SYBASE®
An SAP® Company

XML サービス

Adaptive Server® Enterprise

15.7

ドキュメント ID：DC20146-01-1570-01
改訂：2011 年 8 月

Sybase, Inc., One Sybase Drive, Dublin, CA 94568.

目次

第 1 章 XML サービスの概要 1
- XML の機能 1
- 概要 2
  - データベースにおける XML 2
  - サンプル XML ドキュメント 3
  - XML ドキュメント・タイプ 7

第 2 章 XML クエリ関数 9
- XML クエリ関数 9
  - 「例」の項 10
- xmlextract 10
- xmltest 17
- xmlparse 21
- xmlrepresentation 24
- xmlvalidate 26
- option_strings：一般的なフォーマット 36
  - クエリ関数のオプション値 36

第 3 章 XML 言語と XML 問い合わせ言語 39
- 文字セットのサポート 39
- URI のサポート 39
- ネームスペースのサポート 40
- XML スキーマのサポート 40
- XML ドキュメントの定義済みエンティティ 40
- XPath クエリの定義済みエンティティ 41
- スペース 42
- 空の要素 42
- XML 問い合わせ言語 43
  - XPath でサポートされる構文とトークン 43
  - XPath の演算子 44
  - XPath 関数 46
- カッコで囲んだ式 51
  - カッコとサブスクリプト 51
  - カッコと union 53

第 4 章 for xml マッピング関数 55
for xml 句 55
for xml サブクエリ 59
for xml schema と for xml all 61

第 5 章 XML マッピング 67
SQLX オプション 67
SQLX オプションの定義 69
SQLX データのマッピング 77
重複したカラム名と名前のないカラムのマッピング 77
XML 名への SQL 名のマッピング 80
XML 値への SQL 値のマッピング 83

第 6 章 XML における国際化 (I18N) のサポート 85
概要 85
Unicode データ型 86
サロゲート・ペア 86
数値文字表現 86
クライアントとサーバ間の変換 86
文字セットと XML データ 87
for xml における I18N 87
オプション文字列 88
for xml における数値文字表現 88
header オプション 89
例外 89
例 89
xmlparse における I18N 92
オプション 93
xmlextract における I18N 93
NCR オプション 93
xmlextract におけるソート順 94
XML サービスにおけるソート順 94
xmlvalidate における I18n 95
NCR オプション 95

第 7 章 xmltable() 97
概要 97
xmltable と derived table の構文 97
xmltable 98

付録 A **sample_docs サンプル・テーブル 115**
sample_docs テーブルのカラムとロー 115
sample_docs テーブルのカラム 115
sample_docs テーブルのロー 116
sample_docs テーブル 117
テーブル・スクリプト (publishers テーブルの場合) 118
publishers テーブルの表現 118
titles テーブルの表現 119

付録 B **XML サービスと外部ファイル・システム・アクセス 125**
使用開始にあたって 125
XML サービスと外部ファイル・システム・アクセスの有効化..... 125
外部ファイル・システムを使用した文字セット変換 126
例 126
XML ドキュメントの設定とプロキシ・テーブルの作成 127
例：XML ドキュメントから本のタイトルを抽出する 128
例：XML ドキュメントまたは XML クエリ結果を Adaptive Server テーブルにインポートする 128
例：ファイル・システムに解析済み XML ドキュメントを格納する ... 129
例：外部ファイル・アクセスを伴う 'xmlerror' オプションの機能 130
例：xmlextract に 'xmlerror=message' オプションを指定する 131
例：'xmlerror=message' オプションを伴う XML ドキュメントと XML 以外のドキュメントの解析 131
例：XML 以外のドキュメントに 'xmlerror=null' オプションを使用する..... 132

付録 C **Java ベースの XQL プロセッサとネイティブ XML プロセッサ間のマイグレート 135**
概要..... 135
ドキュメントとクエリのマイグレート..... 135
Java ベースの XQL プロセッサとネイティブ XML プロセッサ間でのドキュメントのマイグレート 136
Java ベースの XQL プロセッサとネイティブ XML プロセッサ間でのテキスト・ドキュメントのマイグレート 136
再生成したコピーからのドキュメントのマイグレート..... 136
Java ベースの XQL プロセッサからのテキスト・ドキュメントの再生成 137
ネイティブ XML プロセッサからのテキスト・ドキュメントの再生成 138
ネイティブ XML プロセッサと Java ベースの XQL プロセッサ間でのクエリのマイグレート 138

付録 D xmltable() のサンプル・アプリケーション 139
サンプル・テーブル 139
depts ドキュメントの使用 143
depts ドキュメントの構造 143
depts ドキュメントからの SQL テーブルの作成 143

索引 147

第 1 章

XML サービスの概要

この章では、Adaptive Server® Enterprise の XML サービス機能について説明します。

トピック名	ページ
XML の機能	1
概要	2

XML の機能

XML サービスには次の機能があります。

- XML の生成：標準 SQLX 形式の XML ドキュメントとして結果セットを返す select コマンドの for xml 句。
- XML の格納：
  - char、varchar、text、unichar、univarchar、unitext カラム内の文字データ、または解析済み XML として格納された XML ドキュメントのサポート。
  - XML ドキュメントの解析とインデックス付けを行い、解析済みのインデックス付き表現を格納用に生成する xmlparse。
  - DTD または XML スキーマの定義に照らして XML ドキュメントを検証する xmlvalidate。
- XMLのクエリと細分化：XML ドキュメントに対してクエリを実行しデータの抽出を行う xmltest および xmlextract。
- I18N のサポート：非 ASCII データを含む XML ドキュメントの生成、格納、クエリ、抽出のサポートを含む、XML ドキュメントでの Unicode および非 ASCII サーバ文字セットのサポート。

概要

HTML (ハイパーテキスト・マークアップ言語) と同様、XML はマークアップ言語であり、SGML (汎用マークアップ言語) のサブセットです。しかし、XML はより完全で統制がとれており、使用するアプリケーション指向のマークアップ・タグを定義することができます。こうした特性のため、XML はデータ交換に特に適しています。

Adaptive Server に格納されたデータから XML 形式のドキュメントを生成できます。逆に Adaptive Server に XML ドキュメントを保存し、データをそこから抽出することもできます。また、Adaptive Server を使用して Web 上に保存された XML ドキュメントを検索することもできます。

XML はマークアップ言語であり、Web 出版やドキュメントの分散処理において HTML をしのぐ機能性を提供するために開発された SGML のサブセットです。

データベースにおける XML

- XML ドキュメントは厳密な句構造を持っているので、データの検索やアクセスが容易です。たとえば、次のように、すべての要素には開始タグと対応する終了タグが必要です。

  * *段落* *

- XML を使うと、顧客番号と項目番号など、異なるタイプのデータを区別するタグを開発し、使用することができます。
- XML を使うと、アプリケーション固有のドキュメント・タイプを作成でき、ドキュメントの種類を区別することができます。
- XML ドキュメントでは、XML データのさまざまな表示方法が可能となります。HTML ドキュメントと同様に XML ドキュメントに含まれるのはマークアップと内容のみで、フォーマット指示は含まれません。フォーマット指示は通常、クライアントで提供されます。

XML は SGML ほど複雑ではありませんが、HTML より複雑で、柔軟性があります。XML と HTML は通常、同じブラウザやプロセッサで読み込むことができますが、XML には、HTML よりも効率的にドキュメントを共有できる特性があります。

XML ドキュメントは、Adaptive Server 内で次のような形式で保存できます。

- データ型が char、varchar、unichar、univarchar、text、unitext、java.lang.String、image のいずれかであるカラム内の文字データ
- image カラム内の解析済み XML

サンプル XML ドキュメント

このサンプル・ドキュメント Order は、購入注文アプリケーション用に作成されています。顧客が出した注文は、日付と顧客 ID で識別します。注文のあった各項目には、項目 ID、項目名、数量、数量単位の情報があります。

画面には次のように表示されます。

ORDER

Date: July 4, 2003

Customer ID: 123

Customer Name: Acme Alpha

Items:

Item ID	Item Name	Quantity
987	Coupler	5
654	Connector	3 dozen
579	Clasp	1

この注文データを XML では次のように表現できます。

```
<?xml version="1.0"?>
    <Order>
 <Date>2003/07/04</Date>
 <CustomerId>123</CustomerId>
 <CustomerName>Acme Alpha</CustomerName>
   <Item>
 <ItemId> 987</ItemId>
 <ItemName>Coupler</ItemName>
 <Quantity>5</Quantity>
 </Item>
<Item>
 <ItemId>654</ItemId>
 <ItemName>Connector</ItemName>
 <Quantity unit="12">3</Quantity>
 </Item>
<Item>
 <ItemId>579</ItemId>
 <ItemName>Clasp</ItemName>
 <Quantity>1</Quantity>
 </Item>
</Order>
```

XML ドキュメントには、次のような 2 つの特性があります。

- XML ドキュメントは、項目表示を指定するためのフォント、スタイル、色を指示しません。
- マークアップ・タグは厳密にネストされています。各開始タグ (<*tag*>) は、対応する終了タグ (</*tag*>) を持っています。

注文データの XML ドキュメントは、4 つの主要素から成り立っています。

- XML 宣言 <?xml version="1.0"?>。これは、"Order" を XML ドキュメントとして識別するためのものです。

  各ドキュメントの XML 宣言は、明示的にまたは暗黙的に文字コード (文字セット) を指定します。XML はドキュメントを文字データとして表現します。明示的に文字セットを指定するには、文字セットを XML 宣言の中に入れます。次に例を示します。

  ```
  <?xml version="1.0" encoding="ISO-8859-1">
  ```

  文字セットを XML 宣言の中に入れない場合、Adaptive Server ではデフォルトの文字セットである UTF8 が使用されます。

  注意 クライアントとサーバでデフォルトの文字セットが異なる場合、Adaptive Server は標準の文字セット変換をスキップします。これは、宣言された文字セットと実際の文字セットを一致させるためです。「XML における国際化 (I18N) のサポート」(85 ページ) を参照してください。

- ユーザ作成の要素タグ。<Order>...</Order>、<CustomerId>...</CustomerId>、<Item>...</Item> など。
- テキスト・データ。"Acme Alpha"、"Coupler"、"579" など。
- 要素タグに埋め込まれた属性。<Quantity unit = "12"> など。この埋め込みを使用して、要素をカスタマイズできます。

これらのコンポーネントや厳密にネストされた要素タグを持つドキュメントを、「**正しい形式の XML ドキュメント**」と呼びます。前述の例では、要素タグが対応するデータを表し、ドキュメントにはフォーマット指示がないことに注意してください。

次に、別の XML ドキュメントの例を示します。

```
<?xml version="1.0"?>
 <Info>
    <OneTag>1999/07/04</OneTag>
    <AnotherTag>123</AnotherTag>
    <LastTag>Acme Alpha</LastTag>
   <Thing>
       <ThingId> 987</ThingId>
       <ThingName>Coupler</ThingName>
       <Amount>5</Amount>
       <Thing/>
```

```
  <Thing>
   <ThingId>654</ThingId>
   <ThingName>Connecter</ThingName>
</Thing>
   <Thing>
       <ThingId>579</ThingId>
       <ThingName>Clasp</ThingName>
       <Amount>1</Amount>
   </Thing>
</Info>
```

この “Info” という例も、正しい形式の XML ドキュメントで、Order ドキュメントと同じ構造とデータを持っています。しかし、Info が使用するドキュメント・タイプ定義 (DTD) は Order ドキュメントの DTD と異なるため、Order ドキュメント用に設計されたプロセッサには認識されません。DTD の詳細については、「XML ドキュメント・タイプ」(7 ページ) を参照してください。

Order データの HTML 表示

購入注文アプリケーションについて考えてみます。顧客が注文書を送信します。注文書 (Order) は、Date、CustomerID で識別され、1 つ以上の項目がリストされます。各項目には ItemID、ItemName、Quantity、units があります。

このような注文書のデータは、次のように画面に表示されます。

ORDER

Date: July 4, 1999

Customer ID: 123

Customer Name: Acme Alpha

Items:

Item ID	Item Name	Quantity
987	Coupler	5
654	Connector	3 dozen
579	Clasp	1

このデータでは、名前が Acme Alpha で Customer ID が 123 の顧客が、1999 年 7 月 4 日にカプラ (coupler)、コネクタ (connector)、留め具 (clasp) の注文書を送信しています。

この注文書データを表示する HTML テキストは次のようになります。

```
<html>
<body>
<p>ORDER
<p>Date:  July 4, 1999
<p>Customer ID:  123
<p>Customer Name:  Acme Alpha
<p>Items:</p>
<table bgcolor=white align=left border="3"
    cellpadding=3>
<tr><td><B>Item ID   </B></tr>
    <td><B>Item Name   </B></tr>
    <td><B>Quantity   </B>
    </td></td></tr>
<tr><td>987</td>
    <td>Coupler</td>
    <td>5</td></tr>
<tr><td>654</td>
    <td>Connector</td>
    <td>3 dozen</td></tr>
<tr><td>579</td>
    <td>Clasp</td>
    <td>1</td></tr>
</table>
</body>
</html>
```

この HTML テキストには、次のような制限があります。

- データとフォーマット仕様の両方が含まれています。
  - データとは、Customer ID、さまざまな Customer Name、Item Name、Quantity を指します。
  - フォーマット仕様とは、フォント・スタイル (*....*)、色 (*bcolor=white*)、レイアウト (*<table>....</table>*)、および *Customer Name* などの補助フィールド名を指します。
- HTML ドキュメントの構造は、データの抽出にはあまり適していません。

  テーブルなどの要素では、カッコで囲んだ開始タグと終了タグが必要ですが、パラグラフ・タグ (“*<p>*”) などの他の要素では、終了タグを省略できる場合があります。

  パラグラフ・タグ (“*<p>*”) などの要素は、さまざまな種類のデータで使用されるため、Customer ID の 123 と Item ID の 123 を区別することは、周辺のフィールド名からコンテキストを推測しないと難しくなります。

このようにデータとフォーマットの両方が含まれていること、また厳密な句構造がないことが原因で、HTML ドキュメントをさまざまな表示スタイルに適合させたり、データ交換や格納に HTML ドキュメントを使用することが難しくなります。XML は、HTML に似ていますが、このような欠点に対処した制限や拡張機能が含まれています。

XML ドキュメント・タイプ

「**ドキュメント・タイプ定義 (DTD)**」は、XML ドキュメントのクラス構造を定義し、クラス間の識別を可能にします。DTD は、1 つのクラスに特有の要素と属性の定義リストです。DTD を一度設定すると、他のドキュメントからその DTD を参照したり、DTD を現在の XML ドキュメントに埋め込んだりできます。

XML Order ドキュメントの DTD は以下のとおりです (「サンプル XML ドキュメント」(3 ページ) の項を参照してください)。

```
<!ELEMENT Order (Date, CustomerId, CustomerName, Item+)>
 <!ELEMENT Date (#PCDATA)>
 <!ELEMENT CustomerId (#PCDATA)>
 <!ELEMENT CustomerName (#PCDATA)>
 <!ELEMENT Item (ItemId, ItemName, Quantity)>
 <!ELEMENT ItemId (#PCDATA)>
 <!ELEMENT ItemName (#PCDATA)>
 <!ELEMENT Quantity (#PCDATA)>
 <!ATTLIST Quantity units CDATA #IMPLIED>
```

この DTD では次のことを指定しています。

- 1 つの注文に対する必須要素は、日付、顧客 ID、顧客名、1 つまたは複数の項目です。プラス記号 “+” は 1 つまたは複数の項目を表します。プラス記号が付いた項目は必須です。同じ位置の疑問符は、オプションの要素を表します。要素内のアスタリスクは、1 つの要素が 0 回またはそれ以上発生することを示します (たとえば、上記の 1 行目に “Item*” というアスタリスクが付いていた場合、注文書内の項目数は 0 でも任意の数であってもかまいません)。
- “(#PCDATA)” で定義された要素は文字テキストです。
- 最終行の “<ATTLIST...>” 定義は、Quantity 要素に “units” 属性があることを指定し、最終行の行末の “#IMPLIED” は、“units” 属性がオプションであることを示します。

XML ドキュメントの文字テキストに制約はありません。たとえば、Quantity 要素のテキストを数値に限定する方法がないので、以下が有効となります。

```
<Quantity unit=”Baker’s dozen”>three</Quantity>
<Quantity unit=”six packs”>plenty</Quantity>
```

要素のテキストは、XML データを処理するアプリケーションによって制限されます。

XML の DTD は、<?xml version="1.0"?> 命令に従っている必要があります。DTD を XML ドキュメントの中に入れることも、外部 DTD を参照することもできます。

- 外部 DTD を参照するには、次のようにします。

```
<?xml version="1.0"?>
 <!DOCTYPE Order SYSTEM  "Order.dtd”>
 <Order>
...
 </Order>
```

- DTD を埋め込んだ場合は、次のようになります。

```
<?xml version="1.0"?>
 <!DOCTYPE Order [
 <!ELEMENT Order (Date, CustomerId, CustomerName,
 Item+)>
 <!ELEMENT Date (#PCDATA)
 <!ELEMENT CustomerId (#PCDATA)>
 <!ELEMENT CustomerName (#PCDATA)>
 <!ELEMENT Item (ItemId, ItemName, Quantity)>
 <!ELEMENT ItemId (#PCDATA)>
 <!ELEMENT ItemName (#PCDATA)>
 <!ELEMENT Quantity (#PCDATA)>
 <!ATTLIST Quantity units CDATA #IMPLIED>

]>
 <Order>
       <Date>1999/07/04</Date>
       <CustomerId>123</CustomerId>
       <CustomerName>Acme Alpha</CustomerName>

   <Item>
       ...
 </Item>
 </Order>
```

XML ドキュメントでは DTD は必須ではありません。しかし、「valid な XML ドキュメント」では DTD があり、その DTD に従っています。

第 2 章

XML クエリ関数

この章では、XML クエリ関数について詳しく説明し、*option_string* パラメータの一般的なフォーマットについて説明します。

トピック名	ページ
XML クエリ関数	9
xmlextract	10
xmltest	17
xmlparse	21
xmlrepresentation	24
xmlvalidate	26
option_strings：一般的なフォーマット	36

XML クエリ関数

この項では、SQL 文で XML ドキュメントにアクセスして処理するための SQL 拡張機能について説明します。xmlextract、xmlparse、xmltest は、XML サービスによって導入された新しい予約語です。

関数は、次のとおりです。

表 2-1: XML クエリ関数

関数	説明
xmlextract	XML ドキュメントに XML クエリ式を適用し、選択した結果を返す組み込み関数。
xmltest	XML ドキュメントに XML クエリ式を適用し、ブール値の結果を返す SQL 述部。
xmlparse	処理の効率を高めるために XML ドキュメントの解析とインデックス付けを行う組み込み関数。
xmlrepresentation	特定の image カラムに解析済み XML ドキュメントが含まれるかどうかを判断する組み込み関数。
xmlvalidate	DTD または XML スキーマに照らして XML ドキュメントを検証する組み込み関数。

「例」の項

これらの関数の説明には例が含まれており、テーブルの作成と移植のためのスクリプトを示す「付録 A sample_docs サンプル・テーブル」を参考にしたものです。

xmlextract

XML_query_expression を *xml_data_expression* に適用し、結果を返す組み込み関数です。この関数は、SQL の substring オペレーションに似ています。

構文

xmlextract_expression ::=
    xmlextract (*xml_query_expression*,*xml_data_expression*
    [*optional_parameters*])
xml_query_expression ::=*basic_string_expression*
xml_data_expression ::= *general_string_expression*
optional_parameters ::=
    options_parameter
  |*returns_type*
  | *options_parameter returns_type*
options_parameter ::= [,] option *option_string*
returns_type ::= [,] returns *dataype*
datatype ::= {*string_type* | *computational_type* | *date_time_type* }
string_type ::= *char* (*integer*) | *varchar* (*integer*)
  | *unichar* (*integer*) | *univarchar* (*integer*)
  | *text* | *unitext* | *image*
computational_type ::= *integer_type* | *decimal_type* | *real_type*
  | *date_time_type*
integer_type ::= [unsigned] {integer | int | tinyint | smallint | bigint}
decimal_type ::= {decimal | dec | numeric } [(integer [, integer])]
real_type ::= real | float | double precision
date_time_type ::= date | time | datetime
option_string ::= [,] *basic_string_expression*

説明

注意 I18N データについては、「XML における国際化 (I18N) のサポート」(85 ページ) を参照してください。

- *basic_string_expression* は、データ型が character、varchar、unichar、univarchar、または *java.lang.String* の *sql_query_expression* です。
- *general_string_expression* は、データ型が text、image、character、varchar、unitext、unichar、univarchar、または *java.lang.String* の *sql_query_expression* です。
- *xmlextract* 式は、SQL 言語の文字式が許可されているすべての場所で使用できます。
- *options_parameter* のデフォルト値は空の文字列です。null のオプション・パラメータは空の文字列として扱われます。

- *xml_query_expression* の値、または xmlextract() のドキュメント引数が null の場合、xmlextract() の結果は null です。
- *xml_data_expression* パラメータの値は、XML クエリ式を実行するためのランタイム・コンテキストです。
- xmlextract() のデータ型は *returns_type* で指定されます。
- *returns_type* のデフォルト値は text です。
- *returns_type* に整数なしの varchar が指定されている場合、デフォルト値は 255 です。
- *returns_type* に精度 (最初の整数) なしの numeric または decimal が指定されている場合、デフォルト値は 18 です。位取り (2 番目の整数) なしで指定されている場合、デフォルトは 0 です。
- クエリ引数またはドキュメント引数が null である場合、xmlextract は null を返します。
- XPath query が無効である場合、xmlextract は例外を引き起こします。
- xmlextract の初期結果は、*xml_query_expression* を *xml_data_expression* に適用した結果です。この結果は XPath 標準によって指定されます。
- *returns_type* に *string_type* が指定されている場合、初期結果の値はそのデータ型の文字列ドキュメントとして返されます。
- *returns_type* に *computational_type* または *datetime_type* データ型が指定されている場合、初期結果はそのデータ型に変換されて返されます。変換は、convert 組み込み関数に対して指定されているルールに従います。

注意　初期結果は convert 組み込み関数に適した値である必要があります。そのためには、XML クエリ式で text() 参照を使用する必要があります。後述の例を参照してください。

注意　次の項目については、「第 3 章 XML 言語と XML 問い合わせ言語」を参照してください。

- 外部 URI 参照、XML ネームスペース、XML スキーマに対する制限。
- 定義済みエンティティと対応する文字列 *&* (&)、*<* (<)、*>* (>)、*&quote;* (“)、*'* (’) の処理方法。エンティティにはセミコロンを含めます。
- スペースの処理方法。
- 空の要素の処理方法。

option_string

option_string の一般的なフォーマットについては「option_strings：一般的なフォーマット」(36 ページ)を参照してください。

xmlextract 関数でサポートされるオプションは次のとおりです。

```
xmlerror = {exception | null | message}
ncr = {no | non_ascii | non_server}
```

ncr オプションとそのデフォルト値については、「XML における国際化 (I18N) のサポート」(85 ページ)を参照してください。

例外

xml_data_expression の値が有効な XML でないか、すべてブランクまたは空の文字列である場合は、次のようになります。

- 明示的なオプションまたはデフォルトのオプションで xmlerror=exception と指定した場合は、例外が発生します。
- 明示的なオプションまたはデフォルトのオプションで xmlerror=null と指定した場合は、null 値が返されます。
- 明示的なオプションまたはデフォルトのオプションで xmlerror=message と指定した場合は、例外メッセージを伴う XML 要素を含む文字列が返されます。この値は有効な XML です。
- グローバル変数 *@@error* は、xmlerror の値が exception、null、または message のいずれであっても、最後のエラーのエラー番号を返します。

xmlextract_expression の *returns_type* が *string_type* で、*xml_query_expression* パラメータの実行時の評価結果がその型の最大長よりも長い場合は、例外が発生します。

例

以下の例は、「付録 A sample_docs サンプル・テーブル」に記載されている *sample_docs* テーブルを使用しています。

この例は、bookstore/book/price が 55 であるか from 属性が “Harvard” である bookstore/book/author/degree を持つドキュメントのタイトルを選択します。

```
select xmlextract('/bookstore/book[price=55
   | author/degree/[@from="Harvard"] ]/title'
     text_doc )
from sample_docs
------------------------------------------------------
<title>History of Trenton</title>
<title>Trenton Today, Trenton Tomorrow</title>

NULL

NULL
```

次の例は、ロー要素に 10 未満の price 要素または “Boston” に等しい city 要素があるドキュメントの row/pub_id 要素を選択します。

このクエリは次の 3 つのローを返します。

- bookstore ローからの null 値
- publishers ローからの 1 つの “<row>...</row>” 要素
- titles ローからの 4 つの “<row>...</row>” 要素

```
select xmlextract('//row[price<10 | city="Boston" ]/pub_id',
     text_doc) from sample_docs2>
-----------------------------------
NULL
XML サービス<pub_id>0736</pub_id>

<pub_id>0736</pub_id>
<pub_id>0877</pub_id>
<pub_id>0736</pub_id>
<pub_id>0736</pub_id>

(3 rows affected)
```

次の例は、“Seven Years in Trenton” の価格を整数として選択します。このクエリには複数の手順があります。

1　“Seven Years in Trenton” の価格を XML 要素として選択します。

```
select xmlextract
('/bookstore/book[title="Seven Years in Trenton"]/price',text_doc)
from sample_docs
where name_doc='bookstore'
-----------------------------------
<price>12</price>
```

2　returns integer 句を追加して、定価を integer として選択します。

```
select xmlextract
     ('/bookstore/book[title="Seven Years in Trenton"]/price',
    text_doc returns integer)
    from sample_docs
    where name_doc='bookstore'
Msg 249, Level 16, State 1:
Line 1:
Syntax error during explicit conversion of VARCHAR value '<price>12</price>' to
an INT field.
```

3 returns 句で numeric、money、または date-time データ型を指定するには、XML クエリが指定されたデータ型への変換に適した値を返す必要があります。したがって、クエリで text() 参照を使用することで XML タグを削除する必要があります。

```
select xmlextract
  ('/bookstore/book[title="Seven Years in Trenton"]/price/text()',
  text_doc returns integer)
  from sample_docs
  where name_doc='bookstore'
-----------
         12
```

4 returns 句で numeric、money、または date-time データ型を指定するには、XML クエリがリストではなく 1 つの値を返す必要もあります。たとえば、次のクエリは価格のリストを返します。

```
select xmlextract
    ('/bookstore/book/price',
    text_doc)
    from sample_docs
    where name_doc='bookstore'
-----------
<price>12</price>
<price>55</price>
<price intl="canada" exchange="0.7">6.50</price>
```

5 text() 参照を追加すると次の結果が生成されます。

```
select xmlextract
('/bookstore/book/price/text()',
text_doc)
from sample_docs
where name_doc='bookstore'
-----------------------------
12556.50
```

6 returns integer 句を指定すると例外が発生し、結合された値が整数への変換に適していないことが示されます。

```
select xmlextract
   ('/bookstore/book/price/text()',
   text_doc returns integer)
   from sample_docs
   where name_doc='bookstore'
Msg 249, Level 16, State 1:
Line 1:
Syntax error during explicit conversion of VARCHAR value
'12556.50' to an INT field.
```

xmlerror オプションを示すために、次のコマンドでは *sample_docs* テーブルに無効なドキュメントを挿入します。

```
insert into sample_docs (name_doc, text_doc)
values ('invalid doc', '<a>unclosed element<a>')

(1 row affected)
```

次の例では、xmlerror オプションが xmlextract 関数による無効な XML ドキュメントの処理方法を判断します。

- xmlerror=exception (デフォルト) の場合は、例外が発生します。

```
select xmlextract('//row', text_doc
   option 'xmlerror=exception')
from sample_docs

Msg 14702, Level 16, State 0:
Line 2:
XMLPARSE(): XML parser fatal error
   <<The input ended before all started tags
were ended. Last tag started was 'a'>>
   at line 1, offset 23.
```

- xmlerror=null の場合は、null 値が返されます。

```
select xmlextract('//row', test_doc
  option 'xmlerror=null')
from sample_docs

(0 rows affected)
```

- xmlerror=message の場合は、解析済み XML ドキュメントがエラー・メッセージとともに返されます。

```
select xmlextract('//row', test_doc
  option 'xmlerror=message')
from sample_docs
-----------------------------------

<xml_parse_error>The input ended before all startedtags
were ended. Last tag started was 'a'</xml_parse_error>
```

xmlerror オプションは、解析済み XML ドキュメントであるドキュメント、または xmlparse によるネストされた明示的な呼び出しで返されるドキュメントには適用されません。

たとえば、次の xmlextract 呼び出しでは、xml_data_expression は未解析の文字列ドキュメントであるため、xmlerror オプションが適用されます。このドキュメントは無効な XML であるため、例外が発生します。xmlerror オプションは、例外メッセージが例外メッセージ付きの XML ドキュメントとして返す必要があることを示します。

```
select xmlextract('/',  '<a>A<a>'  option'xmlerror=message')
-------------------------------------------------
<xml_parse_error>The input ended before all started tags were ended.
   Last tag started was 'a'</xml_parse_error>
```

次の xmlextract 呼び出しでは、xml_data_expression は xmlparse 関数による明示的な呼び出しによって返されます (「xmlparse」(21 ページ) を参照してください)。したがって、外側の xmlextract 呼び出しの xmlerror オプションではなく、明示的な xmlparse 呼び出しのデフォルトの xmlerror オプションが適用されます。デフォルトの xmlerror オプションは exception であるため、明示的な xmlparse 呼び出しでは例外が発生します。

```
select xmlextract('/', xmlparse('<a>A<a>')
     option 'xmlerror=message'))
-------------------------------------------------
Msg 14702, Level 16, State 0:
Line 2:
XMLPARSE(): XML parser fatal error
 <<The input ended before all started tags were ended.
 Last tag started was 'a'>> at line 1, offset 8.
```

xmlparse のネストされた明示的な呼び出しに xmlerror=message オプションを適用するには、xmlparse 呼び出しにオプションとして指定します。

```
select xmlextract('/',
   xmlparse('<a>A<a>' option 'xmlerror=message'))
-------------------------------------------------
<xml_parse_error>The input ended before all started
tags were ended. Last tag started was
'a'</xml_parse_error>
```

未解析 XML ドキュメントと xmlparse のネストされた呼び出しの xmlerror オプションの処理をまとめると、次のようになります。

- xmlerror オプションは、ドキュメントのオペランドが未解析ドキュメントである場合のみ xmlextract で使用されます。
- ドキュメントのオペランドが明示的な xmlparse 呼び出しである場合、その呼び出しの暗黙的または明示的な xmlerror オプションは、xmlextract の暗黙的または明示的な xmlerror オプションを無効にします。

次のコマンドは、*sample_docs* テーブルを元の状態にリストアします。

```
delete from sample_docs
where na_doc='invalid doc'
```

xmltest

XML クエリ式を評価する述部です。XML ドキュメント・パラメータを参照でき、ブール値の結果を返します。SQL の like 述部と似ています。

構文

xmltest_predicate ::=
    xml_query_expression [not] xmltest *xml_data*
    [option *option_string*]
xml_data ::=
    xml_data_expression | (*xml_data_expression*)
xml_query_expression::= *basic_string_expression*
xml_data_expression ::= *general_string_expression*
option_string ::= *basic_string_expression*

説明

注意 I18N データの処理については、「第 6 章 XML における国際化 (I18N) のサポート」を参照してください。

- *basic_string_expression* は、データ型が character、varchar、unichar、univarchar、または java.lang.String の *sql_query_expression* です。
- *general_string_expression* は、データ型が character、varchar、unichar、univarchar、text、unitext、または java.lang.String の *sql_query_expression* です。
- xmltest 述部は、SQL 言語の SQL 述部が許可されているすべての場所で使用できます。
- 次のように指定する xmltest 呼び出しがあるとします。

  ```
  X not xmltest Y options Z
  ```

  これは次と同等です。

  ```
  not X xmltest Y options Z
  ```

- xmltest() の *xml_query_expression* または *xml_data_expression* が null である場合、xmltest() の結果は不定になります。
- *xml_query_expression* の値、または xmlextract() のドキュメント引数が null の場合、xmlextract() の結果は null です。
- *xml_data_expression* パラメータの値は、*XPath* 式を実行するためのランタイム・コンテキストです。
- xmltest() は、次のようにブール値の *true* または *false* に評価されます。
  - xmltest() の *xml_query_expression* が、結果が *empty* (*not empty*) である XPath 式の場合、xmltest() は *false* (*true*) を返す。
  - xmltest() の *xml_query_expression* が、結果がブール値の *false* (*true*) である XPath 式の場合、xmltest() は *false* (*true*) を返す。
  - XPath 式が無効である場合、xmltest は例外を引き起こします。

注意 次の項目については、「第 3 章 XML 言語と XML 問い合わせ言語」を参照してください。

- 外部 URI 参照、XML ネームスペース、XML スキーマに対する制限。
- 定義済みエンティティと対応する文字列 *&* (&)、*<* (<)、*>* (>)、*&quote;* (“)、*'* (’) の処理方法。エンティティにはセミコロンを含めます。
- スペースの処理方法。
- 空の要素の処理方法。

オプション

option_string の一般的なフォーマットについては、「option_strings：一般的なフォーマット」(36 ページ) を参照してください。

xmltest 述部でサポートされるオプションは、xmlerror = {*exception* | *null*} です。

xmlextract と xmlparse でサポートされる代替 message は、xmltest では有効ではありません。「例外」の項を参照してください。

例外

xml_data_expression の値が有効な XML でないか、すべてブランクまたは空の文字列である場合は、次のようになります。

- 明示的なオプションまたはデフォルトのオプションで xmlerror=exception と指定した場合は、例外が発生します。
- 明示的なオプションまたはデフォルトのオプションで xmlerror=null と指定した場合は、null 値が返されます。
- xmlerror=message を指定した場合、null 値が返されます。

例

以下の例は、「付録 A sample_docs サンプル・テーブル」に記載されている *sample_docs* テーブルを使用しています。

この例は、*text_doc* に “Boston” と等しい row/city 要素を含む各ローの *name_doc* を選択します。

```
select name_doc from sample_docs
where  '//row[city="Boston"]' xmltest text_doc
  name_doc
------------------------
  publishers

(1 row affected)
```

次の例では、xmltest 述部は、ブール値の *false*/*true* の結果と *empty*/*not-empty* の結果に対して *false*/*true* を返します。

```
-- A boolean true is 'true':
select case when '/a="A"' xmltest '<a>A</a>'
            then 'true' else 'false' end2>
-----
true

-- A boolean false is 'false'
select case when '/a="B"' xmltest '<a>A</a>'
            then 'true' else 'false' end
-----
false

-- A non-empty result is 'true'
select case when '/a' xmltest '<a>A</a>'
             then 'true' else 'false' end

-----  true
-- An empty result is 'false'
select case when '/b' xmltest '<a>A</a>'
            then 'true' else 'false' end
-----
false

-- An empty result is 'false' (second example)
select case when '/b="A"' xmltest '<a>A</a>'
            then 'true' else 'false' end
-----
false
```

xmlerror オプションを示すために、次のコマンドでは *sample_doc*s テーブルに無効なドキュメントを挿入します。

```
insert into sample_docs (name_doc, text_doc)
values ('invalid doc', '<a>unclosed element<a>)

(1 row affected)
```

次の例では、xmlerror オプションが xmltest 述部による無効な XML ドキュメントの処理方法を判断します。

- xmlerror=exception (デフォルトの結果) の場合は、例外が発生し、グローバル変数 *@@error* にはエラー メッセージ 14702 が含まれます。

```
select name_doc from sample_docs
where '//price<10/*' xmltest text_doc
   option 'xmlerror=exception'

Msg 14702, Level 16, State 0:
Line 2:
XMLPARSE(): XML parser fatal error
   <<The input ended before all started tags were
ended. Last tag started was 'a'>> at line 1,
offset 23.
```

  @@error の内容を表示するには、以下のように入力します。

```
select @@error
------------
     14702
(1 row affected)
```

- xmlerror=null または xmlerror=message の場合は、null (不明) 値が返され、グローバル変数 *@@error* にはエラー メッセージ 14701 が含まれます。

```
select name_doc from sample_docs
where '//price<10/*' xmltest text_doc
   option 'xmlerror=null'

(0 rows affected)
```

  @@error の内容を表示するには、以下のように入力します。

```
select @@error
---------
     14701
(1 row affected)
```

次のコマンドは、*sample_docs* テーブルを元の状態にリストアします。

```
delete from sample_docs
where name_doc='invalid doc'
```

xmlparse

パラメータとして渡される XML ドキュメントを解析し、ドキュメントの解析済み形式を含む image 値を返す組み込み関数です。

構文

xmlparse_call ::=
    xmlparse(*general_string_expression*
    [*options_parameter*][*returns_type*])
options_parameter ::= [,] option *option_string*
option_string ::= *basic_string_expression*
returns type ::= [,] returns {image | binary | varbinary [(*integer*)]}

説明

注意　I18N データの処理については、「第 6 章 XML における国際化 (I18N) のサポート」を参照してください。

- returns 句を省略した場合、デフォルトは returns image です。
- *basic_string_expression* は、データ型が character、varchar、unichar、univarchar、または java.lang.String の *sql_query_expression* です。
- *general_string_expression* は、データ型が character、varchar、unichar、univarchar、text、unitext、image、または java.lang.String の *sql_query_expression* です。
- xmlparse() のいずれかのパラメータが null である場合、呼び出し結果は null になります。
- *general_string_expression* がすべてブランクの文字列である場合、xmlparse の結果は空の XML ドキュメントになります。
- xmlparse() では *general_string_expression* が XML ドキュメントとして解析され、解析済みドキュメントを含む image 値が返されます。
- *general_string_expression* が image 式の場合は、サーバ文字セットの文字で構成されると見なされます。

注意　次の項目については、「第 3 章 XML 言語と XML 問い合わせ言語」を参照してください。

- 外部 URI 参照、XML ネームスペース、XML スキーマに対する制限。
- 定義済みエンティティと対応する文字列 *&* (&)、*<* (,)、*>* (>)、*&quote;* (“)、*'* (;) の処理方法。エンティティにはセミコロンを含めます。
- スペースの処理方法。
- 空の要素の処理方法。

オプション

- option_string の一般的なフォーマットについては「option_strings：一般的なフォーマット」(36 ページ) を参照してください。xmlparse 関数でサポートされるオプションは次のとおりです。

  dtdvalidate = {*yes* | *no*}

  xmlerror = {*exception* | *null* | *message* }

  dtdvalidate=yes と指定した場合、XML ドキュメントは埋め込み DTD (存在する場合) に対して検証されます。

  dtdvalidate=no と指定した場合、DTD 検証は実行されません。これがデフォルト値です。

  xmlerror = {*exception* | *null* | *message*}

  xmlerror オプションについては、次の「例外」を参照してください。

例外

xml_data_expression の値が有効な XML ではない場合、次のようになります。

- 明示的なオプションまたはデフォルトのオプションで xmlerror=exception と指定した場合は、例外が発生します。
- 明示的なオプションまたはデフォルトのオプションで xmlerror=null と指定した場合は、null 値が返されます。
- 明示的なオプションまたはデフォルトのオプションで xmlerror=message と指定した場合は、3 つの例外メッセージを伴う XML 要素を含む文字列が返されます。この値は有効な解析済み XML です。
- グローバル変数 *@@error* は、xmlerror の値が exception、null、または message のいずれであっても、最後のエラーのエラー番号を返します。

xml_data_expression の値が有効な XML ではない場合、次のようになります。

- 明示的なオプションまたはデフォルトのオプションで xmlerror=exception と指定した場合は、例外が発生します。
- 明示的なオプションまたはデフォルトのオプションで xmlerror=null と指定した場合は、null 値が返されます。
- 明示的なオプションまたはデフォルトのオプションで xmlerror=message と指定した場合は、例外メッセージを伴う XML 要素を含む文字列が返されます。この値は有効な解析済み XML です。

例

以下の例は、付録 A に記載されている *sample_docs* テーブルを使用しています。

sample_docs テーブルが作成され、初期化されたときには、*text_doc* カラムにはドキュメントが含まれ、*image_doc* カラムは null になっています。*image_doc* カラムを更新すると、*text_doc* カラムに解析済み XML バージョンを含めることができます。

```
update sample_docs
set image_doc = xmlparse(text_doc)

(3 rows affected)
```

このあとに、text カラムの未解析 XML ドキュメントに適用するのと同じ方法で、xmlextract 関数を image カラムの解析済み XML ドキュメントに適用できます。通常、解析済み XML ドキュメントに対するオペレーションは、未解析 XML ドキュメントに対するオペレーションよりも速く実行されます。

```
select name_doc,
  xmlextract('/bookstore/book[title="History of Trenton"]/price', text_doc)
    as extract_from_text_doc,
  xmlextract('/bookstore/book[title="History of Trenton"]/price', image_doc)
    as extract_from_image_doc
from sample_docs

name_doc   extract_from_text_doc  extract_from_image_doc
---------  --------------------  ----------------------
bookstore   <price>55</price>     <price>55</price>
publishers  NULL                  NULL
titles      NULL                  NULL
(3 rows affected)
```

xmlerror オプションを示すために、次のコマンドでは *sample_docs* テーブルに無効なドキュメントを挿入します。

```
insert into sample_docs (name_doc, text_doc) ,
values ('invalid doc', '<a>unclosed element<a>')

(1 row affected)
```

次の例では、xmlerror オプションが xmlparse 関数による無効な XML ドキュメントの処理方法を判断します。

- xmlerror=exception (デフォルト) の場合は、例外が発生します。

```
update sample_docs
set image_doc = xmlparse(text_doc option 'xmlerror=exception')

Msg 14702, Level 16, State 0:
Line 2:
XMLPARSE(): XML parser fatal error
  <<The input ended before all started tags were ended. Last tag started
was 'a'>> at line 1, offset 23.
```

- xmlerror=null の場合は、null 値が返されます。

```
update sample_docs
set image_doc = xmlparse(text_doc option 'xmlerror=null')

select image_doc from sample_docs
where name_doc='invalid doc'
------
NULL
```

- xmlerror=message の場合は、解析済み XML ドキュメントがエラー・メッセージとともに返されます。

```
update sample_docs
set image_doc = xmlparse(text_doc option 'xmlerror=message')

select xmlextract('/', image_doc)
from sample_docs
where name_doc = 'invalid doc'
------------------------
<xml_parse_error>The input ended before all started tags were ended.
Last tag started was 'a'</xml_parse_error>
```

次のコマンドは、sample_docs テーブルを元の状態にリストアします。

```
delete from sample_docs
where name_doc='invalid doc'
```

xmlrepresentation

image パラメータを調べて、パラメータに解析済み XML データが含まれるか、他の種類の image データが含まれるかを示す整数値を返します。

構文

xmlrepresentation_call::=
xmlrepresentation(*parsed_xml_expression*)

説明

- *parsed_xml_expression* は、データ型が image、binary、または varbinary の *sql_query_expression* です。
- xmlrepresentation() のパラメータが null である場合、呼び出しの結果は null になります。
- xmlrepresentation は、オペランドが解析済み XML データの場合は整数の 0 を返し、オペランドが未解析の XML データか、すべてブランクまたは空の文字列の場合は正の整数を返します。

例

以下の例は、「付録 A sample_docs サンプル・テーブル」に記載されている *sample_docs* テーブルを使用しています。

例 1 次の例は、基本的な xmlrepresentation 関数を示します。

```
-- Return a non-zero value
-- for a document that is not parsed XML
select xmlrepresentation(
        xmlextract('/', '<a>A</a>' returns image)

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

```
-- Return a zero for a document that is parsed XML
select xmlrepresentation(
        xmlparse(
          xmlextract('/', '<a>A</a>' returns image))
-----------
0
```

例 2　データ型が image のカラムには、解析済み XML ドキュメント (xmlparse 関数で生成) と未解析 XML ドキュメントの両方を含めることができます。次の例の update コマンドの実行後、sample_docs テーブルの image_doc カラムには、titles ドキュメントに対しては解析済み XML ドキュメント、bookstore ドキュメントに対しては未解析 (文字列) XML ドキュメント、publishers ドキュメント (元の値) に対しては null が含まれます。

```
update sample_docs
set image_doc = xmlextract('/', text_doc returns image)
where name_doc = 'bookstore'

update sample_docs
set image_doc = xmlparse(text_doc)
where name_doc = 'titles'
```

例 3　次のように、xmlrepresentation 関数を使用して、image カラムの値が解析済み XML ドキュメントであるかどうかを判断できます。

```
select name_doc, xmlrepresentation(image_doc)from sample_docs

name_doc
---------     -----------
bookstore     1
publishers    NULL
titles        0

(3 rows affected)
```

例 4　image カラムを更新し、そのすべての値を解析済み XML ドキュメントに設定できます。image カラムに解析済み XML ドキュメントと未解析 XML ドキュメントが混在している場合は、単純な更新では例外が発生します。

```
update sample_docs set image_doc = xmlparse(image_doc)
Msg 14904, Level 16, State 0:
Line 1:
XMLPARSE: Attempt to parse an already parsed XML document.
```

例 5　このような例外は、次のように xmlrepresentation 関数を使用することで回避できます。

```
update sample_docs
set image_doc = xmlparse(image_doc)
where xmlrepresentation(image_doc) != 0

(1 row affected)
```

例 6 次のコマンドは、sample_docs テーブルを元の状態にリストアします。

```
update sample_docs
set image_doc = null
```

xmlvalidate

XML ドキュメントを検証します。

構文

xmlvalidate_call ::=
xmlvalidate (*general_string_expression*, [*optional_parameters*])
optional_parameters ::= *options_parameter*
| *returns_type*
|*options_parameter* returns type
options_parameter ::= [,] option *option_string*
options_string ::= *basic_string_expression*
returns_type ::= [,] returns *string_type*
string_type ::=*char* (*integer*) | *varchar* (*integer*)
| *unichar* (*integer*) |*univarchar* (*integer*)
| *text* | *unitext* |*image* | *java.lang.String*

説明

注意 Unicode の検証については、「第 6 章 XML における国際化 (I18N) のサポート」を参照してください。

- *basic_string_expression* は、データ型が character、varchar、unichar、univarchar、または java.lang.String の *sql_query_expression* です。
- *general_string_expression* は、データ型が character、varchar、unichar、univarchar、text、unitext、または java.lang.String の *sql_query_expression* です。
- xmlvalidate() のいずれかのパラメータが null である場合、呼び出し結果は null になります。
- xmlvalidate_call の結果データ型は *returns_type* で指定されたデータ型になります。

オプション

option_string の一般的なフォーマットについては「option_strings：一般的なフォーマット」(36 ページ) を参照してください。xmlvalidate でサポートされるオプションは次のとおりです。

validation_options::=
[dtdvalidate = {no | yes | strict}]
[schemavalidate = {no | yes}]
[nonamespaceschemalocation = '*schema_uri_list*']
[schemalocation = '*namespace_schema_uri_list*']
[xmlerror = {exception | null | message }]
[xmlvalid = {document | message}]
schema_uri_list::=
schema_uri [*schema_uri*]...
namespace_schema_uri_list ::=
namespacename schema_uri
[*namespacename schema_uri*]...
schema_uri ::= *character_string*
namespacename ::= *character_string*

オプションの説明

- *validation_options* のデフォルトは次のとおりです。

  dtdvalidate = 以下を参照。
  schemavalidate = *no*
  schemalocation = " "
  nonamespaceschemalocation = " "
  xmlerror = exception
  xmlvalid = document

- *validation_option* のキーワードでは大文字と小文字が区別されませんが、*schema_uri_list* と *namespace_schema_uri_list* では大文字と小文字が区別されます。
- 解析または格納するドキュメントをサブジェクト XML ドキュメントと呼びます。
- dtdvalidate のデフォルトは、schemavalidate オプションの暗黙的または明示的な値によって異なります。schemavalidate オプションの値が *no* である場合は、dtdvalidate のデフォルト値は *no* になります。schemavalidate オプション値が *yes* である場合は、dtdvalidate オプションのデフォルト値は *strict* になります。
- schemavalidate = *yes* と指定した場合は、dtdvalidate = *strict* と指定するか、dtdvalidate を省略してください。
- dtdvalidate = *no* および schemavalidate = *no* と指定した場合は、ドキュメントが正しい形式であるかどうかだけがチェックされます。
- schemavalidate = *no* と指定した場合、nonamespaceschemalocation 句および schemalocation 句は無視されます。

- nonamespaceschemalocation 句と schemalocation 句に指定する値は、文字リテラルです。Transact-SQL quoted_identifier オプションが *off* である場合は、アポストロフィ (') と引用符 (") の一方を使用して *option_string* を囲み、他方を使用して nonamespacescemalocation および schemalocation で指定する値を囲みます。Transact-SQL quoted_identifier オプションが *on* である場合は、*option_string* をアポストロフィ (') で囲み、nonamespacescemalocation および schemalocation で指定する値を引用符 (") で囲む必要があります。
- nonamespaceschemalocation には、スキーマ URI のリストを指定します。このリストは、サブジェクト XML ドキュメントの xsi:noNameSpaceschemalocation 句で指定されたスキーマ URI のリストを上書きします。
- schemalocation には、namespace 名とスキーマ URI のペアのリストを指定します。
  - namespace 名は、xmlns 属性が namespace に対して指定する名前です。次の例では、*http://acme.com/schemas.contract* がデフォルトの namespace として宣言されています。

```
<contract xmlns="http://acme.com/schemas.contract">
```

これに対し、次の例では、プレフィクス "co" の namespace として宣言されています。

```
<co:contract xmlns:co="http://acme.com/schemas.contract">
```

namespace 名は、プレフィクスではなく、namespace 宣言そのものによって指定される URI です。

- *schema_uri* は、スキーマ URI を含むリテラル文字列です。*URI_string* の最大長は 1927 文字で、*http* を指定する必要があります。*schema_uri* によって参照されるスキーマは UTF8 または UTF16 としてコード化してください。
- dtdvalidate オプション値は次のとおりです。

  dtdvalidate=*no*：DTD またはスキーマの検証は実行されません。ドキュメントの形式が正しいかどうかがチェックされます。

  dtdvalidate=*yes*：ドキュメントによって指定された任意の DTD に照らしてドキュメントが検証されます。

  dtdvalidate=*strict*：このオプションは、schemavalidate オプションに依存します。
  - schemavalidate=*no*：
    DTD をサブジェクト XML ドキュメントで指定する必要があります。その DTD に照らしてドキュメントが検証されます。

- schemavalidate=*yes*：
  サブジェクト XML ドキュメントのすべての要素を DTD またはスキーマ内で宣言する必要があります。それらの宣言に照らして各要素が検証されます。

- schemavalidate オプション値は次のとおりです。

  schemavalidate=*no* と指定した場合は、サブジェクト XML ドキュメントに対してスキーマ検証が実行されません。

  schemavalidate=*yes* と指定した場合は、スキーマ検証が実行されます。

- *general_string_expression* (たとえば *XC*) が、*option_string* 句で指定された検証オプションを渡す XML ドキュメントである場合、結果は次のようになります。

  xmlvalid で doc と指定したときは、xmlvalidate の結果は次のようになります。

  ```
  convert(text, XC)
  ```

  xmlvalid で message と指定したときは、xmlvalidate の結果は次の XML ドキュメントになります。

  ```
  <xmlvalid/>
  ```

- *general_string_expression* が、*option_string* 句で指定された検証オプションを渡す XML ドキュメントでない場合、結果は次のようになります。

  option_string で xmlerror=*exception* と指定した場合は、例外メッセージを伴う例外が発生します。

  option_string で xmlerror=message と指定した場合は、次の形式の XML ドキュメントが返されます。E1、E2 などは、検証エラーを説明するメッセージです。

```
<xml_validation_errors>
     <xml_validation_error>E1</xml_validation_error>
     <xml_validation_error>E2</xml_validation_error>
...
     <xml_validation_warning>W1</xml_validation_warning>
     <xml_vvalidation_fatal_error>E3<xml_validation_fatal_error>
</xml_validation_errors>
```

option_string で xmlerror=*null* と指定した場合は、null 値が返されます。

例外

xml_data_expression の値が有効な XML ではない場合、次のようになります。

- 明示的なオプションまたはデフォルトのオプションで xmlerror=exception と指定した場合は、例外が発生します。
- 明示的なオプションまたはデフォルトのオプションで xmlerror=null と指定した場合は、null 値が返されます。
- 明示的なオプションまたはデフォルトのオプションで xmlerror=message と指定した場合は、すべての例外メッセージを伴う XML 要素を含む文字列が返されます。この値は有効な解析済み XML です。
- グローバル変数 *@@error* は、xmlerror の値が exception、null、または message のいずれであっても、最後のエラーのエラー番号を返します。
- 検証に必要な Web リソースを使用できない場合は、例外が発生します。
- ソース XML ドキュメントが無効であるか、正しい形式でない場合は、例外が発生します。メッセージには、検証の失敗が示されます。

例

表 2-2 の XML DTD および XML スキーマは検証句を示しています。

- *dtd_emp* と *schema_emp* は、単一のテキスト要素である "<*emp_name*>" を定義します。
- *dtd_cust* と *schema_cust* は、単一のテキスト要素である "<*cust_name*>" を定義します。
- *ns_schema_emp* と *ns_schema_cust* は、ターゲット namespace を指定する変形です。

表 2-2: DTD とスキーマの例および URI

URI	ドキュメント
http://test/dtd_emp.dtd	<!ELEMENT emp_name (#PCDATA)>
http://test/dtd_cust.dtd	<!ELEMENT cust_name (#PCDATA)>
http://test/schema_emp.xsd	<xsd:schema xmlns:xsd ="http://www.w3.org/2001/XMLSchema" targetNamespace ="http://test/ns_schema_emp"> <xsd:element name="emp_name" type="xsd:string"/> </xsd:schema>
http://test/ns_schema_emp.xsd	<xsd:schema xmlns:xsd ="http://www.w3.org/2001/XMLSchema" targetNamespace ="http://test/ns_schema_emp"> <xsd:element name="emp_name" type="xsd:string"/> </xsd:schema>

URI	ドキュメント
http://test/schema_cust.xsd	<xsd:schema xmlns:xsd ="http://www.w3.org/2001/XMLSchema"> <xsd:element name="cust_name" type="xsd:string"/> </xsd:schema>
http://test/ns_schema_cust.xsd	<xsd:schema xmlns:xsd ="http://www.w3.org/2001/XMLSchema" targetNamespace ="http://test/ns_schema_cust"> <xsd:element name="cust_name" type="xsd:string"/> </xsd:schema>

例 1　次の例は、XML ドキュメントを text カラムに格納するテーブルを作成します。このテーブルを使用して、xmlvalidate の呼び出し例を示します。つまり、xmlvalidate は、text カラムに格納されたドキュメントを明示的に検証します。

```
create table text_docs(xml_doc text null)
```

例 2　次の例は、DTD 宣言のないドキュメントと、検証オプション *dtdvalidate=yes* を指定する xmlvalidate を示しています。挿入されたドキュメントの形式が正しく、dtdvalidate が *strict* として指定されていないため、このコマンドは成功します。

```
insert into text_docs
values (xmlvalidate(
  '<employee_name>John Doe</employee_name>',
  option 'dtdvalidate=yes'))
---------
(1 row inserted)
```

例 3　次の例は、DTD 宣言のないドキュメントと、検証オプション *dtdvalidate=strict* を指定する xmlvalidate を示しています。厳密な DTD 検証では、ドキュメントの各要素が DTD によって指定されている必要があるため、xmlvalidate で例外が発生します。

```
insert into text_docs
values(xmlvalidate(
  '<emp_name>John Doe</emp_name>',
option 'dtdvalidate=strict'))
--------
EXCEPTION
```

例 4 最後の例では、検証が失敗したときに例外が発生しました。代わりに、オプション xmlerror を使用すると、検証が失敗したときに xmlvalidate は null を返します。

```
insert into text_docs
values(xmlvalidate(
  '<emp_name>John Doe</emp_name>'
option 'dtdvalidate=strict xmlerror=null'))
-------
null
```

例 5 xmlerror を使用すると、検証が失敗したときに、xmlvalidate が XML エラー・メッセージを XML ドキュメントとして返すように指定することもできます。

```
insert into text_docs
values(xmlvalidate(
  '<emp_name>John Doe</emp_name>'
option 'dtdvalidate=strict xmlerror=message'))
--------
<xml_validation_errors>
<xml_validation_error>(1:15)Document is invalid:
  no grammar found.<xml_validation_error>
<xml_validation_error>(1:15)Document root element
   "employee name",must match DOCTYPE root
   "null."</xml_validation_error>
</xml_validation_errors>
```

例 6 次の例は、DTD と検証オプション *dtdvalidate=yes* の両方を参照するドキュメントを指定する xmlvalidate を示しています。このコマンドは成功します。

```
insert into text_docs
values(xmlvalidate(
  '<DOCTYPE emp_name PUBLIC "http://test/dtd_emp.dtd">
  <emp_name>John Doe</emp_name>',
option 'dtdvalidate=yes'))
-------
(1 row inserted)
```

例 7 この例は、DTD を参照するドキュメントと、検証オプション *dtdvalidate=yes* を指定する xmlvalidate を示しています。挿入されたドキュメントとドキュメント内で参照されている DTD が一致しないため、例外が発生します。

```
insert into text_docs
values(xmlvalidate(
  '<DOCTYPE emp_name PUBLIC "http://test/dtd_cust.dtd">
<emp_name>John Doe</emp_name>',
option 'dtdvalidate=yes'))
--------
EXCEPTION
```

例 8　次の例は、スキーマ宣言のないドキュメントと、検証オプション *schemavalidate=yes* を指定する xmlvalidate を示しています。'<emp_name>' 要素に宣言がないため、このコマンドは失敗します。

```
insert into text_docs
values(xmlvalidate('<emp_name>John Doe</emp_name>',
  option 'schemavalidate=yes'))
-------
EXCEPTION
```

例 9　次の例は、スキーマ宣言を含むドキュメントと、検証オプション *schemavalidate=yes* を指定する xmlvalidate を示しています。このドキュメントは、ネームスペースを使用しません。ドキュメントと、ドキュメント内で参照されているスキーマが一致するため、コマンドは成功します。

```
insert into text_docs
values(xmlvalidate(
  '<emp_name xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi: noNamespaceSchemaLocation="http://test/schema_emp.xsd">
  John Doe</emp_name>'
 option 'schemavalidate=yes'))
--------
(1 row inserted)
```

例 10　次の例は、ネームスペースを指定するドキュメントと、検証オプション *schemavalidate=yes* を指定する xmlvalidate を示しています。ドキュメントと、ドキュメント内で参照されているスキーマが一致するため、コマンドは成功します。

```
insert into text_docs
values(xmlvalidate(
  '<emp:emp_name xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:emp="http://test/ns_schema_emp"
   xsi: SchemaLocation="http://test/ns_schema_emp
      http://test/ns_schema_emp.xsd">
  John Doe</emp:emp_name>'
 option 'schemavalidate=yes'))
--------
(1 row inserted)
```

例 11　次の例は、スキーマ宣言を含むドキュメントと、検証オプション *schemavalidate=yes* を指定する xmlvalidate を示しています。ドキュメントとドキュメント内で参照されているスキーマが一致しないため、このコマンドは失敗します。

```
insert into text_docs
values (xmlvalidate(
  '<emp_name xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="http://test/schema_cust.xsd">
   John Doe</emp_name>'
 option 'schemavalidate=yes'))
-------
EXCEPTION
```

例 12 次の例は、スキーマ宣言を含むドキュメントと、検証オプション *schemavalidate=yes* を指定する xmlvalidate を示しています。このドキュメントは、ネームスペースを指定しています。ドキュメントとドキュメント内で参照されているスキーマが一致しないため、コマンドは失敗します。

```
insert into text_docs
values(xmlvalidate(
  '<emp:emp_name
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:emp="http://test/ns_schema_cust"
    xsi:schemaLocation=
       "http://test/ns_schema_cust http://test/ns_schema_cust.xsd">
    John Doe</emp:emp_name>',
   option 'schemavalidate=yes'))
------------
EXCEPTION
```

xmlvalidate の検証オプションは、*http://test/ns_schema_emp.xsd* の nonamespaceschemalocation を指定します。

例 13 次の例は、スキーマ宣言を含むドキュメントと検証オプション *schemavalidate=yes*、およびschemalocation 句と nonamespaceschemalocation を指定する xmlvalidate を示しています。

このドキュメントは *http://test/schema_cust.xsd* の schemaLocation を指定し、xmlvalidate の検証オプションは *http://test/ns_schema_emp.xsd* の schemalocation を指定します。

ドキュメントと xmlvalidate 内で参照されているスキーマが一致し、それによってドキュメント内で参照されているスキーマが上書きされるため、このコマンドは成功します。

```
insert into text_docs
values (xmlvalidate(
  '<emp:emp_name
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:emp="http://test/schema_emp"
    xsi:schemaLocation="http://test/ns_schema_emp
         http://test/schema_cust.xsd">
    John Doe</emp:emp_name>',
   option 'schemavalidate=yes,
    schemalocation= "http://test/ns_schema_emp
         http://test/ns_schema_emp.xsd"
    nonamespaceschemalocation="http://test/schema_emp.xsd" '))
-------------
(1 row inserted)
```

例 14　次の例は、スキーマ宣言を含むドキュメントと検証オプション *schemavalidate=yes*、および schemalocation 句と nonamespaceschemalocation 句を指定する xmlvalidate を示しています。

このドキュメントは *http://test/schema_cust.xsd* の noNamespaceSchemaLocation を指定し、xmlvalidate の検証オプションは *http://test/ns_schema_emp.xsd* の nonamespaceschemalocation を指定します。

ドキュメントと xmlvalidate 内で参照されているスキーマが一致しないため、このコマンドは失敗します。ただし、このドキュメントは、ドキュメント内で参照されているスキーマとは一致しています。

```
insert into text_docs
values(xmlvalidate(
  '<customer_name
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="http://test/schema_cust.xsd">
  John Doe</customer_name>'
  option 'schemavalidate=yes,
schemalocation="http://test/ns_schema_emp http://test/ns_schema_emp.xsd"
    nonamespaceschemalocation="http://test/schema_emp.xsd" '))
-----------
EXCEPTION
```

例 15　次の例は、スキーマ宣言を含むドキュメントと検証オプション *schemavalidate=yes*、および schemalocation 句と nonamespaceschemalocation 句を指定する xmlvalidate を示しています。

このドキュメントは *http://test/schema_cust.xsd* の schemaLocation を指定し、xmlvalidate の検証オプションは *http://test/ns_schema_emp.xsd* の *schemalocation* を指定します。

ドキュメントと xmlvalidate 内で参照されているスキーマが一致しないため、このコマンドは失敗します。ただし、このドキュメントは、ドキュメント内で参照されているスキーマとは一致しています。

```
insert into text_docs
values(xmlvalidate(
  '<cust:cust_name
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:cust="http://test/schema_cust"
    xsi:schemaLocation="http://test/schema_cust
          http://test/schema_cust.xsd">
   John Doe</cust:cust_name>',
  option 'schemavalidate=yes,
   schemalocation="http://test/ns_schema_emp
http://test/ns_schema_emp.xsd"
   nonamespaceschemalocation="http://test/schema_emp.xsd" '))
-------
(1 row inserted)
```

option_strings：一般的なフォーマット

この項では、XML サービスのオプション文字列パラメータの一般的なフォーマット、構文、処理を示します。個々のオプションの動作は、そのオプションを参照する関数で説明しています。

option_string パラメータを持つ関数は、すべてのオプションの union を受け入れ、その関数に適用されないオプションを無視します。

この「union オプション」アプローチでは、すべての XML サービス関数に対して 1 つの *option_string* 変数を使用できます。

構文

option_string::= *basic_string_expression*

説明

- *option_string* パラメータのランタイム値の完全な構文は次のとおりです。

  option_string_value ::= *option* [[,] *option*] ...
  option ::= *name* = *value*
  name ::= option name as listed below
  value ::= *simple_identifier* | *quoted_string*

- option_string パラメータが null の場合、空の文字列はすべてブランクになります。
- 最初の option の前、最後の option の後ろ、option の間、等号の前後には、任意の数のスペースを使用できます。
- 各 options は、カンマまたはスペースで区切ることができます。
- *option_value* には、先頭が文字で後ろに文字、数字、アンダースコアが続く単純な識別子、または引用符で囲まれた文字列を使用できます。引用符で囲まれた文字列には、埋め込み引用符に関する通常の SQL 表記規則を使用します。
- option のセットとそれらを適用できる関数を表 2-3 に示します。オプションの説明については、各関数の説明を参照してください。

クエリ関数のオプション値

注意 アンダーラインは、この表のキーワードを指定する option のデフォルト値を示します。カッコは、SQL 名を指定する option のデフォルト値を示します。空の文字列またはシングル・スペース文字は、文字列値を指定する option のデフォルト値を指定します。

表 2-3: オプション文字列の値

オプション名	オプションの値	関数
binary	hex | base64	for_xml 句
columnstyle	element | attribute	for xml 句
dtdvalidate	yes | no	xmlvalidate
entitize	yes | no | conditional	for xml 句
format	yes | no	for xml 句
header	yes | no |encoding	for xml 句
nonamespaceschemalocation	「xmlvalidate」 を参照。	xmlvalidate
ncr	non_ascii | non_server | no デフォルト値については、関数の説明を参照。	for xml 句、xmlextract
nullstyle	attribute | omit	for xml 句
prefix	SQL 名 (C) デフォルト値は C です。	for_xml 句
root	yes | no	for xml 句
rowname	SQL 名 (row)	for_xml 句
schemalocation	「xmlvalidate」 を参照。	for xml 句
schemavalidate	yes | no	xmlvalidate
statement	yes | no	forxml 句
tablename	SQL 名 (resultset)	for xml 句
targetns	引用符で囲まれた URI 文字列	for_xml 句
xmlerror	exception | null | message	XML オペランドを伴うすべての関数
xmlvalid	document | message	xmlvalidate
xsidecl	yes | no	for xml 句

第 3 章

XML 言語と XML 問い合わせ言語

XML クエリ関数は、XML ドキュメントに対しては XML 1.0 標準、XML クエリに対しては XPath 1.0 標準をサポートしています。この章では、XML サービスでサポートされるこれらの標準のサブセットについて説明します。

トピック名	ページ
文字セットのサポート	39
URI のサポート	39
ネームスペースのサポート	40
XML スキーマのサポート	40
XML ドキュメントの定義済みエンティティ	40
XPath クエリの定義済みエンティティ	41
スペース	42
空の要素	42
XML 問い合わせ言語	43
カッコで囲んだ式	51

文字セットのサポート

XML サービスは、SQL サーバでサポートされている文字セットをサポートします。I18N の詳細については、「第 6 章 XML における国際化 (I18N) のサポート」を参照してください。

URI のサポート

XML ドキュメントは、2 つのコンテキスト、つまり、href 属性またはドキュメント・テキストというコンテキストと、DTD の外部参照、エンティティ定義、XML スキーマ、ネームスペース宣言というコンテキストで URI (Universal Resource Indicator) を指定します。

href 属性またはドキュメント・テキストとしての URI の使用には制限はなく、XML サービスは http URL を指定する外部参照 URL を解決します。

file、ftp、または relative URI を指定する外部参照 URI はサポートされていません。

ネームスペースのサポート

ネームスペース宣言と参照を含む XML ドキュメントは、無制限に解析と格納ができます。

ただし、ネームスペース・プレフィクスを持つ XML 要素と属性名が xmlextract と xmltest の XML 式で参照されている場合、ネームスペース・プレフィクスとコロンはその要素名または属性名の一部として扱われます。これらはネームスペース参照としては処理されません。

XML スキーマのサポート

xmlvalidate については、「第 7 章 xmltable()」を参照してください。

XML ドキュメントの定義済みエンティティ

引用符 (")、アポストロフィ (')、より小さい (<)、より大きい (>)、アンパサンド (&) の特殊文字は、XML の句読表記に使用され、それぞれ *"*、*'*、*<*、*>*、*&* という定義済みエンティティで表されます。セミコロンがエンティティに含まれることに注意してください。

次の一連の例で示すように、属性や要素内では "<" または "&" を使用できません。

```
select xmlparse("<a atr='<'/>")

Msg 14702, Level 16, State 0:
Line 1:
XMLPARSE(): XML parser fatal error <<A '<' character cannot be
used in attribute 'atr', except through <&gt;> at line 1,
offset 14.

select xmlparse("<a atr1='&'>")

Msg 14702, Level 16, State 0:
Line 1:
XMLPARSE(): XML parser fatal error
<<Expected entity name for reference>>
at line 1, offset 11

select xmlparse("<a> < </a>")

Msg 14702, Level 16, State 0:
Line 2:
XMLPARSE(): XML parser fatal error
```

```
<<Expected an element name>>
at line 1, offset 6.

select xmlparse(" & ")
Msg 14702, Level 16, State 0:
Line 1:
XMLPARSE(): XML parser fatal error
<<Expected entity name for reference>>
at line 1, offset 6.
```

代わりに、次のように定義済みエンティティ *<* と *&* を使用します。

```
select xmlextract("/",
    "<a atr='&lt; &amp;'> &lt; &amp; </a>" )
--------------------------------
          <a atr="&lt; &amp;"> &lt; &amp; </a>
```

アポストロフィで区切られた属性内で引用符を使用でき、また引用符で区切られた属性内でアポストロフィを使用できます。これらの記号は、定義済みエンティティ *"* または *'* に置き換えられます。次の例では、SQL 文字リテラル規則に従うために、'yes' という単語を囲む引用符またはアポストロフィを二重にしています。

```
select xmlextract("/", "<a atr=' ""yes"" '/> " )
--------------------------------
           <a atr=" "yes" "></a>

select xmlextract('/', '<a atr=" ''yes'' "/> ' )
----------------------------
            <a atr=" 'yes' "></a>
```

要素内では引用符とアポストロフィを使用できます。これらは、次の例で示すように、定義済みエンティティ *"* と *'* に置き換えられます。

```
select xmlextract("/", " ""yes"" and 'no' " )
-------------------------------------
          &quot;yes&quot; and 'no'
```

XPath クエリの定義済みエンティティ

XML 特殊文字を含む文字リテラルで XML クエリを指定する場合は、これらをプレーンな文字または定義済みエンティティとして記述できます。次の例は、2 つのポイントを示します。

- XML ドキュメントには、要素 "<a>" が含まれます。その値は XML 特殊文字 &<>" であり、定義済みエンティティ *&<>"* で表されます。

- XML クエリは、同じ XML 特殊文字を持つ文字リテラルを指定します。これも定義済みエンティティで表されます。

```
select xmlextract('/a="&amp;&lt;&gt;&quot;"',
      "<a>&amp;&lt;&gt;&quot;</a>")
---------------------------------
          <a>&amp;&lt;&gt;&quot;</a>
```

次の例も同じですが、XML クエリがプレーンな XML 特殊文字で文字リテラルを指定している点が異なります。これらの XML 特殊文字は、クエリが評価される前に定義済みエンティティに置き換えられます。

```
select xmlextract("/a='&<>""' " ,
            "<a>&amp;&lt;&gt;&quot;</a>")
---------------------------------
          <a>&amp;&lt;&gt;&quot;</a>
```

スペース

すべてのスペースは保持され、クエリで重要な意味を持ちます。

```
select xmlextract("/a[@atr='  this  or  that  ' ]",
     "<a atr='  this  or  that  '><b>  which  or  what
      </b></a>")
-------------------------------------------------
     <a atr="  this  or  that  ">
     <b>  which  or  what  </b></a>

select xmlextract("/a[b='  which  or  what  ']",
      "<a atr='  this  or  that  '><b>  which  or  what
       </b></a>")
-----------------------------------------
      <a atr='  this  or  that  '>
       <b>  which  or  what  </b></a>
```

空の要素

"<a/>" のスタイルで入力される空の要素は、"<a>" のスタイルで格納され、返されます。

```
select xmlextract("/",
       "<doc><a/> <b></b></doc>")
---------------------------------------
         <doc>
         <a></a>
         <b></b></doc>
```

XML 問い合わせ言語

XML サービスは、標準の XPath 言語のサブセットをサポートしています。そのサブセットは、次の項の構文とトークンによって定義されます。

XPath でサポートされる構文とトークン

XML サービスは、次の XPath 構文をサポートしています。

```
xpath::= or_expr
or_expr::= and_expr | and_expr TOKEN_OR or_expr
and_expr::= union_expr | union_expr TOKEN_AND and_expr
union_expr::= intersect_expr
      | intersect_expr TOKEN_UNION union_expr
intersect_expr::= comparison_expr
      | comparison_expr TOKEN_INTERSECT intersect_expr
comparison_expr::= range_exp
      | range_expr general_comp comparisonRightHandSide
general_comp::= TOKEN_EQUAL | TOKEN_NOTEQUAL
      | TOKEN_LESSTHAN | TOKEN_LESSTHANEQUAL
      | TOKEN_GREATERTHAN | TOKEN_GREATERTHANEQUAL
range_expr::= unary_expr | unary_expr TOKEN_TO unary_expr
unary_expr::= TOKEN_MINUS path_expr
      | TOKEN_PLUS path_expr
      | path_expr
comparisonRightHandSide::= literal
path_expr::= relativepath_expr | TOKEN_SLASH
      | TOKEN_SLASH relativepath_expr
      | TOKEN_DOUBLESLASH relativepath_expr
relativepath_expr::= step_expr
      | step_expr TOKEN_SLASH relativepath_expr
      | step_expr TOKEN_DOUBLESLASH relativepath_expr
step_expr::= forward_step predicates
      | primary_expr predicates
      | predicates
primary_expr::= literal | function_call | (xpath)
function_call::=
      tolower([xpath])
      | toupper([xpath])
      | normalize-space([xpath])
      | concat([xpath [,xpath]...])
forward_step::= abbreviated_forward_step
abbreviated_forward_step::= name_test
      | TOKEN_ATRATE name_test
      | TOKEN_PERIOD
name_test::= q_name | wild_card | text test
text_test ::= TOKEN_TEXT TOKEN_LPAREN TOKEN_RPAREN
literal::= numeric_literal | string_literal
wild_card::= TOKEN_ASTERISK
q_name::= TOKEN_ID
string_literal::= TOKEN_STRING
numeric_literal::= TOKEN_INT | TOKEN_FLOATVAL|
      | TOKEN_MINUS TOKEN_INT
      | TOKEN_MINUSTOKEN_FLOATVAL
predicates::=
      | TOKEN_LSQUARE expr TOKEN_RSQUARE predicates
      | TOKEN_LSQUARE expr TOKEN_RSQUARE
```

次のトークンは、XPath の XML サービス・サブセットによってサポートされています。

```
APOS ::= '"'
DIGITS ::= [0-9]+
NONAPOS ::= '^"'
NONQUOTE ::= '^"'
NONSTART ::= LETTER | DIGIT | '.' | '-' | '_' | ':'
QUOTE ::= '"'
START ::= LETTER | '_'
TOKEN_AND ::= 'and'
TOKEN_ASTERISK  ::= '*'
TOKEN_ATRATE  ::= '@'
TOKEN_COMMA ::= ','
TOKEN_DOUBLESLASH  ::= '//'
TOKEN_EQUAL ::=  '='
TOKEN_GREATERTHAN ::=  '>'
TOKEN_GREATERTHANEQUAL ::=  '>='
TOKEN_INTERSECT ::=   'intersect'
TOKEN_LESSTHAN ::= '<'
TOKEN_LESSTHANEQUAL ::= '<='
TOKEN_LPAREN  ::= '('
TOKEN_LSQUARE  ::= '['
TOKEN_MINUS  ::= '-'
TOKEN_NOT ::= 'not'
TOKEN_NOTEQUAL ::= '!='
TOKEN_OR ::= 'or'
TOKEN_PERIOD  ::= '.'
TOKEN_PLUS  ::= '+'
TOKEN_RPAREN  ::= ')'
TOKEN_RSQUARE  ::= ']'
TOKEN_SLASH  ::= '/'
TOKEN_TO ::= 'to   '
TOKEN_UNION ::= '|'  | 'union'
TOKEN_ID  ::= START [NONSTART...]
TOKEN_FLOATVAL  ::= DIGITS | '.'DIGITS | DIGITS'.'DIGITS
TOKEN_INT  ::= DIGITS
TOKEN_STRING  ::=
      QUOTE NONQUOTE... QUOTE
    | APOS NONAPOS... APOS
TOKEN_TEXT ::=  'text'
```

XPath の演算子

この項では、XML プロセッサでサポートされる XPath サブセットを指定します。

XPath の基本演算子

表 3-1 は、サポートされる基本的な XPath 演算子を示します。

表 3-1: XPath の基本演算子

演算子	説明
/	パス (子) : 子演算子 ('/') は、左辺のコレクションの直下の子から選択する。
//	子孫：子孫演算子 ('//') は、左辺のコレクションの任意の子孫から選択する。
*	要素の子の収集：要素は、'*' コレクションで置換することで名前を使用しないで参照できる。
@	属性：属性名の前には '@' 記号が付く。
[]	フィルタ：コレクションにフィルタ句 '[ ]' を追加することにより、任意のコレクションに制約と分岐を適用できる。フィルタは、any セマンティックを指定した SQL の where 句に類似している。フィルタ内には、サブクエリと呼ばれるクエリが含まれる。コレクションがフィルタ内に配置された場合、コレクションに任意のメンバが含まれている場合はブール値 “true” が生成され、コレクションが空の場合には “false” が生成される。
[n]	インデックス : インデックスは、主にノードのセット内の特定ノードの検索に使用される。インデックスは角カッコで囲む。最初のノードがインデックス 1 である。
text()	現在のコンテキスト・ノードのテキスト・ノードを選択する。

XPath の集合演算子

表 3-2 (46 ページ) は、サポートされる XPath 集合演算子を示します。

表 3-2: XPath の集合演算子

演算子	説明
union |	共用体：union 演算子 (ショートカットは '|') は、左側のクエリと右側のクエリの値を結合したセットを返す。重複値はフィルタされ、結果リストはドキュメント順にソートされる。
intersect	積：intersect 演算子は、2 つの集合に共通する要素のセットを返す。
( )	グループ：カッコを使用すると、コレクション演算子をグループ化できる。
. (ドット)	ピリオド：ドット項は、検索コンテキストとの関連で評価される。この項は、この検索コンテキストの参照ノードのみを含むセットに評価される。
ブール演算子 (*and* と *or*)	ブール式は、サブクエリ内で使用できる。
and	ブール式の “and”。
or	ブール式の “or”。

XPath の比較演算子

表 3-3 は、サポートされる XPath 比較演算子を示します。

表 3-3: XPath の比較演算子

演算子	説明
=	等号
!=	不等号
<	より小さい
>	より大きい
>=	以下
<=	以上

XPath 関数

Adaptive Server は、次の XPath 文字列関数をサポートしています。

- toupper
- tolower
- normalize-space
- concat

一般的なガイドラインと例

この項では、XPath 式で関数を使用する際の一般的なガイドラインを示します。このガイドラインは前述のすべての関数に適用されます。以下の例はすべて tolower を使用します。この関数は 1 つの引数を小文字で返します。

ステップ式を使用するところでは、どこにでも関数呼び出しを使用できます。

例 1　XPath クエリの最上位として使用される関数は、最上位関数呼び出しと呼ばれます。次のクエリは、最上位関数呼び出しとして使用される tolower を示します。

```
select xmlextract
('tolower(//book[title="Seven Years in Trenton"]//first-name)', text_doc)
from sample_docs where name_doc='bookstore'
----------------------------------------
joe
```

最上位関数呼び出しのパラメータには絶対パス式を指定します。つまり、パラメータはスラッシュ (/) またはスラッシュ 2 つ (//) で始めます。

例 2　関数呼び出しのパラメータには、述部を含む複雑な XPath 式を指定できます。また、ネストした関数呼び出しにすることもできます。

```
select xmlextract
('//book[normalize-space(tolower(title))="seven years in trenton"]/author',
text_doc)
from sample_docs where name_doc='bookstore'

----------------------------------------
<author>
     <first-name>Joe</first-name>
     <last-name>Bob</last-name>
     <award>Trenton Literary Review
     Honorable Mention</award>
</author>
```

例 3　関数を相対ステップとして使用できます。相対ステップは相対関数呼び出しとも呼ばれます。次のクエリは、相対関数呼び出しとして使用される tolower を示します。

```
select xmlextract
( '//book[title="Seven Years in Trenton"]//tolower(first-name)', text_doc)
from sample_docs where name_doc='bookstore'

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

joe
```

この例は、相対関数のパラメータには相対パス式を指定する必要があることを示します。つまり、スラッシュ (/) またはダブル・スラッシュ (//) で始めることはできません。

例 4

最上位関数および相対関数は、どちらもパラメータにリテラルを使用できます。次に例を示します。

```
select xmlextract( 'tolower("aBcD")' ,text_doc),
    xmlextract( '/bookstore/book/tolower("aBcD")', text_doc)
from sample_docs where name_doc='bookstore'

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

 abcd      abcd
```

例 5

文字列関数は、そのパラメータのテキストに作用します。つまり、text() が暗黙的に適用されます。たとえば、次のクエリは first-name 要素を XML フラグメントとして返します。

```
select xmlextract
( '//book[title="Seven Years in Trenton"]//firstnamei, text_doc)
from sample_docs where name_doc='bookstore'

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

<first-name>Joe</first-name>
```

次のクエリは、first-name の XML フラグメントのテキストを返します。

```
select xmlextract
( '//book[title="Seven Years in Trenton"]//first-name/text()', text_doc)
from sample_docs where name_doc='bookstore'

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

Joe
```

次のクエリは first-name 要素に tolower を適用します。この関数は要素のテキストに暗黙的に作用します。

```
select xmlextract
('//book[title="Seven Years in Trenton"] //tolower(first-name)', text_doc)
from sample_docs where name_doc='bookstore'

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

joe
```

次の例は、明示的にパラメータとして XML 要素のテキストを渡します。これは前の例と同じ結果になります。

```
select xmlextract
( '//book[title="Seven Years in Trenton"]//tolower(first-name/text())',
text_doc)
from sample_docs where name_doc='bookstore'

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

joe
```

例 6

パスの 1 つのステップとして相対関数呼び出しを適用します。そのパスが評価されると XML ノードのシーケンスが生成され、各ノードの相対関数呼び出しが実行されます。その結果、関数呼び出し結果のシーケンスが生成されます。たとえば、次のクエリは first_name ノードのシーケンスを生成します。

```
select xmlextract( '/bookstore/book/author/first-name', text_doc)
from sample_docs where name_doc='bookstore'
--------------------------------
<first-name>Joe</first-name><first-name>Mary</first-name>
<first-name>Toni</first-name>
```

次のクエリは、前のクエリの最終ステップを toupper 呼び出しに置き換えて、両方の関数呼び出し結果のシーケンスを生成します。

```
select xmlextract('/bookstore/book/author/toupper(first-name)', text_doc)
from sample_docs where name_doc='bookstore'
--------------------------------
JOEMARYTONI
```

ここで、concat を使用して関数結果のシーケンスを区切ることができます。「concat」(51 ページ) の例を参照してください。

例 7

tolower、toupper、normalize-space のパラメータはどれも 1 つです。相対関数呼び出しにこれらの関数を指定するときにパラメータを省略すると、現在のノードが暗黙のパラメータになります。たとえば、次の例は明示的にパラメータを指定した tolower を示します。

```
select xmlextract
('//book[title="Seven Years in Trenton"]//tolower(first-name)', text_doc)
from sample_docs where name_doc='bookstore'
--------------------------------------------------
joe
```

次の例は、同じクエリですが、パラメータを暗黙的に指定しています。

```
select xmlextract
('//book[title="Seven Years in Trenton"]//first-name/tolower()', text_doc)
from sample_docs where name_doc='bookstore'
------------------------------------------
joe
```

相対関数呼び出しが複数のノードに適用される場合も、呼び出しにパラメータを暗黙的に指定できます。次に例を示します。

```
select xmlextract('//book//first-name/tolower()', text_doc)
from sample_docs where name_doc='bookstore'
-------------------------------------------
joemarymarytoni
```

関数

この項では、XML サービスを強化する個々の関数について説明します。

tolower と toupper

説明 tolower は小文字で、toupper は大文字で引数を返します。

構文 tolower(*string-parameter*)
toupper(*string-parameter*)

例 この例では、toupper を使用して引数の値を大文字で返します。

```
select xmlextract
('//book[title="Seven Years in Trenton"]//toupper(first-name)', text_doc)
from sample_docs where name_doc='bookstore'
----------------------------------------
JOE
```

normalize-space

説明 引数の値を返すとき、次の 2 つの変更を行います。

- 先頭または末尾にあるスペース文字を削除する。
- 先頭文字以外の 2 つ以上連続するスペース文字をすべて 1 つのスペース文字に置き換える。

構文 normalize-space(*string-parameter*)

例 この例は、先頭と末尾にスペース文字があり、**改行**文字と**タブ**文字が埋め込まれているパラメータに normalize-space を適用します。

```
select xmlextract
('normalize-space(" Normalize space example. ")', text_doc)
from sample_docs where name_doc='bookstore'
------------------------
Normalize space example.
```

スペース文字や大文字小文字の使用状況が不明な値をテストするとき、XPath 述部で normalize-space と tolower または toupper を使用すると便利です。次の述部は、title 要素での大文字小文字やスペース文字の使用状況には影響されません。

```
select xmlextract
('//magazine[normalize-space(tolower(title)="tracking trenton")]//price',
text_doc)
from sample_docs where name_doc='bookstore'
--------------------------
<price>55</price>
```

concat

説明　concat は引数の値を連結した文字列を返します。0 個以上のパラメータを取ります。

構文　concat(*string-parameter* [,*string-parameter*]...)

例　concat は xmlextract の 1 回の呼び出しで複数の要素を返すことができます。たとえば、次のクエリは first-name 要素と last-name 要素の両方を返します。

```
select xmlextract('//author/concat(first-name, last-name)', text_doc)
from sample_dcs where name_doc='bookstore'
----------------------------------------
JoeBobMaryBobToniBob
```

また、concat を使用して、結果をフォーマットしたり、区切ったりすることもできます。次に例を示します。

```
select xmlextract
('//author/concat(",first(",first-name, ")-last(",last-name, ") ")' , text_doc)
from sample_docs where name_doc='bookstore'
------------------------------------------
first(Joe)-last(Bob) first(Mary)-last(Bob) first(Toni)-last(Bob)
```

カッコで囲んだ式

Adaptive Server ではカッコで囲んだ式をサポートしています。この項では、XPath におけるカッコで囲んだ式の一般的な構文について説明します。以降の各項では、カッコをサブスクリプトや union に使用する方法について説明します。

カッコとサブスクリプト

サブスクリプトは直前にある式に適用されます。パス内の複数の式をグループ化するには、カッコを使用します。この項の例は、カッコをサブスクリプトとともに使用する方法を示します。

サブスクリプトを使用しない次の一般的な例は、book 要素内のすべてのタイトルを返します。

```
select xmlextract('/bookstore/book/title', text_doc) from
sample_docs where name_doc='bookstore'
------------------------------
<title>Seven Years in Trenton</title>
<title>History of Trenton</title>
<title>Tracking Trenton</title>
<title>Treanton Today, Trenton Tomorrow</title>
<title>Whos Who in Trenton</title>
```

最初のタイトルのみを示すには、“[1]” サブスクリプトを使用した次のクエリを入力できます。

```
select xmlextract
('/bookstore/book/title[1]', text_doc)
from sample_docs where name_doc='bookstore'
-----------------------------------------------
<title>Seven Years in Trenton</title>
<title>History of Trenton</title>
<title>Tracking Trenton</title>
<title>Treanton Today, Trenton Tomorrow</title>
<title>Whos Who in Trenton</title>
```

しかし、上のクエリは書店にある最初の本のタイトルではなく、それぞれの本の最初のタイトルを返します。同様に、“[2]” サブスクリプトを使用する次のクエリでは、書店にある 2 番目の本のタイトルではなく、それぞれの本の 2 番目のタイトルを返します。本にはタイトルが 1 つしかないため、結果は空になります。

```
select xmlextract
('/bookstore/book/title[2]', text_doc)
from sample_docs where name_doc='bookstore'
---------------------------------
NULL
```

以上のクエリは書店ではなく本の *i* 番目のタイトルを返します。これは、サブスクリプト演算 (および述部一般) が直前の項目に適用されるためです。本の 2 番目のタイトルではなく書店全体の 2 番目の本のタイトルを返すには、サブスクリプトの適用対象の要素をカッコで囲みます。次に例を示します。

```
select smlextract
('(/bookstore/booktitle)[2]', text_doc)
from sample_docs where name_doc='bookstore'
-------------------------------------------
<title>History of Trenton</title>
```

パスはすべてカッコでグループ化できます。次に例を示します。

```
select xmlextract('(//title)[2]', text_doc)
from sample_docs where name_doc='bookstore'
-------------------------------------------
<title>History of Trenton</title>
```

カッコと union

カッコを使用して 1 つのステップ内の演算をグループ化することもできます。たとえば、次のクエリは書店にあるすべての本のタイトルを返します。

```
select xmlextract('/bookstore/book/title', text_doc) from
sample_docs where name_doc='bookstore'
----------------------------------------
<title>Seven Years in Trenton</title>
<title>History of Trenton</title>
<title>Trenton Today, Trenton Tomorrow</title>
<title>Who's Who in Trenton</title>
```

上のクエリは本のタイトルしか返しません。雑誌 (magazine) のタイトルを返すには、クエリを次のように変更します。

```
select xmlextract('/bookstore/magazine/title', text_doc)
from sample_docs where name_doc='bookstore'
----------------------------------------
<title>Tracking Trenton</title>
```

書店にあるすべての商品のタイトルを返すには、次のようにクエリを変更します。

```
select xmlextract('/bookstore/*/title', text_doc)
from sample_docs where name_doc='bookstore'
---------------------------------
<title>Seven Years in Trenton</title>
<title>History of Trenton</title>
<title>Tracking Trenton</title>
<title>Trenton Today, Trenton Tomorrow</title>
<title>Whos Who in Trenton</title>
```

書店に本と雑誌以外の商品がある場合 (カレンダ、新聞など)、union (垂直線) 演算子を使用し、クエリ・パスをカッコで囲んで、本と雑誌のタイトルのみを問い合わせることができます。次に例を示します。

```
select xmlextract('/bookstore/(book|magazine)/title', text_doc)
from sample_docs where name_doc='bookstore'
---------------------------------
<title>Seven Years in Trenton</title>
<title>History of Trenton</title>
<title>Tracking Trenton</title>
<title>Trenton Today, Trenton Tomorrow</title>
<title>Whos Who in Trenton</title>
```

第 4 章

for xml マッピング関数

この章では、for xml XML マッピング関数を詳細に説明し、その例を示します。

for xml 句

結果セットの XML 表現を返す SQL の select 文を指定します。

構文

```
select ::=
   select [ all | distinct ] select_list
   [into_clause ]
   [where_clause ]
   [group_by_clause ]
   [having_clause ]
   [order_by_clause ]
   [compute_clause ]
   [read_only_clause ]
   [isolation_clause ]
   [browse_clause ]
   [plan_clause]
   [for_xml_clause]
for_xml_clause ::=
   for xml [schema | all] [option option_string] [returns_clause]
option_string ::= basic_string_expression
returns_clause ::=
   returns { char [(integer)] | varchar [(integer)]
   |unichar [(integer)] | univarchar [(integer)]
   |text | unitext | java.lang.String}
```

注意 オプション文字列の詳細については、「option_strings：一般的なフォーマット」(36 ページ) を参照してください。

注意 I18N データで for xml を使用する方法の詳細については、第 6 章 「XML における国際化 (I18N) のサポート」(85 ページ) を参照してください。

説明

- for xml 句は、SQL の select 文の新しい句です。前述した select の構文には、for xml 句をはじめとするすべての句が含まれます。
- その他の select 文の句の構文と説明は、『ASE リファレンス・マニュアル：コマンド』に記載されています。
- for xml 句は、string として表される java.lang.string データ型をサポートしています。その他の Java 型は objectID として表されます。

- **注意** for xml schema および for xml all については、「for xml schema と for xml all」(61 ページ) を参照してください。

---

for xml 句のバリエーションは次のとおりです。

a select 文で for xml 句が指定されている場合は、select 文自体を基本の select と見なし、for xml select のある select 文を for xml select と見なします。たとえば、次の文を考えてみます。

```
select 1, 2 for xml
```

ここで、基本の select は select 1, 2 であり、for xml select は select 1, 2 for xml です。

b for xml schema select コマンドまたはサブクエリには、schema を指定する *for_xml_clause* があります。

c for xml all select コマンドまたはサブクエリには、all を指定する *for_xml_clause* があります。

- for xml select 文には into_clause、compute_clause、read_only_clause、isolation_clause、browse_clause、または plan_clause を含めることができません。
- for xml select は、コマンド create view、declare cursor、subquery、または execute command では指定できません。
- for xml select は union ではジョインできませんが、union を含むことはできます。たとえば、次の文は使用できます。

```
select * from T
union
select * from U
for xml
```

しかし、次の文は使用できません。

```
select * from T for xml
union
select * from U
```

- for xml select の値は、基本の select 文の結果の XML 表現です。XML ドキュメントのフォーマットは、「第 5 章 XML マッピング」で説明されている SQLX フォーマットです。
- returns 句は、for xml クエリまたはサブクエリによって生成された XML ドキュメントのデータ型を指定します。returns 句によってデータ型が指定されない場合、デフォルトは text になります。
- for xml select 文が返す結果セットは、*incremental* オプションによって異なります。
  - *incremental* = *no* の場合は、単一のローと単一のカラムを含む結果セットを返します。そのカラムの値は、基本の select 文の結果の SQLX-XML 表現です。これは、デフォルトのオプションです。
  - *incremental* = *yes* の場合は、基本の select 文の各ローに対するローを含む結果セットを返します。root オプションが *yes* (デフォルト・オプション) を指定する場合、最初のローは XML の開始ルート要素を指定し、最後のローは XML の終了ルート要素を指定します。

    たとえば、次の select 文は、それぞれ 2 つ、1 つ、2 つ、4 つのローを返します。

    ```
    select 11, 12 union select 21, 22
    select 11, 12 union select 21, 22 for xml
    select 11, 12 union select 21, 22
         for xml option "incremental=yes root=no"
    select 11, 12 union select 21, 22
         for xml option "incremental=yes root=yes"
    ```

- for xml クエリの結果生成される datetime 値の date フィールドと time フィールドは、現在 ANSI SQL-XML 標準で記載されているとおり、デリミタ 'T' (文字 T) で区切られます。このフォーマットでない場合、標準 XML パーサの検証に失敗します。

たとえば、このクエリを Adaptive Server 12.5.2 で実行した場合、結果は以下のようになります。

```
select getdate() for xml
```

```
<resultset
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
< row >
< C1 > 2008-05-30 11:42:19 < /C1 >
< /row >
< /resultset >
```

ただし、Adaptive Server 15.0.2 では、同じクエリの結果は以下のようになります。

```
select getdate() for xml

<resultset
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
< row >
< C1 > 2008-05-30T11:41:42 < /C1 >
< /row >
< /resultset >
```

オプション

option_string の一般的なフォーマットについては、「option_strings：一般的なフォーマット」(36 ページ) を参照してください。for xml 句のオプションについては、「SQLX オプション」(67 ページ) を参照してください。

例外

基本の select 文の実行中に発生する SQL 例外は、for xml select によって発生します。たとえば、次の両方の文で 0 による除算の例外が発生します。

```
select 1/0
select 1/0 for xml
```

例

次は、for_xml 句の例です。

```
select pub_id, pub_name
from pubs2.dbo.publishers
for xml
----------------
<resultset
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<row>
   <pub_id>0736</pub_id>
   <pub_name>NewAgeBooks</pub_name>
</row>

<row>
   <pub_id>0877</pub_id>
   <pub_name>Binnet & Hardley</pub_name>
</row>

<row>
   <pub_id>1389</pub_id>
   <pub_name>Algodata Infosystems</pub_name>
</row>

</resultset>
```

for xml サブクエリ

Transact-SQL では、式サブクエリはカッコで囲まれたサブクエリです。式サブクエリは単一のカラム (値は式サブクエリの結果) を持ち、単一のローを返す必要があります。式サブクエリは、式を使用できるほとんどすべての場所で使用できます。サブクエリの詳細については、『Transact-SQL® ユーザーズ・ガイド』を参照してください。

for xml サブクエリ機能では、for xml 句を式サブクエリとして含む任意のサブクエリを使用できます。

構文

subquery ::= select [all | distinct] *select_list*
(select *select_list*
[from *table_reference* [, *table_reference*]...]
[where *search_conditions*]
[group by *aggregate_free_expression* [*aggregate_free_expression*]...]
[having *search_conditions*]
[*for_xml_clause*])
for_xml_clause:: = See “for xml schema and for xml all” on page 64
table_reference::= table_view_name |*ANSI_join* | *derived_table*
table_view_name::= See SELECT in Vol. 2, iCommands, in the “Reference Manual”
ANSI_join::= See SELECT in Vol. 2, “Commands,” in the “Reference Manual”
derived_table::= (subquery) as *table_name*

説明

- for xml 句を含む select コマンドは select 文の結果を表す XML ドキュメントを生成し、その XML ドキュメントを結果セット (単一ローと単一カラムを含む) として返します。この結果セットには、一般的な結果セットの処理手法を使用してアクセスできます。
- for xml 句およびその *option_string* の概要については、「for xml 句」(55 ページ) を参照してください。schema キーワードと return 句をサポートする xml 句の拡張機能については、「for xml schema と for xml all」(61 ページ) を参照してください。
- for xml サブクエリは、for xml 句を含むサブクエリです。
- for xml サブクエリを式サブクエリとして使用することもできますが、これらの間には相違点がいくつかあります。たとえば、通常の式サブクエリには次の制約が適用されますが、for xml サブクエリには適用されません。
  - select リスト内に複数の項目を指定できない。
  - select リスト内に text カラムおよび image カラムを指定できない。
  - group by 句や having 句を指定できない。
- for xml select 内または別の for xml サブクエリ内に for xml サブクエリを指定できない。

- for xml サブクエリは、次のコマンドで使用できません。
  - for xml select
  - create view
  - declare cursor
  - select into
  - any/all、in/not in、exists/not exists などの限定述語サブクエリとしての使用
- for xml サブクエリを相関サブクエリにすることはできません。相関サブクエリの詳細については、『Transact-SQL ユーザーズ・ガイド』を参照してください。
- text または unitext データ型を返す for xml 句は、ネストされたスカラ・サブクエリでは使用できません。
- for xml サブクエリのデータ型は、*for_xml_clause* の returns 句により指定されます。returns 句によりデータ型が指定されない場合、デフォルトのデータ型は text です。

例外

- 例外は、*for_xml_clause* に指定されているものと同じです。
- サブクエリの結果を変換できないデータ型が returns 句に指定されている場合は、例外が発生します。その場合、結果は指定されたデータ型に変換できません。

例

例 1 for xml サブクエリは、XML ドキュメントを文字列値として返します。この値は、文字列カラムまたは変数に代入したり、引数としてストアド プロシージャまたは組み込み関数に渡したりできます。次に例を示します。

```
declare @doc varchar(16384)
set @doc = (select * from systypes for xml returns varchar(16384))
select @doc
--------------
```

例 2 for xml サブクエリの結果を文字列引数として渡すには、次のように入力します。

```
select xmlextract('//row[usertype = 18]',
     (select * from systypes for xml))
------------
```

例 3　for xml サブクエリを insert または update の値として指定するには、次のように入力します。

```
create table docs_xml(id integer, doc_xml text)
insert into docs_xml
   select(1, (select * from systypes for xml)
--------

update docs_xml
set doc_xml = (select * from sysobjects for xml)
where id = 1
------------
```

for xml schema と for xml all

この項では、for xml 句のその他の形式について説明します。XML スキーマ、XML スキーマと XML DTD、または XML データ・ドキュメントを生成できます。

説明

- for xml schema 句が指定された select 文やサブクエリは、schema 述部のない for xml 句が select 文に含まれる場合に生成されるのと同じ SQLX XML 結果セットを記述する XML ドキュメントを生成します。
- この for xml サブクエリの結果は次の xml 値になります。

  ```
  (subquery for xml schema option option_string)
  ```

- for xml all 句が指定された select 文やサブクエリは、SQLX 結果セット、XML スキーマ、結果セットを記述した XML DTD を含む XML ドキュメントを生成します。これらは、次の要素を持つ単一の XML ドキュメントに格納されます。
  - <*multiple-results*> ― ルート要素
  - <*multiple-results-item type="result-set"*> ― 次のものを格納する要素。

    <*multiple-results-item-dtd*> ― 結果セットの DTD

    <*multiple-result-item-schema*> ― 結果セットの XML スキーマ

    <*multiple-result-item-data*> ― 結果セット

オプション

option_string の一般的なフォーマットについては、「option_strings：一般的なフォーマット」(36 ページ) を参照してください。for xml 句のオプションについては、「SQLX オプション」(67 ページ) を参照してください。

例外

拡張機能の例外は、「SQLX オプション」(67 ページ) で指定されているものと同じです。

使用例

次の例は、for xml schema および for xml all の使用方法を示します。

例 1 この例では、for xml all サブクエリは次のものを返します。

- XML スキーマ
- XML スキーマと XML DTD
- XML ドキュメントとしての結果セット

これらはすべて同じ文字列値で返されます。返された値は、string カラムまたは変数に代入したり、文字列引数としてストアド プロシージャまたは関数に渡したりできます。

```
declare @doc varchar(16384)
set @doc = (select * from systypes for xml all returns varchar(16384))
select @doc
-----------
```

例 2 次の例では、for xml schema サブクエリの結果を文字列引数として渡しています。

```
select xmlextract('//row[usertype=18]'
   (select * from systypes for xml all))
---------
```

例 3 次の例では、for xml all サブクエリを insert コマンドまたは update コマンドの値として指定しています。

```
create table docs_xml(id integer, doc_xml xml)
insert into docs_xml
   values(1,(select * from sysobjects for xml all)
where id=1
```

結果の例

次の例は、上記の例の各コマンドによって生成された結果を示します。

例 1 次の例は、基本の select for xml 文の結果を示します。

```
select "a", 1 for xml
-----------
<resultset
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <row>
    <C1>a</C1>
    <C2>1</C2>
  </row>

</resultset>

(1 row affected)
```

例 2　次の例は、例 1 の結果セットを記述する XML スキーマを返す for xml schema を示します。

```
select "a", 1 for xml schema
--------------
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:sqlxml="http://www.iso-standards.org/mra/9075/sqlx">

<xsd:import namespace="http://www.w3.org/20001/XMLSchema"
 schemaLocation="http://www.iso-standards.org/mra/9075/sqlx.xsd"/>

<xsd:complexType name="RowType.resultset"
  <xsd:sequence>
   <xsd:element name="C1" type="VARCHAR_1"/>
   <xsd:element name="C2" type="INTEGER"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="TableType.resultset"
  <xsd: sequence>
   <xsd:element name="row" type="RowType.resultset"
    minOccurs="0" maxOccurs="unbounded"/>
   </xsd:sequence>
</xsd:complexType

<xsd:simpleType name="VARCHAR_1">
  <xsd:restriction base="xsd:string".
   <xsd:length value="1"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="INTEGER">
  <xsd:restriction base="xsd:integer">
   <xsd:maxInclusive value="2147483647"/>
   <xsd:minInclusive value="-2147483648"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:element name+"resultset" type="TableType.resultset"/>
</xsd:schema>

(1 row affected)
```

例 3 次の例は、スキーマ、DTD、結果セットのデータを返す for xml all の使用例です。

```
    select 'a', 1 for xml all
    -----------
    <multiple results>

    <multiple-results-item type="result-set">
    <multiple-results-item-dtd>

    <!DOCTYPE resultset [
    <!ELEMENT resultset(row*)>
    <!ELEMENT row (C1,C2)>
    <!ELEMENT C1(#PCDATA)>
    <!ELEMENT C2(#PCDATA)>

    ]>

</multiple-results-item-dtd>

</multiple-results-item-schema>

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmln:sqlxml="http://www.iso-standards.org/mra/9075/sqlx">

 <xsd:import namespace="http://2=www.w3.org/2001/XMLSchema"
  schemaLocation="http://www.iso-standards.org/mra/9075/sqlx.xsd"/>

<xsd:complexType name="RowType.resultset">
 <xsd:sequence>
  <xsd:element name="C1"type="VARCHAR_1" />
  <xsd:element name="C2"type="INTEGER" />
 </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="TableType.resultset">
 <xsd:sequence>
  <xsd:element name="row" type="RowType.resultset"
    minOccurs="0" maxOccurs="unbounded"/>
 </xsd:sequence>
</xsd:complexType>

<xsd:simpleType name="VARCHAR_1">
 <xsd:restriction base="xsd:string">
  <xsd:length value="1"/>
 </xsd:restriction>
</xsd:simpleType

<xsd:element name="resultset" type="TableType.resultset"/>

</xsd:schema>
```

```
</multiple-results-item-data>

<multiple-results-item-data>

<resultset xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

 <row>
  <C1>a</C1>
  <C2>1</C2>
 </row>

</resultset>

</multiple-results-item-data>

</multiple-results-item>

</multiple-results>

(1 row affected)
```

第 5 章

XML マッピング

select 文の for xml 句は、ANSI SQLX 標準で定義されている SQLX-XML フォーマットを使用して、SQL 結果セットを SQLX-XML ドキュメントにマップします。この章では、for xml 句でサポートされている SQLX-XML フォーマットとオプションについて説明します。

注意 isql を使用して、for xml 句を含む XML ドキュメントを生成した場合、isql により先行の空白がカラム・セパレータとして追加されるため、生成されたドキュメントが無効になることがあります。

トピック名	ページ
SQLX オプション	67
SQLX データのマッピング	77

SQLX オプション

注意 表 5-1 では、下線が引かれているワードがデフォルト値です。

表 5-1: SQLX マッピングのオプション

オプション名	オプションの値	目的
binary	hex | base64	バイナリの表現。for xml 句にのみ適用される。
columnstyle	element | attribute	SQL カラムの表現
entitize	yes | no | cond	for xml 句
format	yes | no	フォーマットを組み込む
header	yes | no | encoding デフォルト値は戻り型によって異なります。第 6 章 XML における国際化 (I18N) のサポートを参照してください。	XML 宣言を組み込む
incremental	yes | no	for xml を指定する select 文から単一のローまたは複数のローを返す
multipleentitize	yes | no	for xml 句

オプション名	オプションの値	目的
nullstyle	attribute | omit	*columnstyle=element* での null の表現
ncr	non_ascii | non_server | no	for xml 句
prefix	SQL 名	生成される名前のベース。デフォルト値は C です
root	yes | no	テーブル名のルート要素を含める
rowname	SQL 名	ロー要素の名前。 デフォルト値は row です。
schemaloc	引用符で囲まれた URI 文字列	*schemalocation* 値
statement	yes | no	SQL クエリを組み込む
tablename	SQL 名	ルート要素の名前。デフォルト値は resultset です
targetns	引用符で囲まれた URI 文字列	*targetnamespace* 値 (存在する場合)
xsidecl	yes | no	for xml 句

SQLX オプションの定義

この項では、表 5-1 に示した SQLX オプションについて説明します。

binary={hex | base64}

このオプションは、データ型が binary、varbinary、または image のカラムを hex と base64 のどちらでコード化して表示するかを示します。この選択は、生成されるドキュメントの処理に使用するアプリケーションによって決まります。base64 コード化は、hex コード化よりもコンパクトです。

columnstyle={element | attribute}

このオプションは、SQL カラムを要素として表すか、XML “row” 要素の属性として表すかを示します。

次の例は、*columnstyle=element* (デフォルト) を示します。

```
select pub_id, pub_name from pubs2..publishers
for xml option "columnstyle=element"
--------------------------------
<resultset xmlns:xsi="http://www.w3.org/2001/
   XMLSchema-instance">

   <row>
     <pub_id>0736</pub_id>
     <pub_name>New Age Books</pub_name>
   </row>

   <row>
     <pub_id>0877</pub_id>
     <pub_name>Binnet & Hardley</pub_name>
   </row>

   <row>
     <pub_id>1389</pub_id>
     <pub_name>Algodata Infosystems</pub_name>
   </row>

</resultset>
```

次の例は、*columnstyle=attribute* を示します。

```
select pub_id, pub_name from pubs2..publishers
for xml option  "columnstyle=attribute"
-------------------------------------------------
<resultset
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

   <row
     pub_id="0736"
     pub_name="New Age Books"
   />
   <row
     pub_id="0877"
     pub_name="Binnet & Hardley"
   />
```

```
    <row
      pub_id="1389"
      pub_name="Algodata Infosystems"
    />
</resultset>
```

entitize = {yes | no | cond}

このオプションは、文字列カラムで、XML の予約文字 (“<>”、“&”、“‘”、“““) を XML エンティティ (<、>、&、'、&quote;) に変換するかどうかを指定します。yes または no を使用して予約文字をエンティティ化するかどうかを示します。cond は、カラム内の最初の文字 (ブランク以外) が “<” でない場合のみ、予約文字をエンティティ化します。for xml では、最初の文字が “<” の文字列カラムは XML ドキュメントと見なされ、エンティティ化されません。

たとえば、次の例では、すべての文字列カラムがエンティティ化されます。

```
select 'a<b' for xml option 'entitize=yes'
----------
<resultset>
   <row>
      <C1><a&lt;b</C1>
  </row>
</resultset>
```

次の例では、どの文字列カラムもエンティティ化されません。

```
select '<ab>' for xml option 'entitize=no'
-------
<resultset>
   <row>
     <C1><ab></C1>
   </row>
</resultset>
```

次の例では、“<” 以外で始まる文字列カラムがエンティティ化されます。

```
select '<ab>', 'a<b' for xml option 'entitize=cond'
---------
<resultset>
   <row>
     <C1><ab></C1>
     <C2>a&lt;b</C2>
   </row>
</resultset>
```

format={yes | no}

このオプションは、改行文字とタブ文字のフォーマットを組み込むかどうかを指定します。

次に例を示します。

```
select 11, 12 union select 21, 22
for xml option "format=no"
------------------------------
<resultset xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<row><C1>11</C1><C2>12</C2></row>
<row><C1>21</C1><C2>22</C2></row>
</resultset>
```

header=
{yes | no | encoding}

このオプションは、生成される SQLX-XML ドキュメントに XML ヘッダ行を組み込むかどうかを示します。XML ヘッダ行は次のとおりです。

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

生成される SQLX-XML ドキュメントをスタンドアロン XML ドキュメントとして使用する場合は、このようなヘッダ行を組み込みます。生成されるドキュメントを他の XML と結合する場合は、ヘッダ行を省略します。

コード化のオプションについては、「XML における国際化 (I18N) のサポート」(85 ページ) を参照してください。

次に例を示します。

```
select 1,2 for xml option "header=yes"
----------------------------------------
<?xml version="1.0" ?>
<resultset xmlns:xsi="http://www.w3.org/2001
    /XMLSchema-instance">
<row>
      <C1>1</C1>
      <C2>2</C2>
   </row>
</resultset>
```

incremental={yes | no}

このオプションは、for xml 句にのみ適用され、forxml 関数には適用されません。for xml 句を指定した select 文で返される値を次のいずれかに指定します。

- *incremental=no* − select 文の結果に対する完全な SQLX-XML ドキュメントを含む、text データ型の単一カラムを持つ単一ローを返します。*incremental=no* がデフォルトのオプションです。
- *incremental=yes* − select 文の結果のローごとに、そのローの XML 要素を含む text データ型の単一カラムを持つ別々のローを返します。
  - *root* オプションが *yes* (デフォルト) の場合、*incremental=yes* オプションは、*tablename* の開始要素と終了要素を含む 2 つの追加のローを返します。

- *root* オプションが *no* である場合、*tablename* オプション (明示的またはデフォルト) は無視されます。追加のローが 2 つ返されることはありません。

  たとえば、次の 3 つの select 文は、それぞれ 1 つのロー、2 つのロー、4 つのローを返します。

```
select 11, 12 union select 21, 22
for xml option "incremental=no"

select 11, 12 union select 21, 22
for xml option "incremental=no root=no"

select 11, 12 union select 21, 22
for xml option "incremental=no root=yes"
```

multipleentitize= {yes | no}

このオプションは for xml all に適用されます。エンティティ化については、「Entitize = yes | no」オプションを参照してください。

ncr= {no | non_ascii | non_server}

「数値文字表現」(86 ページ) を参照してください。

nullstyle= {attribute | omit}

このオプションは、*columnstyle* を指定するか、デフォルトで *columnstyle=element* に指定した場合に使用する null の代替 SQLX 表現を示します。*columnstyle=attribute* を指定した場合には、nullstyle オプションは意味がありません。

nullstyle=omit オプション (デフォルト・オプション) は、null カラムを含むローから null カラムを除外することを指定します。*nullstyle=attribute* オプションは、null カラムを、*xsi:nil=true* 属性を持つ空の要素として組み込むことを指定します。

次の例は、デフォルトである *nullstyle=omit* オプションを示します。

```
select 11, null union select null, 22
for xml option "nullstyle=omit"
--------------------------------
<resultset
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
     <row>
        <C1>11</C1>
     </row>
     <row>
         <C2>22</C2>
     </row>

</resultset>
```

次の例は、*nullstyle=attribute* を示します。

```
select 11, null union select null, 22
for xml option "nullstyle=attribute"
------------------------------------------------------
<resultset
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <row>
      <C1>11</C1>
       <C2 xsi:nil="true"/>
   </row>
   <row>
       <C1 xsi:nil="true"/>
       <C2>22</C2>
   </row>
</resultset>
```

root= {yes | no}

このオプションは、SQLX-XML 結果セットに *tablename* の *root* 要素を含めるかどうかを指定します。デフォルトは *root=yes* です。*root=no* の場合、tablename オプションは無視されます。

```
<resultset xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

   <row>
      <C1>11</C1>
      <C2>12</C2>
  </row>

   <row>
      <C1>21</C1>
      <C2>22</C2>
   </row>

</resultset>


select 11, 12 union select 21, 22
for xml option "root=no"
-----------------------------------------------

    <row>
      <C1>11</C1>
      <C2>12</C2>
   </row>

   <row>
      <C1>21</C1>
      <C2>22</C2>
   </row>
```

rowname=sql_name

このオプションは、“row” 要素の名前を指定します。デフォルトの *rowname* は “row” です。*rowname* オプションは SQL 名であり、これは通常の識別子または区切り識別子です。「XML 名への SQL 名のマッピング」(80 ページ) で説明しているように、区切り識別子は XML 名にマップされます。

次の例は、*rowname=RowElement* を示します。

```
select 11, 12 union select 21, 22
forxml option "rowname=RowElement"
-----------------------------------------
<resultset xmlns:xsi="http://www.w3.org/2001
          /XMLSchema-instance">

   <RowElement>
      <C1>11</C1>
      <C2>12</C2>
   </RowElement>

   <RowElement>
      <C1>21</C1>
      <C2>22</C2>
   </RowElement>

</resultset>
```

schemaloc=uri

このオプションは、生成される SQLX-XML ドキュメントに *xsi:SchemaLocation* または *xsi:noNamespaceSchemaLocation* 属性として組み込む URI を指定します。このオプションのデフォルトは空の文字列であり、スキーマ・ロケーション属性を省略することを示します。

スキーマ・ロケーション属性は、スキーマ対応 XML パーサへのヒントとして機能します。対応する SQLX-XML スキーマを格納する URI がわかっている場合には、SQLX-XML 結果セットに対してこのオプションを指定します。

targetns オプションを指定しないで *schemaloc* オプションを指定した場合は、次の例のように、*schemaloc* が *xsi:noNamespaceSchemaLocation* 属性に配置されます。

```
select 1,2
for xml option "schemaloc='http://thiscompany.com/schemalib' "
-----------------------------------------------
<resultset xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:noNamespaceSchemaLocation=
     "http://thiscompany.com/schemalib">
   <row>
     <C1>1</C1>
     <C2>2</C2>
   </row>
</resultset>
```

targetns オプションを指定して *schemaloc* オプションを指定した場合は、次の例のように、*schemaloc* が *xsi:schemaLocation* 属性に配置されます。

```
select 1,2
for xml option "schemaloc='http://thiscompany.com/schemalib'
        targetns='http://thiscompany.com/samples'"
----------------------------------------------------
<resultset xmlns:xsi="http://www.w3.org/2001
        /XMLSchema-instance"
   xsi:schemaLocation="http://thiscompany.com/schemalib"
xmlns="http://thiscompany.com/samples">

   <row>
      <C1>1</C1>
      <C2>2</C2>
   </row>

</resultset>
```

statement={yes | no}

このオプションは、root 要素に statement 属性を組み込むかどうかを指定します。root=no を指定した場合、statement オプションは無視されます。

```
select name_doc from sample_doc
where name_doc like "book%"
for xml option "statement=yes"
-----------------------------------------------
<resultset statement="select name_doc
  from sample_docs where name_doc like &quot;book%&quot;"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <row>
      <name_doc>bookstore</name_doc>
   </row>
</resultset>
```

tablename=sql_name

このオプションは、結果セットの名前を指定します。デフォルトの *tablename* は “*resultset*” です。

tablename オプションは SQL 名であり、これは通常の識別子または区切り識別子です。「XML 名への SQL 名のマッピング」(80 ページ) で説明しているように、区切り識別子は XML 名にマップされます。

次の例は、*tablename=SampleTable* を示します。

```
select 11, 12 union select 21, 22
for xml option "tablename=SampleTable"
-------------------------------------------------
<SampleTable xmlns:xsi="http://www.w3.org/2001
           /XMLSchema-instance">

   <row>
      <C1>11</C1>
      <C2>12</C2>
   </row>
```

```
      <row>
         <C1>21</C1>
         <C2>22</C2>
      </row>

   </SampleTable>
```

targetns=uri

このオプションは、生成される SQLX-XML ドキュメントに *xmlns* 属性として組み込む URI を指定します。このオプションのデフォルトは空の文字列であり、*xmlns* 属性を省略することを示します。*schemaloc* 属性と *targetns* 属性間の相互作用の説明については、*schemaloc* 属性を参照してください。

```
select 1,2
for xml
option "targetns='http://thiscompany.com/samples'"
---------------------------------------
<resultset xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://thiscompany.com/samples">
   <row>
      <C1>1</C1>
      <C2>2</C2>
   </row>

</resultset>
```

xsidecl={yes | no}

このオプションでは、XML の xsi 属性を宣言するかどうかを指定できます。次に例を示します。

```
select 1 for xml option 'xsidecl=yes'
--------
  <resultset
    xmlns:xsi="http://www.w3.org/2001/XMLScainstance">
     <row>
         <C1>1</C1>
     </row>
  </resultset>


select 1 for xml option 'xsidecl=no'
-------
<resultset>

   <row>

      <C1>1</C1>

   </row>
```

xsi 属性は、nullstyle=attribute の null 値に使用します。

```
select null for xml
     option 'nullstyle=attribute xmldecl=yes'

   If you specify xsidecl=no or <resultset
     xmlns:xsi="http://www.w3.org/2001
     /XMLSchema-instance">
        <row>
            <C1 xsi:nil="true"/>
        </row>
   </resultset>
```

nullstyle=element または nullstyle=attribute を指定する場合、結果として生成される XML ドキュメントを、すでに xsi 属性の宣言を含む大きな XML ドキュメントに埋め込むときは、xsidecl=no と指定できます。

SQLX データのマッピング

この項では、select 文の for xml 句で生成されるドキュメントで使用される SQLX-XML フォーマットについて説明します。SQLX-XML フォーマットは、ANSI SQLX 標準で規定されています。

重複したカラム名と名前のないカラムのマッピング

次のクエリは、同じ名前の 2 つのカラムと、名前のない 3 つのカラムを返します。

```
select t1.title_id, t2.title_id, t2.advance-t1.advance,
t1.price*t1.total_sales, t2.price*t2.total_sales
from pubs2..titles t1, pubs2..titles t2
where t1.price=t2.price and t2.advance-t1.advance>3000
title_id title_id
------   ------    --------   ---------   ---------

BU2075   MC3021    4,875.00   55,978.78   66,515.54
MC2222   BU1032    5,000.00   40,619.68   81,859.05
MC2222   BU7832    5,000.00    40,619.68   81,859.05
```

このデータが XML にマップされると、カラムは (*columnstyle* オプションに応じて) 要素または属性になります。このような要素と属性にはユニークな名前が必要です。したがって、生成される XML は、重複したカラム名には整数のサフィックスを追加し、名前のないカラムにはサフィックス付きのユニークな名前を生成します。次に、前述のクエリを使用した例を示します。

```
select t1.title_id, t2.title_id, t2.advance-t1.advance,
t1.price*t1.total_sales, t2.price*t2.total_sales
from pubs2..titles t1, pubs2..titles t2
where t1.price=t2.price and t2.advance-t1.advance>3000
for xml
-----------------------------------------------------
<resultset xmlns:xsi="http://www.w3.org/2001
         /XMLSchema-instance">

   <row
      <title_id1>BU2075</title_id1>
      <title_id2>MC3021</title_id2>
      <C1>4875.00</C1>
      <C2>55978.78</C2>
      <C3>66515.54</C3>
   </row>

   <row>
      <title_id1>MC2222</title_id1>
      <title_id2>BU1032</title_id2>
      <C1>5000.00</C1>
      <C2>40619.68</C2>
      <C3>81859.05</C3>
   </row>

   <row>
      <title_id1>MC2222</title_id1>
      <title_id2>BU7832</title_id2>
      <C1>5000.00</C1>
      <C2>40619.68</C2>
      <C3>81859.05</C3>
   </row>

</resultset>
```

名前のないカラムに対して XML によって生成される名前が既存のカラム名と一致する場合、XML によって生成される名前はスキップされます。次の例では、名前のない最後のカラムが明示的なカラム名 “C1” を持つため、“C1” は生成されるカラム名として使用されません。

```
select t1.title_id, t2.title_id, t2.advance-t1.advance,
t1.price*t1.total_sales,t2.price*t2.total_sales as C1
from pubs2..titles t1, pubs2..titles t2
where t1.price=t2.price and t2.advance-t1.advance>3000
for xml
------------------------------------------------------
<resultset xmlns:xsi="http://www.w3.org/2001
        /XMLSchema-instance">

<row>
      <title_id1>BU2075</title_id1>
      <title_id2>MC3021</title_id2>
      <C2>4875.00</C2>
      <C3>55978.78</C3>
      <C1>66515.54</C1>
</row>

<row>
      <title_id1>MC2222</title_id1>
      <title_id2>BU1032</title_id2>
      <C2>5000.00</C2>
      <C3>40619.68</C3>
      <C1>81859.05</C1>
</row>

<row>
      <title_id1>MC2222</title_id1>
      <title_id2>BU7832</title_id2>
      <C2>5000.00</C2>
      <C3>40619.68</C3>
      <C1>81859.05</C1>
</row>

</resultset>
```

前述の例では、名前のないカラムに対して生成される名前の形式は、“C1”、“C2” などです。これらの名前は、ベース名 “C” と整数サフィックスから構成されます。*prefix* オプションを使用して別のベース名を指定することもできます。

次の例は、*prefix='column_'* を示します。

```
select t1.title_id, t2.title_id, t2.advance-t1.advance,
t1.price*t1.total_sales, t2.price*t2.total_sales
from pubs2..titles t1, pubs2..titles t2
where t1.price=t2.price and t2.advance-t1.advance>3000
for xml option "prefix=column_"
-------------------------------------
<resultset xmlns:xsi="http://www.w3.org/2001
          /XMLSchema-instance">
   <row>
      <title_id1>BU2075</title_id1>
      <title_id2>MC3021</title_id2>
      <column_1>4875.00</column_1>
      <column_2>55978.78</column_2>
      <column_3>66515.54</column_3>
   </row>

   <row>
      <title_id1>MC2222</title_id1>
      <title_id2>BU1032</title_id2>
      <column_1>5000.00</column_1>
      <column_2>40619.68</column_2>
      <column_3>81859.05</column_3>
   </row>

   <row>
      <title_id1>MC2222</title_id1>
      <title_id2>BU7832</title_id2>
      <column_1>5000.00</column_1>
      <column_2>40619.68</column_2>
      <column_3>81859.05</column_3>
   </row>

</resultset>
```

XML 名への SQL 名のマッピング

SQL テーブルと結果セットの SQLX 表現は、XML の要素名と属性名として SQL 名を使用します。ただし、SQL 名には、XML 名では有効ではないさまざまな文字が含まれる可能性があります。特に、SQL 名には「区切り」識別子が含まれます。これは引用符で囲まれた名前です。区切り識別子には、スペースや句読点などの任意の文字を含めることができます。次に例を示します。

```
"salary + bonus: "
```

これは、有効な SQL 区切り識別子です。そのため、SQLX 標準は、このような文字から有効な XML 名文字へのマッピングを規定しています。

SQLX 名のマッピングの目的は次のとおりです。

- すべての SQL 識別子を扱う
- 元の識別子を再生成できる逆マッピングを保証する

SQLX 名マッピングは、文字の Unicode 表現に基づいています。次のような Unicode 表現を持つ無効な文字があるとします。

```
U+nnnn
```

SQLX 名マッピングの基本規則では、これは次の形式の文字列に置き換えられます。

```
_xnnnn_
```

無効な名前文字の SQLX マッピングでは、Unicode 表現の 4 桁の 16 進数に次のプレフィクスが付きます。

```
_x
```

さらに、アンダースコアのサフィックスが付きます。

たとえば、次のような SQL 結果セットがあるとします。

```
set quoted_identifier on
select 1 as "a + b < c & d", 2 as "<a xsi:nill=""true"">"
----------------------

a + b < c & d <a xsi:nill="true">
------------- --------------------
            1                    2
```

この例の select リストは、定数 (1 と 2) の値を指定し、as 句を使用してそれらの値のカラム名を指定します。これらのカラム名は、XML 名では有効でない文字を含む区切り識別子です。

その結果セットの SQLX マッピングは次のようになります。

```
set quoted_identifier on
select 1 as "a + b < c & d", 2 as "<a xsi:nill=""true"">"
for xml
----------------------------------------------------
<resultset xmlns:xsi="http://www.w3.org/2001
        /XMLSchema-instance">

<row>
<a_x0020__x002B__x0020_b_x0020__x003C__x0020_c_x0020__x0026__x0020_d_x0020_>
1
</a_x0020__x002B__x0020_b_x0020__x003C__x0020_c_x0020__x0026__x0020_d_x0020_>
<_x003C_a_x0020_xsi_x003A_nill_x003D__x0022_true_x0022__x003E_>
2
</_x003C_a_x0020_xsi_x003A_nill_x003D__x0022_true_x0022__x003E_></row>

</resultset>
```

生成される SQLX 結果セットは簡単には判読できませんが、SQLX マッピングは主にアプリケーションによる使用を目的としています。*_xnnnn_* 規則は、ほとんどの SQLX 名マッピングの考慮事項に対処します。

ただし、もう 1 つの要件として、大文字または小文字をどのように組み合わせても XML 名を “XML” という文字からは開始できないことがあります。したがって SQLX 名マッピングでは、このような名前の先頭の “x” または “X” が値 *_xnnnn_* に置き換えられます。最初の “X” のみを置き換えれば XML というフレーズがマスクされるため、“M” と “L” (大文字でも小文字でも) は変更されません。

次に例を示します。

```
select 1 as x, 2 as X, 3 as X99, 4 as xML, 5 as XmLdoc
forxml
------------------------------------------------------
<resultset xmlns:xsi="http://www.w3.org/2001
        /XMLSchema-instance">

   <row>
     <x>1</x>
     <X>2</X>
     <X99>3</X99>
     <_x0078_ML>4</_x0078_ML>
     <_x0058_mLdoc>5</_x0058_mLdoc>
   </row>

</resultset>
```

SQL 名から XML 名へのマッピングの要件は、*tablename*、*rowname*、*prefix* オプションで指定された SQL 名にも適用されます。次に例を示します。

```
select 11, 12 union select 21, 22
for xml option "tablename='table @ start' rowname=' row & columns '
       prefix='C '"
-----------------------------------------------------
<table_x0020__x0040__x0020_start xmlns:xsi="http://www.w3.org/2001
          /XMLSchema-instance">

<_x0020_row_x0020__x0026__x0020_columns_x0020_>
    <C_x0020_1>11</C_x0020_1>
    <C_x0020_2>12</C_x0020_2>
</_x0020_row_x0020__x0026__x0020_columns_x0020_>

<_x0020_row_x0020__x0026__x0020_columns_x0020_>
    <C_x0020_1>21</C_x0020_1>
    <C_x0020_2>22</C_x0020_2>
</_x0020_row_x0020__x0026__x0020_columns_x0020_>

</table_x0020__x0040__x0020_start>
```

XML 値への SQL 値のマッピング

SQL 結果セットの SQLX 表現は、カラムを表す XML の属性または要素の値にカラムの値をマップします。

数値

数値データ型は、SQLX マッピングでは文字列リテラルとして表現されます。次に例を示します。

```
select 1, 2.345, 67e8 for xml
------------------------------------------------------
<resultset xmlns:xsi="http://www.w3.org/2001
       /XMLSchema-instance">

  <row>
      <C1>1</C1>
      <C2>2.345</C2>
      <C3>6.7E9</C3>
  </row>

</resultset>
```

文字値

char、varchar、または text カラムに含まれる文字値には、追加の処理が必要です。SQL データの文字値には、XML で特別な意味を持つ引用符 (")、アポストロフィ (')、より小さい (<)、より大きい (>)、アンパサンド (&) 文字を含めることができます。SQL 文字値が XML 属性または要素値として表現される場合は、それらを表す XML エンティティ ("、'、<、>、&) に置き換える必要があります。

次の例は、XML マークアップ文字を含む SQL 文字値を示します。SQL の select コマンド内の文字リテラルは、埋め込み引用符とアポストロフィを規定する SQL 規則を使用して、アポストロフィを二重にします。

```
select '<name>"Baker''s"</name>'
----------------------
<name>"Baker's"</name>
```

次の例は、XML マークアップ文字が XML エンティティ表現に置き換えられる、文字値の SQLX マッピングを示します。

```
select '<name>"Baker''s"</name>' for xml
------------------------------------------------
<resultset xmlns:xsi="http://www.w3.org/2001
        /XMLSchema-instance">

<row>
<C1>&amp;lt;name&amp;gt;&amp;quot;Baker&amp;apos;s&amp;quot;&amp;lt;/name&amp;gt;<
      /C1>
```

```
</row>

</resultset>
```

バイナリ値

binary、varbinary、または image カラムに含まれるバイナリ値は、オプション *binary={hex|base64}* に応じて hex または base64 のコード化で表現されます。base64 コード化の方がコンパクトです。2 つの表現のどちらを選択するかは、XML データを処理するアプリケーションによって決まります。

第 6 章

XML における国際化 (I18N) のサポート

この章では、非 ASCII データをサポートするための XML サービスの拡張機能について説明します。これは、Unicode ベースを指定する XML 標準をサポートするだけでなく、複数言語にまたがる XML ベース アプリケーションをサポートするためにも必要です。

このマニュアルでは、“I18N” という用語は国際化 (Internationalization) を意味します。I18N は、Internationalization の先頭の “I”、それに続く 18 文字、末尾の “n” を表します。この用語は、Unicode および ASCII セット以外の文字のサポートを意味します。

トピック名	ページ
概要	85
for xml における I18N	87
xmlparse における I18N	92
xmlextract における I18N	93
xmlvalidate における I18n	95

概要

I18N 拡張機能は次の 3 つに分類されます。

- for xml 句における I18N サポート。非 ASCII データを含むドキュメントを生成する。
- xmlparse における I18N。非 ASCII データを含むドキュメントを格納する。
- xmlextract および xmltest における I18N。非 ASCII データを含む XML ドキュメントおよびクエリを処理する。

Unicode データ型

次の用語は、Unicode に使用されるデータ型の種類に関するものです。

- 「文字列データ型」は、char、varchar、text、java.lang.String を意味します。
- 「Unicode データ型」は、unichar、univarchar、unitext、java.lang.String を意味します。
- 「文字列／Unicode カラム」は、データ型が「文字列データ型」または「Unicode データ型」のカラムを意味します。

サロゲート・ペア

「サロゲート・ペア」は、16 ビットを超える可能性がある文字を表すために Unicode で使用される 16 ビット値のペアを意味します。

ほとんどの文字は [0x20, 0xFFFF] の範囲で表され、単一の 16 ビット値で表すことができます。サロゲート・ペアは、[0x010000..0x10FFFF] の範囲内にある文字を表す 16 ビット値のペアです。詳細については、「例 7」(92 ページ) を参照してください。

数値文字表現

数値文字表現 (NCR: Numeric Character Representation) は、XML ドキュメントにおいて ASCII 16 進表記で任意の文字を表す手法です。たとえば、ユーロ記号 “€” の NCR 表現は “€” です。この表記は、SQL の 16 進文字表記 u&'¥20ac' と似ています。

クライアントとサーバ間の変換

サーバの Unicode データは、次のいずれかです。

- UTF-16 データ。unichar、univarchar、unitext、*java.lang.String* に格納される。
- UTF-8 データ。サーバ文字セットが UTF-8 の場合に、char、varchar、text に格納される。

クライアントとサーバ間のデータ転送　次の 3 つの手法のいずれかを使用して、クライアントとサーバ間で univarchar データまたは unitext データを転送します。

- CTLIB または BCP を使用する。データをビット文字列として転送します。クライアント・データは UTF-16 であり、クライアントとサーバの違いに合わせてバイト順序が調整されます。
- ISQL または BCP を使用する。“-J UTF-8” を指定します。データは、クライアントの UTF-8 とサーバの UTF-16 の間で変換されます。

- Java を使用する。データ転送でクライアント文字セット (変換元または変換先) を指定します。UTF-8、UTF-16BE、UTF-16L、UTF-16LE、UTF-16 (BOM を使用)、US-ASCII、または別のクライアント文字セットを指定できます。

注意 JDBC を介して Unicode XML ドキュメントを格納する場合、接続プロパティ 'DISABLE_UNICODE_SENDING'、つまり JDBC 接続から Adaptive Server に Unicode データを送信できるようにする “false” プロパティを指定する必要があります。

クライアント Java アプリケーションでクライアント・ファイルの文字セットを指定する手法 (入力または出力) は、次のサンプル・ディレクトリにある Java アプリケーションで確認できます。

```
$SYBASE/$SYBASE_ASE/sample/JavaXml/JavaXml.zip
```

このディレクトリには *Using-SQLX-mappings* ドキュメントと、Unicode と SQLX の結果セット・ドキュメントもあります。

文字セットと XML データ

XML ドキュメントを文字列カラムまたは変数に格納すると、XML サービスはそのドキュメントがサーバ文字セットを使用していると想定します。また、Unicode カラムまたは変数に格納すると、XML サービスはそのドキュメントが UTF-16 であると想定します。XML ドキュメント内の ENCODING 句はすべて無視されます。

for xml における I18N

この項では、for xml 句を拡張して非 ASCII データを処理する方法について説明します。

非 ASCII 文字を含む Unicode カラムおよび文字列カラムは、for xml 句の *select_list* で指定できます。

returns 句におけるデフォルトのデータ型は text です。

結果としての XML ドキュメントは Unicode 文字列として内部的に生成され、必要に応じて returns 句のデータ型に変換されます。

この句の詳細については、「for xml 句」(55 ページ) を参照してください。

オプション文字列

for xml 句のオプション文字列には u& 形式のリテラルを指定して、文字の SQL 表記を含めることができます。また、rowname、tablename、prefix オプションには Unicode 文字を指定できます。たとえば、次のように入力します。

```
select * from T
for xml
options u&'tablename = ¥0415¥0416 rowname =
          ¥+01d6d prefix = ¥0622'
```

指定した tablename、rowname、または prefix オプションに、単純な識別子では無効な文字が含まれるときは、そのオプションを引用符付き識別子として指定する必要があります。たとえば、次のように入力します。

```
select * from T
for xml
optons u&'tablename = "chars¥0415 and ¥0416"
     rowname = "¥+01d6d1 & ¥+01d160"
     prefix = "¥0622-"'
```

for xml における数値文字表現

select for xml 文の *option_string* は、文字列カラムおよび Unicode カラムの表現を指定する ncr オプションを含みます。

```
ncr = {no | non_ascii | non_server}
```

- *ncr = no* は、文字列カラムおよび Unicode カラムがプレーンな値として表されることを指定します。プレーンな値は、entitize オプションによって、エンティティ化されたりエンティティ化されなかったりします。
- *ncr = non_ascii* と *ncr = non_server* は、非 ASCII である文字列カラムと、デフォルトのサーバ文字セットのメンバではない Unicode カラムが NCR として表されることを指定します。NCR に変換されない文字は、entitize オプションによって、エンティティ化されたりエンティティ化されなかったりします

for xml 句におけるデフォルトの NCR オプションは ncr = non_ascii です。

ncr オプションはカラム値だけに適用され、カラム名や tablename、rowname、prefix オプションで指定された名前には適用されません。XML では、要素名や属性名に NCR を使用できません。

header オプション

for xml 句の header オプションは、新しい値 “encoding” を使用できるように拡張されています。

```
header = {yes | no| encoding}
```

header=encoding の場合、ヘッダは次のようになります。

```
<?xml version = "1/0" encoding = "UTF-16?">
```

値 encoding は、XML のコード化宣言を含む XML ヘッダを使用することを示します。

次の場合、デフォルトの header オプションは *no* です。

- 戻り値のデータ型が Unicode データ型である。
- ncr オプションが *non-ascii* である。
- サーバ文字セットが ISO1、ISO8859_15、ascii_7、または UTF-8 である。

それ以外の場合、デフォルトの header オプションは encoding です。

例外

なし。

例

以下のすべての例では、次のコマンドによって生成されるテーブル例を使用します。

```
create table example_I18N_table (name varchar(10) null,
  uvcol univarchar(10) null)
------------------
insert into example_I18N_table values('Arabic',
  u&'¥622¥623¥624¥625¥626')

insert into example_I18N_table values('Hebrew',
  u&'¥5d2¥5d3¥5d4¥5d5¥5d6')

insert into example_I18N_table values('Russian',
  u&'¥410¥411¥412¥413¥414')
```

図 6-1 のテーブル例には 2 つのカラムがあります。

- 言語を示す varchar カラム。
- その言語のサンプル文字を含む univarchar カラム。サンプル文字は、連続する文字の文字列で構成されます。

```
select * from example_I18N_table
name        uvcol
----------------------------------
Arabic      0x06220623062406250626
Hebrew      0x05d205d305d405d505d6
Russian     0x04100411041204130414

(3 rows affected)
```

例 1

変数が指定されていない select コマンドは、テーブルを表示します。

```
select * from example_I18N_table
name        uvcol
--------    ----------------------------
Arabic      0x06220623062406250626
Hebrew      0x05d205d305d405d505d6
Russian     0x04100411041204130414
3 rows affected)
```

例 2

for xml 句を使用して SQL XML ドキュメントを生成するには、次のように入力します。

```
select * from example_I18N_table for xml
-------------------------------------------------
<resultset xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <row>
    <name>Arabic</name>
<uvcol>&#x622;&#x623;&#x624;&#x625;&#x626;</uvcol>
  </row>
  <row>
    <name>Hebrew</name>
    <uvcol>&#x5d2;&#x5d3;&#x5d4;&#x5d5;&#x5d6;</uvcol>
  </row>
  <row>
   <name>Russian</name>
   <uvcol>&#x410;&#x411;&#x412;&#x413;&#x414;</uvcol>
  </row>}
</resultset>
```

例 3

デフォルトでは、生成された SQLX XML ドキュメントは NCR を使用して非 ASCII 文字を表示します。ブラウザの文字セット・プロパティを Unicode に設定している場合、ドキュメントには、それぞれの実際の非 ASCII 文字であるアラビア語、ヘブライ語、ロシア語、または選択した非 ASCII 文字が表示されます。

ブラウザの文字セット・プロパティが Unicode に設定されていない場合は、アラビア語、ヘブライ語、ロシア語の文字は疑問符として表示されます。

例 4

SQLX XML ドキュメントに非 ASCII 文字をプレーンな文字として含める場合は、ncr オプションで *no* を指定します。

```
select * from example_I18N_table for xml
   option 'ncr=no' returns unitext
-----------------------------------------------------
0x000a003c007200650073007500 6c00740073006500740020007 8006d...etc
```

例 5

ターゲットの文字セットとして UTF-16 または UTF-8 を指定して、例 3 で生成された Unicode ドキュメントをクライアント・ファイルに取得すると、ドキュメントをブラウザに表示できます。その後で、選択した実際の非 ASCII 文字を表示します。

例 6

ncr に *ncr=non_ascii* オプションおよび *ncr=non_server* オプションを指定すると、ASCII またはデフォルトのサーバ文字セットでない場合のみ、文字が NCR に変換されます。次の例の式では、ASCII name カラムと Unicode uvcol カラムの両方で ASCII 文字列値が連結されます。この式の結果は、ASCII 文字と非 ASCII 文字の両方を含む文字列になります。生成された SQLX XML ドキュメントでは、非 ASCII 文字だけが NCR に変換されています。

```
select name + '(' + uvcol + ')'  from example_I18N_table2>
   for xml option 'ncr=non_ascii'
-------------------------------
<resultset xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <row>
   <C1>Arabic(&#x622;&#x623;&#x624;&#x625;&#x626;)</C1>
  </row>
  <row>
   <C1>Hebrew(&#x5d2;&#x5d3;&#x5d4;&#x5d5;&#x5d6;)</C1>
  </row>    <
  row>
   <C1>Russian(&#x410;&#x411;&#x412;&#x413;&#x414;)</C1>
```

```
  </row>
</resultset>
```

ブラウザには、それぞれ実際の非 ASCII 文字であるアラビア語、ヘブライ語、ロシア語を示すドキュメントが表示されます。

例 7

ほとんどの文字は [0x20, 0xFFFF] の範囲のコード・ポイントで表され、単一の 16 ビット値を使用して表すことができます。サロゲート・ペアは、[0x010000..0x10FFFF] の範囲内にある文字を表す 16 ビット値のペアです。ペアの前半は [0xD800..0xDBFF] の範囲にあり、ペアの後半は [0xDC00..0xDFFF] の範囲にあります。このようなペア (H、L) は、次のように計算される文字を表します (16 進算術式)。

```
(H - 0xD800) * 400 + (L - 0xDC00)
```

たとえば、文字 “&#x01D6D1” は、サロゲート・ペア D835, DED1 で表される小文字の太字算術記号です。

```
select convert(unitext, u&'¥+1d6d1')
---------------------
0xd835ded1
```

ncr=non_ascii または ***ncr=non_server*** を指定して、非 ASCII データとサロゲート・ペアを含む SQLX XML ドキュメントを生成する場合、そのサロゲート・ペアは単一の文字として表示され、ペアにはなりません。

```
select convert(unitext, u&'¥+1d6d1')
for xml option 'ncr=non_ascii"
-------------------------------
<resultset
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <row>
    <C1>&#x1d6d1;</C1>
 </row>
</resultset>
```

xmlparse における I18N

xmlparse は、入力 XML ドキュメントに対して Unicode データ型 (unichar、univarchar、unitext、java.lang.String) をサポートします。

オプション

xmlparse は、XML ドキュメントを解析し、解析済みドキュメントおよびその内部インデックスを含む image 値としてドキュメントの表現を返します。この表現は、Unicode 解析済みイメージ XML と呼ばれます。Unicode 解析済みイメージ XML は、image カラムに格納されます。

xmlparse は、文字列データ型を Unicode に変換します。文字列データ型は、常に Unicode のサブセットであるサーバ文字セットに含まれます。したがって、変換中に、データ型の変更による変換例外は発生しません。

xmlparse におけるソート順

XML のソート順の詳細については、「XML サービスにおけるソート順」(94 ページ) を参照してください。

xmlparse は、sp_configure オプションの default xml sort order で指定されたソート順を使用し、XML インデックスにも同じ順序を使用します。XML では、ソート順の名前は、ドキュメントの「解析済み XML ソート順」と呼ばれる、xmlparse によって生成されたイメージ内に格納されます。

解析済みの XML ドキュメントを参照するすべての関数は、解析済み XML ソート順が現在のデフォルト XML ソート順と異なる場合に例外を発生させます。

xmlextract における I18N

xmlextract は、XML ドキュメントに XML クエリ式を適用し、ユーザが選択した結果を返します。入力ドキュメントは、string データ型、Unicode データ型、文字データまたは解析済み XML を含む image データ型のいずれかになります。

returns 句では、抽出された値のデータ型として Unicode データ型を指定できます。

NCR オプション

xmlextract でサポートされる ncr オプションは次のとおりです。

```
ncr = {non_ascii|non_server|no}
```

実行時には、次の場合に ncr オプションが適用されます。

- 結果データ型が文字列または Unicode データ型であり、たとえば numeric、datetime、money ではない。
- XPath クエリに text() が指定されていない。

デフォルトの ncr オプションは次のとおりです。

- 戻り値のデータ型が Unicode データ型の場合、デフォルト値は ncr=no です。
- 戻り値のデータ型が文字列データ型の場合、デフォルト値は ncr=non_server です。

xmlextract におけるソート順

xmlextract におけるソート順については、「XML サービスにおけるソート順」(94 ページ) を参照してください。

xmlextract は、サーバにおける現在のデフォルト・ソート順ではなく、入力 XML ドキュメントに格納された解析済み XML ソート順を使用します。

XML サービスにおけるソート順

***sp_configure* オプション**　XML サービスでは、sp_configure オプションの default xml sort order を定義します。このオプションには、次の 3 つの特性があります。

- 静的である。この設定を実行するには、Adaptive Server を再起動する必要があります。
- オプション値が Unicode ソート順の名前である。詳細については、『システム管理ガイド第 1 巻』のデフォルトの Unicode ソート順の表を参照してください。
- デフォルトのオプション値は *binary* です。

xmlparse　xmlparse は、引数ドキュメントの解析済み表現を返します。これには、ドキュメントの要素と属性のインデックス、およびその値も含まれます。この解析済み表現では、ドキュメントを解析したときに存在した *default xml sort order* が指定されます。

xmlextract　xmlextract は、“//book[author='John Doe']” などの条件を比較する XPath クエリを評価します。xmlextract では、現在の *default xml sort order* と、ドキュメントの *parsed xml sort order* が比較されます。ソート順が異なる場合は、例外が発生します。

xmlextract は、サーバにおける現在のデフォルト・ソート順ではなく、入力 XML ドキュメントに格納された XML ソート順を使用します。

注意　XML サービスでは、単一のデフォルト順序である default xml sort order が使用されます。default Unicode xml sort order と default xml sort order の両方が使用されることはありません。

---

default xml sort order の変更　sp_configure を使用して *default xml sort order* を修正できます。

default xml sort order を修正したら、すでに解析した XML ドキュメントを、Adaptive Server の update コマンドを使用して再解析できます。update については、『リファレンス・マニュアル：コマンド』を参照してください。

```
update xmldocs
set doc = xmlparse(xmlextract('/', doc))
```

xmlvalidate における I18n

xmlvalidate() は、Unicode データ型 (unichar、univarchar、unitext、および java.lang.String) と、string および image データ型をサポートします。xmlvalidate の returns 句では、抽出された値のデータ型として Unicode データ型を指定できます。

NCR オプション

xmlvalidate() でサポートされる ncr オプションは次のとおりです。

```
ncr={non_ascii | non_server | no}
```

実行時、ncr オプションは result 句のデータ型が string または Unicode データ型の場合のみ適用されます。たとえば、オプションは numeric、datetime、または money データ型には適用されません。

デフォルトの NCR オプション

- NCR オプションのデフォルト値は、returns のデータ型が Unicode データ型 (unichar、univarchar、unitext、または java.lang.String) の場合、ncr=no です。
- NCR オプションのデフォルト値は、returns のデータ型が string データ型 (char、varchar、または text) の場合、ncr=non_server です。

第 7 章

xmltable()

この章では、xmltable() 関数について詳細に説明します。

トピック名	ページ
概要	97
xmltable と derived table の構文	97

概要

xmltable() は、XML ドキュメントから複数の値を持つ要素のシーケンスを抽出し、これらの要素の SQL テーブルを構築します。xmltable() を 1 度呼び出すと、繰り返しのたびに xmlextract を複数回呼び出す T-SQL ループに置き換えられます。この関数は抽出テーブル (別の SQL クエリの from 句で指定されたカッコで囲んだサブクエリ) として呼び出されます。xmltable() の呼び出しは、xmltable() によって生成されたテーブルの各行に対して xmlextract 式を 1 回実行するのと同等です。

xmltable() は xmlextract を一般化したものです。両方の関数とも関数の引数である XML ドキュメントから抽出したデータを返します。これらの違いは次のとおりです。

- xmlextract は、1 つの XPath クエリによって識別されたデータを返します。
- xmltable() は、XPath クエリによって識別されたデータのシーケンス、あるいはロー・パターンを抽出し、そのシーケンスの各要素から他の XPath クエリのリストによって識別されたデータ、あるいはカラム・パターンを抽出します。SQL テーブルのすべてのデータを返します。

xmltable と derived table の構文

構文に関する以下の項では、xmltable() の基本的な構文と、xmltable() を使用する場所および方法について説明します。

xmltable

説明

XML ドキュメントからデータを抽出し、SQL テーブルとして返します。

構文

xmltable_expression ::= xmltable
(*row_pattern* passing *xml_argument*
columns *column_definitions*
options_parameter)
row_pattern ::= *character_string_literal*
xml_argument ::= xml_expression
column_definitions ::=
column_definition [{ , *column_definition* }]
column_definition ::=
ordinality_column | *regular_column*
ordinality_column ::= *column_name* datatype **for ordinality**
regular_column ::=
column_name datatype [**default** *literal*] [**null** | **not null**]
[**path** *column_pattern*]
column_pattern ::= *character_string_literal*
options_parameter ::=[,] **option** *option_string*
options_string ::= *basic_string_expression*

抽出テーブルの構文 SQL の from 句から SQL テーブルを返します。

from_clause ::= from *table_reference* [, *table_reference*]...
table_reference ::= *table_view_name* | *ANSI_join* |*derived_table*
table_view_name::=See the select command in Reference Manual Volume 2, "Commands".
ANSI_join::=See the select command in Reference Manual Volume 2, "Commands".
derived_table ::=
(subquery) as *table_name* [(*column_name* [, *column_name*]...)
xmltable_expression as *table_name*

例

抽出テーブルとしての xmltable この例は、抽出テーブルを返す簡単な xmltable() 呼び出しを示します。

```
select * from xmltable('/doc/item'
    passing '<doc><item><id>1</id><name>Box</name></item>'
        +'<item><id>2</lid><name>Jar</name></item></doc>'
    columns id int path 'id',name varchar(20) path 'name') as items_table
id                  name
-----               -----
1                   Box
2                   Jar

(2 rows affected)
```

例 1 この抽出テーブルの構文では、参照しない場合もテーブル名 (items_table) を指定する必要があります。たとえば、以下の例は誤りです。

```
select * from xmltable ('/doc/item'
    passing '<doc><item><id>1</id><name>Box</name></item>'
        +'<item><id>2</id><name>Jar</name></item></doc>'
    columns id int path 'id', name varchar(20) path 'name')
-------------
Msg 102 Level 15, State 1:
Incorrect syntax near ')'
```

簡単なドキュメント参照の例　ドキュメント参照では、passing の後の引数は入力 XML ドキュメントです。以下の例では、ドキュメントがリテラル文字列として指定されています。

```
select * from xmltable ('/doc/item'
    passing '<doc><item><id>1</id><name>Box</name></item>'
        +'<item><id>2</id><name>Jar</name></item></doc>'
    columns id int path 'id', name varchar(20) path 'name') as items_table

id              name
-------         ------
1               Box
2               Jar

(2 rows affected)
```

例 2 次の例は、T-SQL 変数にドキュメントを格納し、その変数を xmltable() 呼び出しで参照します。

```
declare @doc varchar(16384)
set @doc='<doc><item><id>1</id></name>Box</name></item>'
        +'<item><id>2</id><name>Jar</name></item></doc>'

select * from xmltable(/doc/item' passing @doc
    columns id int path 'id', name varchar(20) path 'name') as items_table

id            name
----          -----
1             Box
2             Jar

(2 rows affected)
```

例 3 ドキュメントをテーブルに格納し、スカラ・サブクエリで参照するには、次のようにします。

```
select 100 as doc_id,
'<doc><item><id>1</id><name>Box</name></item><item><id>2</id>
    <name>Jar</name></item></doc>'as doc
into #sample_docs
select* from xmltable('/doc/item'
    passing(select doc from #sample_docs where doc_id=100)
```

```
    columns id int path 'id',name varchar(20) path 'name') as items_table

id            name
-------       -----
1             Box
2             Jar

(2 rows affected)
```

ロー・パターン　xmltable 呼び出しの最初の引数 *row-pattern* (' /doc/item') は、結果が指定したドキュメントの要素のシーケンスである XPath クエリ式です。xmltable 呼び出しは、シーケンスの要素ごとに 1 つのローのテーブルを返します。

例 4 ロー・パターンで空のシーケンスが返されると、結果は次のように空のテーブルになります。

```
select * from xmltable ('//item_entry'
    passing '<doc><item><id>1</id><name>Box</name></item>'
        +'<item><id>2</id><name>Jar</name></item></doc>'
    columns id int path 'id',
        name varchar(20) path 'name') as items_table

id               name
-------         ------


(0 rows affected)
```

例 5 ロー・パターン式を XPath 関数にすることはできません。

```
select * from xmltable ('/doc/item/tolower()'
    passing '<doc><item><id>1</id><name>Box</name></item>'
        +'<item><id>2</id><name>Jar</name></item></doc>'
    columns id int path 'id',
        name varchar(20) path 'name') as items_table

id    name
---  ------
Msg 14825, Level 16, State 0:
Line1:
XPath function call must be at leaf level.
```

カラム・パターン　columns キーワードに続く引数は、カラム定義のリストです。各カラム定義は、create table のように、カラム名とデータ型、およびカラム・パターンと呼ばれるパスを指定します。*column-pattern* は、*row-pattern* によって返されたシーケンスの要素に適用して結果テーブルのカラムのデータを抽出する XPath クエリ式です。

例 6 カラムのデータが XML 属性に含まれている場合は、"@" を使用してカラム・パターンを指定し、属性を参照します。次に例を示します。

```
select * from xmltable ('/doc/item'
    passing '<doc><item id>"1"><name>Box</name></item>'
        +'<item id="2">/id><name><Jar</name></item></doc>'
    columns id int path '@id', name varchar(20)) as items_table

id              name
--------------------
1               Box
2               Jar

(2 rows affected)
```

デフォルトのカラム・パターン　*column-pattern* は一般的に、name のように、指定されている *column_name* と同じです。この場合、column_pattern を省略すると、デフォルトで *column_name* が使用されます。

```
select * from xmltable ('/doc/item'
    passing '<doc><item><id>1</id><name>Box</name></item>'
        +'<item><id>2</id><name>Jar</name></item></doc>'
    columns id int, name varchar(20)) as items_table

id              name
--------------------
1               Box
2               Jar

(2 rows affected)
```

例 7 カラム・パターンでカラム名をデフォルトに使用する場合は、値が XML 属性であるカラムに引用符付き識別子を使用します。次のように結果でカラム名を参照する場合は、その識別子を引用符で囲む必要があります。

```
set quoted_identifier on
select "@id", name from xmltable ('/doc/item'
    passing '<doc><item id="1"><name>Box</name></item>'
        +'<item id="2"><name>Jar</name></item></doc>'
    columns "@id" int, name varchar(20)) as items_table

@id              name
--------------------
1               Box
2               Jar

(2 rows affected)
```

例 8 引用符付き識別子を使用することで、さらに複雑な XPath 式を使用してカラム名をデフォルトのカラム・パターンに指定できます。次に例を示します。

```
set quoted_identifier on
select "@id", "name/short", "name/full" from xmltable ('/doc/item'
    passing '<doc><item id="1"><name><short>Box</short>
             <full>Box, packing, moisture resistant, plain</full>
             </name></item>'
    +'<item id="2"><name><short>Jar</short>
             <full>Jar, lidded, heavy duty</full>
             </name></item></doc>'
    columns "@id" int, "name/short" varchar(20), "name/full" varchar(50))
    as items_table

@id              name/short             name/full
--------------------
1               Box                     Box, packing, moisture resistant, plain
2               Jar                      Jar, lidded, heavy duty

(2 rows affected)
```

Implicit text() この例は、カラム・パターンでは通常暗黙的な関数 text() を示します。text() は XML 要素タグを削除します。たとえば、以下の XPath クエリは選択された要素を XML マークアップ付きで返します。

```
1> declare @doc varchar(16384)
2> set @doc= '<doc><item><id>1</id></name>Box</name></item>'
             +<'item><id>2</id><name>Jar</name></item></doc>'
3> select xmlextract('/doc/item[2]/name', @doc)
------------
<name>Jar</name>
```

例 9 text() を XPath クエリに追加すると、XML タグが削除されます。

```
1> declare @doc varchar(16384)
2> set @doc= '<doc><item><id>1</id></name>Box</name></item>'
             +<'item><id>2</id><name>Jar</name></item></doc>'

3> select xmlextract('/doc/item[2]/name/text()', @doc)
------------
Jar
```

例 10 text() は、ほとんどのカラム・パターンでは暗黙的です。次の例は、id または name カラムのいずれにも、カラム・パターンに text() を指定しません。

```
select * from xmltable ('/doc/item'
    passing '<doc><item><id>1</id><name>Box</name></item>'
        +'<item><id>2</id><name>Jar</name></item></doc>'
    columns id int path 'id', name varchar(20) path 'name') as items_table

id             name
---            -------
1              Box
2              Jar

(2 rows affected

                  )
```

データ型変換　暗黙の SQL convert 文をカラム・パターンから抽出されたデータに適用することで、データ型変換のカラム値を導出できます。次に例を示します。

```
select * from xmltable ('/emps/emp'
    passing '<emps>
<emp><id>1</id><salary>123.45</salary><hired>1/2/2003</hired></emp'
+'<emp><id>2</id><salary>234.56</salary><hired>2/3/2004</hired></emp>'
        +</emps>'
    columns id int path 'id', salary dec(5,2), hired date)
as items_table

id          salary        hired
-------     ---------     --------
1           123.45        Jan 2, 2003
2           234.56        Feb 3, 2004
(2 rows affected)
```

例 11 カラム用に抽出された XML データは、カラムのデータ型に変換可能である必要があります。変換できない場合、例外が発生します。

```
select * from xmltable ('/emps/emp'
    passing '<emps>
+<emp><id>1</id><salary>123.45</salary><hired>1/2/2003</hired></emp'
+'<emp><id>2</id><salary>234.56 C$</salary><hired>2/3/2004</hired></emp>'
+</emps>'
columns id int path 'id', salary dec(5,2), hired date)
as items_table
----------------------
Msg 14841, Level 16, State 3:
Line 1:
XMLTABLE:Failed to convert column pattern result to DECML for column 1.
```

例 12 フォーマットが SQL convert 関数に適していない XML データを処理するには、データを文字列カラムに抽出します (varchar、text、image、java.lang.String)。

```
select * from xmltable ('/emps/emp'
    passing '<emps>
+<emp><id>1</id><salary>123.45</salary><hired>1/2/2003</hired></emp'
+'<emp><id>2</id><salary>234.56 </salary><hired>2/3/2004</hired></emp>'
+</emps>'
columns id int, salary varchar(20), hired date)
as items_table

id            salary         hired
---------------------------------
1             123.45        Jan 2, 2003
2             234.56       Feb 3, 2004
(2 rows affected)
```

ordinality カラム　XML ドキュメント内の要素の順序には意味がある場合があります。

要素は、格納された要素の値の順に並べられることがあります。以下の例では、*<item>* 要素が、格納された *<id>* 要素の値の順に並べられています。

```
<doc>
    <item><id>1<name>Box</name></item>'
    <item><id>2<name>Jar</name></item>'
</doc>
```

任意だが意味のある方法で、要素の順序を決めることもできます。以下の例では、*<item>* 要素の順序は値に基づいていませんが、優先度順である先入れ先出しを反映している可能性があります。そのような順序は、データのアプリケーションに対して意味を持つ場合があります。

```
<doc>
    <item><id>25<name>Box</name></item>'
    <item><id>15<name>Jar</name></item>'
</doc>
```

例 13 xmltable で *ordinality_column* を使用して、入力 XML ドキュメントの要素の順序付けを記録できます。

```
declare @doc varchar(16384)
set @doc = '<doc><item><id>25<name>Box</name></item>'
    +'<item><id>15</id><name>Jar</name></item></doc'
select * from xmltable('/doc/item' passing @doc
    columns item_order int for ordinality,
        id int path 'id',
        name varchar(20) path 'name') as items_table
order by item_order

item_order         id            name
----------         ---           ----
```

```
1                   25            Box
2                   15            Jar
(2 rows affected)
-----------
```

for ordinality 句と item_order カラムを指定しない場合、id 25 のローが id 15 に先行することを示す id も name カラムもありません。for ordinality 句は、出力 SQL ローを入力 XML ドキュメントの要素の順序付けと同じ順序にすることができます。

ordinality カラムのデータ型には、int、tinyint、bigint、numeric、decimal の任意の固定数値データ型を使用できます。numeric および decimal には 0 の位取りが必要です。ordinality 句には real と float は使用できません。

NULL 値　カラム・パターンによって空の結果が返された場合の対応は、`default` 句と `{null | not null}` 句によって異なります。

例 14 次の例は、<*name*> 要素を 2 番目の <*item*> から省略します。name カラムでは、デフォルトで名前を使用できます。

```
select * from xmltable ('//item'
    passing '<doc><item><id>1</id><name>Box</name></item>'
    +'<item><id>2</id></item></doc>'
columns id int path 'id', name varchar(20), path 'name')
as items_table
-------------------------
id            name
------------------
1             Box
2             NULL

(2 rows affected)
```

例 15 次の例は、<*name*> 要素を 2 番目の <*item*> から省略し、name カラムに not null を指定します。

```
select * from xmltable ('/doc/item'
    passing '<doc><item><id>1</id><name>Box</name></item>'
    +'<item><id>2</id></item></doc>'
columns id int path 'id', name varchar(20) not null path 'name')
as items_table
-------------------------
Msg 14847, Level 16, State 1:
Line 1:
XMLTABLE column 0, does not allow null values.
```

例 16 次の例は、default 句を name カラムに追加し、<*name*> 要素を 2 番目の <*item*> から省略します。

```
select * from xmltable ('/doc/item'
    passing '<doc><item><id>1</id><name>Box</name></item>'
    +'<item><id>2</id></item></doc>'
columns id int path 'id' name varchar(20) default '***' path 'name')
as items_table

id            name
------------------
1             Box
2             ***
(2 rows affected)
```

***xmltable* 呼び出しのコンテキスト**　以下の例は、抽出テーブル式で xmltable 呼び出しに使用できる SQL コマンドを示します。

例 17 select － xmltable() は単純な select 文で使用できます。

```
select * from xmltable ('/doc/item'
    passing '<doc><item><id>1</id><name>Box</name></item>'
    +'<item><id>2</id><name>Jar</name></item></doc>'
columns id int path 'id'
    name varchar(20) path 'name')as items_table

id            name
--            ----
1             Box
2             Jar
(2 rows affected)
```

例 18 ビュー定義 － ビュー定義で xmltable を使用して select を指定します。以下の例では、ドキュメントをテーブルに保管し、xmltable を使用してテーブルからデータを抽出することによって、保管されたドキュメントを create view 文で参照します。

```
select 100 as doc_id,
'<doc><item><id>1</id><name>Box</name></item>'
    +'<item><id>2</id><name>Jar</name></item></doc>' as doc
into sample_docs
create view items_table as
    select * from xmltable('/doc/item'
        passing (select doc from sample_docs where doc_id=100)
        columns id int path 'id'
        name varchar(20) path 'name')as xml_extract

id            name
------------------
1             Box
2             Jar
(2 rows affected)
```

例 19 カーソル宣言 − xmltable を使用してカーソルを宣言することで、select を指定します。

```
declare cursor C for
select * from xmltable ('/doc/item'
    passing (select doc from sample_docs where id=100)
    columns id int path 'id'
    name varchar(20) path 'name')as items_table
declare @idvar int
declare @namevar varchar(20)
open C
while @@sqlstatus=0
begin
fetch C into @idvar, @namevar
print 'ID "%1!" NAME"%2!"', @idvar, @namevar
end
-------------------------
ID "1" NAME "Box"
ID "2" NAME "Jar"

(2 rows affected)
```

他のテーブルから update、insert、delete を実行するなど、生成した各ローで複数の動作が必要になるアプリケーションでは、生成された各ローのデータに基づいて、カーソル・ループを使用して xmltable() の結果を処理できます。または xmltable 結果をテンポラリ・テーブルに保存して、カーソル・ループでそのテーブルを処理することもできます。

例 20 select into − select into で xmltable を使用して select を指定します。

```
select * into #extracted_table
from xmltable('/doc/item'
    passing (select doc from sample_docs where doc_id=100
    columns id int path 'id'
        name varchar(20) path 'name') as items_table

select * from #extracted_table

id            name
----          ------------
1             Box
2             Jar
```

例 21 insert − insert コマンドで xmltable を使用して select を指定します。

```
create table #extracted_data (idcol int, namecol varchar(20))
insert into #extracted_data
select * into #extracted_table from xmltable('/doc/item'
    passing (select doc from sample_docs where doc_id=100
    columns id int path 'id', name varchar(20) path 'name')as items_table
select * from extracted_data

id             name
```

```
-----          ---------
1              Box
2              Jar
(2 rows affected)
```

例 22 スカラ・サブクエリ ― xmltable() をサブクエリで使用して select を指定します。xmltable は SQL テーブルを返すため、スカラ・サブクエリで集合か選択のいずれかを実行して、単一のローとカラムをスカラ・サブクエリ結果に返す必要があります。

```
declare @idvar int
set @idvar = 2
select @idvar,
(select name from xmltable ('/doc/item'
    passing(select doc from sample_docs where doc_id=100
    columns id int path 'id',name varchar(20) path 'name') as item_table
where items_table.id=@idvar)
----------------------------
2                     Jar
(1 rows affected)
```

例 23 ジョイン ― カンマ・リストのジョイン、または外部ジョインを使用して、xmltable の結果を他のテーブルにジョインします。

```
create table prices (id int, price decimal (5,2))
insert into prices values(1,123.45)
insert into prices values (2,234.56)
select prices.id,extracted_table.name, prices.price
from prices,(select * from xmltable('/doc/item'
    passing (select doc from sample_docs where doc_id=100
    columns id int path 'id', name varchar(20) path 'name')as a) as
            extracted_table
where prices.id=extracted_table.id

id       name        price
-----          ---------
1              Box   123.45
2              Jar   234.56
(2 rows affected)
```

ドキュメントのテーブルの処理　xmltable() を XML ドキュメントのテーブルの各ローに適用できます。たとえば、次の例では 2 つのカラムを含むテーブルを作成します。

- pubs2_publishers テーブル内の 3 つのパブリッシャのいずれかの pub_id。
- そのパブリッシャにより発行された各ドキュメントのタイトルおよび価格を含む XML ドキュメント。例のテーブルのサイズを小さくするため、価格が $15.00 を超えるタイトルのみ含められます。

```
create table high_priced_titles
(pub_id char(4), titles varchar (1000))
insert into high_priced_titles
select p.pub_id,
    (select title_id, price from pubs2..titles t,pubs2..publishers p
       where price> 15 and t.pub_id=p.pub_id
       for xml
       option 'tablename=expensive_titles, rowname=title')
          returns varchar(1000))as titles
from pubs2..publishers p
select * from high_priced_titles
--------------
pub_id    titles
------    ------------------------------------------------
0736             <expensive_titles>
    <title> <title_id>PS3333</title_id> <price>19.99</price></title>
                 </expensive_titles>

0877             <expensive_titles>
    <title> <title_id>MC2222</title_id> <price>19.99</price></title>
    <title> <title_id>PS1372</title_id> <price>21.59</price></title>
    <title> <title_id>TC3218</title_id> <price>20.95</price></title>
                 </expensive_titles>

01389             <expensive_titles>
    <title> <title_id>BU1032</title_id> <price>19.99</price></title>
    <title> <title_id>BU7832</title_id> <price>19.99</price></title>
    <title> <title_id>PC1035</title_id> <price>22.95</price></title>
    <title> <title_id>PC8888</title_id> <price>20.00</price></title>
                 </expensive_titles>
(3 rows affected)
```

例 24　スカラ・サブクエリで xmltable を使用して、SQL テーブルのように各ローの XML ドキュメントを処理します。たとえば、各出版社のタイトルの最高価格をリストします。

```
select pub_id
(select max(price)
 from xmltable('//title'passing hpt.titles
        columns title_id char(4), price money)
        as extracted_titles, high_priced_titles hpt) as max_price
 from high_priced_titles hpt
--------------
pubid         max_price
----------------------
0736          19.99
0877          21.59
1389          22.95
```

この high_priced_titles テーブルは実質的に階層です。各ローは中間的なノードであり、その title カラムには XML ドキュメント内の各 *title* 要素のリーフ・ノードが含まれています。high_priced_titles には 3 つのローがあります。

その階層をフラット化して、*title* 要素ごとにローを持つテーブルを生成できます。titles カラム内のデータをフラット化してテーブルを生成するため、8 つのローを持つ high_priced_titles_flattened (*titles/title* 要素ごとに 1 つずつ) が、以下の解決法のいずれかを使用します。

解決法 1　high_priced_titles を処理して xmltable を各ロー内のタイトル・ドキュメントに適用するループを使用することで、high_priced_titles_flattened を生成できます。以下の例では、from 句に注目してください。

```
from(select @pub_id_var)as ppp,
    xmltable('//title' passing @titles_var
        columns title_id char(6),price money)as ttt
```

変数 *@pub_id_var* および *@titles_var* は、high_priced_titles の現在のローの pub_id および titles カラムです。from 句は、2 つの抽出テーブルをジョインします。

- `(select @pub_id_var) as ppp`

  これは、pub_id を含む、1 つのローと 1 つのカラムを持つテーブルです。

- `xmltable(...) as ttt`

  これは、現在の high_priced_titles ローの *titles* ドキュメント内に各 *title* 要素のローを持つテーブルを生成します。

階層をフラット化するには、これらの 2 つの抽出テーブルをジョインすることで、titles カラムから生成された各ローに pub_id カラムを付加します。

```
create table high_priced_titles_flattened_1
(pub_id char(4), title_id(char(6), price money)

declare C cursor for select * from high_priced_titles
declare @pub_id_var char(4)
declare @titles_var char(1000)
open C

while @@sqlstatus =0
begin
fetch C into @pub_id_var, @titles_var

insert into high_priced_titles_flattened_1
select *
from (select @pub_id_var) as ppp,(col1),
    xmltable('//title' passing @titles_var
    columns title_id char (6), price money) as ttt
end
select * from high_priced_titles_flattened_1
```

```
pub_id    title_id          price
------    ------------------------------------------------
0736              PS3333               19.99
0877              MC2222               19.99
0877              PS1372               21.59
0877              TC3218               20.95
1389              BU1032               20.95
1389              BU7832               19.99
1389              PC1035               19.99
1389              PC8888               20.00
```

解決法 2　特別なジョインを使用して high_priced_titles テーブルを生成することもできます。

この例では、2 つのテーブル (high_priced_titles as hpt と、xmltable により生成されるテーブル) をジョインします。xmltable の passing 引数は、先行する hpt テーブルを参照します。通常、同じ from 句内の抽出テーブル式で、from 句内のテーブルを参照することは無効です。ただし、同じ from 句内の他のテーブルが同じ from 句内の xmltable 呼び出しより先行していれば、xmltable はそれらのテーブルを参照できます。

```
select hpt.pub_id, extracted_titles.*
into high_priced_titles_flattened_3
from high_priced_titles as hpt,
       xmltable('//title'
       passing htp.titles,
       columns
           title_id char(6)
           price money)as extracted_titles
pub_id           title_id            price
---------        --------------      ---------
0736             PS3333              19.99
0877             MC2222              19.99
0877             PS1372              21.59
0877             TC3218              20.95
1389             BU1032              20.95
1389             BU7832              19.99
1389             PC1035              19.99
1389             PC8888              20.00
```

使用法

- xmltable は、組み込みのテーブル値関数です。
- xmltable 式の結果タイプは SQL テーブルで、そのカラム名とデータ型は *column_definitions* によって指定されます。
- これらのキーワードは xmltable に関連付けられています。
  - 予約されている：for、option
  - 予約されていない：columns、ordinality、passing、path、xmltable

- xmltable 呼び出しの引数の式は、xmltable 呼び出しを含む from 句の前のテーブルのカラム名を参照できます。xmltable 呼び出しに先行するテーブルだけを参照できます。このように、同じ from 句の前のテーブルのカラムを参照することをラテラル参照と言います。次に例を示します。

```
select * from T1, xmltable(...passing T1.C1...)
       as XT2, xmltable(...passing XT2.C2...)as XT3
```

  最初の xmltable 呼び出しでの T1.C1 の参照は、テーブル T1 のカラム C1 のラテラル参照です。2 番目の xmltable 呼び出しの XT2.C2 の参照は、最初の xmltable 呼び出しによって生成されたテーブルのカラム C2 のラテラル参照です。

- xmltable を update または delete 文の from 句で使用することはできません。たとえば、次の文は失敗します。

```
update T set T.C=...
from T,xmltable(...)
where...
```

- xmltable 式により返される SQL テーブルを更新することはできません。
- *regular_columns* のデータ型には任意の SQL データ型を使用できます。
- *regular_column* で default に続く literal は、カラムのデータ型に割り当て可能である必要があります。
- *ordinality_column* は 1 つしか使用できません。この変数に指定するデータ型は、integer、smallint、tiny int、decimal、numeric である必要があります。decimal と numeric には位取り 0 を使用する必要があります。
- *ordinality_column* がある場合、null 入力はできません。他のカラムの null 入力可能プロパティは、`{null | not null}` 句によって指定されます。デフォルトは null です。

  注意 このデフォルトは create table のデフォルト値とは異なります。

- set quoted_identifier の現在の設定は、xmltable 式の句に適用されます。次に例を示します。
  - set quoted_identifier が on の場合、カラム名に引用符付き識別子を使用できます。その場合、*row_pattern*、*column_pattern*、**デフォルト**のリテラルにある各文字列リテラルを一重引用符で囲む必要があります。
  - set quoted_identifier が off の場合、カラム名に引用符付き識別子を使用することはできません。*row_pattern*、*column_pattern*、**デフォルト**のリテラルにある各文字列リテラルは、一重引用符か二重引用符のいずれかで囲むことができます。
- option_string の一般的なフォーマットについては、「option_strings：一般的なフォーマット」を参照してください。

xmltable のロー・パターンとカラム・パターン　xmltable のロー・パターンとカラム・パターンでは、単純なパスだけを使用できます。XPath の単純なパスは、'/' と要素名および属性名を使用した前方検索だけで構成されます。

- *row_pattern* が *xml_argument* によって指定されたドキュメントのルート・レベルから開始されないと、例外が発生します。ロー・パターンは XML ドキュメントのルートから開始する必要があります。
- *row_pattern* で XML 関数を指定すると、例外が発生します。ロー・パターンで XML 関数を指定することはできません。
- column_definition でパスを指定しないと、デフォルトの column_pattern がカラム定義の column_name になります。このデフォルトは、サーバのパスワードの大文字と小文字の区別に従います。たとえば、次の文があるとします。

  ```
  select * from xmltable(...columns name varchar(30),...)
  ```

  サーバが大文字と小文字を区別しない場合、これは次の文と等しくなります。

  ```
  select * from xmltable(...columns name varchar(30) path
  'name',...)
  ```

  サーバが大文字と小文字を区別する場合、最初の文は次の文と等しくなります。

  ```
  select * from xmltable
  (...columns name varchar(30)path 'NAME',...)
  ```

結果テーブルのローの生成

xmltable 式の結果値は T-SQL テーブル RT として次のように定義されます。

- RT は、*row_pattern* を *xml_argument* に適用した結果として得られる XML シーケンスの各要素にローがあります。
- RT のローには各 *column_definition* のカラムがあり、その *column_name* とデータ型は *column_definition* によって指定されます。
- *column_definition* が *ordinality_column* である場合、N 番目のローの値は整数 N になります。
- *column_definition* が *regular_column* である場合、N 番目のローの値は次のようになります。
  - この XPath 式を *xml_argument* に適用した結果が XVAL になります。

    ```
    (row_pattern[N])/column_pattern/text()
    ```

- XVAL が空で、*column_definition* にデフォルトの句が含まれている場合、カラムの値はそのデフォルトの値になります。

  XVAL が空ではなく、*column_definition* を null にしないように指定すると、例外が発生します。

  それ以外の場合、カラムの値は null 値になります。

- XVAL が空ではなく、カラムのデータ型が char、varchar、text、unitext、unichar、univarchar、java.lang.String である場合は、XVAL を非エンティティ化してください。
- カラムの値は次の式の結果になります。

  ```
  convert(datatype,XVAL)
  ```

参照

xmltable を使用したサンプル・アプリケーションについては、『XML サービス』の「付録 D xmltable() のサンプル・アプリケーション」を参照してください。

付 録 A

sample_docs サンプル・テーブル

XML クエリ関数の説明では、sample_docs という名前のテーブルの例を参照します。この章では、このテーブルの作成方法と移植方法を説明します。

sample_docs テーブルには、3 つのカラムと 3 つのローがあります。

sample_docs テーブルのカラムとロー

この項では、sample_docs テーブルの構造を説明します。

sample_docs テーブルのカラム

sample_docs テーブルには次の 3 つのカラムがあります。

- name_doc
- text_doc
- image_doc

示されているドキュメント例では、name_doc は識別名、text_doc は text 表現のドキュメント、image_doc は image カラムに格納されている解析済み XML 表現のドキュメントをそれぞれ指定します。次のスクリプトはテーブルを作成します。

```
create table sample_docs
(name_doc varchar(100),
text_doc text null,
image_doc image null)
```

sample_docs テーブルのロー

sample_docs テーブルには次の 3 つのローがあります。

- ドキュメント例である “bookstore.xml”
- pubs2 データベースの publishers テーブルの XML 表現
- pubs2 データベースの titles テーブルの (選択されたカラムの) XML 表現

次のスクリプトは、ドキュメント例 “bookstore.xml” を sample_docs テーブルのローに挿入します。

```
insert into sample_docs
    (name_doc, text_doc)
    values ( "bookstore",

"<?xml version='1.0' standalone = 'no'?>
<?PI_example Process Instruction ?>
<!--example comment-->
<bookstore specialty='novel'>
<book style='autobiography'>
    <title>Seven Years in Trenton</title>
       <author>
          <first-name>Joe</first-name>
          <last-name>Bob</last-name>
           <award>Trenton Literary Review
             Honorable Mention</award>
      </author>
      <price>12</price>
</book>
<book style='textbook'>
   <title>History of Trenton</title>
      <author>
          <first-name>Mary</first-name>
           <last-name>Bob</last-name>
           <publication>Selected Short Stories of
           <first-name>Mary</first-name>
            <last-name>Bob</last-name>
           </publication>
      </author>
       <price>55</price>
</book>
<?PI_sample Process Instruction ?>
<!--sample comment-->
<magazine style='glossy' frequency='monthly'>
  <title>Tracking Trenton</title>
     <price>2.50</price>
   <subscription price='24' per='year'/>
</magazine>
<book style='novel' id='myfave'>
     <title>Trenton Today, Trenton Tomorrow</title>
     <author>
```

```
            <first-name>Toni</first-name>
            <last-name>Bob</last-name>
            <degree from='Trenton U'>B.A.</degree>
             <degree from='Harvard'>Ph.D.</degree>
             <award>Pulizer</award>
             <publication>Still in Trenton</publication>
              <publication>Trenton Forever</publication>
        </author>
        <price intl='canada' exchange='0.7'>6.50</price>
        <excerpt>
        <p>It was a dark and stormy night.</p>
        <p>But then all nights in Trenton seem dark and
            stormy to someone who has gone through what
            <emph>I</emph> have.</p>
         <definition-list>
            <term>Trenton</term>
            <definition>misery</definition>
         </definition-list>
    </excerpt>
</book>

<book style='leather' price='29.50'
xmlns:my='http://www.placeholdernamehere.com/schema/'>
    <title>Who's Who in Trenton</title>
    <author>Robert Bob</author>
</book>

</bookstore>")
```

sample_docs テーブル

sample_docs テーブルの他の 2 つのローは、pubs2 データベースの publishers テーブルと titles テーブルの XML 表現です。pubs2 データベースは、『Transact-SQL ユーザーズ・ガイド』で説明されているテーブル例のデータベースです。

publishers テーブルと titles テーブルは、このサンプル・データベース内のテーブルです。例を短くするために、titles テーブルの XML 表現には選択されたカラムのみが含まれています。

テーブル・スクリプト (publishers テーブルの場合)

次の 2 つの insert 文は、publishers テーブルのローと authors テーブルのローを sample_docs テーブルに追加します。各ローには、ロー ('publishers'、'titles') を識別するカラムと、対応する pubs2 テーブルの XML 表現を提供する text_doc カラムが含まれます。XML ドキュメントを生成するには、for xml オプションを指定して select コマンドを呼び出します。

```
insert into sample_docs (name_doc, text_doc)
select 'publishers',
(select * from publishers for xml)

insert into sample_docs (name_doc, text_doc)
select 'titles', (select title_id, title, type, pub_id, price,
advance, total_sales
from titles for xml)
```

publishers テーブルの表現

次のサンプル・コードは、「sample_docs テーブル」(117 ページ) のスクリプトで生成された、Pubs 2 データベース内の *publishers* テーブルの XML 表現を示します。

```
set stringsize 16384
select text_doc from sample_docs
where name_doc='publishers'

text_doc
-----------------------------------------
<publishers
  xmlns:xsi="http://www.w3.org/2001/XMLSchema
  instance">

<row>
   <pub_id>0736</pub_id>
   <pub_name>New Age Books</pub_name>
   <city>Boston</city
   <state>MA</state>
</row>

<row>
   <pub_id>0877</pub_id>
   <pub_name>Binnet & Hardley</pub_name>
   <city>Washington</city>
   <state>DC</state>
</row>

<row>
```

```
   <pub_id>1389</pub_id>
   <pub_name>Algodata Infosystems</pub_name>
   <city>Berkeley</city>
   <state>CA</state>
</row>

</publishers>
(1 row affected)
```

titles テーブルの表現

この項では、titles テーブルの選択済みカラムの XML 表現を示します。

```
set stringsize 16384
select text_doc from sample_docs
where name_doc='titles'

 text_doc
-----------------------------------------------------
<titles
xmlns:xsi=”http://www.w3.org/2001/XMLSchema-instance”>

  <row>
     <title_id>BU1032<title_id>
     <title>The Busy Executive’s Data Base
           Guide</title>
     <type>business</type>
     <pub_id>1389</pub_id>
     <price>19.99</price>
     <advance>5000.00</advance>
     <total_sales>4095</total_sales>
  </row>

  <row>
     <title_id>BU1111</title_id>
     <title>Cooking with Computers:
              Surreptitious Balance Sheets</title>
     <type>business </type>
     <pub_id>1389</pib_id>
     <price>11.95</price>
     <advance>5000.00</advance>
     <total_sales>3876</total_sales>
  </row>

  <row>
     <title_id>BU2075</title_id>
     <title>You Can Combat Computer Stress!</title>
     <type>business </type>
```

```
   <pub_id>0736</pub_id>
   <price>2.99</price>
   <advance>10125.00</advance>
   <total_sales>18722</total_sales>
</row>

<row>
   <title_id>BU7832</title_id>
   <title>Straight Talk About Computers</title>
   <type>business </type>
   <pub_id>1389</pub_id>
   <price>19.99</price>
   <advance>5000.00</advance>
   <total_sales>4095</total_sales>
</row>

<row>
   <title_id>MC2222</title_id>
   <title>Silicon Valley Gastronomic Treats</title>
   <type>mod_cook</type>
   <pub_id>0877</pub_id>
   <price>19.99</price>
   <advance>0</advance>
   <total_sales>2032</total_sales>
</row>

<row>
   <title_id>MC3021</title_id>
   <title>The Gourmet Microwave</title>
   <type>mod_cook</type>
   <pub_id>0877</pub_id>


   <price>2.99</price>
   <advance>15000.00</advance>
   <total_sales>22246</total_sales>
</row>

<row>
   <title_id>MC3026</title_id>
   <title>The Psychology of Computer Cooking</title>
   <type>UNDECIDED</type>
   <pub_id>0877</pub_id>
</row>

<row>
   <title_id>PC1035</title_id>
   <title>But Is IT User Friendly?</title>
   <type>popular_comp</type>
   <pub_id>1389</pub_id>
   <price>22.99</price>
```

```
   <advance>7000.00</advance>
   <total_sales>8780</total_sales>
</row>

<row>
   <title_id>PC8888</title_id>
   <title>Secrets of Silicon Valley</title>
   <type>popular_comp</type>
   <pub_id>1389</pub_id>
   <price>20.00</price>
   <advance>8000.00</advance>
   <total_sales>4095</total_sales>
</row>

<row>
   <title_id>PC9999</title_id>
   <title>Net Etiquette</title>
   <type>popular_comp</type>
   <pub_id>1389</pub_id>
</row>

<row>
   <title_id>PS1372</title_id>
   <title>Computer Phobic and Non-Phobic
       Individuals: Behavior Variations</title>
   <type>psychology </type>
   <pub_id>0877</pub_id>
   <price>21.59</price>


   <advance>7000.00</advance>
   <total_sales>375</total_sales>
</row>

<row>
   <title_id>PS2091</title_id>
   <title>Is Anger the Enemy?</title>
   <type>psychology </type>
   <pub_id>0736</pub_id>
   <price>10.95</price>
   <advance>2275.00</advance>
   <total_sales>2045</total_sales>
</row>

<row>
   <title_id>PS2106</title_id>
   <title>Life Without Fear</title>
   <type>psychology </type>
   <pub_id>0736</pub_id>
   <price>7.99</price>
   <advance>6000.00</advance>
```

```
   <total_sales>111</total_sales>
</row>

<row>
   <title_id>PS3333</title_id>
   <title>Prolonged Data Deprivation:
         Four Case Studies</title>
   <type>psychology</type>
   <pub_id>0736</pub_id>
   <price>19.99</price>
   <advance>2000.00</advance>
   <total_sales>4072</total_sales>
</row>

<row>
   <title_id>PS7777</title_id>
   <title>Emotional Security:
       A New Algorithm</title>
   <type>psychology </type>
   <pub_id>0736</pub_id>
   <price>7.99</price>
   <advance>4000.00</advance>
   <total_sales>3336</total_sales>
</row>



<row>
   <title_id>TC3218</title_id>
   <title>Onions, Leeks, and Garlic:
      Cooking Secrets of the Mediterranean</title>
   <type>trad_cook </type>
   <pub_id>0877</pub_id>
   <price>20.95</price>
   <advance>7000.00</advance>
   <total_sales>375</total_sales>
</row>

<row>
   <title_id>TC4203</title_id>
   <title>Fifty Years in Buckingham
        Palace Kitchens</title>
   <type>trad_cook </type>
   <pub_id>0877</pub_id>
   <price>11.95</price>
   <advance>4000.00</advance>
   <total_sales>15096</total_sales>
</row>

<row>
   <title_id>TC7777</title_id>
```

```
    <title>Sushi, Anyone?</title>
    <type>trad_cook </type>
    <pub_id>0877</pub_id>
    <price>14.99</price>
    <advance>8000.00</advance>
    <total_sales>4095</total_sales>
  </row>

</titles>

(1 row affected)
```

付 録 B

XML サービスと外部ファイル・システム・アクセス

Adaptive Server の外部ファイル・システム・アクセス機能では、オペレーティング・システム・ファイルへの SQL テーブルとしてのアクセスを提供します。この付録では、ネイティブ XML プロセッサとファイル・システム・アクセス機能の使用方法について説明します。詳細については、『コンポーネント統合サービス・ユーザーズ・ガイド』を参照してください。

ファイル・システム・アクセス機能を使用する場合、Adaptive Server のコンポーネント統合サービス (CIS) 機能を使用して、外部ファイル・システムのディレクトリ・ツリー全体をマップするプロキシ・テーブルを作成します。その後で、プロキシ・テーブルのデータに対してネイティブ XML プロセッサの組み込み関数を使用して、外部ファイル・システムに格納されている XML ドキュメントを問い合わせます。

外部ディレクトリ再帰アクセスを使用すると、プロキシ・テーブルを親ディレクトリとそのすべての下位ファイルと下位ディレクトリにマップできます。

使用開始にあたって

この項では、外部ファイル・システム・アクセス機能を伴う XML サービスを設定する方法について説明します。

XML サービスと外部ファイル・システム・アクセスの有効化

- 次のように、sp_configure を使用して XML サービス、CIS、ファイル・アクセスを有効にします。

  ```
  sp_configure "emable xml", 1
  ```

- 設定パラメータ enable cis が 1 に設定されていることを確認します。

  ```
  sp_configure "enable cis",1
  ```

- 次のように、sp_configure を使用してファイル・アクセスを有効にします。

  ```
  sp_configure "enable file access", 1
  ```

外部ファイル・システムを使用した文字セット変換

一般に、外部ファイル・システム・テーブルの content カラムは image として処理されます。ただし、Unicode カラム (データ型が unichar、univarchar、unitext、または java.lang.String のカラム) に *content* カラムが割り当てられている場合は、特別な変換が実行されます。Unicode カラムに対するこのような *content* カラムの割り当ては、次のコンテキストで発生します。

- insert コマンドを使用して、*content* カラムを参照するサブクエリから Unicode カラムを挿入する場合。
- update コマンドを使用して、*content* カラムを参照する新しい値で Unicode カラムを更新する場合。
- convert 関数呼び出しによって、変換先の Unicode データ型と、*content* カラムである変換元の値を指定する場合。

content カラムを Unicode に割り当てるときは、次の規則に従います。

- 変換元ドキュメントにバイト順マーク (BOM: Byte Order Mark) が存在する場合、変換元ドキュメントを変換するには、BOM が UTF-8 または UTF-16 を示す必要があります。UCS-4 を示すときは、エラーが発生します。UCS-4 はサポートされていません。
- 変換元ドキュメントに ENCODING 句を含む XML ヘッダが存在するが、BOM が存在しない場合、変換元ドキュメントを変換するには、ENCODING 句でサーバの文字セットまたは UTF-8 を指定する必要があります。サーバのセットまたは UTF-8 以外の文字セットを ENCODING 句が指定するときは、エラーが発生します。
- 変換元ドキュメントに XML ヘッダおよび BOM が存在せず、ENCODING 句を含むヘッダが存在する場合、プロセッサは文字セットを UTF-8 として処理し、変換元ドキュメントを変換します。
- 変換中にエラーが発生しても、文は続行されます。

例

以下の例では、さまざまな XML 組み込み関数を使用して、外部ファイル・システムの XML ドキュメントを問い合わせる方法を示します。

XML ドキュメントの設定とプロキシ・テーブルの作成

以下の例では、作成した *bookstore.1.xml* と *bookstore.2.xml* というファイルに格納されている 2 つの XML ドキュメントを使用します。

```
cat bookstore.1.xml

<?xml version='1.0' standalone = 'no'?>
<!-- bookstore.1.xml example document--!>
<bookstore specialty='novel'>
<book style='autobiography'>
   <title>Seven Years in Trenton</title>
    <author>
        <first-name>Joe</first-name>
         <last-name>Bob</last-name>
         <award>Trenton Literary Review Honorable Mention</award>
        </author>
        <price>12</price>
        </book>
 </bookstore>

cat bookstore.2.xml

<?xml version='1.0' standalone = 'no'?>
<!-- bookstore.2.xml example document--!>
<bookstore specialty='novel'>
   <book style='compbook'>
      <title>Modern Database Management</title>
       <author>
          <first-name>Jeffrey</first-name>
          <last-name>Hoffer</last-name>
       </author>
       <price>112.00</price>
   </book>
</bookstore>
```

ファイル・システム・アクセスでこれらの XML ドキュメントを参照するには、create proxy table を使用します。

次のサンプル・コードは create proxy table の使用方法を示します。at 句のディレクトリ・パス名には、Adaptive Server でアクセスと検索の両方が可能なファイル・システム・ディレクトリを指定します。パス名の末尾に拡張子の ';R' (「再帰 (Recursion)」の意) を追加すると、CIS はそのパス名の下にあるすべてのディレクトリのファイル情報を抽出します。

```
create proxy_table xmlxfsTab external directory
at "/remote/nets3/bharat/xmldocs;R"
select filename from xmlxfsTab f

filename
-----------------------------------------
bookstore.1.xml
bookstore.2.xml
```

```
(2 rows affected)
```

重要なカラムは filename と content です。その他のカラムには、アクセス・パーミッション用のデータなどが入ります。filename カラムにはファイル名 (この例では XML ドキュメント・ファイル名) が入ります。content カラムにはそのファイルの実際のデータが入ります。content カラムのデータ型は image です。

例：XML ドキュメントから本のタイトルを抽出する

```
select filename, xmlextract("//book/title" , content)
from xmlxfsTab

filename
-------------------------------------------------
bookstore.1.xml
<title>Seven Years in Trenton</title>
bookstore.2.xml
<title>Modern Database Management</title>

(2 rows affected)
```

例：XML ドキュメントまたは XML クエリ結果を Adaptive Server テーブルにインポートする

完全な XML ドキュメントまたは XML クエリ結果を、ファイル・アクセス・ディレクトリ構造とデータベース・テーブルの間で、またはファイル・アクセス・ディレクトリ構造間で転送できます。完全な XML ドキュメントを参照するには、XPath のルート演算子 ("/") を指定した xmlextract 関数を使用します。

```
insert into xmldoctab select filename,xmlcol=xmlextract("/",content) into
from xmlxfsTab
------------------
(2 rows affected)
```

この例では、xmlxfsTab.content カラムのデータ型は image です。組み込み関数 xmlextract が返すデフォルト・データ型は text です。したがって、結果が image データ型で返されるように、xmlextract 呼び出しに returns image 句を指定します。

ヘッダを保持するには、xmlextract() の代わりに xmlvalidate() を使用します。

```
insert into xmldoctab select filename,xmlvalidate(content)
from xmlxfsTab
-------------
(2 rows affected)
```

次の例は、新しいサブディレクトリ *XmlDir* を作成します。

```
insert into xmlxfsTab(filename,content)
select filename = 'XmlDir/'+filename,
    xmlextract("/",xmlcol returns image) from xmldoctab
-----------
(2 rows affected)
```

このサンプル・コードでは、新しい *XmlDir* サブディレクトリにある XML ドキュメントを問い合わせます。

```
select filename, xmlextract("//book/title", content)
from xmlxfsTab
where filename like '%XmlDir%' and filetype = 'REG'

filename
---------------------------------
XmlDir/bookstore.1.xml
<title>Seven Years in Trenton</title>
XmlDir/bookstore.2.xml
<title>Modern Database Management</title>

(2 rows affected)
```

例：ファイル・システムに解析済み XML ドキュメントを格納する

次のように、外部ファイル・システムに格納された XML ドキュメントを解析し、Adaptive Server テーブルまたはファイル・アクセス・システムに解析結果を格納できます。

```
insert xmlxfsTab(filename, content)
select 'parsed'+t.filename,xmlparse(t.content) from xmlxfsTab
-----------
(2 rows affected)
```

次のサンプル・コードは、XFS ファイル・システムに格納された解析済みドキュメントを問い合わせます。

```
select filename, xmlextract("//book/title", content)
from xmlxfsTab
where filename like 'parsed%'and filetype = 'REG'
filename
-------------------------------------------
parsedbookstore.1.xml
<title>Seven Years in Trenton</title>
parsedbookstore.2.xml
<title>Modern Database Management</title>

(2 rows affected)
```

次のサンプル・コードは、組み込み関数 xmlrepresentation を使用して、解析済み XML であるファイル・アクセス・ドキュメントのみ (他の種類の外部ファイルではない) を問い合わせます。

```
select filename, xmlextract("//book/title", content)
from xmlxfsTab
where xmlrepresentation(content) = 0
filename
---------------------------------------
parsedbookstore.1.xml
<title>Seven Years in Trenton</title>
parsedbookstore.2.xml
<title>Modern Database Management</title>

(2 rows affected)
```

例：外部ファイル・アクセスを伴う 'xmlerror' オプションの機能

外部 (OS) ファイル・システムにはさまざまなデータ・フォーマットがあり、常に有効な XML ドキュメントが格納されているとは限りません。xmlextract 関数と xmltest 関数の xmlerror オプションを使用すると、無効な XML ドキュメントに対するエラー・アクションを指定できます。

たとえば、ファイル・アクセス・ディレクトリ構造に、*bookstore1.xml* ファイルと *bookstore.2.xml* ファイルとともに *picture.jpg* ファイルと *nonxmldoc.txt* ファイルがあるとします。

```
select filename from xmlxfsTab
filename
----------------------------------------
picture.jpg
bookstore.1.xml
bookstore.2.xml
nonxmldoc.txt

(4 rows affected)
```

次のサンプル・コードは、XML データと XML 以外のデータの両方に対する XML クエリを示します。

```
select filename, xmlextract("//book/title",content)
from xmlxfsTab
--------------
Msg 14702, Level 16, State 0:
Line 1:
XMLEXTRACT(): XML parser fatal error <<An exception occurred!
Type:TranscodingException,
Message:An invalid multi-byte source text sequence was
encountered>> at line 1, offset 1.
```

例：xmlextract に 'xmlerror=message' オプションを指定する

この例では、xmlextract 呼び出しに 'xmlerror= message' オプションを指定します。このオプションを指定すると、XML ドキュメントが有効な XML である場合には XML クエリ結果が返され、無効な XML である場合には XML エラー・メッセージ要素が返されます。

```
select filename, xmlextract("//book/title",content
     option 'xmlerror = message') from xmlxfsTab
filename
-------------------
picture.jpg
<xml_parse_error>An exception occurred!
Type:TranscodingException,
Message:An invalid multi-byte source text sequence was
encountered</xml_parse_error>

bookstore.1.xml
<title>Seven Years in Trenton</title>

bookstore.2.xml
<title>Modern Database Management</title>
nonxmldoc.txt
<xml_parse_error>Invalid document structure</xml_parse_error>

(4 rows affected)
```

例：'xmlerror=message' オプションを伴う XML ドキュメントと XML 以外のドキュメントの解析

このサンプル・コードは、xmlparse 呼び出しに 'xmlerror= message' オプションを指定します。このオプションを指定すると、XML ドキュメントが有効な XML である場合には解析済み XML が格納され、無効な XML である場合には解析済み XML エラー・メッセージ要素が格納されます。

```
insert xmlxfsTab(filename, content)
select 'ParsedDir/'+filename, xmlparse(content option
      'xmlerror = message')
from xmlxfsTab
--------------

(4 rows affected)
```

次のサンプル・コードは、解析済みデータに組み込み関数 xmlextract を適用し、例外メッセージを含む XML 以外のデータ・リストを取得します。

```
select filename, xmlextract('/xml_parse_error', content)
from xmlxfsTab
where '/xml_parse_error' xmltest content and filename like 'ParsedDir%'
----------------
Or with xmlrepresentation builtin
select filename, xmlextract('/xml_parse_error', content)
from xmlxfsTab
where xmlrepresentation(content) =  0
and '/xml_parse_error' xmltest content
filename
---------------------------------
ParsedDir/picture.jpg
<xml_parse_error>An exception occurred!
Type:TranscodingException,
Message:An invalid multi-byte source text sequence was
encountered</xml_parse_error>

ParsedDir/nonxmldoc.txt

<xml_parse_error>Invalid document structure
</xml_parse_error>

(2 rows affected)
```

例：XML 以外のドキュメントに 'xmlerror=null' オプションを使用する

次のサンプル・コードは、ファイル・アクセス・テーブルに 'xmlerror = null' オプションを指定します。

```
select filename, xmlextract("//book/title", content
     option 'xmlerror = null')
from xmlxfsTab
filename
----------------------------
picture.jpg
NULL
bookstore.1.xml
<title>Seven Years in Trenton</title>

bookstore.2.xml
<title>Modern Database Management</title>
nonxmldoc.txt
NULL

(4 rows affected)
```

次のサンプル・コードは、'xmlerror = null' オプションを使用して XML 以外のドキュメント名のリストを選択します。

```
select filename from xmlxfsTab
where '/' not xmltest content
     option 'xmlerror = null'
filename
----------------------------
picture.jpg
nonxmldoc.txt

(2 rows affected)
```

付 録　C

Java ベースの XQL プロセッサとネイティブ XML プロセッサ間のマイグレート

概要

Java ベースの XQL プロセッサとネイティブ XML プロセッサはどちらも問い合わせ言語を実装し、解析済み形式のドキュメントを返しますが、使用する関数とメソッドが異なります。

- ネイティブ XML プロセッサは XML 問い合わせ言語を実装しています。このプロセッサは組み込み関数 xmlparse を備えています。この関数は、組み込み関数 xmlextract と xmltest を使用した効率的な処理に適したドキュメントを解析済みの形式で返します。
- Java ベースの XQL プロセッサは、XQL 問い合わせ言語を実装する、以前のバージョンの機能です。このプロセッサは Java メソッド com.sybase.xml.xql.Xql.parse を備えています。このメソッドは、com.sybase.xml.xql.Xql.query メソッドを使用した処理に適した sybase.aseutils.SybXmlStream オブジェクトであるドキュメントを解析済みの形式で返します。

Java ベースの XQL プロセッサとネイティブ XML プロセッサの間でドキュメントをマイグレートする場合には、次のことに注意してください。

- テキスト形式のドキュメントは、Java ベースの XQL プロセッサとネイティブ XML プロセッサの両方で直接処理できます。
- com.sybase.xml.xql.Xql.parse によって生成された sybase.aseutils.SybXmlStream ドキュメントは、Java ベースの XQL プロセッサでのみ処理できます。組み込み関数 xmlextract または xmltest では処理できません。
- 組み込み関数 xmlparse によって生成された解析済みドキュメントは、組み込み関数 xmlextract と xmltest でのみ処理できます。Java ベースの XQL プロセッサでは処理できません。

ドキュメントとクエリのマイグレート

これ以降の項では、Java ベースの XQL プロセッサとネイティブ XML プロセッサ間でドキュメントとクエリをマイグレートする方法について説明します。

Java ベースの XQL プロセッサとネイティブ XML プロセッサ間でのドキュメントのマイグレート

Java ベースの XQL プロセッサからネイティブ XML プロセッサへドキュメントをマイグレートするには、次の 2 通りの方法があります。

- テキスト形式のドキュメントが利用可能であればそれを使用できます。
- 解析済み形式のドキュメントからテキスト形式のドキュメントを生成できます。

Java ベースの XQL プロセッサとネイティブ XML プロセッサ間でのテキスト・ドキュメントのマイグレート

次のようなテーブルがあるとします。このテーブルの xmlsource カラムにはテキスト形式のドキュメントが格納されています。

```
create table xmltab (xmlsource text, xmlindexed image)
```

ネイティブ XML プロセッサで組み込み関数 xmlextract と xmltest を使用してドキュメントを処理する場合は、次のようにテーブルを更新できます。

```
update xmltab
set xmlindexed = xmlparse(xmlsource)
```

Java ベースの XQL プロセッサで com.sybase.xml.xql.Xql.query メソッドを使用してドキュメントを処理する場合は、次のようにテーブルを更新できます。

```
update xmltab
set xmlindexed
    = com.sybase.xml.xql.Xql.parse(xmlsource)
```

再生成したコピーからのドキュメントのマイグレート

ネイティブ XML プロセッサの組み込み関数 xmlparse または Java ベースの XQL プロセッサの com.sybase.xml.xql.Xql.parse メソッドを使用して、解析済み形式のドキュメントのみを格納したとします。たとえば、これらのドキュメントが次のようにテーブルに格納されているとします。

```
create table xmltab (xmlindexed image)
```

これらのドキュメントのテキストを再生成する場合、テーブルを変更してテキスト・カラムを追加できます。

```
alter table  xmltab add xmlsource text null
```

Java ベースの XQL プロセッサからのテキスト・ドキュメントの再生成

この項では、Java ベースの XQL プロセッサ用に生成された形式からテキスト形式のドキュメントを再生成する方法について説明します。

xmlindexed カラムに、com.sybase.xmlxql.Xql.parse で生成された sybase.aseutils.SybXmlStream データがある場合、次の SQL 文を使用して新しい xmlsource カラムにテキスト形式のドキュメントを再生成できます。

```
update xmltab
set xmlsource
   = xmlextract("/xql_result/*",
      com.sybase.xml.xql.Xql.query("/",xmlindexed) )
```

この文は、テキスト形式のドキュメントを次の 2 つの手順で生成します。

1　"/" クエリが指定された com.sybase.xml.xql.Xql.query 呼び出しが、XML タグ *<xql_result>...</xql_result>* で囲まれたテキスト形式のドキュメントを生成します。

2　"/xql_result/*" クエリが指定された xmlextract 呼び出しが、*<xql_result>...</xql_result>* タグを削除し、元のドキュメントのテキスト形式を返します。

これで、ネイティブ XML プロセッサで組み込み関数 xmlextract と xmltest を使用して xmlsource カラムを直接処理できます。また、次のようにして、ネイティブ XML プロセッサ用の xmlindexed カラムを更新できます。

```
update xmltab
set xmlindexed = xmlparse(xmlsource)
```

xmlsource カラムを追加しない場合は、次の SQL 文のようにこれらの手順を組み合わせることができます。

```
update xmltab
set xmlindexed
  = xmlparse(xmlextract("/xql_result/*",
      com.sybase.xml.xql.Xql.query("/",xmlindexed) ) )
```

この update 文の実行前、xmlindexed カラムには、com.sybase.xml.xql.Xql.parse メソッドで生成された sybase.aseutiles.SybXmlStream 形式のドキュメントが含まれています。update 文の実行後、このカラムには xmlextract メソッドと xmlparse メソッドでの処理に適した解析済み形式のドキュメントが含まれます。

ネイティブ XML プロセッサからのテキスト・ドキュメントの再生成

この項では、ネイティブ XML プロセッサ用に生成された形式からテキスト形式のドキュメントを再生成する方法について説明します。

xmlindexed カラムに、xmlparse 関数で生成されたデータがある場合、次の SQL 文を使用して新しい xmlsource カラムにテキスト形式のドキュメントを再生成できます。

```
update xmltab
set xmlsource  = xmlextract("/",  xmlindexed)
```

これで、次のことが可能になります。

- Java ベースの XQL プロセッサで com.sybase.xml.xql.Xql.query を使用して xmlsource カラムを直接処理する。
- 次の文を使用して、Java ベースの XQL プロセッサでの処理に適した解析済み形式のドキュメントを含む xmlindexed カラムに対して update を実行する。

  ```
  update xmltab
  set xmlindexed
    = com.sybase.xml.xql.Xql.parse(xmlsource)
  ```

xmlsource カラムを追加しない場合は、次の SQL 文のようにこれらの手順を組み合わせることができます。

```
update xmltab
set xmlindexed
    = com.sybase.xml.xql.Xql.parse
      (xmlextract("/", xmlindexed))
```

この update 文の実行前、xmlindexed カラムには組み込み関数 xmlparse で生成された解析済み形式のドキュメントが含まれています。update 文の実行後、このカラムには com.sybase.xml.xql.Xql.parse で生成された、com.sybase.xml.xql.Xql.query での処理に適した解析済み形式のドキュメントが含まれます。

ネイティブ XML プロセッサと Java ベースの XQL プロセッサ間でのクエリのマイグレート

Java ベースの XQL プロセッサで実装される XQL 言語と、ネイティブ XML プロセッサで実装される XML 問い合わせ言語は、どちらも XPath 言語に基づいています。これらの言語は、主に次の 2 つの点が異なります。

- XML 問い合わせ言語のサブスクリプトは "1" で始まり、XQL 言語のサブスクリプトは "0" で始まります。
- Java ベースの XQL プロセッサは "*<xql_result>...</xql_result>*" タグで囲まれた結果を返し、ネイティブ XML プロセッサはこのような結果を返しません。

付 録 D

xmltable() のサンプル・アプリケーション

サンプル・テーブル

この項では、xmltable() のアプリケーションを示す、サンプル XML ドキュメント depts.xml を示します。

```
<sample>
<depts>
   <dept>
        <dept_id>D123</dept_id>
        <dept_name>Main</dept_name>
   <emps>
    <emp>
        <emp_id>E123</emp_id>
        <emp_name>Alex Allen</emp_name>
        <salary>912.34</salary>
        <phones>
           <phone><phone_no>510.555.1987</phone_no></phone>
           <phone><phone_no>510.555.1867</phone_no></phone>
        </phones>
    </emp>
   <emp>
        <emp_id>E234</emp_id>
        <emp_name>Bruce Baker</emp_name>
        <salary>923.45</salary>
        <phones>
        <phone><phone_no>230.555.2333</phone_no></phone>
          </phones>
  </emp>
   <emp>
        <emp_id>E345</emp_id>
        <emp_name>Carl Curtis</emp_name>
        <salary>934.56</salary>
        <phones>
          <phone><phone_no>408.555.3123</phone_no></phone>
          <phone><phone_no>415.555.3987</phone_no></phone>
         <phone><phone_no>650.555.3777</phone_no></phone>
        </phones>
   </emp>
  </emps>
<emps_summary>
<salary_summary>
        <max_salary>934.56</max_salary>
        <total_salary>2770.35</total_salary>
```

```
</salary_summary>
</emps_summary>
<projects>
<project>
   <project_id>PABC</project_id>
   <budget>598.65</budget>
</project>
<project>
   <project_id>PBDC</project_id>
   <budget>587.65</budget>
</project>
<project>
   <project_id>PCDE</project_id>
   <budget>576.54</budget>
</project>

  </projects>
  <projects_summary>
 <budget_summary>
   <max_budget>598.76</max_budget>
   <total_budget>1762.95</total_budget>
 </budget_summary>
  </projects_summary>
</dept>
<dept>
        <dept_id>D234</dept_id>
        <dept_name>Auxiliary</dept_name>
   <emps>
    <emp>
        <emp_id>E345</emp_id>
        <emp_name>Don Davis</emp_name>
        <salary>945.67</salary>
        <phones>
     <phone><phone_no>650.555.5001</phone_no></phone>
         </phones>
     <emp>
        <emp_id>E345</emp_id>
        <emp_name>Earl Evans</emp_name>
        <phones>
     <phone><phone_no>650.555.5001</phone_no></phone>
         </phones>

    </emp>
  </emps>
<emps_summary>
<salary_summary>
        <max_salary>945.67</max_salary>
        <total_salary>945.67</total_salary>
</salary_summary>

</emps_summary>
```

```
<projects>
 <project>

 <project>
   <project_id>PDEF</project_id>
< /project>
 <project>
   <project_id>PEFG</project_id>
   <budget>554.32</budget>
</project>
  </project>
  </projects>
  <projects_summary>
 <budget_summary>
   <max_budget>554.32</max_budget>
   <total_budget>554.32</total_budget>
 </budget_summary>
  </projects_summary>
</dept>
</dept>
<dept>
        <dept_id>D345</dept_id>
        <dept_name>Repair</dept_name>
   <emps>
    <emp>
        <emp_id>E678</emp_id>
        <emp_name>Fred Frank</emp_name>
        <salary>967.89</salary>
        <phones>
           <phone><phone_no>408.555.6111</phone_no></phone>
         </phones>
   </emp
   <emp>>
        <emp_id>E789</emp_id>
        <emp_name>George Gordon</emp_name>

        <salary>978.90</salary>
        <phones>
     <phone><phone_no>510.555.7654</phone_no></phone>
         </phones>
   </emp>
   <emp>
        <emp_id>E901</emp_id>
        <emp_name>Hank Hartley</emp_name>
        <salary>990.12</salary>
        <phones¥>
   </emp
   <emp>
        <emp_id>E678</emp_id>
        <emp_name>Isaak Idle</emp_name>
```

```
        <salary>990.12</salary>
        <phones>
         <phone><phone_no>925.555.9991</phone_no></phone>
         <phone><phone_no>650.555.9992</phone_no></phone>
         <phone><phone_no>415.555.9993</phone_no></phone>
         </phones>
   </emp>
        <emps>
        <emps_summary>
   <salary_summary>
        <max_salary>990.12</max_salary>
        <total_salary>2936.91</total_salary>
  </salary_summary>
  </emps_summary>
  <projects>
    <project>
      <project_id>PFGH</project_id>
      <budget>543.21</budget>
   </project>
   <project>
    <project_id>PGHI</project_id>
   </project>
   <project>
     <project_id>PHIJ</project_id>
     <budget>521.09</budget>
   </project>
   </project>
 </projects>
  <projects_summary>
 <budget_summary>
   <max_budget>543.21</max_budget>
   <total_budget>1064.30</total_budget>


 </budget_summary>
  </projects_summary>
</dept>
</depts>
</sample>
```

depts ドキュメントの使用

depts ドキュメントは、「付録 A sample_docs サンプル・テーブル」の sample_docs テーブルの新しいローに格納されます。サンプルでこのドキュメントを参照するには、以下のように宣言します。

```
declare @dept_doc xml
select @dept_doc from sample_docs where name_doc='depts'
```

depts ドキュメントの構造

depts ドキュメントの構造は以下のとおりです。

```
<depts>
   <dept>
      <emps> - repeats under <depts>
       <emp> - one for each <dept>
          <emp_id> - one for each <emp>
          <emp_name> - one for each <emp>
          <phones> - one for each <emp>
            <phone> - repeats under <phones>
               <phone_no> - one for each <phone>
      <projects> - one for each <dept>
       <project> - repeats for each under <projects>
           <project_id> - one for each <project>
           <dept_id> - one for each <project>
```

depts ドキュメントからの SQL テーブルの作成

depts ドキュメントからのデータの正規化

このデータを SQL テーブルに正規化できます。次に例を示します。

- depts － <*dept_id*> および <*dept_name*> を含む各 <*dept*> 要素のロー
- emps － <*emp_id*>、<*emp_name*>、およびそれらが含まれる <*dept_id*> 要素を含む各 <*emp*> 要素のロー
- emp_phones － これらのすべての要素が含まれる <*phone_no*> および <*dept_id*> 要素を含む各 <*phone*> のロー
- projects － <*project_id*> および <*budget*> 要素と、それらが含まれる <*dept_id*> 要素を含む各 <*project*> のロー

select を使用したテーブルの生成

この項で生成されたすべてのテーブル (depts テーブルを除く) には、XPath の上位の表記を使用して以下の内容を参照するカラム・パターンが存在します。

- リーフ要素。<projects> の下の <*project*> や <*emp*> の下の <*salary*> などです。
- テーブルを <*emp*> として定義する要素を含む要素。たとえば、<*emp_id*> を含みます。

この表記は、ネストされたデータを「フラット化」します。XML データのフラット化の詳細については、「第 7 章 xmltable()」を参照してください。

emps テーブル

この select 文では、dept_id カラムのカラム・パターンは、現在の <*emp*> が含まれる <*dept*> 内の <*dept_id*> 要素を参照します。

```
declare @ dept_doc xml
select @dept_doc = doc from sample_docs where name_doc = 'depts'
select * into emps from xmltable('//emp' passing @dept_doc
        columns emp_id char(4),
        emp_name varchar(50),
        salary money,
        dept_id char(4) pattern '../../dept_id') as dept_extract
select * from emps
--------------------
emp_id              emp_name               salary         dept_id
---------           -------------------    --------       --------
E123                Alex Allen             912.34         D123
E234                Bruce Baker            923.45         D123
E345                Carl Curtis            934.56         D123
E456                Don Davis              945.67         D234
E567                Earl Evans             956.78         D234
E678                Fred Frank             967.89         D345
E789                George Gordon          978.90         D345
E890                Hank Hartley           NULL           D345
E901                Isaak Idle             990.12         D345
```

phones テーブル

phones テーブルでは、emp_id カラムのカラム・パターンは、現在の <*phone*> 要素が含まれる <*emp*> 内の <*emp_id*> 要素を参照します。

```
declare @ dept_doc xml
select @ dept_doc
    = doc from sample_docs where name_doc='depts'
select * into phones
    from xmltable('//phone' passing @ dept_doc
        columns emp_id char(4),'../../emp_id'
            phone_no varchar(50)) as dept_extract
```

```
select * from phones
---------------
emp_id                phone_no
--------               ------------
E123                   510.555.1987
E123                   510.555.1876
E234                   203.555.2333
E345                   408.555.3123
E345                   415.555.3987
E345                   650.555.3777
E567                   650.555.5001
E678                   408.555.6111
E678                   408.555.6222
E789                   510.555.7654
E901                   925.555.9991
E901                   650.555.9992
E901                   415.555.9993
```

projects テーブル

projects テーブルでは、dept_id カラムのカラム・パターンは、現在の <*project*> が含まれる <*dept*> 内の <*dept_id*> 要素を参照します。

```
declare @ dept_doc xml
select @ dept_doc
    = doc from sample_docs where name_doc='depts'
select * into projects
    from xmltable('//project' passing @ dept_doc
        columns project_id char(4),
        budget money,
            dept_id char(4)pattern '../../dept_id')
            as dept_extract
select * from projects
---------------
project_id      budget         dept_id
----------      -------        --------
PABC            598.76         D123
PBCD            587.65         D123
PCDE            576.54         D123
PDEF            565.43         D234
PEFG            554.32         D234
PFGH            543.21         D345
PGHI            NULL           D345
PHIJ            521.09         D345
```

***depts* テーブル**

```
declare @ dept_doc xml
select @ dept_doc
    = doc from sample_docs where name_doc='depts'
select * into depts
    from xmltable('//dept' passing @ dept_doc
        columns dept_id char(4),
            dept_name char(4)) as dept_extract
select * from depts
---------------
dept_id                 dept_name
--------                ------------
D123                    Main
D234                    Auxiliary
D345                    Repair
```

索引

記号

€ ユーロ　86
@@error
　グローバル変数　12
　最後のエラーのエラー番号　12

数字

16 ビット値、サロゲート・ペア　86

A

ample_docs テーブル
　titles テーブル、XML 表現　119
any/all 限定述語サブクエリ、for xml サブクエリを使用できない　60
at 句コマンド　127

B

base64、SQLX オプション　69
BCP、データ転送　86
binary
　SQLX オプション　69
　値　84
　オプション　69
　データ型　69, 84
binary SQLX オプション　67
binary オプション　37
bookstore、サンプル・ドキュメント　116

C

char データ型　2, 83
CIS (コンポーネント統合サービス)　125
columnstyle SQLX オプション　67
columnstyle オプション　37
columnstyle、SQLX オプション　69, 78
concat 関数、XPath 文字列関数　46
concat 文字列関数　46, 51
concat、XPath 文字列関数　46
concat、関数　51
content カラム、Unicode、割り当て規則　126
create proxy table コマンド　127
create view コマンド、for xml サブクエリを使用できない　60
CTLIB、データ転送　86

D

declare cursor コマンド、for xml サブクエリを使用できない　60
default xml sort order、sp_configure　94
default xml sort order、変更　95
DTD　7
　ATTLIST　7
　ELEMENT　7
　#IMPLIED　7
　#PCDATA DTD 要素　7
　valid なXML ドキュメント　8
　アスタリスク (*)　7
　一部のドキュメントで必須ではない　8
　埋め込み　8
　疑問符 (?)　7
　プラス記号 (+)　7
DTD の埋め込み、XML内　8
DTDS およびスキーマ、xmlvalidate　30
dtdvalidate オプション　37
dtdvalidate、検証オプション　27

E

EFS
　例　126
EFS アクセス　125
encoding オプション、xmlparse　93

entitize
SQLX オプション 70
オプション 37, 67
exists/not exists 限定述語サブクエリ、for xml サブクエリを使用できない 60
exists/not exists、限定述語サブクエリ 60

F

for xml
SQLX-XML フォーマット 77
句 67, 71
句、構文と例 55
説明 56
データ・マッピング 77
for xml all 61
for xml header オプション 89
for xml schema 61
for xml select コマンド、for xml サブクエリを使用できない 60
for xml 句
オプション 58
拡張機能 61
拡張機能の説明 61
拡張機能の例 62
限定述語サブクエリとして使用できない 60
構文 55
コマンドで使用できない 60
使用、isql 67
相関サブクエリにすることはできない 60
ネストされたスカラ・サブクエリで使用できない 60
例 58, 89
例外 58
for xml 句拡張機能の例外 61
for xml 句による非 ASCII データの処理 87
for xml サブクエリ 59
構文 59
説明 59
例 60
例外 60
for xml、Unicode 87
for_xml 句
非 ASCII データの生成 85
format SQLX オプション 67
format {yes | no}
SQLX オプション 71
format オプション 37
forxml 関数 71

H

header
SQLX オプション 67, 71
header オプション 37
hex、SQLX オプション 69
HTML
DTD 要素 7
HTML 表示、Order データ 5
Order コード・サンプル 5, 6
制限事項 6
矛盾した要素タグ 6
矛盾、要素をカッコで囲む 6

I

I18N
非 ASCII データ 85
例 89
image カラム 2
image データ型 69, 84
image_doc、sample_docs テーブルのカラム 115
image、データ型 95
in/not in 限定述語サブクエリ、for xml サブクエリを使用できない 60
incremental SQLX オプション 67, 71
info、XML コード・サンプル 4
isql、使用、for xml 句 67
ISQL、データ転送 86

J

Java
クライアント文字セットのサンプル・ディレクトリ 87
Java プロセッサ 1
Java ベースのプロセッサ
テキスト・ドキュメントの再生成 137
ネイティブ XML プロセッサとのマイグレート 136
ネイティブ XML プロセッサへのマイグレート 135, 139
java.lang.String データ型 2
java.lang.String、データ型 95

JDBC コード
プロパティ disable_unichar_sending を使用して Unicode データを送信する、disable_unichar_sending プロパティ 87

M

multipleentitize オプション 67
multiplentities オプション 72

N

NCR オプション
xmlvalidate によりサポート 95
デフォルトのデータ型 95
ncr オプション 37, 68, 72
ncr オプション、デフォルト 88
NCR (数値文字表現)、Unicode 86
ncr、サポートされるオプション文字列 12
nonamespaceschemalocation オプション 37
normalize-space 関数、XPath 文字列関数 46
normalize-space 文字列関数 46, 50
normalize-space、XPath 文字列関数 46
normaliz-space、関数 50
nullstyle
SQLX オプション 68, 72
nullstyle オプション 37, 72

O

option string の値 36
option_string の値、表 37
option_strings
一般的なフォーマット 36
構文 36
説明 36
説明と例 36
パラメータ 9
Order サンプル
HTML 6
XML コード 3
Order の DTD、サンプル・コード 7

P

prefix SQLX オプション 68, 82
prefix オプション 37
publishers、pubs2 データベース・テーブル 116

R

root SQLX オプション 68, 73
root オプション 37
rowname オプション 37
rowname、SQLX オプション 68, 74, 82

S

sample_doc テーブル
ロー 116
sample_docs テーブル
publishers テーブルと titles テーブル 117
publishers テーブル、XML 表現 118
カラム 115
構造 115
テーブル・スクリプト (publishers) 118
sample_docs テーブルのカラム
image_doc 115
name_doc sample_docs テーブルのカラム 115
text_doc 115
schemaloc SQLX オプション 68, 74
schemalocation オプション 37
schemavalidate オプション 37
select into コマンド、for xml サブクエリを使用できない 60
select コマンド 71
select 文
ncr オプション 88
SGML、汎用マークアップ言語 2
sp_configure
XML サービスと外部ファイル・システム・アクセスの有効化 127
sp_configure オプション default xml sort order 94
sp_configure、XML サービスと外部ファイル・システム・アクセスの有効化 125
SQL 拡張機能、クエリ関数 9
SQL 名、例 82
sql_name、SQLX オプション 74

SQLX
  オプション、定義 69
  オプション、表 67
  データ 77
  データ・マッピング 77
SQLX オプション 67
  base64 69
  binary 67, 69
  columnstyle 67, 69, 78
  entitize 70
  format 67
  format={yes | no} 71
  header 67, 71
  hex 69
  incremental 67, 71
  nullstyle 68, 72
  prefix 68, 82
  root 68, 73
  rowname 68, 74, 82
  schemaloc 68, 74
  sql_name 74
  statement 68, 75
  tablename 68, 75, 82
  tablename=sqlname 75
  targetns 68, 76
  targetns=url 76
SQLX-XML
  フォーマット 67
SQLX-XML フォーマット 77
statement SQLX オプション 68, 75
statement オプション 37
string、データ型 95

T

tablename オプション 37
tablename=sqlname、SQLX オプション 75
tablename、SQLX オプション 68, 75, 82
targetns SQLX オプション 68, 76
targetns オプション 37
targetns=url、SQLX オプション 76
text doc sample_docs テーブルのカラム 115
text データ型 2, 83
titles、pubs2 データベース・テーブル 116
tolower 関数、XPath 文字列関数 46
tolower 文字列関数 46, 50
tolower、toupper、関数 50
tolower、XPath 文字列関数 46
toupper 関数、XPath 文字列関数 46
toupper 文字列関数 46
toupper 文字列関数 50
toupper、XPath 文字列関数 46

U

unichar データ型 2
unichar、データ型 95
Unicode 93
  for xml 87
  select 文における ncr オプション 88
  xmlextract 93
  xmlparse 92
  xmlvalidate 95
  オプション文字列 88
  カラム 86
  サロゲート・ペア 86
  数値文字表現 (NCR) 86
  データ型 86
  データ型の種類 86
Unicode カラム
  java.lang.String 126
  unichar 126
  unitext 126
  univarchar 126
Unicode データ型 unichar、univarchar、unitext、java.lang.String 95
Unicode データ型、Unicode データ型 86
Unicode の例、テーブル例 89
Unicode、XML 85
Unicode、非 ASCII データ 85
Unicode、非 ASCII データ、I18N 拡張機能 85
union とカッコ 53
unitext データ型 2
unitext、データ型 95
univarchar データ型 2
univarchar、データ型 95
Universal Resource Indicator (URI) 39
URI (Universal Resource Indicator) 39
URI (Universal Resource Indicator)、サポートされている 39
URI、サポートされていない 39
UTF-16 (Unicode Transformation Format、2 バイト) データ型 86
UTF-8 (Unicode Transformation Format、最大 4 バイト) データ型 86
UTF8、デフォルトの文字セット 4

V

valid なXML ドキュメント　8
varbinary データ型　84
varchar univarchar column　90
varchar データ型　2, 83

X

XFS
　xmlerror=message の指定　131
XML
　DTD サンプル・コード、埋め込み　8
　DTD サンプル・コード、外部参照　8
　DTD 要素、制限　7
　DTD、一部のドキュメントで必須ではない　8
　DTD、指示　7
　HTML との比較　2
　publishers テーブルの表現　118
　SGML のサブセット　2
　titles テーブルの表現　119
　値、マッピング　83
　アプリケーション固有のドキュメント・タイプ　2
　解析済み　2
　カスタム・タグ　2
　クエリ関数　9
　厳密な句構造　2
　サンプル・ドキュメント　3
　スキーマ宣言　40
　宣言、文字セットの指定　4
　データ交換、適合　2
　問い合わせ言語　39
　名前、マッピング　80
　比較、SGML と HTML　2
　マッピング　67
　マッピング関数　55
　読み込み、HTML ブラウザとプロセッサ　2
xml
　サンプル・ドキュメント　116
xml all オプション　72
XML EFS アクセス　125
XML 関数　50
XML サービス
　sp_configure の使用　125
　XPath 文字列関数　46
　外部ファイル・システム・アクセス　125
　外部ファイル・システム・アクセス、sp_configure の使用　127
　カッコで囲んだ式　51
XML サービスにおけるソート順　94
XML 問い合わせ言語、XPath のサブセット　43
XML ドキュメント
　ASE テーブルへのインポート、サンプル・コード　128
　DTD サンプル・コード　7
　格納、Adaptive Server　2
　検索、Web 上　2
　厳密な句構造　2
　サブディレクトリからのクエリ、サンプル・コード　129
　サンプル・コード、Order　3
　生成、Adaptive Server　2
　タグ　2
　正しい形式　4
　正しいドキュメント、DTD の使用　8
　ネームスペースのサポート、XML ドキュメント　40
　ネストされたマークアップ・タグ　4
　パーツ　4
　フォーマット指示、なし　4
　プロキシ・テーブルの作成、サンプル・コード　127
　本のタイトルの抽出、サンプル・コード　128
　文字データとしての XML ドキュメント　4
　例、info　4
XML ドキュメントの例　12
XML ドキュメント、例　12
XML の格納
　image カラム内の解析済み　2
　データ型としての Java クラス　2
XML のクエリ
　xmltest と xmlextract　1
XML の細分化、xmltest と xmlextract　1
XML の生成、for xml を使用した　1
XML を格納するデータ型　1
xmlerror オプション　37
xmlerror、サポートされるオプション文字列　12
xmlexparse
　オプション　22

xmlexrepresentation
  説明 24
xmlextract 85, 93, 94
  ncr オプション 93
  XML の抽出 1
  XQuery 言語サブセットのサポート 40
  関数の説明 10
  組み込みクエリ関数 9
  構文 10
  説明 10
  ソート順 94
  ネームスペース・プレフィクス 40
  例 12
  例外 12
xmlextract コマンド 1
xmlextract、クエリ関数 9
xmlparse 92, 94
  Unicode 92
  関数の説明 21
  組み込み関数 9
  構文 21
  説明 21
  ソート順 93
  非 ASCII データの格納 85
  ユーザによるオプションの修正 95
  例 22
  例外 22
xmlparse コマンド 1
xmlparse における非 ASCII データ 92
xmlparse、encoding オプション 93
xmlrepresentation
  関数の説明 24
  構文 24
  例 24
xmlrepresentation、クエリ関数、解析済み image カラムの判断 9
xmltable
  ordinality カラムの例 104
  関数 97, 98
  参照 114
  使用状況 111
  説明、構文 98
  ドキュメントのテーブルの処理の例 108
  例 98
xmltable()、derived table の構文 97
xmltest 85
  XML のクエリ 1
  オプション 18
  関数 17
  関数述部 17
  クエリ関数 9
  構文と説明 17
  コマンド 1
  説明 17
  例 18
xmltext
  SQL 述部、ブール値の結果を返す 9
  ネームスペース・プレフィクス 40
xmlvalid オプション 37
xmlvalidate
  オプション 27
  オプションの説明 27
  クエリ関数 9
  構文 26
  コマンド 26
  説明 26
  例 30
  例外 30
xmlvalidate コマンド 1
xmlvalidate、NCR オプションをサポート 95
xmlvalidate、Unicode をサポート 95
xmlvalidate、コマンド 26
XPath
  一般的なガイドライン 47
  演算子と関数 44
  カッコで囲んだ式 51
  基本演算子、表 45
  言語 1
  言語サブセット 43
  構文とトークン 43
  サポートされるトークン 44
  集合演算子、表 46
  比較演算子 46
  比較演算子、表 46
  例 47
XPath 1.0 39
XPath の基本演算子 text() 45
XPath の比較演算子 46
XPath 文字列関数 46
  例 47
XQL プロセッサ、ドキュメントのマイグレート 136

XQuery 言語、xmlextract と xmltest によるサポート　40
xscdecl オプション　37
xsidecl オプション　68
xsidecl={yes | no} オプション　76
XSL、拡張スタイル言語　2

あ

値
　binary　84
　数値　83
　文字　83

う

埋め込み DTD　8

え

演算子
　XPath における比較　46
エンティティ
　定義済み、XML 言語　40
　定義済み、XML 問い合わせ言語　41

お

オプション
　binary　37
　columnstyle　37
　dtdvalidate　37
　entitize　37, 67
　for xml 句　58
　format　37
　header　37
　multipleentitize　67
　ncr　37, 68
　nonamespaceschemalocation　37
　nullstyle　37
　prefix　37
　root　37
　rowname　37
　schemalocation　37
　schemavalidate　37
　SQLX　67
　SQLX、定義　69
　statement　37
　tablename　37
　targetns　37
　xmlerror　37
　xmlvalid　37
　xsidecl　37, 68
オプション xsidecl={yes | no}　76
オプション ヘッダ　89
オプション構文、xmlvalidate　27
オプション値、クエリ関数　36, 41
オプション文字列
　for xml　88
　ncr の仕様　88
　Unicode　88
　引用符付き識別子　88
　サポートされる　12
　単純な識別子　88
オプション、for xml all　72
オプション、検証　27

か

解析済み XML　2
ガイドライン、XPath 関数　47
外部ディレクトリ再帰アクセス、プロキシ・テーブルのマップ　125
外部ファイル・システム・アクセス、XML 内　125
外部ファイル・システム、Unicode カラム　126
外部ファイル・システム、文字セット変換　126
拡張機能
　for xml 句　61
　for xml 句、説明　61
　for xml 句、例　62
　for xml 句、例外　61
拡張スタイル言語。「XSL」参照
カッコで囲んだ式
　union　53
　サブスクリプト　51
空の要素　42
カラム
　image、解析済み XML　2
関数
　concat　46, 51
　forxml　71
　normalize-space　50
　tolower、toupper　50
　xmlextract　10
　xmlparse　21
　xmlrepresentation　24

xmltable 97, 98
XPath 44
マッピング 55
関数述部 xmltest 17
関数、XML、ドキュメント 50

き

規則、Unicode content カラム 126
基本演算子、XPath、サポート 45

く

クエリ
Java ベースのプロセッサとネイティブ・プロセッサ間のマイグレート 138
マイグレート 135
クエリ関数
XML 9
xmlextract 9
xmlrepresentation 9
xmltest 9
xmlvalidate 9
構文と例 10
テーブル 9
クエリ関数、SQL 拡張機能 9
クエリ関数、オプション値 36, 41
クライアントとサーバ間のデータ
Java の使用 86
転送 86
クライアントとサーバ間のデータ転送 86
クライアントとサーバ間のデータ転送、CTLIB、ISQL、BCP の使用 86
クライアント文字セットとサーバ文字セット、違い 87
クライアント、文字セット 87
グローバル変数
@@error 12

け

言語
XML と XML 問い合わせ 39
XPath 39
XQL 39
検索、Web 上に格納された XML ドキュメント 2
検証オプション 27
dtdvalidate 27

こ

構文
for xml 句 55
for xml 句、説明 55
option_strings 36
xmlparse 21
xmltest 17
XPath トークン 43
サブクエリ、for xml 59
例、xmlextract 10
構文、xmlrepresentation 24
コード化、文字。「文字セット」参照
コード・サンプル
HTML、Order の例 5, 6
XML、Info の例 4
XML、項目の例 5
コマンド
at 句 127
create proxy table 127
select 71
xmlextract 1
xmlparse xmltest 1
xmltest 1
xmlvalidate 1, 26
コマンド create proxy table 127
コンポーネント統合サービス (CIS) 125

さ

サーバ文字セット、クライアントとの違い 87
再生成
テキスト・ドキュメント、Java ベースのプロセッサ 137
テキスト・ドキュメント、ネイティブ XML プロセッサ 138
サブクエリ、for xml 59
サブスクリプト、カッコで囲んだ式 51
サロゲート・ペア、16 ビット値 86
サロゲート・ペア、Unicode 86
参照、XML の外部 DTD 8

サンプル・コード
DTD、Order 例 7
HTML、Order サンプル 6
XML、Info の例 4
XML、Order 例の DTD 7
サンプル・テーブル
sample_docs 115
Unicode の例 89
サンプル・テーブル、sample_docs 115

し

式、XPath におけるカッコで囲んだ 51

す

数値
値 83
データ型 83
数値文字表現 (NCR)、Unicode 86
スキーマのサポート 40
スペース、保持 42
スペース、保持された 42

せ

制約、ドキュメントのマイグレート 135
説明
サブクエリ、for xml 59

そ

相関サブクエリ、for xml 句にすることはできない 60
相対関数呼び出し 47
ソート順
xmlextract 94
xmlparse 93
属性、要素タグに埋め込まれた属性 4

た

タグ
HTML、カッコで囲む際の矛盾 6
HTML、パラグラフ 6
XML でのタグのカスタマイズ 2
厳密にネストされた XML 4
ユーザ作成 4
正しい形式の XML ドキュメント 4

ち

抽出テーブルの構文 98
抽出テーブルの構文、xmltable 97
重複したカラム名、マッピング 77

て

定義済みエンティティ 40, 41
データ型
binary 69, 84
char 2, 83
image 2, 69, 84
text 2, 83
Unicode の種類 86
varbinary 69, 84
varchar 83
数値 83
データ型による XML の格納、xmlparse、xmlvalidate 1
データ型、Unicode
char 86
text 86
unitext 86
univarchar 86
varchar 86
テーブル
option_string の値 37
publishers、XML 表現 118
SQLX オプション 67
titles、XML 表現 119
XPath の基本演算子 45
XPath の集合演算子 46
XPath の比較演算子 46
テキスト・データ、XML 4
デフォルトの ncr オプション 88

と

トークン
XPath、サポート 44
ドキュメント
Java プロセッサとネイティブ・プロセッサ間の
マイグレート 135
クエリ、マイグレート 135
ドキュメント・タイプ定義。「DTD」参照

な

名前のないカラム、マッピング 77

ね

ネイティブ XML プロセッサ 1
Java ベースのプロセッサからのマイグレート 135, 139
Java ベースのプロセッサとのマイグレート 136
テキスト・ドキュメントの再生成 138
ネームスペース
サポート、XML ドキュメント 40
宣言と参照 40
ネストされたスカラ・サブクエリ、for xml 句を
使用できない 60

は

汎用マークアップ言語 (SGML) 2

ひ

非 ASCII データ
for xml 句における生成 85
XML ドキュメントとクエリの処理 85
xmlparse での格納 85
処理 93
非 ASCII データの処理 93
非 ASCII データを含む XML ドキュメントの処理 85
非 ASCII データ、Unicode、I18N 85
非 ASCII データ、サポート 85

ふ

ファイル・システム・アクセス、機能 125
フォーマット
SQLX-XML 67
フォーマット指示、XSL による 2
複数言語、XML ベースのアプリケーション 85
プラス記号 (+)、XML ドキュメント・タイプ定義 7
プレーンな文字または定義済みエンティティ 41
プロキシ・テーブル、ファイル・システム・アクセス
機能で作成 125

へ

変数、HTML
... 6
<table>...</table>、レイアウト 6
bcolor、色 6
CustomerID 5, 6
CustomerName 6
ItemID 5
ItemName 5
order 5
Quantities 6
Quantity 5
units 5
データ 6
変数、XML タグ 4

ほ

保持されたスペース 42

ま

マイグレート
Java ベースのプロセッサとネイティブ XML プロセッ
サ間のテキスト・ドキュメント 136
Java ベースのプロセッサとネイティブ XML プロセッ
サ間のドキュメント 135
Java ベースのプロセッサとネイティブ・プロセッサ間
135, 139
XQL プロセッサとネイティブ XML プロセッサ間の
ドキュメント 136

クエリ　138
再生成したコピーからのドキュメント　136
制約　135
ドキュメントとクエリ　135
マッピング
SQL 値　83
SQL 値、例　83
SQL 名、目的　80
XML 値　83
XML 名　80
重複したカラム名　77
重複したカラム名、例　78
名前のないカラム　77

む

矛盾した要素タグ、HTML　6

も

文字セット
UTF8、デフォルト　4
XML　4
XML データ　87
一致、宣言と実際　4
クライアントとサーバの違い　87
指定　4
変換、スキップする　4
文字セット転送のサンプル・ディレクトリ、Java　87
文字セットのサポート　39
文字セットの指定　4
文字セット変換　126
文字セット変換、Java　87
文字値、例　84
文字リテラル　41
文字列／Unicode カラム　86

ゆ

ユーザ作成の要素タグ　4
ユーザによる変更、xmlparse の default xml sort order
オプション　95
ユーロ記号、€　86

よ

要素タグ
HTML　6
埋め込まれた属性　4
カスタマイズ　4
厳密なネスト　4
ユーザ作成　4
要素のカスタマイズ　4
要素、空　42

れ

例
for xml 句　58, 89
XML ドキュメントの解析、XFS を伴う
xmlerror=message の使用　131
xmlerror オプション、XFS　130
xmlerror=message オプションの指定、XFS　131
xmlerror=null の使用　132
外部ファイル・システムに解析済み XML ドキュメント
を格納する　129
サブクエリ、for xml　60
例外
for xml 句　58
サブクエリ、for xml　60
発生する　12