helion.pl
KSIĘGARNIA INTERNETOWA · WYDAWNICTWO

IDŹ DO
PRZYKŁADOWY ROZDZIAŁ
SPIS TREŚCI

KATALOG KSIĄŻEK
KATALOG ONLINE
ZAMÓW DRUKOWANY KATALOG

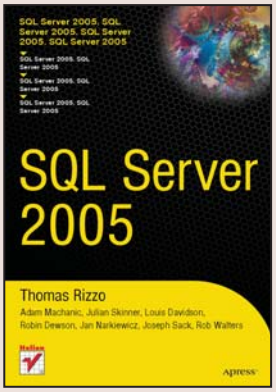
TWÓJ KOSZYK
DODAJ DO KOSZYKA

CENNIK I INFORMACJE
ZAMÓW INFORMACJE O NOWOŚCIACH
ZAMÓW CENNIK

CZYTELNIA
FRAGMENTY KSIĄŻEK ONLINE

SQL Server 2005

Autorzy: Thomas Rizzo, Adam Machanic, Robin Dewson, Rob Walters, Joseph Sack, Julian Skin
Tłumaczenie: Daniel Kaczmarek, Daniel Lehun
ISBN: 978-83-246-0555-2
Tytuł oryginału: Pro SQL Server 2005 (Pro)
Format: B5, stron: 816



Praktyczny przewodnik po SQL Server 2005 dla programistów i administratorów baz danych

- Jakie nowe funkcje oferuje SQL Server 2005?
- Jak zintegrować system bazodanowy z .NET?
- Jak wykorzystać możliwości SQL Server 2005 w otoczeniu biznesowym?

Każda nowa wersja SQL Server zawiera coraz więcej funkcji i udostępnia nowe możliwości. Tak też jest w przypadku SQL Server 2005, który ma nie tylko ulepszony system zarządzania bazami danych, ale również zabezpieczania ich. System ten oferuje rozszerzenia języka T-SQL przydatne zarówno programistom, jak i administratorom, umożliwia integrację baz z platformą .NET, a także posiada znacznie bardziej rozbudowane funkcje do manipulowania danymi XML. SQL Server 2005 wyposażony został też w wiele innych mechanizmów pozwalających zwiększyć produktywność w pracy z bazami danych i zapewnić ich lepsze funkcjonowanie.

Książka „SQL Server 2005" to przegląd usprawnień najnowszej wersji tego systemu bazodanowego napisany przez profesjonalistów dla profesjonalistów. Dowiesz się z niej, jak w praktyce wykorzystać możliwości SQL Server 2005. Nauczysz się tworzyć rozbudowane raporty przy użyciu Raporting Services, przeprowadzać analizy danych za pomocą Analysis Services, automatyzować zadania, dodawać do baz danych obiekty platformy .NET oraz stosować wygodne funkcje do obsługi danych w formacie XML.

- Zarządzanie SQL Server 2005 za pomocą SSMS
- Korzystanie z rozszerzeń języka T-SQL
- Integrowanie baz danych z .NET
- Zastosowania formatu XML w SQL Server 2005
- Raportowanie przy użyciu Raporting Services
- Analizowanie danych za pomocą Analysis Services
- Zabezpieczanie baz danych
- Kolejkowanie komunikatów przy użyciu Service Broker
- Obsługa powiadomień za pomocą Notification Services
- Automatyzowanie zadań

Wydawnictwo Helion
ul. Kościuszki 1c
44-100 Gliwice
tel. 032 230 98 63
e-mail: helion@helion.pl

Spis treści

O autorach 15

O redaktorach technicznych 19

Wprowadzenie 21

Rozdział 1. Przegląd funkcji i instalacja SQL Servera 27

Historia rozwoju SQL Servera 27
Przegląd funkcji SQL Servera 2005 28
Wydania 30
Funkcje 31
Instalacja 35
Minimalne wymagania systemowe 35
Proces instalacji 37
Rejestracja serwera 47
Przykładowe bazy danych 49
Instalacja dwóch równoległych instancji 50
Uaktualnianie baz danych SQL Servera 2000 i 7.0 51
Podsumowanie 51

Rozdział 2. Technologie zarządzania SQL Serverem 53

Łączenie się z SQL Serverem i zarządzanie nim 54
Menu kontekstowe dla każdego typu obiektów 54
Obsługa mobilnych baz danych 54
Obsługa SQL Servera 2000 56
Nowe funkcje interfejsu użytkownika 57
Asynchroniczny widok drzewa i filtrowanie obiektów 57
Niemodalne okna dialogowe o zmiennym rozmiarze 58
Nowe funkcje generowania skryptów i harmonogramów zadań 58
Autoryzacja kodu 60
Panel wyników 62
Activity Monitor 62
Widoki Summary 63
Rozszerzenia funkcjonalne 63
Dedykowane połączenie administracyjne 64
Wizualizacja zakleszczeń 65
Korelacja monitora Performance Monitor 65
Import i eksport danych dotyczących zarejestrowanych serwerów 68
Maintenance Plan Designer 68

Zmiany w agencie SQL Server Agent ... 69
Database Mail ... 69
Widoki katalogowe oraz dynamiczne widoki zarządcze ... 71
Domyślny ślad serwera ... 73
Rozszerzenia narzędzia Profiler ... 74
SQL Configuration Manager ... 75
Surface Area Configurator ... 76
Rozszerzona pomoc oraz integracja ze społecznością użytkowników SQL Servera ... 77
SQLCMD ... 78
Server Management Objects ... 80
Podsumowanie ... 84

Rozdział 3. Rozszerzenia T-SQL dla programistów ... 85
Rozszerzenia mające wpływ na DML ... 86
Unieważnienie przestarzałego stylu złączeń zewnętrznych ... 87
Wspólne wyrażenia tabelowe ... 88
Operator TOP ... 97
Rozszerzenia klauzuli FROM ... 101
OUTPUT ... 111
Funkcje rankingowe ... 112
EXCEPT i INTERSECT ... 119
Synonimy ... 122
Ogólne programowanie ... 123
Obsługa błędów ... 124
Rozszerzenie .WRITE instrukcji UPDATE ... 132
EXECUTE ... 134
Kontekst zabezpieczeń kodu ... 134
Deklaracje .NET ... 138
Podsumowanie ... 139

Rozdział 4. Rozszerzenia T-SQL dla administratorów baz danych ... 141
Widoki metadanych ... 141
Widoki zgodności ... 142
Widoki katalogowe ... 143
Dynamiczne widoki i funkcje zarządcze ... 145
Procedury wyzwalane DDL ... 146
Tworzenie i zmienianie procedur wyzwalanych DDL ... 147
Usuwanie procedur wyzwalanych DDL ... 149
Udostępnianie i blokowanie procedur wyzwalanych DDL ... 149
Odczytywanie procedur wyzwalanych DDL w widokach katalogowych ... 149
Programowanie procedur wyzwalanych DDL przy użyciu funkcji eventdata() ... 150
Rozszerzenia w zakresie indeksowania i wydajności ... 152
Indeksowanie online ... 153
Kontrola blokowania w trakcie tworzenia indeksów ... 154
Tworzenie indeksów z dołączonymi dodatkowymi kolumnami ... 155
Modyfikowanie indeksów ... 156
Używanie statystyk dla skorelowanych kolumn DATETIME ... 160
Zwiększanie wydajności sortowania dla trzeciorzędnych ustawień sortowania ... 161
Partycjonowanie tabel i indeksów ... 163
Funkcje partycjonowania ... 164
Schematy partycjonowania ... 165
Tworzenie partycjonowanych tabel i indeksów ... 165
Dodawanie i usuwanie partycji ... 166
Modyfikowanie funkcji i schematów partycjonowania ... 168

Dołączanie i usuwanie tabel z partycji ... 169
Zarządzanie partycjami tabel i indeksów ... 170
Rozszerzenia tabel i widoków ... 171
Rozszerzenia w widokach indeksowanych ... 171
Stałe kolumny obliczane ... 172
Migawki ... 173
Poziom izolacji transakcji SNAPSHOT ... 173
Migawki baz danych ... 178
Rozszerzenia w zakresie integralności danych ... 180
Weryfikacja stron bazy danych ... 180
Przełączanie bazy danych w tryb awaryjny ... 181
Podsumowanie ... 182

Rozdział 5. Integracja z .NET ... 183
Podstawowe informacje na temat integracji SQL Servera i .NET ... 184
Dlaczego SQL Server 2005 obsługuje CLR? ... 184
Kiedy powinno się używać procedur CLR? ... 185
Kiedy nie powinno się używać procedur CLR? ... 186
Sposób, w jaki SQL Server obsługuje .NET — architektura rozwiązania ... 186
Model programistyczny .NET w SQL Serverze ... 187
Modyfikacje ADO.NET związane z obsługą SQL Servera ... 188
Przegląd nowych przestrzeni nazw .NET dla SQL Servera ... 188
Implementacja procedury składowanej ... 189
Tworzenie projektu SQL Servera Project w Visual Studio 2005 ... 190
Budowa procedury składowanej ... 194
Dodawanie parametrów ... 195
Definicja problemu ... 196
Użycie obiektu SqlPipe ... 198
Kompletne rozwiązanie — implementacja ciała procedury składowanej ... 201
Testowanie procedury składowanej ... 203
Debugowanie procedury ... 205
Zgłaszanie wyjątków w procedurach CLR ... 206
Wdrażanie procedur CLR ... 210
Podsumowanie ... 212

Rozdział 6. Programowanie podzespołów ... 213
Typy CLR zdefiniowane przez użytkownika ... 214
Zastosowania typów definiowanych przez użytkownika ... 215
Dodawanie typu zdefiniowanego przez użytkownika do projektu SQL Servera ... 215
Elementy typu zdefiniowanego przez użytkownika ... 217
Prosty przykład: typ PhoneNumber ... 221
Kolejny przykład: typ StringArray ... 229
Zarządzanie typami zdefiniowanymi przez użytkownika ... 238
Funkcje CLR zdefiniowane przez użytkownika ... 238
Dodawanie funkcji zdefiniowanej przez użytkownika do projektu Visual Studio ... 239
Szablon funkcji zdefiniowanej przez użytkownika w Visual Studio 2005 ... 240
Atrybut SqlFunction ... 241
Skalarne funkcje zdefiniowane przez użytkownika ... 242
Definiowanie funkcji tabelowych zdefiniowanych przez użytkownika ... 245
Zarządzanie funkcjami CLR zdefiniowanymi przez użytkownika ... 250
Agregaty CLR zdefiniowane przez użytkownika ... 251
Dodawanie agregatu definiowanego przez użytkownika do projektu SQL Servera ... 252
Elementy agregatu zdefiniowanego przez użytkownika ... 253

Procedury wyzwalane CLR zdefiniowane przez użytkownika ... 260
Dodawanie procedury wyzwalanej CLR zdefiniowanej przez użytkownika do projektu SQL Servera ... 261
Implementowanie procedur wyzwalanych CLR ... 261
Zarządzanie procedurami wyzwalanymi zdefiniowanymi przez użytkownika ... 265
Zarządzanie podzespołami ... 266
Uwaga dotycząca Visual Studio 2005 ... 266
Podsumowanie ... 267

Rozdział 7. SQL Server i XML ... 269
Czym jest XML? ... 269
Czym są XPath i XMLDOM? ... 271
Składnia XPath ... 272
Funkcje XPath ... 274
XMLDOM — XML Document Object Model ... 274
Klasy XPathDocument, XPathNavigator oraz XPathExpression ... 276
Wstawianie danych XML do bazy danych ... 277
Czym jest SQLXML? ... 278
Konfiguracja SQL Servera ... 279
OPENXML ... 279
SQLXML — widoki XML używające adnotowanych schematów XML ... 286
Diagramy uaktualniające SQLXML ... 291
Zbiorcze ładowanie danych XML ... 294
Odczytywanie danych XML z bazy danych — klauzula FOR XML ... 298
Klauzula FOR XML po stronie serwera ... 298
Klauzula FOR XML po stronie klienta ... 304
Używanie szablonów ... 304
Rozszerzenia klauzuli FOR XML ... 305
Programowanie SQLXML w .NET i COM ... 306
SqlXmlCommand ... 307
SqlXmlParameter ... 307
SqlXmlAdapter ... 307
SqlXmlException ... 308
Przykładowe kody ... 309
Klauzula FOR XML — przetwarzanie po stronie serwera i po stronie klienta ... 310
Sposób użycia obiektu XMLTextReader ... 311
Używanie parametrów w SQLXML ... 312
Wykonywanie zapytań XPath lub zapytań SQL z szablonami ... 313
Interakcja z zestawem danych ADO.NET ... 313
Programowanie diagramów uaktualniających ... 314
Podsumowanie ... 314

Rozdział 8. Obsługa XML-a i XQuery w SQL Serverze 2005 ... 315
Sposób użycia typu danych XML ... 316
Sposób przechowywania danych XML w SQL Serverze ... 318
Tworzenie kolumn XML ... 320
Ustawianie uprawnień do tworzenia schematów ... 325
Nakładanie ograniczeń na kolumny XML ... 326
Ograniczenia typu danych XML ... 327
Wstawianie danych w kolumnach XML ... 328
Użycie SSIS względem danych XML ... 328
Zbiorcze ładowanie danych XML ... 330
Tworzenie własnego zapytania lub aplikacji ... 330

Zapytania na danych XML ... 331
Podstawy języka XQuery ... 332
Podstawowe metody zapytań XML ... 338
Zapytania międzydomenowe ... 340
Modyfikowanie danych XML ... 341
Ograniczenia zakresu modyfikacji danych XML ... 342
Indeksowanie danych XML w celu zwiększenia wydajności ... 343
Sposób działania indeksów na danych XML ... 344
Drugorzędne indeksy XML ... 345
Przeszukiwanie pełnotekstowe a typ danych XML ... 346
Dynamiczne widoki zarządcze i dane XML ... 347
Aplikacje i dane XML ... 348
Obsługa usług sieciowych XML ... 349
Tworzenie węzła końcowego ... 350
Używanie zaawansowanych usług sieciowych ... 356
Monitorowanie wydajności usług sieciowych XML ... 361
Podsumowanie ... 362

Rozdział 9. SQL Server 2005 Reporting Services ... 363

Komponenty Reporting Services ... 364
Report Server ... 365
Katalog metadanych ... 366
Report Designer ... 366
Aplikacja WWW Report Manager ... 368
Bezpieczeństwo Reporting Services ... 370
Tworzenie prostego raportu w SSRS 2000 ... 370
Uruchamianie projektanta ... 371
Praca ze źródłami danych i zestawami danych ... 371
Definiowanie układu i podglądanie raportu ... 372
Praca z wyrażeniami ... 373
Wdrażanie raportu ... 373
Uaktualnianie SQL Server 2000 Reporting Services ... 373
Zmiany w licencjonowaniu Reporting Services ... 375
Zintegrowane SQL Server Management Studio ... 376
Przewodnik — Management Studio i Reporting Services ... 377
Zmiany w sposobie zarządzania ... 382
Dostawca WMI ... 383
Sieciowe usługi zarządzania i wykonawcze ... 386
Narzędzie konfiguracji Reporting Services ... 386
Usprawnienia w zakresie projektowania i wykonywania raportów ... 387
Edytor wyrażeń Expression Editor ... 388
Parametry wielowartościowe ... 388
Kontrolka wyboru daty ... 391
Sortowanie interaktywne ... 392
Integracja z Analysis Services ... 393
Przewodnik — tworzenie raportu w BIDS ... 394
Nagłówki pływające ... 402
Zmiany źródeł danych: wyrażenia, usługi sieciowe i XML, SSIS i SAP ... 403
Własne elementy raportów ... 409
Integracja z Visual Studio i kontrolki ReportViewer ... 409
Korzystanie z kontrolek WinForm ... 410
Oprogramowywanie kontrolek ReportViewer ... 413
Obiekty LocalReport oraz ServerReport ... 417

Integracja z SharePoint ... 417
Zapytania i raporty ad hoc użytkownika ... 418
Klient ReportBuilder ... 419
Semantic Model Definition Language ... 420
Przewodnik — ReportBuilder ... 422
Podsumowanie ... 428

Rozdział 10. Analysis Services ... 429

Rozszerzenia SSAS 2005 ... 430
Architektura ... 430
Wydajność, skalowalność, dostępność ... 432
Użyteczność ... 432
Projektowanie ... 433
Instalacja ... 435
Na czym polega analiza? ... 436
OLAP, OLTP i hurtownie danych ... 436
Terminologia wykorzystywana w systemach OLAP ... 437
Kostki ... 438
Komórki ... 438
Miary i tabele faktów ... 438
Wymiary i atrybuty ... 439
Hierarchie ... 439
Projekty Analysis Services ... 440
Definiowanie źródeł danych ... 442
Projektanty a kreatory ... 445
Definiowanie widoków źródeł danych ... 446
Definiowanie kostek ... 450
Wdrażanie projektów oraz konfigurowanie projektów do wdrożenia ... 454
Operacje na kostkach ... 456
Przeglądanie kostek ... 457
Przeglądanie kostek z hierarchiami ... 459
Zarządzanie wyświetlanymi danymi ... 462
Obliczenia i MDX ... 465
Kluczowe wskaźniki wydajności (KPI) ... 468
Analysis Services Scripting Language (ASSL) ... 472
Przykładowy obiekt ASSL — widok źródła danych z zapytaniem nazwanym ... 473
Analizowanie kodu ASSL ... 475
Podsumowanie ... 476

Rozdział 11. Bezpieczeństwo ... 477

Dwa zdania o użytkowniku sa ... 478
Surface Area Configuration ... 478
Połączenia zdalne ... 478
Dedykowane połączenie administracyjne ... 479
.NET Framework ... 479
Database Mail ... 480
SQLMail ... 480
Service Broker ... 481
Połączenia HTTP ... 481
Tworzenie lustrzanej kopii baz danych ... 481
Web Assistant ... 481
xp_cmdshell ... 482
Zdalne zapytania ad hoc ... 482
Rozszerzone procedury składowane automatyzacji OLE ... 482
Rozszerzone procedury składowane SMO i DMO ... 482

Obiekty zabezpieczające i obiekty zabezpieczane ... 483
Obiekty zabezpieczające ... 483
Obiekty zabezpieczane ... 493
Uprawnienia ... 496
Typy uprawnień ... 496
Zarządzanie uprawnieniami ... 498
Code Access Security ... 502
Imperatywna i deklaratywna implementacja CAS ... 502
Użycie CAS z SQL Serverem ... 503
Szyfrowanie ... 507
Hierarchia szyfrowania w SQL Serverze 2005 ... 508
Szyfrowanie hasłem podanym przez użytkownika ... 509
Szyfrowanie kluczem symetrycznym ... 510
Szyfrowanie kluczem asymetrycznym ... 511
Szyfrowanie certyfikatem ... 512
Certyfikaty i usługi sieciowe ... 513
Podsumowanie ... 516

Rozdział 12. Service Broker ... 517

Czym jest Service Broker? ... 518
Architektura technologii Service Broker ... 518
Scenariusze użycia narzędzia Service Broker ... 522
Tworzenie aplikacji Service Broker ... 523
Udostępnianie narzędzia Service Broker ... 524
Tworzenie typów komunikatów ... 524
Tworzenie kontraktów ... 524
Tworzenie kolejek ... 525
Tworzenie usług ... 526
Tworzenie procedur składowanych aplikacji Service Broker ... 526
Przykładowa aplikacja Service Broker ... 530
Wyznaczanie tras i bezpieczeństwo w technologii Service Broker ... 537
Tworzenie rozproszonych aplikacji Service Broker ... 537
Przykładowa rozproszona aplikacja Service Broker ... 541
Podsumowanie ... 562

Rozdział 13. Automatyzacja i monitorowanie ... 563

SQL Server Agent ... 564
Krok 1. Łączenie się z SQL Serverem ... 565
Krok 2. Tworzenie zadania agenta ... 567
Rozszerzenia bezpieczeństwa ... 571
Podsystemy agenta ... 580
Współużytkowanie harmonogramów zadań ... 581
Zapisywanie do tabeli sysjobstepslogs ... 584
Zdarzenia WMI oraz alerty agenta ... 585
Liczniki wydajności agenta ... 587
Uaktualnianie agenta ... 588
Plany utrzymania ... 590
Tworzenie planu utrzymania ... 591
Okno narzędziowe ... 592
Okno dokumentu narzędzia Maintenance Plan Designer ... 592
SQLCMD ... 598
Łączenie się z SQL Serverem ... 599
Przekazywanie zmiennych ... 600
Używanie dedykowanego połączenia administracyjnego ... 601
Tworzenie skryptów ... 601

Database mail ... 602
Informacje ogólne ... 603
Konfigurowanie Database Mail ... 605
Wysyłanie poczty ... 609
SQL Profiler ... 610
Korelacja Performance Monitora ... 614
ShowPlan ... 616
Wizualizacja zakleszczeń ... 617
Podsumowanie ... 618

Rozdział 14. Integration Services ... 621
Nowe funkcje w SSIS ... 623
Nowe środowisko IDE w SSIS ... 624
Łączenie się z SSIS w Management Studio ... 625
Tworzenie nowego projektu SSIS w BIDS ... 626
Podstawy działania SSIS ... 627
Obszar projektowania przepływu sterowania ... 627
Obszar projektowania przepływu danych ... 629
Obszar projektowania procedur obsługi zdarzeń ... 643
Przeglądarka pakietów Package Explorer ... 645
Menedżery połączeń ... 645
Przeglądarka rozwiązań Solution Explorer ... 646
Okno właściwości Properties ... 647
Zadania okna narzędziowego Control Flow ... 648
Kontenery ... 649
Zadania Analysis Services ... 653
Zadanie Data Flow ... 653
Zadania Execute Package ... 654
Zadanie Bulk Insert ... 654
Zadanie Execute SQL ... 654
Zadanie Execute Process ... 656
Zadanie File System ... 656
Zadanie File Transfer Protocol ... 656
Zadania Maintenance Plan ... 657
Zadanie Message Queue ... 657
Zadanie Send Mail ... 658
Zadania Scripting ... 658
Zadanie Web Service ... 661
Zadania WMI ... 662
Zadanie XML ... 662
Zadania projektanta przepływu danych Data Flow Designer ... 662
Adaptery źródeł ... 662
Adaptery obiektów docelowych ... 663
Transformacje ... 664
Zapisywanie zdarzeń w dziennikach ... 670
Konfiguracje ... 674
Sposób użycia narzędzia Package Configuration Organizer ... 675
Zmienne ... 678
Ograniczenia pierwszeństwa przepływu prac ... 681
Punkty kontrolne ... 682
Transakcje ... 683
Debugowanie ... 684
Wizualne debugowanie przepływu sterowania ... 684
Wizualne debugowanie przepływu danych ... 685

Przeglądarki danych ... 685
Okno punktów przerwań ... 686
Inne okna debugowania ... 687
Narzędzie SSIS Package Deployment Utility ... 687
Wykonywanie migracji pakietów SQL Servera 2000 ... 688
Tworzenie harmonogramów pakietu SSIS ... 688
Podsumowanie ... 689

Rozdział 15. Tworzenie lustrzanych kopii baz danych ... 691
Definicja wysokiej dostępności ... 693
Ogólne informacje na temat sporządzania lustrzanych kopii baz danych ... 694
Mechanizm tworzenia kopii lustrzanych a inne rozwiązania ... 697
Konfiguracja mechanizmu tworzenia lustrzanych kopii baz danych ... 698
Wymagania wstępne, połączenia i bezpieczeństwo ... 699
Tworzenie kopii bezpieczeństwa i odtwarzanie głównej bazy danych ... 703
Ustanawianie partnerstwa serwera głównego i serwera lustrzanego ... 704
Zmiana poziomów zabezpieczeń transakcji ... 705
Stany sporządzania kopii lustrzanych, sygnały taktujące i kworum ... 706
Inicjowanie przywracania działania ... 707
Zawieszanie i przywracanie działania mechanizmu sporządzania lustrzanych kopii baz danych ... 708
Wyłączanie mechanizmu tworzenia lustrzanych kopii baz danych ... 709
Indeksowanie pełnotekstowe a sporządzanie lustrzanych kopii baz danych ... 709
Service Broker a sporządzanie lustrzanych kopii baz danych ... 710
Konfigurowanie mechanizmu tworzenia lustrzanych kopii baz danych w Management Studio ... 710
Aplikacje klienckie i sporządzanie kopii lustrzanych ... 715
Monitorowanie procesu sporządzania kopii lustrzanych ... 717
Widoki katalogowe ... 717
Liczniki Performance Monitora ... 721
Profiler ... 721
Dziennik zdarzeń systemu Windows oraz dziennik błędów SQL Servera ... 722
Wydajność a sporządzanie lustrzanych kopii baz danych ... 723
Ograniczenia mechanizmu sporządzania kopii lustrzanych ... 724
Przykładowa aplikacja ... 725
Migawki baz danych a mechanizm sporządzania kopii lustrzanych ... 731
Ogólne informacje na temat migawek baz danych ... 732
Korzystanie z migawek w kodzie T-SQL ... 734
Wydajność systemu z migawkami na serwerach lustrzanych ... 735
Używanie i monitorowanie migawek baz danych oraz zarządzanie nimi ... 736
Programowanie z użyciem migawek baz danych ... 737
Ograniczenia migawek baz danych ... 737
Klastry Windows w SQL Serverze 2005 ... 738
Replikacja w SQL Serverze 2005 ... 739
Podsumowanie ... 740

Rozdział 16. Notification Services ... 741
Architektura Notification Services ... 742
Subskrybenci ... 743
Subskrypcje ... 744
Zdarzenia ... 744
Powiadomienia ... 745
Tworzenie aplikacji Notification Services ... 746
Definiowanie instancji NS — plik konfiguracji instancji ... 746
Definiowanie aplikacji NS — plik definicji aplikacji ... 751

Kompilowanie i uruchamianie aplikacji NS 771
Monitorowanie aplikacji NS i usuwanie z niej błędów 775
Korzystanie z NS w kodzie źródłowym 777
Programowanie NS w Visual Studio 777
Zarządzanie NS w kodzie źródłowym 782
Podsumowanie 782

Skorowidz 783

Rozdział 7.

SQL Server i XML

Popularność języka XML wzrasta z każdym dniem. Niektórzy zdeklarowani zwolennicy koncepcji relacyjnej kręcą z dezaprobatą głowami, gdy słyszą o XML-u. Jednak tak naprawdę XML stanowi uzupełnienie technologii relacyjnych, a dane XML są w większości ustrukturyzowane jak dane relacyjne. Czy to dobrze? Pewnie nie, ponieważ obydwa modele sprawdzają się najlepiej w przypadku danych, dla których zostały stworzone: XML dla danych na wpół ustrukturyzowanych, a relacyjne bazy danych dla danych relacyjnych. Ponadto semantyka mechanizmów przechowywania, odpytywania i modyfikowania danych XML często sprawia kłopot osobom preferującym podejście relacyjne. Jak się przekonamy, XQuery w niczym nie przypomina języka T-SQL, jednak obydwu technologii można używać łącznie, aby rozwiązywać problemy biznesowe. Zanim zagłębimy się w nowe mechanizmy obsługi XML-a dostępne w SQL Serverze 2005, musimy jednak najpierw zrozumieć, czym jest XML i w jaki sposób SQL Server już obsługuje technologię XML.

Na początku niniejszego rozdziału zajmiemy się rzeczami podstawowymi: czym jest XML i w jaki sposób aktualna wersja SQL Servera obsługuje XML za pomocą `FOR XML`, rozdrabniania danych XML i wsparcia `FOR XML` z poziomu .NET Framework. Opisane zostanie również XPath oraz zakres obsługi XPath świadczonej przez SQL Server. Na koniec przyjrzymy się SQLXML oraz pokażemy sposób, w jaki można użyć tej technologii do rozszerzenia zakresu obsługi XML-a przez SQL Server. Jeśli użytkownik nie posiada SQLXML, wersję 3.0 dla SQL Servera 2000 można pobrać za darmo z sieci WWW, natomiast wersja 4.0 jest dołączona do SQL Servera 2005. W następnym rozdziale zobaczymy, w jaki sposób SQL Server zaczął obsługiwać klauzulę `FOR XML` z nowym typem danych `XML` oraz językiem XQuery. Zajmiemy się także obsługą usług sieciowych WWW, która zastąpiła obsługę usług sieciowych ISAPI SQLXML.

Czym jest XML?

Informacja dla tych, którzy przez ostatnich dziesięć lat żyli odizolowani od świata zewnętrznego i nigdy nie zetknęli się z szumem towarzyszącym językowi XML: otóż XML to skrót od *eXtensible Markup Language*. XML pozwala na strukturyzowanie

danych przy użyciu standaryzowanych schematów. Standaryzacja jest właśnie najważniejszą cechą, ponieważ z niej wypływa siła XML-a. Każdy inny system potrafi odczytywać i zapisywać dane zgodne ze standardem, oczywiście uwzględniając fakt, że niektóre systemy mogą czasami odmiennie interpretować dane. XML również pozwala na pozyskiwanie danych XML przy użyciu standardu XPath i przekształcanie danych XML z zastosowaniem XSLT. Obydwa standardy zostaną opisane w dalszej części tego rozdziału.

Ciekawą kwestią jest dokonanie wyboru między danymi XML i danymi relacyjnymi. Co jakiś czas na nowo rozpoczyna się dyskusja na temat przechowywania danych w klasycznym standardzie relacyjnym, w hybrydowym modelu XML/relacyjnym oraz w czystym formacie XML. Naszym zdaniem najlepiej jest wybierać rozwiązanie, które najbardziej przydaje się do rozwiązania danego problemu. Użytkownicy specjalizujący się w technologiach relacyjnych, którzy potrafią zapewnić wysoką wydajność przetwarzania danych relacyjnych, nie mają powodów, by przestawić się na używanie rozwiązań XML-owych. Dane relacyjne można bez trudu udostępniać dla świata zewnętrznego w postaci XML przy użyciu `FOR XML`, natomiast wewnętrznie dane mogą nadal być przechowywane w modelu relacyjnym. Niewarto ulegać zachwytom nad XML-em bez wyraźnego powodu. Na przykład dane XML mają charakter tekstowy, a więc ich rozmiar jest większy niż rozmiar binarnych odpowiedników. XML jest również rozwlekły, ponieważ nie jest znormalizowany jak dane relacyjne, a przez to w jednym dokumencie mogą się powtarzać sekcje tych samych danych. Ponadto model programistyczny XML-a różni się od modelu programistycznego, do którego przyzwyczaili się programiści przetwarzający dane relacyjne.

Poznajmy najpierw podstawową terminologię XML, a potem zaczniemy poznawać coraz głębsze tajemnice technologii, aby móc ją jak najlepiej zrozumieć. Często można usłyszeć ludzi rozprawiających o dokumentach, elementach i atrybutach. Najlepiej jest przyjąć podejście, według którego cała struktura XML-a to dokument, dokument zawiera elementy, a elementy mogą zawierać atrybuty. Poniższy przykładowy dokument XML zawiera jeden dokument, trzy elementy i dwa atrybuty:

```
<?xml version="1.0"?>
<customer>
  <name id="10">Tom Rizzo</name>
  <state region="Northwest">WA</state>
</customer>
```

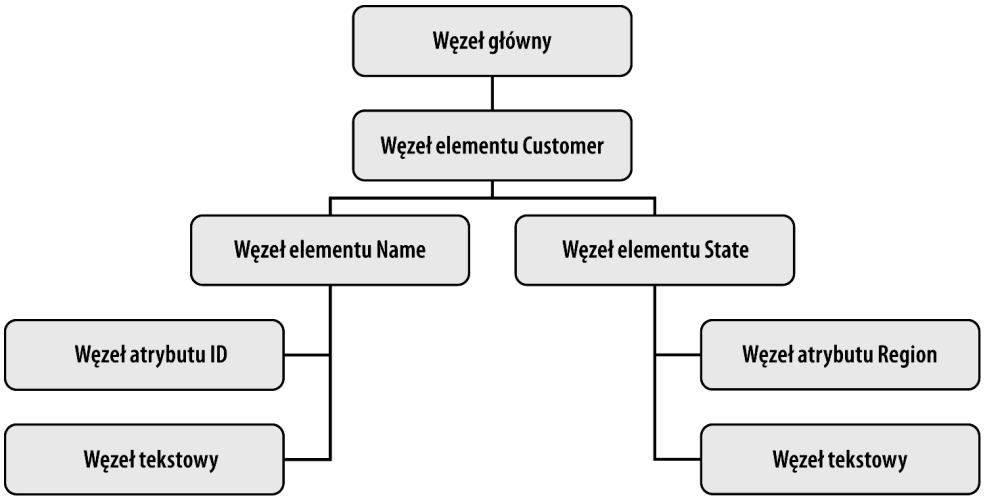
XML posiada schematy i przestrzenie nazw. Nie istnieje wymóg, by nakładać schematy na XML, lecz dzięki schematom i przestrzeniom nazw można w unikatowy sposób definiować poprawne i niepoprawne struktury danych w dokumencie XML. W świecie relacyjnym mamy do czynienia ze strukturami tabel oraz ograniczeniami. Niektóre rozwiązania relacyjne można odwzorowywać na schematy XML, ponieważ schematy XML posiadają typy danych oraz reguły wyznaczające kolejność, moc i inne aspekty dokumenty XML. Dzięki schematom można współużytkować własne dane XML z innymi użytkownikami i zapewnić, że inni użytkownicy nadal będą rozumieć te dane. Przestrzeń nazw XML to kolekcja nazw identyfikowana przez odwołanie URI używane w dokumencie XML przez typy elementów oraz nazwy atrybutów. Dzięki przestrzeniom nazw można używać tych samych nazw pochodzących z różnych źródeł i unikać konfliktów między nimi. Na przykład można użyć tego samego elementu

o nazwie `customer` pochodzącego z dwóch różnych źródeł, jeśli tylko podana zostanie przestrzeń nazw identyfikująca elementy jako należące do odmiennych przestrzeni nazw. Schematy i przestrzenie nazw będą szczególnie ważne, gdy będziemy poznawać nowy typ danych `XML` dostępny w SQL Serverze 2005 oraz omawiać natywny sposób przechowywania danych XML w bazie danych.

Czym są XPath i XMLDOM?

Gdy ma się do dyspozycji zbiór dokumentów XML, bez wątpienia istotną rzeczą jest możliwość wykonywania na tych zbiorach zapytań i odczytywania potrzebnych informacji. XML Path Language (XPath) to język zapytań umożliwiający definiowanie części dokumentu XML, które należy wybrać. XPath posiada parser, który interpretuje składnię, odczytuje dokument XML i wyciąga z niego odpowiednie części. Na przykład z dokumentu XML można odczytywać wszystkich klientów mieszkających w stanie Nowy Jork. W tym celu wystarczy napisać odpowiednią instrukcję XPath.

Język XML jest hierarchiczny, dzięki czemu w XPath można wskazywać ścieżkę lub ścieżki do danych XML, które trzeba pozyskać. Dane XML można traktować jak hierarchię węzłów. Węzłem głównym jest zazwyczaj element dokumentu XML. Następnie pod węzłem głównym tworzona jest struktura drzewiasta dla wszystkich danych XML. Gdyby przykładowy kod XML przedstawiony wcześniej odwzorować na hierarchię ścieżek XML, uzyskalibyśmy drzewo widoczne na rysunku 7.1.



Rysunek 7.1. *Przykładowe drzewo dokumentu XML*

Drzewo nie jest szczególnie imponujące, ale bez wątpienia jest to drzewo. Widać w nim wszystkie elementy, atrybuty i węzły tekstowe. Przy użyciu XPath można odczytywać siedem typów węzłów: węzeł główny, element, atrybut, przestrzeń nazw, instrukcję przetwarzania, komentarz i węzeł tekstowy. W instrukcjach XPath używa się najczęściej elementów, atrybutów, instrukcji przetwarzania oraz węzłów tekstowych.

XPath wykorzystuje się najczęściej do nawigowania wśród różnych węzłów za pomocą **osi XPath** (ang. *XPath axis*). Osie XPath opisują sposób nawigacji w strukturze przez wskazanie punktu początkowego oraz kierunku nawigacji. W XPath występuje 13 osi, lecz najczęściej używa się osi child (potomków) i attribute (atrybutów). W tabeli 7.1 opisano wszystkie 13 dostępnych osi.

Tabela 7.1. *Osie XPath*

Nazwa	Opis
Ancestor	Zawiera węzeł rodzica węzła kontekstu oraz wszystkie następne węzły rodziców tego węzła aż do węzła głównego.
Ancestor-or-self	Zawiera węzły przodków oraz sam węzeł kontekstu aż do węzła głównego.
Attribute	Zawiera atrybuty węzła kontekstu, jeśli węzeł kontekstu jest węzłem elementu.
Child	Zawiera węzły potomków węzła kontekstu.
Descendant	Zawiera węzły potomka, kolejnych potomków i tak dalej węzła kontekstu.
Descendent-or-self	Zawiera sam węzeł kontekstu oraz węzły wszystkich potomków węzła kontekstu.
Following	Zawiera wszystkie węzły w tym samym dokumencie co węzeł kontekstu, które pod względem kolejności w dokumencie występują po węźle kontekstu. Nie zawiera jednak żadnych węzłów potomków, przestrzeni nazw ani atrybutów.
Following-sibling	To samo co oś Following, lecz zawiera wszystkie węzły, których rodzicem jest ten sam węzeł co w przypadku węzła kontekstu.
Namespace	Zawiera węzły przestrzeni nazw węzła kontekstu, o ile węzeł kontekstu jest elementem.
Parent	Zawiera węzeł rodzica węzła kontekstu. Węzeł główny nie posiada rodzica. Oś jest przeciwieństwem osi potomków.
Preceding	To samo co oś Following, lecz zamiast węzłów po węźle kontekstu będzie zawierać węzły występujące przed węzłem kontekstu biorąc pod uwagę kolejność w dokumencie.
Preceding-sibling	To samo co oś Preceding, lecz zawiera wszystkie węzły mające tego samego rodzica co węzeł kontekstu.
Self	Zawiera wyłącznie węzeł kontekstu.

Składnia XPath

Do wyboru węzłów podlegających przetwarzaniu XPath używa zestawu wyrażeń. Najczęściej spotykanym wyrażeniem jest lokalizacja ścieżki. Wyrażenie to zwraca **zestaw węzłów** (ang. *node set*). XPath pozwala na używanie składni pełnej oraz składni skróconej. Pełna składnia lokalizacji ścieżki ma następującą postać:

```
/nazwaOsi::węzełTest[predykat]/nazwaOsi::węzełTest[predykat]
```

W przykładzie występuje znak ukośnika, który odwołuje się do węzła głównego jako do węzła kontekstu. Następnie wskazywana jest oś oraz `węzełTest`, a także opcjonalny predykat. Dalej można umieszczać jedną lub więcej analogicznych struktur, by ostatecznie uzyskać węzły, które nas interesują. Zatem aby odczytać wszystkich klientów,

należałoby użyć pełnej składni XPath przedstawionej poniżej. Jeśli żadna oś nie zostanie wskazana, przyjęta zostanie domyślna oś child.

```
/child::root/child::customer
```

Jednak w większości przypadków używa się składni skróconej. W wersji skróconej powyższa instrukcja XPath będzie mieć postać `//customer`. Podwójny znak ukośnika oznacza oś descendant-or-self. Czasami w składni XPath używa się również znaków globalnych. XPath obsługuje następujące trzy rodzaje symboli wieloznacznych: *, `node()` oraz `@*`. Symbol wieloznaczny * pasuje do wszystkich elementów bez względu na ich typ, nie zwraca jedynie węzłów atrybutów, tekstowych, komentarzy ani instrukcji przetwarzania. Aby uzyskać wszystkie węzły, należy użyć składni `node()`, która zwraca również węzły wymienione przed chwilą. Z kolei symbol wieloznaczny `@*` pasuje do wszystkich węzłów atrybutów. W tabeli 7.2 przedstawiono skróconą postać wyrażeń wykorzystywanych w XPath.

Tabela 7.2. *Skrócone wyrażenia XPath*

Nazwa	Opis
`"domyślna"`	Jeśli nie zostanie podana oś, użyta będzie domyślna oś child.
`@`	Skrót oznaczający atrybuty.
`//`	Skrót oznaczający oś descendant-or-self.
`.`	Skrót oznaczający self.
`..`	Skrót oznaczający rodzica.
*	Symbol wieloznaczny, który umożliwia dopasowywanie wszystkich węzłów elementów bez względu na typ. Symbol nie zwraca jedynie węzłów atrybutów, tekstowych, komentarzy ani instrukcji przetwarzania.
/	Symbol używany jako separator ścieżek. Używany również jako ścieżka bezwzględna dla głównego węzła dokumentu.

Aby zwrócić wszystkie elementy potomne dla węzła `customer` z poprzedniego przykładu i aby uzyskać wyłącznie wszystkie atrybuty, a także aby odczytać wszystkich klientów zamieszkujących w regionie `Northwest`, należałoby użyć następującej składni XPath:

```
/customer/*
/customer/@*
/customer[@region = "Northwest"]
```

Czasami używa się również złożonych lokalizacji ścieżek będących kombinacją większej liczby instrukcji ścieżki. Oprócz pojedynczego znaku ukośnika (/) symbolizującego węzeł główny w XPath istnieje dodatkowa specjalna składnia. Na przykład wszystkich potomków można wskazać za pomocą dwóch znaków ukośnika (//). Można także wskazać węzeł rodzica, wpisując dwie kropki (..). Z kolei aby wybrać bieżący element, można wpisać jedną kropkę (.). Poniższa przykładowa instrukcja XPath wybiera wszystkie węzły elementów występujące pod węzłem głównym, których `ID` ma wartość `10`:

```
//[@id = "10"]
```

Istnieje także możliwość odczytywania atrybutów. Aby wybrać atrybuty, używa się składni `@`. Na przykład aby wybrać atrybut `id`, należy użyć składni `/customer/name/@id`. Czasami trzeba przefiltrować węzły elementów w oparciu o posiadane przez nie atrybuty. Na przykład gdyby istnieli klienci, dla których element stanu nie posiadałby atrybutu regionu, można by takich klientów odczytać, używając zapisu `/customer/state[@name]`.

Funkcje XPath

XPath udostępnia funkcje, dzięki którym można zwracać wartości albo manipulować danymi XML. XPath posiada funkcje ciągów znaków, zestawów węzłów, liczbowe i logiczne. Najczęściej używa się funkcji `position()`, `count()`, `contains()`, `substring()`, `sum()` i `round()`.

- Funkcja `position()` zwraca wskazaną pozycję w dokumencie. Na przykład instrukcja `/customer[position() = 2]` zwróci element `customer` znajdujący się na pozycji numer `2`. Składnię funkcji można skrócić przez pominięcie fragmentu `position() =`. Na przykład zapis `/customer[2]` odpowiada wcześniejszej instrukcji.
- Funkcja `count()` zwraca liczbę węzłów w zestawie węzłów. Na przykład jeśli trzeba zliczyć wszystkich klientów, można wykonać instrukcję `/count(customer)`. A jeśli w dokumencie XML zapisane są również wszystkie zamówienia złożone przez klientów, można je zliczyć instrukcją `/customer/orders/counr(order)`.
- Funkcja `contains()` pobiera dwa ciągi znaków i zwraca `true`, jeśli drugi ciąg znaków zawiera się w pierwszym ciągu znaków. Na przykład gdy trzeba sprawdzić, czy imię „Tom” znajduje się w danym ciągu znaków, można wykonać instrukcję w postaci `/customer/name[contains(.,'Tom')]`.
- Funkcja `substring()` zwraca część wskazanego ciągu znaków. Pierwszym parametrem funkcji jest ciąg znaków. Drugim parametrem jest pozycja początkowa, zaś parametr ostatni wskazuje długość, na przykład `/customer/name[substring(.,1,3)]`.
- Funkcja `sum()`, jak można się domyślić, oblicza sumę liczb. Funkcja pobiera zestaw węzłów, a zatem jeśli trzeba zsumować wszystkie ceny produktów w danym zestawie, można do tego celu użyć funkcji `sum()`, na przykład: `sum(/products/product/price)`.
- Funkcja `round()` zaokrągla liczbę do najbliższej liczby całkowitej.

XMLDOM — XML Document Object Model

XMLDOM to interfejs programistyczny dla dokumentów XML. Dzięki XMLDOM programista może ładować, tworzyć, modyfikować i usuwać dane XML. Aby szybko zrozumieć zasadę działania XMLDOM, najlepiej jest posłużyć się przykładem. Użytkownicy SQL Servera mogą traktować XMLDOM jakby był zestawem danych, ponieważ stanowi on przechowywaną w pamięci reprezentację parsowanego dokumentu XML.

Gdy używany jest XMLDOM, trzeba najpierw zadeklarować obiekt typu `XMLDocument`. Typ `XMLDocument` rozszerza typ `XMLNode`, który reprezentuje węzeł dowolnego typu w dokumencie XML. Po zadeklarowaniu obiektu `XMLDocument` trzeba załadować lub utworzyć dokument XML. Do załadowania dokumentu XML można użyć metody `load` albo `loadxml`, jak w poniższym kodzie:

```
Imports System.Xml.XmlDocument
Dim oXMLDOM As New System.Xml.XmlDocument

oXMLDOM.Load("c:\myxml.xml")
'lub, jeśli XML istnieje już jako ciąg znaków
'oXMLDOM.LoadXml(strXML)
```

Gdy dokument zostanie już załadowany, można przemierzać jego węzły przy użyciu obiektu `XMLNode` oraz właściwości `ChildNodes` właściwości `DocumentElement`. Właściwość `DocumentElement` zwraca dokument XML, a właściwość `ChildNodes` zwraca kolekcję węzłów tworzących dokument. Poniższy kod przemierza dokument XML i zwraca znajdujące się w nim węzły:

```
Dim oXMLNode As System.Xml.XmlNode

Dim strResult As String = ""

For Each oXMLNode In oXMLDOM.DocumentElement.ChildNodes
    strResult += oXMLNode.Name & ": " & _
        oXMLNode.InnerText
Next

MsgBox(strResult)
```

XMLDOM pozwala również na odczytywanie elementów na podstawie nazwy znaczników — służy do tego funkcja `GetElementsByTagName`. Na przykład jeśli istnieje element o nazwie `customer`, można odczytać jego wartość następującą instrukcją:

```
MsgBox(oXMLDOM.GetElementsByTagName("customer").Item(0).InnerText)
```

Funkcja `GetElementsByTagName` zwraca listę węzłów, którą można parsować. Powyższy kod odczytuje jedynie pierwszy węzeł na liście, lecz w razie potrzeby można przejść w pętli przez wszystkie węzły i po kolei je zwracać. W XMLDOM dostępne są również analogiczne funkcje `GetElementsByID` oraz `GetElementsByName`.

A w jaki sposób w XMLDOM można używać XPath? Otóż aby zwrócić listę węzłów odpowiadających instrukcji XPath dla XMLDOM, należy użyć metody `SelectNodes`. Metoda `SelectNodes` wymaga podania wyrażenia, którym może być instrukcja języka XML Stylesheet Language (XSL) lub wyrażenie XPath. Można także skorzystać z metody `SelectSingleNode`, aby uzyskać tylko jeden węzeł zamiast całej listy węzłów. Poniższy kod wykonuje przekazane wyrażenie i przechodzi przez zwrócone węzły, po czym wypisuje ich wartości:

```
Dim oNodes As System.Xml.XmlNodeList = oXMLDOM.SelectNodes(txtXPath.Text)
Dim strReturnString as string = ""

Dim oNode As System.Xml.XmlNode
For Each oNode In oNodes
    strReturnString = oNode.OuterXml
Next

Msgbox(strReturnString)
```

Klasy XPathDocument, XPathNavigator oraz XPathExpression

Szybkie tworzenie aplikacji przy użyciu XMLDOM nie budzi sprzeciwu, lecz w celu zaimplementowania skalowalnych aplikacji .NET wykorzystujących XPath lepiej jest skorzystać z klas `XPathDocument`, `XPathExpression` i `XPathNavigator`.

- Klasa `XPathDocument` to wysoko wydajna pamięć podręczna o charakterze tylko do odczytu dla dokumentów `FOR XML`, której głównym zadaniem jest parsowanie i wykonywanie zapytań XPath na dokumencie.
- Klasa `XPathNavigator` bazuje na modelu danych XPath. Klasa pozwala na wykonywanie zapytań na dowolnej składnicy danych. Dzięki `XPathNavigator` można kompilować często używane wyrażenia XPath.
- Klasa `XPathExpression` reprezentuje skompilowane wyrażenie XPath, które można wykonywać z poziomu klasy `XPathNavigator`.

Kod widoczny poniżej tworzy instancję obiektu `XPathDocument` i ładuje do niego dane XML. Następnie przy użyciu metody `CreateNavigator` kod tworzy obiekt `XPathNavigator`. Ta sama metoda jest również obsługiwana przez klasę `XMLDocument`. Aby wykonać wyrażenie XPath, kod wywołuje metodę `Select` i przekazuje do niej wyrażenie. Jak można zauważyć, zadaniem wyrażenia jest odszukanie klientów o nazwisku „Tom Rizzo” i zwrócenie stanów, w których ci klienci zamieszkują.

```
'utworzenie instancji klasy XPathDocument
Dim oXPathDoc As New System.Xml.XPath.XPathDocument("c:\note.xml")

'utworzenie instancji klasy XPathNavigator
Dim oXPathNav As System.Xml.XPath.XPathNavigator = oXPathDoc.CreateNavigator()

'utworzenie instancji klasy XPathIterator
Dim oXPathNodesIt As System.Xml.XPath.XPathNodeIterator = & _
oXPathNav.Select("//customer/name[. = 'Tom Rizzo']/parent::node()/state")

'utworzenie instancji ciągu znaków do przechowywania wyników
Dim strResult as string = ""

'klasa XPathIterator posłuży do nawigowania po wygenerowanym zestawie wyników,
'a następnie do wyświetlenia wybranych firm Parent Companies
Do While oXPathNodesIt.MoveNext
    strResult += oXPathNodesIt.Current.Value
Loop

Msgbox(strResult)
```

Wstawianie danych XML do bazy danych

Mamy już nieco więcej informacji na temat XML-a i XPath, zatem możemy zająć się metodami wstawiania danych XML do SQL Servera. Można to zrobić na kilka różnych sposobów. Pierwszy z nich polega na wstawieniu danych XML do kolumny typu `nvarchar` w bazie danych w ramach zwykłej instrukcji `INSERT`. Jest to standardowa technika wstawiania tekstu w kolumnie. W SQL Serverze 2005 można także zamiast kolumny tekstowej skorzystać z nowego typu danych `XML`.


Uwaga

> Typ danych `XML` zostanie opisany w następnym rozdziale, w którym przedstawione będą również sposoby użycia SQL Server Integration Services (SSIS) do wczytywania danych jako danych typu `XML`.

Dane XML można wstawiać do bazy danych również na trzy inne sposoby:

- Dane XML można rozdrobnić i wstawić do poszczególnych wierszy i kolumn w ramach pojedynczej instrukcji na bazie danych. Do tego celu należy użyć dostawcy zestawów wierszy `OPENXML`. `OPENXML` udostępnia widok zestawu wierszy na dokumencie XML i pozwala na pisanie instrukcji T-SQL parsujących dane XML.
- Kolejny sposób polega na wykorzystaniu diagramów aktualizujących. **Diagramy aktualizujące** (ang. *updategrams*) to struktury danych, których można używać do wyrażania zmian w danych za pomocą obrazów sprzed zmiany i po zmianie. Diagram aktualizujący jest odczytywany przez SQLXML, który na tej podstawie generuje polecenia SQL niezbędne do wprowadzenia zmian.
- Ostatni sposób polega na wykorzystaniu dostawcy XML `BulkLoad` dostępnego w SQLXML. Przy użyciu dostawcy można pobrać duży zestaw danych XML i w szybki sposób załadować go do SQL Servera.

Od razu nasuwa się pytanie o to, który z przedstawionych sposobów jest najlepszy. Otóż każdy z nich ma swoje wady i zalety. Jeśli najważniejszą kwestią jest jak najszybsze i jak najbardziej wydajne załadowanie danych XML do SQL Servera, wówczas najlepiej skorzystać z dostawcy `BulkLoad`. Dostawca `BulkLoad` nie próbuje wczytać wszystkich danych XML do pamięci, lecz w zamian wczytuje dane XML w postaci strumienia, dokonuje ich interpretacji i ładuje do SQL Servera. Dostawca `BulkLoad` jest oddzielnym komponentem, zatem jeśli zachodzi konieczność użycia mechanizmu ładującego dane wewnątrz procedury składowanej albo funkcji zdefiniowanej przez użytkownika, dostawcy `BulkLoad` nie można użyć. Można go natomiast używać w rozszerzonej procedurze składowanej (XP) poprzez wywołanie, lecz takie rozwiązanie jest rzadko stosowane i wiążą się z nim dodatkowe problemy. Dodatkowe problemy wiążą się głównie ze znacznym stopniem skomplikowania procedur rozszerzonych, trudnością w ich debugowaniu oraz, w przypadku, gdy procedura rozszerzona okaże się nieprawidłowo zaimplementowana, groźbą naruszenia zasad bezpieczeństwa serwera.

Z kolei OPENXML może być używany w procedurach składowanych i funkcjach zdefiniowanych przez użytkownika, ponieważ jest to część natywnego języka T-SQL. Ceną jest jednak niższa wydajność takiego rozwiązania. OPENXML wymaga, by do parsowania dokumentów mających później podlegać przetwarzaniu używać procedury składowanej sp_xml_preparedocument. Procedura składowana ładuje specjalną wersję parsera MSXML o nazwie MSXMLSQL, którego zadaniem jest parsowanie dokumentu XML, a następnie cały dokument XML zostaje załadowany do pamięci. Użycie OPENXML wymaga napisania dodatkowego kodu oraz wiąże się z większym wykorzystaniem zasobów. Więcej informacji na ten temat zostanie przedstawionych w dalszej części rozdziału, w punkcie „SQLXML: Widoki XML używające adnotowanych schematów XML".

Diagramy aktualizujące przydają się najbardziej w aplikacjach, w których trzeba zmodyfikować bazę danych i tworzenie adnotowanych schematów (o których więcej mówimy w dalszej części rozdziału) oraz wprowadzanie zmian na podstawie adnotowanych schematów nie budzi oporów. SQLXML pobiera diagramy uaktualniające i tłumaczy je na instrukcje DML języka SQL. Głównym ograniczeniem diagramów aktualizujących jest to, że jeśli w instrukcjach SQL DML trzeba zastosować logikę biznesową, wówczas użycie diagramów staje się niemożliwe, ponieważ nie można uzyskiwać dostępu do wygenerowanych instrukcji DML.

Zanim przejdziemy dalej i zaczniemy poznawać sposoby wykorzystania przedstawionych technologii, musimy najpierw odpowiednio skonfigurować SQL Server.

Czym jest SQLXML?

SQLXML to dodatkowy zestaw technologii udostępniany oddzielnie od SQL Servera 2000. Zestaw dostępnych technologii zawiera obsługę diagramów aktualizujących, dostawcę SQLXML BulkLoad, obsługę FOR XML po stronie klienta oraz obsługę SOAP. Najwyższą wersją SQLXML dla SQL Servera 2000 jest wersja 3.0. W przypadku SQL Servera 2005 SQLXML 4.0 jest dystrybuowany łącznie z serwerem, lecz może być także dystrybuowany jako samodzielny komponent, a ponieważ technologie udostępniane w SQLXML mogą być wykorzystywane również poza serwerem, trzeba było nadać im oddzielną nazwę. Należy pamiętać o tym, by nie mylić SQLXML z typem danych XML dostępnym w SQL Serverze 2005, co zdarza się niektórym osobom.

Nie należy również mylić SQLXML ze standardem SQL/XML, znanym również jako standard SQLX. SQLX to standard ANSI/ISO, który definiuje sposób używania danych XML w relacyjnych bazach danych. Microsoft jest członkiem roboczego komitetu zajmującego się rozwojem standardu SQLX. Aktualnie SQL Server nie obsługuje standardu SQLX, lecz udostępnia odpowiadające mu funkcje w zakresie czynności obsługiwanych przez standard. Na przykład SQLX definiuje sposób publikowania danych XML, co w SQL Serverze jest realizowane przy użyciu słowa kluczowego FOR XML. Natomiast do dekompozycji danych XML można używać typu danych XML albo dostawcy OPENXML. Dodatkowo SQL Server udostępnia funkcje, które nie są objęte standardem, takie jak łączenie zapytań XQueries oraz zapytań relacyjnych.

Konfiguracja SQL Servera

W przypadku SQLXML 3.0 SQL Server 2000 wymagał obecności serwera IIS nasłuchującego poleceń SQL w postaci zapytań URL albo wywołań SOAP. Proces nasłuchujący ISAPI, który zostanie opisany później, parsuje polecenie SQL, wykonuje je i zwraca wynik. W SQL Serverze 2005 serwer posiada wbudowany mechanizm nasłuchiwania na porcie wywołań HTTP bez konieczności angażowania serwera IIS. Dla celów przykładu prezentowanego w tym rozdziale utworzymy prosty węzeł końcowy, aby móc wysyłać do serwera zapytania i diagramy aktualizujące, natomiast szczegółowy opis sposobu obsługi usług sieciowych znajduje się w następnym rozdziale. Jeśli dotychczas używane było narzędzie konfigurujące IIS udostępniane przez SQLXML 3.0, obecnie w SQL Serverze 2005 nie jest już ono potrzebne, a zamiast niego można skorzystać z obsługi węzła końcowego po stronie serwera. Jeśli natomiast nadal istotne jest pozostawienie serwera IIS jako warstwy pośredniczącej, trzeba będzie nadal używać procesu SQLISAPI udostępnianego w SQLXML 3.0, nasłuchującego dla SQL Servera 2000 albo SQL Servera 2005. SQLXML można także wywoływać z poziomu aplikacji, ponieważ komponent obsługuje model zarządzanych obiektów, o czym więcej powiemy w punkcie „Programowanie SQLXML w .NET i COM" w dalszej części rozdziału.

Aby skonfigurować serwer, wystarczy wykonać polecenie `CREATE ENDPOINT` i pozwolić serwerowi na nasłuchiwanie wsadowych poleceń T-SQL. Utworzymy w tym celu wirtualny katalog *pubs*, lecz należy pamiętać, że katalog ten nie pojawi się na liście katalogów wirtualnych serwera IIS. Warto mieć to na uwadze, ponieważ może istnieć węzeł końcowy, który będzie usiłował używać tego samego portu co istniejący węzeł końcowy IIS. SQL i IIS nie mogą jednak nasłuchiwać jednocześnie na tym samym porcie. Odpowiedni kod przedstawia się następująco:

```
CREATE ENDPOINT pubs
  STATE = STARTED
  AS HTTP (
    path='/pubs',
    AUTHENTICATION=(INTEGRATED),
    PORTS = (CLEAR)
  )
  FOR SOAP(
    WSDL = DEFAULT,
    BATCHES=ENABLED
  )
GO
```

OPENXML

Zamiast własnoręcznie parsować XML do postaci wierszy poprzez ładowanie i parsowanie danych XML, a następnie iterowanie przez te dane i generowanie odpowiednich poleceń T-SQL, można użyć funkcji `OPENXML`. Składnia `OPENXML` może na początku wydać się skomplikowana, lecz gdy się ją wypróbuje na praktycznym przykładzie, okaże się, że jest ona całkiem zrozumiała:

```
OPENXML(uchwyt int [in],wzorzecwiersza nvarchar[in],[znaczniki byte[in]])
[WITH (DeklaracjaSchematu | NazwaTabeli)]
```

Pierwszym parametrem jest całkowitoliczbowy uchwyt dokumentu XML. **Uchwyt** to po prostu unikatowy całkowitoliczbowy identyfikator dokumentu. Można go odczytać przy użyciu wbudowanej procedury składowanej `sp_xml_preparedocument`. Gdy dokument XML jest przekazywany jako parametr, procedura `sp_xml_preparedocument` wykonuje parsowanie dokumentu i zwraca identyfikator, który należy przekazać do funkcji `OPENXML`. Przekazywany dokument XML może mieć postać tekstową albo może być to wartość typu XML będącego nowością w SQL Serverze 2005. Opcjonalnie można przekazać przestrzeń nazw URI (ang. *Uniform Resource Identifier*) dla wyrażenia XPath. Użycie tego parametru będzie zależeć od sposobu uwzględnienia przestrzeni nazw w dokumencie XML. Jeśli przestrzenie nazw nie będą używane, wówczas w większości wywołań procedury składowanej parametr również nie będzie używany.

Z kolei wbudowana procedura składowana `sp_xml_removedocument` pobiera całkowitoliczbowy uchwyt do dokumentu XML i usuwa wewnętrzną reprezentację tego dokumentu znajdującą się w pamięci, utworzoną przez procedurę `sp_xml_preparedocument`. Procedurę `sp_xml_removedocument` należy wywoływać po zakończeniu przetwarzania dokumentu XML. Jeśli procedura nie zostanie wywołana, SQL Server zniszczy reprezentację znajdującą się w pamięci w momencie rozłączenia się sesji, która tę reprezentację utworzyła. Jednak wykorzystywanie takiej metody nie jest zalecaną praktyką.

Drugim parametrem funkcji `OPENXML` jest wyrażenie XPath, przy użyciu którego ma zostać wykonane parsowanie wierszy. Może to być dowolne wyrażenie od najprostszego po bardzo skomplikowane.

Trzeci parametr ma charakter opcjonalny i służy do przełączania mechanizmu odwzorowywania z bazującego na atrybutach do bazującego na elementach. Domyślnie `OPENXML` używa odwzorowania bazującego na atrybutach, któremu odpowiada wartość `0`. Jeśli dokument XML bazuje na elementach, wówczas odwzorowanie należy przełączyć na bazujące na elementach przez przypisanie parametrowi wartości `2`. Z kolei wartość `1` nakazuje SQL Serverowi, by domyślnie używał odwzorowań bazujących na atrybutach, natomiast względem kolumn nieprzetworzonych stosował odwzorowanie bazujące na elementach. Wartość `8` parametru wskazuje, aby nadmiarowego tekstu nie kopiować do metawłaściwości `@mp:xmltext`, o której powiemy za chwilę.

Ostatnim elementem jest klauzula `WITH`. Klauzula wskazuje definicję schematu dla nowo tworzonego zestawu wierszy albo tabelę, jeśli wiadomo, że XML będzie odwzorowywany na tabelę, która już istnieje. Definicja schematu ma następując format:

```
NazwaKolumny TypKolumny [WzorzecKolumny | MetaProperty][, NazwaKolumny TypKolumny
[WzorzecKolumny | MetaProperty]...]
```

Poszczególne części definicji mają następujące znaczenie:

- `NazwaKolumny` to nazwa kolumny w tabeli.
- `TypKolumny` to typ danych SQL dla kolumny. Jeśli typ danych `XML` i typ danych SQL różnią się od siebie, mamy wówczas do czynienia z koercją. Oznacza to, że SQL Server spróbuje znaleźć najbliższy natywny typ danych, w którym będzie można przechowywać dane.

- `WzorzecKolumny` to wyrażenie XPath, które wskazuje funkcji `OPENXML` sposób odwzorowywania wartości XML na kolumny SQL. Na przykład można jawnie wskazać funkcji `OPENXML`, że dla danej wartości kolumny należy użyć określonego atrybutu albo elementu węzła rodzica. Jeśli wzorzec kolumny nie zostanie wskazany, użyte zostanie wskazane odwzorowanie domyślne na atrybut lub element.
- `MetaProperty` to atrybut metawłaściwości, który ma zostać umieszczony w kolumnie. Atrybuty metawłaściwości w dokumencie XML to atrybuty, które opisują właściwości jednostek XML (elementów, atrybutów albo innych węzłów DOM). Atrybuty te nie istnieją fizycznie w tekście dokumentu XML, lecz `OPENXML` udostępnia metawłaściwości dla wszystkich jednostek XML. Dzięki metawłaściwościom można wyodrębniać informacje takie jak pozycje lokalne czy informacje o przestrzeniach nazw węzłów XML. W ten sposób można uzyskiwać dane bardziej szczegółowe, niżby to wynikało z tekstowej reprezentacji dokumentu. Metawłaściwości można odwzorowywać w instrukcji `OPENXML` na kolumny zestawu wierszy, używając parametru `WzorzecKolumny`.

W tabeli 7.3 przedstawiono dostępne wartości atrybutu `MetaProperty`.

Tabela 7.3. *Dostępne wartości atrybutu MetaProperty*

Nazwa	Opis
`@mp:id`	Unikatowy identyfikator węzła DOM, który jest identyfikatorem prawidłowym do momentu ponownego parsowania dokumentu.
`@mp:localname`	Lokalny element nazwy węzła. Element ten można umieścić w kolumnie, jeśli nazwa węzła będzie potrzebna w późniejszym czasie.
`@mp:namespaceuri`	Zwraca przestrzeń nazw URI bieżącego elementu.
`@mp:prefix`	Zwraca prefiks przestrzeni nazw bieżącego elementu.
`@mp:prev`	Zwraca identyfikator węzła poprzedniego rówieśnika.
`@mp:xmltext`	Zwraca tekstową wersję danych XML. Reprezentacja tekstowa przydaje się do przetwarzania nadmiarowego oraz do obsługi w kodzie bazy danych nieznanych sytuacji. Na przykład gdy `MetaProperty` jest używana jako przepełnienie i XML ulegnie zmianie, nie trzeba będzie zmieniać kodu w bazie danych, aby tę zmianę obsłużyć.
`@mp:parentid`	Zwraca identyfikator węzła rodzica.
`@mp:parentlocalname`	Zwraca lokalną nazwę węzła rodzica.
`@mp:parentnamespaceuri`	Zwraca przestrzeń nazw URI rodzica.
`@mp:parentprefix`	Zwraca prefiks rodzica.

Przyjrzyjmy się teraz kilku przykładowym sposobom użycia funkcji `OPENXML`. Wykorzystamy poniższy prosty dokument XML:

```
<ROOT>
  <authors>
    <au_id>172-32-1176</au_id>
    <au_lname>White</au_lname>
    <au_fname>Johnson</au_fname>
```

```
    <phone>408 496-7223</phone>
    <address>10932 Bigge Rd.</address>
    <city>Menlo Park</city>
    <state>CA</state>
    <zip>94025</zip>
    <contract>1</contract>
  </authors>
  <authors>
    <au_id>213-46-8915</au_id>
    <au_lname>Green</au_lname>
    <au_fname>Marjorie</au_fname>
    <phone>415 986-7020</phone>
    <address>309 63rd St. #411</address>
    <city>Oakland</city>
    <state>CA</state>
    <zip>94618</zip>
    <contract>1</contract>
  </authors>
</ROOT>. . .
```



Uwaga

> Niektórzy czytelnicy zauważyli już pewnie w kodzie XML sztuczkę, jaką się posłużyliśmy. Generowanie przykładowych danych XML to żadna frajda, dlatego użyliśmy funkcji `FOR XML`, którą później opiszemy, na tabeli `authors` w bazie danych `pubs`. Jedyna dokonana zmiana wynika z faktu, że wolimy posługiwać się elementami zamiast atrybutami, dlatego `FOR XML` została skonfigurowana w taki sposób, by zwróciła dane w formacie bazującym na elementach.

Celem przykładu jest odczytanie dokumentu XML i zapisanie go w tabeli relacyjnej. Ponieważ używana jest już baza danych `pubs`, najłatwiej będzie odczytać dane i umieścić je w nowej tabeli `authorsXML` w bazie `pubs`. Wykorzystamy domyślne ustawienia funkcji `OPENXML` i nie będziemy używać żadnego jej parametru opcjonalnego. Kod odczyta dokument XML z odwzorowaniem bazującym na elementach, wykona parsowanie dokumentu i umieści go w tabeli `authorsXML`.

```
CREATE TABLE [authorsXML] (
    [title] [varchar] (20),
    [au_id] [varchar] (11)
) ON [PRIMARY]
GO
DECLARE @idoc int
DECLARE @doc varchar(1000)
SET @doc ='
<ROOT>
  <authors><au_id>172-32-1176</au_id><au_lname>White</au_lname>
<au_fname>Johnson</au_fname><title>książka1</title>
<phone>408 496-7223</phone><address>10932 Bigge Rd.</address>
<city>Menlo Park</city><state>CA</state><zip>94025</zip>
<contract>1</contract></authors>
<authors><au_id>213-46-8915</au_id><au_lname>Green</au_lname>
<au_fname>Marjorie</au_fname><title>książka2</title>
<phone>415 986-7020</phone><address>309 63rd St.
#411</address><city>Oakland</city><state>CA</state>
<zip>94618</zip>
<contract>1</contract></authors>
```

```
</ROOT>'
--utworzenie wewnętrznej reprezentacji dokumentu XML
EXEC sp_xml_preparedocument @idoc OUTPUT, @doc
--wykonanie instrukcji SELECT, która używa dostawcy zestawu wierszy OPENXML
INSERT AuthorsXML (title, au_id)
SELECT title, au_id
FROM OPENXML (@idoc, '/ROOT/authors',2)
     WITH (au_id varchar(11),
           au_lname varchar(40),
           au_fname varchar(20),
           title varchar(20),
           phone char(12)
          )

EXEC sp_xml_removedocument @idoc
```

Gdybyśmy nieco zmienili powyższy kod i usunęli instrukcję `INSERT`, pozostawiając tylko instrukcję `SELECT` w postaci `SELECT *`, SQL Server zwróciłby dane XML po parsowaniu mające postać zestawu wierszy.

Wynik zapytania byłby następujący:

```
au_id       au_lname            au_fname                phone
----------- ------------------- ----------------------- --------------
172-32-1176 White               Johnson                 408 496-7223
213-46-8915 Green               Marjorie                415 986-7020
```

Teraz jednak widać, że niektóre dane XML nie zostaną zapisane, na przykład adres, miasto, stan, kod pocztowy i kontrakt. Jeśli konieczne będzie zapisanie wszystkich danych, można odczytać cały dokument XML, tworząc dodatkową kolumnę i w definicji schematu wykorzystując polecenie `@mp:xmltext` w następujący sposób:

```
catchall nvarchar(1000) '@mp:xmltext'
```

Ostatni przykład użycia funkcji `OPENXML` pokazuje, w jaki sposób można pobrać wynik działania funkcji `OPENXML` i zapisać go w tabeli. Tę samą czynność wykonano w poprzednim przykładzie, lecz tym razem pokażemy dodatkowo, jak w `OPENXML` można nawigować po dokumencie XML, korzystając z wyrażenia XPath. Ponieważ `OPENXML` zwraca relacyjny zestaw wierszy, można uzyskany wynik połączyć z inną tabelą, a następnie zapisać wynik we własnej tabeli. Po wywołaniu tabeli dane XML można traktować jak każde inne dane o charakterze relacyjnym. W naszym przykładzie użyjemy wynikowego zestawu wierszy XML do połączenia danych z zawartością tabeli wydawców, aby zwrócić tylko tych autorów, którzy mieszkają w tym samym mieście co wydawca.

```
DECLARE @idoc int
DECLARE @doc varchar(1000)
SET @doc ='
<ROOT>
  <authors><au_id>172-32-1176</au_id><au_lname>White</au_lname>
<au_fname>Johnson</au_fname>
<phone>408 496-7223</phone><address>10932 Bigge Rd.</address>
```

```
<city>Menlo Park</city><state>CA</state><zip>94025</zip>
<contract>1</contract>
  <books>
    <title>Moja książka1</title>
    <title>Moja książka 2</title>
  </books>
</authors>
<authors><au_id>213-46-8915</au_id><au_lname>Green</au_lname>
<au_fname>Marjorie</au_fname>
<phone>415 986-7020</phone><address>309 63rd St. #411</address>
<city>Boston</city><state>MA</state>
<zip>94618</zip><contract>1</contract>
  <books>
    <title>Moja książka 3</title>
    <title>Moja książka 4</title>
  </books>
</authors>
</ROOT>'
--utworzenie wewnętrznej reprezentacji dokumentu XML
EXEC sp_xml_preparedocument @idoc OUTPUT, @doc

SELECT a.title, a.au_lname, p.pub_name, p.city
FROM OPENXML (@idoc, '/ROOT/authors/books',2)
     WITH (title varchar(20) './title',
           au_id varchar(11) '../au_id',
           au_lname varchar(40) '../au_lname',
           au_fname varchar(20) '../au_fname',
           phone char(12) '../phone',
           city varchar(20) '../city'
) AS a
INNER JOIN publishers AS p
    ON a.city = p.city
EXEC sp_xml_removedocument @idoc
```

Wynik powinien mieć następującą postać:

```
title           au_lname                  pub_name            city
--------------- ------------------------- ------------------- --------------
Moja książka3   Green                     New Moon Books      Boston
```

Najlepszym sposobem wykorzystania funkcji `OPENXML` jest użycie jej w procedurze składowanej, zwłaszcza jeśli dane XML są pobierane z warstwy pośredniej i umieszczane w bazie danych. Zamiast wykonywać parsowanie w warstwie pośredniej, można wysłać dane XML w postaci tekstowej do procedury składowanej, a potem na serwerze wykonać parsowanie danych i umieścić je w tabeli w ramach jednej operacji. W ten sposób znacznie zyskamy na wydajności i ograniczymy ilość danych przesyłanych w sieci w porównaniu z przypadkiem, gdy parsowanie byłoby wykonywane w warstwie pośredniej i konieczne byłoby wysyłanie poleceń T-SQL do serwera w celu zapisania danych.

Jeśli dane XML po parsowaniu mają być wielokrotnie wykorzystywane, wówczas zamiast co chwila wywoływać funkcję `OPENXML`, lepiej jest je zapisać w zmiennej tabelowej. Znacznie przyspieszy to przetwarzanie i zwolni zasoby, które serwer będzie

mógł wykorzystać do realizacji innych zadań. Przedstawiona poniżej przykładowa procedura wykorzystuje OPENXML. Warto zwrócić uwagę na użycie nowego typu danych nvarchar(max). W SQL Serverze 2000 należałoby użyć tekstowego typu danych. We wszystkich nowych rozwiązaniach zamiast typu danych text powinno się używać typu nvarchar(max), ponieważ obsługa typu text może zostać zakończona w którejś z przyszłych wersji serwera.

```
CREATE PROCEDURE update_authors_OPENXML (
@doc nvarchar(max))

AS
SET NOCOUNT ON
--uchwyt dokumentu:
DECLARE @idoc INT

--utworzenie wewnętrznej reprezentacji dokumentu XML
EXEC sp_xml_preparedocument @idoc OUTPUT, @doc
--wykonanie instrukcji SELECT, która używa dostawcy zestawów wierszy OPENXML
INSERT AuthorsXML (title, au_id)
SELECT title, au_id
FROM OPENXML (@idoc, '/ROOT/authors/books',2)
     WITH (title varchar(20) './title',
           au_id varchar(11) '../au_id',
           au_lname varchar(40) '../au_lname',
           au_fname varchar(20) '../au_fname',
           phone char(12) '../phone'
          )

--wykonanie procedury składowanej

EXEC update_authors_OPENXML '
<ROOT>
  <authors><au_id>172-32-1176</au_id><au_lname>White</au_lname>
<au_fname>Johnson</au_fname><phone>408 496-7223</phone>
<address>10932 Bigge Rd.</address><city>Menlo
Park</city><state>CA</state><zip>94025</zip><contract>1</contract>
  <books>
    <title>Moja książka1</title>
    <title>Moja książka 2</title>
</books>
</authors>
<authors><au_id>213-46-8915</au_id><au_lname>Green</au_lname>
<au_fname>Marjorie</au_fname><phone>415 986-7020</phone>
<address>309 63rd St. #411</address><city>Oakland</city><state>CA</state>
<zip>94618</zip><contract>1</contract>
  <books>
    <title>Moja książka 3</title>
    <title>Moja książka 4</title>
  </books>
</authors>
</ROOT>'
```

SQLXML — widoki XML używające adnotowanych schematów XML

Schematy XML definiują strukturę dokumentu XML w taki sam sposób, w jaki schemat relacyjny definiuje strukturę relacyjnej bazy danych. Przy użyciu schematów można definiować kryteria decydujące o tym, czy dokument XML jest zgodny ze specyfikacją. Na przykład można zdefiniować elementy, atrybuty, hierarchie elementów, kolejność elementów, typy danych elementów i atrybutów, a także dowolne wartości domyślne dla elementów i atrybutów. Schematy nie są wymagane w dokumentach XML, lecz zaleca się ich używanie zwłaszcza, jeśli dane XML będą współużytkowane z innymi aplikacjami, które nie będą potrafiły rozpoznawać danych XML lub które nie będą potrafiły tworzyć danych XML bez rozumienia schematu.

Standardem dla schematów jest XML Schema Definition (XSD). Wcześniej stosowano także technologię opracowaną przez XML, noszącą nazwę XML Data Reduced (XDR). Technologia XRD była prekursorem XSD i nie powinno się już używać jej w aplikacjach.

W SQL Serverze można tworzyć schemat XML, który odwzorowuje się na strukturę relacyjną przy użyciu pewnych specjalnych znaczników schematu. Rozwiązanie takie przydaje się wówczas, gdy trzeba utworzyć widok XML danych relacyjnych. Widok nie tylko pozwala na wykonywanie zapytań na danych relacyjnych i otrzymywanie wyników w postaci XML, ale również umożliwia zachowywanie zmian dzięki diagramom uaktualniającym oraz zbiorczemu ładowaniu SQLXML. Tworzenie schematu adnotowanego wymaga trochę pracy, lecz jeśli później użytkownik będzie intensywnie korzystał z danych XML, poświęcenie nieco większej ilości czasu na utworzenie schematu na pewno się opłaci. Ponadto schematów warto jest używać wraz z diagramami uaktualniającymi, o których więcej powiemy w punkcie „Diagramy uaktualniające SQLXML" w dalszej części rozdziału.


Uwaga

W niniejszym rozdziale zakładamy, że czytelnik posiada już pewną wiedzę na temat schematów XML. Czytelnicy, którzy takiej wiedzy nie mają, powinni przeczytać wstęp do opisu schematów XML na witrynie organizacji W3C pod adresem *http://www.w3.org/TR/xmlschema-0*.

Visual Studio zawiera bardzo poręczny edytor schematów XML, dzięki któremu użytkownik nie musi generować schematów ręcznie. Poniżej znajduje się standardowy schemat XML dla danych XML `authors`, których używaliśmy w poprzednich punktach. Jak widać, schemat XML jest dokumentem XML. System rozpoznaje, że dany dokument jest schematem dzięki temu, że zadeklarowano w nim przestrzeń nazw `xs`, która używa przestrzeni nazw XSD. Przestrzeń nazw XSD stanowi odwołanie do przestrzeni nazw W3C XSD, którą jest `http://www.w3c.org/2001/XMLSchema`. Odwołaniu przypisywany jest alias `xs`, a wszystkie elementy używają aliasu w schemacie jako prefiksu.

Warto również zwrócić uwagę na sposób, w jaki w schemacie zadeklarowano element o nazwie `AuthorsXMLNew`, który zawiera pozostałą część danych XML. Następnie w schemacie występuje typ `complexType`, który stanowi deklarację sekwencji ele-

mentów XML. Wśród elementów występują ID, imię, nazwisko, numer telefonu oraz inne informacje dotyczące autorów. Ważny jest sposób, w jaki w elementach deklaruje się typ. Schematy mogą definiować typy danych dla elementów i atrybutów, zatem w naszym schemacie deklarowane są ciągi znaków, nieoznaczona liczba całkowita oraz nieoznaczony bajt. Schemat może zawierać również deklaracje innych typów, takich jak daty, `Boolean`, dane binarne i inne.

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema attributeFormDefault="unqualified"
elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="AuthorsXMLNew">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="au_id" type="xs:string" />
        <xs:element name="au_lname" type="xs:string" />
        <xs:element name="au_fname" type="xs:string" />
        <xs:element name="phone" type="xs:string" />
        <xs:element name="address" type="xs:string" />
        <xs:element name="city" type="xs:string" />
        <xs:element name="state" type="xs:string" />
        <xs:element name="zip" type="xs:unsignedInt" />
        <xs:element name="contract" type="xs:unsignedByte" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Gdy istnieje już schemat bazowy i trzeba utworzyć na jego podstawie schemat adnotowany, który będzie używany z SQLXML, należy wprowadzić odpowiednie zmiany. Najpierw trzeba dodać odwołanie do odwzorowania schematu XML. W tym celu należy zmodyfikować schemat XML, dodając do niego przestrzeń nazw dla odwzorowania schematu SQL Servera, którą jest `urn:schemas-microsoft-com:mapping-schema`. Dzięki temu schematowi posiadany schemat XML można odwzorowywać na schemat relacyjnej bazy danych. Przestrzeni nazw nadamy alias `sql`, aby w odwołaniach do niej móc używać symbolu `sql:`. Zatem aby zmodyfikować poprzedni schemat w taki sposób, by obsługiwał on odwzorowanie SQL Servera, należałoby użyć nowego schematu w następującej postaci:

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema id="XMLSchema1" targetNamespace="http://tempuri.org/XMLSchema1.xsd"
  elementFormDefault="qualified"
  xmlns="http://tempuri.org/XMLSchema1.xsd"
  xmlns:mstns="http://tempuri.org/XMLSchema1.xsd"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:sql="urn:schemas-microsoft-com:mapping-schema">
. . .
```

W różnych dokumentach często występuje również przestrzeń nazw `urn:schemas-microsoft-com:xml-sql`. Przestrzeń ta daje dostęp do funkcji SQLXML, której można używać w szablonach lub zapytaniach XPath.

Odwzorowanie domyślne

Nietrudno się zorientować, że w poprzednim schemacie dodano po prostu przestrzeń nazw dla celów odwzorowywania. Schemat nie został wskazany, ponieważ SQL Server obsługuje domyślne odwzorowanie między schematem relacyjnym oraz schematem XML. Na przykład typ złożony `authors` zostałby automatycznie odwzorowany na tabelę `authors`, ciąg znaków `au_id` zostałby automatycznie odwzorowany na kolumnę `au_id` i tak dalej.

Odwzorowanie jawne

Odwzorowania między schematem i typami danych SQL można także definiować w sposób jawny. Odwzorowań domyślnych można używać w bardzo prostych aplikacjach, natomiast w większości przypadków używa się odwzorowania jawnego. Przyczyną są zwykle różnice między schematami XML i relacyjnym albo chęć utrzymania większej kontroli nad sposobem wykonywania odwzorowań oraz użycia typów danych. Odwzorowanie między jednostką XML a tabelą SQL definiuje się przy użyciu znacznika `sql:relation`, który należy do schematu odwzorowania SQLXML. Dla kolumn używa się znacznika `sql:field`. Dostępny jest również znacznik `sql:datatype`, dzięki któremu można jawnie odwzorowywać typy danych XML na typy danych SQL i zapobiegać wykonywaniu konwersji niejawnej. Jeśli więc zrezygnujemy z domyślnego odwzorowywania dla naszego schematu i dodamy wspomniane znaczniki, schemat w nowej postaci wyglądałby następująco:

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:sql="urn:schemas-microsoft-
com:mapping-schema">
  <xs:element name="AuthorsXMLNew" sql:relation="AuthorsXMLNew">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="au_id" type="xs:string" sql:field="au_id" />
        <xs:element name="au_lname" type="xs:string" sql:field="au_lname" />
        <xs:element name="au_fname" type="xs:string" sql:field="au_fname" />
        <xs:element name="phone" type="xs:string" sql:field="phone" />
        <xs:element name="address" type="xs:string" sql:field="address" />
        <xs:element name="city" type="xs:string" sql:field="city" />
        <xs:element name="state" type="xs:string" sql:field="state" />
        <xs:element name="zip" type="xs:unsignedInt" sql:field="zip" />
        <xs:element name="contract" type="xs:unsignedByte" sql:field="contract"
        sql:datatype="bit" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Relacje

W relacyjnych bazach danych można tworzyć relacje między danymi na podstawie kluczy, natomiast w celu opisania zdefiniowanych relacji w XML-u należy użyć schematów adnotowanych.

Jednak w schemacie adnotowanym relacje nabiorą hierarchicznego charakteru ze względu na użycie odwzorowania `sql:relationship`. Można to traktować jak złączenie tabeli. Odwzorowanie relacji posiada element rodzica, który definiuje relację albo tabelę rodzica. W odwzorowaniu występuje także element rodzic-klucz, który wskazuje używany klucz, a ten z kolei może składać się z wielu kolumn. Elementy potomka oraz pary potomek-klucz spełniają taką samą funkcję wobec potomka jak analogiczne elementy dla rodzica.

Schematy udostępniają również funkcję odwrotną, dzięki której relacje można wyłączać. Jeśli z jakiegoś powodu odwzorowanie powinno różnić się od relacji klucz główny-klucz obcy zdefiniowanej w tabeli, relacja zostanie wyłączona dzięki atrybutowi `inverse`. Sytuacja taka występuje w diagramach uaktualniających, o których powiemy więcej w punkcie „Diagramy uaktualniająceSQLXML" w dalszej części rozdziału. Atrybutu `inverse` używa się tylko w odniesieniu do diagramów uaktualniających.

Wyobraźmy sobie, że posiadamy dane o autorach i w schemacie relacyjnym dane autorów są powiązane z danymi na temat książek za pomocą identyfikatora autora. Odwzorowanie schematu można zmienić w taki sposób, by uwzględniało ono ustanowioną relację — odpowiedni kod znajduje się poniżej. Zwróćmy uwagę na fakt, że odwzorowanie relacji znajduje się w specjalnej sekcji schematu XSD.

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema id="XMLSchema1" targetNamespace="http://tempuri.org/XMLSchema1.xsd"
  elementFormDefault="qualified"
  xmlns="http://tempuri.org/XMLSchema1.xsd"
  xmlns:mstns="http://tempuri.org/XMLSchema1.xsd"
  xmlns:sql="urn:schemas-microsoft-com:mapping-schema"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="Root">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Authors" sql:relation="Authors">
          <xs:complexType>
            <xs:sequence>
            <xsd:element name="Books" sql:relation="Books">
              <xsd:annotation>
                <xsd:appinfo>
                  <sql:relationship name="BookAuthors"
                        parent="Authors"
                        parent-key="au_id"
                        child="Books"
                        child-key="bk_id" />
                </xsd:appinfo>
              </xsd:annotation>
            <xsd:complexType>
                <xsd:attribute name="bk_id" type="xsd:integer" />
                <xsd:attribute name="au_id" type="xsd:string" />
            </xsd:complexType>
            </xsd:element>
              <xs:element name="au_id" type="xs:string"
                  sql:field="au_id"></xs:element>
              <xs:element name="au_lname" type="xs:string"
```

```
                    sql:field="au_lname"></xs:element>
                <xs:element name="au_fname" type="xs:string"
                    sql:field="au_fname"></xs:element>
. . .
                <xs:element name="contract" type="xs:boolean"
                    sql:field="contract"
                    sql:datatype="bit"></xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Odwzorowywanie kolumn kluczy przy użyciu znacznika sql:key-fields

Wiemy już, w jaki sposób odwzorowuje się relacje, warto więc sprawdzić, jak wymusić na SQL Serverze prawidłowe zagnieżdżanie danych XML. Aby zapewnić prawidłowe zagnieżdżanie danych, należy wskazać kolumny kluczy występujące w tabeli, które odegrają największą rolę w tworzeniu hierarchii XML. Aby wskazać serwerowi odpowiednią kolejność, należy użyć odwzorowania `sql:key-fields`. Wskazuje ono kolumny, które zawierają wartości kluczowe. Poniższy schemat wskaże SQL Serverowi, że kolumną kluczową jest kolumna `au_id`:

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema id="XMLSchema1" targetNamespace="http://tempuri.org/XMLSchema1.xsd"
  elementFormDefault="qualified"
  xmlns="http://tempuri.org/XMLSchema1.xsd"
  xmlns:mstns="http://tempuri.org/XMLSchema1.xsd"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="Root">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Authors" sql:relation="Authors"
          sql:key-fields="au_id">
. . .
```

Pomijanie danych w wynikowych danych XML przy użyciu znacznika sql:mapped

Znacznik `sql:mapped` pozwala wskazywać, czy dany element lub atrybut w schemacie XSD powinien być odwzorowywany na obiekt bazy danych. Jeżeli chcemy zrezygnować z domyślnego odwzorowania i pominąć daną XML w danych wynikowych, należy użyć atrybutu `sql:mapped`. Zdarzają się przypadki, w których niektóre dane XML mają charakter nadmiarowy i nie powinny trafiać do tabeli — na przykład gdy użytkownik nie ma kontroli nad schematem XML i chce pominąć dane, ponieważ w tabeli nie występuje kolumna odpowiednia dla danych. Atrybut `sql:mapped` jest atrybutem logicznym, w którym wartość `true` oznacza, że odwzorowanie powinno zostać wykonane, natomiast `false` — że odwzorowanie nie powinno mieć miejsca.

Tworzenie elementu stałego

Gdy istnieje potrzeba utworzenia stałego elementu dokumentu XML, który powinien występować nawet w sytuacji braku odwzorowania do tabeli, należy użyć odwzorowania `sql:is-constant`. Odwzorowanie ma charakter logiczny, w którym wartość `true` powoduje, że element zawsze będzie się pojawiał w dokumencie XML. Odwzorowanie `sql:is-constant` bardzo się przydaje do tworzenia głównego elementu dokumentu XML. Poniżej znajduje się przykład zastosowania odwzorowania:

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema id="XMLSchema1"
  targetNamespace="http://tempuri.org/XMLSchema1.xsd" . . .>
  <xs:element name="Root" sql:is-constant="true">
. . .
```

Ograniczanie wartości przy użyciu filtra

W niektórych przypadkach konieczne jest filtrowanie wartości zwracanych do dokumentu XML na podstawie wartości z bazy danych. Do tego celu używa się odwzorowań `sql:limit-field` oraz `sql:limit-value`, które służą do definiowania kolumny podlegającej filtrowaniu oraz wartości, na podstawie której filtruje się zawartość kolumny. Wskazywanie wartości stanowiącej podstawę filtrowania nie jest wymagane, ponieważ domyślnie SQL Server przyjmie jako podstawę filtrowania wartość `null`. Ponadto użytkownik może zdefiniować większą liczbę wartości filtrujących dla wielu odwzorowań. Znajdujący się poniżej skrócony przykład ilustruje schemat, który ogranicza zbiór zwracanych autorów do osób mieszkających w Bostonie:

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema id="XMLSchema1"
  targetNamespace="http://tempuri.org/XMLSchema1.xsd" . . .>
  <xs:element name="Root" sql:is-constant="true">
. . .
          <xs:element name="Authors"
              sql:relation="Authors"
              sql:limit-field="city"
              sql:limit-value="Boston">
. . .
```

Pozostałe mechanizmy odwzorowywania schematów

W większości odwzorowywanych schematów używane będą odwzorowania opisane w poprzednich punktach. Pozostałe dostępne odwzorowania wraz z krótkim opisem zostały przedstawione w tabeli 7.4. Na końcu rozdziału znajduje się kompletny przykład schematu adnotowanego.

Diagramy uaktualniające SQLXML

W poprzednich punktach opisano sposoby rozdrabniania danych przy użyciu `OPENXML` oraz odczytywania danych z SQL Servera przy użyciu schematów adnotowanych. Schematy adnotowane stanowią podstawę diagramów uaktualniających, dlatego zostały opisane najpierw. Dzięki diagramom uaktualniającym można modyfikować dane

Tabela 7.4. *Pozostałe mechanizmy odwzorowywania schematów*

Nazwa	Opis
`sql:encode`	Wskazuje, czy należy zwracać adres URL lub dane binarne typu BLOB. Wartość `URL` powoduje zwrócenie adresu URL, natomiast wartość `default` oznacza, że należy zwrócić dane w bazowym formacie z kodowaniem base-64.
`sql:identity`	Umożliwia odwzorowywanie kolumny identyfikatorów SQL. Wartość `ignore` oznacza, że SQL Server wygeneruje wartość identyfikatora na podstawie ustawień w schemacie relacyjnym, natomiast wartość `useValue` wskazuje, że należy użyć innej wartości. Zwykle odwzorowaniu przypisuje się wartość `ignore`, chyba że używany jest diagram uaktualniający.
`sql:max-depth`	Służy do wskazywania głębokości rekurencji wykonywanej w relacji rodzica i potomka. Użytkownik może wskazać liczbę z zakresu od 1 do 50. Przykładem zastosowania może być generowanie struktury organizacyjnej, w której pracownicy mają przełożonych i podwładnych.
`sql:overflow-field`	Pozwala na wskazywanie kolumny bazy danych, która będzie zawierać dane nadmiarowe. Jeżeli część danych XML nie została odwzorowana na relacyjną bazę danych, wówczas dane tego rodzaju trafią do kolumny danych nadmiarowych. Jako wartość odwzorowania wskazuje się nazwę kolumny.
`sql:use-cdata`	Służy do wskazywania, czy dane zwracane przez SQL Server powinny być umieszczane w sekcji `CDATA` i przez to być traktowane przez parsery XML jako zwykłe dane tekstowe. W celu włączenia mechanizmu należy odwzorowaniu przypisać wartość `1`.

znajdujące się w SQL Serverze, używając do tego celu formatu XML. Zamiast pisać odpowiedni kod T-SQL, zmiany w danych przedstawia się w postaci obrazów sytuacji sprzed zmiany i po niej mających format XML. Diagramy uaktualniające można wykonywać z poziomu ADO lub ADO.NET, o czym przekonamy się dzięki kompletnemu przykładowi zamieszczonemu na końcu rozdziału.

Pierwszym krokiem pozwalającym zrozumieć działanie diagramów uaktualniających XML jest poznanie używanej przez diagramy przestrzeni nazw `urn:schemas-microsoft-com:xml-updategram`. W deklaracji przestrzeni nazw przestrzeń nazw diagramów uaktualniających jest zazwyczaj skracana do postaci `updg`.

Każdy diagram uaktualniający musi mieć co najmniej jeden element `sync`, który jest elementem XML i zawiera dane mające podlegać zmianie zdefiniowane przy użyciu elementów `before` i `after`. Liczba elementów `sync` może być większa niż jeden. Każdy element `sync` odpowiada jednej transakcji, co oznacza, że wszystkie zmiany wynikające z definicji elementu są zatwierdzane albo zostają w całości wycofane. Element `before` zawiera dane w postaci sprzed zmiany. Zwykle konieczne jest wskazanie klucza, na podstawie którego SQL Server będzie mógł odnaleźć dane mające ulec zmianie. Element `before` może zmieniać wyłącznie jeden wiersz tabeli.

Element `after` zawiera zmienione dane. Nietrudno się domyślić, że wstawienie danych będzie wymagało zdefiniowania odpowiedniej wartości elementu `after`, natomiast element `before` nie będzie posiadał żadnej wartości. Z kolei dla usunięcia danych trzeba wskazać wartość elementu `before`, a `after` będzie pusty. Dla operacji uaktualniania danych konieczne jest zdefiniowanie wartości zarówno dla elementu `before`, jak i `after`. Poniżej znajduje się diagram uaktualniający w najprostszej postaci:

```
<ROOT xmlns:updg="urn:schemas-microsoft-com:xml-updategram">
  <updg:sync [mapping-schema= "AnnotatedSchemaFile.xml"] >
    <updg:before>
      ...
    </updg:before>
    <updg:after>
      ...
    </updg:after>
  </updg:sync>
</ROOT>
```

Opcjonalnie można wskazać plik z adnotowanym schematem, który jawnie odwzoruje elementy z diagramu uaktualniającego na kolumny w tabelach. Jeśli plik z adnotowanym schematem nie zostanie wskazany, SQL Server użyje odwzorowania domyślnego, o czym wspomniano już w poprzednim punkcie dotyczącym odwzorowywania schematów adnotowanych. Należy również zaznaczyć, że możliwe jest jednoczesne używanie odwzorowań bazujących na elementach oraz odwzorowań bazujących na atrybutach. Jednak ze względu na czytelność schematu zaleca się, by zawsze korzystać tylko z jednego rodzaju odwzorowania.

Aby w diagramie uaktualniającym wskazać wartość pustą, należy zastosować atrybut `nullvalue` elementu `sync`, który zdefiniuje symbol zastępczy dla wartości `null`. Na przykład jeśli wartości `null` ma odpowiadać wartość `nothing`, można by użyć poniższego diagramu uaktualniającego, którego składnia bazuje na atrybutach:

```
<?xml version="1.0"?>
<authorsupdate xmlns:updg=
    "urn:schemas-microsoft-com:xml-updategram">
  <updg:sync updg:nullvalue="nothing">
    <updg:before>
      <Authors au_id="172-32-1176"/>
    </updg:before>
    <updg:after>
      <Authors state="nothing" phone="nothing"/>
    </updg:after>
  </updg:sync>
</authorsupdate>
```

W diagramach uaktualniających można także używać parametrów — służy do tego składnia `$nazwaparametru`. Na przykład aby utworzyć parametr dla podzbioru autorów, można by użyć następującego diagramu uaktualniającego:

```
<?xml version="1.0"?>
<authorsupdate xmlns:updg=
    "urn:schemas-microsoft-com:xml-updategram">
  <updg:sync updg:nullvalue="nothing">
    <updg:before>
      <Authors au_id="$AuthorID"/>
    </updg:before>
    <updg:after>
      <Authors state="nothing" phone="nothing"/>
    </updg:after>
  </updg:sync>
</authorsupdate>
```

Jeśli trzeba użyć kolumny identyfikatorów i przekazywać wartości identyfikatorów między różnymi tabelami, można do tego celu użyć atrybutu `at-identity`. Atrybut `at-identity` jest symbolem zastępczym, w którego miejsce w trakcie przetwarzania SQL Server wstawi prawidłową wartość. Jeśli wartość identyfikatora trzeba przekazać z powrotem do klienta, można do tego celu użyć atrybutu `returnid`. Dzięki temu SQL Server zwróci dokument XML zawierający wartość identyfikatora ustanowioną po wykonaniu diagramu uaktualniającego na serwerze.

Działanie opisanych atrybutów powinno być jasne po przeanalizowaniu przykładu. Aby wstawić nowego autora do tabeli `authors`, usunąć już istniejącego autora oraz zmienić dane jeszcze innego autora, należałoby użyć zaprezentowanego poniżej diagramu uaktualniającego na tabeli `authors`. W kolejnym punkcie pokażemy, jak należy zaimplementować kod .NET z klasami SQLXML, aby móc wykonać poniższy diagram.

```
<?xml version="1.0"?>
<authorsupdate xmlns:updg=
    "urn:schemas-microsoft-com:xml-updategram">
  <updg:sync updg:nullvalue="nothing">
    <updg:before>
    </updg:before>
    <updg:after>
      <Authors au_id="123-22-1232" au_fname="Tom" state="WA" phone="425-882-8080"/>
    </updg:after>
    <updg:before>
      <Authors au_id="267-41-2394"/>
    </updg:before>
    <updg:after>
    </updg:after>
    <updg:before>
      <Authors au_id="238-95-7766"/>
    </updg:before>
    <updg:after>
      <Authors city="Oakland" phone="212-555-1212"/>
    </updg:after>
  </updg:sync>
</authorsupdate>
```

Zbiorcze ładowanie danych XML

Jeśli do SQL Servera trzeba załadować duży zbiór danych XML, najlepiej jest zastosować do tego celu mechanizm `Bulkload` SQLXML. **Nie należy** natomiast używać do tego celu diagramów uaktualniających ani `OPENXML`, ponieważ obydwa wspomniane komponenty nie radzą sobie najlepiej pod względem wydajnościowym z ładowaniem dużych ilości danych XML. Od razu można zadać pytanie, jaki zbiór jest dużym zbiorem danych. Zależy to od wielu czynników, takich jak rozmiar i złożoność danych XML. Można ładować setki małych plików XML albo jeden duży plik XML. Jeśli używany serwer posiada kilka procesorów, znaczną ilość pamięci operacyjnej i szybkie dyski, można ewentualnie użyć `OPENXML`. Najlepiej natomiast jest przeprowadzić test i sprawdzić, które z rozwiązań będzie dawało akceptowalne rezultaty w przypadku ładowania określonego zbioru danych.

`Bulkload` XML to obiekt, który wywołuje się jako element obiektowego modelu SQLXML. Ten z kolei wywołuje funkcje zbiorczego ładowania danych SQL Servera, które wykonają ładowanie ze źródła danych XML do serwera. Przed uruchomieniem zbiorczego ładowania danych XML warto jest uruchomić śledzenie. W ten sposób będzie można sprawdzić, jakie dokładnie polecenia wykonuje `Bulkload`, a także naprawić ewentualne błędy powstałe w trakcie ładowania oraz uzupełnić brakujące dane.

`Bulkload` rozszerza zakres możliwości schematów odwzorowujących, o których mówiliśmy we wcześniejszych punktach rozdziału. Schemat odwzorowujący wskazuje komponentowi `Bulkload` miejsca w bazie danych, do których powinny trafić ładowane dane XML. Model obiektowy `Bulkload` jest bardzo prosty. Zawiera on jedną metodę o nazwie `Execute` oraz szereg właściwości, które służą do konfigurowania sposobu wykonywania zbiorczego ładowania danych. Metoda `Execute` przyjmuje dwa parametry. Pierwszym z nich jest ścieżka do pliku ze schematem odwzorowywania. Drugi parametr jest opcjonalny i wskazuje ścieżkę lub strumień pliku XML, który ma zostać zaimportowany.

Znamy już sposób wykonywania zbiorczego ładowania danych, warto zatem teraz zajrzeć do tabeli 7.5 prezentującej poszczególne właściwości. Następnie opisane zostaną najbardziej interesujące z nich, których zaleca się używać.

Tabela 7.5. *Właściwości obiektu Bulkload*

Nazwa	Opis
`Bulkload`	Właściwość logiczna, która wskazuje, czy należy wykonać zbiorcze ładowanie danych. Jeśli konieczne jest jedynie utworzenie schematu w bazie danych bez ładowania danych, właściwości należy przypisać wartość `false`. Wartością domyślną jest `true`.
`CheckConstraints`	Właściwość logiczna o domyślnej wartości `false`, która wskazuje, czy należy sprawdzać ograniczenia takie jak klucz główny i klucz obcy. Jeśli któreś z ograniczeń zostanie naruszone, zwrócony zostanie błąd.
`ConnectionCommand`	Właściwość pozwala na wskazywanie obiektu `Command` zamiast obiektu `ConnectionString` z właściwością `ConnectionString`. Jeśli wskazywany jest obiekt `Command`, wówczas właściwości `Transaction` należy przypisać wartość `true`.
`ConnectionString`	Ciąg znaków, który służy do przekazywania ciągu połączenia do SQL Servera.
`ErrorLogFile`	Ciąg znaków służący do wskazywania ścieżki, w której mają zostać zapisane błędy powstałe w trakcie zbiorczego ładowania danych. Dla każdego błędu powstanie jeden wpis, a błąd zauważony najpóźniej będzie występował na początku listy.
`FireTriggers`	Właściwość logiczna wskazująca, czy w trakcie ładowania danych w tabeli docelowej należy uruchamiać procedury wyzwalane. Wartością domyślną jest `false`.
`ForceTableLock`	Właściwość logiczna wskazująca, czy na czas zbiorczego ładowania danych należy zablokować całą tabelę. Wartością domyślną jest `false`.
`IgnoreDuplicateKeys`	Właściwość logiczna wskazująca, czy należy ignorować przypadki, w których do tabeli wstawiane są zduplikowane klucze. Wartością domyślną jest `false`, która oznacza, że występowanie duplikatów kluczy będzie ignorowane. Jeśli właściwości przypisana zostanie wartość `true` i w danych wystąpi zduplikowana wartość klucza, rekord nie zostanie wstawiony do tabeli.

Tabela 7.5. *Właściwości obiektu Bulkload — ciąg dalszy*

Nazwa	Opis
`KeepIdentity`	Właściwość logiczna wskazująca, czy należy zachować wartości identyfikatorów znajdujące się w zbiorze XML czy też SQL Server powinien automatycznie wygenerować wartości identyfikatorów. Domyślnie właściwość ma wartość `true`, dzięki czemu `Bulkload` zachowuje wartości identyfikatorów znajdujące się w dokumencie XML.
`KeepNulls`	Właściwość o wartości domyślnej `true`, która wskazuje, czy w przypadku, gdy kolumna nie zawiera wartości albo gdy nie należy używać domyślnej wartości zdefiniowanej dla kolumny, wstawiać wartość `null`.
`SchemaGen`	Właściwość o wartości domyślnej `false`, która wskazuje, czy przed rozpoczęciem zbiorczego ładowania danych należy utworzyć tabele relacyjne. Więcej informacji na temat właściwości znajduje się w dalszej części punktu.
`SGDropTables`	Właściwość o wartości domyślnej `false`, która wskazuje, czy należy usunąć i na nowo utworzyć odpowiednie tabele czy też pozostawić tabele już istniejące. Właściwość jest używana wraz z właściwością `SchemaGen`. Wartość `true` powoduje, że tabele zostaną usunięte i utworzone na nowo.
`SGUseID`	Właściwość o wartości domyślnej `false`, która wskazuje, czy należy użyć identyfikatora ze schematu odwzorowującego, aby w tabeli relacyjnej utworzyć klucz główny. Jeśli właściwości przypisana zostanie wartość `true`, wówczas jako typ danych którejś z kolumn ze schematu odwzorowującego należy wskazać `dt::type="id"`.
`TempFilePath`	Ciąg znaków wskazujący ścieżkę, w której należy tworzyć pliki tymczasowe. Jeśli właściwości nie zostanie przypisana żadna wartość, wówczas pliki tymczasowe będą tworzone w lokalizacji wskazywanej przez zmienną środowiskową `TEMP`. Właściwość nie ma znaczenia, jeśli właściwości `Transaction` nie zostanie przypisana wartość `true`.
`Transaction`	Właściwość o wartości domyślnej `false`, która wskazuje, czy zbiorcze ładowanie danych powinno zostać wykonane w ramach pojedynczej transakcji. Jeśli właściwości zostanie przypisana wartość `true`, wówczas przed załadowaniem danych do SQL Servera wszystkie operacje zostaną zapisane w pliku tymczasowym. Jeśli wystąpi błąd, cała operacja zbiorczego ładowania danych zostanie wycofana. Właściwości `Transaction` nie można przypisać wartości `true`, jeśli ładowane są dane binarne.
`XML Fragment`	Właściwość logiczna wskazująca, czy ładowane dane XML są fragmentem czy nie. **Fragment** to dokument XML, który nie posiada węzła głównego. Właściwości należy przypisać wartość `true`, jeśli XML jest fragmentem lub nie przypisywać jej żadnej wartości w przypadku, gdy XML nie jest fragmentem, ponieważ domyślną wartością właściwości jest `false`.

Pierwszą właściwością, której działanie należy zrozumieć, jest właściwość logiczna `Transaction`. Zwykle pozostawia się jej domyślną wartość `false`, aby zbiorcze ładowanie danych nie zachodziło w ramach jednej transakcji. W ten sposób zwiększona zostanie wydajność całej operacji, lecz z drugiej strony nie będzie można jej wycofać w przypadku, gdyby pojawił się jakiś błąd.

Kolejną właściwością jest logiczna właściwość `XMLFragment`. Jeśli przypisze się jej wartość `true`, `Bulkload` pozwoli na załadowanie fragmentów XML, czyli dokumentów XML bez węzła głównego.

Jeśli w tabeli zdefiniowano ograniczenia i trzeba ich przestrzegać w trakcie zbiorczego ładowania danych przez `Bulkload`, wówczas właściwości `CheckConstraints` należy przypisać wartość `true`. Domyślnie `Bulkload` wyłącza sprawdzanie ograniczeń, zyskując w ten sposób na wydajności. Bez względu na to, czy ograniczenia będą sprawdzane czy też nie, najpierw w schemacie odwzorowującym powinno się umieszczać klucze główne, a dopiero potem klucze obce.

Jeśli klucze obce mają być ignorowane, logicznej właściwości `IgnoreDuplicateKeys` należy przypisać wartość `true`. Jest to rozwiązanie przydatne w sytuacjach, gdy osoba generująca dane nie wie, jakie dane już znajdują się w tabeli docelowej i chcemy uniknąć sytuacji, gdy `Bulkload` kończy się niepowodzeniem z powodu wystąpienia zduplikowanych kluczy. `Bulkload` nie zatwierdzi wiersza ze zduplikowanym kluczem, lecz od razu przejdzie do przetwarzania następnego wiersza.

Wielu projektantów baz danych wykorzystuje kolumny identyfikatorów, aby zagwarantować unikatowość kluczy w tabeli. Czasami ładowany dokument XML posiada już element mogący spełniać rolę identyfikatora, który trzeba zastosować, zamiast wykorzystywać SQL Server do generowania wartości przy użyciu własnego algorytmu serwera. W tym celu właściwości `KeepIdentity` należy przypisać wartość `true`, która jest domyślną wartością właściwości. Należy przy tym pamiętać, że jest to wartość globalna, dlatego nie można zmusić SQL Servera do generowania w niektórych przypadkach własnego identyfikatora, a w pozostałych przypadkach używać identyfikatorów zawartych w danych XML.

Właściwość `KeepNulls` obiektu `Bulkload` domyślnie ma wartość `false`. Oznacza to, że `Bulkload` nie wstawi automatycznie wartości `null` w kolumnach, dla których w dokumencie XML zabrakło odpowiadającego atrybutu lub elementu. Jeśli właściwości zostanie przypisana wartość `true`, należy zachować szczególną ostrożność, ponieważ `Bulkload` zakończy swe działanie niepowodzeniem w przypadku, gdy występowanie wartości `null` w kolumnie będzie zabronione. W sytuacji, gdy właściwość będzie mieć wartość `true` i dla kolumny zdefiniowana będzie wartość domyślna, `Bulkload` i tak nie wstawi w kolumnie wartości domyślnej.

Kolejną interesującą właściwością `Bulkload` jest właściwość `ForceTableLock`, która powoduje zablokowanie tabeli na czas zbiorczego ładowania danych. Dzięki blokadzie można zwiększyć wydajność operacji ładowania, lecz w zamian inni użytkownicy nie będą mieli dostępu do tabeli. Domyślną wartością właściwości jest `false`, co oznacza, że `Bulkload` zakłada blokadę na tabelę przy każdej operacji wstawiania wiersza.

Jeżeli tabele docelowe jeszcze nie istnieją, `Bulkload` może odpowiednie tabele utworzyć. W tym celu właściwości `SchemaGen` należy przypisać wartość `true`. `Bulkload` odczyta typy danych ze schematu odwzorowującego i na ich podstawie automatycznie wygeneruje prawidłowy schemat bazy danych. Jeśli już istnieje tabela lub kolumna o takiej samej nazwie i należy ją usunąć i utworzyć na nowo, wówczas właściwości `SGDropTables` należy przypisać wartość `true`.

Jeden z kolejnych punktów opisuje sposób użycia Bulkload w środowisku zarządzanym. Bulkload obsługuje .NET i COM, dzięki czemu technologii zbiorczego ładowania można używać w obydwu środowiskach.

Odczytywanie danych XML z bazy danych — klauzula FOR XML

Klauzula FOR XML dodana do SQL Servera 2000 pozwala na odczytywanie danych relacyjnych w formacie XML bez konieczności przechowywania tych danych w formacie XML. Biorąc pod uwagę to, jak wielu administratorów baz danych chce przechowywać dane w formacie relacyjnym, a jednocześnie przenosić dane relacyjne do innych systemów w postaci XML ze względu na elastyczność i uniwersalny sposób obsługi języka XML, klauzula FOR XML jest niewątpliwie bardzo cennym narzędziem SQL Servera. W niniejszym podrozdziale opisane zostanie działanie klauzuli FOR XML zarówno z perspektywy serwera, jak i klienta, aby przedstawić sposób transformowania danych relacyjnych do formatu XML.

Klauzula FOR XML po stronie serwera

Zapewne wielu czytelników już używa klauzuli FOR XML, ponieważ jest to najprostszy sposób odczytywania danych relacyjnych z SQL Servera i uzyskiwania ich w formacie XML. W uproszczonej formie składnia klauzuli FOR XML rozszerzającej składnię zapytań przedstawia się następująco:

```
SELECT lista kolumn
FROM lista tabel
WHERE kryteria filtrowania
FOR XML RAW | AUTO | EXPLICIT [, XMLDATA] [, ELEMENTS]
    [, BINARY BASE64]
```

Na końcu podrozdziału przeanalizujemy zmiany, jakie zaszły w działaniu FOR XML w SQL Serverze 2005, na razie jednak skupimy się na najczęstszych przypadkach, w których używa się klauzuli. Pierwsze z tych przypadków to użycie klauzuli w trybach AUTO i RAW. Niektórzy użytkownicy korzystają również z trybu EXPLICIT, lecz znacznie rzadziej niż z pierwszych dwóch trybów. Najczęstszym powodem, dla których użytkownicy nie korzystają z trybu EXPLICIT, jest fakt, że AUTO i RAW w zupełności zaspokajają ich potrzeby. Kolejnym powodem jest to, że sposób użycia trybu EXPLICIT jest dość skomplikowany, o czym też się przekonamy. Jeśli wystarczy użycie pierwszych dwóch trybów, zalecamy ich wykorzystanie, ponieważ generowanie bardziej skomplikowanych struktur XML w trybie EXPLICIT sprawia, że wielu użytkowników rwie sobie przy tym włosy z głowy.

Tryb RAW

W trybie RAW klauzula FOR XML zwraca kolumny jako atrybuty i wiersze jako elementy row. Poniżej znajduje się przykład zastosowania klauzuli FOR XML RAW:

```
USE pubs
GO
SELECT * FROM Authors FOR XML RAW
```

Oto fragment wyników zwróconych przez zapytanie:

```
<row au_id="172-32-1176" au_lname="White" au_fname="Johnson"
phone="408 496-7223" address="10932 Bigge Rd." city="Menlo Park"
state="CA" zip="94025" contract="1"/>
<row au_id="213-46-8915" au_lname="Green" au_fname="Marjorie"
phone="415 986-7020" address="309 63rd St. #411" city="Oakland"
state="CA" zip="94618" contract="1"/>
<row au_id="238-95-7766" au_lname="Carson" au_fname="Cheryl"
phone="415 548-7723" address="589 Darwin Ln." city="Berkeley"
state="CA" zip="94705" contract="1"/>
<row au_id="267-41-2394" au_lname="O&apos;Leary" au_fname="Michael"
phone="408 286-2428" address="22 Cleveland Av. #14" city="San Jose"
state="CA" zip="95128" contract="1"/>
```

Jak widać, każdemu wierszowi odpowiada jeden element wiersza row, a dla każdej niepustej kolumny tworzony jest atrybut elementu wiersza. Jeśli odczytywane są dane binarne, trzeba dopisać klauzulę BINARY BASE64. Z kolei aby wraz z danymi XML uzyskać także schemat XML-Data, należy użyć klauzuli XMLDATA.

Tryb AUTO

W trybie AUTO klauzula FOR XML działa podobnie jak w trybie RAW, to znaczy każdy wiersz jest zwracany w postaci elementu, a wartości w kolumnach są zwracane jako atrybuty. Jedyna różnica wiąże się z nazwą elementu reprezentującego wiersz, którą w trybie AUTO jest nazwa tabeli. Spójrzmy zatem na wynik poniższego zapytania:

```
USE pubs
GO
SELECT * FROM Authors FOR XML AUTO
```

Skrócone wyniki prezentują się następująco:

```
<Authors au_id="172-32-1176" au_lname="White" au_fname="Johnson"
phone="408 496-7223" address="10932 Bigge Rd." city="Menlo Park"
state="CA" zip="94025" contract="1"/>
<Authors au_id="213-46-8915" au_lname="Green" au_fname="Marjorie"
phone="415 986-7020" address="309 63rd St. #411" city="Oakland"
state="CA" zip="94618" contract="1"/>
<Authors au_id="238-95-7766" au_lname="Carson" au_fname="Cheryl"
phone="415 548-7723" address="589 Darwin Ln." city="Berkeley"
state="CA" zip="94705" contract="1"/>
<Authors au_id="267-41-2394" au_lname="O&apos;Leary" au_fname="Michael"
phone="408 286-2428" address="22 Cleveland Av. #14" city="San Jose"
state="CA" zip="95128" contract="1"/>
<Authors au_id="274-80-9391" au_lname="Straight" au_fname="Dean"
phone="415 834-2919" address="5420 College Av." city="Oakland"
```

```
state="CA" zip="94609" contract="1"/>
<Authors au_id="341-22-1782" au_lname="Smith" au_fname="Meander"
phone="913 843-0462" address="10 Mississippi Dr." city="Lawrence"
state="KS" zip="66044" contract="0"/>
```

Nazwa tabeli wyznacza element dla każdego węzła, a wartości kolumn są atrybutami danego elementu. Zagnieżdżanie elementów jest uzależnione od kolejności w klauzuli SELECT, dlatego kolejności pól należy poświęcić szczególną uwagę. Ponadto w zapytaniu SELECT z klauzulą FOR XML nie można używać klauzuli GROUP BY, można natomiast wykonywać sortowanie przy użyciu klauzuli ORDER BY. Zamiast GROUP BY można używać zagnieżdżonych zapytań SELECT i uzyskać podobne wyniki, lecz takie rozwiązanie zmniejsza nieco wydajność działania. W przypadku wykonywania złączeń tryb AUTO odpowiednio zagnieździ zestaw wyników, co w większości przypadków będzie działaniem pożądanym. Jeśli natomiast wyniki nie mają być zagnieżdżane, trzeba skorzystać z trybu EXPLICIT i samemu nadać format danych XML. Na przykład jeśli trzeba wykonać złączenie danych z tabel Publishers oraz Titles i w danych XML wszystkie tytuły powinny być zagnieżdżane pod odpowiadającymi im wydawcami, należałoby wykonać następujący kod:

```
USE pubs
GO
SELECT Publishers.Pub_Name, Titles.Title, Titles.Price
  FROM Titles, Publishers WHERE Publishers.Pub_ID = Titles.Pub_ID
  FOR XML AUTO
```

Skrócone wyniki zapytania są następujące:

```
<Publishers Pub_Name="Algodata Infosystems">
  <Titles Title="The Busy Executive&apos;s Database Guide" Price="19.9900"/>
  <Titles Title="Cooking with Computers:
    Surreptitious Balance Sheets"
    Price="11.9500"/>
</Publishers>
<Publishers Pub_Name="New Moon Books">
  <Titles Title="You Can Combat Computer Stress!" Price="2.9900"/>
</Publishers>
<Publishers Pub_Name="Algodata Infosystems">
  <Titles Title="Straight Talk About Computers" Price="19.9900"/>
</Publishers>
```

Wraz z klauzulą FOR XML AUTO można także używać opcji ELEMENTS. Użytkownicy preferujący używanie elementów zamiast atrybutów mogą w ten sposób użyć trybu AUTO i uzyskać wyniki w postaci bazującej na elementach, a nie atrybutach. Osobiście wolimy stosować składnię bazującą na elementach, która zwiększa wprawdzie rozmiar danych XML przede wszystkim przez obecność znaczników otwierających i zamykających, jednak wynikowe dane XML są wówczas bardziej czytelne i zrozumiałe.

Tryb EXPLICIT

Ostatnim trybem jest EXPLICIT. Jak wskazuje nazwa trybu, umożliwia on sprawowanie pełnej kontroli nad sposobem generowania danych XML. Użytkownik musi opisać sposób, w jaki ma zostać ustrukturyzowany wynikowy dokument XML, a SQL Server wypełni tak opisany dokument odpowiednimi informacjami. Aby opisać dokument XML, trzeba użyć **tabeli uniwersalnej**. Tabela uniwersalna zawiera po jednej kolumnie

tabeli dla każdej wartości, która ma być zwracana, oraz dwa dodatkowe znaczniki, z których jeden unikatowo identyfikuje znaczniki w danych XML, a drugi identyfikuje relacje rodzic-potomek. Pozostałe kolumny opisują dane. Tabela 7.6 ilustruje uniwersalną tabelę dla trybu EXPLICIT.

Tabela 7.6. *Tabela uniwersalna*

Znacznik	Rodzic	Dyrektywa kolumny 1	Dyrektywa kolumny 2
1	Null	Dana	Dana
2	1	Dana	Dana
3	2	Dana	Dana

Dyrektywy opisują sposób, w jaki dane mają być wyświetlane w tabeli. Dyrektywy to specjalne polecenia, których używa się do wskazania SQL Serverowi sposobu parsowania danych. Format dyrektyw prezentuje się następująco:

```
Element!Znacznik!Atrybut!Dyrektywa
```

Poszczególne fragmenty dyrektywy są od siebie oddzielane znakiem wykrzyknika. Utwórzmy zatem prostą, przykładową tabelę, która będzie używać formatowania wskazanego powyżej. Załóżmy, że trzeba wyświetlić autorów, lecz wartość au_id powinna być atrybutem w wynikowych danych XML, a pozostałe danych powinny mieć charakter elementów. Takiego wyniku nie uzyskamy ani w trybie RAW, ani w trybie AUTO, ponieważ w żadnym z nich nie można uzyskać jednocześnie danych bazujących na atrybutach i elementach. Zapytanie zwracające dane w pożądanej przez nas postaci będzie mieć następującą postać:

```
SELECT 1 as Tag, NULL as Parent,
    au_id as [Authors!1!au_id], au_lname as [Authors!1]
    FROM Authors FOR XML EXPLICIT
```

Pierwszą rzeczą, która się rzuca w oczy, jest obecność kolumn Tag i Parent na liście kolumn. Kolumny Tag i Parent są wymagane do zidentyfikowania znacznika bieżącego elementu, którym jest liczba całkowita z przedziału od 1 do 255, oraz rodzica bieżącego elementu. W naszym przykładzie dane nie są zagnieżdżane, rodzic ma zawsze wartość null, a znacznikiem jest zawsze wartość 1, ponieważ zawsze odwołujemy się do tego samego rodzica. Dalej widać, że używana jest klauzula AS, która zmienia nazwę danych tak, aby opisać pożądane formatowanie XML. Nazwa nadawana kolumnie au_id wskazuje SQL Serverowi, że należy użyć elementu Authors, identyfikatora znacznika 1 oraz nazwy atrybutu. Pozostałe dane powinny mieć charakter elementów, zatem jako ich nazwa wskazywany jest jedynie element oraz nazwa znacznika. Na końcu wskazywana jest klauzula FOR XML EXPLICIT, ponieważ nie chcemy uzyskiwać w wyniku tabeli uniwersalnej opisującej strukturę XML, lecz samą przetworzoną strukturę XML. Zapytanie zwróci wyniki w następującej postaci:

```
<Authors au_id="409-56-7008">Bennet</Authors>
<Authors au_id="648-92-1872">Blotchet-Halls</Authors>
<Authors au_id="238-95-7766">Carson</Authors>
<Authors au_id="722-51-5454">DeFrance</Authors>
<Authors au_id="712-45-1867">del Castillo</Authors>
<Authors au_id="427-17-2319">Dull</Authors>
. . .
```

Jak widać, nazwisko autora jest zwracane jako wartość elementu. Można sprawić, by nazwisko było elementem zagnieżdżanym pod elementem Authors. W tym celu zapytanie należy nieco zmodyfikować, aby używać dyrektywy elementu w sposób pokazany poniżej:

```
SELECT 1 as Tag, NULL as Parent, au_id as [Authors!1!au_id],
    au_lname as [Authors!1!au_lname!element]
    FROM Authors FOR XML EXPLICIT
```

W tabeli 7.7 przedstawiono wszystkie dostępne dyrektywy oraz opis każdej z nich.

Tabela 7.7. *Dyrektywy FOR XML EXPLICIT*

Nazwa	Opis
cdata	Umieszcza dane w sekcji CDATA.
element	Wskazuje, że wskazany element należy zakodować (tak, by na przykład znak > został zapisany w postaci >) i zaprezentować jako podelement.
elementxsinil	Dyrektywy należy użyć, jeśli konieczne jest zwrócenie elementów wygenerowanych dla wartości null. W ten sposób zostanie utworzony element z atrybutem xsi:nil=TRUE.
ID	Pozwala na wskazywanie identyfikatora dla elementu. Każda dyrektywa ID wymaga, by w klauzuli FOR XML żądane były dane XMLDATA.
IDREF	Umożliwia atrybutom wskazywanie atrybutów będących identyfikatorami, aby umożliwić tworzenie połączeń wewnątrz dokumentów.
IDREFS	Działa podobnie do IDREF, ponieważ również pozwala na tworzenie połączeń wewnątrz dokumentów, lecz używa do tego celu struktury IDREFS zamiast IDREF.
hide	Ukrywa wyniki, aby nie umieszczać ich w generowanym dokumencie XML.
xml	Działa tak samo jak dyrektywa element, lecz nie wykonuje kodowania.
xmltext	Przydaje się do kolumn nadmiarowych OPENXML, ponieważ odczytuje kolumnę i dokleja ją do dokumentu.

Zajmijmy się teraz przykładem nieco bardziej skomplikowanym. Aby zwrócić wszystkich autorów wraz z tytułami ich książek oraz wysokością honorariów, należałoby najpierw wykonać zapytanie UNION ALL i połączyć ze sobą odpowiednie dane z poszczególnych tabel, a następnie trzeba by było zagnieździć wyniki tak, by wynikowa hierarchia XML odpowiednio odzwierciedlała autorów, honoraria i tytuły. Zanim jednak napiszemy odpowiednie zapytanie i uzyskamy wyniki, warto przytoczyć kilka wskazówek oraz opisać sposoby eliminowania błędów z zapytań FOR XML EXPLICIT. Przede wszystkim wielu użytkowników uzyskuje błąd, według którego nie otwarto jeszcze znacznika rodzica. Aby zidentyfikować błąd w zapytaniach FOR XML EXPLICIT, najprościej jest usunąć tę klauzulę z zapytania i sprawdzić postać uzyskiwanych wyników. W efekcie zwrócona zostanie uniwersalna tabela, w której można poszukać błędów. Najprostszym sposobem rozwiązania błędu związanego z brakiem otwartego znacznika rodzica jest zamieszczenie kolumny znacznika w klauzuli ORDER BY. Dzięki temu zyskamy pewność, że żaden znacznik nie zostanie przetworzony przed znacznikiem go poprzedzającym, co jest przyczyną występowania problemu.

Wróćmy jednak do zapytania, które powinno wygenerować wyniki w sposób opisany przed chwilą. W zapytaniu od razu będzie można zauważyć, że za pomocą kolumn `Tag` i `Parent` definiuje się kilka relacji rodzic-potomek. W zapytaniu zostanie również użyta sztuczka z klauzulą `ORDER BY`, aby zyskać pewność, że znaczniki rodzica znajdą się w dokumencie XML, zanim rozpoczęte zostanie przetwarzanie znaczników potomków.

```
SELECT 1 AS Tag, NULL AS Parent,
  Authors.au_fname AS [Authors!1!au_fname!element],
  Authors.au_lname AS [Authors!1!au_lname!element],
  NULL AS [Titleauthor!2!Royaltyper],
  NULL AS [Titles!3!Title!element]
FROM
Authors

UNION ALL

SELECT 2 AS Tag, 1 AS Parent,
  au_fname,
  au_lname,
  royaltyper,
  NULL
FROM Authors INNER JOIN Titleauthor ON
Authors.au_id = Titleauthor.au_id

UNION ALL

SELECT 3 AS Tag, 2 AS Parent,
  au_fname,
  au_lname,
  royaltyper,
  title
FROM Authors INNER JOIN Titleauthor ON Authors.au_id = Titleauthor.au_id
INNER JOIN Titles ON Titles.title_id = Titleauthor.title_id
ORDER BY [Authors!1!au_fname!element], [Authors!1!au_lname!element],
[Titleauthor!2!royaltyper], Tag
FOR XML EXPLICIT
```

Ucięta tabela uniwersalna prezentuje się następująco:

```
Tag    Parent    Authors!1!au_fname!element    Authors !1 !au_lnameelement
1      NULL      Abraham                       Bennet
2      1         Abraham                       Bennet
3      2         Abraham                       Bennet
1      NULL      Akiko                         Yokomoto
2      1         Akiko                         Yokomoto
3      2         Akiko                         Yokomoto
1      NULL      Albert                        Ringer
2      1         Albert                        Ringer
3      2         Albert                        Ringer
2      1         Albert                        Ringer
3      2         Albert                        Ringer
```

Z kolei ucięty zbiór wynikowy ma następującą postać:

```
<Authors>
  <au_fname>Abraham</au_fname>
  <au_lname>Bennet</au_lname>
```

```
    <Titleauthor Royaltyper="60">
      <Titles>
        <Title>The Busy Executive&apos;s Database Guide</Title>
      </Titles>
    </Titleauthor>
  </Authors>
  <Authors>
    <au_fname>Akiko</au_fname>
    <au_lname>Yokomoto</au_lname>
    <Titleauthor Royaltyper="40">
      <Titles>
        <Title>Sushi, Anyone?</Title>
      </Titles>
    </Titleauthor>
  </Authors>
  . . .
```

Już na pierwszy rzut oka widać, że `FOR XML EXPLICIT` daje znacznie większe możliwości, ale klauzula ta jest zdecydowanie bardziej skomplikowana w użyciu. Jeśli wystarczy użycie klauzuli `FOR XML AUTO` lub `RAW` i stosowanie trybu `FOR XML EXPLICIT`, wówczas napisanie odpowiedniego zapytania będzie znacznie łatwiejsze. Jednak w sytuacjach, gdy się okaże, że `FOR XML AUTO` i `RAW` nie spełniają oczekiwań, zawsze można posłużyć się klauzulą `FOR XML EXPLICIT`.

Klauzula FOR XML po stronie klienta

Dotąd pisane przez nas zapytania `FOR XML` były przetwarzane na serwerze. Jednak SQLXML pozwala także na przetwarzanie zapytań `FOR XML` po stronie klienta. Zamiast przesyłać sformatowane wyniki XML z serwera do klienta, SQL Server przesyła zestaw wierszy do SQLXML, a SQLXML formatuje wyniki już po stronie klienta. Przykład takiego rozwiązania zostanie przedstawiony w jednym z kolejnych punktów, poświęconym opisowi klas SQLXML.

Używanie szablonów

W trakcie wykonywania zapytań na SQL Serverze z wykorzystaniem SQLXML można używać szablonów. Szablony grupują technologie, które zostały opisane już wcześniej w tym rozdziale. W szablonach można używać zapytań SQL i XPath oraz trzeba używać adnotowanych schematów, które utworzono dla widoku XML. Schemat może być wpisywany bezpośrednio bądź ładowany z pliku. Aby zdefiniować szablon, trzeba utworzyć plik używający przestrzeni nazw `urn:schemas-microsoft-com:xml-sql`. Po wskazaniu przestrzeni nazw do szablonu można przekazywać zapytania SQL i XPath. SQLXML będzie przechowywać szablony w pamięci podręcznej, aby zwiększyć wydajność. Poniższy szablon wykonuje zapytanie SQL:

```
<Root><sql:query xmlns:sql=""urn:schemas-microsoft-com:xml-sql"">
SELECT * FROM Authors FOR XML AUTO</sql:query></Root>
```

Aby użyć zapytania XPath, składnię `sql:query` należy zastąpić składnią `sql:xpath-query`. Poniższy przykład odczytuje wszystkich autorów:

```
<Root><sql:xpath-query xmlns:sql=""urn:schemas-microsoft-com:xml-sql"">/Authors
</sql:xpath-query></Root>
```

Sposób użycia szablonów zostanie opisany w punkcie „Programowanie SQLXML w .NET i COM" w dalszej części niniejszego rozdziału.

Rozszerzenia klauzuli FOR XML

Słowo kluczowe `FOR XML` wykonujące transformację danych relacyjnych do postaci danych XML jest obsługiwane również w SQL Serverze 2005, lecz posiada także kilka dodatkowych rozszerzeń. Do głównych rozszerzeń w zakresie funkcji `FOR XML` należą obsługa typu danych `XML`, znaczne ułatwienie sposobu definiowania instrukcji `FOR XML EXPLICIT`, a także możliwość umieszczania XSD bezpośrednio w kodzie. Użytkownicy, którzy nie korzystali jeszcze z `FOR XML EXPLICIT`, szybko docenią to narzędzie. Zrozumienie składni klauzuli było wyzwaniem nawet dla doświadczonych programistów, a gdy zachodziła konieczność sformatowania bardziej skomplikowanych danych XML w SQL Serverze 2000, jedynym rozwiązaniem było użycie właśnie klauzuli `FOR XML EXPLICIT`.

Zwracanie wartości przy użyciu typu danych `XML` jest obsługiwane przez dodatkową dyrektywę `TYPE` klauzuli `FOR XML`. Gdy w wywołaniu zostanie umieszczona dyrektywa `TYPE`, wówczas zamiast generować dane XML i zwracać je w postaci tekstu SQL Server zwróci dane wynikowe typu `XML`. Dzięki temu na zbiorze danych wynikowych można wykonywać zapytania XQuery i odszukiwać pożądane informacje. Zbiór wynikowy można także przypisywać zmiennej albo wstawiać go do kolumny typu `XML`. Ponadto istnieje możliwość zagnieżdżania instrukcji `FOR XML` i generowania tak hierarchii zamiast używania `FOR XML EXPLICIT`. W ten sposób szybko można przekształcać dane relacyjne, a nawet dane używające typu danych XML, na wartość typu `XML`. Poniższy kod ilustruje sposób użycia nowej dyrektywy `TYPE` oraz przekazywania jej wyników do XQuery, o czym więcej powiemy w następnym rozdziale:

```
SELECT (SELECT * FROM authors FOR XML AUTO, ELEMENTS,
TYPE).query('count(//author)')
```

Nowy tryb `PATH` pozwala na wskazywanie przy użyciu XPath miejsca w hierarchii XML, w którym powinno się umieścić wartość kolumny. W ten sposób można zrezygnować z używania skomplikowanej i rozwlekłej klauzuli `FOR XML EXPLICIT` i zamiast niej generować bardziej złożone dane XML, korzystając z zagnieżdżonych instrukcji `FOR XML` i nowego trybu `PATH`. Poniżej znajduje się przykład użycia nowego trybu `PATH`. Przykładowy kod zmienia nazwę głównego elementu na `AuthorsNew`, a także tworzy nowy typ złożony o nazwie `Names`, który przechowuje imię i nazwisko.

```
SELECT au_fname "Names/FirstName", au_lname "Names/LastName"
FROM authors FOR XML PATH('AuthorsNew')
```

Ostatnim rozszerzeniem jest obsługa XSD bezpośrednio w kodzie, dostępna w trybach `RAW` i `AUTO`. Opcjonalnie można przekazywać dyrektywę `XMLSCHEMA` bezpośrednio w kodzie zapytania. Poniższy przykładowy kod ilustruje sposób użycia dyrektywy `XMLSCHEMA` oraz wyniki zwracane przez serwer:

```
SELECT * FROM authors FOR XML RAW('Authors'), XMLSCHEMA('urn:example.com')

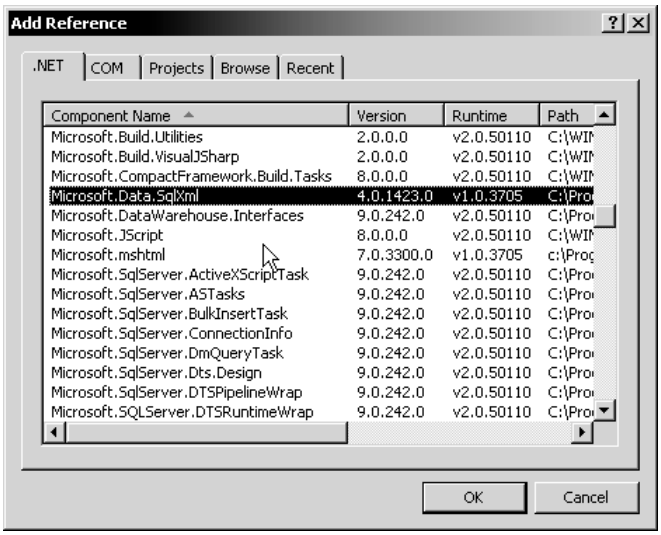
<xsd:schema targetNamespace="urn:example.com"
  xmlns:xsd=http://www.w3.org/2001/XMLSchema
  xmlns:sqltypes="http://schemas.microsoft.com/sqlserver/2004/sqltypes"
  elementFormDefault="qualified">
  <xsd:import namespace="http://schemas.microsoft.com/sqlserver/2004/sqltypes"

schemaLocation="http://schemas.microsoft.com/sqlserver/2004/sqltypes/sqltypes.xsd" />
  <xsd:element name="Authors">
    <xsd:complexType>
. . .
    </xsd:complexType>
  </xsd:element>
</xsd:schema>


<Authors xmlns="urn:example.com" au_id="172-32-1176"
    au_lname="White" au_fname="Johnson" phone="408 496-7223"
    address="10932 Bigge Rd." city="Menlo Park"
    state="CA" zip="94025" contract="1" />
```

Programowanie SQLXML w .NET i COM

SQLXML 4.0 udostępnia model obiektowy, dzięki któremu funkcje SQLXML można oprogramowywać zarówno w kodzie zarządzanym, jak i niezarządzanym. W kodzie niezarządzanym, na przykład w ADO, należy używać sterownika `SQLXMLOLEDB`. Dostawca ten używa nowego klienta SQL Native Client. Ze względu na ograniczony zakres książki programowanie SQLXML w kodzie niezarządzanym nie będzie opisywane. Zamiast tego skupimy się na kodzie zarządzanym. W Visual Studio do kodu zarządzanego należy dodać odwołanie do `Microsoft.Data.SqlXml`, jak na rysunku 7.2.



Rysunek 7.2. *Dodawanie odwołania do SQLXML*

Podzespół zarządzany przez SQLXML posiada cztery klasy: `SqlXmlCommand`, `SqlXmlParameter`, `SqlXmlAdapter` oraz `SqlXmlException`. Dzięki wymienionym klasom można wysyłać polecenia do SQL Servera i przetwarzać wyniki po stronie klienta, czyli na przykład przetwarzać instrukcje `FOR XML` albo wykonywać szablony XML. W kolejnych punktach opiszemy szczegółowo każdą z klas, ich metody i właściwości. Przedstawione zostaną również sposoby programowania SQLXML.

SqlXmlCommand

`SqlXmlCommand` to jedna z najważniejszych klas, która będzie najczęściej używana w trakcie korzystania z funkcji SQLXML. W tabeli 7.8 przedstawiono wszystkie metody klasy, natomiast tabela 7.9 prezentuje wszystkie właściwości `SqlXmlCommand`.

Tabela 7.8. *Metody klasy SqlXmlCommand*

Nazwa	Opis
`ClearParameters`	Czyści wszystkie parametry utworzone dla konkretnego obiektu polecenia.
`CreateParameter`	Tworzy obiekt `SqlXmlParameter`, za pośrednictwem którego można definiować nazwę i wartość parametru.
`ExecuteNonQuery`	Wykonuje zapytanie, lecz nie zwraca żadnej wartości. Rozwiązanie takie przydaje się, gdy trzeba wywołać diagram uaktualniający, który nie zwraca wartości.
`ExecuteToStream`	Wykonuje zapytanie i zwraca wyniki do istniejącego obiektu `Stream` przekazanego do metody.
`ExecuteStream`	Wykonuje zapytanie i zwraca wyniki w postaci nowego obiektu `Stream`.
`ExecuteXMLReader`	Wykonuje zapytanie i zwraca wyniki w obiekcie `XMLReader`.

SqlXmlParameter

Klasa `SqlXmlParameter` pozwala na przekazywanie do kodu parametrów. Klasa ma bardzo prostą konstrukcję, ponieważ posiada jedynie dwie właściwości: `Name` oraz `Value`. Użytkownik wskazuje nazwę `Name` parametru, na przykład `customerid`, oraz wartość `Value`, którą należy przypisać parametrowi. Obiekt `SqlXmlParameter` tworzy się przez wywołanie metody `CreateParameter` obiektu `SqlXmlCommand`.

SqlXmlAdapter

Obiekt `SqlXmlAdapter` zapewnia komunikację między zestawami danych .NET oraz funkcjami SQLXML. Konstruktor obiektu występuje w trzech postaciach. W pierwszej postaci konstruktor pobiera obiekt `SqlXmlCommand`, który jest wypełniony odpowiednimi informacjami pozwalającymi na nawiązanie połączenia ze źródłem danych. W drugiej postaci pobierane jest polecenie mające postać tekstową, typ polecenia w postaci obiektu `SqlXmlCommand` oraz ciąg znaków będący ciągiem połączenia. Ostatnia postać konstruktora jest bardzo podobna do postaci drugiej — jedynie zamiast ciągu znaków z tekstem polecenia pobierany jest obiekt `Stream`.

Tabela 7.9. *Właściwości klasy SqlXmlCommand*

Nazwa	Opis
`BasePath`	Ścieżka do katalogu bazowego, przydatna do definiowania w aplikacji ścieżek do plików arkuszy XSL, schematów odwzorowujących czy schematów XSD.
`ClientSideXML`	Właściwość logiczna, która w przypadku posiadania wartości `true` nakazuje SQLXML przekształcenie zestawów wierszy do postaci XML po stronie klienta.
`CommandStream`	Właściwość pozwala na definiowanie polecenia przy użyciu strumienia. Możliwość taką wykorzystuje się wówczas, gdy trzeba wykonać polecenie pochodzące z pliku.
`CommandText`	Odczytuje lub ustawia tekst polecenia, które ma zostać wykonane.
`CommandType`	Umożliwia ustawienie lub odczytanie typu polecenia przy użyciu następujących wartości: `SQLXMLCommandType.Diffgram`, `SQLXMLCommandType.Sql`, `SQLXMLCommandType.Template`, `SQLXMLCommandType.TemplateFile`, `SQLXMLCommandType.XPath` oraz `SQLXMLCommandType.UpdateGram`.
`Namespaces`	Pozwala na wskazywanie dla danych XML przestrzeni nazw w formacie `xmlns:x='urn:myschema:Mojaprzestrzeńnazw'`. Gdy używane są zapytania XPath kwalifikowane przestrzenią nazw, wówczas konieczne jest wskazanie przestrzeni nazw właśnie przy użyciu właściwości `Namespaces`.
`OutputEncoding`	Właściwość wskazuje rodzaj kodowania danych wynikowych. Domyślnie dane są kodowane w formacie UTF-8, lecz użytkownik może także wskazać formaty ANSI, Unicode lub inne prawidłowe wartości kodowania.
`RootTag`	Właściwość definiuje główny znacznik dokumentu XML, jeśli jest on wymagany. Zazwyczaj jest to ciąg znaków `root`.
`SchemaPath`	Wskazuje ścieżkę dostępu oraz nazwę pliku schematu. Jeżeli używane są ścieżki względne bazujące na wartości właściwości `BasePath`, wówczas SQLXML będzie szukać pliku w katalogu `BasePath`. Można także wskazywać ścieżki bezwzględne w postaci *c:\mojaaplikacja\mójschemat.xml*.
`XslPath`	Działa tak samo jak właściwość `SchemaPath`, lecz zamiast ścieżki do pliku schematu wskazuje ścieżkę dostępu do pliku transformacji XSL.

Utworzony obiekt posiada tylko dwie metody. Pierwszą z nich jest metoda `Fill`, do której przekazuje się zestaw danych ADO.NET. SQLXML wypełni przekazany zestaw danych danymi zwróconymi przez polecenie. Tak utworzony zestaw danych można następnie modyfikować przy użyciu standardowych funkcji obsługi zestawów danych oraz wywoływać drugą metodę obiektu o nazwie `Update`. Parametrem metody `Update` jest zestaw danych. W trakcie uaktualniania danych w tabeli SQLXML wykonuje blokowanie optymistyczne.

SqlXmlException

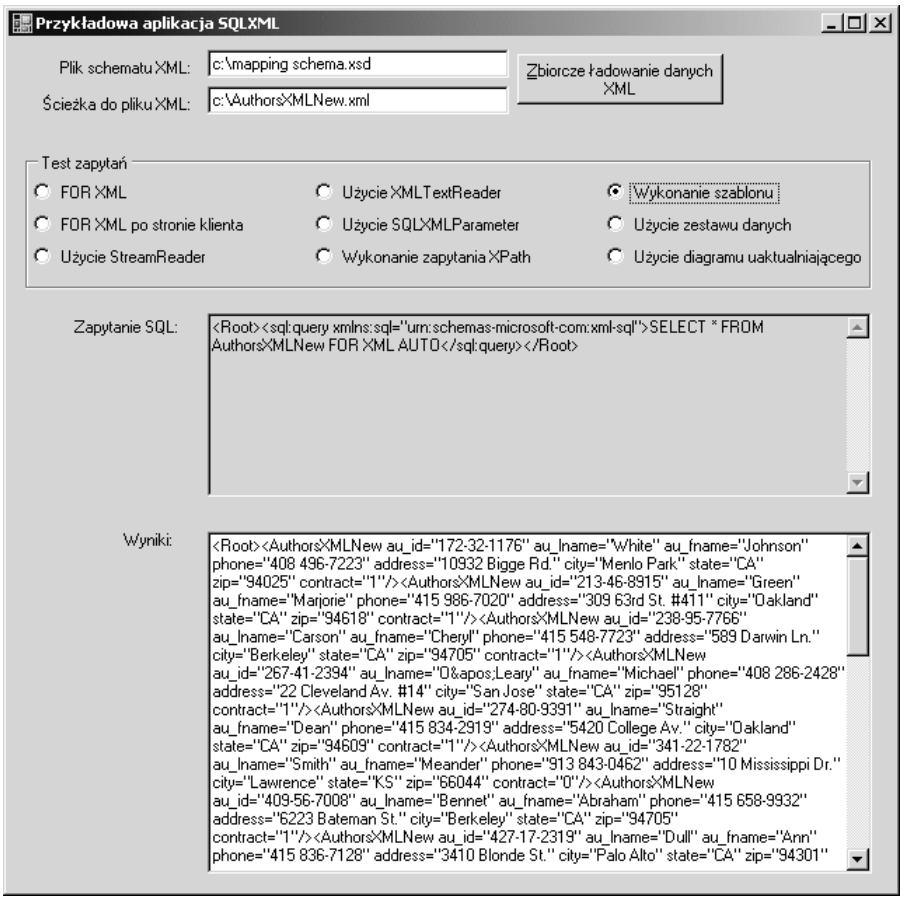
Obiekt `SqlXmlException` dziedziczy po standardowym obiekcie `SystemException` i umożliwia zwracanie wyjątków SQLXML jako części kodu. Obiekt posiada właściwość `ErrorStream`, której używa się do zwracania błędów. Poniższy kod wykorzystuje właściwość `ErrorStream` do wyświetlenia wszystkich błędów przechwyconych w ramach wyjątku:

```
. . .
    Catch ex As Microsoft.Data.SqlXml.SqlXmlException
        ex.ErrorStream.Position = 0
        Dim oSR As New System.IO.StreamReader(ex.ErrorStream)
        Dim strResult As String = oSR.ReadToEnd()
        System.Console.WriteLine(strResult)

    End Try
```

Przykładowe kody

Aby zilustrować sposób wykorzystania funkcji udostępnianych przez SQLXML, utworzymy przykładową aplikację. Nasza aplikacja będzie wykonywać zbiorcze ładowanie danych XML do bazy danych SQL, a następnie wykonywać na danych poszczególne funkcje opisywane w tym rozdziale, takie jak klauzula `FOR XML`, integracja zestawów danych, uruchamianie szablonów, używanie diagramów uaktualniających, a także wykonywanie przetwarzania po stronie klienta oraz używanie obiektów `XMLTextReader`. Interfejs użytkownika naszej przykładowej aplikacji widnieje na rysunku 7.3.



Rysunek 7.3. *Interfejs użytkownika przykładowej aplikacji*

Aplikacja posiada już odwołanie do SQLXML, zatem nie trzeba ponownie wykonywać tego kroku. Aby rozpocząć przetwarzanie danych, należy najpierw załadować dane XML do bazy danych i rozdrobnić je w poszczególnych kolumnach relacyjnych. W kodzie można by użyć `OPENXML`, lecz zamiast tego zdecydowaliśmy się zastosować `Bulkload` XML. Aby móc skorzystać z obiektu `Bulkload`, należy najpierw dodać w Visual Studio odwołanie do obiektu COM `Bulkload`. Odpowiedni komponent nosi nazwę *Microsoft SQLXML Bulkload 4.0 Type Library*. Następnie w aplikacji trzeba utworzyć obiekt `Bulkload`. Poniższe polecenie wykona odpowiednią operację:

```
Dim oXMLBulkLoad As New SQLXMLBULKLOADLib.SQLXMLBulkLoad4Class()
```

Następnie trzeba ustawić odpowiednie właściwości obiektu `Bulkload`. Nie możemy z góry zakładać, że tabela, do której dane mają zostać załadowane zbiorczo, już istnieje, dlatego właściwości `SchemaGen` przypisywana jest wartość `true`. Ponadto jeśli docelowa tabela już istnieje, powinno się ją usunąć, a zatem również właściwości należy przypisać wartość `true`. W przykładowej aplikacji definiujemy także inne właściwości, takie jak lokalizacja pliku z błędami, wskazanie, czy mamy do czynienia z fragmentem XML, a także informacja, czy należy utrzymać identyfikatory. Najważniejszą właściwością jest `ConnectionString`, ponieważ wskazuje ona obiektowi `Bulkload` sposób, w jaki należy połączyć się z serwerem. Po ustawieniu wszystkich właściwości aplikacja wywołuje metodę `Execute` obiektu `Bulkload` i przekazuje do niej plik schematu odwzorowującego oraz plik z danymi XML, które należy zbiorczo załadować. Pliki schematu odwzorowującego oraz danych XML znajdują się w katalogu przykładowej aplikacji. Pełen kod wykonujący opisane czynności przedstawia się następująco:

```
oXMLBulkLoad.ErrorLogFile = "c:\myerrors.log"
oXMLBulkLoad.SchemaGen = True
oXMLBulkLoad.KeepIdentity = False
oXMLBulkLoad.BulkLoad = True
oXMLBulkLoad.SGDropTables = True
oXMLBulkLoad.XMLFragment = True
oXMLBulkLoad.ConnectionString = strConnectionString
oXMLBulkLoad.Execute(txtXMLSchema.Text, txtXMLFile.Text)
```

Klauzula FOR XML — przetwarzanie po stronie serwera i po stronie klienta

Po zbiorczym załadowaniu danych XML można zacząć ich przetwarzanie. Dane, które aktualnie są rozdrobnione w poszczególnych kolumnach, można na przykład z powrotem odczytać w postaci XML. Do tego celu można użyć klauzuli `FOR XML`. Jak pamiętamy, SQLXML pozwala na przetwarzanie danych XML na serwerze oraz na kliencie, natomiast nasza aplikacja pozwoli na wybranie jednego z dwóch dostępnych rozwiązań. W kodzie użyjemy wspólnej metody, która będzie wykonywać wszystkie zapytania na przykładowych danych. Metoda będzie pobierać szereg parametrów, które będą między innymi wskazywać, czy przekazywane zapytanie jest zapytaniem SQL, szablonem czy też diagramem uaktualniającym. Pierwszą czynnością wykonywaną przez metodę jest utworzenie obiektu `SqlXmlCommand` przedstawionego poniżej. Jako ciąg zapytania użyty jest znany już zapewne większości użytkownikom standardowy ciąg połączenia w postaci `"Provider=SQLOLEDB;server=localhost;database=pubs;integrated security=SSPI"`.

```
Dim oSQLXMLCommand As New Microsoft.Data.SqlXml.SqlXmlCommand(strConnectionString)
```

Następnie jako typ polecenia trzeba wskazać typ odpowiedni dla przekazywanego zapytania. Dla standardowych zapytań SQL typem polecenia jest `Sql`, jak w poniższym kodzie:

```
oSQLXMLCommand.CommandType = Microsoft.Data.SqlXml.SqlXmlCommandType.Sql
```

Aby wysłać zapytanie `FOR XML` do serwera, w obiekcie `SqlXmlCommand` trzeba zdefiniować treść polecenia. Zapytanie jest przekazywane do metody, dlatego użyjemy do tego celu zmiennej `strQuery`:

```
'przypisanie zapytania
oSQLXMLCommand.CommandText = strQuery
```

Zapytanie `FOR XML` można przetwarzać na serwerze albo na kliencie, zatem właściwości `ClientSideXml` obiektu polecenia trzeba przypisać wartość `true` lub `false`. Wartość `true` będzie oznaczać, że przetwarzanie ma zostać wykonane na kliencie. Po zdefiniowaniu wartości właściwości można już wykonać zapytanie i przeanalizować wyniki. Poniższy kod używa obiektu `StreamReader` do odczytania wyników i wstawienia ich do pola tekstowego dla danych wynikowych. Do tego samego celu można użyć także obiektu `XMLTextReader`, który przedstawimy w dalszej części punktu.

```
'sprawdzenie, czy należy przetwarzać po stronie klienta
If bUseClientSide = True Then
    oSQLXMLCommand.ClientSideXml = True
End If

Dim oStream As System.IO.Stream
oStream = oSQLXMLCommand.ExecuteStream()

oStream.Position = 0
Dim oStreamReader As New System.IO.StreamReader(oStream)
txtResults.Text = oStreamReader.ReadToEnd()
oStreamReader.Close()
```

Jak można wywnioskować na podstawie przytoczonego kodu, użycie klauzuli `FOR XML` w SQLXML jest całkiem proste. Najtrudniejszą sprawą jest zapewnienie, że zapytanie `FOR XML` zostanie wykonane poprawnie i zwróci prawidłowe wyniki.

Sposób użycia obiektu XMLTextReader

W niektórych przypadkach zamiast odczytywać dane wynikowe zwracane przez zapytanie SQLXML przy użyciu obiektu `StreamReader` lepiej jest użyć obiektu `XMLTextReader`. W porównaniu z obiektem `StreamReader` obiekt `XMLTextReader` zapewnia szybszy dostęp do danych XML oraz jest bardziej elastyczny pod względem możliwości nawigowania w danych XML. Obiekt `XMLTextReader` parsuje dane XML i pozwala na wykonywanie na nich zapytań przy użyciu XPath. Aby móc użyć obiektu `XMLTextReader`, wystarczy zamienić wywołanie metody `ExecuteStream` na wywołanie metody `ExecuteXMLReader` obiektu `SqlXmlCommand`. Po utworzeniu obiektu czytającego dane można skorzystać z metod i właściwości obiektu czytającego dane XML i nawigować po danych XML. Poniższy kod wykorzystuje obiekt czytający dane XML i wyświetla uzyskane wyniki w interfejsie przykładowej aplikacji:

```
'użycie obiektu XMLTextReader
Dim oXMLTextReader As System.Xml.XmlTextReader
oXMLTextReader = oSQLXMLCommand.ExecuteXmlReader()
Dim strXML As String = ""

While oXMLTextReader.Read()
    'poziom elementu
    If oXMLTextReader.NodeType = XmlNodeType.Element Then
        strXML += "<" & oXMLTextReader.Name & ""
    ElseIf oXMLTextReader.NodeType = XmlNodeType.EndElement Then
        strXML += "</" & oXMLTextReader.Name & ">"
    End If

'wyszukanie atrybutów
    If oXMLTextReader.HasAttributes() Then
    Dim i As Integer = 0
    Do While (oXMLTextReader.MoveToNextAttribute())
        i += 1
        strXML += " " & oXMLTextReader.Name & "=" & oXMLTextReader.Value
        If oXMLTextReader.AttributeCount = i Then
            'ostatni atrybut — zamknięcie elementu
            strXML += " />"
        End If
    Loop

End If
End While

txtResults.Text = strXML
oXMLTextReader.Close()
```

Jak widać, w przypadku prostych operacji takich jak wyświetlanie danych XML użycie obiektu czytającego dane XML jest jedynie dodatkowym utrudnieniem, ponieważ w celu wyświetlenia danych trzeba najpierw wykonać ich parsowanie. Jednak gdy trzeba uzyskać dodatkowe informacje na temat danych XML, takie jak liczba atrybutów albo elementów, lub gdy konieczne jest użycie większego zakresu operacji nawigowania w danych, wówczas właśnie obiekt `XMLTextReader` najlepiej się sprawdzi.

Używanie parametrów w SQLXML

Aby móc użyć parametrów w SQLXML, należy utworzyć obiekt `SqlXmlParameter`. Nasze zapytanie musi wskazywać, że przekazywany będzie parametr. Konieczne jest również odpowiednie zdefiniowanie właściwości obiektu `SqlXmlParameter`. Poniższy fragment kodu prezentuje sposób użycia parametru w zapytaniach SQLXML:

```
strQuery = "SELECT * FROM " & strTable & " WHERE city = ? FOR XML AUTO, ELEMENTS"
. . .
Dim oSQLXMLParameter As Microsoft.Data.SqlXml.SqlXmlParameter
oSQLXMLParameter = oSQLXMLCommand.CreateParameter()
oSQLXMLParameter.Name = "city"
oSQLXMLParameter.Value = "Oakland"
. . .
```

Wykonywanie zapytań XPath lub zapytań SQL z szablonami

SQLXML pozwala na wykonywanie zapytań XPath lub SQL. W naszej przykładowej aplikacji użyjemy szablonu, aby wykonać zapytanie SQL oraz zwykłą instrukcję XPath dla zapytania XPath. W aplikacji można by również użyć szablonu dla zapytania XPath, lecz celem aplikacji jest przedstawienie sposobu użycia typu polecenia XPath. Poniższy kod ustawia zapytanie SQL z szablonem:

```
'załadowanie zapytania
strQuery = "<Root><sql:query xmlns:sql=""urn:schemas-microsoft-com:xml-sql"">SELECT
* FROM " & strTable & " FOR XML AUTO</sql:query></Root>"
```

Następnie jako typ polecenia trzeba wskazać szablon — dopiero wtedy możliwe będzie wykonanie zapytania SQL z szablonem. W aplikacji wskazujemy również element główny oraz ścieżkę do pliku adnotowanego schematu XSD:

```
oSQLXMLCommand.CommandType = Microsoft.Data.SqlXml.SqlXmlCommandType.Template
oSQLXMLCommand.SchemaPath = txtXMLSchema.Text
oSQLXMLCommand.RootTag = "ROOT"
```

Wyniki będą przetwarzane przy użyciu obiektu `StreamReader`, który był prezentowany już wcześniej, dlatego nie będziemy go przedstawiać po raz drugi.

Aby wykonać zapytanie XPath ponownie, trzeba odpowiednio zdefiniować zapytanie:

```
strQuery = "/AuthorsXMLNew[city='Oakland']"
```

W aplikacji zapytanie XPath jest używane w sposób bezpośredni, dlatego w obiekcie `SqlXmlCommand` jako typ zapytania trzeba wskazać XPath. Podobnie jak w przypadku zapytania SQL z szablonem, powinno się ustawić węzeł główny oraz wskazać ścieżkę dostępu do adnotowanego schematu XSD. Potem powtórnie trzeba użyć obiektu `StreamReader`, aby przetworzyć wyniki w sposób przedstawiony wcześniej.

```
oSQLXMLCommand.CommandType = Microsoft.Data.SqlXml.SqlXmlCommandType.XPath
oSQLXMLCommand.SchemaPath = txtXMLSchema.Text
oSQLXMLCommand.RootTag = "ROOT"
```

Interakcja z zestawem danych ADO.NET

SQLXML potrafi komunikować się z zestawem danych ADO.NET za pośrednictwem obiektu `SqlXmlAdapter`. Przy użyciu obiektu `SqlXmlAdapter` można wypełniać zestaw danych. Następnie można używać obiektu `Dataset` w taki sam sposób, w jaki używa się go w ADO.NET. Poniższy fragment kodu z przykładowej aplikacji tworzy zapytanie, wykonuje je przy użyciu obiektu `SqlXmlAdapter`, po czym wypełnia zestaw danych uzyskanymi wynikami. Do wypisania zwróconych wartości kod wykorzystuje obiekty obsługi strumieni:

```
strQuery = "SELECT * FROM " & strTable & " WHERE city = 'oakland' FOR XML AUTO,
ELEMENTS"
. . .

Dim oSQLXMLDataAdapter As New _
Microsoft.Data.SqlXml.SqlXmlAdapter(oSQLXMLCommand)
```

```
Dim oDS As New System.Data.DataSet()
oSQLXMLDataAdapter.Fill(oDS)
'wyświetlenie danych XML
Dim oMemStream As New System.IO.MemoryStream()
Dim oStreamWriter As New System.IO.StreamWriter(oMemStream)
oDS.WriteXml(oMemStream, System.Data.XmlWriteMode.IgnoreSchema)
oMemStream.Position = 0
Dim oStreamReader As New System.IO.StreamReader(oMemStream)
txtResults.Text = oStreamReader.ReadToEnd()
oMemStream.Close()
```

Programowanie diagramów uaktualniających

Ostatni fragment przykładowej aplikacji, którym się zajmiemy, dotyczy diagramów uaktualniających. Diagramy uaktualniające pozwalają na uaktualnianie danych na SQL Serverze przy użyciu istniejących dokumentów XML. Diagram uaktualniający jest tworzony w kodzie przy użyciu obiektu `StringBuilder`. Następnie jako typ polecenia wskazywany jest typ `UpdateGram`. Pozostała część kodu jest taka sama jak początkowy kod wykonujący polecenie i odczytujący wyniki, zatem ten fragment pominiemy.

```
Dim strUpdateGram As New System.Text.StringBuilder()
strUpdateGram.Append("<?xml version='1.0'?><AuthorsXMLNewupdate ")
strUpdateGram.Append("xmlns:updg='urn:schemas-microsoft-com:xml-updategram'>")
strUpdateGram.Append("<updg:sync
updg:nullvalue='nothing'><updg:before></updg:before>")
strUpdateGram.Append("<updg:after><AuthorsXMLNew au_id='123-22-1232'")
strUpdateGram.Append(" au_fname='Tom' state='WA' phone='425-882-8080'/>")
strUpdateGram.Append("</updg:after>")
strUpdateGram.Append("<updg:before><AuthorsXMLNew")
strUpdateGram.Append(" au_id='267-41-2394'/></updg:before>")
strUpdateGram.Append("<updg:after></updg:after>")
strUpdateGram.Append("<updg:before><AuthorsXMLNew")
strUpdateGram.Append(" au_id='238-95-7766'/></updg:before>")
strUpdateGram.Append("<updg:after><AuthorsXMLNew")
strUpdateGram.Append(" city='Oakland' phone='212-555-1212'/>")
strUpdateGram.Append("</updg:after></updg:sync></AuthorsXMLNewupdate>")

strQuery = strUpdateGram.ToString()
. . .

oSQLXMLCommand.CommandType = Microsoft.Data.SqlXml.SqlXmlCommandType.UpdateGram
```

Podsumowanie

W rozdziale opisano technologie obsługi języka XML dostępne w SQL Serverze 2000 oraz w SQL Serverze 2005. Do technologii tych należą `OPENXML`, rozszerzenia zapytań `FOR XML`, obiekt XML `Bulkload` oraz szereg rozwiązań SQLXML. W następnym rozdziale przedstawionych zostanie kilka obecnych w SQL Serverze 2005 dodatkowych rozwiązań w zakresie obsługi danych XML. Opisane zostaną również zmiany, jakim uległy niektóre rozwiązania zaprezentowane w tym rozdziale.