
fn:distinct-values($d/x/b)
```

結果は ("1", "a", "1.0", "A") です。

fn:exists 関数

fn:exists 関数では、要素、属性、テキスト・ノード、アトミック値 (例えば整数)、XML 文書など、多種の項目の存在を検査できます。引数として指定された式が空の結果 (空のシーケンス) を生じる場合、fn:exists は false を返します。引数が空のシーケンス以外の何かを返した場合、fn:exists は true を返します。

構文

sequence-expression
任意のタイプの任意のシーケンスまたは空のシーケンス。

戻り値

戻り値は、*sequence-expression* が空のシーケンスでない場合 true です。 *sequence-expression* が空のシーケンスになる場合、戻り値は false です。

例

例 1: phone の子要素を持つ customer 要素があるかどうかを確認します。存在する場合、fn:exists 関数は true を返します。

```
fn:exists($info/customer/phone)
```

例 2: Cid という属性を持つ customer 要素があるかどうかを確認します。存在する場合、fn:exists 関数は true を返します。

```
fn:exists($info/customer/@Cid)
```

例3: comment 要素にテキスト・ノードがあるかどうかを確認します。この例では、comment 要素がテキスト・ノードのない空の要素である場合、fn:exists は false を返します。また、comment 要素がまったくない場合にも、fn:exists は false を返します。

```
fn:exists($info/comment/text())
```

fn:hours-from-dateTime 関数

fn:hours-from-dateTime 関数は、ローカル時間帯形式の xs:dateTime 値の時間コンポーネントを戻します。

構文

dateTime-value
時間コンポーネントが取り出される dateTime 値。

dateTime-value は xs:dateTime タイプであるか、空のシーケンスです。

戻り値

dateTime-value は xs:dateTime タイプであり、戻り値は xs:integer タイプであり、値の範囲は 0 から 23 です。 値は、*dateTime-value* の時間コンポーネントです。

dateTime-value が空のシーケンスである場合、戻り値は空のシーケンスです。

例

次の関数は、UTC-8 時間帯の 2009 年 1 月 31 日午後 2:00 の dateTime 値の時間コンポーネントを戻します。

```
fn:hours-from-dateTime(xs:dateTime("2009-01-31T14:00:00-08:00"))
```

戻り値は 14 です。

fn:hours-from-duration 関数

fn:hours-from-duration 関数は、期間値の時間コンポーネントを戻します。

構文

duration-value
時間コンポーネントが取り出される期間値。

duration-value は空シーケンスであるか、 xs:dayTimeDuration、xs:duration、または xs:yearMonthDuration タイプのいずれかの値です。

戻り値

戻り値は、*duration-value* のタイプによって異なります。

- *duration-value* が xs:dayTimeDuration タイプまたは xs:duration タイプの場合、戻り値は xs:integer タイプであり、 -23 から 23 (両端を含む) の範囲内の値です。値は、xs:dayTimeDuration としてキャストされた *duration-value* の時間コンポーネントです。 *duration-value* が負の数である場合、この値も負の数になります。
- *duration-value* が xs:yearMonthDuration タイプの場合、戻り値は xs:integer タイプの 0 です。
- *duration-value* が空のシーケンスである場合、戻り値は空のシーケンスです。

xs:dayTimeDuration としてキャストされた *duration-value* の時間コンポーネントは、 `((S mod 86400) idiv 3600)` で計算された時間数 (整数) です。値 *S* は、日コンポーネントおよび月コンポーネントを削除するために xs:dayTimeDuration としてキャストされた *duration-value* の秒の総数です。1 日を秒数の値で表すと 86400 であり、1 時間を秒数の値で表すと 3600 です。

例

以下の関数は、期間が 126 時間の時間コンポーネントを戻します。

```
fn:hours-from-duration(xs:dayTimeDuration("PT126H"))
```

戻り値は 6 です。期間の総時間数を計算する場合、126 時間は 5 日と 6 時間に変換されます。 この期間は P5DT6H と等しく、これは 6 時間の時間コンポーネントを持ちます。

fn:hours-from-time function

fn:hours-from-time 関数は、ローカル時間帯形式の xs:time 値の時間コンポーネントを戻します。

構文

time-value
時間コンポーネントが取り出される時刻値。

time-value は xs:time タイプであるか、空のシーケンスです。

戻り値

time-value が空のシーケンスではない場合、戻り値は xs:integer タイプであり、値の範囲は 0 から 23 です。 値は、*time-value* の時間コンポーネントです。

time-value が空のシーケンスである場合、戻り値は空のシーケンスです。

例

次の関数は、UTC-8 時間帯の午前 9:30 の時刻値の時間コンポーネントを戻します。

```
fn:hours-from-time(xs:time("09:30:00-08:00"))
```

戻り値は 9 です。

fn:implicit-timezone 関数

fn:implicit-timezone 関数は、時間帯が付かない日付、時刻、または dateTime の値が比較演算または算術演算で使用される場合の時間帯を戻します。

暗黙的な時間帯の値は、PT0S です。

構文

戻り値

暗黙的な時間帯の戻り値は、 xs:dayTimeDuration タイプです。

例

次の関数は、 xs:dayTimeDuration("PT0S") を戻します。

```
fn:implicit-timezone()
```

fn:last 関数

fn:last 関数は、現在処理中の項目のシーケンス内の値の数を戻します。

構文

戻り値

現在処理中のシーケンスが空のシーケンスでない場合、戻り値はシーケンス内の値の数を示す xs:integer 値です。現在処理中のシーケンスが空のシーケンスである場合、戻り値は空のシーケンスです。

以下の場合、エラーが戻されます。

- fn:last が "/" または "//" によりそのコンテキスト項目から分離されている。

  例えば、以下の式はサポートされません。

  ```
  /a/b/c/fn:last
  /a/b/[c/fn:last=3]
  ```

- コンテキスト・ノードが descendant 軸または descendant-or-self 軸を持っている。

  例えば、以下の式はサポートされません。

  ```
  /a/b/descendant::c[fn:last()=1]
  ```

- コンテキスト・ノードがフィルター式であり、そのフィルター式が descendant 軸または descendant-or-self 軸を使用するステップを持っているか、またはネストされたフィルター式である。

  例えば、以下の式はサポートされません。

  ```
  /a/(b/descendant::c)[fn:last()=1]
  ```

例

サンプル CUSTOMER 表で、カスタマー 1003 の顧客文書は以下のようになります。

```
<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>
```

以下の照会は、文書内の最後の電話番号の xml 値を持つ 1 行を取得します。この照会は、fn:last 関数を使用して電話番号項目の数を判別し、その後 fn:last の結果を使用して最後の電話番号を指します。

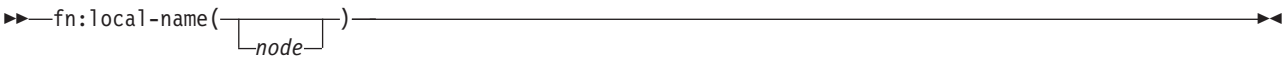
```
SELECT X.* FROM CUSTOMER, XMLTABLE
     ('declare default element namespace "http://posample.org";
          $D/customerinfo/phone[fn:last()]'
       PASSING CUSTOMER.INFO AS "D") X WHERE CID=1003
```

戻された行には、<phone type="cottage">613-555-3278</phone> という値を持つ単一の XML 列があります。

fn:local-name 関数

fn:local-name 関数は、ノードのローカル名・プロパティーを戻します。

構文



ノード ローカル名が取得されるノード。 *node* が指定されていない場合、fn:local-name は現行コンテキスト・ノードについて評価されます。

戻り値

戻り値は xs:string 値です。値は、*node* が指定されているかどうか、および *node* の値によって異なります。

- *node* が指定されていない場合、コンテキスト・ノードのローカル名が戻されます。
- *node* が以下の条件に該当する場合、ゼロ長ストリングが戻されます。
  - *node* が空のシーケンスである。
  - *node* が、文書ノード、コメント、またはテキスト・ノードである。これらのノードは、名前を持っていません。
- 以下の場合、エラーが戻されます。
  - コンテキスト・ノードが未定義である。
  - コンテキスト項目がノードでない。
  - *node* が複数のノードのシーケンスである。
- それ以外の場合、*node* の拡張名のローカル名部分を含む xs:string 値が戻されます。

例

次の例は、ノード b のローカル名を戻します。

```
SELECT * FROM XMLTABLE('fn:local-name($d/x/b)'
                       PASSING XMLPARSE(DOCUMENT
                                '<x><b><c></c></b></x>')
                               AS "d"
                       COLUMNS RESULTNAME VARCHAR(100) PATH 'http://posample.org')  X
```

戻り値は "b" です。

次の例は、引数を持たない fn:localname() がコンテキスト・ノードを戻すことを示しています。

サンプル CUSTOMER 表で、カスタマー 1001 の顧客文書は以下のようになります。

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

次の例は、コンテキスト・ノードのローカル名を戻します。

```
SELECT XV.* FROM CUSTOMER,
       XMLTABLE(XMLNAMESPACES(DEFAULT 'http://posample.org'),
                '$X/customerinfo/*[fn:last()]/fn:local-name()'
                PASSING CUSTOMER.INFO AS "X")
                COLUMNS LOCALNAME CLOB(1K) PATH '.') XV
 WHERE CID=1001
```

戻り値は "phone" です。

db2-fn:local-timezone 関数

db2-fn:local-timezone 関数は、ローカル・システムの時間帯を戻します。

構文

戻り値

戻り値は、協定世界時 (UTC) からのローカル時間帯のオフセットを表す xs:dayTimeDuration 値です。

例

以下の関数はローカル時間帯を戻します。

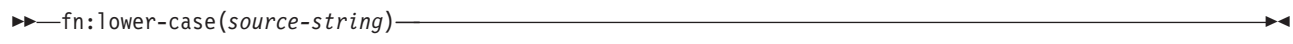
```
db2-fn:local-timezone()
```

東部標準時のローカル時間帯でこの関数を呼び出すと、戻り値は -PT5HI になります。

fn:lower-case 関数

fn:lower-case 関数は、ストリングを小文字に変換します。

構文



source-string
小文字に変換されるストリング。

source-string は、タイプ xs:string か、または空のシーケンスです。

戻り値

source-string が空のシーケンスでない場合、戻り値は、各文字が対応する小文字に変換された *source-string* である xs:string 値です。対応する小文字を持たない各文字は、オリジナルの形式で戻り値に含まれます。

source-string が空のシーケンスである場合、戻り値はゼロ長ストリングです。

例

以下の関数は、ストリング "Wireless Router TB2561" を小文字に変換します。

```
fn:lower-case("Wireless Router TB2561")
```

戻り値は "wireless router tb2561" です。

fn:matches 関数

fn:matches 関数は、ストリングが指定されたパターンに一致するかどうかを判別します。

構文

source-string
: パターンと比較されるストリング。

  source-string はリテラル・ストリングであるか、xs:string 値または空のシーケンスに解決される XPath 式です。

pattern *source-string* と比較する正規表現。正規表現は、検索パターン内の 1 つのストリングまたはストリングのグループを定義する、文字、パターン・マッチング文字、および演算子のセットです。

  pattern はストリング・リテラルです。

flags *pattern* の *source-string* への一致を制御する以下のいずれかの値を含むストリング・リテラル:

  s ドット (.) がすべての文字と一致することを示します。

  s フラグが指定されていない場合、ドット (.) は改行文字 (#x0A) を除くすべての文字と一致します。

  m 脱字記号 (^) が任意の行の先頭 (改行文字の後の位置) に一致し、ドル記号 ($) が任意の行の末尾 (改行文字の前の位置) に一致することを示します。

  m フラグが指定されていない場合、脱字記号 (^) はストリング全体の先頭に一致し、ドル記号 ($) はストリング全体の末尾に一致します。

  i 文字 "a" から "z" および "A" から "Z" について、一致で大/小文字を区別しないことを示します。

  i フラグが指定されていない場合、大/小文字の区別があるマッチングが行われます。

  x *pattern* 内の空白文字 (#x09、#x0A、#x0D、および #x20) が無視されることを示します。ただし、それらが文字クラス内にある場合は除きます。文字クラス内の空白文字が無視されることはありません。

  x フラグが指定されていない場合、空白文字はマッチングに使用されます。

戻り値

source-string が空のシーケンスでない場合、戻り値は、*source-string* が *pattern* に一致する場合に true である xs:boolean 値です。 *source-string* が *pattern* に一致しない場合、戻り値は false です。

マッチングの規則は、以下のとおりです。

- *pattern* がストリング開始または行開始文字である脱字記号 (^)、またはストリング終了または行終了文字であるドル記号 ($) を含まない場合、*source-string* のいずれかのサブストリングが *pattern* に一致すると *source-string* は *pattern* に一致します。
- *pattern* がストリング開始または行開始文字である脱字記号 (^) を含む場合、*source-string* が *source-string* の先頭または *source-string* 内の 1 つの行の先頭から *pattern* に一致する場合のみ、*source-string* は *pattern* に一致します。

- *pattern* がストリング終了または行終了文字であるドル記号 ($) を含む場合、*source-string* が *source-string* の末尾または *source-string* 内の 1 つの行の末尾の *pattern* に一致する場合のみ、*source-string* は *pattern* に一致します。
- m フラグは、以下のことを決定します。
  - 一致が、ストリングの先頭または行の先頭から行われるかどうか
  - 一致が、ストリングの末尾または行の末尾から行われるかどうか

source-string が空のシーケンスである場合、*source-string* はゼロ長ストリングであるとみなされ、*pattern* がゼロ長ストリングに一致する場合に *source-string* が *pattern* に一致します。

例

ストリング内のサブストリングに対するパターンのマッチングの例: 次の関数は、ストリング "ac" または "bd" がストリング "abbcacadbdcd" 内のどこかに現れるかどうかを判別します。

```
fn:matches("abbcacadbdcd","(ac)|(bd)")
```

戻り値は true です。

ストリング全体に対するパターンのマッチングの例: 次の関数は、ストリング "ac" または "bd" がストリング "bd" に一致するかどうかを判別します。脱字記号 (^) 文字およびドル記号 ($) 文字は、ソース・ストリングの先頭で開始しソース・ストリングの末尾で終了する一致でなければならないことを示します。

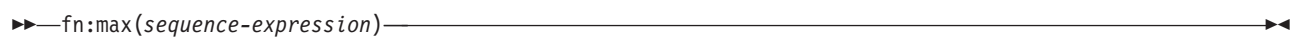
```
fn:matches("bd","^(ac)|(bd)$")
```

戻り値は true です。

fn:max 関数

fn:max 関数は、シーケンス内の最大値を戻します。

構文



sequence-expression

空のシーケンスまたは項目すべてが次のいずれかのタイプであるシーケンスです。

- 数値
- String
- xs:date
- xs:dateTime
- xs:time
- xs:dayTimeDuration
- xs:yearMonthDuration

タイプ xs:untypedAtomic の入力項目は、xs:double にキャストされます。数値入力項目は、タイプ・プロモーションとサブタイプ置換の組み合わせにより比較できる最小共通タイプに変換されます。

戻り値

sequence-expression が空のシーケンスでない場合、戻り値は、*sequence-expression* 内の値のうちで最大である、タイプ xs:anyAtomicType の値です。戻り値のデータ・タイプは、*sequence-expression* 内の項目のデータ・タイプと同じか、または *sequence-expression* 内の項目がプロモートされる共通データ・タイプです。

sequence-expression が 1 つの項目を含む場合、その項目が戻されます。*sequence-expression* が空のシーケンスである場合、空のシーケンスが戻されます。シーケンスが値 NaN を含む場合、NaN が戻されます。

例

以下の照会は、シーケンス (500, 1.0E2, 40.5) の最大値であるタイプ double の値を含む単一行を戻します。

```
SELECT * FROM
 XMLTABLE (XMLNAMESPACES (DEFAULT 'http://posample.org'),
           '$d'
           PASSING XMLPARSE(DOCUMENT '<x xmlns="http://posample.org">
                           <b>500</b><b>1.0E2</b><b>40.5</b></x>') AS "d"
           COLUMNS RES DOUBLE PATH 'fn:max(x/b)') X
```

値は xs:double データ・タイプにプロモートされます。 関数は、5.0E2 という xs:double 値を戻します。この値はここで SQL double データ・タイプに変換されます。

fn:min 関数

fn:min 関数は、シーケンス内の最小値を戻します。

構文

sequence-expression

空のシーケンスまたは項目すべてが次のいずれかのタイプであるシーケンスです。

- 数値
- String
- xs:date
- xs:dateTime
- xs:time
- xs:dayTimeDuration
- xs:yearMonthDuration

タイプ xs:untypedAtomic の入力項目は、xs:double にキャストされます。数値入力項目は、タイプ・プロモーションとサブタイプ置換の組み合わせにより比較できる最小共通タイプに変換されます。

戻り値

sequence-expression が空のシーケンスでない場合、戻り値は *sequence-expression* 内の最小値であるタイプ xs:anyAtomicType の値です。戻り値のデータ・タイプは、*sequence-expression* 内の項目のデータ・タイプと同じか、または *sequence-expression* 内の項目がプロモートされる共通データ・タイプです。

sequence-expression が 1 つの項目を含む場合、その項目が戻されます。*sequence-expression* が空のシーケンスである場合、空のシーケンスが戻されます。シーケンスが値 NaN を含む場合、NaN が戻されます。

例

次の照会は、シーケンス (500, 1.0E2, 40.5) の最小値を戻します。

```
SELECT * FROM
 XMLTABLE (XMLNAMESPACES (DEFAULT 'http://posample.org'),
           '$d'
           PASSING XMLPARSE(DOCUMENT '<x xmlns="http://posample.org">
                             <b>500</b><b>1.0E2</b><b>40.5</b></x>') AS "d"
           COLUMNS RES DOUBLE PATH 'fn:min(x/b)') X
```

値は xs:double タイプにプロモートされます。関数は、4.05E1 という xs:double 値を戻します。この値はここで SQL double データ・タイプに変換されます。

fn:minutes-from-dateTime 関数

fn:minutes-from-dateTime 関数は、ローカル時間帯形式の xs:dateTime 値の分コンポーネントを戻します。

構文

dateTime-value
分コンポーネントが取り出される dateTime 値。

dateTime-value は xs:dateTime タイプであるか、空のシーケンスです。

戻り値

dateTime-value は xs:dateTime タイプであり、戻り値は xs:integer タイプであり、値の範囲は 0 から 59 です。 値は、*dateTime-value* の分コンポーネントです。

dateTime-value が空のシーケンスである場合、戻り値は空のシーケンスです。

例

次の関数は、UTC-8 時間帯の 2009 年 1 月 23 日午前 9:42 の dateTime 値の分コンポーネントを戻します。

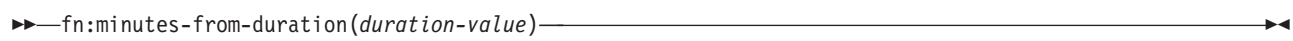
```
fn:minutes-from-dateTime(xs:dateTime("2009-01-23T09:42:00-08:00"))
```

戻り値は 42 です。

fn:minutes-from-duration 関数

fn:minutes-from-duration 関数は、期間の分コンポーネントを戻します。

構文



duration-value
分コンポーネントが取り出される期間値。

duration-value は、空のシーケンスか、または xs:dayTimeDuration、xs:duration、xs:yearMonthDuration のいずれかのタイプの値です。

戻り値

戻り値は、*duration-value* のタイプによって異なります。

- *duration-value* が xs:dayTimeDuration タイプまたは xs:duration タイプの場合、戻り値は xs:integer タイプであり、 -59 から 59 (両端を含む) の範囲内の値です。値は、xs:dayTimeDuration としてキャストされた *duration-value* の分コンポーネントです。 *duration-value* が負の数である場合、この値も負の数になります。
- *duration-value* が xs:yearMonthDuration タイプの場合、戻り値は 0 です。
- *duration-value* が空のシーケンスである場合、戻り値は空のシーケンスです。

xs:dayTimeDuration としてキャストされた *duration-value* の分コンポーネントは、 `((S mod 3600) idiv 60)` として計算された分数 (整数) です。値 *S* は、年コンポーネントおよび月コンポーネントを除去するために xs:dayTimeDuration としてキャストされた *duration-value* の秒の総数です。

例

以下の関数は、期間が 2 日と 16 時間 93 分の分コンポーネントを戻します。

```
fn:minutes-from-duration(xs:dayTimeDuration("P2DT16H93M"))
```

戻り値は 33 です。 この期間の総分数を計算する場合、93 分は 1 時間と 33 分に変換されます。この期間は分コンポーネント 33 分を持つ、P2DT17H33M と等しいと言えます。

fn:minutes-from-time 関数

fn:minutes-from-time 関数は、ローカル時間帯形式の xs:time 値の分コンポーネントを戻します。

構文

time-value

分コンポーネントが取り出される時刻値。

time-value は xs:time タイプであるか、空のシーケンスです。

戻り値

time-value は xs:time タイプであり、戻り値は xs:integer タイプであり、値の範囲は 0 から 59 です。 値は、*time-value* の分コンポーネントです。

time-value が空のシーケンスである場合、戻り値は空のシーケンスです。

例

次の関数は、 UTC-8 時間帯の午前 8:59 の時刻値の分コンポーネントを戻します。

```
fn:minutes-from-time(xs:time("08:59:00-08:00"))
```

戻り値は 59 です。

fn:month-from-date 関数

fn:month-from-date 関数は、ローカル時間帯の形式の xs:date 値の月コンポーネントを戻します。

構文

date-value
月コンポーネントが取り出される日付値。

date-value は xs:date タイプであるか、空のシーケンスです。

戻り値

date-value は xs:date タイプであり、戻り値は xs:integer タイプであり、値の範囲は 1 から 12 です。 値は *date-value* の月コンポーネントです。

date-value が空のシーケンスである場合、戻り値は空のシーケンスです。

例

次の関数は、 2009 年 12 月 1 日の日付値の月コンポーネントを戻します。

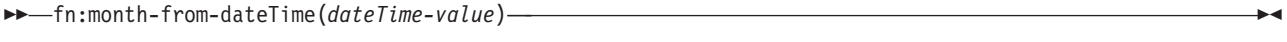
```
fn:month-from-date(xs:date("2009-12-01"))
```

戻り値は 12 です。

fn:month-from-dateTime 関数

fn:month-from-dateTime 関数は、ローカル時間帯の形式の xs:dateTime 値の月コンポーネントを戻します。

構文



dateTime-value
月コンポーネントが取り出される dateTime 値。

dateTime-value は xs:dateTime タイプであるか、空のシーケンスです。

戻り値

dateTime-value は xs:dateTime タイプであり、戻り値は xs:integer タイプであり、値の範囲は 1 から 12 です。 値は、*dateTime-value* の月コンポーネントです。

dateTime-value が空のシーケンスである場合、戻り値は空のシーケンスです。

例

次の関数は、 UTC-8 時間帯の 2009 年 10 月 31 日午前 8:15 の dateTime 値の月コンポーネントを戻します。

```
fn:month-from-dateTime(xs:dateTime("2009-10-31T08:15:00-08:00"))
```

戻り値は 10 です。

fn:months-from-duration 関数

fn:months-from-duration 関数は、期間値の月コンポーネントを戻します。

構文

duration-value
月コンポーネントが取り出される期間値。

duration-value は、空のシーケンスか、または xs:dayTimeDuration、xs:duration、xs:yearMonthDuration のいずれかのタイプの値です。

戻り値

戻り値は、*duration-value* のタイプによって異なります。

- *duration-value* が xs:duration タイプまたは xs:yearMonthDuration タイプの場合、戻り値は xs:integer タイプであり、 -11 から 11 (両端を含む) の範囲内の値です。値は、xs:yearMonthDuration としてキャストされた *duration-value* の月コンポーネントです。 *duration-value* が負の数である場合、この値も負の数になります。
- *duration-value* が xs:dayTimeDuration タイプの場合、戻り値は 0 です。
- *duration-value* が空のシーケンスである場合、戻り値は空のシーケンスです。

xs:yearMonthDuration としてキャストされた *duration-value* の月コンポーネントは、 *duration-value* の月の総数を 12 で除算して余った月数 (整数) です。

例

以下の関数は、期間が 20 年と 5 カ月の月コンポーネントを戻します。

```
fn:months-from-duration(xs:duration("P20Y5M"))
```

戻り値は 5 です。

以下の関数は、yearMonthDuration が -9 年と -13 カ月の月コンポーネントを戻します。

```
fn:months-from-duration(xs:yearMonthDuration("-P9Y13M"))
```

戻り値は -1 です。 期間の総月数を計算する場合、-13 カ月は -1 年と -1 カ月に変換されます。この期間は -P10Y1M と等しく、これは -1 カ月の月コンポーネントを持ちます。

以下の関数は、期間が 14 年 11 カ月 40 日 13 時間の月コンポーネントを戻します。

```
xquery fn:months-from-duration(xs:duration("P14Y11M40DT13H"))
```

戻り値は 11 です。

fn:name 関数

fn:name 関数は、ノード名の接頭部とローカル名の部分を戻します。

構文

ノード 名前が取得されるノードの修飾名。 *node* が指定されていない場合、fn:name は現行コンテキスト・ノードについて評価されます。

戻り値

戻り値は xs:string 値です。値は、*node* の値によって異なります。

- *node* が以下の条件に該当する場合、ゼロ長ストリングが戻されます。
  - *node* が空のシーケンスである。
  - *node* が、文書ノード、コメント、またはテキスト・ノードである。これらのノードは、名前を持っていません。
- 以下の場合、エラーが戻されます。
  - コンテキスト・ノードが未定義である。
  - コンテキスト項目がノードでない。
  - *node* が複数のノードのシーケンスである。
- それ以外の場合、*node* の接頭部 (存在する場合) とローカル名を含む xs:string 値が戻されます。

例

以下の例では、各 b ノードの修飾名を含む CLOB 列を持つ 1 つの行が戻されます。

```
SELECT * FROM
 XMLTABLE (  'declare namespace ns1="http://posample.org";
      $d/x/ns1:b/fn:name(.)'
      PASSING XMLPARSE(DOCUMENT
                      '<x xmlns:n="http://posample.org">
                        <n:b><n:c></n:c></n:b></x>')
               AS "d")
      COLUMNS COL1 CLOB(1K) PATH(.) ) X
```

戻り値は "n:b" です。

次の例は、引数を持たない fn:name() がコンテキスト・ノードを戻すことを示しています。

サンプル CUSTOMER 表で、カスタマー 1001 の顧客文書は以下のようになります。

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

次の例は、コンテキスト・ノードの修飾名を戻します。

```
SELECT * FROM CUSTOMER,
 XMLTABLE('declare default element namespace "http://posample.org";
       $X/customerinfo/phone/fn:name()'
       PASSING INFO AS "X")
       COLUMNS RESULT1 CLOB(1K) PATH '.') X
 WHERE CID=1001
```

戻り値は "phone" です。

fn:normalize-space 関数

fn:normalize-space 関数は、ストリングから前後の空白文字を除去し、ストリング内の複数の連続した空白文字を単一のブランク文字に置き換えます。

構文

source-string

空白文字が正規化されるストリング。

source-string は xs:string 値または空のシーケンスです。

source-string が指定されていない場合、fn:normalize-space の引数は、fn:string 関数を使用して xs:string 値に変換される現行のコンテキスト・アイテムです。

戻り値

戻り値は、以下の操作が *source-string* に対して実行された結果の xs:string 値です。

- 前後の空白文字を削除する。
- 1 つ以上の連続する空白文字の各内部シーケンスが、シングル・スペース (U+0020) 文字に置換される。

空白文字とは、スペース文字 (U+0020)、復帰 (U+000D)、改行 (U+000A)、またはタブ (U+0009) のことです。

source-string が空のシーケンスである場合、ゼロ長ストリングが戻されます。

例

以下の関数は、余分な空白文字をストリング "a b c d " から削除します。

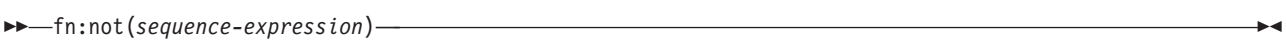
```
fn:normalize-space(" a b c d " )
```

戻り値は "a b c d" です。

fn:not 関数

fn:not 関数は、シーケンス式の有効なブール値が true の場合に false を戻します。シーケンス式の有効なブール値が false の場合、fn:not は true を戻します。

構文



sequence-expression
任意のタイプの項目を含むシーケンス、または空のシーケンス。

戻り値

戻り値は xs:boolean 値です。*sequence-expression* の有効なブール値が false の場合、この関数は true を戻します。*sequence-expression* の有効なブール値が true の場合、この関数は false を戻します。

例

以下の関数は true を戻します。

```
fn:not("a"="b")
```

以下の関数は false を戻します。

```
fn:not("false")
```

fn:position 関数

fn:position 関数は、現在処理中のシーケンスにおけるコンテキスト項目の位置を戻します。

position 関数は、通常は述部で使用されます。ただし、それを使用して、そのコンテキスト項目の各オカレンスの位置を生成することもできます。

構文

戻り値

戻り値は、現在処理中のシーケンス内のコンテキスト・アイテムの位置を示す xs:integer 値です。 シーケンス内の最初の項目の位置は 1 です。コンテキスト項目が未定義の場合は、エラーが戻されます。position 関数は、コンテキスト項目を含むシーケンスの順序が決まっている場合にのみ、決まった結果を戻します。

以下の場合、エラーが戻されます。

- コンテキスト・ステップが descendant または descendant-or-self 軸である。
- コンテキストがアトミック値のシーケンスである。
- fn:position が述部内のネストされたフィルター式の一部として使用されている。I

  ```
  $x/a[$y/c/fn:position()] $x/a[(b/c)[2]]
  ```

例

以下の照会は、文書内の <c> 要素のシーケンスにある 2 番目の要素を持つ XML 列を含む 1 つの行を戻します。

```
<x xmlns="http://posample.org"><b><c>x</c><c>y</c><c>z</c></b></x>

SELECT * FROM
     XMLTABLE(XMLNAMESPACES(DEFAULT 'http://posample.org'),
              '$d/x/b/c[fn:position()=2]'
              PASSING XMLPARSE(DOCUMENT '<x xmlns="http://posample.org">
                                         <b><c>x</c><c>y</c><c>z</c></b></x>') AS "d"
              COLUMNS RESULT XML PATH '.') X
```

戻り値は <c>y</c> です。

以下の照会は、<a><b><c> の各オカレンスの位置を単一の XML 値として戻します。

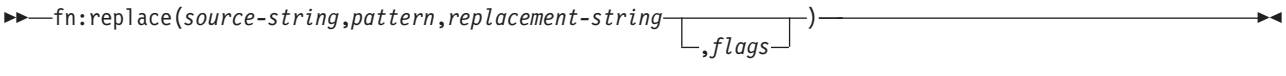
```
SELECT * FROM
     XMLTABLE('.'
              PASSING XMLPARSE(DOCUMENT
                       '<a><b><c>c1</c></b><b><c>c2</c><c>c3</c></b></a>')
              COLUMNS RESULT_POS XML PATH '/a/b/c/fn:position()') X
```

戻り値 "1 2 3" です。

fn:replace 関数

fn:replace 関数は、ストリング内のそれぞれの文字のセットを指定されたパターンと比較し、パターンに一致する文字列を別の文字のセットで置換します。

構文



source-string
置換対象の文字を含むストリング。

source-string はリテラル・ストリングであるか、xs:string 値または空のシーケンスに解決される Xpath 式です。

pattern *source-string* と比較する正規表現。正規表現は、検索パターン内の 1 つのストリングまたはストリングのグループを定義する、文字、パターン・マッチング文字、および演算子のセットです。

pattern はストリング・リテラルです。

replacement-string
source-string 内で *pattern* に一致する文字列を置換する文字列を含むストリング。

replacement-string は xs:string 値です。リテラル $ 記号は ¥$ と記述する必要があります。リテラル ¥ 記号は ¥¥ と記述する必要があります。

flags *pattern* の *source-string* への一致を制御する以下のいずれかの値を含むストリング・リテラル:

s ドット (.) がすべての文字と一致することを示します。

s フラグが指定されていない場合、ドット (.) は改行文字 (#x0A) を除くすべての文字と一致します。

m 脱字記号 (^) が任意の行の先頭 (改行文字の後の位置) に一致し、ドル記号 ($) が任意の行の末尾 (改行文字の前の位置) に一致することを示します。

m フラグが指定されていない場合、脱字記号 (^) はストリング全体の先頭に一致し、ドル記号 ($) はストリング全体の末尾に一致します。

i 文字 "a" から "z" および "A" から "Z" について、一致で大/小文字を区別しないことを示します。

i フラグが指定されていない場合、大/小文字の区別があるマッチングが行われます。

x *pattern* 内の空白文字 (#x09、#x0A、#x0D、および #x20) が無視されることを示します。ただし、それらが文字クラス内にある場合は除きます。文字クラス内の空白文字が無視されることはありません。

x フラグが指定されていない場合、空白文字はマッチングに使用されます。

戻り値

source-string が空のシーケンスでない場合、戻り値は、*source-string* のコピーに対して以下の操作を実行した結果の xs:string 値です。

- *source-string* で *pattern* に一致する文字列が検索されます。
  - *source-string* の 2 つのオーバーラップするサブストリングが *pattern* に一致する場合、先頭文字が *source-string* 内で最初に現れるサブストリングのみが、*pattern* に一致するとみなされます。
  - *pattern* が複数の代替文字セットを含んでおり、それらの代替文字セットが *source-string* 内の同じ位置で始まる文字列に一致する場合、*source-string* 内の文字列に一致する *pattern* 内の最初の文字のセットのみが、*pattern* に一致するとみなされます。
- *pattern* に一致する *source-string* 内のそれぞれの文字のセットが、*replacement-string* に置換されます。

pattern が *source-string* に検出されない場合、*source-string* が戻されます。

pattern がゼロ長ストリングに一致する場合、エラーが戻されます。

source-string が空のシーケンスである場合、ゼロ長ストリングが戻されます。

例

次の関数は、"a" に単一文字が続くか "b" に単一文字が続くインスタンスをすべて "*@" で置換します。

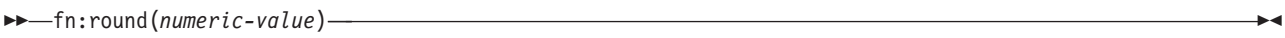
```
fn:replace("abbcacadbdcd","(a(.))|(b(.))","*@")
```

戻り値は "*@*@*@*@*@cd" です。

fn:round 関数

fn:round 関数は、指定された数値に最も近い整数を戻します。

構文



numeric-value

アトミック値または空のシーケンス。

numeric-value がアトミック値である場合、以下のいずれかのタイプを持ちます。

- xs:double
- xs:decimal
- xs:integer
- 上記のいずれかのタイプから派生したタイプ

戻り値

numeric-value が空のシーケンスでない場合、戻り値は *numeric-value* に最も近い整数です。 戻り値のデータ・タイプは、*numeric-value* のデータ・タイプによって異なります。

- *numeric-value* が xs:double、xs:decimal、または xs:integer である場合、戻される値は *numeric-value* と同じタイプになります。
- *numeric-value* が xs:double、xs:decimal、または xs:integer から派生したデータ・タイプである場合、戻される値は *numeric-value* の直接の親のデータ・タイプになります。

numeric-value が空のシーケンスである場合、戻り値は空のシーケンスです。

例

正の引数を使用した例: 以下の関数は、0.5 の丸め値を戻します。

```
fn:round(0.5)
```

戻り値は 1 です。

負の引数を使用した例: 以下の関数は、(-1.5) の丸め値を戻します。

```
fn:round(-1.5)
```

戻り値は -1 です。

fn:seconds-from-dateTime 関数

fn:seconds-from-dateTime 関数は、ローカル時間帯形式の xs:dateTime 値の秒コンポーネントを戻します。

構文

dateTime-value

秒コンポーネントが取り出される dateTime 値。

dateTime-value は xs:dateTime タイプであるか、空のシーケンスです。

戻り値

dateTime-value は xs:dateTime タイプであり、戻り値は xs:decimal タイプであり、値は 0 以上、60 未満です。 値は、*dateTime-value* の秒コンポーネント (小数部を含む) です。

dateTime-value が空のシーケンスである場合、戻り値は空のシーケンスです。

例

次の関数は、UTC-8 時間帯の 2009 年 2 月 8 日午後 2:16:23 の dateTime 値の秒コンポーネントを戻します。

```
fn:seconds-from-dateTime(xs:dateTime("2009-02-08T14:16:23-08:00"))
```

戻り値は 23 です。

次の関数は、UTC 時間帯の 2009 年 6 月 23 日 午前 9:16:20.43 の dateTime 値の秒コンポーネントを戻します。

```
fn:seconds-from-dateTime(xs:dateTime("2009-06-23T09:16:20.43Z"))
```

戻り値は 20.43 です。

fn:seconds-from-duration 関数

fn:seconds-from-duration 関数は、期間の秒コンポーネントを戻します。

構文

duration-value
秒コンポーネントが取り出される期間値。

duration-value は、空のシーケンスか、または xs:dayTimeDuration、xs:duration、xs:yearMonthDuration のいずれかのタイプの値です。

戻り値

戻り値は、*duration-value* のタイプによって異なります。

- *duration-value* が xs:dayTimeDuration タイプまたは xs:duration タイプの場合、戻り値は xs:decimal タイプであり、 -60 より大きく 60 未満の値です。値は、 xs:dayTimeDuration としてキャストされた *duration-value* の秒コンポーネント (小数部分を含む) です。*duration-value* が負の数である場合、この値も負の数になります。
- *duration-value* が xs:yearMonthDuration タイプの場合、戻り値は xs:integer タイプの 0 です。
- *duration-value* が空のシーケンスである場合、戻り値は空のシーケンスです。

xs:dayTimeDuration としてキャストされた *duration-value* の秒コンポーネント (小数部分を含む) は、(*S* mod 60) として計算されます。値 *S* は、年コンポーネントおよび月コンポーネントを除去するために xs:dayTimeDuration としてキャストされた *duration-value* の秒の総数 (小数部分を含む) です。

例

以下の関数は、期間が 150.5 秒の期間の秒コンポーネントを戻します。

```
fn:seconds-from-duration(xs:dayTimeDuration("PT150.5S"))
```

戻り値は 30.5 です。 期間の総秒数を計算する場合、150.5 秒は 2 分と 30.5 秒に変換されます。 この期間は PT2M30.5S と等しく、30.5 秒の秒コンポーネントを持ちます。

fn:seconds-from-time 関数

fn:seconds-from-time 関数は、ローカル時間帯形式の xs:time 値の秒コンポーネントを戻します。

構文

time-value
秒コンポーネントが取り出される時刻値。

time-value は xs:time タイプであるか、空のシーケンスです。

戻り値

time-value は xs:time タイプであり、戻り値は xs:decimal タイプであり、値は 0 以上、60 未満です。 値は、*time-value* の秒コンポーネント (小数部を含む) です。

time-value が空のシーケンスである場合、戻り値は空のシーケンスです。

例

次の関数は、 UTC-8 時間帯の午前 08:59:59 の時刻値の秒コンポーネントを戻します。

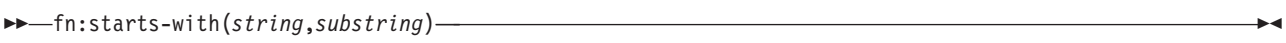
```
fn:seconds-from-time(xs:time("08:59:59-08:00"))
```

戻り値は 59 です。

fn:starts-with 関数

fn:starts-with 関数は、ストリングが指定されたサブストリングで開始しているかどうかを判別します。サブストリングは、デフォルトの照合を使用して突き合わされます。

構文



string *substring* を検索するストリング。

string は、xs:string データ・タイプであるか、空のシーケンスです。*string* が空のシーケンスの場合、*string* はゼロ長ストリングに設定されます。

substring
検索するサブストリング。

substring は、xs:string データ・タイプであるか、空のシーケンスです。

戻り値

戻り値は、次のいずれかの条件が満たされれば、xs:boolean 値 `true` です。

- *substring* が *string* の始めにある。
- *substring* が空のシーケンスまたはゼロ長ストリングである。

それ以外の場合、戻り値は `false` です。

例

以下の関数は、ストリング 'Test literal' がストリング 'lite' で始まっているかどうかを判別します。

```
fn:starts-with('Test literal','lite')
```

戻り値は `false` です。

fn:string 関数

fn:string 関数は、値のストリング表記を戻します。

構文

value ストリングとして表される値。

value は、ノードまたはアトミック値、あるいは空のシーケンスです。

value が指定されていない場合、fn:string は、現行コンテキストの項目について評価されます。現行コンテキストの項目が未定義の場合は、エラーが戻されます。

戻り値

value が空のシーケンスでない場合、xs:string 値が戻されます。

- *value* がノードの場合、戻り値は *value* ノードのストリング値プロパティーです。
- *value* がアトミック値の場合、戻り値は *value* を xs:string タイプにキャストした結果です。

value が空のシーケンスの場合、結果はゼロ長ストリングです。

例

以下の関数は、123 のストリング表記を戻します。

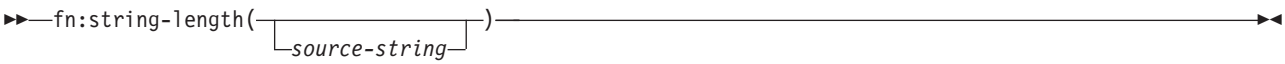
```
fn:string(xs:integer(123))
```

戻り値は '123' です。

fn:string-length 関数

fn:string-length 関数は、ストリングの長さを戻します。

構文



source-string
: その長さが戻されるストリング。

  source-string は、xs:string データ・タイプであるか、空のシーケンスです。

戻り値

source-string が空のシーケンスでない場合、戻り値は、*source-string* の文字数である xs:integer 値です。

source-string が空のシーケンスである場合、戻り値は xs:integer 値 0 です。

source-string が指定されていない場合、 fn:string-length の引数は、デフォルトで、コンテキスト項目のストリング値になります。コンテキスト項目が定義されていない場合は、エラーが発生します。

例

以下の関数は、ストリング "Test literal" の長さを戻します。

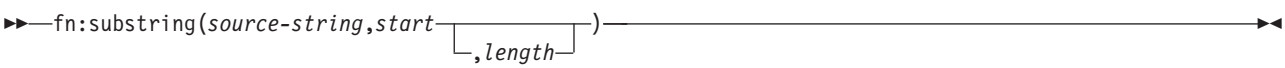
```
fn:string-length("Test literal")
```

戻り値は 12 です。

fn:substring 関数

fn:substring 関数は、ストリングのサブストリングを戻します。

構文



source-string
サブストリングが取得されるストリング。

source-string は、xs:string データ・タイプであるか、空のシーケンスです。

start サブストリングの *source-string* の開始位置。*source-string* の最初の位置は 1 です。 *start*<=0 の場合、*start* は 1 に設定されます。

start は xs:double データ・タイプです。

length サブストリングの長さ。*length* のデフォルトは、*source-string* の長さです。 *start*+*length*-1 が *source-string* の長さよりも大きい場合、*length* は (*source-string* の長さ)-*start*+1 に設定されます。

length は xs:double データ・タイプです。

戻り値

source-string が空のシーケンスでない場合、戻り値は、位置 *start* で始まり、*length* バイトの *source-string* のサブストリングである xs:string 値です。 *source-string* が空のシーケンスである場合、結果はゼロ長ストリングです。

例

以下の関数は、ストリング 'Test literal' の 6 番目の文字で開始する 7 文字を戻します。

```
fn:substring('Test literal',6,7)
```

戻り値は 'literal' です。

fn:sum 関数

fn:sum 関数は、シーケンス内の値の合計を戻します。

構文

sequence-expression
以下のアトミック・タイプのいずれかの項目を含むシーケンス、または空のシーケンス。

- xs:double
- xs:decimal
- xs:integer
- 上記のいずれかのタイプから派生したタイプ
- xs:dayTimeDuration
- xs:yearMonthDuration
- xs:untypedAtomic

シーケンス内のすべての値は、同じタイプまたはそのタイプから派生したタイプでなければなりません。また、単一の共通タイプにプロモートできなければなりません。 xs:untypedAtomic 値は、xs:double データ・タイプにプロモートされます。派生したタイプは、その直接の親データ・タイプにプロモートされます。

戻り値

sequence-expression が空のシーケンスでない場合、戻り値は *sequence-expression* 内の値の合計です。 戻り値のデータ・タイプは、*sequence-expression* の項目のデータ・タイプ、または *sequence-expression* の項目がプロモートされるデータ・タイプと同じです。

sequence-expression が空のシーケンスの場合、fn:sum は 0.0E0 を戻します。

例

以下の関数は、シーケンス (500, 1.0E2, 40.5) の合計を戻します。

```
fn:sum((500, 1.0E2, 40.5))
```

値は xs:double データ・タイプにプロモートされます。 戻り値は 6.405E2 です。

fn:timezone-from-date 関数

fn:timezone-from-date 関数は、 xs:date 値の時間帯コンポーネントを戻します。

構文

date-value
時間帯コンポーネントの抽出元の日付値。

date-value は xs:date タイプであるか、空のシーケンスです。

戻り値

date-value が xs:date タイプであり、明示的な時間帯コンポーネントが付いている場合、戻り値は xs:dayTimeDuration です。値は、-PT14H 時間から PT14H 時間 (両端を含む) の範囲内です。値は、*date-value* 時間帯コンポーネントの UTC 時間帯からの偏差です。

date-value に明示的な時間帯コンポーネントが付いていないか、これが空シーケンスの場合、戻り値は空シーケンスです。

例

次の関数は、 UTC-8 時間帯の 2009 年 3 月 13 日の日付値の時間帯コンポーネントを戻します。

```
fn:timezone-from-date(xs:date("2009-03-13-08:00"))
```

戻り値は -PT8H です。

fn:timezone-from-dateTime 関数

fn:timezone-from-dateTime 関数は、 xs:dateTime 値の時間帯コンポーネントを戻します。

構文

dateTime-value
時間帯コンポーネントの抽出元の dateTime 値。

dateTime-value は xs:dateTime タイプであるか、空のシーケンスです。

戻り値

dateTime-value が xs:dateTime タイプであり、明示的な時間帯コンポーネントが付いている場合、戻り値は xs:dayTimeDuration タイプで、-PT14H から PT14H (両端を含む) の範囲内の値です。値は、*dateTime-value* 時間帯コンポーネントの UTC 時間帯からの偏差です。

dateTime-value に明示的な時間帯コンポーネントが付いていないか、これが空シーケンスである場合、戻り値は空シーケンスです。

例

次の関数は、 UTC-8 時間帯の 2009 年 10 月 30 日午前 7:30 の dateTime 値の時間帯コンポーネントを戻します。

```
fn:timezone-from-dateTime(xs:dateTime("2009-10-30T07:30:00-08:00"))
```

戻り値は -PT8H です。

次の関数は、UTC+10:30 時間帯の 2009 年 1 月 1 日午後 2:30 の dateTime 値の時間帯コンポーネントを戻します。

```
fn:timezone-from-dateTime(xs:dateTime("2009-01-01T14:30:00+10:30"))
```

戻り値は PT10H30M です。

fn:timezone-from-time 関数

fn:timezone-from-time 関数は、 xs:time 値の時間帯コンポーネントを戻します。

構文

time-value
時間帯コンポーネントの抽出元の時刻値。

time-value は xs:time タイプであるか、空のシーケンスです。

戻り値

time-value が xs:time タイプであり、明示的な時間帯コンポーネントが付いている場合、戻り値は xs:dayTimeDuration タイプで、-PT14H から PT14H (両端を含む) の範囲内の値です。値は、*time-value* 時間帯コンポーネントの UTC 時間帯からの偏差です。

time-value に明示的な時間帯コンポーネントが付いていないか、これが空シーケンスである場合、戻り値は空シーケンスです。

例

次の関数は、 UTC-5 時間帯の正午 12 時の時刻値の時間帯コンポーネントを戻します。

```
fn:timezone-from-time(xs:time("12:00:00-05:00"))
```

戻り値は -PT5H です。

次の関数で、午後 1:00 の時刻値には時間帯コンポーネントが付いていません。

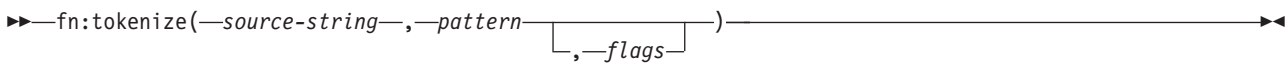
```
fn:timezone-from-time(xs:time("13:00:00"))
```

戻り値は空のシーケンスです。

fn:tokenize 関数

fn:tokenize 関数は、ストリングをサブストリングのシーケンスに分割します。

構文



source-string
サブストリングのシーケンスに分割されるストリング。

source-string はリテラル・ストリングであるか、xs:string 値または空のシーケンスに解決される XPath 式です。

pattern *source-string* 内のサブストリング間の区切り文字。

pattern は、正規表現を含むストリング・リテラルです。正規表現は、検索パターン内の 1 つのストリングまたはストリングのグループを定義する、文字、パターン・マッチング文字、および演算子のセットです。

flags *pattern* の *source-string* への一致を制御する以下のいずれかの値を含むストリング・リテラル:

s ドット (.) がすべての文字と一致することを示します。

s フラグが指定されていない場合、ドット (.) は改行文字 (#x0A) を除くすべての文字と一致します。

m 脱字記号 (^) が任意の行の先頭 (改行文字の後の位置) に一致し、ドル記号 ($) が任意の行の末尾 (改行文字の前の位置) に一致することを示します。

m フラグが指定されていない場合、脱字記号 (^) はストリング全体の先頭に一致し、ドル記号 ($) はストリング全体の末尾に一致します。

i 文字 "a" から "z" および "A" から "Z" について、一致で大/小文字を区別しないことを示します。

i フラグが指定されていない場合、大/小文字の区別があるマッチングが行われます。

x *pattern* 内の空白文字 (#x09、#x0A、#x0D、および #x20) が無視されることを示します。ただし、それらが文字クラス内にある場合は除きます。文字クラス内の空白文字が無視されることはありません。

x フラグが指定されていない場合、空白文字はマッチングに使用されます。

戻り値

source-string が空のシーケンスまたはゼロ長ストリングでない場合、戻り値は、*source-string* に対して以下の操作を実行した結果の xs:string 値のシーケンスです。

- *source-string* で *pattern* に一致する文字列が検索されます。
- *pattern* が複数の代替文字セットを含んでおり、それらの代替文字セットが *source-string* 内の同じ位置で始まる文字列に一致する場合、*source-string* 内の文字列に一致する *pattern* 内の最初の文字のセットのみが、*pattern* に一致するとみなされます。

- *pattern* に一致しないそれぞれの文字のセットは、結果シーケンス内の項目になります。
- *pattern* が *source-string* の先頭の文字列に一致する場合、戻されるシーケンスの最初の項目はゼロ長ストリングです。
- *source-string* 内に *pattern* に対して 2 つの連続する一致が検出された場合、ゼロ長ストリングがシーケンスに追加されます。
- *pattern* が *source-string* の末尾の文字列に一致する場合、戻されるシーケンスの最後の項目はゼロ長ストリングです。

pattern が *source-string* に検出されない場合、*source-string* が戻されます。

pattern がゼロ長ストリングに一致する場合、エラーが戻されます。

source-string が空のシーケンスであるか、またはゼロ長ストリングである場合、結果は空のシーケンスです。

例

次の関数は、疑問符 (?) 文字を除去して、残りの文字からシーケンスを作成することにより、ストリング "?A?B?C?D?" からシーケンスを作成します。

```
fn:tokenize("?A?B?C?D?","\?")
```

戻り値は、シーケンス ("", "A", "B", "C", "D", "") です。

fn:translate 関数

fn:translate 関数は、ストリング内の選択された文字列を置換文字列で置換します。

構文

source-string
: その中に含まれる文字が変換されるストリング。

  source-string は、xs:string データ・タイプであるか、空のシーケンスです。

original-string
: 変換可能な文字を含むストリング。

  original-string は xs:string データ・タイプです。

replacement-string
: *original-string* 内の文字列を置換する文字列を含むストリング。

  replacement-string は xs:string データ・タイプです。

  replacement-string の長さが *original-string* の長さより大きい場合、*replacement-string* 内の追加文字列は無視されます。

戻り値

source-string が空のシーケンスでない場合、戻り値は、以下の操作が実行された場合の結果である xs:string 値です。

- *original-string* に現れる *source-string* 内のそれぞれの文字について、*source-string* 内の文字を、*original-string* 内の文字と同じ位置に現れる *replacement-string* 内の文字で置換します。

  original-string の長さが *replacement-string* の長さより大きい場合、*original-string* には現れるが、*original-string* における文字位置が *replacement-string* 内に対応する位置を持たないような、*source-string* 内の各文字を削除します。

  ある文字が *original-string* 内に複数回現れる場合、*original-string* におけるその文字の最初のオカレンスの位置が、使用される *replacement-string* 内の文字を決定します。
- *original-string* に現れない *source-string* 内の各文字については、その文字をそのままにしておきます。

source-string が空のシーケンスである場合、ゼロ長ストリングが戻されます。

例

次の関数は、ストリング "—aaa—" の文字 a を文字 A で置き換えて - 文字を削除します。

```
fn:translate("---aaa---","a-","A")
```

戻り値は "AAA" です。

fn:upper-case 関数

fn:upper-case 関数は、ストリングを大文字に変換します。

構文

source-string

大文字に変換されるストリング。

source-string は、xs:string データ・タイプであるか、空のシーケンスです。

戻り値

source-string が空のシーケンスでない場合、戻り値は、各文字が対応する大文字に変換された *source-string* である xs:string 値です。対応する大文字を持たない各文字は、オリジナルの形式で戻り値に含まれます。

source-string が空のシーケンスである場合、戻り値はゼロ長ストリングです。

例

以下の関数を実行すると、ストリング 'Test literal 1' が大文字に変換されます。

```
fn:upper-case("Test literal 1")
```

戻り値は "TEST LITERAL 1" です。

次の関数の引数は、"ii" に解決されます。

```
fn:upper-case("&#x131;i")
```

戻り値は "II" です。

fn:year-from-date 関数

fn:year-from-date 関数は、ローカル時間帯の形式の xs:date 値の年コンポーネントを戻します。

構文

date-value
年コンポーネントが取り出される日付値。

date-value は xs:date タイプであるか、空のシーケンスです。

戻り値

date-value が xs:date タイプの場合、戻り値は xs:integer タイプです。値は *date-value* の年コンポーネントであり、負の値にはなりません。

date-value が空のシーケンスである場合、戻り値は空のシーケンスです。

例

次の関数は、 2009 年 10 月 29 日の日付値の年コンポーネントを戻します。

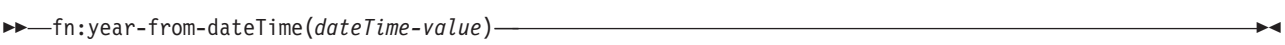
```
fn:year-from-date(xs:date("2009-10-29"))
```

戻り値は 2009 です。

fn:year-from-dateTime 関数

fn:year-from-dateTime 関数は、ローカル時間帯の形式の xs:dateTime 値の年コンポーネントを戻します。

構文



dateTime-value
年コンポーネントが取り出される dateTime 値。

dateTime-value は xs:dateTime タイプであるか、空のシーケンスです。

戻り値

dateTime-value が xs:dateTime タイプである場合、戻り値は xs:integer タイプです。 値は *dateTime-value* の年コンポーネントであり、負の値にはなりません。

dateTime-value が空のシーケンスである場合、戻り値は空のシーケンスです。

例

次の関数は、UTC-8 時間帯の 2009 年 10 月 29 日午前 8:00 の dateTime 値の年コンポーネントを戻します。

```
fn:year-from-dateTime(xs:dateTime("2009-10-29T08:00:00-08:00"))
```

戻り値は 2009 です。

fn:years-from-duration 関数

fn:years-from-duration 関数は、期間の年コンポーネントを戻します。

構文

duration-value
年コンポーネントが取り出される期間値。

duration-value は、空のシーケンスか、または xs:dayTimeDuration、xs:duration、xs:yearMonthDuration のいずれかのタイプの値です。

戻り値

戻り値は、*duration-value* のタイプによって異なります。

- *duration-value* が xs:yearMonthDuration タイプまたは xs:duration タイプの場合、戻り値は xs:integer タイプです。値は、xs:yearMonthDuration としてキャストされた *duration-value* の年コンポーネントです。*duration-value* が負の数である場合、この値も負の数になります。
- *duration-value* が xs:dayTimeDuration タイプである場合、戻り値は xs:integer タイプであり、0 です。
- *duration-value* が空のシーケンスである場合、戻り値は空のシーケンスです。

xs:yearMonthDuration としてキャストされた *duration-value* の年コンポーネントは、年数 (整数) です。これは、xs:yearMonthDuration としてキャストされた *duration-value* の次の総数を12 で除算することにより決まります。

例

以下の関数は、期間が -4 年 -11 カ月 -320 日の年コンポーネントを戻します。

```
fn:years-from-duration(xs:duration("-P4Y11M320D"))
```

戻り値は -4 です。

以下の関数は、9 年 13 カ月の期間の年コンポーネントを戻します。

```
fn:years-from-duration(xs:yearMonthDuration("P9Y13M"))
```

戻り値は 10 です。 期間の総年数を計算する場合、13 カ月は 1 年と 1 カ月に変換されます。 この期間は年コンポーネント 10 年を持つ、P10Y1M と等しいと言えます。

コードに関するライセンス情報および特記事項

IBM は、お客様に、すべてのプログラム・コードのサンプルを使用することができる非独占的な著作使用権を許諾します。お客様は、このサンプル・コードから、お客様独自の特別のニーズに合わせた類似のプログラムを作成することができます。

強行法規で除外を禁止されている場合を除き、IBM、そのプログラム開発者、および供給者は「プログラム」および「プログラム」に対する技術的サポートがある場合にはその技術的サポートについて、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。

いかなる場合においても、IBM および IBM のサプライヤーならびに IBM ビジネス・パートナーは、その予見の有無を問わず発生した以下のものについて賠償責任を負いません。

1. データの喪失、または損傷。
2. 直接損害、特別損害、付随的損害、間接損害、または経済上の結果的損害
3. 逸失した利益、ビジネス上の収益、あるいは節約すべかりし費用

国または地域によっては、法律の強行規定により、上記の責任の制限が適用されない場合があります。

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。

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 Corporation
Software Interoperability Coordinator, Department YBWA
3605 Highway 52 N
Rochester, MN 55901
U.S.A.

本プログラムに関する上記の情報は、適切な使用条件の下で使用することができますが、有償の場合もあります。

本書で説明されているライセンス・プログラムまたはその他のライセンス資料は、IBM 所定のプログラム契約の契約条項、IBM プログラムのご使用条件、またはそれと同等の条項に基づいて、IBM より提供されます。

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。

本書はプランニング目的としてのみ記述されています。記述内容は製品が使用可能になる前に変更になる場合があります。

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。これらのサンプル・プログラムは特定物として現存するままの状態で提供されるものであり、いかなる保証も提供されません。 IBM は、お客様の当該サンプル・プログラムの使用から生ずるいかなる損害に対しても一切の責任を負いません。

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的創作物にも、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_.

プログラミング・インターフェース情報

本書「SQL プログラミング」には、プログラムを作成するユーザーが IBM i のサービスを使用するためのプログラミング・インターフェースが記述されています。

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、『www.ibm.com/legal/copytrade.shtml』 をご覧ください。

Adobe、Adobe ロゴ、PostScript、PostScript ロゴは、Adobe Systems Incorporated の米国およびその他の国における登録商標または商標です。

Linux は、Linus Torvalds の米国およびその他の国における登録商標です。

Microsoft、Windows、Windows NT および Windows ロゴは、Microsoft Corporation の米国およびその他の国における商標です。

UNIX は The Open Group の米国およびその他の国における登録商標です。

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。

使用条件

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。

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。

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。

索引

日本語, 数字, 英字, 特殊文字の順に配列されています。なお, 濁音と半濁音は清音と同等に扱われています。

[ア行]

[カ行]

[サ行]

[タ行]

[ナ行]

[ハ行]

[マ行]

[ヤ行]

[ラ行]

A

B

C

D

E

F

H

I

J

L

M

N

P

R

S

T

U

X

Y

プログラム番号: 5770-SS1

Printed in Japan

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