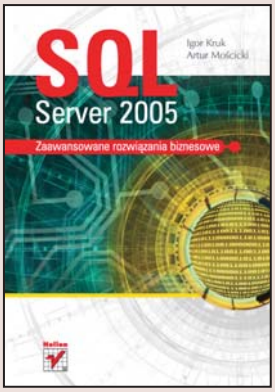
SQL Server 2005. Zaawansowane rozwiązania biznesowe

Autor: Igor Kruk, Artur Mościcki
ISBN: 978-83-246-1333-5
Format: 158x235, stron: 312



Zdobądź wiedzę o tworzeniu zaawansowanych aplikacji bazodanowych!

- Jak używać tabel tymczasowych do tworzenia specjalnych hierarchii?
- Do czego służą zmienne tablicowe?
- Na czym polega konwertowanie danych relacyjnych do formatu XML?

SQL Server 2005 to pierwsza wersja serwera, w której dane XML są przechowywane i przetwarzane faktycznie jako XML, a nie jako pochodne danych tekstowych lub binarnych, jak to było w wersjach poprzednich. W SQL Server 2005 możemy użyć typu danych XML jako kolumny, zmiennej lokalnej lub parametru. Możemy w niej przechowywać całe dokumenty XML lub tylko ich fragmenty (niezawierające elementu głównego, tzw. root node). Integracja z platformą Microsoft NET oraz ulepszone funkcje Business Intelligence pozwalają programistom na skupienie się na najważniejszych zadaniach, bez konieczności pracy w nieznanym środowisku, a przedsiębiorstwom dają możliwość przekształcania informacji w lepsze rozwiązania biznesowe.

Książka „SQL Server 2005. Zaawansowane rozwiązania biznesowe" przedstawia jeden z najpopularniejszych serwerów bazodanowych służących do budowy różnych systemów informatycznych, czyli SQL Server 2005. Ten obszerny podręcznik zawiera szczegółowe informacje oraz przykłady dotyczące wielowymiarowych baz danych oraz wymagań, jakie mogą pojawić się podczas budowy mechanizmów ich zasilania danymi źródłowymi. Czytając go, dowiesz się, jak tworzyć efektywne i wydajne aplikacje oraz nauczysz się wdrażać nowatorskie pomysły, które każdemu przedsiębiorstwu przyniosą wymierne korzyści biznesowe.

- Perspektywy
- Procedury i funkcje
- Wyzwalacze
- Dynamiczny SQL
- Tabele tymczasowe i zmienne tablicowe
- Transakcje i wyjątki w aplikacjach biznesowych
- Full Text Search
- XML
- Database Mail
- Rozwiązania biznesowe
- Integracja z .NET i CLR
- SQL Server Integration Services

Dowiedz się, jak tworzyć efektywne aplikacje bazodanowe i wdrażaj korzystne rozwiązania programistyczne dla biznesu!

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

Spis treści

Rozdział 8.

XML

Wstęp

SQL Server 2005 to pierwsza wersja serwera, w której dane *XML* są przechowywane i przetwarzane faktycznie jako *XML*, a nie jako pochodne danych tekstowych lub binarnych, jak to było w poprzednich wersjach serwera. Załadowanie danych *XML* np. w SQL Server 2000 było stosunkowo proste, jednak już dostęp do tych danych, modyfikowanie i wyszukiwanie konkretnych obiektów wymagały złożonych operacji. SQL Server 2000 umożliwiał wykonanie tylko dwóch poleceń związanych z obsługą formatu *XML*:

- `OPENXML` — umożliwia załadowanie dokumentu *XML* do pamięci SQL Servera, a następnie utworzenie z niego zbioru rekordów relacyjnych.
- `FOR XML` — umożliwia zapis danych relacyjnych, będących wynikiem zapytania *SQL* do postaci *XML*.

Wkrótce po dacie premiery SQL Server 2000 Microsoft zrozumiał, że jego najnowsza platforma bazodanowa nie wspiera obsługi danych i formatu *XML* na tyle, by sprostać oczekiwaniom i wymaganiom biznesowych użytkowników i twórców aplikacji w tym zakresie. Widząc, jak ważnym formatem w świecie biznesowych aplikacji bazodanowych stał się *XML*, Microsoft chciał za wszelką cenę zwiększyć jego integrację z SQL Server 2000. Jednak po oficjalnej premierze serwera firma mogła zaproponować tylko darmowe dodatki — pakiety, które rozszerzały SQL Server w tym zakresie. Pierwszym takim pakietem był *SQLXML* (*XML for SQL Server*), który dostarczał m.in. narzędzia do bardzo szybkiego wczytywania danych *XML*. Kolejnym pakietem był *MSXML* (*Microsoft XML Core Services*), który zawierał m.in. parser *XML*. Już wtedy jasne było, że w kolejnej wersji SQL Server musi nastąpić rewolucja w podejściu do formatu *XML*.

Integrację *XML* z SQL Server 2005 należy rozpatrywać w następujących obszarach:

- nowy typ danych *XML*,
- ograniczenia w kolumnach typu *XML*,

- *XML Schema Collection*,
- metody dostępu i obsługi danych *XML*.

Powyższym zagadnieniom poświęcony został ten rozdział książki. Zanim jednak zajmiemy się szczegółami tych zagadnień, przypomnijmy sobie podstawowe informacje na temat danych, plików i formatu *XML*.

Informacje o formacie XML

Skrót *XML* pochodzi od *Extensible Markup Language* (z ang.: rozszerzalny język znaczników). Język ten służy do łatwego przechowywania i wymiany danych pomiędzy różnymi aplikacjami, systemami i platformami. Format *XML* jest obecnie wykorzystywany w wielu obszarach informatycznych. Swoją popularność zawdzięcza temu, że jest stosunkowo prosty do zrozumienia oraz że zapisywany jest w postaci plików tekstowych. Czyni go to bardzo łatwo edytowalnym. Niewątpliwą zaletą formatu *XML* jest także fakt, że umożliwia on separację warstwy danych od warstwy prezentacji. Łatwo zrozumieć to przy porównaniu go z językiem *HTML*, w którym poszczególne tagi określają, w jaki sposób zawarte w pliku dane są prezentowane. *XML* także składa się z tagów (elementów), jednak w tym przypadku opisują one przechowywane dane, nie mówiąc nic na temat ich prezentacji w aplikacjach czy przeglądarce. O ile w *HTML* mamy do dyspozycji zdefiniowany zbiór tagów, np. `<b>Igor Kruk</b>`, w *XML* definiujemy własne tagi na potrzeby danych, które chcemy w tym pliku zapisać, np. `<FirstName>` ↪`Igor Kruk</FirstName>`. Tak jak w *HTML* poszczególne tagi mają swoje opcje, np. `<font color="black">`, tak też w *XML* do elementów przypisywane są atrybuty, które lepiej opisują te elementy. W poniższym przykładzie przechowującym dane o klientach użyte zostały atrybuty `Gender` (z ang.: płeć) i `Name` (z ang.: nazwisko):

```
<Clients>
       <Client Genre="M" Name="FirstClient" />
       <Client Genre="F" Name="SecondClient" />
       ...
</Clients>
```

Przechowywanie danych XML w SQL Server 2005

W SQL Server 2005 możemy użyć typu danych *XML* jako kolumny, zmiennej lokalnej lub parametru. W kolumnie tego typu możemy przechowywać całe dokumenty *XML* lub tylko ich fragmenty (niezawierające elementu głównego, tzw. *root node*). Typu danych *XML* używamy w taki sam sposób jak innych typów. Poniższy przykład tworzy tabelę `myMixes`, w której będziemy przechowywać dane na temat muzycznych megamiksów. W kolumnie `Title` zapisany będzie tytuł miksu, kolumna `Tracklist` będzie zaś przechowywała listę utworów w formacie *XML*:

```
CREATE TABLE myMixes
(MixID int IDENTITY(1,1) PRIMARY KEY,
Title varchar(255) NOT NULL,
Tracklist XML NULL)
```

Wstawimy teraz dane do utworzonej tabeli. W pierwszym przykładzie wykorzystana została instrukcja `INSERT`, w której dane są wstawiane jako zwykły tekst i zamieniane przy użyciu funkcji `CAST` na typ *XML*.

```
INSERT INTO myMixes (Title, Tracklist)
VALUES ('Party_Beats_Vol.2_mixed_by_Raven',
CAST('
  <Mix>
   <Tracks>
    <Track id="1">Raven - Intro</Chapter>
    <Track id="2">Track02 - Title02</Chapter>
    <Track id="3">Track03 - Title03</Chapter>
    <Track id="4">Track04 - Title 04</Chapter>
    </Tracks>
  </Mix>' as XML
  )
)
```

W drugim przykładzie deklarujemy zmienną `mixInfo` typu *XML*, przypisujemy jej wartość, a następnie wstawiamy przy użyciu polecenia `INSERT` do tabeli `myMixes`:

```
DECLARE @mixInfo XML
SET @mixInfo =
CAST('
  <Mix>
   <Tracks>
    <Track id="1">Raven - Intro</Track>
    <Track id="2">Track02 - Title02</Track>
    <Track id="3">Track03 - Title03</Track>
    <Track id="4">Track04 - Title 04</Track>
   </Tracks>
  </Mix>' as XML
  )

INSERT INTO myMixes (Title, Tracklist)
VALUES ('Orange_Dance_mixed_by_Raven', @mixInfo)
```

W obydwu przykładach dane zostały jawnie skonwertowane na typ *XML*. Podczas tej konwersji SQL Server wykonał tylko podstawowe sprawdzenie, czy dane mają format XML, np. czy wszystkie tagi otwierające mają odpowiednie tagi zamykające. Nie sprawdza natomiast, czy mają one określoną — oczekiwaną przez nas —strukturę. W powyższych przykładach struktura danych *XML* zakładała istnienie elementów `Mix`, `Tracks` i `Track` z atrybutem `id`. Gdybyśmy podjęli próbę wstawienia do tabeli `myMixes` danych *XML* w innej strukturze, SQL Server nie zgłosiłby błędu, bo nie wie tak naprawdę, jaka powinna być struktura wstawianych danych. Do wprowadzania ograniczeń na dane *XML* służą *schematy XML*.

Sprawdzanie poprawności danych XML przy użyciu schematów

Schematy XML (*XML Schema Difinition*) przechowywane są w plikach *XSD*. Zawierają one informacje, jak powinien wyglądać poprawny plik *XML* związany z danym schematem. *Schematy XML* mogą być wykorzystywane przez SQL Server 2005 do kontrolowania poprawności danych *XML* podczas ich dodawania i modyfikowania. *Schematy XML* są przechowywane w SQL Server 2005 jako obiekty. Listę wszystkich dostępnych schematów XML otrzymamy, wykonując poniższe zapytanie:

```
SELECT * FROM sys.XML_schema_collections
```

Tworzenie obiektów danych typu *XML* wymaga zdefiniowania w bazie danych zbiorów schematów. Służy do tego instrukcja `CREATE XML SCHEMA COLLECTION`. Instrukcja ta tworzy zbiór schematów mogący składać się z jednego lub więcej schematów, z których każdy opisuje jedną przestrzeń nazw. Składnia tego polecenia jest następująca:

```
CREATE XML SCHEMA COLLECTION [ <relational_schema>. ]sql_identifier
AS Expression
```

Polecenie przyjmuje dwa parametry. Pierwszy to unikalna nazwa schematu, drugi to jego definicja.

Listę dostępnych w bazie danych *schematów XML* znajdziemy w lokalizacji: *Nazwa bazy danych/Programmability/Types/XML Schema Collections*.

Zobaczmy jeden przykładowy schemat `Production.ManuInstructionsSchemaCollection` z bazy `AdventureWorks`:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:t="http://schemas.
↪microsoft.com/sqlserver/2004/07/adventure-works/ProductModelManuInstructions"
  targetNamespace="http://schemas.microsoft.com/sqlserver/2004/07/adventure-works/
  ↪ProductModelManuInstructions"
  elementFormDefault="qualified">
 <xsd:element name="root">
  <xsd:complexType mixed="true">
   <xsd:complexContent mixed="true">
    <xsd:restriction base="xsd:anyType">
     <xsd:sequence>
      <xsd:element name="Location" maxOccurs="unbounded">
       <xsd:complexType mixed="true">
        <xsd:complexContent mixed="true">
         <xsd:restriction base="xsd:anyType">
          <xsd:sequence>
           <xsd:element name="step" type="t:StepType" maxOccurs="unbounded" />
          </xsd:sequence>
          <xsd:attribute name="LocationID" type="xsd:integer" use="required" />
          <xsd:attribute name="SetupHours" type="xsd:decimal" />
          <xsd:attribute name="MachineHours" type="xsd:decimal" />
```

```
          <xsd:attribute name="LaborHours" type="xsd:decimal" />
          <xsd:attribute name="LotSize" type="xsd:decimal" />
         </xsd:restriction>
        </xsd:complexContent>
       </xsd:complexType>
      </xsd:element>
     </xsd:sequence>
    </xsd:restriction>
   </xsd:complexContent>
  </xsd:complexType>
 </xsd:element>
 <xsd:complexType name="StepType" mixed="true">
  <xsd:complexContent mixed="true">
   <xsd:restriction base="xsd:anyType">
   <xsd:choice minOccurs="0" maxOccurs="unbounded">
    <xsd:element name="tool" type="xsd:string" />
    <xsd:element name="material" type="xsd:string" />
    <xsd:element name="blueprint" type="xsd:string" />
    <xsd:element name="specs" type="xsd:string" />
    <xsd:element name="diag" type="xsd:string" />
   </xsd:choice>
  </xsd:restriction>
</xsd:complexContent>
</xsd:complexType>
</xsd:schema>
```

W powyższym schemacie znajdziemy definicję elementu `Location`:

```
<xsd:element name="Location" maxOccurs="unbounded">
```

Elementowi temu przypisany jest atrybut `LocationID`:

```
<xsd:attribute name="LocationID" type="xsd:integer" use="required"/>
```

Parametr `use="required"` oznacza, że atrybut ten musi wystąpić w pliku *XML* powiązanym z tym schematem. Parametr `type="xsd:integer"` oznacza, że atrybut `LocationID` musi być typu całkowitoliczbowego.

Spróbujmy teraz utworzyć nową tabelę zawierającą kolumnę typu *XML*, do której przypiszemy omawiany *schemat*. Następnie przetestujemy działanie sprawdzania poprawności wstawianych danych *XML* przez ten *schemat*.

Poniższe polecenie *T-SQL* tworzy tabelę o nazwie `TEST` składającą się z dwóch kolumn: identyfikatora wiersza i danych w formacie *XML*. Do tabeli przypisywany jest schemat `Production.ManuInstructionsSchemaCollection`.

```
CREATE TABLE dbo.TEST
(
  rowID int IDENTITY(1,1) PRIMARY KEY,
  dane XML (Production.ManuInstructionsSchemaCollection) NULL
)
```

Pierwszy przykład wstawia do tabeli `TEST` dane *XML* zgodne z całym schematem `Production.ManuInstructionsSchemaCollection`:

```
INSERT INTO TEST (dane) VALUES (
CAST(
'<t:root xmlns:t="http://schemas.microsoft.com/sqlserver/2004/07/adventure-works/
➥ProductModelManuInstructions">
<t:Location LotSize="0.0" SetupHours="0.0" LocationID="1" MachineHours="0.0"
➥LaborHours="0.0">
        <t:step>
                <t:tool>String</t:tool>
        </t:step>
  </t:Location>
</t:root>' AS XML))
```

Jeśli dokładnie przeanalizowalibyśmy treść wcześniej omawianego schematu, znaleźlibyśmy odwołania do wszystkich elementów z powyższego przykładu.

W kolejnym przykładzie atrybutowi `LocationID` przypiszemy wartość tekstową — niezgodną ze schematem:

```
INSERT INTO TEST (dane) VALUES (
CAST(
'<t:root xmlns:t="http://schemas.microsoft.com/sqlserver/2004/07/adventure-works/
➥ProductModelManuInstructions">
<t:Location LotSize="0.0" SetupHours="0.0" LocationID="tekst" MachineHours="0.0"
➥LaborHours="0.0">
        <t:step>
                <t:tool>String</t:tool>
         </t:step>
  </t:Location>
</t:root>' AS XML))
```

Próba wykonania tego polecenia zakończy się następującym komunikatem o błędzie, informującym, że wartość atrybutu `LocationID` jest niepoprawna:

```
XML Validation: Invalid simple type value: 'tekst'. Location: /*:root[1]/
➥*:Location[1]/@*:LocationID
```

W kolejnym przykładzie pominiemy definicję wszystkich atrybutów elementu `Location`.

```
INSERT INTO TEST (dane) VALUES (
CAST(
'<t:root xmlns:t="http://schemas.microsoft.com/sqlserver/2004/07/adventure-works/
➥ProductModelManuInstructions">
  <t:Location>
        <t:step>
                <t:tool>String</t:tool>
        </t:step>
  </t:Location>
</t:root>' AS XML))
```

Przy próbie wykonania tego polecenia dostaniemy następujący komunikat:

```
XML Validation: Required attribute 'LocationID' is missing. Location: /*:root[1]/
➥*:Location[1]
```

Zwróćmy uwagę, że SQL Server 2005 informuje, że brak tylko atrybutu `LocationID`. Nie wspomina natomiast nic o pozostałych atrybutach: `LotSize`, `SetupHours`, `MachineHours`, `LaborHours`. Wynika to z tego, że atrybuty te nie mają w definicji *schematu* `Production.ManuInstructionsSchemaCollection` parametru `use="required"`.

Jak zauważyliśmy w powyższych przykładach, dzięki przypisywaniu *schematów XML* do kolumn tego typu mamy możliwość automatycznej weryfikacji poprawności wstawianych i modyfikowanych danych. Przedstawione w tym rozdziale informacje dotyczące *schematów XML* obejmują tylko część tej tematyki. Zainteresowanych tematem *schematów XML* odsyłamy na stronę *http://www.w3.org/XML/Schema*.

Metody dostępu do danych XML i ich obsługi

W SQL Server 2005 dostępnych jest pięć metod umożliwiających operowanie na danych typu *XML*. Są to:

- `exist`,
- `nodes`,
- `query`,
- `value`,
- `modify`.

Metody `exist`, `nodes`, `query`, `value` zalicza się do języka *XQuery*, który służy do wyszukiwania informacji w danych typu *XML*. Przyjrzyjmy się teraz bliżej poszczególnym metodom.

Metoda exist

Metoda `exist` pozwala sprawdzić, czy w danych *XML* istnieje określony obiekt. Zwraca ona wartość `1` (`True`), gdy obiekt znajduje się w danych *XML*, w przeciwnym razie zwracana jest wartość `0` (`False`). Posłużmy się omawianym wcześniej schematem *XML* i sprawdźmy, czy w tabeli `test` w kolumnie `dane` znajduje się element `Location` z atrybutem `LocationID` o wartości `1`. Poniższe polecenie *T-SQL* wykonuje to sprawdzenie:

```
SELECT * FROM test
WHERE dane.exist('declare namespace t="http://schemas.microsoft.com/sqlserver/
➥2004/07/adventure-works/ProductModelManuInstructions";
/t:root/t:Location[@LocationID=1]') = 1
```

Zwróćmy uwagę na konstrukcję tego polecenia. Metoda `exist` jest wykonywana na kolumnie `dane`, którą wcześniej zdefiniowaliśmy jako typu *XML*. W nawiasie znajduje się deklaracja przestrzeni nazw `t`, a następnie odwołanie do danych w postaci */rodzic/potomek[atrybut]*.

Należy także zaznaczyć, że w powyższym przykładzie adres *URL* w deklaracji przestrzeni nazw musi znajdować się w jednym wierszu. W książce został złamany ze względu na ograniczenia w druku.

Metoda query

Wykorzystując metodę `query` oraz poprawnie zdefiniowane zapytanie *XQuery*, mamy łatwy dostęp do danych *XML*. Metoda pobiera fragment danych *XML* i zwraca je w postaci tekstowej, a nie *XML*. W poniższym przykładzie odwołujemy się do bazy `AdventureWorks` i tabeli `Production.ProductionModel`. Pobierane są elementy `steps` z danych *XML* dla wiersza z identyfikatorem `10`.

```
SELECT
        ProductModelID,
        Instructions.query('declare namespace t="http://schemas.microsoft.com/
        ➥sqlserver/2004/07/adventure-works/ProductModelManuInstructions";
         /t:root/t:Location/t:step') AS Steps
FROM Production.ProductModel
WHERE ProductModelID = 10
```

Deklarowanie przestrzeni nazw wewnątrz zapytania *T-SQL* wpływa negatywnie na jego czytelność. Lepszym rozwiązaniem jest wykorzystanie polecenia `WITH NAMESPACE ()`. Powyższe zapytanie będzie wyglądać teraz następująco:

```
WITH XMLNAMESPACES ('http://schemas.microsoft.com/sqlserver/2004/07/adventure-works/
➥ProductModelManuInstructions' AS t)

SELECT ProductModelID, Instructions.query('/t:root/t:Location/t:step') AS Steps
FROM Production.ProductModel
WHERE ProductModelID = 10
```

Metoda value

Metoda `value` służy do pobierania konkretnych wartości elementów lub ich atrybutów w postaci jednego z podstawowych typów danych, np. `int`, `varchar`. Metoda ta przyjmuje dwa argumenty. Pierwszy to poprawnie skonstruowane polecenie `XQuery`, drugi zaś to nazwa podstawowego typu danych, w którym mają być zwrócone wyniki. Dozwolone są wszystkie typy danych oprócz `XML`, `image`, `text`, `ntext`, `timestamp` oraz typów zdefiniowanych przez użytkownika.

Wykorzystajmy ponownie tabelę `Production.ProductModel` z bazy `AdventureWorks`. Załóżmy, że chcemy uzyskać wartość atrybutu `LotSize`, drugiego elementu `Location` dla wiersza z identyfikatorem `10`. Zapytanie może wyglądać tak jak poniżej (dla czytelności i przejrzystości zapytania ponownie zastosowaliśmy polecenie `WITH NAMESPACE`):

```
WITH XMLNAMESPACES ('http://schemas.microsoft.com/sqlserver/2004/07/adventure-works/
➥ProductModelManuInstructions' AS t)

SELECT ProductModelID,
Instructions.value('(/t:root/t:Location/@LotSize)[2]',
'decimal (5,2)') AS Location
FROM Production.ProductModel
WHERE ProductModelID = 10
```

W wyniku otrzymujemy jeden wiersz:

```
ProductModelId          Location
10                      1.00
```

Cały czas pamiętajmy, że adres *URL* w deklaracji przestrzeni nazw musi znajdować się w jednym wierszu.

Jeśli taka konwersja nie będzie możliwa, SQL Server 2005 zgłosi błąd. W poniższym przykładzie chcemy uzyskać wartość elementu `material` i nadać jej dziesiętny typ danych:

```
WITH XMLNAMESPACES ('http://schemas.microsoft.com/sqlserver/2004/07/adventure-works/
↪ProductModelManuInstructions' AS t)

SELECT ProductModelID,
Instructions.value('(/t:root/t:Location/t:step/t:material)[2]',
'decimal (5,2)') AS Location
FROM Production.ProductModel
WHERE ProductModelID = 10
```

Przy próbie wykonania tego polecenia dostaniemy komunikat o błędzie:

```
Error converting data type nvarchar to numeric.
```

Wynika to z faktu, że wartość elementu `material` ma postać tekstową. Zatem w poprawnym zapytaniu zwracanej wartości może być przypisany typ danych *varchar*. Przedstawia to poniższy przykład:

```
WITH XMLNAMESPACES ('http://schemas.microsoft.com/sqlserver/2004/07/adventure-works/
↪ProductModelManuInstructions' AS t)

SELECT ProductModelID,
Instructions.value('(/t:root/t:Location/t:step/t:material)[2]',
'varchar(255)') AS Material
FROM Production.ProductModel
WHERE ProductModelID = 10
```

W wyniku otrzymujemy jeden wiersz:

```
ProductModelId          Material
10                      Acme Polish Cream
```

Metoda nodes

Metoda `nodes` umożliwia przekształcenie danych typu *XML* w dane o strukturze relacyjnej. Dla każdego wystąpienia odpowiednio zdefiniowanego w zapytaniu *XQuery* elementu tworzony jest oddzielny wiersz w wynikowej tabeli relacyjnej. Wróćmy do przykładu z tego rozdziału dotyczącego danych *XML* o megamiksach. W poniższym przykładzie w pierwszej kolejności definiowane są dane w postaci *XML*. Następnie w zapytaniu *T-SQL* metoda nodes konwertuje każde wystąpienie elementu `Track` na postać relacyjną — jednego wiersza w tabeli `Tab` w kolumnie `Col`. W klauzuli `SELECT` posłużymy się poznaną wcześniej metodą `value` do uzyskania wartości każdego elementu:

```
DECLARE @mixInfo XML
SET @mixInfo =
CAST('
  <Mix>
   <Tracks>
    <Track id="1">Raven - Intro</Track>
    <Track id="2">Track02 - Title02</Track>
    <Track id="3">Track03 - Title03</Track>
    <Track id="4">Track04 - Title 04</Track>
   </Tracks>
  </Mix>' as XML
  )

SELECT Tab.Col.value('.', 'varchar(255)') AS tracklist
FROM @mixInfo.nodes('/Mix/Tracks/Track') as Tab(Col)
```

Wynikiem zapytania są cztery wiersze:

```
tracklist
Raven - Intro
Track02 - Title02
Track03 - Title03
Track04 - Title 04
```

Zwróćmy uwagę, że gdybyśmy nie użyli w klauzuli `SELECT` jednej z XML-owych metod (`exist`, `query`, `value`) i spróbowali wykonać następujące zapytanie:

```
SELECT * FROM @mixInfo.nodes('/Mix/Tracks/Track') as Tab(Col)
```

to SQL Server zwróciłby komunikat o błędzie:

```
The column 'Col' that was returned from the nodes() method cannot be used directly.
➥It can only be used with one of the four XML datatype methods exist(), nodes(),
➥query(), and value() or in IS NULL and IS NOT NULL checks.
```

Metoda modify

Metoda `modify` jest rozszerzeniem opracowanym przez firmę Microsoft do języka *XQuery* i nazwanym *XML Data Manipulation Language — XML DML*. Standardowe metody języka *XQuery*, omówione wcześniej w tym rozdziale, służą tylko do pobierania informacji z *XML*. Metoda `modify`, jako jedyna, umożliwia modyfikowanie danych *XML*. *XML DML* udostępnia nowe polecenia:

- `insert`,
- `delete`,
- `replace value of`.

Do modyfikowania danych *XML* będziemy wykorzystywać standardowe polecenie *UPDATE*, w którym wykorzystamy metodę *modify* z jednym z powyższych poleceń.