SYBASE®

XML サービス

Adaptive Server® Enterprise

15.5

ドキュメント ID：DC20146-01-1550-01
改訂：2009 年 10 月

マニュアルの注文

マニュアルの注文を承ります。ご希望の方は、サイベース株式会社営業部または代理店までご連絡ください。マニュアルの変更は、弊社の定期的なソフトウェア・リリース時にのみ提供されます。

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

目次

はじめに

対象読者

次のことを行うユーザを対象としています。

- SQL データベースに完全な XML ドキュメントを格納する。
- SQL データベース内の XML ドキュメントのデータのテストおよび抽出を行う。
- XML ドキュメントから抽出したデータを格納する。
- SQL データから XML ドキュメントを生成する。
- SQL データを XML として処理する。

このマニュアルの内容

このマニュアルは、次のように構成されています。

- 「第 1 章 XML サービスの概要」では、データベースにおける XML と Sybase XML サービスの新しい XML 機能の概要について説明します。
- 「第 2 章 XML クエリ関数」では、SQL 文内の XML ドキュメントの処理とクエリについて説明します。これらのクエリ関数は、格納された XML ドキュメント (通常のユーザ・ドキュメント) と、for xml 句、forxmlj 関数、または SQL データの XML ビューを提供する同様のツールによって生成された SQLX-XML ドキュメントの両方に適用できます。これらの関数の詳細については、「第 4 章 XML マッピング関数」を参照してください。
- 「第 3 章 XML 言語と XML 問い合わせ言語」では、サポートされる XPath 言語サブセットの仕様など、XML クエリ関数がサポートする XML ドキュメントと問い合わせ言語の機能について説明します。
- 「第 4 章 XML マッピング関数」では、SQL データと SQLX-XML フォーマットの XML ドキュメントとの間をマップする関数について説明します。
- 「第 5 章 XML マッピング」では、XML マッピング関数でサポートされる XML ドキュメントの SQLX-XML フォーマットについて説明します。
- 「第 6 章 XML における国際化 (I18N) のサポート」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。
- 「付録 A sample_docs テーブル例」では、関数例で使用される sample_docs テーブルについて説明します。

- 「付録 B XML サービスと外部ファイル・システム・アクセス」では、XFS での XML 機能の使用方法の例を示します。
- 「付録 C XML サービスの設定」では、ネイティブの C++ プロセッサと、Adaptive Server バージョン 12.5 以降に含まれる Java プロセッサの両方をインストールする場合のガイドラインを示します。
- 「付録 D Java ベースの XQL プロセッサ」では、XQL を使用して Adaptive Server からロー・データを選択し、結果を XQL を使用して XML ドキュメントとして表示する方法について説明します。
- 「付録 E 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」では、問い合わせ言語の実装や、解析済み形式のドキュメントを返す際に使用する種類の異なる関数およびメソッドと、双方向の変換方法を説明します。
- 「付録 F xmltable() のサンプル・アプリケーション」。

関連マニュアル

Adaptive Server® Enterprise には、次のマニュアルが用意されています。必要に応じて参照してください。

- 使用しているプラットフォームの『リリース・ノート』－ マニュアルには記載できなかった最新の情報が記載されています。

  『リリース・ノート』の最新版 (英語版) にはインターネットからアクセスできます。この製品の CD-ROM がリリースされたあとに追加された重要な製品情報やマニュアル情報を確認する場合は、Sybase Technical Library を参照してください。
- 使用しているプラットフォームの『インストール・ガイド』－ すべての Adaptive Server および関連する Sybase 製品のインストール、アップグレード、設定の手順について説明しています。
- 『Adaptive Server Enterprise 新機能ガイド』－ Adaptive Server バージョン 15.0 の新しい機能について説明しています。また、新しい機能をサポートするためのシステム変更や、既存のアプリケーションに影響する変更についても説明しています。
- 『ASE Replicator ユーザーズ・ガイド』－ プライマリ・サーバから 1 つ以上のリモート Adaptive Server に対して基本的な複写を行うための Adaptive Server の Adaptive Server Replicator 機能の使用方法について説明しています。
- 『コンポーネント統合サービス・ユーザーズ・ガイド』－ リモートの Sybase データベースおよび Sybase 以外のデータベースへ接続するための Adaptive Server コンポーネント統合サービス機能について説明しています。
- 使用しているプラットフォームの『Adaptive Server Enterprise 設定ガイド』－ Adaptive Server の特定の設定作業を行う方法について説明しています。

- 『Enhanced Full-Text Search Specialty Data Store ユーザーズ・ガイド』– Verity で全文検索機能を使用して Adaptive Server Enterprise のデータを検索する方法について説明しています。
- 『用語解説』– Adaptive Server マニュアルで使用されている技術用語について説明しています。
- 『Historical Server ユーザーズ・ガイド』– Historical Server を使用して、SQL Server® と Adaptive Server のパフォーマンス情報を取得する方法について説明しています。
- 『Adaptive Server Enterprise における Java』– Adaptive Server データベースで Java クラスをデータ型、関数、ストアド・プロシージャとしてインストールして使用する方法について説明しています。
- 『Job Scheduler ユーザーズ・ガイド』– コマンド・ラインまたはグラフィカル・ユーザ・インタフェース (GUI) を使用して、ローカルまたはリモートの Adaptive Server でジョブをインストールして設定する方法、および作成してスケジュールする方法について説明しています。
- 『Messaging Services ユーザーズ・ガイド』– Real Time Messaging Services を使用して、TIBCO Java Message Service と IBM WebSphere MQ Messaging Services をすべての Adaptive Server データベース・アプリケーションと統合する方法について説明します。
- 『Monitor Client Library プログラマーズ・ガイド』– Adaptive Server のパフォーマンス・データにアクセスする Monitor Client Library アプリケーションの記述方法について説明しています。
- 『Monitor Server ユーザーズ・ガイド』– Monitor Server を使用して、SQL Server と Adaptive Server のパフォーマンス統計を取得する方法について説明しています。
- 『パフォーマンス&チューニング・シリーズ』– Adaptive Server で最高のパフォーマンスを実現するためのチューニング方法について説明しています。
  - 『基本』– Adaptive Server のパフォーマンスに関する問題の理解と調査の基本について説明しています。
  - 『ロックと同時実行制御』– さまざまなロック・スキームを使用して Adaptive Server のパフォーマンスを向上させる方法と、同時実行性を最小限に抑えるようにインデックスを選択する方法について説明しています。
  - 『クエリ処理と抽象プラン』– オプティマイザがクエリを処理する方法と抽象プランを使用してオプティマイザのプランの一部を変更する方法について説明しています。
  - 『物理データベースのチューニング』– データの物理的配置、データに割り付けられた領域、テンポラリ・データベースの管理方法について説明しています。

- 『sp_sysmon による Adaptive Server の監視』− sp_sysmon を使用して Adaptive Server のパフォーマンスをモニタリングする方法について説明しています。
- 『統計的分析によるパフォーマンスの向上』− Adaptive Server で統計情報がどのように保存され、表示されるかについて説明しています。また、set statistics コマンドを使用して、サーバの統計情報を分析する方法について説明しています。
- 『Using the Monitoring Tables』− Adaptive Server のモニタリング・テーブルに統計情報や診断情報を問い合わせる方法について説明しています。

- 『クイック・リファレンス・ガイド』– コマンド、関数、システム・プロシージャ、拡張システム・プロシージャ、データ型、ユーティリティの名前と構文の包括的な一覧表を記載したポケット版 (PDF 版は通常サイズ) のマニュアルです。
- 『ASE リファレンス・マニュアル』− 詳細な Transact-SQL® 情報を記載しています。このマニュアルは以下の 4 冊に分かれています。
  - 『ビルディング・ブロック』− Transact-SQL のデータ型、関数、グローバル変数、式、識別子とワイルドカード、予約語。
  - 『コマンド』− Transact-SQL のコマンド。
  - 『プロシージャ』− Transact-SQL のシステム・プロシージャ、カタログ・ストアド・プロシージャ、システム拡張ストアド・プロシージャ、dbcc ストアド・プロシージャ。
  - 『テーブル』− Transact-SQL のシステム・テーブルと dbcc テーブル。
- 『システム管理ガイド』でさらに詳しく説明しています。
  - 『第 1 巻』− 設定パラメータ、リソースの問題、文字セット、ソート順、システムの問題の診断方法を含め、システム管理の基本の概要について説明しています。後半は、セキュリティ管理に関する詳細な説明です。
  - 『第 2 巻』− 物理的なリソースの管理、デバイスのミラーリング、メモリとデータ・キャッシュの設定、マルチプロセッサ・サーバとユーザ・データベースの管理、データベースのマウントとマウント解除、セグメントの作成と使用、reorg コマンドの使用、データベース一貫性の検査方法についての手順とガイドラインを説明しています。後半では、システムとユーザ・データベースをバックアップおよびリストアする方法について説明しています。
- 『システム・テーブル・ダイアグラム』− システム・テーブルと、そのエンティティとの関係をポスター形式で図解しています。フル・サイズのダイアグラムは印刷版だけで参照できます。コンパクト版は PDF 形式で参照できます。

- 『Transact-SQL ユーザーズ・ガイド』－ リレーショナル・データベース言語の拡張版である Sybase の Transact-SQL について説明しています。このマニュアルでは、データベース管理システムの操作に慣れていない方のために、テキストブック形式で説明しています。また、pubs2 と pubs3 サンプル・データベースについても説明します。
- 『トラブルシューティング・シリーズ』(リリース 15.0 用)
  - 『トラブルシューティング：エラー・メッセージと詳細な解決方法』－ Sybase® Adaptive Server® Enterprise の使用時に発生する可能性のある問題について、トラブルシューティング手順を説明しています。このマニュアルで取り上げられている問題は、Sybase 製品の保守契約を結んでいるサポート・センタに最も頻繁に寄せられるものです。
  - 『トラブルシューティング＆エラー・メッセージ・ガイド』－ 発生頻度が高い Adaptive Server のエラー・メッセージの解決方法について詳しい手順を説明しています。ここに示されているメッセージのほとんどにはエラー番号 (master..sysmessages テーブルから取得) が含まれていますが、エラー番号がなく Adaptive Server のエラー・ログにのみ出現するエラー・メッセージもあります。
- 『暗号化カラム・ユーザーズ・ガイド』– Adaptive Server を使用して暗号化カラムを設定し、使用する方法について説明しています。
- 『Adaptive Server 分散トランザクション管理機能の使用』－ 分散トランザクション処理環境での Adaptive Server DTM 機能の設定、使用、トラブルシューティングについて説明しています。
- 『高可用性システムにおける Sybase フェールオーバの使用』－ Sybase のフェールオーバ機能を使用して、Adaptive Server を高可用性システムのコンパニオン・サーバとして設定する方法について説明しています。
- 『Unified Agent および Agent Management Console』– Unified Agent について説明します。Unified Agent は、分散 Sybase リソースを管理、モニタ、制御するためのランタイム・サービスを提供します。
- 『ユーティリティ・ガイド』－ オペレーティング・システム・レベルで実行される isql および bcp などの、Adaptive Server のユーティリティ・プログラムについて説明しています。
- 『Web Services ユーザーズ・ガイド』－ Adaptive Server 用の Web Services の設定、使用、トラブルシューティングについて説明しています。
- 『XA インタフェース統合ガイド for CICS、Encina、TUXEDO』－ X/Open XA トランザクション・マネージャを備えた Sybase DTM XA インタフェースを使用する方法について説明しています。
- 『Adaptive Server Enterprise における XML サービス』－ データベースに XML 機能を導入する、Sybase ネイティブの XML プロセッサと Sybase Java ベースの XML のサポートについて、また XML サービスに準拠したクエリとマッピング用の関数について説明しています。

その他の情報

Sybase Getting Started CD、SyBooks™ CD、Sybase® Product Manuals Web サイトを利用すると、製品について詳しく知ることができます。

- Getting Started CD には、PDF 形式のリリース・ノートとインストール・ガイド、SyBooks CD に含まれていないその他のマニュアルや更新情報が収録されています。この CD は製品のソフトウェアに同梱されています。Getting Started CD に収録されているマニュアルを参照または印刷するには、Adobe Acrobat Reader が必要です (CD 内のリンクを使用して Adobe の Web サイトから無料でダウンロードできます)。
- SyBooks CD には製品マニュアルが収録されています。この CD は製品のソフトウェアに同梱されています。Eclipse ベースの SyBooks ブラウザを使用すれば、使いやすい HTML 形式のマニュアルにアクセスできます。

  一部のマニュアルは PDF 形式で提供されています。これらのマニュアルは SyBooks CD の PDF ディレクトリに収録されています。PDF ファイルを開いたり印刷したりするには、Adobe Acrobat Reader が必要です。

  SyBooks をインストールして起動するまでの手順については、Getting Started CD の『SyBooks インストール・ガイド』、または SyBooks CD の *README.txt* ファイルを参照してください。
- Sybase Product Manuals Web サイトは、SyBooks CD のオンライン版であり、標準の Web ブラウザを使用してアクセスできます。また、製品マニュアルのほか、EBFs/Updates、Technical Documents、Case Management、Solved Cases、ニュース・グループ、Sybase Developer Network へのリンクもあります。

  Sybase Product Manuals Web サイトは、Product Manuals にあります。(http://www.sybase.com/support/manuals/)

Web 上の Sybase 製品の動作確認情報

Sybase Web サイトの技術的な資料は頻繁に更新されます。

❖ **製品認定の最新情報にアクセスする**

1. Web ブラウザで Technical Documents を指定します。(http://www.sybase.com/support/techdocs/)
2. [Certification Report] をクリックします。
3. [Certification Report] フィルタで製品、プラットフォーム、時間枠を指定して [Go] をクリックします。
4. [Certification Report] のタイトルをクリックして、レポートを表示します。

❖ **コンポーネント認定の最新情報にアクセスする**

1 Web ブラウザで Availability and Certification Reports を指定します。(http://certification.sybase.com/)

2 [Search By Base Product] で製品ファミリとベース製品を選択するか、[Search by Platform] でプラットフォームとベース製品を選択します。

3 [Search] をクリックして、入手状況と認定レポートを表示します。

❖ **Sybase Web サイト (サポート・ページを含む) の自分専用のビューを作成する**

MySybase プロファイルを設定します。MySybase は無料サービスです。このサービスを使用すると、Sybase Web ページの表示方法を自分専用にカスタマイズできます。

1 Web ブラウザで Technical Documents を指定します。(http://www.sybase.com/support/techdocs/)

2 [MySybase] をクリックし、MySybase プロファイルを作成します。

Sybase EBF とソフトウェア・メンテナンス

❖ **EBF とソフトウェア・メンテナンスの最新情報にアクセスする**

1 Web ブラウザで Sybase Support Page (http://www.sybase.com/support) を指定します。

2 [EBFs/Maintenance] を選択します。MySybase のユーザ名とパスワードを入力します。

3 製品を選択します。

4 時間枠を指定して [Go] をクリックします。EBF/Maintenance リリースの一覧が表示されます。

鍵のアイコンは、「Technical Support Contact」として登録されていないため、一部の EBF/Maintenance リリースをダウンロードする権限がないことを示しています。未登録でも、Sybase 担当者またはサポート・コンタクトから有効な情報を得ている場合は、[Edit Roles] をクリックして、「Technical Support Contact」の役割を MySybase プロファイルに追加します。

5 EBF/Maintenance レポートを表示するには [Info] アイコンをクリックします。ソフトウェアをダウンロードするには製品の説明をクリックします。

表記規則

次の項では、このマニュアルで使用されている表記について説明します。

SQL は自由な形式の言語で、1 行内のワード数や、改行の仕方に規則はありません。このマニュアルでは、読みやすくするため、例や構文を文の句ごとに改行しています。複数の部分からなり、2 行以上にわたる場合は、字下げしています。複雑なコマンドの書式には、修正された BNF (Backus Naur Form) 記法が使用されています。

表 1 に構文の規則を示します。

表 *1:* このマニュアルでのフォントと構文規則

要素	例
コマンド名、プロシージャ名、ユーティリティ名、その他のキーワードは sans serif フォントで表記する。	select sp_configure
データベース名とデータ型は sans serif フォントで表記する。	master データベース
ファイル名、変数、パス名は斜体で表記する。	システム管理ガイド *sql.ini* ファイル *column_name* *$SYBASE/ASE* ディレクトリ

変数 (ユーザが入力する値を表す語) がクエリまたは文の一部である場合は Courier フォントの斜体で表記する。	`select column_name`
	`from table_name`
	`where search_conditions`

カッコはコマンドの一部として入力する。	compute *row_aggregate* (*column_name*)
2 つのコロンと等号は、構文が BNF 表記で記述されていることを示す。この記号は入力しない。「〜と定義されている」ことを意味する。	`::=`
中カッコは、その中のオプションを 1 つ以上選択しなければならないことを意味する。コマンドには中カッコは入力しない。	`{cash, check, credit}`
角カッコは、オプションを選択しても省略してもよいことを意味する。コマンドには角カッコは入力しない。	`[cash | check | credit]`
中カッコまたは角カッコの中のカンマで区切られたオプションをいくつでも選択できることを意味する。複数のオプションを選択する場合には、オプションをカンマで区切る。	`cash, check, credit`
パイプまたは縦線は複数のオプションのうち 1 つだけを選択できることを意味する。	`cash | check | credit`
省略記号 (...) は、直前の要素を必要な回数だけ繰り返し指定できることを意味する。	`buy thing = price [cash | check | credit] [, thing = price [cash | check | credit]]...` この例では、製品 (thing) を少なくとも 1 つ購入 (buy) し、価格 (price) を指定する必要があります。支払方法を選択できる。角カッコで囲まれた項目の 1 つを選択する。追加品目を、必要な数だけ購入することもできる。各 buy に対して、購入した製品 (thing)、価格 (price)、オプションで支払方法 (cash、check、credit のいずれか) を指定します。

- 次は、オプション句のあるコマンドの構文の例です。

  sp_dropdevice [*device_name*]

  複数のオプションを持つコマンドの例を示します。

  select *column_name*
  from *table_name*
  where *search_conditions*

  構文では、キーワード (コマンド) は通常のフォントで表記し、識別子は小文字で表記します。ユーザが提供するワードは斜体で表記します。

- Transact-SQL コマンドの使用例は次のように表記します。

  ```
  select * from publishers
  ```

- 次は、コンピュータからの出力例です。

```
pub_id    pub_name                city          state
-------   ---------------------   -----------   -----
0736      New Age Books           Boston        MA
0877      Binnet & Hardley        Washington    DC
1389      Algodata Infosystems    Berkeley      CA

(3 rows affected)
```

このマニュアルでは、例に使用する文字はほとんどが小文字ですが、Transact-SQL のキーワードを入力するときは、大文字と小文字は区別されません。たとえば、SELECT、Select、select はすべて同じです。

テーブル名などのデータベース・オブジェクトの大文字と小文字を Adaptive Server が区別するかどうかは、Adaptive Server にインストールされたソート順によって決まります。シングルバイト文字セットを使用している場合は、Adaptive Server のソート順を再設定することによって、大文字と小文字の区別の取り扱い方を変更できます。詳細については、『システム管理ガイド』を参照してください。

アクセシビリティ機能

このマニュアルには、アクセシビリティを重視した HTML 版もあります。この HTML 版マニュアルは、スクリーン・リーダーで読み上げる、または画面を拡大表示するなどの方法により、その内容を理解できるよう配慮されています。

Adaptive Server HTML マニュアルは、連邦リハビリテーション法第 508 条のアクセシビリティ規定に準拠していることがテストにより確認されています。第 508 条に準拠しているマニュアルは通常、World Wide Web Consortium (W3C) の Web サイト用ガイドラインなど、米国以外のアクセシビリティ・ガイドラインにも準拠しています。

注意 アクセシビリティ・ツールを効率的に使用するには、設定が必要な場合もあります。一部のスクリーン・リーダーは、テキストの大文字と小文字を区別して発音します。たとえば、すべて大文字のテキスト (ALL UPPERCASE TEXT など) はイニシャルで発音し、大文字と小文字の混在したテキスト (Mixed Case Text など) は単語として発音します。構文規則を発音するようにツールを設定すると便利かもしれません。詳細については、ツールのマニュアルを参照してください。

Sybase のアクセシビリティに対する取り組みについては、Sybase Accessibility (http://www.sybase.com/accessibility) を参照してください。Sybase Accessibility サイトには、第 508 条と W3C 標準に関する情報へのリンクもあります。

不明な点があるときは

Sybase ソフトウェアがインストールされているサイトには、Sybase 製品の保守契約を結んでいるサポート・センタとの連絡担当の方 (コンタクト・パーソン) を決めてあります。マニュアルだけでは解決できない問題があった場合には、担当の方を通して Sybase のサポート・センタまでご連絡ください。

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。

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」(103 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。

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 *datatype*
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」(103 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 jjava.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」(103 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_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 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。 このオプションは、Adaptive Server Enterprise 12.5 の Java ベースの XQL プロセッサとの互換性を保つためにあります。

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

  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	マニュアル
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	マニュアル
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 この例は、スキーマ宣言を含むドキュメントと、*schemavalidate=yes* 検証オプションを指定する 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 この例は、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。

たとえば、forxmlj には *XML document* パラメータはありませんが、xmlerror オプション (無効な XML オペランドの動作を指定するオプション) を含む *option_string* を受け入れます。

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	forxmlj と for xml 句
columnstyle	element | attribute	forxmlj と for xml 句
dtdvalidate	yes | no	xmlvalidate
entitize	yes | no | conditional	forxmlj と for xml 句
format	yes | no	forxmlj と for xml 句
header	yes | no |encoding	forxmlj と for xml 句
incremental	yes | no	for xmlj 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。	forxmlj、for xml 句、xmlextract
nullstyle	attribute | omit	forxmlj と for xml 句
nullclause	null | empty	forsqlcreatej forsqlscriptj
prefix	SQL 名 (C) デフォルト値は C です。	forxmlj と for xml 句
root	yes | no	forxmlj と for xml 句
rowname	SQL 名 (row)	forxmlj と for xml 句
schemalocation	「xmlvalidate」を参照。	forxmlj と for xml 句
schemavalidate	yes | no	xmlvalidate
statement	yes | no	for xmlj と for xml 句
tablename	SQL 名 (resultset)	forxmlj と for xml 句
targetns	引用符で囲まれた URI 文字列	forxmlj と 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	forxmlj と 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。

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 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 *&apol:*, 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
>=	以上
<=	以下

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"]//firstname', 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

XML マッピング関数

この章では、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」(103 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」(79 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, “Commands, 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)
  ```

  この xml 値は、次のクエリ結果の java.lang.String 値と同じです。

  ```
  forxmlj('subquery', 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」(79 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」(79 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)
```

forxmlj、forxmldtdj、forxmlschemaj、forxmlallj

注意　この項の関数は Java ベースであるため、サーバに関数をインストールしないと使用できません。詳細については、「付録 D Java ベースの XQL プロセッサ」を参照してください。

Java ベースの forxml 関数は、SQL クエリの結果セットを SQLX-XML スキーマ、結果セット・ドキュメント、またはその両方にマップします。SQL クエリは、任意の SQL クエリ式を含む文字列です。

forxmlj は、select 文の for xml 句によって提供される関数形式のマッピングです。これらの違いは次のとおりです。

- 関数の引数、update 文の set 句、insert 文の値リストなどの一部のコンテキストでは、forxmlj 関数は使用できても for xml を指定した select 文は使用できません。
- for xml 句が指定された select 文は、*returns_clause* で指定されたデータ型の結果を返します。forxmlj 関数は、結果を java.lang.String として返します。
- for xml 句を指定した select 文は、*incremental* オプションに応じて単一のローまたは複数のローを返します。forxmlj 関数は、単一の結果を返します。

構文

forxmljfunction ::=
    forxmlj(*sql_query_expression*, *option_string*)
    | forxmldtdj(*sql_query_expression*, *option_string*)
    | forxmlschemaj(*sql_query_expression*, *option_string*)
forxmlallj_procedure::=
    execute forxmlallj
        sql_query_expression, *option_string*
        rs_target_out, *schema_target_out*, *dtd_target_out*
sql_query_expression::=*basic_string_expression*
option_string::=*basic_string_expression*

説明

- *basic_string_expression* は、データ型が char、varchar、unichar、univarchar、または java.lang.String の *sql_query_expression* です。
- forxmlj のいずれかのパラメータが null の場合、呼び出し結果は null になります。
- *sql_query_expression* がすべてブランクまたは空の文字列の場合、呼び出し結果は空の文字列になります。
- *sql_query_expression* には有効な SQL の select 文が含まれている必要があります。select 文には from 句、where 句、group by 句、having 句、order by 句を含めることができます。into 句、compute 句、read_only 句、isolation 句、browse 句、または plan 句を含めることはできません。
- forxmlj は *sql_query_expression* を評価し、SQLX 結果セットとしてフォーマットされた結果セットを含む SQLX-XML ドキュメントを返します。

- forxmldtdj は *sql_query_expression* を評価し、そのクエリの SQLX-XML 結果セットを記述する XML DTD を返します。
- forxmlschemaj は *sql_query_expression* を評価し、そのクエリの SQL-XML 結果セットを記述する SQLX-XML スキーマを返します。
- forxmlallj プロシージャは *sql_query_expression* を評価し、そのクエリの SQLX-XML 結果セット、スキーマ、DTD を返します。

注意 SQL 結果セットの SQLX-XML 表現の説明については、「第 5 章 XML マッピング」を参照してください。

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、「第 5 章 XML マッピング」を参照してください。

例外

sql_query_expression の実行中に発生する SQL 例外は、forxmlj 関数によって発生します。

例

次は、forxmlj 関数の例です。

```
set stringsize 16384
select forxmlj
       ("select pub_id, pub_name
          from pubs2.dbo.publishers", "")
----------
<resultset xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
<row>
   <pub_id>0736</pub_id>
   <pub_name>New AgeBooks</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>
```

次は、forxmldtdj 関数の例です。

```
set stringsize 16384
select forxmldtdj
      ("select pub_id, pub_name
         from pubs2.dbo.publishers",
       "tablename=extract nullstyle=omit")
-----------
<!ELEMENT extract (row*)>
<!ELEMENT row (pub_id, pub_name?)>
<!ELEMENT pub_id (#PCDATA)>
<!ELEMENT pub_name (#PCDATA)>
```

次は、forxmlschemaj 関数の例です。

```
set stringsize 16384
select forxmlschemaj
     ("select pub_id, pub_name
        from pubs2.dbo.publishers",
      "tablename=extract nullstyle=omit")

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

<xsd:simpleType name="CHAR_4">
   <xsd:restriction base="xsd:string">
      <xsd:length value="4"/>
   </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="VARCHAR_40">
   <xsd:restriction base="xsd:string">
      <xsd:length value="40"/>
   </xsd:restriction>
</xsd:simpleType>

<xsd:complexType name="RowType.extract">
   <xsd:sequence>
       <xsd:element name="pub_id" type="CHAR_4"
          minOccurs="0" MaxOccurs="1"/>
        <xsd:element name="pub_name" type="VARCHAR_40"
          minOccurs="0" maxOccurs="1"/>
   </xsd:sequence>
</xsd:complexType>

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

```
</xsd:complexType>

<xsd:element name="extract" type="TableType.extract"/>
</xsd:schema>
```

次は、forxmlallj プロシージャの例です。

```
set stringsize 16384
declare @rs varchar(16384)
declare @schema varchar(16384)
declare @dtd varchar(16384)
execute forxmlallj
  "select pub_id, pub_name from pubs2.dbo.publishers",
  "name=extract  null=attribute",
  @rs out, @schema    out, @dtd out
------------
<extract
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>
</extract>

<xsd:schema
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:sqlxml=
   "http://www.iso-standards.org/mra/9075/sqlx">
<xsd:simpleType name="CHAR_4">
   <xsd:restriction base="xsd:string">
      <xsd:length value="4"/>
   </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="VARCHAR_40">
   <xsd:restriction base="xsd:string">
     <xsd:length value="40"/>
   </xsd:restriction>
</xsd:simpleType>

<xsd:complexType name="RowType.extract">
   <xsd:sequence>
```

```
      <xsd:element name="pub_id" type="CHAR_4"
           nullable="true" />
      <xsd:element name="pub_name" type="VARCHAR_40"
           nullable="true" />
   </xsd:sequence>
</xsd:complexType>

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

```
<!ELEMENT extract (row*)>
<!ELEMENT row (pub_id, pub_name)>
<!ELEMENT pub_id (#PCDATA)>
<!ELEMENT pub_name (#PCDATA)>
```

forsqlcreatej、forsqlinsertj、forsqlscriptj

注意　この項の関数は Java ベースであるため、サーバに関数をインストールしないと使用できません。

Java ベースの forsql 関数は、SQLX-XML スキーマと SQLX-XML 結果セット・ドキュメントを SQL スクリプトにマップします。

- SQLX-XML スキーマと結果セット・ドキュメントは、forxmlj 関数で生成される形式です。
- forsqlschemaj 関数は SQLX-XML スキーマを SQL の create コマンドにマップし、SQLX-XML スキーマで記述されるデータに適したテーブルを作成します。
- forxmlinsertj 関数は、SQLX-XML 結果セットを一連の SQL の insert コマンドにマップし、SQLX-XML 結果セットによって記述されるデータを再作成します。

- forxmlscriptj 関数は、SQLX-XML スキーマと SQLX-XML 結果セットの両方を SQL の create コマンドにマップし、SQLX-XML スキーマによって記述されるデータに適したテーブルと、SQLX-XML 結果セットによって記述されるデータを再作成する一連の SQL の insert コマンドを作成します。

構文

sqlx_to_sql_script_function ::=
forsqlcreatej(*sqlx_schema*, *option_string*)
| forsqlinsertj(*sqlx_resultset*, *option_string*)
|forsqlscriptj(*sqlx_schema*, *sqlx_resultset*, *option_string*)
sqlx_schema::=*basic_string_expression*
sqlx_resultset::=*basic_string_expression*
option_string::=*basic_string_expression*

説明

- *basic_string_expression* は、データ型が char、varchar、unichar、univarchar、または java.lang.String の *sql_query_expression* です。
- forsqlcreatej、forsqlschemaj、または forsqlscriptj のいずれかのパラメータが null の場合、呼び出し結果は null になります。
- *sqlx_schema* または *sqlx_resultset* がすべてブランクか空の文字列の場合、呼び出し結果は空の文字列になります。
- *sqlx_schema* には、SQLX-XML スキーマを含む有効な XML ドキュメントが含まれている必要があります。
- *sqlx_resultset* には、SQLX-XML 結果セットを含む有効な XML ドキュメントが含まれている必要があります。
- forsqlcreatej は、*sqlx_schema* によって記述されるデータに適した SQL テーブルを作成する、SQL の create コマンドを生成します。
- forsqlinsertj は、SQL テーブルに *sqlx_resultset* のデータを移植する、一連の SQL の insert コマンドを生成します。

  この関数は、対応するスキーマがなくても SQLX-XML 結果セットを操作するため、生成される insert コマンドでは、すべてのデータが varchar であることが想定されます。
- forsqlscriptj は、*sqlx_resultset* のデータを SQL テーブルに移植する、SQL の create と一連の SQL の insert コマンドを生成します。

  この関数は SQLX-XML スキーマと結果セットの両方を操作するため、create は *sqlx_schema* のカラム・データ型を指定し、insert コマンドではそれらのデータ型が想定されます。
- 生成されるスクリプトでは、すべての識別子に引用符で囲まれた識別子を使用します。これは、通常の識別子への後続の参照には影響しません。

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。

forsqlcreatej、forsqlinsertj、forsqlscripj の各関数では、次の「例外」の項で説明するオプションがサポートされています。

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

例外

sqlx_schema または *sqlx_resultset* の値が有効な XML ではない場合は、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次のように指定した場合、

  ```
  xmlerror=exception
  ```

  次の例外が発生します。

  ```
  invalid XML data
  ```

- 明示的なオプションまたはデフォルトのオプションで次のように指定した場合、

  ```
  xmlerror=null
  ```

  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次のように指定した場合、

  ```
  xmlerror=message
  ```

  例外メッセージを含む XML 要素の文字列が返されます。この値は SQL コメントの形式であるため、返される値は有効な SQL です。

例

次は、forsqlcreatej 関数の例です。

```
set stringsize 16384
declare @schema varchar(16384)
select @schema = forxmlschemaj(
   "select pub_id, pub_name from pubs2.dbo.publishers",
   "tablename=extract  null=attribute")
select forsqlcreatej(@schema, "")
------------
CREATE TABLE "extract"(
   "pub_id" CHAR(4) null,
   "pub_name" VARCHAR(40) null )
```

次は、forsqlinsertj 関数の例です。

```
set stringsize 16384
declare @rs varchar(16384)
select @rs = forxmlj(
   "select pub_id, pub_name from pubs2.dbo.publishers")
select forsqlinsertj(@rs, "")
------------
   --Begin table "resultset"
insert into "resultset"
   ("pub_id", "pub_name")
    values ( '0736', 'New Age Books')
insert into "resultset"
   ("pub_id", "pub_name")
    values ( '0877', 'Binnet & Hardley')
```

```
insert into "resultset"
   ("pub_id", "pub_name")
    values ( '1389', 'Algodata Infosystems')
--End table "resultset"
```

次は、forsqlscriptj 関数の例です。

```
set stringsize 16384
declare @rs varchar(16384)
declare @schema varchar(16384)
declare @dtd varchar(16384)
execute forxmlallj
   "select pub_id, pub_name from pubs2.dbo.publishers",
   "tablename=extract  null=attribute",
    @rs out, @schema out, @dtd out
declare @script varchar(16384)

select @script = forsqlscriptj(@schema, @rs, "")
select @script
execute ("set quoted_identifier on " + @script )
execute ("select pub_id, pub_name from extract")
execute ("drop table extract")
-----------
(return status = 0)

Return parameters:

*****Values of @rs, @schema, and @dtd omitted********
(1 row affected)
(1 row affected)

CREATE TABLE "extract"(
   "pub_id" CHAR(4) null,
    "pub_name" VARCHAR(40) null )

--Begin table "extract"
insert into "extract"
   ("pub_id", "pub_name")
    values ( '0736', 'New Age Books')
insert into "extract"
   ("pub_id", "pub_name")
    values ( '0877', 'Binnet & Hardley')
insert into "extract"
   ("pub_id", "pub_name")
    values ( '1389', 'Algodata Infosystems')
--End table "extract"

(1 row affected)
(1 row affected)
(1 row affected)
(1 row affected)
```

```
pub_id pub_name
------ --------------------
1)  New Age Books
2)  Binnet & Hardley
3)  Algodata Infosystems

(3 rows affected)
```

Java 関数を使用して階層構造の XML ドキュメントと SQL データをマップする

Adaptive Server は、SQL テーブルまたは結果セットと階層構造の XML ドキュメントとの間でデータをマップする次の 2 つのクライアント指向の Java ベース XML 関数をサポートしています。コマンド・ライン・トレース・フラグは次のとおりです。

- ForXmlTree － SQL テーブルまたは結果セットの集合をツリー構造の XML ドキュメントにマップします。
- OpenXml － ツリー構造の XML ドキュメントの繰り返しデータを SQL テーブルに抽出します。

以降の各項では、サンプル・データを使用して、ForXmlTree と OpenXml の使用方法の概要と例を示します。詳細については、*$SYBASE/$SYBASE_ASE/sample/XML/xml-util.{doc, pdf}* を参照してください。

サンプル・データとツリー構造の XML 表現

SQL データはテーブルに格納され、外部キー・カラムとプライマリ・キー・カラムによってテーブル間のツリー構造関係が形成されます。このようなデータを XML で記述する場合、ツリー構造関係は通常ネストされた要素で表されます。

たとえば、表 4-1 に示すデータを含むテーブルを考えてみます。

表 *4-1:* サンプル・テーブル

テーブル・データ
depts(dept_id, dept_name)
emps(emp_id, emp_name, dept_id)
emp_phones(emp_id, phone_no
projects(project_id, dept_id)

表 4-1 のデータをツリー構造の XML で表すと、次のようになります。

```
<sample xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<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>510.555.1876</phone_no></phone>
                   <!-- other phone elements for this emp -->
               </phones>
            <!-- Other emp elements for this dept -- >
       </emps>

       <projects>
             <project>
                  <project_id>PABC</project_id>
                  <budget>598.76</budget>
             </project>
             <!-- Other project elements for this dept - ->
         </projects>
     </dept>
    <!-- other dept elements for this set of depts.-->
</depts>
</sample>
```

ForXmlTree を使用して階層構造の XML に SQL データをマップする

Java ベースの関数 ForXmlTree は、SQL テーブルまたは結果セットの集合をツリー構造の XML ドキュメントにマップします。この関数は Adaptive Server 12.5.1 で導入された SQL の select コマンドの for xml 句を基にしています。

select...for xml は次のタスクを実行します。

- 1 つの SQL 結果セットを 1 つの XML ドキュメントにマップする。
- SQL 結果セットから XML への直接マッピングを生成する。たとえば select が、各ローに 20 のカラムを含む 1000 ローの結果セットを返す場合、for xml が返す XML ドキュメントには 1000 個のロー要素があり、各要素に 20 個のカラム要素が含まれます。

Java ベースの関数 ForXmlTree には次のような特徴があります。

- SQL サーバ、クライアントのコマンド・ライン、クライアントまたはサーバの Java アプリケーションのいずれかで呼び出すことができる。
- 結果セットの集合を 1 つのツリー構造の XML ドキュメントにマップする。
- 出力ツリーとそのツリーの各ノードに含める SQL データを記述する *<forxmltree>* 指定引数が必要である。
- ツリー構造の出力 XML ドキュメントの各ノードに for xml 形式の XML データ・マッピングを生成する。

このため、ForXmlTree は 2 次元の for xml マッピングができると考えることができます。たとえば、ForXmlTree に次の *<forxmltree>* を入力すると、「サンプル・データとツリー構造の XML 表現」(73 ページ) に示した XML ドキュメントが生成されます。

```
1) <!-- A forxmltree spec for depts-emps-phones-projects, with aggregation -->
2) <forxmltree treename="sample">
3) <node> <!-- The node element for depts -->
4)    <query> select * from depts order by dept_id </query>
5)    <options> tablename=depts rowname=dept </options>
6)    <link variablename="@dept_id" columnname="dept_id" type="char(11)" />
7)    <node> <!-- The node element for emps, under depts -->
8)       <query>
9)            select emp_id, emp_name, salary from emps e
10)            where e.dept_id = @dept_id order by emp_id
11)       </query>
12)       <options> tablename=emps rowname=emp   </options>
13)        <link variablename="@emp_id" columnname="emp_id" type="char(6)"/>
14)          <node> <!-- The node element for phones, under emps -->
15)            <query>
16)             select phone_no from emp_phones ep where ep.emp_id = @emp_id
17)            </query>
18)            <options> tablename=phones rowname=phone </options>
19)          </node> <!-- End the node for phones -->
20)    </node> <!-- End the node for emps --
21)   <node> <!-- The node element for projects, under dept -->
22)       <query>
23)           select project_id, budget from projects p
24)           where p.dept_id = @dept_id order by project_id
25)       </query>
26)       <options> tablename=projects rowname=project </options>
27)   </node> <!-- End the node for projects -->
28) </node> <!-- End the node for depts -->
29) </forxmltree>
```

OpenXml を使用して階層構造の XML を SQL にマップする

「ForXmlTree を使用して階層構造の XML に SQL データをマップする」(74 ページ) で説明した ForXmlTree 関数は SQL テーブルまたは結果セットの集合を階層構造の XML ドキュメントにマップします。OpenXml 関数はこれとは逆の処理を行い、入力 XML ドキュメントから SQL テーブルのデータを抽出します。

OpenXml は Adaptive Server 12.5.1 で導入された xmlextract 関数に似ています。この関数は特定の XML ドキュメントから指定したデータ値を抽出します。xmlextract では XML ドキュメントと 1 つの XPath クエリ式を指定し、XPath クエリを XML ドキュメントに適用した結果を返します。

Java ベースの OpenXml 関数には次のような特徴があります。

- クライアントのコマンド・ラインまたはクライアントの Java アプリケーションから呼び出すことができる。SQL サーバでの使用は想定されていません。
- 特定の XML ドキュメントを示す引数と、目的の出力ローを抽出する XPath クエリと各出力ローの必要なカラムを抽出する Xpath クエリを指定するオプション・セットを示す引数が必要である。

したがって、OpenXml は 2 次元の xmlextract と見なすことができます。

OpenXml は次のどちらか一方または両方のアクションを行います。

- SQL テーブルを作成し、抽出したデータを移植する SQL スクリプトを生成する。
- そのスクリプトを実行し、抽出したデータで SQL テーブルを作成する。

以下の例は、「サンプル・データとツリー構造の XML 表現」(73 ページ) に示した XML ドキュメントが *example-document.xml* に格納されていると想定しています。

例 1

次の例は、XML ドキュメントから depts、emps、emp_phones、projects の各テーブルを抽出する 4 つのクライアント・コマンド・ライン呼び出しを示します。

```
java jcs.xmlutil.OpenXml -i "file:example-document.xml" ¥
     -r "file:depts.opt" -o "depts.sql"

java jcs.xmlutil.OpenXml -i "file:example-document.xml" ¥
       -r "file:emps.opt" -o "emps.sql"

java jcs.xmlutil.OpenXml -i "file:example-document.xml" ¥
       -r "file:emp-phones.opt" -o "emp-phones.sql"

java jcs.xmlutil.OpenXml -i "file:example-document.xml" ¥
       -r "file:projects.opt" -o "projects.sql"
```

例 2

この例は、例 1 に示したコマンド・ラインで呼び出されるオプションの内容を示します。これらのオプションは、OpenXml 呼び出しで抽出するデータと、その格納先の SQL テーブルを指定します。

```
-- Content of input file "depts.opt
"tablename='depts_ext'
rowpattern='//dept'
columns=
  '   dept_id char( 4 ) "/@dept_id"
      dept_name varchar(50) "/@dept_name" '

-- Content of input options file "emps.opt"
tablename='emps_ext'
rowpattern='//dept/emps/emp'
columns=
  '   emp_id char( 4 ) "/emp_id/text()"
      emp_name varchar(50) "/emp_name/text()"
       dept_id char(4) "/../../@dept_id"
       salary dec(7,2) "/salary/text()"

'-- Content of input options file "emp-phones.opt"
tablename='emp_phones_ext'
rowpattern='/sample/dept/emps/emp/phone'
columns=  '   emp_id char( 4 ) "/../emp_id/text()"
       phone_no varchar(20) "/@phone_no" '

--Content of input options file "projects.opt"
tablename='projects_ext'
rowpattern='//dept/projects/project'
columns=
  '   project_id char( 4 ) "/project_id/text()"
      dept_id char(4) "/../../@dept_id"
      budget dec(7,2) "/budget/text()" '
```

例 3

この例は、最初の OpenXml 呼び出しで生成される SQL スクリプトを示します。このスクリプトは、テーブルを作成し、depts テーブルの抽出データをそのテーブルに移植します。例 1 に示した後続の OpenXml 呼び出しはそれぞれ、emps、emp_phones、projects のデータに対する同様のスクリプトを生成します。

```
-- output file depts.sql

create table depts_ext
    (dept_id char( 4 ) null, dept_name varchar(50) null
)
insert into depts_ext values('D123', 'Main')

insert into depts_ext values('D234', 'Auxiliary')

insert into depts_ext values('D345', 'Repair')
```

複数の結果セット・クエリに対する Java SQLX マッピング

select ... for xml 文と Java ベースの SQLX マッピング関数は 1 つの SQL 結果セットを SQLX フォーマットの 1 つの XML ドキュメントにマップします。Adaptive Server が提供する Java ベースの SQLX マッピング関数 forxmlmultiplej は、SQL クエリの複数の結果セットを 1 つの XML ドキュメントにマップします。

forxmlmultiplej

説明

SQL クエリの結果セットを XML ドキュメントにマップします。複数の結果セットをマップできます。

構文

```
forxmlmultiplej_function ::=
    forxmlmultiplej(sql_query_expression, option_string)
```

オプション

sql_query_expression と *option_string* については、『Adaptive Server Enterprise における XML Services』の「第 4 章 XML マッピング関数」にある「forxmlj、forxmldtdg、forxmlschemaj、forxmlallj」を参照してください。

使用法

- *sql_query_expression* は複数の結果セットを返すことができ、SQL の print コマンドを含めることができます。
- forxmlmultiplej の例と詳細については、*$SYBASE/$SYBASE_ASE/sample/XML/Using-SQLX-mappings.htm* で、複数の結果セットについての項目を参照してください。

第 5 章

XML マッピング

select 文の for xml 句と forxmlj 関数は、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 句と forxmlj 関数の両方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。

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。forxmlj に適用される。
columnstyle	element | attribute	SQL カラムの表現
entitize	yes | no | cond	forxmlj と 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

オプション名	オプションの値	目的
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	forxmlmultiplej と for xml 句
nullstyle	attribute | omit	*columnstyle=element* での null の表現
ncr	non_ascii | non_server | no	forxmlj と 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	forxmlj と 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。

次の例は、デフォルト・オプションである *binary=hex* を示します。

```
select forxmlj("select 0x012131415161718191a1b1c1d1e1f1",
"binary=hex")
-------------------------------------------
<resultset
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <row>
      <C1>012131415161718191A1B1C1D1E1F1</C1>
   </row>

</resultset>
```

次の例は、*binary=base64* を示します。

```
select forxmlj("select 0x012131415161718191a1b1c1d1e1f1",
"binary=base64")
-------------------------------------------
<resultset xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<row>
     <C1>ASExQVFhcYGRobHB0eHx</C1>
</row>

</resultset>
```

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 (<、&apos >、&、&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」(103 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」(104 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>

select forxmlj("select 11, 12 union select 21, 22","root=no")
```

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」(92 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」(92 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 句と forxmlj 関数の両方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 (@quot;、'、<、>、&) 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。forxmlj 関数の文字リテラル引数は、埋め込み引用符を二重にします。

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

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

例については、「SQLX オプション」(79 ページ) を参照してください。

SQLX スキーマ・マッピング

forxmlschemaj 関数と forxmlallj 関数は、指定された結果セットの SQLX-XML ドキュメントを記述する XML スキーマを生成します。この項では、生成されるこのような XML スキーマの概要を示します。これらの XML スキーマは、一般に XML ツールによってのみ使用されるため、各行を詳細に理解する必要はありません。

概要

次の SQL 結果セットには 5 つのカラムがあり、そのデータ型はそれぞれ varchar(3)、numeric(3,1)、varbinary(2)、numeric(3,1)、numeric(3,2) です。

```
select 'abc', 12.3, 0x00, 45.6, 7.89
--- ------ ---- ------ ------
abc   12.3 0x00   45.6   7.89
```

このデータの SQLX-XML 結果セットは次のとおりです。

```
select forxmlj("select 'abc', 12.3, 0x00, 45.6, 7.89", "")
------------------------------------------------------
<resultset xmlns:xsi="http://www.w3.org/2001
       /XMLSchema-instance">
   <row>
      <C1>abc</C1>
      <C2>12.3</C2>
      <C3>00</C3>
      <C4>45.6</C4>
      <C5>7.89</C5>
   </row>

</resultset>
```

このドキュメントを記述する SQLX-XML スキーマは次のとおりです。

```
select forxmlschemaj("select 'abc', 12.3, 0x00, 45.6, 7.89", "")
----------------------------------------
<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/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_3" />
        <xsd:element name="C2" type="NUMERIC_3_1" />
        <xsd:element name="C3" type="VARBINARY_2" />
        <xsd:element name="C4" type="NUMERIC_3_1" />
        <xsd:element name="C5" type="NUMERIC_3_2" />
     </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_3">
     <xsd:restriction base="xsd:string">
         <xsd:length value="3"/>
     </xsd:restriction>
   </xsd:simpleType>

   <xsd:simpleType name="NUMERIC_3_1">
     <xsd:restriction base="xsd:decimal">
        <xsd:totalDigits value="3"/>
        <xsd:fractionDigits value="1"/>
     </xsd:restriction>
   </xsd:simpleType>

   <xsd:simpleType name="VARBINARY_2">
     <xsd:restriction base="xsd:hexBinary">
        <xsd:length value="2"/>
     </xsd:restriction>
     </xsd:simpleType>

   <xsd:simpleType name="NUMERIC_3_2">
     <xsd:restriction base="xsd:decimal">
        <xsd:totalDigits value="3"/>
        <xsd:fractionDigits value="2"/>
```

```
     </xsd:restriction>
   </xsd:simpleType>

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

</xsd:schema>
```

この XML スキーマには、次の 5 つのコンポーネントがあります。

- このサンプル XML スキーマの最後の部分には、XML ドキュメントの 4 種類のデータ型に対して単純な XML タイプを宣言する 3 つの *xsd:simpleType* 要素があります。これらの *simpleType* 宣言は、各タイプの XML ベース・タイプを指定し、SQL データの長さ特性を定義する *xsd:restriction* 要素を指定します。*simpleType* 宣言にはそれぞれ、VARCHAR_3、NUMERIC_3_1、VARBINARY_2、NUMERIC_3_2 という XML 名があります。
- XML スキーマには、SQL データ型、長さ、精度の属性の組み合わせごとに別々の *xsd:simpleType* が含まれます。たとえば、NUMERIC_3_1 と NUMERIC_3_2 には異なるタイプがあります。ただし、NUMERIC_3_1 タイプに 2 つのカラムがある場合でも、このタイプには *xsd:simpleType* 宣言が 1 つしかありません。これらのカラムの要素宣言は、どちらも同じ単純なタイプ名 NUMERIC_3_1 を参照します。
- サンプル XML スキーマの最初の部分は、各カラムの要素を定義する row タイプの *xsd:complexType* です。これらの各要素宣言は、前に説明した単純なタイプ名を持つ要素のデータ型を指定します。
- サンプル XML スキーマの中ごろの部分は、結果セットの *xsd:complexType* であり、前に定義された row タイプを持つ row 要素のシーケンスとして宣言されます。
- 最後に、サンプル XML スキーマの最後の行は、結果セット・ドキュメントの root 要素を宣言します。

オプション：*columnstyle=element*

columnstyle=element に対して生成される XML スキーマのフォーマットは、rowtype 宣言の XML 要素としてカラムを指定します。次に例を示します。

```
select forxmlschemaj("select 1,2", "columnstyle=element")
-----------------------
<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/2001/XMLSchema"
   schemaLocation="http://www.iso-standards.org/mra/9075/sqlx.xsd" />

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

オプション：*columnstyle=attribute*

columnstyle=attribute に対して生成される XML スキーマのフォーマットは、*columnstyle=element* の XML スキーマと同様です。違いは、カラムが rowtype 宣言の XML 属性として指定されることだけです。次に例を示します。

```
select forxmlschemaj("select 1,2", "columnstyle=attribute")
-----------------------
<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/2001/XMLSchema"
   schemaLocation="http://www.iso-standards.org/mra/9075/sqlx.xsd" />

   <xsd:complexType name="RowType.resultset">

      <xsd:attribute name="C1" type="INTEGER" use="required"/>
      <xsd:attribute name="C2" type="INTEGER" use="required"/>

  </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="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>
```

オプション：nullstyle=omit

nullstyle=omit に対して生成される XML スキーマのフォーマットは、それぞれの null 入力可能なカラム宣言の *minOccurs="0"* 属性と *maxOccurs="1"* 属性を指定します。次に例を示します。

```
select forxmlschemaj("select 1,null", "nullstyle=omit")
-----------------------
<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/2001/XMLSchema"
 schemaLocation="http://www.iso-standards.org/mra/9075/sqlx.xsd" />

   <xsd:complexType name="RowType.resultset">
      <xsd:sequence>
         <xsd:element name="C1" type="INTEGER" />
         <xsd:element name="C2" type="INTEGER"
             minOccurs="0" maxOccurs="1"/>
      </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="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>
```

オプション：*nullstyle=attribute*

nullstyle=attribute に対して生成される XML スキーマのフォーマットは、それぞれの null 入力可能なカラム宣言の *nullable="true"* 属性を指定します。次に例を示します。

```
select forxmlschemaj("select 1,null", "nullstyle=attribute"
-----------------------
<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/2001/XMLSchema"
   schemaLocation="http://www.iso-standards.org/mra/9075/sqlx.xsd" />

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

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。

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」(110 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
-----------------------------------------------------
0x000a003c00720065007300750006c00740073006500740020007800 6d...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」(112 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」(112 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。

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 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任意の固定数値データ型を使用できます。int、tinyint、bigint、numeric、または decimal。numeric と decimal の位取りは 0 でなければなりません。ordinality 列が real または float であってはなりません。

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。
  - 予約されているキーワード：for、option
  - 予約されていないキーワード：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 F 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。

次のスクリプトは、「forxmlj、forxmldtdj、forxmlschemaj、forxmlallj」(65 ページ) で説明されている forxmlj 関数を使用して publishers テーブルと titles テーブルの XML 表現を生成します。

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 authers 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"、"authors") 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、Java の forxmlj 関数を呼び出します。

```
insert into sample_docs (name_doc, text_doc)
    values ('publishers',
    forxmlj('select * from pubs2..publishers',
            'tablename=publishers'))

insert into sample_docs (name_doc, text_doc)
    values ('authors',
    forxmlj('select title_id, title
             type, pub_id, price,
             advance, total_sales
             from pubs2..authors',
            'tablename=authors')
```

注意 このスクリプトは、forxmlj 関数を使用します。この関数は Java ベースの関数であり、使用するにはインストールする必要があります。この関数のインストール方法については、「付録 C XML サービスの設定」を参照してください。

publishers テーブルの表現

次のサンプル・コードは、「sample_docs テーブル」(135 ページ) のスクリプトで生成された、pubs2 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)
```

ユーザがヘッダを保持する場合、xmlextract() の代わりに xmlvalidate() を使用してください。

```
insert into xmldoctab select filename ,xmlvalidate(content)
from xmlxfsTab
```

付 録　C

XML サービスの設定

この付録では、統合 XML プロセッサと Java ベース・プロセッサの設定方法について説明します。

Java ベースの SQLX マッピング関数のインストール

「第 4 章 XML マッピング関数」の関数は Java ベースであるため、サーバに関数をインストールしないと使用できません。この項では、Java ベースの関数をインストールする方法について説明します。

Java ベースの XML 関数

次の関数は、サーバにインストールした後で使用できます。

- forxmlj
- forxmldtdj
- forxmlschemaj
- forxmlallj
- forsqlcreatej
- forsqlinsertj
- forsqlscriptj

これらの機能をインストールするためのガイドラインと設定スクリプトは、ソース・コードと JavaDoc とともに次のディレクトリにあります。

$SYBASE/$SYBASE_ASE/sample

マッピング関数のインストール

Java ベースの SQLX マッピング関数をインストールするには、この項で説明する手順に従います。

環境変数

表 C-1 の環境変数はすでにサーバ・ユーティリティにあります。

表 *C-1:* 環境変数

変数	値
$ISERVER	isql ユーティリティと installjava ユーティリティの "-S" パラメータ
$INTERFACES	isql ユーティリティと installjava ユーティリティの "-I" パラメータ
$DB	isql ユーティリティと installjava ユーティリティの "-D" パラメータ

パーサのインストール

Java ベースの XML パーサをインストールするには、*$SYBASE/$SYBASE_ASE/sample* ディレクトリで参照される *setup* ディレクトリで make install-xerces コマンドを実行するか、次のようなクライアント・ユーティリティ・コマンドを使用します。

```
installjava -f $SYBASE/$SYBASE_ASE/lib/xerces.jar¥
    -j "xerces_jar"¥
    -D $DB  -S $ISERVER -I $INTERFACES¥
     -update  -Usa -P""
```

注意 forsqlcreatej、forsqlinsertj、forsqlscriptj には Java ベースの XML パーサが必要です。forxmlj、forxmldtdj、forxmlschemaj、forxmlallj には必要ありません。

マッピング関数のインストール

Java ベースの SQLX マッピング・クラスをインストールするには、*$SYBASE/$SYBASE_ASE/sample* ディレクトリで参照される *setup* ディレクトリで make install-sqlx コマンドを実行するか、次のようなクライアント・ユーティリティ・コマンドを使用します。

```
installjava
     -f../SQLX-examples/sqlx.jar -j"sqlx_jar"¥
     -D $DB -S $ISERVER -I $INTERFACES
         -update -Usa -P"
```

エイリアス名の作成

SQLX マッピング・クラスの Java メソッド用 SQL エイリアス名を作成するには、*$SYBASE/$SYBASE_ASE/sample* で参照される *setup* ディレクトリで make sqlx-aliases コマンドを実行するか、次のようなサーバ SQL コマンドを使用します。

```
create procedure forxmlallj
  (queryparmparm java.lang.String, optionparm
         java.lang.String,
     out rsout java.lang.String,
     out schemaout java.lang.String,
     out dtdout java.lang.String )
 language java parameter style java
external name "jcs.sqlx.ForXml.forXmlAll"

create function forxmlj
  (queryparm java.lang.String, optionparm
         java.lang.String)
  returns java.lang.String
  language java parameter style java
  external name "jcs.sqlx.ForXml.forXml"

create function forxmlschemaj
  (queryparm java.lang.String,optionparm
         java.lang.String)
  returns java.lang.String
  language java parameter style java
  external name "jcs.sqlx.ForXml.forXmlSchema"

create function forxmldtdj
  (queryparm java.lang.String, optionparm
        java.lang.String)
  returns java.lang.String
  language java parameter style java
  external name "jcs.sqlx.ForXml.forXmlDTD"

create function forsqlcreatej
  (schemax java.lang.String, optionparm
        java.lang.String)
  returns java.lang.String
  language java parameter style java
  external name "jcs.sqlx.SqlxCommand.forSqlCreate"

create function forsqlinsertj
  (inDoc java.lang.String, optionparm java.lang.String)
  returns java.lang.String
  language java parameter style java
  external name "jcs.sqlx.SqlxCommand.forSqlInsert"

create function forsqlscriptj
 (schemax java.lang.String, inDoc java.lang.String,
```

```
optionparm java.lang.String)
returns java.lang.String
language java parameter style java
external name "jcs.sqlx.SqlxCommand.forSqlScript"
```

付 録　D

Java ベースの XQL プロセッサ

この章では、XQL 言語を使って Adaptive Server からロー・データを選択し、それを XML ドキュメントとして表示する方法について説明します。

XML サービスは Java ベースの XQL プロセッサを提供します。Java ベースの XQL プロセッサは XPath の拡張機能である XQL 言語を実装しています。

このプロセッサは、XPath ベースの XML クエリ機能の予備として実装されています。その機能はネイティブ XML プロセッサの機能に取って替わられています。

Java ベースの XQL プロセッサは、サーバにインストールすることも、サーバの外部で実行することもできます。サーバの外部での実行は、コマンド・ラインでの Java プログラムの実行に似ています。

この付録では、まず、Java ベースの XQL プロセッサをスタンドアロン・プログラムとして Adaptive Server の外部で実行する方法について説明します。次に、Adaptive Server の内部で実行する方法について説明します。

Java ベースの XQL プロセッサの設定

Java ベースの XQL プロセッサを使用する前に、CLASSPATH 変数を設定し、プロセッサをインストールし、メモリ要件を設定する必要があります。

CLASSPATH 環境変数の設定

Adaptive Server 外部でスタンドアロン・プログラムを作成するには、*xerces.jar* のディレクトリと *xml.zip* のディレクトリを CLASSPATH 環境変数に含める必要があります。UNIX の場合は、次のように入力します。

```
setenv CLASSPATH $SYBASE/$SYBASE_ASE/lib/xerces.jar
$SYBASE/_SYBASE_ASE/lib/xml.zip
```

Windows の場合は、次のように入力します。

```
set CLASSPATH = D:¥%SYBASE%¥_SYBASE_ASE¥lib¥xerces.jar
D:¥%SYBASE%¥%SYBASE_ASE%¥lib¥xml.zip
```

Adaptive Server への Java ベースの XQL プロセッサのインストール

この項では、Adaptive Server で Java が使用できる状態になっていることを前提としています。Java を有効にする方法については、『Adaptive Server® Enterprise における Java』を参照してください。

installjava を使って Adaptive Server に JAR ファイルをコピーし、その JAR ファイルに含まれている Java クラスを現在のデータベースで使用できるようにします。構文は次のとおりです。

```
installjava
 -f file_name
 [-new | -update ]
 ...
```

各パラメータの意味は、次のとおりです。

- *file_name* は、サーバにインストールする JAR ファイルの名前です。
- new は、これが新規ファイルであることをサーバに知らせます。
- update は、既存の JAR ファイルを更新していることをサーバに知らせます。

installjava の詳細については、『ユーティリティ・ガイド』を参照してください。

Adaptive Server で XML へのサポートを追加するには、*xml.zip* ファイルと *xerces.jar* ファイルをインストールする必要があります。これらのファイルは、*$SYBASE/_SYBASE_ASE/lib/xml.zip* ディレクトリと *$SYBASE/_SYBASE_ASE./lib/xerxes.jar* ディレクトリにあります。

xml.zip をインストールするには、次のように入力します。

```
installjava -Usa -P -Sserver_name -f $SYBASE/_SYBASE_ASE/lib/xml.zip
```

xerces.jar をインストールするには、次のように入力します。

```
installjava -Usa -P -Sserver_name -f $SYBASE/_SYBASE_ASE/lib/xerces.jar
```

注意 *xerces.jar* をデータベースにインストールする場合は、tempdb のサイズを 10MB 増やしてください。

Adaptive Server 内部で Java ベースの XQL プロセッサを実行する場合のメモリ要件

Java ベースの XQL プロセッサで参照する XML データのサイズによっては、メモリを増やすことが必要な場合があります。サイズが 2K の一般的な XML ドキュメントの場合、Java サービスの設定パラメータを表 D-1 に示す値に設定することをおすすめします。設定パラメータの詳細については、『Sybase Adaptive Server システム管理ガイド』を参照してください。

表 *D-1: Java* サービスのメモリ・パラメータ

セクション	リセット値
enable java	1
size of process object heap	5000
size of shared class heap	5000
size of global fixed heap	5000

Java ベースの XQL プロセッサの使用

XML のロー・ドキュメントから解析済みドキュメントへの変換

XML のロー text ドキュメントまたはロー image ドキュメントを変換して解析し、その結果を格納するには、parse() メソッドを使用します。alter table コマンドを使用して、XML のロー・ドキュメントを変換します。次に例を示します。

```
alter table XMLTEXT add xmldoc IMAGE null
update XMLTEXT
set xmldoc = com.sybase.xml.xql.Xql.parse(xmlcol)
```

この例では、XMLTEXT テーブルの xmlcol カラムを解析済みデータに変換し、それを xmldoc カラムに格納します。

XML ドキュメントの挿入

XML ドキュメントを挿入するには、parse() メソッドを使用します。このメソッドでは、XML ドキュメントを引数として取得して sybase.aseutils.SybXmlStream を返します。

Adaptive Server では、image データまたは text データと InputStream の間に暗黙的なマッピングがあります。image カラムまたは text カラムを、キャスト処理せずに parse() に渡すことができます。parse() UDF は、ドキュメントを解析して sybase.ase.SybXmlStream を返します。Adaptive Server では、それを使用してデータを image カラムに書き込みます。このデータの書き込み先は image カラムのみであり、text カラムには書き込まれません。次に示す insert 文では、XMLDAT は image カラム xmldoc のあるテーブルを指します。

```
insert XMLDAT
values (...,
com.sybase.xml.xql.Xql.parse("<xmldoc></xmldoc>"),
    ...)
```

XML ドキュメントの更新

ドキュメントを更新するには、元のデータを削除して新しいデータを挿入します。ドキュメントまたはドキュメントの一部を更新する回数は、読み込み回数に比べて多くありません。更新は次のような文になります。

```
update XMLDAT
set xmldoc =
com.sybase.xml.xql.Xql.parse("<xmldoc></xmldoc>")
```

XML ドキュメントの削除

XML ドキュメントの削除は、text カラムの削除と似ています。たとえば、XMLDAT というテーブルを削除するには、次のように入力します。

```
delete XMLDAT
```

XQL の使用

XQL (XML 問い合わせ言語) は、XML の汎用問い合わせ言語として設計されています。XQL は、パスベースの問い合わせ言語です。XML ドキュメントの要素とテキストの指定やフィルタに使用します。また、XPath の自然な拡張機能でもあります。XQL では、簡潔で理解しやすい表記を使用して、特定の要素を指定したり、特定の特性を持ったノードを検索したりします。XQL のナビゲーションは、XML ツリー内の要素間で行われます。

最も一般的な XQL 演算子を以下に示します。

- 子演算子 (*/*) － 階層を示します。次の例では、<*bookstore*> 要素の子である <*book*> 要素が、xmlimage テーブルの xmlcol カラムから返されます。

  ```
  select
  com.sybase.xml.xql.Xql.query("/bookstore/book",
  xmlcol)
  from xmlimage
  ```

- 子孫演算子 (*//*) － 間に存在するすべてのレベルがクエリによって検索されることを示します。子孫演算子を使用した検索では、要素のオカレンスが XML 構造のどのレベルにあっても検出されます。次のクエリでは、<*excerpt*> 要素にある <*emph*> 要素のすべてのインスタンスを検出します。

```
select com.sybase.xml.xql.Xql.query
   ("/bookstore/book/excerpt//emph",xmlcol)
from xmlimage

<xql_result>
        <emph>I</emph>
</xql_result>
```

- 等号演算子 (=) － 要素の内容または属性の値を指定します。次のクエリでは、"last-name = Bob" である例をすべて検出します。

```
select com.sybase.xml.xql.Xql.query
   ("/bookstore/book/author[last-name='Bob']", xmlcol)
from xmlimage

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

- フィルタ演算子 ([ ]) － 左側にあるノードのセットを、角カッコ ([ ]) 内の条件に基づいてフィルタします。次の例では、book 要素にリストされている名前 (first-name) が Mary の作家 (author) のオカレンスを検出します。

```
select com.sybase.xml.xql.Xql.query
    ("/bookstore/book[author/first-name = 'Mary']", xmlcol)
from xmlimage
<xql_result>
        <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>
```

- サブスクリプト演算子 ([*index_ordinal*]) – ある要素の特定のインスタンスを検出します。次の例では、XML ドキュメント内にリストされている 2 番目の本 (book) を検索します。XQL で付けられる番号は 0 から始まることに注意してください。

```
select com.sybase.xml.xql.Xql.query("/bookstore/book[1]", xmlcol)
from xmlimage
Query  returned true and the  result is
<xql_result>
               <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>
</xql_result>
```

- ブール式 – フィルタ演算子内ではブール式を使用できます。たとえば、次のクエリでは、<*degree*> と <*award*> をそれぞれ 1 つ以上含む <*authors*> 要素がすべて返されます。

```
select com.sybase.xml.xql.Xql.query
("/bookstore/book/author[degree and award]", xmlcol)
from xmlimage

<xql_result>
        <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>
</xql_result>
```

パフォーマンスに影響するクエリ構造

この項では、Java ベースの XQL プロセッサのさまざまな使用例について説明します。

例

クエリに where 句を配置すると、処理に影響します。たとえば、次のクエリでは、名前 (first-name) が Mary である作家 (author) の本 (book) をすべて選択します。

```
select com.sybase.xml.xql.Xql.query
     ("/bookstore/book[author/first-name ='Mary']", xmlcol)
from XMLDAT
where
   com.sybase.xml.xql.Xql.query
    ("/bookstore/book
    [author/first-name= 'Mary']", xmlcol)!=
   convert(com.sybase.xml.xql.Xql, null)>>EmptyResult
-----------------------------------------------------
<xql_result ><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></xql_result>
```

Java ベースの XQL プロセッサのその他の使用法

注意　Sybase では、この項で説明する XQL パッケージの使用法をサポートしていません。これらの使用法には、JDK 1.2 以降が必要です。

スタンドアロン・アプリケーションの com.sybase.xml.xql.XqlDriver を使用すると、XML ドキュメントに対するクエリをコマンド・ラインから実行できます。

com.sybase.xml.xql.Xql で提供されている Java パッケージ・メソッドを使用すると、Java アプリケーションで XML ドキュメントに対してクエリを実行できます。また、この Java パッケージ・メソッドを、Adaptive Server での XML ドキュメントに対するクエリにも使用できます。この場合は、Java VM 機能を使用します。

com.sybase.xml.xql.XqlDriver は、ローカル・システムにファイルとして格納されている XML ドキュメントに対してのみ解析とクエリを実行できます。com.sybase.xml.xql.XqlDriver を使用して、データベースまたはネットワーク上にある XML ドキュメントに対して解析やクエリを実行することはできません。

com.sybase.xml.xql.XqlDriver は、XQL スクリプトを開発したり XQL を学習したりする場合に役立ちます。ただし、com.sybase.xml.xql.XqlDriver は、スタンドアロン・プログラムとしてのみ使用することをおすすめします。別の Java アプリケーションの一部としての使用はおすすめできません。com.sybase.xml.xql.XqlDriver に main() メソッドが含まれているからです。Java プログラムには、main() メソッドを 1 つしか含められません。main() メソッドが含まれている別の Java プログラムに com.sybase.xml.xql.XqlDriver を含めると、アプリケーションは両方の main() メソッドを実装しようとするため、Java でエラーが発生します。

アプリケーションでは、XML　クエリ・エンジンとのインタフェースに com.sybase.xml.xql.Xql クラスを使用することをおすすめします。このクラスのメソッドの詳細については、「com.sybase.xml.xql.Xql のメソッド」(168 ページ) を参照してください。

com.sybase.xml.xql.XqlDriver 構文

com.sybase.xml.xql.XqlDriver の構文は次のとおりです。

```
java com.sybase.xml.xql.XqlDriver
-qstring XQL_query
-validate true | false
-infile string
-outfile string
-help
-saxparser string
```

各パラメータの意味は、次のとおりです。

- qstring は、実行する XQL クエリを指定します。
- validate は、XML ドキュメントの妥当性を検査します。
- infile は、クエリを実行する XML ドキュメントです。
- outfile は、解析済み XML ドキュメントを格納するオペレーティング・システム・ファイルです。
- help は、com.sybase.xml.xql.XqlDriver 構文を表示します。
- saxparser は、SAX 2.0 に準拠する CLASSPATH パーサの名前を指定します。

クエリの例

次のクエリでは、*bookstore.xml* から本 (book) のタイトル (title) をすべて選択します。

```
java com.sybase.xml.xql.XqlDriver -qstring "/bookstore/book/title"
     -infile bookstore.xml

Query  returned true and the  result is

<xql_result>
<title>Seven Years in Trenton</title>
<title>History of Trenton</title>
<title>Trenton Today, Trenton Tomorrow</title>
</xql_result>
```

次の例では、*bookstore.xml* から作家 (author) の名前 (first-name) をすべてリストします。XQL は 0 から始まる番号付けシステムを使用しています。つまり、"0" はファイルにある要素の 1 番目のオカレンスを指定します。

```
java com.sybase.xml.xql.XqlDriver
     -qstring "/bookstore/book/author/first-name[0]"
     -infile bookstore.xml
Query  returned true and the  result is

<xql_result>
           <first-name>Joe</first-name>
           <first-name>Mary</first-name>
           <first-name>Toni</first-name>
</xql_result>
```

次の例では、*bookstore.xml* から姓 (last-name) が "Bob" である作家 (author) をすべてリストします。

```
java com.sybase.xml.xql.XqlDriver
     -qstring "/bookstore/book/author[last-name='Bob']"
     -infile bookstore.xml

Query  returned true and the  result is

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

ドキュメントの検証

valid オプションは、パーサを呼び出して、クエリ中の XML ドキュメントがその DTD に準拠していることを確認します。validate オプションを実行するには、スタンドアロン XML ドキュメントに有効な DTD がなければなりません。

たとえば、次のコマンドでは、*bookstore.xml* ドキュメントがその DTD に準拠していることを確認します。

```
java com.sybase.xml.xql.XqlDriver -qstring "/bookstore" -validate
     -infile bookstore.xml
```

スタンドアロン・アプリケーションに対する Java ベースの XQL プロセッサの使用

XQL を使用すると、XML データを処理するためのスタンドアロン・アプリケーション、JDBC クライアント、JavaBeans、EJB を開発できます。com.sybase.xml.xql.Xql の query() メソッドと parse() メソッドを使用すると、XML ドキュメントに対するクエリと解析を実行できます。スタンドアロン・アプリケーションが作成できれば、Adaptive Server が提供する結果セットに依存する必要はありません。代わりに、オペレーティング・システム・ファイルとして格納されている XML ドキュメントまたは Web 上に格納されている XML ドキュメントに対してクエリを実行できます。

スタンドアロン・アプリケーションの例

次の例では、FileInputStream() クエリを使用して *bookstore.xml* を読み込み、URI() メソッドを使用して書店 (bookstore) のすべての本に関する情報を含む Web ページ *bookstore.xml* を読み込みます。

```
String result;
FileInputStream XmlFile = new FileInputStream("bookstore.xml");
if ((result =
         Xql.query("/bookstore/book/author/first-name", XmlFile))
         != Xql.EmptyResult )
{
       System.out.println(result);
}else{
       System.out.println("Query returned false¥n");
```

```
}
URI _uri = new URI("http://mybookstore/bookstore.xml");
if ((result =
         Xql.query("/bookstore/book/author/first-name",uri.openStream()))
         != Xql.EmptyResult )
{
       System.out.println(result);
}else{
       System.out.println("Query returned false¥n");}
```

EJB の例

ユーザは EJB サーバ上でクエリ・エンジンとして動作する EJB コードの一部を記述できます。

以下のコード例には、*XmlBean* と呼ばれる EJB が含まれています。*XmlBean* には query() メソッドが含まれており、Web 上のあらゆる XML ドキュメントへのクエリが可能です。このコンポーネントでは、query() が XmlDoc オブジェクトを作成してからドキュメントに対してクエリを実行します。

リモート・インタフェースは次のようになります。

```
public interface XmlBean extends javax.ejb.EJBObject
{
      /**
        * XQL Method*/
     public String XQL(String query, URI location)
     throwsjava.rmi.RemoteException;}
```

Bean の実装は次のようになります。

```
public class XmlBean extends java.lang.Object implements
javax.ejb.SessionBean
{
      ....
      /***
      * XQL Method
      */
     public String XQL(String query, java.net.URI location) throws
         java.rmi.RemoteException
{
try {
      String result;
      if((result =
       Xql.query(query, location.openStream())) !=
       Xql.EmptyResult)
{
       return (result);
       }else{
return (null);
       }
       }catch(Exception e){
```

```
            throw new java.rmi.RemoteException(e.getMessage()));
                      }
....}
}
```

クライアント・コードは次のようになります。

```
....Context ctx = getInitialContext();
// make the instance of the class in Jaguar
XmlBeanHome -beanHome =
(XmlBeanHome)ctx.lookup(“XmlBean”);
_xmlBean = (XmlBean)_beanHome.create();
URI u = new URI("http://mywebsite/bookstore.xml");
String res= xmlBean.XQL("/bookstore/book/author/first-name",u);
```

com.sybase.xml.xql.Xql のメソッド

この項では、com.sybase.xml.xql.Xql 固有のメソッドについて説明します。

parse(String xmlDoc)

説明

Java 文字列を引数として受け取り、*SybXmlStream* を返します。これにより、XQL を使用してドキュメントにクエリを実行できます。

構文

parse(**String** *xml_document*)

各パラメータの意味は、次のとおりです。

- String は、Java 文字列です。
- *xml_document* は、文字列が含まれている XML ドキュメントです。

例

次の例では、*SybXmlStream* が返されます。

```
SybXmlStream xmlStream = Xql.parse("<xml>..</xml>")
```

使用法

パーサでは次の処理は行われません。

- DTD が提供されている場合のドキュメントの検証
- 外部 DTD の解析
- 外部リンク (XLinks など) の実行
- IDREF 間のナビゲーション

parse(InputStream xml_document, boolean validate)

説明　InputStream と boolean フラグを引数として受け取ります。フラグは、指定された DTD に従ってパーサがドキュメントを検証しなければならないことを示します。この例では、*SybXmlStream* が返されます。これにより、XQL を使用してドキュメントにクエリを実行できます。

構文　parse(**InputStream** *xml_document, boolean validate)*

各パラメータの意味は、次のとおりです。

- *InputStream* は、入力ストリームです。
- *xml_document* は、入力ストリームの発生元の XML ドキュメントです。

例　次の例では、*SybXmlStream* が返されます。

```
SybXmlStream is = Xql.parse(new
FileInputStream("file.xml"), true")
```

使用法

- フラグ内の true 値は、パーサが指定された DTD に従ってドキュメントを検証することを示します。
- フラグ内の false 値は、パーサが指定された DTD に従ってドキュメントを検証しないことを示します。
- パーサでは次の処理は行われません。
  - 外部 DTD の解析
  - 外部リンク (XLinks など) の実行
  - IDREF 間のナビゲーション

query(String query, String xmlDoc)

説明　XML ドキュメントにクエリを実行します。XML ドキュメントを入力引数として使用します。

構文　query(**String query**,*String xmlDoc*)

各パラメータの意味は、次のとおりです。

- *String query* は、検索対象の文字列です。
- *String xmldoc* は、クエリを実行する XML ドキュメントです。

例　次の例では、結果を Java 文字列として返します。

```
String result= Xql.query("/bookstore/book/author",
"<xml>...</xml>");
```

使用法　Java 文字列を返します。

query(String query, InputStream xmlDoc)

説明 入力ストリームを 2 番目の引数として使用して XML ドキュメントにクエリを実行します。

構文 query(**String query,*InputStream xmlDoc***)

各パラメータの意味は、次のとおりです。

- *String query* は、検索対象の文字列です。
- *Input Stream xmlDoc* は、クエリを実行する XML ドキュメントです。

例 次の例では、*bookstore.Xql* にリストされた作家 (author) について書店 (bookstore) にクエリを実行します。

```
FileInputStream xmlStream = new FileInputStream("doc.xml");
String result = Xql.query("/bookstore/book/author", xmlStream);
```

次の例では、URI を探索引数として使用し、Web 上の XML ドキュメントにクエリを実行します。

```
URI xmlURI = new URI("http://mywebsite/doc.xml");
String result = Xql.query("/bookstore/book/author", xmlURI.openStream());
```

使用法 Java 文字列を返します。

query(String query, SybXmlStream xmlDoc)

説明 解析済み XML ドキュメントを 2 番目の引数として使用し、XML ドキュメントにクエリを実行します。

構文 query(String query, *SybXmlStream*)

各パラメータの意味は、次のとおりです。

- *String query* は、検索対象の文字列です。
- *xmldoc* は、クエリを実行する解析済み XML ドキュメントです。

例 次の例では、*bookstore.Xml* にリストされた作家 (author) について書店 (bookstore) にクエリを実行します。

```
SybXmlStream xmlStream = Xql.parse("<xml>..</xml>");
String result = Xql.query("/bookstore/book/author",xmlStream);
```

sybase.aseutils.SybXmlStream

説明　クエリ時に InputStream が解析済み XML データにアクセスするために必要なインタフェースを定義します。

構文　sybase.aseutils.SybXmlStream interface

com.sybase.xml.xql.store.SybMemXmlStream

説明　メイン・メモリに解析済み XML ドキュメントを保持します。Sybase によって提供された SybXMLStream の実装です。

構文　com.sybase.xml.xql.store.SybMemXmlStream

使用法　parse() メソッドは、XML ドキュメントを解析したあと SybMemXmlStream のインスタンスを返します。

com.sybase.xml.xql.store.SybFileXmlStream

説明　解析済み XML ドキュメントを格納したファイルにクエリを実行できます。

構文　com.sybase.xml.xql.store.SybFileXmlStream {*file_name*}

file_name は、解析済み XML ドキュメントの格納先のファイルの名前です。

例　次の例では、RandomAccessFile のメンバがファイルを読み込み、データ・ストリームを配置します。

```
SybXmlStream xis = Xql.parse("<xml>..</xml>");
FileOutputStream ofs = new FileOutputStream("xml.data");
((SybMemXmlStream)xis).writeToFile(ofs);

SybXmlStream is = new SybFileXmlStream("xml.data");
String result = Xql.query("/bookstore/book/author", is);
```

setParser(String parserName)

説明　この静的メソッドでは、parse メソッドが使用するパーサを指定します。指定したパーサ・クラスが CLASSPATH から使用可能であり、SAX 2.0 に準拠していることを確認してください。

構文　setParser *(String parserName)*

string には、parser クラスの名前を指定します。

例

```
Xql.setParser("com.yourcompany.parser")
```

resetParser

説明 この静的メソッドでは、パーサを Sybase から提供されているデフォルト・パーサ (*xerces.jar*) にリセットします。

構文 resetParser

例 次の例では、パーサを Sybase デフォルト・パーサにリセットします。

```
xql.resetParser()
```

付 録 E

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 F

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

数字

A

B

C

D

E

F

H

I

J

M

N

O

P

Q

R

S

T

U

V

W

X

あ

い

え

お

か

き

く

け

こ

さ

し

す

せ

そ

た

ち

て

と

な

ね

は

ひ

ふ

へ

ほ

ま

む

め

も

ゆ

よ

れ

ろ