helion.pl
KSIĘGARNIA INTERNETOWA · WYDAWNICTWO

IDŹ DO
PRZYKŁADOWY ROZDZIAŁ
SPIS TREŚCI

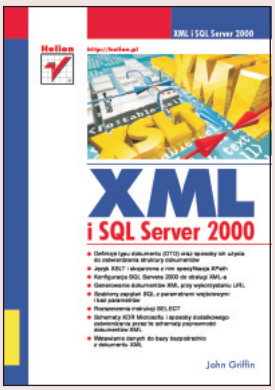
KATALOG KSIĄŻEK
KATALOG ONLINE
ZAMÓW DRUKOWANY KATALOG

TWÓJ KOSZYK
DODAJ DO KOSZYKA

CENNIK I INFORMACJE
ZAMÓW INFORMACJE O NOWOŚCIACH
ZAMÓW CENNIK

CZYTELNIA
FRAGMENTY KSIĄŻEK ONLINE

# XML I SQL Server 2000




Autor: John Griffin
Tłumaczenie: Jacek Baszkiewicz
ISBN: 83-7197-652-6
Tytuł oryginału: XML and SQL Server 2000
Format: B5, stron: 322

„XML i SQL Server 2000" jest idealnym źródłem informacji dla tych programistów SQL, którzy chcą poznać możliwości, jakie daje XML w SQL Serverze 2000. Jeśli jesteś twórcą stron WWW i chcesz korzystać z XML-a podczas tworzenia swoich projektów, ale obawiasz się złożoności jego standardów i zastosowań - książka ta pozwoli Ci opanować je bardzo szybko.

John Griffin zaczyna od przypomnienia podstaw XML-a, następnie przechodzi do gruntownego omówienia definicji typu dokumentu (DTD) oraz ich użycia do zatwierdzania dokumentów. Przedstawia wiele sposobów generowania dokumentów XML i manipulowania nimi w SQL Serverze 2000. Oprócz tego prezentuje język XSLT, specyfikację XPath oraz schematy XDR Microsoftu.

Liczne przykłady oraz metodyczne podejście do tworzenia dokumentów XML i wstawiania danych z dokumentów XML do bazy powodują, że książka ta powinna stać się lekturą obowiązkową dla każdego programisty poszukującego specjalistycznego, pełnego i szczegółowego przeglądu możliwości XML-a w SQL Serverze 2000.

Wydawnictwo Helion
ul. Chopina 6
44-100 Gliwice
tel. (32)230-98-63
e-mail: helion@helion.pl

# Spis treści


**O Autorze** ..... **9**

**Wprowadzenie** ..... **11**

**Rozdział 1. XML** ..... **15**
Początki XML-a ..... 16
Dokument XML ..... 17
Definicja typu dokumentu (DTD) ..... 27
Schematy XML ..... 42
Tematy zaawansowane ..... 58
Odpowiedzi do dwóchwiczeń z rozdziału ..... 63

**Rozdział 2. Arkusze stylów XSLT** ..... **67**
XSLT ..... 67
Czym są arkusze stylów ..... 69
Węzły i drzewa zamiast dokumentów ..... 73
Struktura arkuszy stylów ..... 78
Szablony ..... 81
Formatowanie liczb ..... 96
Przetwarzanie warunkowe ..... 99
Tryby ..... 104
Przykład — kwartalny raport podatkowy przedsiębiorstwa ..... 106

**Rozdział 3. IIS i katalogi wirtualne** ..... **115**
Katalogi wirtualne ..... 115
Program zarządzający katalogami wirtualnymi ..... 116
Tworzenie katalogów wirtualnych za pomocą programu zarządzającego katalogami wirtualnymi ..... 118
Model obiektowy zarządzania katalogami wirtualnymi ..... 126
Tworzenie katalogu wirtualnego za pomocą modelu obiektowego ..... 133

**Rozdział 4. Uzyskiwanie dostępu do SQL-a 2000 poprzez HTTP** ..... **135**
Architektura dwu-, trzy- i n-warstwowa ..... 136
Możliwości protokołu HTTP ..... 139
Encje ..... 141
Wykonywanie instrukcji SQL poprzez HTTP ..... 143
Wywoływanie plików szablonów poprzez HTTP ..... 150

Wywoływanie składowanych procedur ... 156
Uzyskiwanie dostępu do obiektów baz danych poprzez HTTP ... 158
Wysyłanie szablonów przez formularze HTML ... 158

**Rozdział 5. Tworzenie widoków XML ze schematami XDR ... 161**
Spojrzenie ogólne ... 162
Schematy XDR ... 162
BizTalk ... 178
Rozszerzenia do schematów XDR ... 180
Domyślne wartości atrybutów schematów ... 202
Używanie rozszerzonych schematów XDR w zapytaniach ... 204
Wydajność schematu i jego buforowanie ... 206

**Rozdział 6. Wykorzystanie zapytań XPath ... 207**
Ograniczenia implementacji ... 208
Różnice pomiędzy implementacją Microsoftu a specyfikacją W3C ... 209
Typy danych XPath ... 210
Ścieżki lokalizacji ... 214
Uzyskiwanie dostępu do obiektów baz danych poprzez HTTP ... 231

**Rozdział 7. FOR XML ... 235**
Składnia ... 236
Ograniczenia FOR XML ... 237
Tryb RAW ... 241
Tryb AUTO ... 242
Tryb EXPLICIT ... 247

**Rozdział 8. OPENXML ... 263**
Użycie OPENXML ... 263
Meta właściwości ... 276

**Dodatek A Schemat bazy danych Northwind ... 285**
Tabela Categories ... 287
Tabela CustomerCustomerDemo ... 287
Tabela CustomerDemographics ... 288
Tabela Customers ... 288
Tabela dtproperties ... 289
Tabela Employees ... 289
Tabela EmployeeTerritories ... 290
Tabela Order Details ... 291
Tabela Orders ... 291
Tabela Products ... 292
Tabela Region ... 293
Tabela Shippers ... 293
Tabela Suppliers ... 294

**Dodatek B Instrukcje XSLT z odniesieniami do miejsc w tekście, gdzie zostały użyte ... 295**

**Dodatek C Funkcje XPath i ich przykłady ..... 299**

**Dodatek D Typy danych Microsoft XML ..... 303**

**Dodatek E Zasoby ..... 305**

XML ..... 305
XSLT ..... 305
Przestrzenie nazw ..... 305
Schemat XML ..... 306
SQL Server 2000 ..... 306
Inne zasoby ..... 306

**Słowniczek ..... 307**

**Skorowidz ..... 315**

Rozdział 4.

# Uzyskiwanie dostępu do SQL-a 2000 poprzez HTTP

W początkowych trzech rozdziałach omówiliśmy specyfikację XML, język XSLT i niezbędne do konfigurowania wirtualnych katalogów macierzystych kroki w SQL Serverze. W tym rozdziale najpierw omówimy architekturę klient-serwer, aby wyjaśnić w jaki sposób współdziałają ze sobą różne składniki systemowe, serwery aplikacji, serwery baz danych i inne elementy. Następnie przyjrzymy się różnorodnym sposobom wykorzystania protokołu HTTP podczas wykonywania instrukcji SQL na SQL Serverze. Obejmuje to użycie plików szablonów w celu wygenerowania danych XML. Wykorzystanie protokołu HTTP poprzez URL uprości nasze zadania, ponieważ większość osób pracujących w przemyśle komputerowym dobrze zna ten proces.

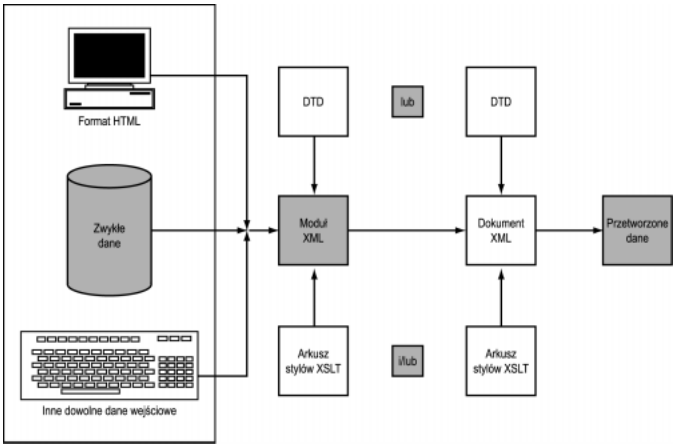
Ten rozdział omawia następujące tematy:

- Ogólna architektura klient-serwer w konfiguracjach dwu-, trzy- i n-warstwowych.
- Możliwości HTTP SQL Servera 2000.
- Encje w XML-u i adresach URL.
- Tworzenie dokumentów XML przez zapytania SQL Servera poprzez HTTP.
- Tworzenie dokumentów XML za pomocą plików szablonów XML.
- Tworzenie dokumentów XML za pomocą składowanych procedur.

Myślę, że nadszedł czas na zdefiniowanie *pliku szablonu* (ang. *template file*), o którym już tyle powiedzieliśmy. Nie jest to żaden nowy język, którego musiałbyś się uczyć, zrelaksuj się więc. Nie ma on również nic wspólnego z elementami szablonu XSLT, które przedstawione zostały w rozdziale 2., „Arkusze stylów XSLT". Szablony te są w zasadzie plikami XML zawierającymi jedną lub więcej instrukcji SQL. Szablony stosowane do bazy danych poprzez mechanizmy, o których dowiesz się z treści tego rozdziału pomagają w wygenerowaniu danych w formacie XML.

Spójrzmy po raz ostatni na diagram przetwarzania XML, którego używaliśmy w rozdziale 1., „XML" i rozdziale 2. (zobacz rysunek 4.1).

**Rysunek 4.1.**
*Przetwarzanie XML*




Tak, to naprawdę ostatni raz. Omówiliśmy już wszystkie składniki diagramu, które pozostaną składnikami modułu XML przez całą resztę tej książki. Modułem XML jest SQL Server 2000.

# Architektura dwu-, trzy- i n-warstwowa

Aby lepiej zrozumieć jak współdziałają ze sobą składniki typu klient-serwer, a jednocześnie lepiej zrozumieć możliwości i budowę systemu, musimy omówić pokrótce architekturę typu klient-serwer.

Każdy składnik systemu typu klient-serwer realizuje jedną lub kilka funkcji logicznych. Omówienie rozpoczniemy od zdefiniowania tych funkcji, a następnie pokażemy w jaki sposób są one rozproszone w obrębie całego systemu typu klient-serwer. Oto cztery funkcje będące podstawowymi blokami, z których budowane są wszystkie aplikacje:

- *Logika przechowywania danych*. Większość aplikacji musi przechowywać dane, czy jest to pojedynczy plik, czy potężna baza danych. Logika ta obejmuje takie tematy, jak wejście-wyjście i zatwierdzanie danych.
- *Logika dostępu do danych*. Jest to przetwarzanie, które zwykle ma formę zapytań SQL, wymagane, aby móc uzyskać dostęp do przechowywanych danych, (czy to pochodzących z wiersza poleceń analizatora zapytań, czy z składowanej procedury).
- *Logika aplikacji*. Jest to sama aplikacja, która może być prosta bądź złożona. Logika aplikacji nazywana jest również *regułami biznesowymi* firmy.
- *Logika prezentacji*. Jest to projekcja danych dla użytkownika oraz pobranie danych od niego.

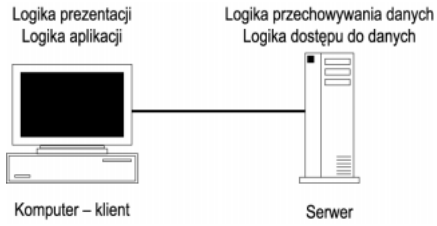
Po zdefiniowaniu tych *procesów funkcjonalnych*, możemy teraz przyjrzeć się temu, jak różne konfiguracje typu klient-serwer dzielą między siebie te funkcje i jaki mają

wpływ na rozbudowę. Zaczniemy od architektury dwuwarstwowej, potem przejdziemy do trójwarstwowej i n-warstwowej. W końcu omówimy pokrótce przykładową konfigurację SQL Servera i IIS.

# Dwuwarstwowa architektura typu klient-serwer

Najprostszą formą architektury typu klient-serwer jest struktura dwuwarstwowa, składająca się, uwierz lub nie, ze składnika klienta i składnika serwera. Jej dobrym przykładem jest statyczna witryna WWW (zobacz rysunek 4.2).

**Rysunek 4.2.** *Dwuwarstwowa architektura typu klient-serwer*

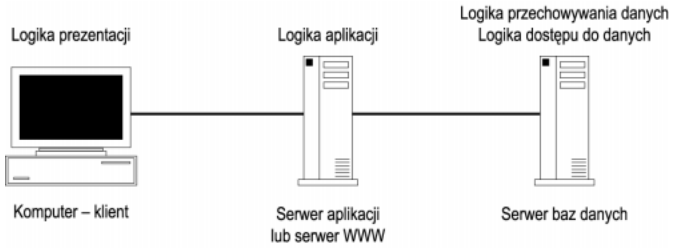



W tej konfiguracji warstwa klienta przyjmuje żądania użytkownika i realizuje logikę aplikacji, która tworzy zapytania do bazy danych i przesyła je do serwera. Serwer przyjmuje żądania, realizuje logikę dostępu do danych i przesyła wyniki do klienta. Klient przyjmuje wyniki i przedstawia je użytkownikowi.

# Trójwarstwowa architektura typu klient-serwer

Następny stopień to architektura trójwarstwowa. Zwróć uwagę na rysunek 4.3.

**Rysunek 4.3.** *Trójwarstwowa architektura typu klient-serwer*




Ten projekt wykorzystuje trzy różne punkty centralne. W tym przypadku klient jest odpowiedzialny za logikę prezentacji, serwer aplikacji odpowiada za logikę aplikacji, a zadaniem osobnego serwera baz danych jest realizacja logiki przechowywania i dostępu do danych. Przykładem tej struktury mogą być duże witryny WWW, gdzie serwery baz danych oddzielone są od serwerów WWW.

# N-warstwowa architektura typu klient-serwer

Ostatnim typem architektury (co nie znaczy, że najmniej ważnym), jest n-warstwowa architektura typu klient-serwer. Jest to w zasadzie konfiguracja dowolna (zobacz rysunek 4.4).




**Rysunek 4.4.** *N-warstwowa architektura typu klient-serwer*

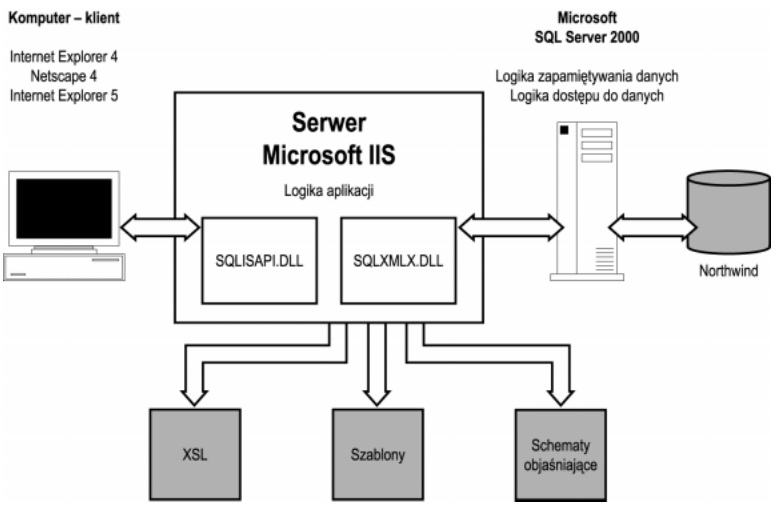
Na tym rysunku pojawiają się więcej niż trzy punkty centralne. Klient odpowiedzialny jest za logikę prezentacji, serwer (lub serwery) baz danych za logikę przechowywania i dostępu do danych, a logika aplikacji leży w gestii dwóch lub więcej różnych grup serwerów. W naszym przykładzie jeden z serwerów należących do warstwy logiki aplikacji jest serwerem WWW, natomiast drugi stricte serwerem aplikacji". Nie jest to wymóg konieczny tej architektury. Dopuszczalna jest dowolna kombinacja dwóch lub większej liczby typów serwerów aplikacji.

Podstawową cechą dającą przewagę n-warstwowej architekturze typu klient-serwer nad architekturą trójwarstwową (oraz trójwarstwowej nad dwuwarstwową) jest to, że *wyrównuje obciążenie* (ang. *load balancing*), rozdzielając przetwarzanie pomiędzy większą liczbę serwerów. Poza tym odpowiednia konfiguracja n-warstwowa umożliwia lepszą integrację z innymi składnikami, jest łatwiejsza wrozwoju, testowaniu i zarządzaniu.

# Trójwarstwowa architektura Microsoftu dla serwera IIS i SQL Server 2000

Patrząc na rysunek 4.5 zauważysz, że wygląda on nieco inaczej — ma zwrócić uwagę na pewne istotne szczegóły. Na pierwszy rzut oka widać, że jest to architektura trójwarstwowa. Nie ma tutaj niczego nowego; klient odpowiedzialny jest za logikę prezentacji, SQL Server 2000 (lub serwery) za logikę przechowywania i dostępu do danych, a IIS za logikę aplikacji. Nieco dokładniej chciałbym opisać w tym przypadku działanie serwera IIS.

Kiedy zapytanie typu URL jest przekazywane do serwera IIS, analizuje on wirtualny katalog macierzysty zawarty w URL-u i upewnia się, czy biblioteka *SQLISAPI.DLL* została zarejestrowana dla tego właśnie katalogu. Powinno to być zrobione podczas konfiguracji wirtualnego katalogu macierzystego, dokonywanej na jeden z dwóch sposobów przedstawionych w rozdziale 3., „IIS i katalogi wirtualne".




**Rysunek 4.5.** *Typowa architektura klient-serwer Microsoftu*

*SQLISAPI.DLL* wraz z innymi bibliotekami *DLL* zestawia połączenie z SQL Serverem zidentyfikowanym w wirtualnym katalogu macierzystym. Po zestawieniu połączenia i stwierdzeniu, że polecenie w adresie URL jest poleceniem XML, jest ono przekazywane do biblioteki *SQLXMLX.DLL*. Wykonuje ona to polecenie i zwraca wyniki. Wszystkie dostępne funkcje XML-a są zawarte w bibliotece *SQLXMLX.DLL*.

Jeśli weźmiesz pod uwagę to, co już omówiliśmy i cofniesz się do tej części rozdziału trzeciego, która opisywała zakładkę *Advanced*, to sam się przekonasz, dlaczego jej konfiguracja była taka ważna. Jeśli nie można odnaleźć pliku *SQLISAPI.DLL*, nic nie działa.

Jak to widać na rysunku 4.5, wszystkie pliki szablonów, pliki schematów i arkusze stylów XSLT znajdują się na serwerze IIS.

# Możliwości protokołu HTTP

Teraz powiemy sobie krótko o tym, czego możemy dokonać za pomocą protokołu HTTP. Dalsza część tego rozdziału poświęcona będzie szczegółowej analizie jego funkcji.

## Umieszczanie zapytania SQL bezpośrednio w adresie URL

Zwróć uwagę na następujący zapis:

```
http://IISServer/Northwind?sql=SELECT+*+FROM+Employees+FOR+XML+AUTO&root=root
```

Umieszczenie zapytania SQL bezpośrednio w adresie URL jest dość proste, nie sądzisz? Po adresie URL wskazującym katalog wirtualny *Northwind* wstaw znak zapytania oraz `sql=`, a następnie samo zapytanie SQL. Oddziel od siebie wszystkie słowa w zapytaniu znakiem plus (+). Parametr `&root` wyjaśnimy w jednym z następnych punktów, „Poprawne składniowo dokumenty, fragmenty i `&root`”.



`FOR XML AUTO` jest nowym rozszerzeniem instrukcji `SELECT`, pojawiającym się w SQL Serverze 2000. Instrukcja ta zostanie omówiona szerzej w rozdziale 8., „OPENXML". Na razie musisz wiedzieć, że zwraca ona wynik zapytania SQL w postaci dokumentu XML zamiast standardowego zbioru rekordów danych, do których prawdopodobnie jesteś przyzwyczajony. Możesz się również spotkać z zapisami `FOR XML RAW` i `FOR XML EXPLICIT`. Na razie jednak zaczekaj; dojdziemy do tego. Jeśli opuścisz instrukcję `FOR XML`, jak na przykład tutaj:

```
http://griffinj/Northwind?sql=select+*+from+Employees&root=root
```

to w przeglądarce ujrzysz wiadomość o błędzie, która będzie podobna do pokazanej na wydruku 4.1.

**Wydruk 4.1.** *Błąd wygenerowany na skutek braku instrukcji FOR XML*

```
<?xml version="1.0" encoding="utf-8" ?>
<root>
  <?MSSQLError HResult="0x80004005" Source="Microsoft XML Extension to SQL Server"
   Description="Streaming not supported over multiple column result"?>
</root>
```

# Określanie szablonu bezpośrednio w adresie URL

Oto przykład bezpośredniego określenia szablonu:

```
http://IISServer/Northwind?template=<ROOT+xmlns:sql="urn:schemas-microsoft-com:
➥xml-sql"><sql:query>SELECT+*+FROM+Employees+FOR+XML+AUTO
</sql:query></ROOT>
```

Widzisz teraz przykład pliku szablonu. Ma on formę dokumentu XML i zawiera przynajmniej jedną instrukcję SQL.

Szablony umożliwiają zwracanie danych w postaci poprawnych składniowo dokumentów XML. Jak zobaczysz wkrótce, nie jest to konieczne podczas określania instrukcji SQL bezpośrednio w adresie URL. Poza tym niektóre instrukcje SQL mogą być bardzo długie. Jeśli znajdowałyby się one w pliku szablonu, byłyby bardziej czytelne od tych zawartych w URL-u wraz z dodatkowymi znakami (znaki plus).

# Deklarowanie pliku szablonu w adresie URL

Zamiast pisać bardzo długie instrukcje w URL-u, co wygląda tak, jak to pokazywał przykład z poprzedniego punktu, możemy umieścić zapytanie SQL w pliku szablonu, a następnie odwoływać się do niego w adresie URL w następujący sposób:

```
http://IISServer/Northwind/TemplateVirtualName/template.xml
```

Pamiętaj, że katalog *TemplateVirtualName* został określony przy użyciu programu zarządzającego katalogami wirtualnymi.

Trzymanie użytkownika z dala od szczegółów dotyczących bazy danych zwiększa również bezpieczeństwo.

## Określanie zapytań XPath o plik schematu w adresie URL

Poniższy przykład pokazuje, jak to wygląda:

```
http://IISServer/Northwind/SchemaVirtualName/schemafile.xml/
➥Employee[@EmployeeID=6]
```

Katalog *SchemaVirtualName* został określony przy użyciu programu zarządzającego katalogami wirtualnymi, a `Employee[@EmployeeID=6]` jest zapytaniem XPath o plik *schemafile.xml*.

## Określanie obiektów baz danych bezpośrednio w adresie URL

Obiekty baz danych, takie jak tabele czy widoki, mogą zostać określone w adresie URL, a następnie w stosunku do nich można zastosować zapytanie XPath, które spowoduje wygenerowanie odpowiednich wyników. Pokazuje to następujący przykład:

```
http://IISServer/Northwind/dbobjectVirtualName/XpathQuery
```

Zapytanie XPath umieszczane jest jako ostatnia jednostka w adresie URL, bezpośrednio za *VirtualDirectoryName*.

# Encje

Kiedy omawialiśmy dokumenty XML w rozdziale 1., wspomnieliśmy o pewnych znakach specjalnych, które muszą być traktowane w sposób szczególny, ponieważ są one różnie interpretowane w zależności od miejsca, jakie zajmują w dokumencie. Aby uzyskać więcej informacji, zajrzyj pod adres *http://www.landfield.com/rfcs/rfc1738.html*.

Musimy również wziąć pod uwagę znaki specjalne występujące w adresie URL. Pewne znaki spełniają określone funkcje, kiedy używa się ich w URL-u.

## Encje w XML-u

Znaki wyszczególnione w tabeli 4.1 nie powinny być używane pomiędzy znacznikami w dokumencie XML. Znaki te mają specjalne znaczenie w XML-u i spowodują błędną interpretację dokumentu podczas analizy składniowej. Prawidłowe zamienniki, których powinno się używać w miejsce znaków, zawarto w tabeli.



Gruntowne omówienie tych znaków wraz z opisem ich działania znajdziesz w dokumencie RFC 2396. Ten RFC jest dostępny bezpłatnie na stronie *http://www.landfield.com/rfcs/rfc2396.html*.

**Tabela 4.1.** *Zamienniki znaków*

| Znak | Zamiennik |
|---|---|
| & (ampersand) | Użyj `&` |
| ' (apostrof) | Użyj `'` |
| < (znak mniejszości) | Użyj `<` |
| > (znak większości) | Użyj `>` |
| " (cudzysłów) | Użyj `"` |

Zwróćmy uwagę na przykładowy plik szablonu pokazany na wydruku 4.2. Zobaczysz, dlaczego te encje są potrzebne. Wydruk 4.3 prezentuje wynik.

**Wydruk 4.2.** *Encje w plikach szablonów*

```
<ROOT xmlns:sql="urn:schemas-microsoft-com:xml-sql">
  <sql:query>
    SELECT CustomerID, OrderDate, Freight
    FROM Orders
    WHERE Freight &gt; 800       <!-- &gt; zamiennik '>' -->
    FOR XML AUTO
  </sql:query>
</ROOT>
```

**Wydruk 4.3.** *Wynik zamiany encji*

```
<ROOT xmlns:sql="urn:schemas-microsoft-com:xml-sql">
  <Orders CustomerID="QUEEN" OrderDate="1996-12-04T00:00:00"
  Freight="890.78" />
  <Orders CustomerID="QUICK" OrderDate="1997-05-19T00:00:00"
  Freight="1007.64" />
  <Orders CustomerID="QUICK" OrderDate="1997-10-03T00:00:00"
  Freight="810.05" />
  <Orders CustomerID="SAVEA" OrderDate="1998-04-17T00:00:00"
  Freight="830.75" />
</ROOT>
```

# Encje w adresach URL

Kiedy wywołujesz zapytanie w adresie URL, musisz bardzo ostrożnie posługiwać się ze znakami wyszczególnionymi w tabeli 4.2. Wszystkie te znaki są wymagane w tym lub innym punkcie interpretacji adresu URL. Również dokument RFC 2396 zawiera szczegółowe omówienie tych znaków.

Oto przykład bezpośredniego zapytania SQL w URL-u:

```
http://IISServer/Northwind?sql=SELECT+*+FROM+Employees+WHERE+LastName+LIKE+
➥'D%'+FOR+XML+AUTO&root=root
```

Próbujemy tutaj wyszukać wszystkie informacje dotyczące pracowników, których nazwisko rozpoczyna się na literę „D”. Ponieważ znak `%` jest jednym ze specjalnych

**Tabela 4.2.** *Znaki specjalne w adresach URL*

| Znak | Opis | Wartość szesnastkowa |
|---|---|---|
| + | Oznacza spację (znaki spacji nie mogą być używane w URL-ach). | %20 |
| / | Oddziela katalogi i podkatalogi. | %2F |
| ? | Oddziela URL od parametrów. | %3F |
| % | Określa znaki specjalne. | %25 |
| # | Zaznacza zakotwiczenie zakładki. | %23 |
| & | Oddziela od siebie parametry określone w URL-u. | %26 |

znaków w URL-ach, próba umieszczenia tego zapytania bezpośrednio w URL-u spowoduje wystąpienie kilku błędów. Aby ominąć ten problem, musisz użyć zamienników znaków specjalnych w postaci wartości szesnastkowych, tak jak pokazano poniżej:

```
http://IISServer/Northwind?sql=SELECT+*+FROM+Employees+WHERE+LastName+LIKE+
➥'D%25'+FOR+XML+AUTO&root=root
```

Należy tutaj wspomnieć o jeszcze jednej rzeczy. Istnieją takie przypadki, gdy może się okazać niezbędne użycie kombinacji znaków specjalnych zarówno XML-a, jak i URL-a. Spójrz na poniższy przykładowy szablon, który może być określony bezpośrednio w URL-u. Sprawdź, czy potrafisz dostrzec błąd.

```
<ROOT xmlns:sql="urn:schemas-microsoft-com:xml-sql"><sql:query>SELECT+
➥CustomerID,OrderDate,Freight+FROM+Orders+WHERE+Freight+>+800+FOR+XML+AUTO
</sql:query></ROOT>
```

Mam nadzieję, że teraz już wiesz, że znak większości (>) nie może zostać użyty w tym miejscu, musimy więc zamienić go na wymaganą encję XML.

```
<ROOT xmlns:sql="urn:schemas-microsoft-com:xml-sql"><sql:query>SELECT+
➥CustomerID,OrderDate,Freight+FROM+Orders+WHERE+Freight+&gt;+800+FOR+XML+AUTO
</sql:query></ROOT>
```

Czy to zrobiłeś? Jeśli tak, to dobrze; jeśli jednak na tym poprzestałeś, to jeszcze za mało. Ampersand (&) to znak specjalny adresów URL, więc również powinien zostać zastąpiony. Zamień go na szesnastkową wartość %26.

```
<ROOT xmlns:sql="urn:schemas-microsoft-com:xml-sql"><sql:query>SELECT+
➥CustomerID,OrderDate,Freight+FROM+Orders+WHERE+Freight+%26gt;+800+FOR+XML+AUTO
</sql:query></ROOT>
```

# Wykonywanie instrukcji SQL poprzez HTTP

Dotąd pokazaliśmy wiele przykładów obrazujących sposób wykorzystania protokołu HTTP do pobierania dokumentów XML z SQL Servera. Nigdzie nie podaliśmy formalnej definicji składni. Taką definicję zawiera tabela 4.3. Omówimy również nieco więcej szczegółów dotyczących umieszczania zapytań w adresie URL — wbudowane procedury, szablony, wykorzystanie arkuszy stylów XSLT itd.

**Tabela 4.3.** *Wyjaśnienie składni HTTP*

| Słowo kluczowe | Opis |
|---|---|
| `iisserver` | Serwer IIS, do którego należy uzyskać dostęp, np. *www.newriders.com*. |
| `virtualroot` | Wirtualny katalog macierzysty skonfigurowany za pomocą programu do zarządzania katalogami wirtualnymi (graficznego lub programowego). |
| `virtualname` | Nazwa wirtualna zdefiniowana w czasie konfiguracji wirtualnego katalogu macierzystego. Może mieć ona jeden z trzech typów: szablon, schemat lub obiekt bazy danych. |
| `[/pathinfo]` | Informacja o ścieżce służąca lokalizacji plików szablonów bądź schematów, uzupełniająca informacje określone podczas konfiguracji nazwy wirtualnej. Nie jest potrzebna dla obiektów baz danych. |
| `[/XPathExpression]` | Określany dla plików schematów lub typów *dbobject*. |
| `?sql` | Ogranicza ciąg zapytania SQL. |
| `SqlString` | Zapytanie SQL lub nazwa procedury składowanej. Zwykle zawiera rozszerzenie `FOR XML`, chyba, że zwracane dane już są w formacie XML. Przykładem może być procedura zwracająca dane XML. |
| `?template` | Ogranicza ciąg zapytania SQL sformatowany jako dokument XML. |
| `param` | Jest to albo nazwa parametru, albo jedno z poniższych określeń: |
| | `contenttype` — Określa format zawartości zwracanego dokumentu, aby umożliwić przeglądarce WWW wybranie odpowiedniej metody wyświetlania. Określany w dwóch częściach: `contenttype` i `subtype`. Jest wysyłany w nagłówku HTTP, aby stał się dokumentem typu MIME. text/XML, text/HTML i image/gif oznaczają odpowiednio dokument XML, dokument HTML i obrazek GIF. |
| | `outputencoding` — Zestaw znaków użyty do stworzenia generowanego dokumentu XML. Domyślne ustawienie to UTF-8. Szablony określane bezpośrednio w adresie URL poprzez `template=` są tworzone w Unicode. Pliki szablonów mogą same określać `outputencoding`, ponieważ są dokumentami XML. |
| | `root` — Kiedy parametr jest określony, zwracane dane są otoczone nazwami elementów, nadanymi w celu wygenerowania poprawnego składniowo dokumentu XML. |
| | `xsl` — URL arkusza stylów XSLT używanego do przetwarzania zwracanych danych. Domyślnie wyjściowe dokumenty są kodowane w UTF-8, o ile to ustawienie nie jest odwołane przez instrukcję kodowania w pliku XSL. Jeśli określono parametr `outputencoding`, odwołuje on instrukcję kodowania z pliku XSL. |

A oto formalna składnia URL, umożliwiająca dostęp z poziomu SQL ISAPI:

```
http://iisserver/virtualroot/virtualname[/pathinfo][/XPathExpression]
[?param=value[&param=value]...n]
```

lub:

```
http://iisserver/virtualroot?{sql=SqlString | template=XMLTemplate}
[&param=value[&param=value]...n]
```



Kodowanie znaków jest określone za pomocą atrybutu `encoding=` w deklaracji XML. Specyfikacja XML wyraźnie informuje o tym, że XML korzysta z ISO 10646, międzynarodowego, standardowego zakresu znaków 31-bitowych, obejmującego większość języków świata. Planuje się, że będzie on nadzbiorem Unicode; można go znaleźć pod adresem *http://www.iso.ch*.

Specyfikacja mówi (2.2): „Wszystkie procesory XML muszą przyjmować kodowanie UTF-8 i UTF-16 standardu ISO 10646...". UTF-8 jest kodowaniem Unicode na znaki 8-bitowe: grupa pierwszych 128 z nich jest taka sama, jak w ASCII; reszta używana jest do kodowania Unicode w sekwencjach od 2 do 6 bajtów. UTF-8 w swojej jednooktetowej formie to to samo, co ISO 646 IRV (ASCII), możesz więc nadal używać ASCII dla języka angielskiego lub innego języka nieakcentowanego wykorzystującego alfabet łaciński. Zapamiętaj, że kodowanie UTF-8 nie jest kompatybilne z ISO8859-1 (ISO Latin-1) po dziesiętnej wartości 126 (koniec ASCII). UTF-16 jest podobne do UTF-8, ale ze schematem reprezentacji następnych 16 płaszczyzn znaków 64k jako dwóch znaków 16-bitowych.

Bez względu na zastosowane kodowanie, każdy znak zestawu ISO 10646 można przedstawić za pomocą dziesiętnego lub szesnastkowego ekwiwalentu tego ciągu bitowego. Nie jest więc ważne, jakiego zestawu znaków używasz, ponieważ zawsze możesz określić poszczególne znaki używając &#dddd; (kodowania znaków dziesiętnych) lub &#xHHHH; (kodowania znaków szesnastkowych dużymi literami). Zarówno terminologia, jak i te cyfry mogą nieco wprawiać w zakłopotanie: zwróć uwagę na słownik pojęć ISO 10646 dostępny pod adresem *http://www.cns-web.bu.edu/djohnson/web_files/i18n/ISO-10646.hmtl*[1].

# Poprawne składniowo dokumenty, fragmenty i &root

Jak obiecałem, wyjaśnimy teraz parametr `&root=root` użyty w kilku URL-ach i plikach szablonów w poprzednim podrozdziale.

Wróć myślami do naszej dyskusji na temat poprawnych składniowo dokumentów XML. Jednym z warunków, który musi być spełniony, aby dokument XML został uznany za poprawny, jest posiadanie jednego elementu najwyższego poziomu. Przypomnij sobie nasz dokument `RESUMES`, pokazany w poniższym przykładzie:

```
<RESUMES xmlns='http://www.myorg.net/tags'>
   <PERSON PERSONID="p1">
         <NAME>
...
</RESUMES>
```

Nasz dokument zawiera jeden element najwyższego poziomu `<RESUMES>`. Chociaż są również inne wymagania w stosunku do poprawnych składniowo dokumentów, to jeśli ten jeden nie jest spełniony — dokument nie przechodzi pozytywnie testu.

Jak to się ma do parametru `&root` w adresie URL? Parametr `&root` określa nazwę elementu głównego dokumentu. Wynikiem jest dokument XML wraz z tym najbardziej znaczącym elementem najwyższego poziomu. Zwróćmy uwagę na pewne przykłady wykorzystujące części kodu, który podaliśmy wcześniej:

```
http://IISServer/Nnorthwind?sql=SELECT+*+FROM+Employees+FOR+XML+AUTO&root=root
```

[1] Więcej o kodowaniu polskich znaków diakrytycznych można znaleźć na Polskiej Stronie Ogonkowej (*http://www.uci.agh.edu.pl/ogonki/*) — *przyp. red.*

Stosujemy tutaj zapis `&root=root`. Stworzony zostanie dokument zawierający pojedynczy element najwyższego poziomu, którym jest `<ROOT>`. Spodziewasz się, że w następnym przykładzie zwrócony zostanie fragment dokumentu, ponieważ nie został określony element główny. Lecz otrzymasz wiadomość o błędzie: „Tylko jeden element najwyższego poziomu może pojawić się w dokumencie XML". Ponieważ brakuje elementu głównego (`root`), wszystkie elementy odnoszące się do pracowników są traktowane jako elementy najwyższego poziomu, co nie jest dopuszczalne.

```
ttp://IISServer/Northwind?sql=SELECT+*+FROM+Employees+FOR+XML+AUTO
```

Te same warunki muszą być spełnione w wypadku plików szablonów. Załóżmy, że dysponujemy plikiem szablonu przedstawionym na wydruku 4.4.

**Wydruk 4.4.** *Plik szablonu bez elementu głównego*

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

Użyjemy tego szablonu w następującym URL-u:

```
http://IISServer/Northwind/TemplateVirtualName/template.xml
```

Plik szablonu dostarczy pojedynczego elementu najwyższego poziomu poprzez deklarację elementu `<ROOT>`.

Aby się upewnić, że naprawdę rozumiemy deklarację elementu głównego, zmienimy parametr w pliku szablonu na `root=EMPS`. Poniższy fragment dokumentu wynikowego pokazuje, jak zmieni się element główny dokumentu.

```
<EMPS xmlns:sql="urn:schemas-microsoft-com:xml-sql">
      <Employees EmployeeID="1" LastName="Davolio" ...
...
</EMPS>
```

# Zapytania do wielu tabel

Kierowanie pytań do wielu tabel pociąga za sobą skutki, które musisz uważnie rozpatrzyć, jeśli chcesz tworzyć dokumenty zachowujące właściwą kolejność elementów. Można przyjąć następującą zasadę: „Kolejność, w jakiej podawane są tabele w zapytaniu SQL określa kolejność zagnieżdżania elementów". Przyjrzymy się tabelom zamówień (*Orders*) i pracowników (*Employees*) w bazie Northwind w dwóch przykładach (zobacz też dodatek A, „Schemat bazy danych Northwind").

Oto pierwsze zapytanie:

```
http://iisserver/Northwind?sql=SELECT+TOP+2+Orders.OrderID,+Employees.
LastName,+Orders.ShippedDate+FROM+Orders,+Employees+WHERE+Orders.
EmployeeID=Employees.EmployeeID+Order+by+Employees.EmployeeID,
OrderID+FOR+XML+AUTO&root=ROOT
```

Zwróci ono wynik pokazany na wydruku 4.5.

**Wydruk 4.5.** *Wynik zapytania skierowanego do wielu tabel*

```
<?xml version="1.0" encoding="utf-8" ?>
<ROOT>
  <Orders OrderID="10258" ShippedDate="1996-07-23T00:00:00">
    <Employees LastName="Davolio" />
      </Orders>
      <Orders OrderID="10270" ShippedDate="1996-08-02T00:00:00">
    <Employees LastName="Davolio" />
  </Orders>
</ROOT>
```

Zróbmy teraz to samo z trzema tabelami, dodając tabelę ze szczegółami zamówień (*Order Details*).

```
http://iisserver/Nwind?sql=SELECT+TOP+2+Orders.OrderID,+Employees.
LastName,+Orders.ShippedDate,+[Order+Details].UnitPrice,+[Order+
Details].ProductID+FROM+Orders,+Employees,+[Order+Details]+WHERE+
Orders.EmployeeID=Employees.EmployeeID+AND+Orders.OrderID=[Order+
Details].OrderID+Order+by+Employees.EmployeeID,Orders.OrderID+FOR+
XML+AUTO&root=ROOT
```

Wydruk 4.6 pokazuje wynik tego zapytania

**Wydruk 4.6.** *Wynik zapytania skierowanego do trzech tabel*

```
<?xml version="1.0" encoding="utf-8" ?>
<ROOT>
  <Orders OrderID="10258" ShippedDate="1996-07-23T00:00:00">
    <Employees LastName="Davolio">
      <Order_x0020_Details UntPrice="15.2" ProductID="2" />
      <Order_x0020_Details UntPrice="17" ProductID="5" />
    </Employees>
  </Orders>
</ROOT>
```

Wydruk 4.7 pokazuje wynik, jaki można otrzymać po wykonaniu tego samego zapytania SQL, ale z przesuniętym elementem `Employees.LastName` na koniec.

**Wydruk 4.7.** *Wynik umieszczenia Employees.LastName na pozycji ostatniego elementu*

```
<?xml version="1.0" encoding="utf-8" ?>
<ROOT>
  <Orders OrderID="10258" ShippedDate="1996-07-23T00:00:00">
     <Order_x0020_Details UntPrice="15.2" ProductID="2">
       <Employees LastName="Davolio" />
     </Order_x0020_Details>
     <Order_x0020_Details UntPrice="17" ProductID="5" />
       <Employees LastName="Davolio" />
     <Order_x0020_Details>
  </Orders>
</ROOT>
```

Chcę, abyś tutaj zwrócił uwagę na kluczowy problem, jakim jest zależność występowania elementów w wynikowym dokumencie XML od kolejności umieszczania ich w zapytaniach SQL. Jest to szczególnie widoczne w różnicach pomiędzy wydrukami 4.6 i 4.7.

# Przekazywanie parametrów

Możliwe jest przekazywanie parametrów do zapytań SQL w adresach URL. Proces ten znany jest jako zastępowanie *w czasie wykonywania* (ang. *run-time*), w przeciwieństwie do zastępowania *w czasie projektowania* (ang. *design-time*). W pierwszym przypadku do określenia miejsca, w którym parametr ma być zastąpiony w czasie wykonywania, używamy symbolu wypełnienia. Obszarem zablokowanym jest znak `?`, który musi być określony w URL-u jako `%3F`. Oto przykład:

```
http://iisserver/Northwind?sql=SELECT+TOP+4+OrderID+FROM+Orders+WHERE+
➥EmployeeID=%3F+FOR+XML+AUTO&EmployeeID=5&root=ROOT
```

Wydruk 4.8 jest dokumentem wynikowym.

**Wydruk 4.8.** *Wynik zastąpienia parametrów w naszym URL-u*

```
<?xml version="1.0" encoding="utf-8" ?>
<ROOT>
  <Orders OrderID="10248" />
  <Orders OrderID="10254" />
  <Orders OrderID="10269" />
  <Orders OrderID="10297" />
</ROOT>
```

Przekazywanie wielu parametrów odbywa się poprzez wstawienie większej liczby oddzielonych od siebie pozycji ze znakami zapytania. Ten URL przekazuje dwa parametry:

```
http://iisserver/Northwind?sql=SELECT+TOP+4+OrderID+FROM+Orders+WHERE+
➥EmployeeID=%3F+AND+CustomerID=%3F+FOR+XML+AUTO&EmployeeID=5&CustomerID=
➥'VINET'&root=ROOT
```

Może to nie wygląda na duży problem i pewnie takim nie jest dla Ciebie, ale poczekaj do chwili, gdy zaczniemy określać w URL pliki szablonów, które mają zaprojektowane zapytania o parametry. Dojdziemy do tego w krótkim czasie.

# Kluczowe słowo XSL

Teraz przejdziemy do wykorzystania tego, o czym mówiliśmy w rozdziale 2. Korzystanie z arkuszy stylów XSLT pozwala na uzyskanie elastyczności potrzebnej do manipulowania wytwarzanym wyjściem XML. Możemy stworzyć „w locie" plik HTML w celu jego natychmiastowego (bądź późniejszego) wyświetlenia lub możemy zamienić zwracany dokument XML na inny, aby go dalej przetwarzać. To ostatnie zdarza się częściej niż mógłbyś przypuszczać, jak w przypadku powiązanych wiadomości EDI (Electronic Data Interchange). Z następującego zapytania SQL stwórzmy stronę HTML:

```
http://iisserver/Northwind?sql=SELECT+TOP+4+OrderID,EmployeeID,Shipname+
➥FROM+Orders+WHERE+EmployeeID=5+FOR+XML+AUTO&xsl=order.xsl &root=ROOT
```

W tym przykładzie plik XSL położony jest w wirtualnym katalogu macierzystym. Wynikowy dokument XML pokazano na wydruku 4.9, a arkusz stylów na wydruku 4.10.

**Wydruk 4.9.** *okument XML zawierający informacje o zamówieniach*

```
<?xml version="1.0" encoding="utf-8" ?>
<ROOT>
  <Orders OrderID="10248" EmployeeID="5" Shipname="Vins et alcools Chevalier" />
  <Orders OrderID="10254" EmployeeID="5" Shipname="Chop-Suey Chinese" />
  <Orders OrderID="10269" EmployeeID="5" Shipname="White Clover Markets" />
  <Orders OrderID="10297" EmployeeID="5" Shipname="Blondel pere et fils" />
</ROOT>
```

**Wydruk 4.10.** *Arkusz stylów XSLT stosowany do informacji o zamówieniach*

```
<?xml version='1.0'?>
<xsl:stylesheet xmlns:xsl='http://www.3.org/XSL/Transform/1.0'>
<xsl:output media-type="text/html" />

  <xsl:template match="/">
    <HTML>
    <BODY>
      <TABLE width='400' border='1'>
      <TR>
      <TD><B>Order ID</B></TD>
      <TD><B>Ship Name</B></TD>
      </TR>
      <xsl:apply-templates/>
      </TABLE>
    </BODY>
    </HTML>
</xsl:template>

<xsl:template match="Orders">
  <TR>
    <TD>
      <xsl:value-of select="@OrderID" />
    </TD>
    <TD>
      <xsl:value-of select="@Shipname" />
    </TD>
  </TR>
</xsl:template>
</xsl:stylesheet>
```

Wynikiem jest prosta tabela składająca się z numeru identyfikacyjnego zamówienia i nazwy przewoźnika, co pokazano na rysunku 4.4.

**Tabela 4.4.** *Wynikowa tabela HTML*

| Order ID | Shipname |
|---|---|
| 10248 | Vins et alcools Chevalier |
| 10254 | Chop-Suey Chinese |
| 10269 | White Clover Markets |
| 10297 | Blondel pere et fils |

Również w tym przypadku wynik może wydawać się prosty, ale potencjalnie może być dosyć skomplikowany. Dane mogą być wyszukiwane z bazy i prezentowane użytkownikowi poprzez przeglądarkę w czasie rzeczywistym. Mogą być generowane strony statyczne i przechowywane do przeglądania, jeśli klient zgłosi taką potrzebę. W tym przypadku arkusz stylów XSLT funkcjonuje jak szablon HTML (łatwiej jest wykorzystywać szablony niż pisać ponownie dokumenty HTML). Dokumenty e-commerce przesyłane między firmami mogą być tworzone z istniejących zapytań bez konieczności modyfikacji samego zapytania. Arkusz stylów XSLT może manipulować danymi w dowolny sposób w celu stworzenia nowego, odpowiedniego dokumentu XML.

Naszą uwagę skupimy teraz na użyciu plików szablonów do generowania dokumentów XML. Omówimy ten temat jeszcze bardziej szczegółowo, niż to zrobiliśmy do tej pory.

# Wywoływanie plików szablonów poprzez HTTP

Jeśli wrócisz do niektórych zapytań stworzonych w poprzednim podrozdziale, stwierdzisz, że są one czasami trudne do odczytania. Spójrz na zapytanie, które wygenerowało kody pokazane na listingach 4.6 i 4.7 i powiedz mi natychmiast, do czego służą. Wątpię w to, czy potrafisz.

Pliki szablonów posiadają te same możliwości, co zapytania SQL wpisane bezpośrednio w adresy URL. Pliki szablonów mogą realizować następujące funkcje:

- określać zapytania SQL lub zapytania XPath,
- definiować parametry, które mogą być przekazywane do tych zapytań,
- określać element najwyższego poziomu (główny) dokumentu XML,
- deklarować przestrzenie nazw,
- określać arkusz stylów XSLT stosowany do wyników.

Dodatkowymi zaletami plików szablonów są: łatwość ich odczytu i, jak mówią niektórzy, również łatwość ich tworzenia. Dodatkowo, pozbawiają one użytkownika dostępu do szczegółów dotyczących bazy danych, co zwiększa bezpieczeństwo. Edycja pliku przez użytkownika może się okazać niemożliwa, ale ponieważ może on zobaczyć adres URL, może również go zmienić lub napisać swój własny i uzyskać informacje, których być może nie chcesz mu udzielić. Poza tym mniej jest wymagań szkoleniowych, ponieważ użytkownik musi znać jedynie nazwę pliku i parametry, które mogą być przekazywane.

## Użycie szablonów XML

Do tej pory, gdy pisaliśmy pliki szablonów, w celu określenia instrukcji do wykonania używaliśmy tylko elementu `<sql:query>`. W szablonie pliku oprócz elementu `<sql:query>` mogą pojawić się jeszcze cztery inne elementy. Wydruk 4.11. przedstawia ogólny format pliku szablonu, a w tabeli 4.5 wyjaśniono znaczenie każdego z elementów.

**Wydruk 4.11.** *Format szablonu XML*

```
<ROOT xmlns:sql="urn:schemas-microsoft-com:xml-sql"
      sql:xsl="XSL FileName" >
  <sql:header>
    <sql:param>...</sql:param>
    <sql:param>...</sql:param>...n
  </sql:header>
  <sql:query>
    instrukcja(e) sql
  </sql:query>
    <sql:xpath-query mapping-schema="SchemaFileName.xml">
    Zapytanie XPath
  </sql:xpath-query>
</ROOT>
```

**Tabela 4.5.** *Elementy szablonu XML*

| Element | Opis |
|---|---|
| <ROOT> | Ten znacznik zawiera element najwyższego poziomu (jest określany również jako *znacznik główny*) wynikowego dokumentu XML. Może mieć dowolną nazwę. |
| <sql:header> | Ten znacznik jest używany do przechowywania wartości nagłówka. W bieżącej implementacji SQL Servera 2000 w znaczniku tym może być określony tylko element <sql:param>. Znacznik <sql:header> występuje w roli znacznika zawierającego, pozwalającego na definiowanie wielu parametrów. Zapewnia to większą wydajność, ponieważ wszystkie definicje parametrów znajdują się w jednym miejscu. Jest to podobne do deklarowania zmiennych na początku składowanych procedur T-SQL. |
| <sql:param> | Ten element definiuje parametry, które są przekazywane zapytaniom wewnątrz szablonu. Każdy element <sql:param> definiuje jeden parametr. W znaczniku <sql:header> można zdefiniować większą liczbę elementów <sql:param>. |
| <sql:query> | Te element określa zapytania SQL. W szablonie możesz mieć wiele elementów <sql:query>. |
| | Jeśli w szablonie jest wiele znaczników <sql:query> i jeden z nich zawiedzie, przetwarzanie innych będzie kontynuowane. |
| <sql:xpath-query> | Element określa zapytanie XPath. Nazwa pliku schematu musi zostać określona za pomocą atrybutu mapping-schema. |
| | Jeśli w szablonie jest wiele znaczników <sql:xpath-query> i jeden z nich zawiedzie, przetwarzanie innych będzie kontynuowane. |
| <sql:xsl> | Określa arkusz stylów, który ma zostać zastosowany do dokumentu wynikowego. Podana może być względna lub bezwzględna ścieżka do pliku. Jeśli podana została ścieżka względna, odwołuje się ona do katalogu szablonów (*Template*), zdefiniowanego za pomocą programu zarządzania katalogami wirtualnymi. |
| mapping-schema | Jeśli wywołujesz zapytanie XPath w szablonie, atrybut ten identyfikuje skojarzony z nim schemat XDR. Ścieżka dostępu do niego może być określana na podobnych zasadach, co w elemencie <sql:xsl>. |

A oto kilka przykładów użycia szablonów i plików szablonów w adresach URL. Użyję ponownie podanych wcześniej przykładów zapytań URL SQL, aby zilustrować różnice.

Mamy tutaj prostą instrukcję `SELECT` odnoszącą się do jednej tabeli, określoną bezpośrednio w URL-u:

```
http://IISServer/Northwind/?template=<ROOT+xmlns:sql="urn:schemas-microsoft-
➥com:xml-sql"><sql:query>SELECT+LastName,FirstName+FROM+Employees+FOR+XML+AUTO
</sql:query></ROOT>
```

Wynik prezentuje wydruk 4.12.

**Wydruk 4.12.** *Wynikowy dokument XML dla szablonu określonego w URL-u*

```
<ROOT xmlns:sql="urn:schemas-microsoft-com:xml-sql">
  <Employees LastName="Davolio" FirstName="Nancy" />
  <Employees LastName="Fuller" FirstName="Andrew" />
  <Employees LastName="Leverling" FirstName="Janet" />
  <Employees LastName="Peacock" FirstName="Margaret" />
  <Employees LastName="Buchanan" FirstName="Steven" />
  <Employees LastName="Suyama" FirstName="Michael" />
  <Employees LastName="King" FirstName="Robert" />
  <Employees LastName="Callahan" FirstName="Laura" />
  <Employees LastName="Dodsworth" FirstName="Anne" />
</ROOT>
```

Z tego szablonu stwórzmy plik szablonu — zapisany w bardziej czytelny sposób (zobacz wydruk 4.13).

**Wydruk 4.13.** *Zapytanie SQL zmienione na format szablonu*

```
<ROOT xmlns:sql="urn:schemas-microsoft-com:xml-sql">
  <sql:query>
    SELECT LastName, FirstName
    FROM Employees
    FOR XML AUTO
  </sql:query>
</ROOT>
```

Zakładając, że ten szablon zostanie zapisany jako plik *template1.xml* w katalogu szablonów nazw wirtualnych, wywołania szablonu dokonamy za pomocą następującego adresu URL:

```
http://iisserver/Northwind/templates/template1.xml
```

Spójrzmy na jeszcze jeden przykład. Kiedy tworzyliśmy zapytanie do trzech tabel, URL wyglądał następująco:

```
http://iisserver/Nwind?sql=SELECT+TOP+2+Orders.OrderID,+Employees.
LastName,+Orders.ShippedDate,+[Order+Details].UnitPrice,+[Order+
Details].ProductID+FROM+Orders,+Employees,+[Order+Details]+WHERE+
Orders.EmployeeID=Employees.EmployeeID+AND+Orders.OrderID=[Order+
Details].OrderID+Order+by+Employees.EmployeeID,Orders.OrderID+FOR+
XML+AUTO&root=ROOT
```

Plik szablonu będący zamiennikiem tego zapytania prezentuje wydruk 4.14.

**Wydruk 4.14.** *Długie zapytanie SQL zapisane ponownie w pliku szablonu*

```
<ROOT xmlns:sql="urn:schemas-microsoft-com:xml-sql">
  <sql:query>
    SELECT TOP 2
      Orders.OrderID,
      Employees.LastName,
      Orders.ShippedDate,
      [Order Details].UnitPrice,
      [Order Details].ProductID
    FROM
      Orders, Employees,[Order Details]
    WHERE
      Orders.EmployeeID=Employees.EmployeeID
    AND
      Orders.OrderID=[Order Details].OrderID
    ORDER BY
      Employees.EmployeeID,Orders.OrderID
    FOR XML AUTO
  </sql:query>
</ROOT>
```

Ten plik szablonu pokaże taki sam wynik, co kod pokazany na wydruku 4.6. Czy nie uważasz, że jest łatwiejszy do odczytu niż adres URL?

# Przekazywanie parametrów szablonu

Możemy przekazywać parametry zarówno do zapytań SQL, jak i do szablonów. Elementu `<sql:header>` używa się do definiowania parametrów; można im również przydzielić wartość domyślną. Wartości domyślne są używane w czasie wykonywania kodu, jeśli wartości parametrów nie zostały określone jawnie.

## Jawne wartości domyślne i przekazywanie parametrów

W następnym przykładzie chcemy przedstawić wartości pól `CustomerID`, `OrderID`, `RequiredDate` i `Freight` dla określonej w URL-u wartości `CustomerID`. Przyjrzyj się dokładnie plikowi szablonu z wydruku 4.15. Nasze zapytanie, jak można się było tego spodziewać, jest umieszczone w elemencie `<sql:query>`. Dodatkowo określiliśmy wprost domyślną wartość `VINET` dla `CustomerID`. Realizuje to element `<sql:param>`. Element `<sql:header>` przechowuje wszystkie parametry i ich wartości.

**Wydruk 4.15.** *Plik customer.xml*

```
<ROOT xmlns:sql="urn:schemas-microsoft-com:xml-sql">
  <sql:header>
    <sql:param name='CustomerID'>VINET</sql:param>
  </sql:header>
  <sql:query>
    SELECT CustomerID, OrderID, RequiredDate, Freight
    FROM Orders
```

```
    WHERE CustomerID=@CustomerID
    FOR XML AUTO
  </sql:query>
</ROOT>
```

Wartość elementu, który jest sparametryzowany w zapytaniu jest określona przez nazwę elementu poprzedzoną znakiem @. W tym przypadku jest to CustomerID. Nie pomyl użycia symbolu małpy w tym miejscu z jego użyciem przy atrybucie XML. Są to dwie całkowicie różne encje. To zastosowanie jest specyficzne dla parametryzowanych wyrażeń Microsoftu. Jeśli wywołamy ten plik szablonu (*customer.xml*) w adresie URL, otrzymamy wynik pokazany na wydruku 4.16.

```
http://iisserver/Northwind/templates/customer.xml
```

**Wydruk 4.16.** *Wynik wywołania pliku customer.xml bez przekazania wartości CustomerID*

```
<ROOT xmlns:sql="urn:schemas-microsoft-com:xml-sql">
  <Orders CustomerID="VINET" OrderID="10248" RequiredDate="1996-08-01T00:00:00"
  Freight="32.38" />
  <Orders CustomerID="VINET" OrderID="10274" RequiredDate="1996-09-03T00:00:00"
  Freight="6.01" />
  <Orders CustomerID="VINET" OrderID="10295" RequiredDate="1996-09-30T00:00:00"
  Freight="1.15" />
  <Orders CustomerID="VINET" OrderID="10737" RequiredDate="1997-12-09T00:00:00"
  Freight="7.79" />
  <Orders CustomerID="VINET" OrderID="10739" RequiredDate="1997-12-10T00:00:00"
  Freight="11.08" />
</ROOT>
```

Ponieważ wartość parametru CustomerID nie została przekazana w URL-u, plik szablonu użyje domyślnej wartości VINET. Jeśli przekażemy parametr o wartości WELLI, otrzymamy wynik przedstawiony na wydruku 4.17. Oto adres URL:

```
http://iisserver/Northwind/templates/customer.xml?CustomerID=WELLI
```

**Wydruk 4.17.** *Część wyniku po określeniu parametru CustomerID=WELLI*

```
<ROOT xmlns:sql="urn:schemas-microsoft-com:xml-sql">
  <Orders CustomerID="WELLI" OrderID="10256" RequiredDate="1996-08-12T00:00:00"
  Freight="13.97" />
  <Orders CustomerID="WELLI" OrderID="10420" RequiredDate="1997-02-18T00:00:00"
  Freight="44.12" />
  <Orders CustomerID="WELLI" OrderID="10585" RequiredDate="1997-07-29T00:00:00"
  Freight="13.41" />
...
</ROOT>
```

## Przekazywanie wielu parametrów

Zapewne myślisz, że przekazywanie wielu parametrów nie sprawia żadnych problemów — i masz rację. Parametry można po prostu przedstawić w postaci listy w elemencie <sql:header> oraz nadać im domyślne wartości. Spójrz na wydruk 4.18 oraz na wynik pokazany na wydruku 4.19.

**Wydruk 4.18.** *Shipvia.xml — wiele parametrów w szablonie*

```
<ROOT xmlns:sql="urn:schemas-microsoft-com:xml-sql">
  <sql:header>
    <sql:param name='ShipVia'>1</sql:param>
    <sql:param name='ShipCountry'>France</sql:param>
  </sql:header>
  <sql:query>
    SELECT TOP 4 CustomerID, OrderID, Freight
    FROM Orders
    WHERE ShipVia=@ShipVia
    AND ShipCountry=@ShipCountry
    ORDER BY OrderID
    FOR XML AUTO
  </sql:query>
</ROOT>
```

**Wydruk 4.19.** *Wynik kodu z wydruku 4.18*

```
<ROOT xmlns:sql="urn:schemas-microsoft-com:xml-sql">
  <Orders CustomerID="VICTE" OrderID="10251" Freight="41.34" />
  <Orders CustomerID="BLONP" OrderID="10265" Freight="55.28" />
  <Orders CustomerID="VINET" OrderID="10274" Freight="6.01"  />
  <Orders CustomerID="BONAP" OrderID="10331" Freight="10.19" />
</ROOT>
```

Wypisanie adresu URL *http://iisserver/Northwind/templates/shipvia.xml*, który wywołuje szablon z wydruku 4.18, daje wyniki pokazane na wydruku 4.19, ponieważ nie zostały przekazane żadne parametry, a użyte zostały domyślne wartości: `1` dla `ShipVia` i `France` dla `ShipCountry`. Mógłbyś również przekazać tylko jeden z tych parametrów. Wówczas wartość domyślna zostałaby zastąpiona wartością określoną przez Ciebie, a drugi parametr użyłby wartości domyślnej.

# Określanie arkusza stylów XSLT

Nie jest również niczym nowym zastosowanie arkusza stylów XSLT do wyników pliku szablonu. Po prostu określ nazwę arkusza stylów w atrybucie `sql:xsl` elementu `ROOT`. Przyjrzyjmy się raz jeszcze przykładowi podanemu w jednym z wcześniejszych punktów, „Kluczowe słowo XSL", który ilustrował użycie arkusza stylów XSLT.



Arkusze stylów są rozbudowywane za pomocą pewnych niezależnych (od SQL Servera) aplikacji wykorzystujących model dokumentu, który należy przekształcić. Podczas testów w trakcie rozbudowy przyczynami występowania błędów są puste dokumenty HTML, zmienione wyjścia XML lub próby przekształcenia dokumentu XML, który okazuje się pusty. Wszystkie te błędy przypisać można albo niewłaściwemu dokumentowi XML, albo niewłaściwym danym.

Wydruk 4.20 przedstawia plik szablonu *XSLDemo.xml*

**Wydruk 4.20.** *XSLDemo.xml*

```
<ROOT xmlns:sql='urn:schemas-microsoft-com:xml-sql'
sql:xsl='order.xsl'>
  <sql:query>
    SELECT TOP 4 OrderID, EmployeeID, Shipname
    FROM Orders
    WHERE EmployeeID=5
    FOR XML AUTO
  <sql:query>
</ROOT>
```

Zastosowany arkusz stylów pokazany jest na wydruku 4.9, wyniki — na wydruku 4.10, a tabela 4.4 to tabela wynikowa.

# Wywoływanie składowanych procedur

Do tej pory wywoływaliśmy albo zapytania SQL, albo szablony w adresie URL, albo pliki szablonów dostępne poprzez URL. Zdarzyło się nam jednak rażące przeoczenie: chodzi o *składowane procedury* (ang. *stored procedures*) i sposoby ich wykonywania. W gruncie rzeczy tworzyliśmy imitacje składowanych procedur poprzez użycie plików szablonów. Teraz użyjemy składowanych procedur tak, jak powinno się to robić (bez wątpienia musimy wiedzieć, jak z nich korzystać, zarówno w zapytaniach URL, jak i w plikach szablonów).



Umożliwianie użytkownikom pisania i wywoływania składowanych procedur w odniesieniu do bazy danych nie jest najbezpieczniejszym rozwiązaniem. Administratorzy powinni zezwalać użytkownikom na odczyt i wykonywanie składowanych procedur, ale nie na wpisywanie plików do `TemplateVirtualDirectory`. W innym wypadku będziesz się borykał z najróżniejszymi problemami.

Wydruk 4.21 pokazuje składowana procedurę, której będziemy używać w czasie omawiania tego zagadnienia.

**Wydruk 4.21.** *Przykładowa składowana procedura*

```
IF EXISTS (SELECT name FROM sysobjects
  WHERE name = 'OrderInfo' AND type = 'P')
  DROP PROCEDURE OrderInfo
GO
CREATE PROCEDURE OrderInfo
AS
  SELECT       OrderID, CustomerID
  FROM         Orders
  WHERE        CustomerID='CHOPS'
FOR XML AUTO
GO
```

Ta składowana procedura może być wywołana przez następujący URL:

```
http://IISServer/Northwind?sql=EXECUTE+OrderInfo&root=ROOT
```

Wydruk 4.22 pokazuje plik wynikowy.

**Wydruk 4.22.** *Wynik wywołania przykładowej składowanej procedury*

```
<?xml version="1.0" encoding="utf-8" ?>
<ROOT>
  <Orders OrderID="10254" CustomerID="CHOPS" />
  <Orders OrderID="10370" CustomerID="CHOPS" />
  <Orders OrderID="10519" CustomerID="CHOPS" />
  <Orders OrderID="10731" CustomerID="CHOPS" />
  <Orders OrderID="10746" CustomerID="CHOPS" />
  <Orders OrderID="10966" CustomerID="CHOPS" />
  <Orders OrderID="11029" CustomerID="CHOPS" />
  <Orders OrderID="11041" CustomerID="CHOPS" />
</ROOT>
```

Przekazywanie parametrów jest realizowane za pomocą symbolu `@` — przedstawia to wydruk 4.23.

**Wydruk 4.23.** *Przekazywanie parametru składowanej procedurze*[2]

```
...
  SELECT       OrderID, CustomerID
  FROM         Orders
  WHERE        CustomerID=@CustomerID
  FOR XML AUTO
...
```

Składowana procedura może być wywołana poprzez URL na dwa sposoby. Pierwsza metoda jest następująca:

```
http://IISServer/Northwind?sql=execute+OrderInfo+CHOPS&root=ROOT
```

Metoda ta udostępnia wartości `CHOPS` uwzględniając jej pozycję. Jeśli przekazywane byłyby dwa parametry, mógłbyś je umieścić jeden za drugim, a wtedy będą poprawnie przekazane.

Druga metoda jest następująca:

```
http://IISServer/Nwind?sql=execute+OrderInfo+@CustomerID=CHOPS&root=ROOT
```

Najczęściej używamy właśnie tej metody; dostarcza ona wartości `CHOPS` uwzględniając jej nazwę.

[2] Oczywiście należy pamiętać o zadeklarowaniu parametru w implementacji procedury zaraz po nazwie procedury w postaci { *@parameter data_type* } — *przyp. red.*

# Uzyskiwanie dostępu do obiektów baz danych poprzez HTTP

Pozostało wyjaśnić sposób dostępu do ostatniej encji. Omówiliśmy zapytania SQL i szablony bezpośrednio umieszczane w URL-ach, jak również użycie plików szablonów. Teraz zajmiemy się uzyskiwaniem dostępu do obiektów baz danych (tabel, widoków itd.).

# Wysyłanie szablonów przez formularze HTML

Ostatnim ćwiczeniem, które zamierzam zaprezentować w tym rozdziale, jest wysyłanie szablonów za pomocą formularzy HTML. W środowisku WWW jest to prawdopodobnie najbardziej powszechny sposób wywoływania plików szablonów. Powiedzmy sobie wyraźnie: użytkownicy swobodnie operują formularzami HTML i są z nimi dobrze zaznajomieni. Widują je niemal za każdym razem, kiedy uzyskują dostęp do Internetu lub firmowego intranetu, jeśli taki istnieje. Formularze są łatwe do stworzenia i zapewniają wysoki poziom interakcji.

Poniższy plik szablonu posiada jeden parametr, który jest przekazywany do zapytania. Można dodać więcej parametrów. Szablon ten zostanie zapisany jako plik HTML i umieszczony w katalogu nie należącym do drzewa wirtualnego katalogu macierzystego. Plik *SQLISAPI.DLL* nie spodziewa się pliku HTML i nie będzie funkcjonował poprawnie, jeśli plik HTML znajdzie się w tym drzewie katalogów.

Spójrzmy na sam formularz HTML, który jest przedstawiony na wydruku 4.24.

**Wydruk 4.24.** *Wysyłanie szablonu przez formularz HTML*

```
<head>
<TITLE>Przykładowy formularz</TITLE>
</head>
<body>
Wyszukane zostaną pola OrderID, OrderDate i Freight dla danej wartości CustomerID
<form action="http://griffinjnt4s/nwind" method="POST">
<B>CustomerID:</B>
<input type=text name=CustomerID value='VINET'>
<input type=hidden name=contenttype value=text/xml>
<input type=hidden name=template value='
<ROOT xmlns:sql="urn:schemas-microsoft-com:xml-sql">
<sql:header>
      <sql:param name="CustomerID">VINET</sql:param>
</sql:header>

<sql:query>
      SELECT OrderID, OrderDate, Freight
      FROM Orders
```

```
      WHERE CustomerID=@CustomerID
      FOR XML AUTO
</sql:query>
</ROOT>
'>

<P><input type="submit">
</form>
</body>
```

Pole tekstowe HTML dostarcza wartości, która jest przekazywana w zapytaniu. Zapytanie zwraca następnie pola `OrderID`, `OrderDate` i `Freight`.

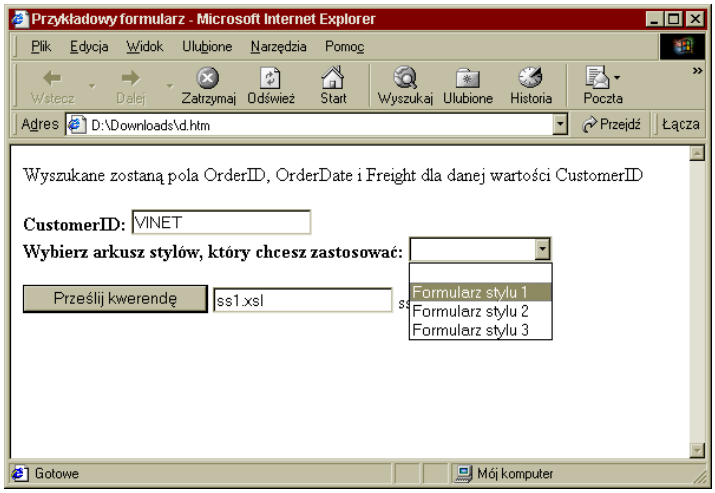
Użytkownik ma również możliwość wyboru arkusza stylów XSLT, który należy zastosować do danych. Uzupełnij kod z wydruku 4.24 wierszami prezentowanymi na wydruku 4.25.

**Wydruk 4.25.** *Dodatkowy kod HTML dla listy rozwijanej*

```
<br><B>
Wybierz arkusz stylów, który chcesz zastosować:</B>
<select name=stylesheets size='1'>
  <option value=''></option>
  <option value='ss1.xsl'>Arkusz stylów 1</option>
  <option value='ss2.xsl'>Arkusz stylów 2</option>
  <option value='ss3.xsl'>Arkusz stylów 3</option>
</select>
```

Kod doda do strony listę rozwijaną, pozwalającą użytkownikowi na wybranie arkusza stylów (zobacz rysunek 4.6).

**Rysunek 4.6.** *Dodanie listy rozwijanej umożliwiającej wybór arkusza stylów*




Uzupełnij również formularz dodatkowym polem wejściowym i polem parametru:

```
<input type="text" name="stylesheet" value="ss1.xsl">
<sql:param name="stylesheet">ss1</sql:param>
```

W końcu zmodyfikuj instrukcję <ROOT>:

```
<ROOT xmlns:sql="urn:schemas-microsoft-com:xml-sql"
sql:xsl="@stylesheet">
```

To ćwiczenie kończy niniejszy rozdział. Powinieneś zwrócić uwagę na fakt, że wygenerowane przez nas dokumenty są znane pod pojęciem *dokumentów opartych na atrybutach* (ang. *attribute-centric documents*). Wszystkie dane przydzielane są atrybutom elementów w dokumencie. Ich przeciwieństwem są *dokumenty oparte na elementach* (ang. *element-centric documents*). Wszystkie dane zwracane są jako serie zagnieżdżonych elementów, czego prawdopodobnie się spodziewałeś w pierwszej kolejności. SQL Server 2000 dostarcza sposobów generowania dokumentów opartych na elementach. Porozmawiamy o nich szerzej podczas omawiania klauzuli FOR XML w rozdziale 8.