



# Wykorzystanie XML w środowisku SQL Serwer

mgr inż. Paweł Kozłowski  
p\_kozlowski@poczta.fm

# XML – wstęp

# XML

- ▶ Służy do przechowywania danych konfiguracyjnych
  - ▶ Zastosowanie w procesach serializacji danych
  - ▶ Wykorzystywany do przesyłania danych przez usługi sieciowe (Web Services)
- 
- ▶ Idealny do przechowywania danych obiektowych i relacyjnych
  - ▶ Nadaje się do przechowywania różnego typu struktur danych

# XML

Typowy dokument xml składa się z węzłów:

- ▶ Elementy

`<element></element>`

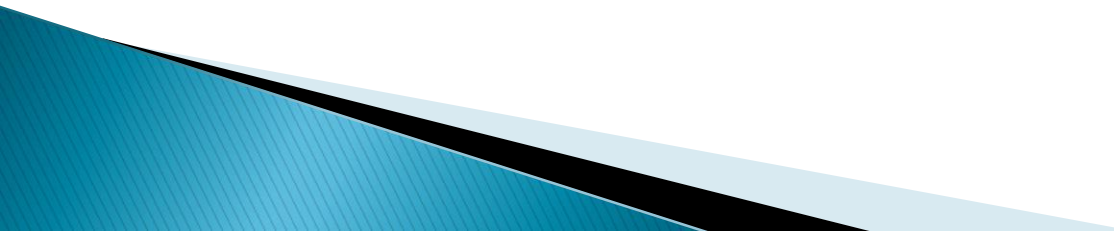
- ▶ Atrybuty

`<element atrybut="wartość atrybutu"/>`

- ▶ Tekst

`<element>Tekst</element>`

Wszystkie reguły xml opisane przez W3C (World Wide Web Consortium)



# XML – podstawowe zasady

- ▶ Dokument xml musi mieć jeden znacznik główny (root)
- ▶ Każdy element musi być zawsze zamknięty

`<element></element>`  
`<element/>`

- ▶ W elemencie atrybut może wystąpić tylko raz

~~`<element atrybut="wartość1" atrybut="wartość2"/>`~~  
`<element atrybut="wartość1" ATRYBUT="wartość2"/>`

- ▶ Elementy nie mogą na siebie nachodzić

~~`<element1><element2></element1></element2>`~~  
`<element1><element2></element2></element1>`

# Typ danych XML w MS SQL Server

Przykłady realizowane na bazie Cukierki



# Typ danych XML

- ▶ Pojawił się od SQL 2005
- ▶ Służy do przechowywania dokumentów xml lub ich fragmentów
- ▶ Max rozmiar danych w kolumnie lub zmiennej – 2GB
- ▶ Danych typu XML nie można
  - porównać,
  - sortować
  - grupować klauzulą GROUP BY
- ▶ Kolumna xml nie może być częścią klucza indeksu, choć może być dołączona do indeksu klauzulą INCLUDE

# Typ danych XML – c.d.

- ▶ W kolumnach typu XML mogą być wykorzystywane przestrzenie nazw (namespace)
- ▶ Deklaracja xml (1-sza linijka dokumentu) jest automatycznie usuwana z kolumny xml

`<?xml version=" 1.0" encoding=„UTF-8” ?>`

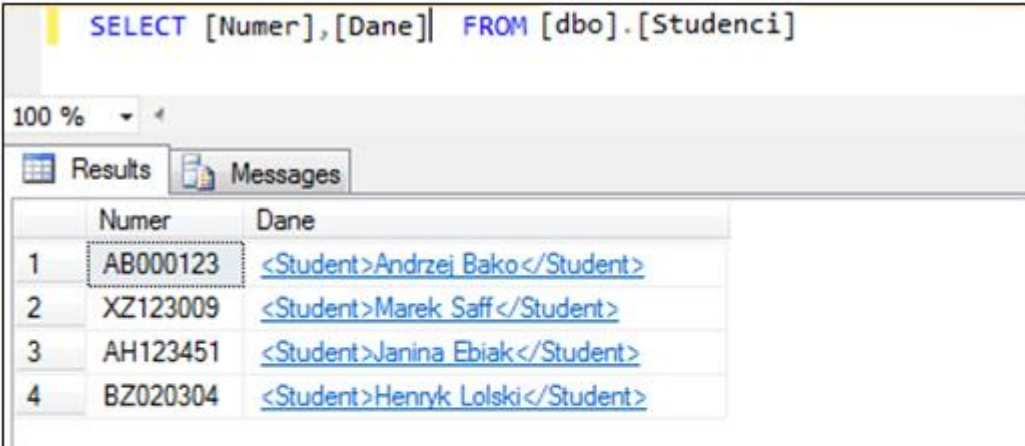
- ▶ W przypadku, gdy kolumna lub zmienna xml nie ma podpętego schematu, nie obowiązuje zasada jednego elementu głównego (root) – **możliwość przechowywania fragmentów xml**

# Przykład – tworzenie kolumny XML

- ▶ Przechowywanie danych xml w formie natywnej – bezpośrednio w formacie xml

```
CREATE TABLE Studenci  
(  
  Numer nchar(10) not null,  
  Dane xml not null,  
);  
GO
```

```
INSERT INTO Studenci  
VALUES('AB000123', N'<Student>Andrzej Bako</Student>'),  
( 'XZ123009', N'<Student>Marek Saff</Student>'),  
( 'AH123451', N'<Student>Janina Ebiak</Student>'),  
( 'BZ020304', N'<Student>Henryk Loliski</Student>');  
GO
```



	Numer	Dane
1	AB000123	<a href="#">&lt;Student&gt;Andrzej Bako&lt;/Student&gt;</a>
2	XZ123009	<a href="#">&lt;Student&gt;Marek Saff&lt;/Student&gt;</a>
3	AH123451	<a href="#">&lt;Student&gt;Janina Ebiak&lt;/Student&gt;</a>
4	BZ020304	<a href="#">&lt;Student&gt;Henryk Loliski&lt;/Student&gt;</a>

# Kolekcje schematów XSD

## XSD – eXtensible Schema Definition

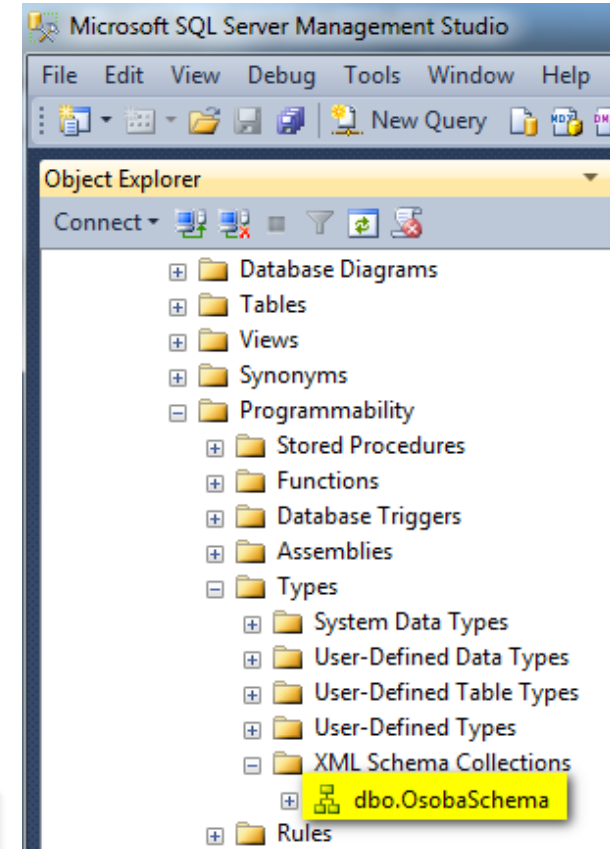
- ▶ Wykorzystywane do walidacji danych xml przechowywanych w tabelach
- ▶ Schemat rejestrujemy w bazie SQL jako kolekcję schematów XSD (XML Schema Collection)

# Tworzenie kolekcji schemów XSD

```
CREATE XML SCHEMA COLLECTION OsobaSchema AS
```

```
'<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" >
  <xs:element name="osoba">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="nazwisko" type="xs:string"/>
        <xs:element name="imie" type="xs:string"
          maxOccurs="2"/>
        <xs:element name="telefon" type="xs:string"/>
        <xs:element name="data_urodzenia"
          type="xs:date"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
'
```

```
CREATE TABLE Osoby
( ID integer PRIMARY KEY,
  Osoba xml(OsobaSchema))
```



# Tworzenie kolekcji schemów XSD

Schemat można zaimportować z pliku xsd

Use BazaSQL

```
DECLARE @xsd as XML
SET @xsd = (SELECT * FROM OPENROWSET(BULK 'C:\schemat.xsd',
SINGLE_BLOB) as X);
CREATE XML SCHEMA COLLECTION XMLSchemaProducts AS @xsd;
```

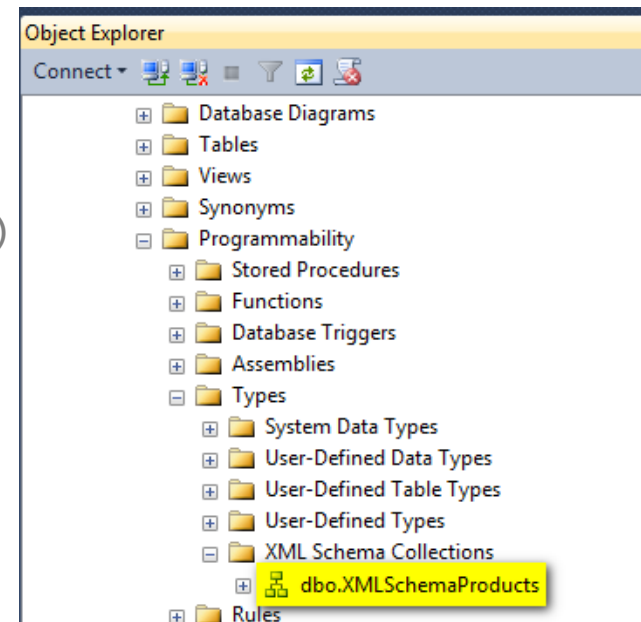
Deklaracja zmiennej xml  
bazującej na schemacie xsd

```
DECLARE @zmienna XML ([dbo].[XMLSchemaProducts])
```

Ćwiczenie



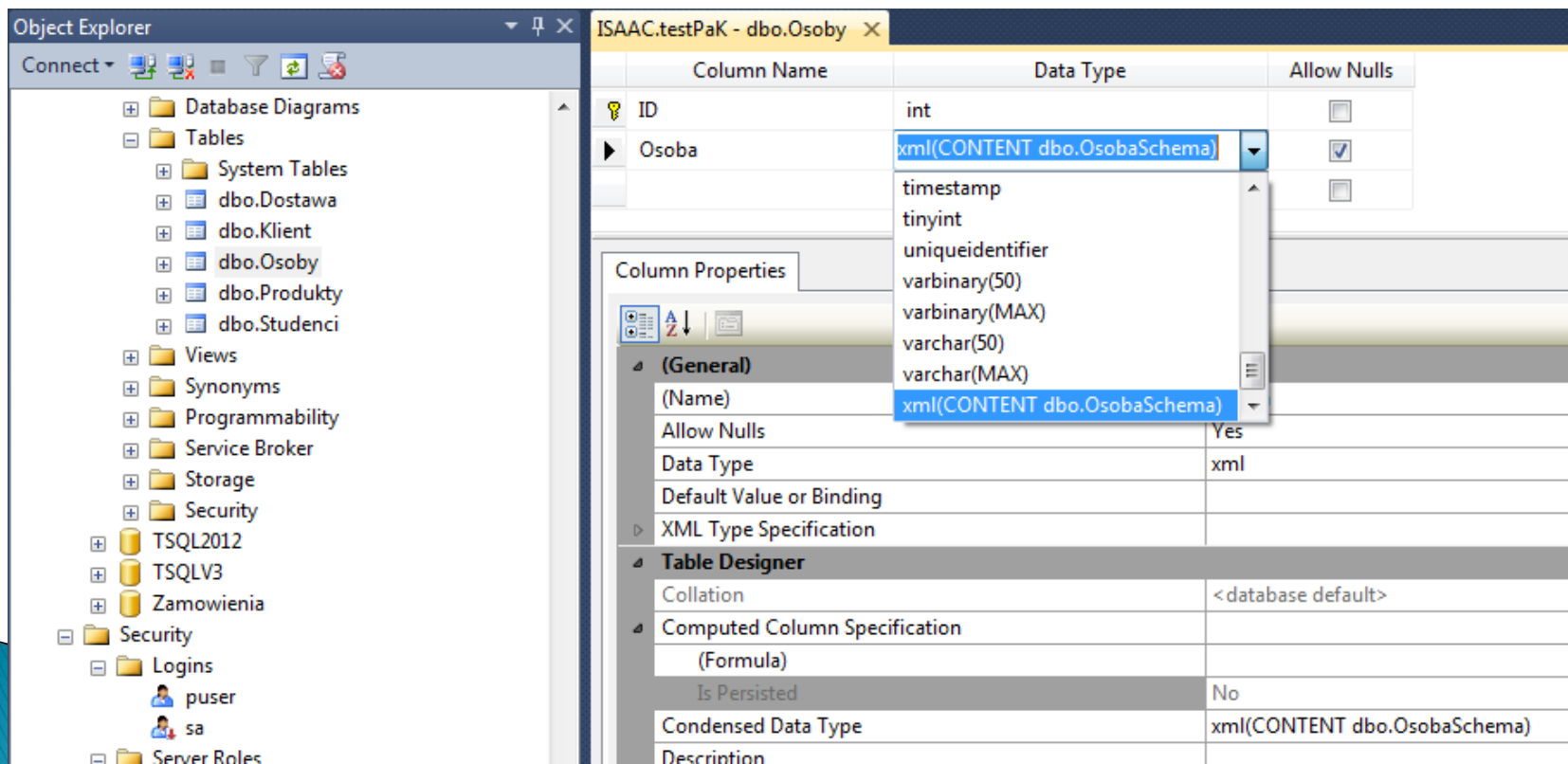
Wykład 2 - wykorzystanie XML na SQL.zip



# Schematy XML na kolumnach XML w tabelach

- ▶ Zarejestrowane XML SCHEMA COLLECTION można podpiąć pod kolumnę typu XML – [laboratorium 1, ćw. 10](#)

```
CREATE TABLE Osoby  
( ID integer PRIMARY KEY,  
  Osoba xml(OsobaSchema))
```



The screenshot shows the SQL Server Enterprise Manager interface. On the left, the 'Object Explorer' pane displays the database structure, including 'Database Diagrams', 'Tables', 'Views', 'Synonyms', 'Programmability', 'Service Broker', 'Storage', 'Security', 'TSQL2012', 'TSQLV3', 'Zamowienia', 'Security', 'Logins', and 'Server Roles'. The 'dbo' schema is expanded, showing tables like 'Dostawa', 'Klient', 'Osoby', 'Produkty', and 'Studenci'. The 'Osoby' table is selected, and its properties are displayed in the 'Table Designer' pane. The 'Osoba' column is highlighted, and its properties are shown in the 'Column Properties' pane. The 'Data Type' is set to 'xml', and the 'XML Schema Collection' is set to 'dbo.OsobaSchema'.

Column Name	Data Type	Allow Nulls
ID	int	<input type="checkbox"/>
Osoba	xml(CONTENT dbo.OsobaSchema)	<input checked="" type="checkbox"/>

Column Properties

(General)

(Name)

Allow Nulls

Data Type

Default Value or Binding

XML Type Specification

Table Designer

Collation

Computed Column Specification

(Formula)

Is Persisted

Condensed Data Type

Description

# Schematy XML na kolumnach XML w tabelach

- ▶ Import danych do kolumny XML kontrolowanej przez schemat XML

```
DECLARE @docxml
```

```
SET @doc = '<?xml version="1.0" encoding="utf-8"?>
```

```
<osoba>
```

```
  <nazwisko>Karwowski</nazwisko>
```

```
  <imie>Waldemar</imie>
```

```
  <imie>Jan</imie>
```

```
  <telefon>225937273</telefon>
```

```
  <data_urodzenia>1970-01-14</data_urodzenia>
```

```
</osoba>
```

```
,
```

```
INSERT INTO Osoby (ID, Osoba)
```

```
VALUES(1, @doc)
```

Import danych zgodnych ze  
schematem XML

# Schematy XML na kolumnach XML w tabelach

- ▶ Import danych do kolumny XML kontrolowanej przez schemat XML

INSERT INTO Osoby

```
VALUES(1, N'<Student>Andrzej Bako</Student>'),  
(2, N'<Student>Marek Saff</Student>'),  
(3, N'<Student>Janina Ebiak</Student>'),  
(4, N'<Student>Henryk Lolcki</Student>');
```

GO

Import danych niezgodnych  
ze schematem XML

```
INSERT INTO Osoby  
VALUES(1, N'<Student>Andrzej Bako</Student>'),  
(2, N'<Student>Marek Saff</Student>'),  
(3, N'<Student>Janina Ebiak</Student>'),  
(4, N'<Student>Henryk Lolcki</Student>');  
GO  
|
```

100 %

Messages

Msg 6913, Level 16, State 1, Line 1  
XML Validation: Declaration not found for element 'Student'. Location: /\*:Student[1]

# Schematy XML na kolumnach XML w tabelach

- ▶ Zarejestrowane XML SCHEMA COLLECTION można podejrzeć min w SQL Server Management Studio

The screenshot displays the SQL Server Enterprise Manager interface. On the left, the 'Object Explorer' pane shows the database structure, including 'Database Diagrams', 'Tables', and 'System Tables'. The 'Tables' folder is expanded, showing a list of tables: 'dbo.Dostawa', 'dbo.Klient', 'dbo.Osoby', 'dbo.Produkty', and 'dbo.Studenci'. The 'dbo.Osoby' table is selected. The main pane shows the 'Column Properties' dialog for the 'Osoba' column. The 'Data Type' is set to 'xml'. The 'XML Type Specification' section is expanded, showing the '(Schema Collection)' set to 'dbo.OsobaSchema' and 'Is XML Document' set to 'sys.sys'. The 'Table Designer' section is also expanded, showing the 'Computed Column Specification' set to 'xml(CONTENT dbo.OsobaSchema)'. A red arrow points to the 'dbo.OsobaSchema' dropdown menu, and a red box highlights the 'sys.sys' value in the 'Is XML Document' field.

Column Name	Data Type	Allow Nulls
ID	int	<input type="checkbox"/>
Osoba	xml(CONTENT dbo.OsobaSchema)	<input checked="" type="checkbox"/>

Column Properties	
<b>(General)</b>	
(Name)	Osoba
Allow Nulls	Yes
Data Type	xml
Default Value or Binding	
<b>XML Type Specification</b>	
(Schema Collection)	dbo.OsobaSchema
Is XML Document	sys.sys
<b>Table Designer</b>	
Collation	<database default>
<b>Computed Column Specification</b>	
(Formula)	
Is Persisted	No
Condensed Data Type	xml(CONTENT dbo.OsobaSchema)
Description	

# Schematy XML na kolumnach XML w tabelach

Wszystkie zarejestrowane XML SCHEMA COLLECTION można sprawdzić komendą:

```
select * from sys.xml_schema_collections
```

Schemat można wyrejestrować poniższą komendą ale tylko w przypadku, gdy nie ma żadna kolumna nie jest od niego zależna:

```
Drop XML SCHEMA COLLECTION OsobaSchema
```

The screenshot displays the SQL Server Enterprise Manager interface. On the left, the Object Explorer shows the database structure, including tables (dbo.Dostawa, dbo.Klient, dbo.Osoby, dbo.Produkty, dbo.Studenci), Views, Synonyms, Programmability (Stored Procedures, Functions, Database Triggers, Assemblies), Types (System Data Types, User-Defined Data Types, User-Defined Table Types, User-Defined Types), XML Schema Collections (highlighted), and Rules. The 'dbo.OsobaSchema' XML Schema Collection is selected under the XML Schema Collections folder. On the right, the SQL Query window shows the query 'select \* from sys.xml\_schema\_collections' and its results. The results table has columns: xml\_collection\_id, schema\_id, principal\_id, name, create\_date, and modify\_date. Two rows are shown: one for 'sys' (xml\_collection\_id 1, schema\_id 4) and one for 'OsobaSchema' (xml\_collection\_id 65543, schema\_id 1). Red arrows point from the 'dbo.OsobaSchema' in the Object Explorer to the 'OsobaSchema' row in the query results.

xml_collection_id	schema_id	principal_id	name	create_date	modify_date
1	4	NULL	sys	2010-04-02 16:59:23.123	2010-04-02 16:59:23.683
65543	1	NULL	OsobaSchema	2017-10-10 01:29:41.153	2017-10-10 01:29:41.153

# Schematy XML na kolumnach XML w tabelach

Listę elementów zarejestrowanych w XML SCHEMA COLLECTION można sprawdzić poniższym poleceniem:

```
SELECT CP.*
FROM sys.xml_schema_components AS CP
JOIN sys.xml_schema_collections AS C
ON CP.xml_collection_id = C.xml_collection_id
WHERE C.name = 'OsobaSchema';

GO
```

```
SELECT CP.*
FROM sys.xml_schema_components AS CP
JOIN sys.xml_schema_collections AS C
ON CP.xml_collection_id = C.xml_collection_id
WHERE C.name = 'XMLSchemaOsoba';
GO
```

100 %

ResultsMessages

	xml_component_id	xml_collection_id	xml_namespace_id	is_qualified	name	symbol_space	symbol_space_desc	kind	kind_desc	derivation	derivation_desc	base_xml_component_id	scoping_xml_component_id
1	65631	65545	1	1	osoba	E	ELEMENT	E	ELEMENT	N	NONE	NULL	NULL
2	65632	65545	1	0	NULL	T	TYPE	K	COMPLEX_TYPE	R	RESTRICTION	6	65631
3	65633	65545	1	0	NULL	M	MODEL_GROUP	M	MODEL_GROUP	N	NONE	NULL	65632
4	65634	65545	1	0	nazwisko	E	ELEMENT	E	ELEMENT	N	NONE	NULL	65633
5	65635	65545	1	0	imie	E	ELEMENT	E	ELEMENT	N	NONE	NULL	65633
6	65636	65545	1	0	telefon	E	ELEMENT	E	ELEMENT	N	NONE	NULL	65633
7	65637	65545	1	0	data_urodzenia	E	ELEMENT	E	ELEMENT	N	NONE	NULL	65633

# Schematy XSD na kolumnach typu XML

Schemat można odtworzyć z zarejestrowanego XML SCHEMA COLLECTION przy użyciu poniższego polecenia:

```
SELECT xml_schema_namespace(N' ', N'XMLSchematOsoba')  
GO
```

The screenshot displays the SQL Server Enterprise Manager interface. The top pane shows the execution of the following T-SQL query:

```
SELECT xml_schema_namespace(N' ', N'XMLSchematOsoba')  
GO
```

The bottom pane is divided into two sections. The left section, titled "Results", shows a single row of data with the column name "(No column name)" and the value:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema..."
```

The right section, titled "xmlresult8.xml", displays the full XML schema definition for the "osoba" element:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">  
  <xsd:element name="osoba">  
    <xsd:complexType>  
      <xsd:complexContent>  
        <xsd:restriction base="xsd:anyType">  
          <xsd:sequence>  
            <xsd:element name="nazwisko" type="xsd:string" />  
            <xsd:element name="imie" type="xsd:string" maxOccurs="2" />  
            <xsd:element name="telefon" type="xsd:string" />  
            <xsd:element name="data_urodzenia" type="xsd:date" />  
          </xsd:sequence>  
        </xsd:restriction>  
      </xsd:complexContent>  
    </xsd:complexType>  
  </xsd:element>  
</xsd:schema>
```

# Język XPath i XQuery

# Język XPath i XQuery

XPath – XML Path Language

XQuery – XML Query Language

- ▶ Stworzone przez konsorcjum W3C
- ▶ Przeznaczone do operacji na danych XML
  - Wyszukiwanie określonych węzłów
  - Modyfikowanie struktury xml

# Język XPath

- ▶ Język XPath jest wykorzystywany do zapytań XQuery
- ▶ Język XPath umożliwia wskazanie odpowiedniego węzła dokumentu XML
  - **Prawy ukośnik (/)** – wskazuje na element główny dokumentu
  - **Dwa prawe ukośniki (//)** – wskazuje na dowolne miejsce dokumentu
  - **Znak @** – poprzedza nazwy atrybutów
- ▶ Bezpośredni dostęp do węzła:
  - /sklep/ceny[1]/gruszki
  - /sklep/ceny[2]/data

## Przykład

```
<sklep>
  <ceny>
    <data>
      <dzien>22</dzien>
      <miesiac>kwiecień</miesiac>
      <rok>2009</rok>
    </data>
    <gruszki kraj="Polska">3.25</gruszki>
    <cytryny kraj="Grecja">4.12</cytryny>
    <banany kraj="Ekwador">5.19</banany>
  </ceny>
```

//ceny/data/dzien

# Instrukcje i funkcje XPath

- ▶ Składnia użycia instrukcji języka XPath jest następująca:

**Instrukcja::nazwa elementu**  
lub  
**Instrukcja::funkcja**

- ▶ **Instrukcje** – wykorzystywane dla ułatwienia dostępu do węzłów w zależności od ich położenia względem węzła odniesienia
  - child – węzły dzieci
  - descendant – węzły potomkowie
  - parent – węzeł rodzic
  - self – bieżący
  - descendant-or-self – suma descendant i self
- ▶ **Funkcje:**
  - node() – wyszukanie węzła
  - text() – wyszukanie tekstu (treści)
  - comment() – wyszukanie komentarza
  - processing-instruction() – wyszukanie instrukcji przetwarzającej



slajd 22-24.sql

– Przykłady:

- `//ceny[2]/descendant::text()`
- `//ceny[2]/parent::node()`
- `//ceny/data/dzien/child::text()`

# XPath – dostęp do węzłów i atrybutów

## ▶ Skróty instrukcji języka XPath:

- `child::` – ‘`’`
- `parent::node()` – ‘`..`’
- `descendant-or-self::node()` – ‘`//`’
- `self::node()` – ‘`.`’
- `[position()=5]` – ‘`[5]`’ – tzw. Predykat
- `@nazwa_atrybutu` – odczytanie wartości atrybutu

## ▶ Przykłady:

- `//data/miesiac/text()`
- `/sklep/ceny[position()=1]/data/miesiac/text()`
- `/sklep/ceny[1]/data/miesiac/child::text()`
- `//banany/@kraj`

# XPath – operatory

## Operatory porównania i logiczne:

- |                      |         |
|----------------------|---------|
| • Równy              | - '='   |
| • Nierówny           | - '!='  |
| • Większy od         | - '>'   |
| • Większy lub równy  | - '>='  |
| • Mniejszy od        | - '<'   |
| • Mniejszy lub równy | - '<='  |
| • Iloczyn            | - 'and' |
| • Alternatywa        | - 'or'  |
| • Zaprzeczenie       | - 'not' |

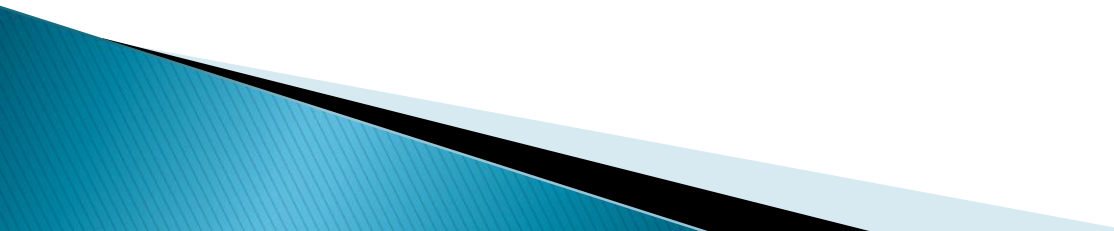
## Operatory Matematyczne:

- |               |         |
|---------------|---------|
| • Dodawanie   | - '+'   |
| • Odejmowanie | - '-'   |
| • Mnożenie    | - '*'   |
| • Dzielenie   | - 'div' |
| • Modulo      | - 'mod' |

- **Przykłady - wyszukiwanie węzłów o określonej zawartości**
  - //ceny[gruszki < 5.10]
  - //ceny[cytryny >2.40]
  - //data[dzien=22]/rok

# XPath – funkcje tekstowe

## Operacje dotyczące tekstu:

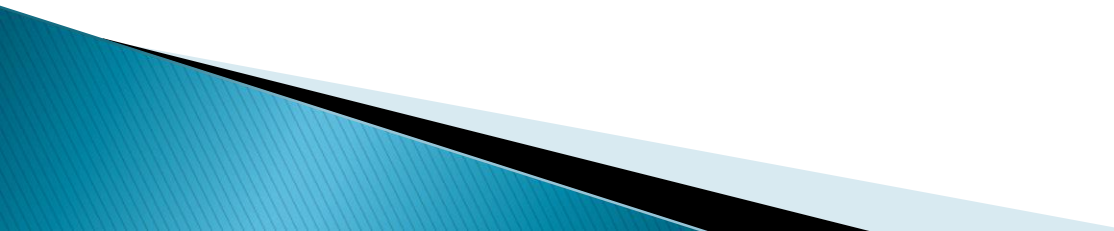
- ▶ `concat(tekst1, tekst2..)` – łączenie tekstów
  - ▶ `contains(tekst, wzór)` – sprawdzenie czy tekst zawiera wzór
  - ▶ `normalize-space(tekst)` – usuwa zbędne spacje
  - ▶ `starts-with(tekst, wzór)` – sprawdzenie czy tekst zaczyna się od wzoru
  - ▶ `string-length(tekst)` – obliczenie długości tekstu
  - ▶ `substring(tekst, początek, długość)` – wycina z tekstu od wskazanego miejsca określona ilość znaków
  - ▶ `substring-after(tekst, wzor)` – zwraca tekst po wystąpieniu określonego wzorca
  - ▶ `substring-before(tekst, wzor)` – zwraca tekst występujący przed wzorcem
  - ▶ `translate(tekst, stare, nowe)` – zamienia w tekście stare znaki na nowe
- 

# XPath – inne tekstowe

## Operacje dotyczące liczb:

- ▶ `ceiling(liczba)` – zwraca liczbę całkowitą większą bądź równą danej liczbie
- ▶ `floor(liczba)` – zwraca liczbę całkowitą mniejszą bądź równą danej liczbie
- ▶ `round(liczba)` – zwraca najbliższą liczbę całkowitą
- ▶ `sum(zbiór.węzłów)` – sumuje wartości węzłów

## Pozostałe operacje:

- ▶ `count(element)` – zwraca liczbę wystąpień elementów
  - ▶ `position()` – podaje pozycję elementu
  - ▶ `last()` – podaje pozycję ostatniego elementu
- 

# XPath – przykłady użycia funkcji

- Ceny gruszek z kraju "\*\*\*\*cja"
  - `//ceny/gruszki[substring(@kraj,5)="cja"]`
- Czy gruszek jest więcej niż 2:
  - `count(//gruszki)>2`
- Obliczenie średniej ceny gruszek:
  - `sum(//gruszki) div count(//gruszki)`
- Obliczenie średniej ceny gruszek 24-go:
  - `sum(//gruszki[../data/dzien=24]) div count(//gruszki[../data/dzien=24])`

# Język XQuery i metody danych

# Język XQuery

- ▶ Służy do tworzenia zapytań operujących na danych typu XML
- ▶ Wykorzystuje ścieżki XPath do węzłów w danych xml
- ▶ Może zawierać instrukcje logiczne i sterujące
- ▶ Obsługuje funkcje agregujące: SUM, Count, Avg, Min i Max
- ▶ Funkcje dostępu do relacyjnych kolumn i zmiennych:
  - sql:variable() i sql:column()

# Przykłady użycia XPath i XQuery

Dodanie nowego element nadrzędnego (root) danych xml z wykorzystaniem XQuery:

```
DECLARE @doc XML = '<pracownik>Jan Kowalski</pracownik>'
```

```
SELECT @doc = @doc.query('<firma>{ /pracownik }</firma>')
```

```
SELECT @doc
```

Nawias klamrowy służy oddzieleniu literałów od wyrażeń zwracających dynamicznie fragmenty dokumentu xml

## Wynik:

```
<firma>  
  <pracownik>Jan Kowalski</pracownik>  
</firma>
```

# Przykłady użycia XPath i XQuery

```
DECLARE @doc XML = '
```

```
<root>
```

```
  <element nr="1">Jan</element>
```

```
  <element nr="2">studiuje</element>
```

```
  <element nr="3">na</element>
```

```
  <element nr="4">pierwszym</element>
```

```
  <element nr="5">roku</element>
```

```
</root>'
```

```
SELECT @doc.query(' //element')
```

## Wynik:

```
<element nr="1">Jan</element>
```

```
<element nr="2">studiuje</element>
```

```
<element nr="3">na</element>
```

```
<element nr="4">pierwszym</element>
```

```
<element nr="5">roku</element>
```

# Przykłady użycia XPath i XQuery

## Przykład 3:

```
DECLARE @doc XML = '
```

```
<root>
```

```
  <element nr="1">Jan</element>
```

```
  <element nr="2">studiuje</element>
```

```
  <element nr="3">na</element>
```

```
  <element nr="4">pierwszym</element>
```

```
  <element nr="5">roku</element>
```

```
</root>'
```

```
SELECT @doc.query(' //element[2]/text()')
```

Wynik:

studiuje

# Przykłady użycia XPath i XQuery

## Przykład 4:

```
DECLARE @doc XML = '
```

```
<root>
```

```
  <element nr="1">Jan</element>
```

```
  <element nr="2">studiuje</element>
```

```
  <element nr="3">na</element>
```

```
  <element nr="4">pierwszym</element>
```

```
  <element nr="5">roku</element>
```

```
</root>'
```

```
SELECT @doc.query('//element[@nr > 2]')
```

## Wynik:

```
<element nr="3">na</element>
```

```
<element nr="4">pierwszym</element>
```

```
<element nr="5">roku</element>
```

# XQuery – przykład wykorzystania sql:variable

Funkcja **sql:variable** zwraca zawartość zmiennej do wyrażenia XQuery

## Przykład:

```
DECLARE @i INT = 5
DECLARE @doc XML = '
<root>
<element nr="1">Jan</element>
<element nr="2">studiuje</element>
<element nr="3">na</element>
<element nr="4">pierwszym</element>
<element nr="5">roku</element>
</root>'
```

```
SELECT @doc.query('//element[@nr = sql:variable("@i")]')
```

## Wynik:

```
<element nr="5">roku</element>
```

# Język XQuery – klauzula FOR

- ▶ Klauzula FOR może zawierać wiele zmiennych
- ▶ Klauzula FOR iteruje po każdej wartości uzyskanej w wyniku ewaluacji wyrażenia przypisanego do zmiennej.
- ▶ Użycie wielu zmiennych w klauzuli FOR powoduje, że dla każdej wartości zmiennej nadrzędnej wykonywana jest iteracja za pomocą zmiennej podrzędnej.

# Język XQuery – klauzula FOR – przykład

```
declare @x xml
set @x='
<Instrukcja Produkt="Rower" >
  <Naprawa Usterka="Dziurawa dętka" >
    <Krok>Zdejmij koło</Krok>
    <Krok>Zdejmij oponę</Krok>
    <Krok>Wymień detkę</Krok>
    <Krok>Założ oponę</Krok>
    <Krok>Napompuj koło</Krok>
    <Krok>Założ koło</Krok>
  </Naprawa>
  <Naprawa Usterka="Brak światła" >
    <Krok>Wyjmij baterie</Krok>
    <Krok>Wstaw nowe baterie</Krok>
  </Naprawa>
</Instrukcja>'

SELECT @x.query(
  '<Root>
  {
  for $k in /Instrukcja/Naprawa[1]/Krok return string($k)
  }
  </Root>')

```

xmlresult2.xml

<Root>Zdejmij koło Zdejmij oponę Wymień detkę Założ oponę Napompuj koło Założ koło</Root>

# Język XQuery – klauzula FOR – przykład 2

```
declare @x xml
set @x=
'<Faktury>
  <Faktura>
    <Klient>Jan Narab</Klient>
    <Pozycje>
      <Pozycja NumerProduktu="2" Cena="1.99" Ilość="1" />
      <Pozycja NumerProduktu="3" Cena="2.99" Ilość="2" />
      <Pozycja NumerProduktu="5" Cena="1.99" Ilość="1" />
    </Pozycje>
  </Faktura>
  <Faktura>
    <Klient>Marian Kesur</Klient>
    <Pozycje>
      <Pozycja NumerProduktu="2" Cena="1.99" Ilość="1"/>
    </Pozycje>
  </Faktura>
</Faktury>'
```

```
--Metoda 1
SELECT @x.query(
  '<Klienci>
  {
    for $Faktura in /Faktury/Faktura
    return $Faktura/Klient
  }
  </Klienci>')
```

```
--Metoda 2
SELECT @x.query(
  '<Klienci>
  {
    /Faktury/Faktura/Klient
  }
  </Klienci>')
```

```
xmlresult6.xml
<Klienci>
  <Klient>Jan Narab</Klient>
  <Klient>Marian Kesur</Klient>
</Klienci>
```

# Typ danych XML – metody XQuery

- ▶ Lista metod używanych w zapytaniach XQuery do dokumentów typu XML:
  - query – pobiera węzły
  - value – zwraca wartości
  - exist – sprawdza istnienie elementu
  - nodes – rozbija dane w formacie XML na poszczególne wiersze w zbiorze wynikowym
  - modify – modyfikuje sekcje danych w formacie XML

# Typ danych XML – metoda **query**

- ▶ Umożliwia zwrócenie fragmentów danych za pomocą zapytań XQuery
- ▶ Przyjmuje 1 parametr
- ▶ Zwraca wartość typu XML

# Typ danych XML – metoda **query** – przykład 1

```
DECLARE @doc xml
```

```
SET @doc = '
```

```
<Faktury>
```

```
<Faktura>
```

```
<Klient>Jan Narab</Klient>
```

```
<Pozycje>
```

```
<Pozycja NumerProduktu="2" Cena="1.99" Ilość="1" />
```

```
<Pozycja NumerProduktu="3" Cena="2.99" Ilość="2" />
```

```
<Pozycja NumerProduktu="5" Cena="1.99" Ilość="1" />
```

```
</Pozycje>
```

```
</Faktura>
```

```
<Faktura>
```

```
<Klient>Marian Kesur</Klient>
```

```
<Pozycje>
```

```
<Pozycja NumerProduktu="2" Cena="1.99" Ilość="1"/>
```

```
</Pozycje>
```

```
</Faktura>
```

```
</Faktury>
```

```
'
```

```
SELECT @doc.query('///Faktura[1]/Klient/text()') as NazwiskoKlienta
```

	NazwiskoKlienta
1	<a href="#">Jan Narab</a>

# Typ danych XML – metoda **query** – przykład 2

```
DECLARE @doc xml
```

```
SET @doc = '
```

```
<Faktury>
```

```
<Faktura>
```

```
<Klient>Jan Narab</Klient>
```

```
<Pozycje>
```

```
<Pozycja NumerProduktu="2" Cena="1.99" Ilość="1" />
```

```
<Pozycja NumerProduktu="3" Cena="2.99" Ilość="2" />
```

```
<Pozycja NumerProduktu="5" Cena="1.99" Ilość="1" />
```

```
</Pozycje>
```

```
</Faktura>
```

```
<Faktura>
```

```
<Klient>Marian Kesur</Klient>
```

```
<Pozycje>
```

```
<Pozycja NumerProduktu="2" Cena="1.99" Ilość="1"/>
```

```
</Pozycje>
```

```
</Faktura>
```

```
</Faktury>
```

```
'
```

```
SELECT @doc.query(' //Faktura[1]/Pozycje/Pozycja')
```

xmlresult8.xml

```
<Pozycja NumerProduktu="2" Cena="1.99" Ilość="1" />  
<Pozycja NumerProduktu="3" Cena="2.99" Ilość="2" />  
<Pozycja NumerProduktu="5" Cena="1.99" Ilość="1" />
```

# Typ danych XML – metoda **query** – przykład 3

```
DECLARE @doc xml
```

```
SET @doc = '
```

```
<Faktury>
```

```
  <Faktura>
```

```
    <Klient>Jan Narab</Klient>
```

```
    <Pozycje>
```

```
      <Pozycja NumerProduktu="2" Cena="1.99" Ilość="1" />
```

```
      <Pozycja NumerProduktu="3" Cena="2.99" Ilość="2" />
```

```
      <Pozycja NumerProduktu="5" Cena="1.99" Ilość="1" />
```

```
    </Pozycje>
```

```
  </Faktura>
```

```
  <Faktura>
```

```
    <Klient>Marian Kesur</Klient>
```

```
    <Pozycje>
```

```
      <Pozycja NumerProduktu="2" Cena="1.99" Ilość="1"/>
```

```
    </Pozycje>
```

```
  </Faktura>
```

```
</Faktury>
```

```
'
```

```
SELECT @doc.query(' //Faktura[1]/Pozycje/Pozycja[@Cena=1.99]')
```

xmlresult18.xml

```
<Pozycja NumerProduktu="2" Cena="1.99" Ilość="1" />  
<Pozycja NumerProduktu="5" Cena="1.99" Ilość="1" />
```

# Typ danych XML – metoda **query** – przykład 4

```
declare @x xml
set @x=
'<Faktury>
  <Faktura>
    <Klient>Jan Narab</Klient>
    <Pozycje>
      <Pozycja NumerProduktu="2" Cena="1.99" Ilość="1" />
      <Pozycja NumerProduktu="3" Cena="2.99" Ilość="2" />
      <Pozycja NumerProduktu="5" Cena="1.99" Ilość="1" />
    </Pozycje>
  </Faktura>
  <Faktura>
    <Klient>Marian Kesur</Klient>
    <Pozycje>
      <Pozycja NumerProduktu="2" Cena="1.99" Ilość="1"/>
    </Pozycje>
  </Faktura>
</Faktury>'
```

```
SELECT @x.query('/Faktury/Faktura/Klient')
```

xmlresult22.xml

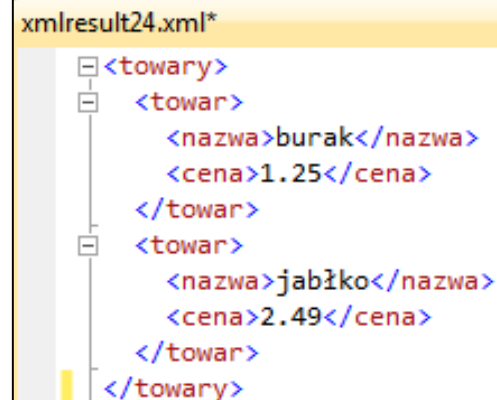
```
<Klient>Jan Narab</Klient>
<Klient>Marian Kesur</Klient>
```

# Typ danych XML – metoda query – przykład 5

Klauzula LET pozwala przypisać wartości do zmiennych, aby je później wykorzystać przy budowie wynikowego dokumentu XML

```
declare @x xml
set @x='
<produkty>
  <produkt>
    <nazwa>burak</nazwa>
    <opis>buraczek czerwony</opis>
    <rodzaj>warzywo</rodzaj>
    <ilosc>155</ilosc>
    <cena>1.25</cena>
  </produkt>
  <produkt>
    <nazwa>jabłko</nazwa>
    <opis>słodkie, soczyste, czerwone</opis>
    <rodzaj>owoc</rodzaj>
    <ilosc>105</ilosc>
    <cena>2.49</cena>
  </produkt>
  <produkt>
    <nazwa>gruszka</nazwa>
    <opis>słodka, soczysta, miękka</opis>
    <rodzaj>owoc</rodzaj>
    <ilosc>19</ilosc>
    <cena>4.59</cena>
  </produkt>
</produkty>'
```

```
Select @x.query('
<towary>{
  for $p in /produkty/produkt
    let $n := $p/nazwa[1]
    let $c := $p/cena[1]
    where $c < 2.50
    order by $n
  return
    <towar>
      <nazwa>{$n/text()}</nazwa>
      <cena>{$c/text()}</cena>
    </towar>
}
</towary>')
```



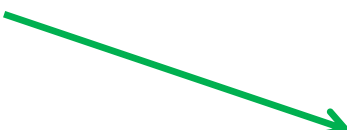
```
xmlresult24.xml*
<?xml version='1.0' encoding='UTF-8' />
< towary >
  < towar >
    < nazwa >burak< /nazwa >
    < cena >1.25< /cena >
  < /towar >
  < towar >
    < nazwa >jabłko< /nazwa >
    < cena >2.49< /cena >
  < /towar >
< /towary >
```

# Typ danych XML – metoda **value**

Pozwala zwrócić z XML wartości skalarne przekonwertowane na inny typ danych SQL Server

```
DECLARE @doc xml
SET @doc = '
<Faktury>
  <Faktura>
    <Klient>Jan Narab</Klient>
    <Pozycje>
      <Pozycja NumerProduktu="2" Cena="1.99" Ilość="1" />
      <Pozycja NumerProduktu="3" Cena="2.99" Ilość="2" />
      <Pozycja NumerProduktu="5" Cena="1.99" Ilość="1" />
    </Pozycje>
  </Faktura>
  <Faktura>
    <Klient>Marian Kesur</Klient>
    <Pozycje>
      <Pozycja NumerProduktu="2" Cena="1.97" Ilość="1"/>
    </Pozycje>
  </Faktura>
</Faktury>'

DECLARE @cena Decimal(3,2)
SELECT @cena = @doc.value('(/*Faktura[2]/Pozycje/Pozycja/@Cena)[1]', 'Decimal(3,2)')
SELECT @cena as Cena
```



Results		Messages	
Cena			
1	1.97		

# Typ danych XML – metoda **exist**

- ▶ Pozwala wykrywać w danych XML istnienie węzłów spełniających zadany warunek określony wyrażeniem Xquery
- ▶ Zwraca wartość typu bit:
  - 1 – gdy istnieje co najmniej 1 węzeł spełniający warunek
  - 0 – gdy nie istnieje żaden węzeł spełniający zadany warunek
  - NULL – gdy w danych XML występuje wartość NULL

# Typ danych XML – metoda **exist** – przykład 1

```
DECLARE @doc xml
```

```
SET @doc = '
```

```
<Faktury>
```

```
  <Faktura>
```

```
    <Klient>Jan Narab</Klient>
```

```
    <Pozycje>
```

```
      <Pozycja NumerProduktu="2" Cena="1.99" Ilość="1" />
```

```
      <Pozycja NumerProduktu="3" Cena="2.99" Ilość="2" />
```

```
      <Pozycja NumerProduktu="5" Cena="1.99" Ilość="1" />
```

```
    </Pozycje>
```

```
  </Faktura>
```

```
  <Faktura>
```

```
    <Klient>Marian Kesur</Klient>
```

```
    <Pozycje>
```

```
      <Pozycja NumerProduktu="2" Cena="1.97" Ilość="1"/>
```

```
    </Pozycje>
```

```
  </Faktura>
```

```
</Faktury>'
```

```
SELECT @doc.exist('//Faktura[Klient[1] eq "Jan Narab"]')
```

```
SELECT @doc.exist('//Faktura[Klient[2] eq "Jan Narab"]')
```

Prawda

Results		Messages	
(No column name)			
1	1		

Nieprawda

(No column name)	
1	0

# Typ danych XML – metoda **exist** – przykład 2

```
DECLARE @doc xml
```

```
SET @doc = '
```

```
<Faktury>
```

```
  <Faktura>
```

```
    <Klient>Jan Narab</Klient>
```

```
    <Pozycje>
```

```
      <Pozycja NumerProduktu="2" Cena="1.99" Ilość="1" />
```

```
      <Pozycja NumerProduktu="3" Cena="2.99" Ilość="2" />
```

```
      <Pozycja NumerProduktu="5" Cena="1.99" Ilość="1" />
```

```
    </Pozycje>
```

```
  </Faktura>
```

```
  <Faktura>
```

```
    <Klient>Marian Kesur</Klient>
```

```
    <Pozycje>
```

```
      <Pozycja NumerProduktu="2" Cena="1.97" Ilość="1"/>
```

```
    </Pozycje>
```

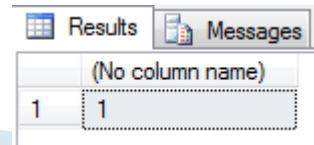
```
  </Faktura>
```

```
</Faktury>'
```

```
DECLARE @b bit
```

```
SET @b = @doc.exist('/Faktury/Faktura/Pozycje/Pozycja[@Cena =2.99 ]')
```

```
SELECT @b
```



The screenshot shows the SQL Server Enterprise Manager interface. The 'Results' tab is active, displaying a single row of data. The first column is labeled '1' and the second column is labeled '1'. A green arrow points from the text 'Istnieje pozycja z ceną 2.99' to the 'Messages' tab, which is currently empty.

(No column name)	
1	1

Istnieje pozycja z ceną 2.99

# Typ danych XML – metoda **nodes**

- ▶ Metoda zwraca kolumnę węzłów, czyli kolumnę xml, którą możemy odpytywać np. metodą query

```
DECLARE @doc xml
SET @doc = '
<Faktury>
  <Faktura>
    <Klient>Jan Narab</Klient>
    <Pozycje>
      <Pozycja NumerProduktu="2" Cena="1.99" Ilość="1" />
      <Pozycja NumerProduktu="3" Cena="2.99" Ilość="2" />
      <Pozycja NumerProduktu="5" Cena="1.99" Ilość="1" />
    </Pozycje>
  </Faktura>
</Faktury>
<Faktura>
  <Klient>Marian Kesur</Klient>
  <Pozycje>
    <Pozycja NumerProduktu="2" Cena="1.97" Ilość="1"/>
  </Pozycje>
</Faktura>
'

SELECT T.c.query('.') as wynik
from @doc.nodes('/Faktury/Faktura/Pozycje')T(c)
```

Powstanie tablica T z kolumną c.

Ponieważ w klauzuli SELECT podano '.', zapytanie zwróci bieżące elementy z nagłówkiem kolumny „wynik”:

wynik	
1	<Pozycje><Pozycja NumerProduktu="2" Cena="1.99"...
2	<Pozycje><Pozycja NumerProduktu="2" Cena="1.97"...

wynik23.xml
<Pozycje> <Pozycja NumerProduktu="2" Cena="1.99" Ilość="1" /> <Pozycja NumerProduktu="3" Cena="2.99" Ilość="2" /> <Pozycja NumerProduktu="5" Cena="1.99" Ilość="1" /> </Pozycje>

wynik25.xml
<Pozycje> <Pozycja NumerProduktu="2" Cena="1.97" Ilość="1" /> </Pozycje>

## Przykład 2

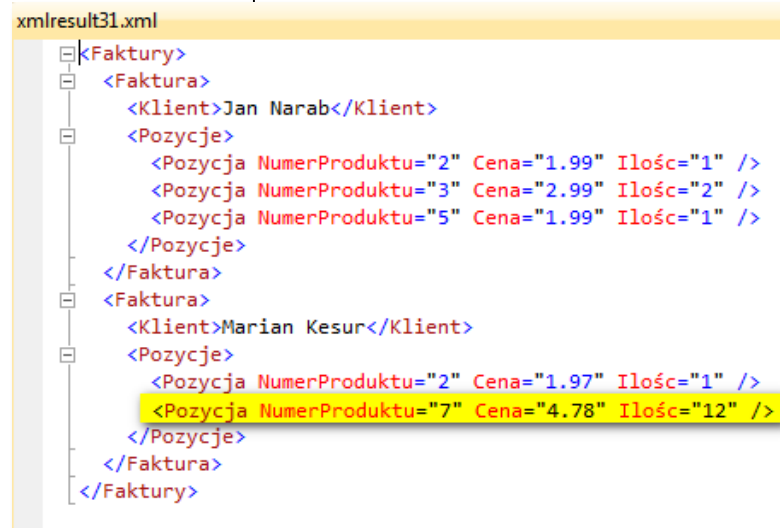
```
SELECT T.c.query('//Pozycja') as wynik
from @doc.nodes('/Faktury/Faktura/Pozycje')T(c)
```

# Typ danych XML – metoda **modify (insert)**

- ▶ Metoda umożliwia wstawianie, modyfikację i usuwanie węzłów dokumentu XML

```
DECLARE @doc xml
SET @doc = '
<Faktury>
  <Faktura>
    <Klient>Jan Narab</Klient>
    <Pozycje>
      <Pozycja NumerProduktu="2" Cena="1.99" Ilość="1" />
      <Pozycja NumerProduktu="3" Cena="2.99" Ilość="2" />
      <Pozycja NumerProduktu="5" Cena="1.99" Ilość="1" />
    </Pozycje>
  </Faktura>
  <Faktura>
    <Klient>Marian Kesur</Klient>
    <Pozycje>
      <Pozycja NumerProduktu="2" Cena="1.97" Ilość="1"/>
    </Pozycje>
  </Faktura>
</Faktury>
'

SET @doc.modify('
insert <Pozycja NumerProduktu="7" Cena="4.78" Ilość="12" />
after(/Faktury/Faktura[2]/Pozycje/Pozycja)[1]')-- alternatywą dla after jest before
SELECT @doc
```



```
xmlresult31.xml
<Faktury>
  <Faktura>
    <Klient>Jan Narab</Klient>
    <Pozycje>
      <Pozycja NumerProduktu="2" Cena="1.99" Ilość="1" />
      <Pozycja NumerProduktu="3" Cena="2.99" Ilość="2" />
      <Pozycja NumerProduktu="5" Cena="1.99" Ilość="1" />
    </Pozycje>
  </Faktura>
  <Faktura>
    <Klient>Marian Kesur</Klient>
    <Pozycje>
      <Pozycja NumerProduktu="2" Cena="1.97" Ilość="1" />
      <Pozycja NumerProduktu="7" Cena="4.78" Ilość="12" />
    </Pozycje>
  </Faktura>
</Faktury>
```

# Typ danych XML – metoda **modify (replace)**

- ▶ Metoda umożliwia wstawianie, modyfikację i usuwanie węzłów dokumentu XML

```
DECLARE @doc xml
SET @doc = '
<Faktury>
  <Faktura>
    <Klient>Jan Narab</Klient>
    <Pozycje>
      <Pozycja NumerProduktu="2" Cena="1.99" Ilość="1" />
      <Pozycja NumerProduktu="3" Cena="2.99" Ilość="2" />
      <Pozycja NumerProduktu="5" Cena="1.99" Ilość="1" />
    </Pozycje>
  </Faktura>
  <Faktura>
    <Klient>Marian Kesur</Klient>
    <Pozycje>
      <Pozycja NumerProduktu="2" Cena="1.97" Ilość="1"/>
    </Pozycje>
  </Faktura>
</Faktury>
'

SET @doc.modify('
replace value of (/Faktury/Faktura[2]/Klient/text())[1]
with "Nikodem Buła" ')
SELECT @doc
```

xmlresult33.xml

```
<Faktury>
  <Faktura>
    <Klient>Jan Narab</Klient>
    <Pozycje>
      <Pozycja NumerProduktu="2" Cena="1.99" Ilość="1" />
      <Pozycja NumerProduktu="3" Cena="2.99" Ilość="2" />
      <Pozycja NumerProduktu="5" Cena="1.99" Ilość="1" />
    </Pozycje>
  </Faktura>
  <Faktura>
    <Klient>Nikodem Buła</Klient>
    <Pozycje>
      <Pozycja NumerProduktu="2" Cena="1.97" Ilość="1" />
    </Pozycje>
  </Faktura>
</Faktury>
```

# Typ danych XML – metoda **modify (delete)**

- ▶ Metoda umożliwia wstawianie, modyfikację i usuwanie węzłów dokumentu XML

```
DECLARE @doc xml
SET @doc = '
<Faktury>
  <Faktura>
    <Klient>Jan Narab</Klient>
    <Pozycje>
      <Pozycja NumerProduktu="2" Cena="1.99" Ilość="1" />
      <Pozycja NumerProduktu="3" Cena="2.99" Ilość="2" />
      <Pozycja NumerProduktu="5" Cena="1.99" Ilość="1" />
    </Pozycje>
  </Faktura>
</Faktury>
<Faktura>
  <Klient>Marian Kesur</Klient>
  <Pozycje>
    <Pozycja NumerProduktu="2" Cena="1.97" Ilość="1"/>
  </Pozycje>
</Faktura>
</Faktury>
'

SET @doc.modify('
delete (/Faktury/Faktura/Pozycje/Pozycja)[1]
')

SELECT @doc
```

Pozycja pierwsza w pierwszej fakturze została usunięta

xmlresult36.xml

```
<Faktury>
  <Faktura>
    <Klient>Jan Narab</Klient>
    <Pozycje>
      <Pozycja NumerProduktu="3" Cena="2.99" Ilość="2" />
      <Pozycja NumerProduktu="5" Cena="1.99" Ilość="1" />
    </Pozycje>
  </Faktura>
  <Faktura>
    <Klient>Marian Kesur</Klient>
    <Pozycje>
      <Pozycja NumerProduktu="2" Cena="1.97" Ilość="1" />
    </Pozycje>
  </Faktura>
</Faktury>
```

# Klauzula FOR XML

# Klauzula FOR XML

- ▶ Zwraca dane relacyjne (wynik zapytania T-SQL) w postaci XML

FOR XML *mode* [, ELEMENTS] [, *inne opcje*] [ROOT]

Gdzie *mode*:

- RAW
  - AUTO
  - PATH
  - EXPLICIT
- 

Klauzula FOR XML **RAW**

# Klauzula FOR XML RAW – generowanie schematu

- ▶ Klauzula FOR XML RAW – zwraca wszystkie wiersze w postaci prostych dokumentów xml, gdzie kolumny są reprezentowane w postaci atrybutów (jak w FOR XML AUTO)
- ▶ Każdy rekord jest reprezentowany w postaci osobnego dokumentu xml
- ▶ Dane, jeśli nawet pochodzą z 2 różnych tabel (relacji parent-child) reprezentowane będą w postaci płaskich rekordów w formacie xml
- ▶ Finalny wynik nie jest do końca poprawny – brakuje elementu „root” w hierarchii

```
select * from Pracownicy for xml raw
```

```
select * from Pracownicy for xml raw
```

100 %

Results Messages

XML\_F52E2B61-18A1-11d1-B105-00805F49916B

1 <row NumerPracownika="1" Imię="Maria" Nazwisko="...

XML\_F52E2B61-18A1-11d1-B105-00805F49916B13.xml

```
<row NumerPracownika="1" Imię="Maria" Nazwisko="Borucka" Stanowisko="Prezes" Pensja="1100"
<row NumerPracownika="2" Imię="Jeremiasz" Nazwisko="Drewniak" Stanowisko="Dyrektor" Pensja="1100"
<row NumerPracownika="3" Imię="Ewa" Nazwisko="Braun" Stanowisko="Agent marketingowy" Pensja="1100"
<row NumerPracownika="4" Imię="Franciszek" Nazwisko="Malowski" Stanowisko="Koordynator" Pensja="1100"
<row NumerPracownika="5" Imię="Adrianna" Nazwisko="Snycerska" Stanowisko="Spedytor" Pensja="1100"
<row NumerPracownika="6" Imię="Urszula" Nazwisko="Halicka" Stanowisko="Sprzedawca" Pensja="1100"
<row NumerPracownika="7" Imię="Jan" Nazwisko="Orłóń" Stanowisko="Specjalista finansowy" Pensja="1100"
<row NumerPracownika="8" Imię="Karol" Nazwisko="Bjuty" Stanowisko="Dyrektor" Pensja="7800"
<row NumerPracownika="9" Imię="Daniel" Nazwisko="Żylecki" Stanowisko="Projektant" Pensja="1100"
<row NumerPracownika="10" Imię="Donata" Nazwisko="Pietrzyk" Stanowisko="Sprzedawca" Pensja="1100"
<row NumerPracownika="11" Imię="Katarzyna" Nazwisko="Krauzy" Stanowisko="Prac. administr." Pensja="1100"
<row NumerPracownika="13" Imię="Antoni" Nazwisko="Pierchała" Stanowisko="Sprzedawca" Pensja="1100"
<row NumerPracownika="14" Imię="Henryk" Nazwisko="Czyński" Stanowisko="Spedytor" Pensja="1100"
<row NumerPracownika="15" Imię="Maciej" Nazwisko="Dąbrowski" Stanowisko="Stażysta" Podleg.
```

# Klauzula FOR XML RAW – generowanie schematu

- Jeśli chcemy narzucić własną nazwę elementów (zamiast „row”), wystarczy do poprzedniego zapytania dodać w nawiasie proponowaną nazwę:

```
select * from Pracownicy for xml raw ('Pracownik')
```

The screenshot shows a SQL query window with the following text:

```
select * from Pracownicy for xml raw ('Pracownik')
```

Below the query window, the 'Results' tab is active, displaying the XML output for the first row. The XML is as follows:

```
XML_F52E2B61-18A1-11d1-B105-00805F49916B
1 <Pracownik NumerPracownika="1" Imię="Maria" Nazw...
```

Below the results, the 'Messages' tab is active, displaying the XML output for the first row. The XML is as follows:

```
XML_F52E2B61-18A1-11d1-B105-00805F49916B2.xml
<Pracownik NumerPracownika="1" Imię="Maria" Nazwisko="Borucka" Stanowisko="Prezes" Pensja="11000.00" />
<Pracownik NumerPracownika="2" Imię="Jeremiasz" Nazwisko="Drewniak" Stanowisko="Dyrektor" Pensja="7800.00" />
<Pracownik NumerPracownika="3" Imię="Ewa" Nazwisko="Braun" Stanowisko="Agent marketingowy" Pensja="7800.00" />
<Pracownik NumerPracownika="4" Imię="Franciszek" Nazwisko="Malowski" Stanowisko="Koordynator sped" Pensja="7800.00" />
<Pracownik NumerPracownika="5" Imię="Adrianna" Nazwisko="Snycerska" Stanowisko="Spedytor" Pensja="7800.00" />
<Pracownik NumerPracownika="6" Imię="Urszula" Nazwisko="Halicka" Stanowisko="Sprzedawca" Pensja="7800.00" />
<Pracownik NumerPracownika="7" Imię="Jan" Nazwisko="Orłóń" Stanowisko="Specjalista finansowy" Pensja="7800.00" />
<Pracownik NumerPracownika="8" Imię="Karol" Nazwisko="Bjuty" Stanowisko="Dyrektor" Pensja="7800.00" />
<Pracownik NumerPracownika="9" Imię="Daniel" Nazwisko="Żyłecki" Stanowisko="Projektant" Pensja="2400.00" />
<Pracownik NumerPracownika="10" Imię="Donata" Nazwisko="Pietrzyk" Stanowisko="Sprzedawca" Pensja="7800.00" />
<Pracownik NumerPracownika="11" Imię="Katarzyna" Nazwisko="Krauzy" Stanowisko="Prac. administrac." Pensja="7800.00" />
<Pracownik NumerPracownika="13" Imię="Antoni" Nazwisko="Pierchała" Stanowisko="Sprzedawca" Pensja="7800.00" />
<Pracownik NumerPracownika="14" Imię="Henryk" Nazwisko="Czyński" Stanowisko="Spedytor" Pensja="2200.00" />
<Pracownik NumerPracownika="15" Imię="Marek" Nazwisko="Dąbrowski" Stanowisko="Stażysta" Podlega=...
```

# Klauzula FOR XML RAW – generowanie schematu

- ▶ Chcąc dodać element główny, należy dodać klauzulę ROOT

```
select * from Pracownicy for xml raw, root('Pracownicy')
```

The screenshot displays the SQL Server Enterprise Manager interface. At the top, a query window shows the following SQL statement:

```
select * from Pracownicy for xml raw, root('Pracownicy')
```

Below the query window, the 'Results' tab is active, showing a single row of XML data. The XML output is:

```
<Pracownicy><row NumerPracownika="1" Imię="Mari..."
```

Below the results, a file explorer shows the generated XML file: XML\_F52E2B61-18A1-11d1-B105-00805F49916B14.xml. The file content is displayed as follows:

```
<Pracownicy>  
<row NumerPracownika="1" Imię="Maria" Nazwisko="Borucka" Stanowisko="Prezes" Pen  
<row NumerPracownika="2" Imię="Jeremiasz" Nazwisko="Drewniak" Stanowisko="Dyrekt  
<row NumerPracownika="3" Imię="Ewa" Nazwisko="Braun" Stanowisko="Agent marketing  
<row NumerPracownika="4" Imię="Franciszek" Nazwisko="Malowski" Stanowisko="Koort  
<row NumerPracownika="5" Imię="Adrianna" Nazwisko="Snycerska" Stanowisko="Spedyt  
<row NumerPracownika="6" Imię="Urszula" Nazwisko="Halicka" Stanowisko="Sprzedawc  
<row NumerPracownika="7" Imię="Jan" Nazwisko="Orłóń" Stanowisko="Specjalista fin  
<row NumerPracownika="8" Imię="Karol" Nazwisko="Bjuty" Stanowisko="Dyrektor" Pen  
<row NumerPracownika="9" Imię="Daniel" Nazwisko="Żylecki" Stanowisko="Projektant  
<row NumerPracownika="10" Imię="Donata" Nazwisko="Pietrzyk" Stanowisko="Sprzedaw  
<row NumerPracownika="11" Imię="Katarzyna" Nazwisko="Krauzy" Stanowisko="Prac.  
<row NumerPracownika="13" Imię="Antoni" Nazwisko="Pierchała" Stanowisko="Sprzed  
<row NumerPracownika="14" Imię="Henryk" Nazwisko="Czyński" Stanowisko="Spedytor  
<row NumerPracownika="15" Imię="Maciej" Nazwisko="Dąbrowski" Stanowisko="Staży  
</Pracownicy>
```

# Klauzula FOR XML RAW – generowanie schematu

- ▶ Poprzedni przykład, ale z dodaną klauzulą Elements – dane zamiast w atrybutach są reprezentowane w postaci odrębnych elementów:

```
select * from Pracownicy for xml raw, elements, root('Pracownicy')
```

The screenshot displays the SQL Server Enterprise Manager interface. At the top, the query `select * from Pracownicy for xml raw, elements, root('Pracownicy')` is entered. Below the query, the 'Results' tab shows the XML output. The first row of the result set is expanded, showing the XML structure. The XML is rooted at `<Pracownicy>` and contains two `<row>` elements. Each `<row>` element contains several sub-elements representing employee data.

XML\_F52E2B61-18A1-11d1-B105-00805F49916B

1 `<Pracownicy><row><NumerPracownika>1</NumerPracow...`

XML\_F52E2B61-18A1-11d1-B105-00805F49916B15.xml

```
<Pracownicy>
  <row>
    <NumerPracownika>1</NumerPracownika>
    <Imię>Maria</Imię>
    <Nazwisko>Borucka</Nazwisko>
    <Stanowisko>Prezes</Stanowisko>
    <Pensja>11000.00</Pensja>
    <Premia>1.0</Premia>
    <NazwaDziału>Marketing</NazwaDziału>
    <DataUrodzenia>1951-12-17T00:00:00</DataUrodzenia>
    <DataZatrudnienia>1990-05-30T00:00:00</DataZatrudnienia>
  </row>
  <row>
    <NumerPracownika>2</NumerPracownika>
    <Imię>Jeremiasz</Imię>
    <Nazwisko>Drewniak</Nazwisko>
    <Stanowisko>Dyrektor</Stanowisko>
    <Pensja>8000.00</Pensja>
```

Klauzula FOR XML **AUTO**

# Klauzula FOR XML AUTO (baza Cukierki)

- ▶ Klauzula FOR XML w trybie AUTO zwraca każdy wiersz w postaci odrębnego dokumentu XML, a wartości zapisane w kolumnach zwraca jako atrybuty

```
select * from Pracownicy for xml auto
```

select \* from Pracownicy for xml auto

Results Messages

XML\_F52E2B61-18A1-11d1-B105-00805F49916B

<Pracownicy NumerPracownika="1" Imię="Maria" Nazwisko="Borucka" Stanowisko="Prezes" Pensja="11000.00" Premia="1.0" NazwaDziału="Mar...

XML\_F52E2B61-18A1-11d1-B105-00805F49916B1.xml

```
<Pracownicy NumerPracownika="1" Imię="Maria" Nazwisko="Borucka" Stanowisko="Prezes" Pensja="11000.00" Premia="1.0" NazwaDziału="Marketing" />
<Pracownicy NumerPracownika="2" Imię="Jeremiasz" Nazwisko="Drewniak" Stanowisko="Dyrektor" Pensja="8000.00" Premia="0.8" NazwaDziału="Marketing" />
<Pracownicy NumerPracownika="3" Imię="Ewa" Nazwisko="Braun" Stanowisko="Agent marketingowy" Pensja="4000.00" Premia="0.6" NazwaDziału="Marketing" />
<Pracownicy NumerPracownika="4" Imię="Franciszek" Nazwisko="Malowski" Stanowisko="Koordynator spedycji" Pensja="4500.00" Premia="0.7" NazwaDziału="Logistyka" />
<Pracownicy NumerPracownika="5" Imię="Adrianna" Nazwisko="Snycerska" Stanowisko="Spedytor" Pensja="3200.00" Podlega="4" NazwaDziału="Logistyka" />
<Pracownicy NumerPracownika="6" Imię="Urszula" Nazwisko="Halicka" Stanowisko="Sprzedawca" Pensja="1900.00" Premia="0.8" NazwaDziału="Sprzedaż" />
<Pracownicy NumerPracownika="7" Imię="Jan" Nazwisko="Orłóń" Stanowisko="Specjalista finansowy" Pensja="3700.00" Premia="0.7" NazwaDziału="Finanse" />
<Pracownicy NumerPracownika="8" Imię="Karol" Nazwisko="Bjuty" Stanowisko="Dyrektor" Pensja="7800.00" Premia="0.7" Podlega="4" NazwaDziału="Finanse" />
<Pracownicy NumerPracownika="9" Imię="Daniel" Nazwisko="Żylecki" Stanowisko="Projektant" Pensja="2400.00" Podlega="3" NazwaDziału="Inżynieria" />
<Pracownicy NumerPracownika="10" Imię="Donata" Nazwisko="Pietrzyk" Stanowisko="Sprzedawca" Pensja="1900.00" Premia="0.8" NazwaDziału="Sprzedaż" />
<Pracownicy NumerPracownika="11" Imię="Katarzyna" Nazwisko="Krauz" Stanowisko="Prac. administrac." Pensja="2100.00" Podlega="4" NazwaDziału="Sprzedaż" />
<Pracownicy NumerPracownika="13" Imię="Antoni" Nazwisko="Pierchała" Stanowisko="Sprzedawca" Pensja="2100.00" Premia="0.9" NazwaDziału="Sprzedaż" />
<Pracownicy NumerPracownika="14" Imię="Henryk" Nazwisko="Czyński" Stanowisko="Spedytor" Pensja="2200.00" Podlega="4" NazwaDziału="Logistyka" />
<Pracownicy NumerPracownika="15" Imię="Maciej" Nazwisko="Dąbrowski" Stanowisko="Stażysta" Podlega="4" NazwaDziału="Logistyka" />
```

## Klauzula FOR XML AUTO– 2 tabele (baza Cukierki)

- ▶ W przeciwieństwie do trybu RAW tryb AUTO w przypadku 2 tabel powiązanych kluczem obcym **struktura wynikowego xml nie jest już płaska – automatycznie zagnieżdża elementy**
- ▶ Dodatkowo klauzula ROOT dodaje element główny

```
SELECT top(5) Klient.NumerKlienta, Klient.Nazwisko,  
              Zamówienia.DataZamówienia,  
              Zamówienia.ImięOdbiorcy,  
              Zamówienia.NazwiskoOdbiorcy  
FROM Klienci as Klient, Zamówienia  
WHERE Klient.NumerKlienta = Zamówienia.NumerKlienta  
ORDER BY Klient.NumerKlienta FOR XML AUTO, ROOT('Klienci')
```

# Klauzula FOR XML – 2 tabele (baza Cukierki)

```
SELECT top(5) Klient.NumerKlienta, Klient.Nazwisko, Zamówienia.DataZamówienia,  
Zamówienia.ImięOdbiorcy, Zamówienia.NazwiskoOdbiorcy  
FROM Klienci as Klient, Zamówienia  
WHERE Klient.NumerKlienta = Zamówienia.NumerKlienta  
ORDER BY Klient.NumerKlienta FOR XML AUTO, ROOT('Klienci')
```

100 %

Results Messages

	NumerKlienta	Nazwisko	DataZamówienia	ImięOdbiorcy	NazwiskoOdbiorcy
1	1	Jadowska	1998-01-07 00:00:00	Julia	Kania
2	1	Jadowska	1998-08-09 00:00:00	Teresa	Jadowska
3	2	Pludowski	1998-10-30 00:00:00	Katarzyna	Adamczyk
4	3	Karbowski	1998-10-13 00:00:00	Jan	Piasecki
5	4	Sitecki	1998-05-16 00:00:00	Waldemar	Jurecki

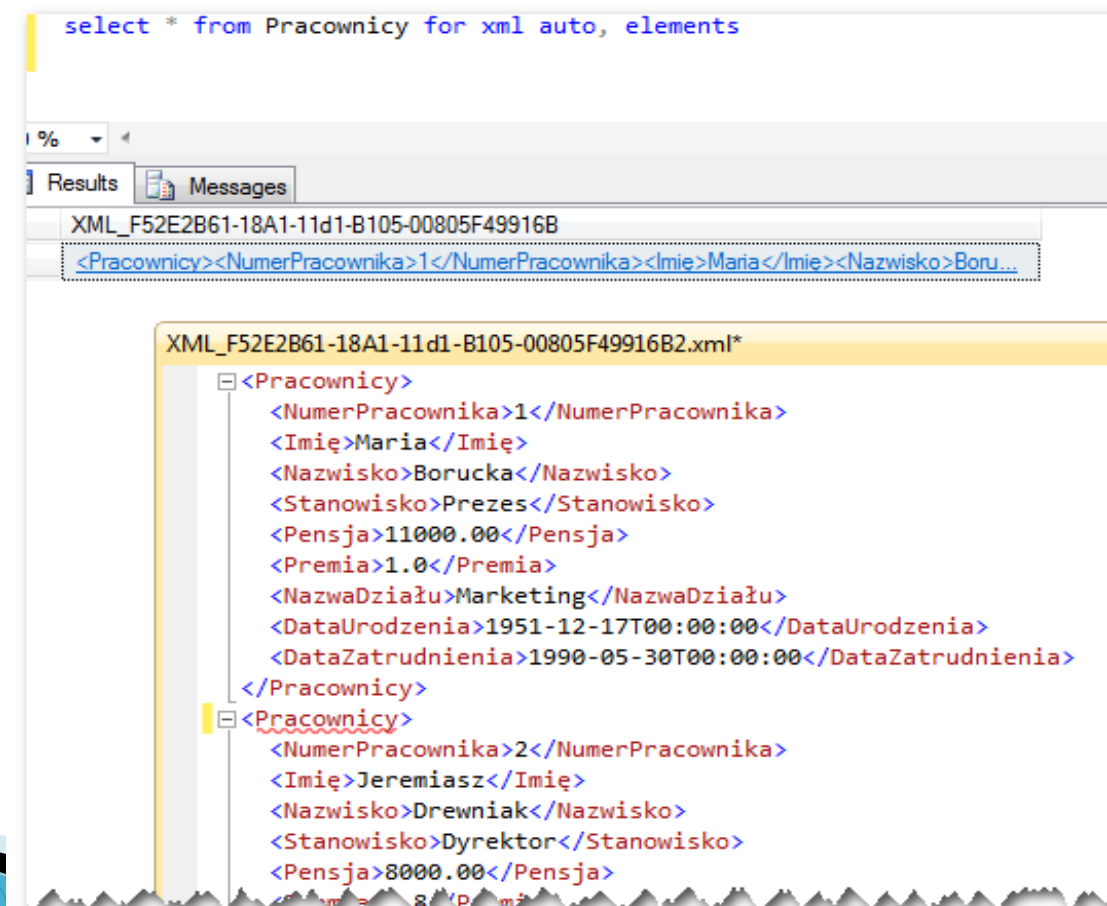
XML\_F52E2B61-18A1-11d1-B105-00805F49916B8.xml

```
<Klienci>  
  <Klient NumerKlienta="1" Nazwisko="Jadowska">  
    <Zamówienia DataZamówienia="1998-01-07T00:00:00" ImięOdbiorcy="Julia" NazwiskoOdbiorcy="Kania" />  
    <Zamówienia DataZamówienia="1998-08-09T00:00:00" ImięOdbiorcy="Teresa" NazwiskoOdbiorcy="Jadowska" />  
  </Klient>  
  <Klient NumerKlienta="2" Nazwisko="Pludowski">  
    <Zamówienia DataZamówienia="1998-10-30T00:00:00" ImięOdbiorcy="Katarzyna" NazwiskoOdbiorcy="Adamczyk" />  
  </Klient>  
  <Klient NumerKlienta="3" Nazwisko="Karbowski">  
    <Zamówienia DataZamówienia="1998-10-13T00:00:00" ImięOdbiorcy="Jan" NazwiskoOdbiorcy="Piasecki" />  
  </Klient>  
  <Klient NumerKlienta="4" Nazwisko="Sitecki">  
    <Zamówienia DataZamówienia="1998-05-16T00:00:00" ImięOdbiorcy="Waldemar" NazwiskoOdbiorcy="Jurecki" />  
  </Klient>  
</Klienci>
```

# Klauzula FOR XML – elements (baza Cukierki)

- ▶ Klauzula **FOR XML AUTO, ELEMENTS** zwraca wartości zapisane w kolumnach jako elementy

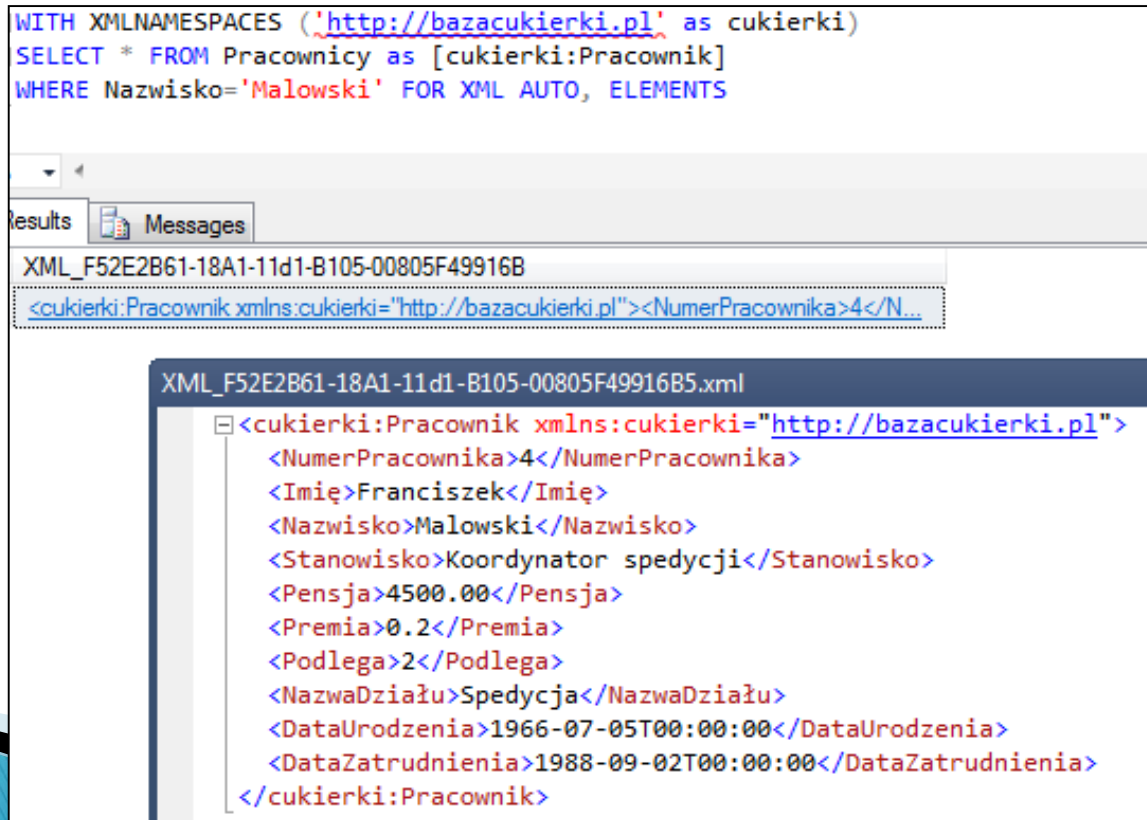
```
select * from Pracownicy for xml auto, elements
```



# Klauzula FOR XML i przestrzeń nazw

- ▶ Klauzula WITH XMLNAMESPACES daje możliwość dodania przestrzeni nazw

```
WITH XMLNAMESPACES ('http://bazacukierki.pl' as cukierki)
SELECT * FROM Pracownicy as [cukierki:Pracownik]
WHERE Nazwisko='Malowski' FOR XML AUTO, ELEMENTS
```



```
WITH XMLNAMESPACES ('http://bazacukierki.pl' as cukierki)
SELECT * FROM Pracownicy as [cukierki:Pracownik]
WHERE Nazwisko='Malowski' FOR XML AUTO, ELEMENTS
```

Results Messages

XML\_F52E2B61-18A1-11d1-B105-00805F49916B

<cukierki:Pracownik xmlns:cukierki="http://bazacukierki.pl"><NumerPracownika>4</N...


XML\_F52E2B61-18A1-11d1-B105-00805F49916B5.xml

```
<cukierki:Pracownik xmlns:cukierki="http://bazacukierki.pl">
  <NumerPracownika>4</NumerPracownika>
  <Imię>Franciszek</Imię>
  <Nazwisko>Malowski</Nazwisko>
  <Stanowisko>Koordynator spedycji</Stanowisko>
  <Pensja>4500.00</Pensja>
  <Premia>0.2</Premia>
  <Podlega>2</Podlega>
  <NazwaDziału>Spedycja</NazwaDziału>
  <DataUrodzenia>1966-07-05T00:00:00</DataUrodzenia>
  <DataZatrudnienia>1988-09-02T00:00:00</DataZatrudnienia>
</cukierki:Pracownik>
```

# Klauzula FOR XML AUTO – generowanie schematu

- ▶ Klauzula FOR XML AUTO, XMLSCHEMA wygeneruje schemat xsd na podstawie danych relacyjnych

```
select * from Pracownicy for xml auto, XMLSCHEMA('Pracownicy')
```



The screenshot displays the SQL Server Enterprise Manager interface. At the top, a query window shows the command: `select * from Pracownicy for xml auto, XMLSCHEMA('Pracownicy')`. Below the query window, the 'XML\_F52E2B61-18A1-11d1-B105-00805F49916B9.xml' file is open, showing the generated XSD schema. The schema is for the namespace 'Pracownicy' and defines an element 'Pracownicy' of complex type. It includes attributes: 'NumerPracownika' (integer), 'Imię' (nvarchar(50)), 'Nazwisko' (nvarchar(50)), 'Stanowisko' (nvarchar(25)), and 'Pensja' (integer). The schema is displayed with a tree view on the left and a detailed view on the right.

```
<xsd:schema targetNamespace="Pracownicy" xmlns:schema="Pracownicy" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:sqltypes="http://schemas.microsoft.com/sqlserver/2004/sqltypes" schemaLocation="http://schemas.microsoft.com/sqlserver/2004/sqltypes/Pracownicy.xsd">
  <xsd:import namespace="http://schemas.microsoft.com/sqlserver/2004/sqltypes" schemaLocation="http://schemas.microsoft.com/sqlserver/2004/sqltypes/Pracownicy.xsd" />
  <xsd:element name="Pracownicy">
    <xsd:complexType>
      <xsd:attribute name="NumerPracownika" type="sqltypes:int" use="required" />
      <xsd:attribute name="Imię">
        <xsd:simpleType>
          <xsd:restriction base="sqltypes:nvarchar" sqltypes:localeId="1033" sqltypes:sqlCompareOptions="IgnoreCase" />
          <xsd:maxLength value="50" />
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:attribute>
    <xsd:attribute name="Nazwisko">
      <xsd:simpleType>
        <xsd:restriction base="sqltypes:nvarchar" sqltypes:localeId="1033" sqltypes:sqlCompareOptions="IgnoreCase" />
        <xsd:maxLength value="50" />
      </xsd:restriction>
    </xsd:simpleType>
    </xsd:attribute>
    <xsd:attribute name="Stanowisko">
      <xsd:simpleType>
        <xsd:restriction base="sqltypes:nvarchar" sqltypes:localeId="1033" sqltypes:sqlCompareOptions="IgnoreCase" />
        <xsd:maxLength value="25" />
      </xsd:restriction>
    </xsd:simpleType>
    </xsd:attribute>
    <xsd:attribute name="Pensja">
```

Klauzula FOR XML **PATH**

# Klauzula FOR XML PATH

- ▶ Tryb PATH klauzuli FOR XML umożliwia podjęcie decyzji, czy dana kolumna z zapytania T-SQL będzie reprezentowana jako **element**, czy **atrybut** w wyjściowym dokumencie XML
- ▶ Nazwy elementów/atrybutów – sterowane aliasami kolumn w zapytaniu T-SQL
- ▶ Można użyć słowa kluczowego **ROOT** aby dodać element główny w XML

## 3 przypadki:

- ▶ **Nazwy elementów/atrybutów** – są pobierane z nazwy kolumn/aliasów zapytania T-SQL
- ▶ **Symbol @** – powoduje wyświetlenie kolumny/aliasu jako **atrybutu** w XML
- ▶ **Znaki /** – dają kontrolę położenia kolumny w wyjściowym XML

# Klauzula FOR XML PATH – przykład 1

SELECT

KodPudełka AS '@Kod',

NazwaPudełka AS 'Nazwa'

FROM Pudełka

FOR XML PATH ('Pudełko'), ROOT ('Pudełka')

The screenshot displays the SQL Server Enterprise Manager interface. On the left, the 'Query Editor' window shows the following SQL query:

```
SELECT KodPudełka AS '@Kod',  
       NazwaPudełka AS 'Nazwa'  
FROM Pudełka  
FOR XML PATH ('Pudełko'), ROOT ('Pudełka')
```

Below the query editor, the 'Results' pane shows the output of the query. The first result is an XML document with the following structure:

```
<Pudełka>  
<Pudełko Kod="ALPY">  
  <Nazwa>Kolekcja alpejska</Nazwa>  
</Pudełko>  
<Pudełko Kod="DELI">  
  <Nazwa>Delicja z masłem orzechowym</Nazwa>  
</Pudełko>  
<Pudełko Kod="FANT">  
  <Nazwa>Fantazja mokka</Nazwa>  
</Pudełko>  
<Pudełko Kod="GORZ">  
  <Nazwa>Słodko-gorzkie</Nazwa>  
</Pudełko>  
</Pudełka>
```

On the right, the 'XML Viewer' pane displays the same XML document in a tree view, showing the hierarchy of the XML data.

# Klauzula FOR XML PATH – przykład 2

SELECT

```
NazwaCukierka AS '@Nazwa',  
KodCukierka AS 'NumerCukierka',  
TypCzekolady AS 'Typ/Czekolada',  
TypNadzienia AS 'Typ/Nadzienie'
```

FROM Cukierki

```
FOR XML PATH('Cukierek') , ROOT ('Produkty')
```

The screenshot displays the SQL Server Enterprise Manager interface. On the left, a query window shows the following SQL code:

```
SELECT  
    NazwaCukierka AS '@Nazwa',  
    KodCukierka AS 'NumerCukierka',  
    TypCzekolady AS 'Typ/Czekolada',  
    TypNadzienia AS 'Typ/Nadzienie'  
FROM Cukierki  
FOR XML PATH('Cukierek') , ROOT ('Produkty')
```

Below the query window, the 'Results' pane shows the XML output for the first row:

```
XML_F52E2B61-18A1-11d1-B105-00805F49916B  
1 <Produkty><Cukierek Nazwa="Marcepanowa jaskółka"...
```

On the right, a separate window displays the full XML document structure:

```
XML_F52E2B61-18A1-11d1-B105-00805F49916B.xml  
<Produkty>  
  <Cukierek Nazwa="Marcepanowa jaskółka">  
    <NumerCukierka>B01</NumerCukierka>  
    <Typ>  
      <Czekolada>Biala</Czekolada>  
      <Nadzienie>Marcepan</Nadzienie>  
    </Typ>  
  </Cukierek>  
  <Cukierek Nazwa="Slodka Lilia">  
    <NumerCukierka>B02</NumerCukierka>  
    <Typ>  
      <Czekolada>Biala</Czekolada>  
      <Nadzienie>Brak</Nadzienie>  
    </Typ>  
  </Cukierek>  
  <Cukierek Nazwa="Zlamane serce">  
    <NumerCukierka>B03</NumerCukierka>  
    <Typ>  
      <Czekolada>Biala</Czekolada>  
      <Nadzienie>Brak</Nadzienie>  
    </Typ>  
  </Cukierek>  
</Produkty>
```

# Klauzula FOR XML PATH – przykład 3

SELECT top(5)

Klient.NumerKlienta as '@NrKlienta',

Klient.Nazwisko,

Zamówienia.NumerZamówienia as 'Zamowienie/NumerZamowienia',

Zamówienia.DataZamówienia as 'Zamowienie/DataZamowienia',

Zamówienia.ImięOdbiorcy as 'Zamowienie/Odbiorca/ImieOdbiorcy',

Zamówienia.NazwiskoOdbiorcy as 'Zamowienie/Odbiorca/NazwiskoOdbiorcy'

FROM Klienci as Klient, Zamówienia

WHERE Klient.NumerKlienta = Zamówienia.NumerKlienta

ORDER BY Klient.NumerKlienta FOR XML PATH('Klient'), ROOT('Klienci')

The screenshot displays a SQL query in the 'Query Editor' window and its resulting XML output in the 'Results' window. The query is a SELECT statement with a TOP(5) clause, joining 'Klienci' and 'Zamówienia' tables. The columns are mapped to XML elements using the FOR XML PATH clause. The XML output is shown in a tree view, with the first result expanded to show the nested structure of the XML. Red arrows indicate the mapping from the query columns to the XML elements: '@NrKlienta' to the 'NrKlienta' attribute, 'Nazwisko' to the 'Nazwisko' element, 'NumerZamówienia' to the 'NumerZamowienia' element, 'DataZamówienia' to the 'DataZamowienia' element, 'ImięOdbiorcy' to the 'ImieOdbiorcy' element, and 'NazwiskoOdbiorcy' to the 'NazwiskoOdbiorcy' element.

```
SELECT top(5)
  Klient.NumerKlienta as '@NrKlienta',
  Klient.Nazwisko,
  Zamówienia.NumerZamówienia as 'Zamowienie/NumerZamowienia',
  Zamówienia.DataZamówienia as 'Zamowienie/DataZamowienia',
  Zamówienia.ImięOdbiorcy as 'Zamowienie/Odbiorca/ImieOdbiorcy',
  Zamówienia.NazwiskoOdbiorcy as 'Zamowienie/Odbiorca/NazwiskoOdbiorcy'
FROM Klienci as Klient, Zamówienia
WHERE Klient.NumerKlienta = Zamówienia.NumerKlienta
ORDER BY Klient.NumerKlienta FOR XML PATH('Klient'), ROOT('Klienci')
```

XML\_F52E2B61-18A1-11d1-B105-00805F49916B15.xml

```
<Klienci>
  <Klient NrKlienta="1">
    <Nazwisko>Jadowska</Nazwisko>
    <Zamowienie>
      <NumerZamowienia>6</NumerZamowienia>
      <DataZamowienia>1998-01-07T00:00:00</DataZamowienia>
      <Odbiorca>
        <ImieOdbiorcy>Julia</ImieOdbiorcy>
        <NazwiskoOdbiorcy>Kania</NazwiskoOdbiorcy>
      </Odbiorca>
    </Zamowienie>
  </Klient>
  <Klient NrKlienta="1">
    <Nazwisko>Jadowska</Nazwisko>
    <Zamowienie>
      <NumerZamowienia>300</NumerZamowienia>
      <DataZamowienia>1998-08-09T00:00:00</DataZamowienia>
      <Odbiorca>
        <ImieOdbiorcy>Teresa</ImieOdbiorcy>
        <NazwiskoOdbiorcy>Jadowska</NazwiskoOdbiorcy>
      </Odbiorca>
    </Zamowienie>
  </Klient>
  <Klient NrKlienta="2">
    <Nazwisko>Pludowski</Nazwisko>
```

# Łączenie 2 dokumentów XML – przykład

- ▶ Nie ma możliwości złączenia 2 oddzielnych dokumentów XML przy użyciu operatora konkatenacji (+)
- ▶ Złączenia można dokonać zapytaniem T-SQL poprzez złączenie 2 zmiennych typu XML

```
DECLARE @xml1 as XML =  
(SELECT Imię, Nazwisko, Miasto from [dbo].[Klienci] WHERE Nazwisko = 'Jadowska'  
FOR XML RAW ('Klient'))
```

```
DECLARE @xml2 as XML =  
(SELECT Imię, Nazwisko, Miasto from [dbo].[Klienci] WHERE Nazwisko = 'Pludowski'  
FOR XML RAW ('Klient'))
```

```
SELECT @xml1, @xml2 FOR XML PATH('')
```

The screenshot displays a SQL Server query window with the following code:

```
DECLARE @xml1 as XML = (SELECT Imię, Nazwisko, Miasto from [dbo].[Klienci] WHERE Nazwisko = 'Jadowska'  
FOR XML RAW ('Klient'))  
  
DECLARE @xml2 as XML = (SELECT Imię, Nazwisko, Miasto from [dbo].[Klienci] WHERE Nazwisko = 'Pludowski'  
FOR XML RAW ('Klient'))  
  
SELECT @xml1, @xml2  
FOR XML PATH('')
```

The Results pane shows the output as an XML document with the filename `XML_F52E2B61-18A1-11d1-B105-00805F49916B29.xml`. The output consists of two XML elements:

```
<Klient Imię="Teresa" Nazwisko="Jadowska" Miasto="Łódź" />  
<Klient Imię="Karol" Nazwisko="Pludowski" Miasto="Warszawa" />
```

The first pane shows the first row of the result set, which is the XML document.

# Łączenie 2 dokumentów XML – przykład c.d.

## ► Poprzedni przykład po dodaniu klauzuli ELEMENTS

```
DECLARE @xml1 as XML = (SELECT Imię, Nazwisko, Miasto from [dbo].[Klienci] WHERE Nazwisko = 'Jadowska'  
FOR XML RAW ('Klient'), ELEMENTS)
```

```
DECLARE @xml2 as XML = (SELECT Imię, Nazwisko, Miasto from [dbo].[Klienci] WHERE Nazwisko = 'Pludowski'  
FOR XML RAW ('Klient'), ELEMENTS)
```

```
SELECT @xml1, @xml2  
FOR XML PATH('')
```

The screenshot displays the SQL Server Enterprise Manager interface. The left pane shows the query editor with the following T-SQL code:

```
DECLARE @xml1 as XML = (SELECT Imię, Nazwisko, Miasto from [dbo].[Klienci] WHERE Nazwisko = 'Jadowska'  
FOR XML RAW ('Klient'), ELEMENTS)  
  
DECLARE @xml2 as XML = (SELECT Imię, Nazwisko, Miasto from [dbo].[Klienci] WHERE Nazwisko = 'Pludowski'  
FOR XML RAW ('Klient'), ELEMENTS)  
  
SELECT @xml1, @xml2  
FOR XML PATH('')
```

The bottom pane shows the 'Results' tab with a single row of XML data:

XML_F52E2B61-18A1-11d1-B105-00805F49916B
<Klient><Imię>Teresa</Imię><Nazwisko>Jadowska</N...

The right pane shows the XML structure for the file 'XML\_F52E2B61-18A1-11d1-B105-00805F49916B30.xml':

```
<Klient>  
  <Imię>Teresa</Imię>  
  <Nazwisko>Jadowska</Nazwisko>  
  <Miasto>Łódź</Miasto>  
</Klient>  
<Klient>  
  <Imię>Karol</Imię>  
  <Nazwisko>Pludowski</Nazwisko>  
  <Miasto>Warszawa</Miasto>  
</Klient>
```

# Łączenie 2 dokumentów XML – przykład c.d.

## ► Poprzedni przykład po dodaniu klauzuli ROOT

```
DECLARE @xml1 as XML = (SELECT Imię, Nazwisko, Miasto from [dbo].[Klienci] WHERE Nazwisko = 'Jadowska'  
FOR XML RAW ('Klient'), ELEMENTS)
```

```
DECLARE @xml2 as XML = (SELECT Imię, Nazwisko, Miasto from [dbo].[Klienci] WHERE Nazwisko = 'Pludowski'  
FOR XML RAW ('Klient'), ELEMENTS)
```

```
SELECT @xml1, @xml2  
FOR XML PATH(''), ROOT('Klienci')
```

The screenshot displays the SQL Server Enterprise Manager interface. The top pane shows the execution of the following SQL query:

```
DECLARE @xml1 as XML = (SELECT Imię, Nazwisko, Miasto from [dbo].[Klienci] WHERE Nazwisko = 'Jadowska'  
FOR XML RAW ('Klient'), ELEMENTS)  
  
DECLARE @xml2 as XML = (SELECT Imię, Nazwisko, Miasto from [dbo].[Klienci] WHERE Nazwisko = 'Pludowski'  
FOR XML RAW ('Klient'), ELEMENTS)  
  
SELECT @xml1, @xml2  
FOR XML PATH(''), ROOT('Klienci')
```

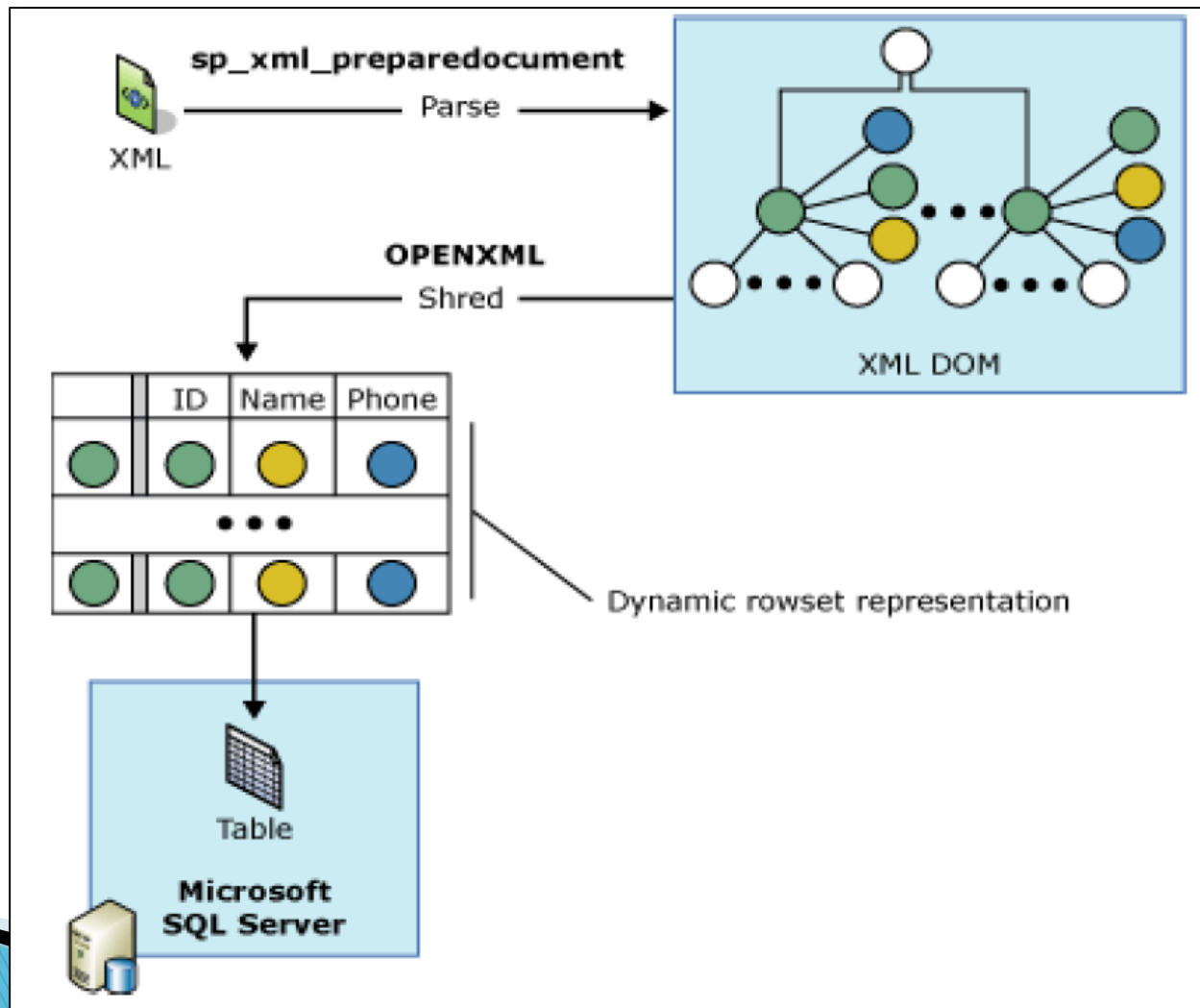
The bottom pane shows the results of the query. The first result is an XML document with the following structure:

```
<Klienci>  
  <Klient>  
    <Imię>Teresa</Imię>  
    <Nazwisko>Jadowska</Nazwisko>  
    <Miasto>Łódź</Miasto>  
  </Klient>  
  <Klient>  
    <Imię>Karol</Imię>  
    <Nazwisko>Pludowski</Nazwisko>  
    <Miasto>Warszawa</Miasto>  
  </Klient>  
</Klienci>
```

# OPENXML

# Działanie OPENXML

OPENXML jest wykorzystywane do przetransferowania danych z dokumentu XML do postaci tabeli



# Wczytywanie danych z XML do tabeli

**OPENXML(iDoc, RowPattern, [Flags],  
[WITH (SchemaDeclaration | TableName)])**

- **iDoc** – uchwyt do reprezentacji dokumentu w pamięci
- **RowPattern** – ścieżka XPath wskazująca jaka część dokumentu ma być przetwarzana
- **Flags**
  - 0 – odwzorowanie atrybutów – wartość domyślna
  - 1 – odwzorowanie atrybutów (może być łączona z 2)
  - 2 – odwzorowanie elementów
  - 3 – odwzorowanie atrybutów lub elementów
- **WITH** – docelowe miejsce składowania danych

# OPENXML – przykład

**sp\_xml\_preparedocument** – tworzy w pamięci reprezentację węzłów

**sp\_xml\_removedocument** – usunięcia dokument z pamięci

```
DECLARE @doc xml--nvarchar(1000)
```

```
SET @doc = '
```

```
<Zamówienie ID = "1011">
```

```
  <Pozycja ProduktID="1" Ilosc="2"/>
```

```
  <Pozycja ProduktID="2" Ilosc="1"/>
```

```
</Zamówienie>'
```

```
DECLARE @xmlDoc integer -- uchwyt w pamięci, pierwszy wolny ID (integer) w sesji, nadawany automatycznie
```

```
EXEC sp_xml_preparedocument @xmlDoc OUTPUT, @doc
```

```
--select @xmlDoc
```

```
SELECT * FROM OPENXML (@xmlDoc, 'Zamówienie/Pozycja', 1)
```

```
WITH
```

```
(Zamówienie integer '@ID', --w miejsce "Zamówienie" można wstawić konstrukcję: ID integer '@ID')
```

```
ProduktID integer,
```

```
Ilosc integer)
```

```
EXEC sp_xml_removedocument @xmlDoc
```

**OPENXML(iDoc, RowPattern, [Flags],  
[WITH (SchemaDeclaration | TableName)])**

## Wynik

	Zamówienie	ProduktID	Ilość
1	1011	1	2
2	1011	2	1

# OPENXML – przykład 2 – XML z przestrzenią nazw

W przypadku, gdy dokument XML ma przypisaną przestrzeń nazw, **sp\_xml\_preparedocument** musi mieć podany 3-ci parametr z określoną przestrzenią nazw

**OPENXML(iDoc, RowPattern, [Flags],  
[WITH (SchemaDeclaration | TableName)])**

```
DECLARE @doc nvarchar(1000)
SET @doc = '
    <Firma:Zamówienie ID = "1011" xmlns:Firma="http://firma.pl">
        <Pozycja ProduktID="1" Ilosc="2"/>
        <Pozycja ProduktID="2" Ilosc="1"/>
    </Firma:Zamówienie>'
DECLARE @xmlDoc integer

EXEC sp_xml_preparedocument @xmlDoc OUTPUT, @doc, '<root xmlns:Firma="http://firma.pl"/>'

SELECT * FROM
OPENXML (@xmlDoc, 'Firma:Zamówienie/Pozycja', 1)
WITH
    (Zamówienie integer '@ID',
    ProduktID integer,
    Ilosc integer)

EXEC sp_xml_removedocument @xmlDoc
```

## Wynik

	Zamówienie	ProduktID	Ilość
1	1011	1	2
2	1011	2	1

# OPENXML – przykład 2

```
CREATE TABLE MoiKlienci (KodKlienta varchar(20) primary key, Nazwisko varchar(20), Firma varchar(20))
CREATE TABLE MojeZamówienia (KodKlienta varchar(20), DataZamówienia datetime)
```

```
DECLARE @docHandle int
DECLARE @xmlDocument nvarchar(max) -- lub typ xml
```

```
SET @xmlDocument = N'
<Operacje>
  <Klient KodKlienta="XYZAA" Nazwisko="Marski" Firma="Spółka ZOO">
    <Zamówienie KodKlienta="XYZAA" DataZamówienia="2012-08-25T00:00:00"/>
    <Zamówienie KodKlienta="XYZAA" DataZamówienia="2012-10-03T00:00:00"/>
  </Klient>
  <Klient KodKlienta="XYZBB" Nazwisko="Staniewicz" Firma="FX Construction">
    Brak zamówień!
  </Klient>
</Operacje>'
```

```
EXEC sp_xml_preparedocument @docHandle OUTPUT, @xmlDocument
```

```
INSERT MoiKlienci
SELECT * FROM OPENXML(@docHandle, N'/Operacje/Klient') WITH MoiKlienci
```

```
INSERT MojeZamówienia
SELECT * FROM OPENXML(@docHandle, N'//Zamówienie') WITH MojeZamówienia
```

```
EXEC sp_xml_removedocument @docHandle
```

```
Select * from MoiKlienci
Select * from MojeZamówienia
```

	KodKlienta	Nazwisko	Firma
1	XYZAA	Marski	Spółka ZOO
2	XYZBB	Staniewicz	FX Construction

	KodKlienta	DataZamówienia
1	XYZAA	2012-08-25 00:00:00.000
2	XYZAA	2012-10-03 00:00:00.000

# Indeksy XML

# Rodzaje indeksów XML

- ▶ Podstawowy indeks XML (ang. Primary XML Index)
- ▶ Pomocniczy indeks XML (ang. Secondary XML Index)
  - Indeks pomocniczy typu PATH
  - Indeks pomocniczy typu VALUE
  - Indeks pomocniczy typu PROPERTY

# Rodzaje indeksów XML

## ► Podstawowy indeks XML (ang. Primary XML Index)

- Musi być założony jako pierwszy spośród wszystkich indeksów XML
- Wymaga założenia klucza głównego na tabeli

```
CREATE PRIMARY XML INDEX NazwaIndeksu On Tabela(kolumna)
```

## ► Pomocniczy indeks XML (ang. Secondary XML Index)

- Indeks pomocniczy typu PATH

- Może poprawić wydajność zapytań wykorzystujących ścieżki Xpath

```
CREATE XML INDEX NazwaIndeksu On Tabela(kolumna)
```

```
USING XML INDEX NazwaIndeksuXML_Primary
```

```
FOR PATH
```

- Indeks pomocniczy typu VALUE

- Może poprawić wydajność wyszukiwania po wartościach węzłów (składnia jak w przyp. PATH – FOR VALUE)

- Indeks pomocniczy typu PROPERTY

- Może poprawić wydajność wyszukiwania wartości z dokumentów XML (składnia jak w przyp. PATH – FOR PROPERTY)

**Dziękuję**