

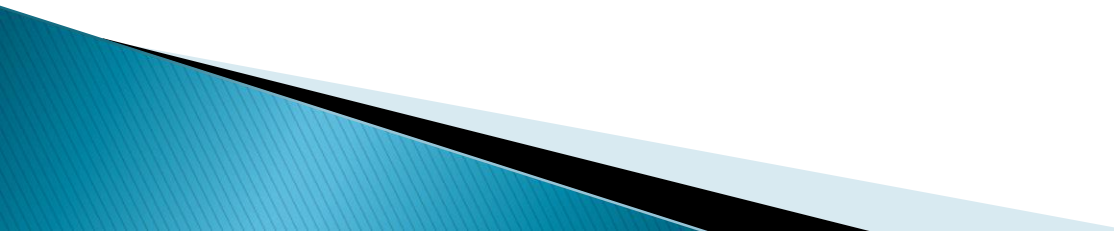


# Język XML

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

# XML – wstęp

# XML – definicja

- ▶ XML – Extensible Markup Language
  - ▶ Zdefiniowany w 1996 roku przez World Wide Web Consortium
  - ▶ Język znacznikowy (jak HTML), może posiadać nieskończoną ilość znaczników
  - ▶ Pozwala definiować strukturę znaczników dla dowolnych dziedzin zastosowań
- 

# XML – zastosowanie

- ▶ Zarządzanie dokumentami, treścią, wiedzą (dokument tekstowy)
- ▶ Elektroniczna wymiana danych, integracja aplikacji

# XML – zalety

- ▶ XML – format samoopisujący się, bardzo dobrze udokumentowany
- ▶ XML jest standardem dostępnym, **nie jest objęty prawami autorskimi**, patentowymi, tajemnicą handlową
- ▶ XML jest przydatny do tworzenia dużych i złożonych dokumentów, gdyż **dane w nim są ustrukturyzowane**;
- ▶ XML zawiera mechanizm pozwalający integrować ze sobą dane z różnych źródeł i wyświetlać je jako pojedynczy dokument;
  - W jednym pliku xml można doskonale zapisać dane hierarchiczne oraz dane relacyjne pobrane z wielu tabel
- ▶ Poszczególne części dokumentu można pokazywać lub ukrywać w zależności od akcji podejmowanych przez użytkownika.

# Aplikacje (dialekty) xml

- ▶ XML możemy nazwać metajęzykiem, na jego bazie zdefiniowano wyspecjalizowane dialekty określając zestaw znaczników (na ogół przy pomocy Schematów XML)

Przykłady:

- ▶ SVG (ang. Scalable Vector Graphics),
- ▶ SMIL (Synchronized Multimedia Integration Language),
- ▶ SOAP (Simple Object Access Protocol),
- ▶ OFX (Open Financial eXchange)

Inne:

- ▶ <https://www.oasis-open.org/standards>
- ▶ <http://www.w3.org/standards/>

# Narzędzia do tworzenia dokumentów XML

Jedne z najpopularniejszych narzędzi do tworzenia i edycji dokumentów XML:

- ▶ Dowolny edytor tekstowy np. Notatnik w MS Windows
- ▶ Edytor rozpoznający składnię np. **Notepad++**
- ▶ Edytor strukturalny wyświetlający XML w postaci drzewa – np. **XMLNotepad**, XMLFox;
- ▶ Edytory środowisk programistycznych, które na ogół rozpoznają składnię XML np. **Visual Studio**
- ▶ Zaawansowane komercyjne oprogramowanie np. Altova XmlSpy

# XML i unicode

- ▶ XML w pełni obsługuje dwubajtowy zestaw znaków Unicode, a także jego bardziej zwarte postaci
- ▶ Unicode pozwala obsłużyć niemalże wszystkie współcześnie używane języki.
- ▶ Dokument XML używa znaczników XML, a domyślnym zestawem znaków jest Unicode (UTF-8, UTF-16)
- ▶ W przypadku SQL Server ograniczenie na wielkość pliku importowanego do kolumny xml w tabeli (w konkretnym rekordzie tabeli) to 2GB

# Dokumenty XML

# Podstawowe elementy dokumentu XML

The diagram illustrates the structure of an XML document with the following components and their corresponding labels:

- `<?xml version="1.0"?>` is labeled as **Deklaracja XML**.
- `<sprawozdanie nr="1313/2015">` is labeled as **Element główny**.
- `nr="1313/2015"` is labeled as **Atrybut**.
- `<autor>Waldemar Karwowski</autor>` is labeled as **Element**.
- `<treść>W dniu` is labeled as **Znacznik początkowy**.
- `<data>17.10.2015</data>` is labeled as **Znacznik końcowy**.
- `<godzina>12:00</godzina>` is labeled as **Zawartość tekstowa**.
- `<temat>XML</temat>` is labeled as **Zawartość tekstowa**.
- `</treść>` is labeled as **Znacznik końcowy**.
- `</sprawozdanie>` is labeled as **Znacznik początkowy**.

```
<?xml version="1.0"?>
<sprawozdanie nr="1313/2015">
  <autor>Waldemar Karwowski</autor>
  <miejsce>SGGW</miejsce>
  <treść>W dniu
    <data>17.10.2015</data>
    o godzinie <godzina>12:00</godzina>
    rozpocząłem wykład na temat
      <temat>XML</temat>
    dla studentów PSDB.
  </treść>
</sprawozdanie>
```

# Zawartość dokumentu XML

## Elementy:

- ▶ wyznaczane za pomocą znaczników (znacznik otwierający i zamykający);
- ▶ ograniczenia na nazwę:
  - może zawierać większość znaków,
  - choć nie może zaczynać się od cyfry, myślnika ani kropki;
  - dopuszczalne na początku są podkreślenia i dwukropki;
- ▶ dla nazw elementów **rozróżnialna jest wielkość liter**;
- ▶ element **niepusty** składa się obowiązkowo ze znacznika otwierającego i zamykającego:

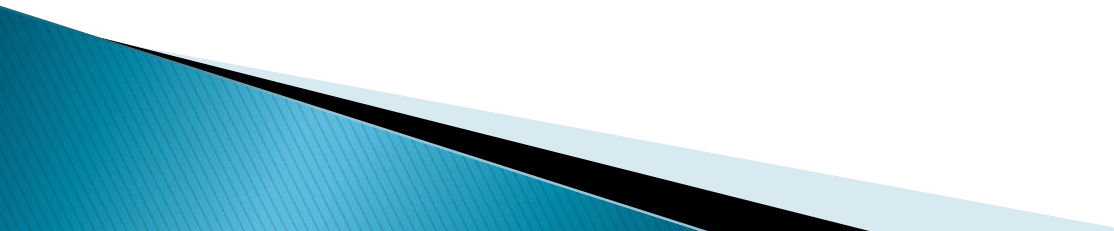
**<osoba>Jan Kowalski</osoba>**

- ▶ element **pusty** może składać się z jednego tylko znacznika, zakończonego znakiem „/”:

**<usunietyWpis />**

# Klasyfikacja elementów XML

## Wyróżniamy:

- ▶ elementy puste (pojedynczy znacznik zakończony „/”);
  - ▶ elementy zawierające tylko tekst;
  - ▶ elementy złożone, tj. posiadające podelementy (parent-child);
  - ▶ elementy o zawartości mieszanej (tj. zarówno tekst jak i podelementy – mogą być na rozmaite sposoby przeplecione);
- 

# Hierarchia elementów XML

- ▶ **Element główny (root)** – zawiera wszystkie pozostałe elementy
- ▶ **Kolejność zawieranych podelementów** często jest istotna dla właściwej interpretacji dokumentu, jest najczęściej określona definicją dokumentu (np. **poprzez schemat XML**);
- ▶ **Elementy złożone** (i mieszane) muszą być poprawnie zagnieżdżone. W efekcie prawidłowy dokument XML posiada **strukturę hierarchiczną**.

# Atrybuty XML

- ▶ Każdy element może zawierać wiele atrybutów
- ▶ Przybierają postać  
**nazwaAtrybutu = "wartość atrybutu"**
  - Wartości atrybutów muszą być ujęte w cudzysłowach lub w apostrofach
- ▶ Ograniczenia na nazwę – podobne, jak w przypadku nazw elementów;  
nie może zaczynać się od cyfry, myślnika ani kropki;
- ▶ **Zawarte w znaczniku otwierającym** (albo w znaczniku elementu pustego);
- ▶ Znacznik może zawierać tylko jeden atrybut o danej nazwie;
- ▶ Rozróżniana jest wielkość liter;
- ▶ Kolejność atrybutów nie jest wymagana standardem, choć zaleca się jej konsekwentne utrzymanie;

# Atrybuty XML – przykład

## Prawidłowy jest zapis

```
<pracownik dane = "Adam Kowalski">  
  <obowiązek funkcja= "prelegent"/>  
  <obowiązek funkcja= "promotor"/>  
</pracownik>
```

Czy poniższy zapis jest prawidłowy? Jeśli tak, to który z nich?

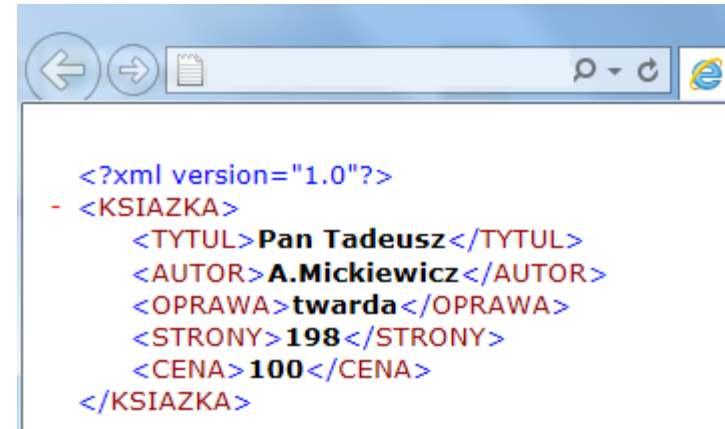
```
<ksiazka autor="Jan Kowalski" autor="Adam Nowak"> Opowieści </ksiazka>  
<ksiazka autor="Jan Kowalski" Autor="Adam Nowak"> Opowieści </ksiazka>  
<ksiazka autor="Jan Kowalski" Autor='Adam Nowak'> Opowiesci </ksiazka>
```

# Atrybuty vs elementy

## Kiedy stosować?

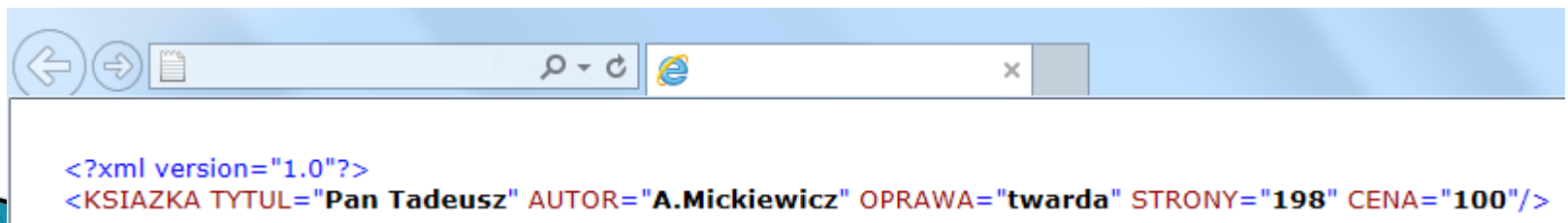
### Przykład bez atrybutów:

```
<KSIAZKA>
  <TYTUL>Pan Tadeusz</TYTUL>
  <AUTOR>A.Mickiewicz</AUTOR>
  <OPRAWA>twarda</OPRAWA>
  <STRONY>198</STRONY>
  <CENA>100</CENA>
</KSIAZKA>
```



### Przykład z wykorzystaniem atrybutów:

```
<KSIAZKA TYTUL="Pan Tadeusz" AUTOR="A.Mickiewicz" OPRAWA="twarda"
STRONY="198" CENA="100"></KSIAZKA>
```



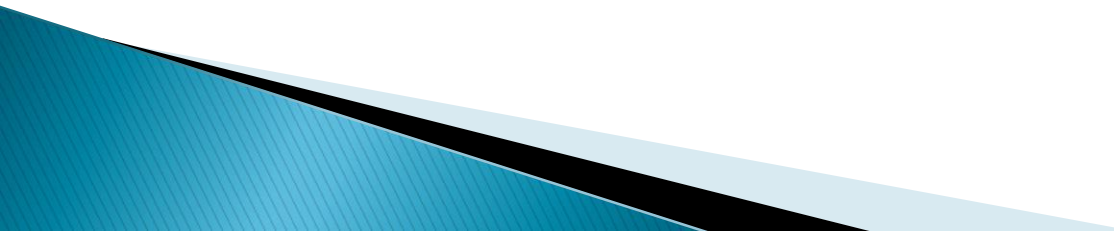
# Zawartość dokumentu XML

Który z fragmentów XML jest prawidłowy?

1. `<root><class>XML</class></root>`
2. `<class><root>XML</root></class>`
3. `<root><class id="XML"> </root>`
4. `<root>XML<class id="XML"/>XML</root>`
5. `<root class="XML"><class id="root"/>XML</root>`

# Zawartość dokumentu XML

Który z fragmentów XML jest prawidłowy?

1. `<name first='Tony' LAST="Romeo"/>`
  2. `<name name="Tony" NAME="ROMEO"/>`
  3. `<_name_ first-name="Tony" last-name="Romeo"/>`
  4. `<-name_ first-name="Tony" last-name="Romeo"/>`
  5. `<name="Tony Romeo" />`
  6. `<name name="first='Tony' last='Romeo'"/>`
- 

# XML – znaki specjalne

- ▶ Z omówionej składni widać, że znaki specjalne to:  
„<”, „>” oraz cudzysłów, jak również apostrof
- ▶ Chcąc użyć ich jako zwykłych znaków w miejscach, gdzie byłoby to niejednoznaczne, należy użyć predefiniowanych symboli zastępczych:  
&gt; , &lt; oraz &quot; i &apos;
- ▶ Wobec tego także zamiast & używamy &amp;

# XML – znaki specjalne

- ▶ Symbole zastępcze rozwiązują problem, choć oczywiście mogą być niewygodne, np. przy formułowaniu w dokumentach XML warunków, jak to ma miejsce np. w językach zapytań.
- ▶ Znaki specjalne mogą być potrzebne w wartościach atrybutów, jednakże można ich uniknąć stosując jako ograniczniki wartości tekstowej symbole apostrofu i cudzysłowu zamiennie.

```
<książka autor="&apos;Jan Kowalski&apos;" Autor="Adam Nowak">  
&apos;Opowieści&apos;  
</książka>
```

## Uwaga:

- ▶ Znaki Unicode wstawiamy &#n; gdzie n to numer znaku  
na przykład: &#252;

# XML – znaki specjalne

Przykładowa strona z listą znaków specjalnych:

<https://unicode-table.com/en/#0414>

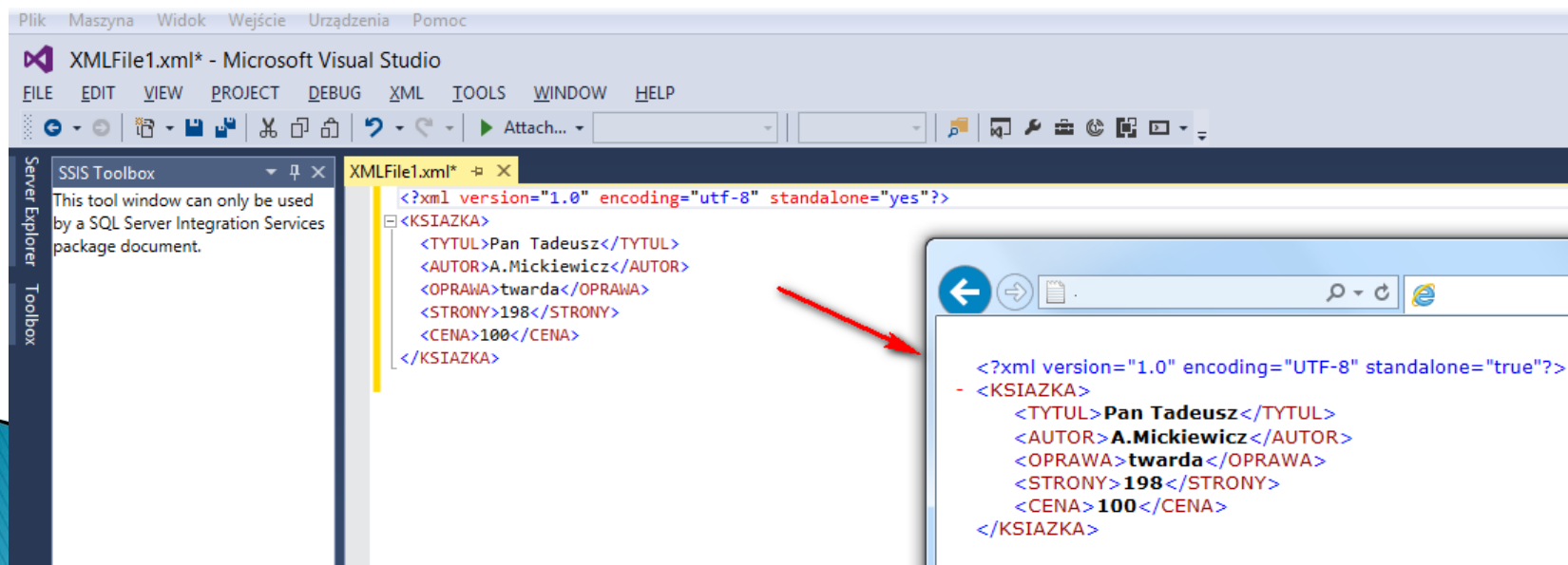
The screenshot displays the Unicode Table website for the Cyrillic Capital Letter De (U+0414). The browser address bar shows the URL <https://unicode-table.com/en/#0414>. The page features a grid of Cyrillic characters. A red arrow points from the URL to the character 'Д' in the grid. Another red arrow points from the character 'Д' to a pop-up window that displays the character 'Д' and its details. The pop-up window includes the text 'Cyrillic Capital Letter De', 'Unicode number: U+0414', and 'HTML-code: &#1044;'. The HTML code is circled in red. On the right side of the page, there is a sidebar with the title 'Cyrillic', a link 'Open in separate page', a range 'Range: 0400— 04FF', a button 'Click to highlight range', and a list of languages: 'Languages: russian, ukrainian, bulgarian'. A world map is also visible in the sidebar.

# Deklaracja XML

- ▶ Pierwszy składnik – tzw. prolog dokumentu.
- ▶ Określa sposób interpretacji dokumentu.
- ▶ Specyfikuje:
  - wersję użytego języka: **parametr version**;
  - rodzaj kodowania znaków: **parametr encoding** (np. unicode, ISO, windows);
  - określenie, czy dokument jest samodzielny, czy zależy od zewnętrznego DTD (parametr standalone).

*DTD – document type definition – starsza wersja XSD, ma kilka ograniczeń*

Przykład xsd:



# Deklaracje w XML – c.d.

- ▶ Umieszczane w symbolach: `<! ... >`
- ▶ Deklaracje *podstawowe*, obecne w zwykłych dokumentach XML to:
  - **komentarze:** `<!-- ... -->`
  - **sekcje danych znakowych:** wyznaczają obszar, w którym można swobodnie używać znaków zastrzeżonych: `<![CDATA[Tutaj może się znaleźć dość dowolny tekst]]>`

# XML – przykład komentarzy i sekcji znakowych

```
<?xml version="1.0" encoding="utf-8" ?>
```

```
<!-- można umieszczać komentarze -->
```

```
<książka>
```

```
  <rozdział nr="1">
```

```
    <tytuł>Wstęp</tytuł>
```

```
    <!-- to też jest dodatkowy komentarz -->
```

```
    <strona>tekst</strona>
```

```
  </rozdział>
```

```
  <rozdział nr="2">
```

```
    <tytuł>Zasady XML</tytuł>
```

```
    <strona>
```

```
      <![CDATA[ <p>paragraf</p>
```

```
        <br/> nowa linia
```

```
        <hr/>
```

```
      ]]>
```

```
    </strona>
```

```
  </rozdział>
```

```
</książka >
```

# Instrukcje przetwarzania XML

- ▶ Instrukcje przetwarzania = *Processing Instructions* – *PI*;
- ▶ umieszczane pomiędzy znakami: `<? ... ?>`
- ▶ Określają **cel** (*PI Target*) oraz **instrukcje**, które mają być wykonane.
- ▶ Cel jest słowem kluczowym (umieszczanym bezpośrednio po otwarciu znacznika), które identyfikuje aplikację mającą dokonać przetwarzania.
- ▶ Preferowany styl oznaczania kodu PHP tworzy właśnie instrukcję przetwarzania zgodną z notacją XML:

`<?php tutaj kod PHP... ?>`

# Przestrzenie nazw XML

# Deklaracja przestrzeni nazw

<**namespacePrefix**:nazwaElementu xmlns:**namespacePrefix** = 'URL'>

- ▶ **Nazwa elementu** lub atrybutu może zawierać co najwyżej jeden dwukropek
- ▶ Nazwę pełną **namespacePrefix:nazwaElementu** nazywamy nazwą kwalifikowaną
- ▶ **URL nie musi istnieć** – ma być unikalny. Zaleca się użycie adresu URL gdzie znajduje się opis struktury, którą chcemy użyć

# NamespacePrefix

<namespacePrefix:nazwaElementu xmlns:namespacePrefix = 'URL'>

- ▶ Określa jednoznacznie daną przestrzeń nazw
- ▶ Musi spełniać ogólne wymogi nazewnictwa w XML
- ▶ Nie może być: xml ani xmlns
- ▶ xmlns: słowo kluczowe

# Użycie przestrzeni nazw

- ▶ Definiowany **namespacePrefix:nazwaElementu** – najczęściej jest to element główny dokumentu.
- ▶ Do elementów odwołujemy się  
**<namespacePrefix:nazwa\_elementu>**

Przykładowo dla zdefiniowanych przestrzeni:

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/XSL/Transform/1.0">  
<xsd:schema xmlns:xsd="http://www.w3.org/1999/XMLSchema">
```

odwołujemy się

- ▶ **<xsl:template>**, **<xsl:apply-templates>**, **<xsl:value-of>**
- ▶ **<xsd:schema>**, **<xsd:element>**, **<xsd:complexType>**

# Użycie przestrzeni nazw – przykład

```
<?xml version="1.0"?>  
<rachunek:komunikat xmlns:rachunek='http://www.bank.com/rachunek'>  
  <rachunek:data>1/1/2014</rachunek:data>  
  <rachunek:treść>Stan konta 1999 PLN</rachunek:treść>  
</rachunek:komunikat>
```

# Wiele przestrzeni nazw w dokumencie

- ▶ Czasem w dokumencie deklarujemy wiele przestrzeni nazw. Można tego dokonać dodając do dokumentu inną deklarację przestrzeni nazw, a następnie w odpowiednich miejscach dodać **elementy kwalifikowane**.

```
<?xml version="1.0"?>
```

```
<rachunek:komunikat
```

```
  xmlns:rachunek='http://www.bank.com/rachunek'>
```

```
  xmlns:treść='http://www.bank.com/tresc'>
```

```
    <rachunek:data>1/1/2014</rachunek:data>
```

```
    <tresc:treść>Stan konta 1999 PLN</tresc:treść>
```

```
</rachunek:komunikat>
```



# Użycie wielu przestrzeni nazw

Przestrzeń nazw może być definiowana na 2 sposoby:

Bezpośrednio w elementach, w których są one wykorzystywane:	W głównym elemencie xml (root)
<pre>&lt;root&gt; &lt;o:table xmlns:o="http://www.w3.org/TR/html4/"&gt;   &lt;o:tr&gt;     &lt;o:td&gt;Jablka&lt;/o:td&gt;     &lt;o:td&gt;Banany&lt;/o:td&gt;   &lt;/o:tr&gt; &lt;/o:table&gt;  &lt;m:table xmlns:m="http://www.uazz.pl/meble"&gt;   &lt;m:name&gt; Afrykanski stolik &lt;/m:name&gt;   &lt;m:width&gt;80&lt;/m:width&gt;   &lt;m:length&gt;120&lt;/m:length&gt; &lt;/m:table&gt; &lt;/root&gt;</pre>	<pre>&lt;root xmlns:o="http://www.w3.org/TR/html4/"       xmlns:m="http://www.uazz.pl/meble"&gt;   &lt;o:table&gt;     &lt;o:tr&gt;       &lt;o:td&gt;Jablka&lt;/o:td&gt;       &lt;o:td&gt;Banany&lt;/o:td&gt;     &lt;/o:tr&gt;   &lt;/o:table&gt;    &lt;m:table&gt;     &lt;m:name&gt; Afrykanski stolik &lt;/m:name&gt;     &lt;m:width&gt;80&lt;/m:width&gt;     &lt;m:length&gt;120&lt;/m:length&gt;   &lt;/m:table&gt; &lt;/root&gt;</pre>

# Domyślna przestrzeń nazw (przestrzeń „xmlns” bez prefixu)

<nazwaElementu **xmlns**='URL'>

- ▶ Dotyczy elementów bez NamespacePrefix
- ▶ Dotyczy wszystkich elementów potomnych bez NamespacePrefix
- ▶ xmlns="" usuwa domyślną przestrzeń nazw
- ▶ Dotyczy elementów, nie dotyczy atrybutów!

<?xml version="1.0"?>

<**wiadomość** **xmlns**='http://www.bank.com/rachunek'  
xmlns:treść='http://www.bank.com/tresc'>

<**data**>1/1/2014</data>

<treść:treść>Stan konta 1999 PLN</treść:treść>

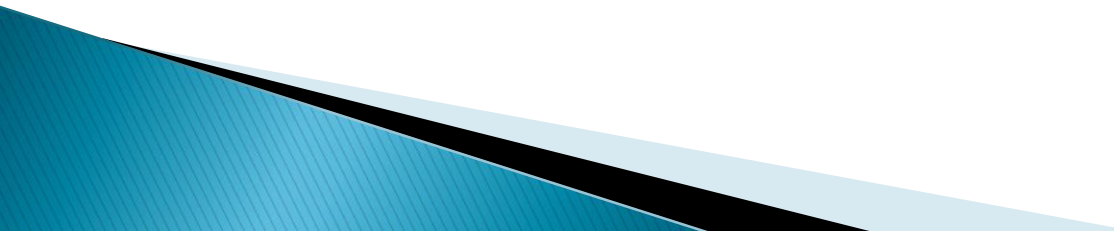
</wiadomość>

# Poprawność budowy XML (well-formed)

Warunkiem koniecznym dla przetwarzania dokumentu XML przez oprogramowanie jest jego poprawna budowa tzn.:

- ▶ Dokument zawiera deklarację XML;
- ▶ Element główny („korzeń”) **występuje raz** w każdym z dokumentów i nie jest zawarty w żadnym innym elemencie.
- ▶ Pozostałe elementy muszą być poprawnie zagnieżdżone. W oparciu o takie zagnieżdżenie mówi się o **relacji rodzic-dziecko**.
- ▶ Poprawne są atrybuty;
- ▶ Spełnione są wszystkie ograniczenia poprawnościowe ujęte w specyfikacji;
- ▶ Wszystkie parsowalne encje do których odwołuje się dokument spełniają warunek poprawności;

# Ważność dokumentu (valid)

- ▶ Dokument jest ważny (poprawny strukturalnie) gdy jest zgodny z definicją schematu XML Schema.
  - ▶ Ważność dokumentu oznacza, że jest on poprawnie zbudowany oraz **posiada deklarację typu dokumentu w prologu** dokumentu zawierającą definicję typu dokumentu lub XML Schema.
  - ▶ Pozostała część dokumentu jest zgodna ze strukturą zdefiniowaną w DTD lub XML Schema.
- 

# Schematy XML

# XML Schema – wg W3C

- ▶ Część wstępna

<http://www.w3.org/TR/xmlschema-0/>

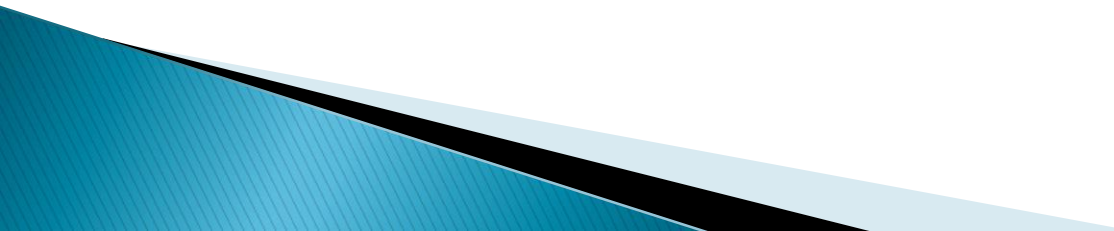
- ▶ Część druga – structures second edition

<http://www.w3.org/TR/xmlschema-1/>

- ▶ Część trzecia – datatypes

<http://www.w3.org/TR/xmlschema-2/>

# Cechy XML Schema

- ▶ Deklarowanie typów danych
    - możliwość deklaracji typów podstawowych
    - możliwość definiowania własnych typów
  - ▶ Możliwość definiowania zbiorów elementów, w których kolejność jest dowolna
  - ▶ Możliwość deklarowania wielu elementów o takiej samej nazwie, ale innym położeniu w dokumencie i innej budowie
  - ▶ Mechanizmy pozwalające na rozszerzanie i uszczegóławianie struktury dokumentów;
- 

# Cechy XML Schema – c.d.

- ▶ Możliwość jednokrotnego definiowania powtarzających się fragmentów struktury (własne typy danych – np. i ich wielokrotnego wykorzystywania w dalszej części schematu

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" >

  <xs:simpleType name="typTelefon">
    <xs:restriction base="xs:long">
      <xs:maxInclusive value="999999999"/>
      <xs:minInclusive value="100000000"/>
    </xs:restriction>
  </xs:simpleType>

  <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="typTelefon"/>
        <xs:element name="data_urodzenia" type="xs:date"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

# Przykładowy dokumenty XML Schema

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="auto">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="fabryka" type="xsd:string"/>
        <xsd:element name="kolor" type="xsd:string"/>
        <xsd:element name="rok" type="xsd:string"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

**Dokument "auto.xsd"**

# Podpięcie schematu xsd do pliku xml

położenie naszej struktury XMLSchema

## Dokument XML

```
<?xml version="1.0"?>
<auto  xsd:noNamespaceSchemaLocation="auto.xsd"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema-instance">
  <fabryka>ford</fabryka >
  <kolor>czerwony</kolor>
  <rok>2014</rok>
</auto>
```

Odwołanie do deklaracji  
XMLSchema

# Deklaracja elementów

- ▶ Deklaracja bez nazwania typu (typ wykorzystany jednokrotnie):

```
<xsd:element name="auto">
  <xsd:complexType>
    ...
  </xsd:complexType>
</xsd:element>
```

- ▶ Deklaracja z nazwaniem typu (typ wykorzystany wielokrotnie):

```
<xsd:element name="auto" type="typAuto"/>

<xsd:complexType name="typAuto">
  ...
</xsd:complexType>
```

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" >

  <xs:simpleType name="typTelefon">
    <xs:restriction base="xs:long">
      <xs:maxInclusive value="999999999"/>
      <xs:minInclusive value="100000000"/>
    </xs:restriction>
  </xs:simpleType>

  <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="typTelefon"/>
        <xs:element name="data_urodzenia" type="xs:date"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

# Deklaracja elementów – c.d.

- ▶ Elementy można deklarować
  - ▶ globalnie – wewnątrz elementu `<xsd:schema/>`
  - ▶ lokalnie – wewnątrz deklaracji typu – `<xsd:complexType/>`
- ▶ Odwoływanie się do elementów zdefiniowanych wcześniej:  
`<xsd:element name="auto" type="xsd:string"/>`

```
<xsd:complexType>
```

```
...
```

```
<xsd:element ref="auto"/>
```

```
...
```

```
</xsd:complexType>
```

# Liczba wystąpień elementów

## Kontrolowanie liczby wystąpień:

- ▶ Deklaracja:
  - minOccurs= "n" – minimalna liczba wystąpień (domyślnie 1);
  - maxOccurs= "n" – maksymalna liczba wystąpień (domyślnie 1),
  - unbounded – dowolna liczba;
- ▶ Deklarować można jedynie w typach lokalnych;

## Przykład:

```
<xsd:element name="auto" type="xsd:string" minOccurs="1"/>  
<xsd:element name="auto" type="xsd:string" maxOccurs="10"/>
```

# Liczba wystąpień elementów – przykład

```
<xs:complexType name="OsobaTyp">
  <xs:sequence>
    <xs:element ref="imie" minOccurs="1" maxOccurs="unbounded"/>
    <xs:element ref="nazwisko"/>
  </xs:sequence>
  <xs:attribute ref="plec" use="required"/>
  <xs:attribute ref="email" use="optional"/>
</xs:complexType>

<xs:simpleType name="PlecTyp">
  <xs:restriction base="xs:string">
    <xs:enumeration value="k"/>
    <xs:enumeration value="m"/>
  </xs:restriction>
</xs:simpleType>
```

# Typy proste

- ▶ Dwa rodzaje typów danych w XML:
  - **Proste** – *element typu prostego przechowuje jeden rodzaj danych*
  - **Złożone** – *przechowuje wiele rodzajów danych (elementów podrzędnych)*

## Typy proste:

- Typy wbudowane
- Typy definiowane. Nie zawierają elementów i atrybutów

```
<xsd:simpleType name="kolor">
```

```
...
```

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

# Typy proste – c.d.

- ▶ Definiują zbiory wartości atomowych (bez wewnętrznej struktury elementów)
- ▶ wszystkie typy wbudowane są typami prostymi  
`<xsd:element name="napis" type="xsd:string"/>`
- ▶ Typy stworzone na bazie typów wbudowanych (używane do określania dopuszczalnych wartości atrybutów i zawartości elementów)

# Typy złożone (Complex Type)

Typy złożone:

- ▶ Zawierające tekst, atrybuty i inne elementy;
- ▶ Określają strukturę dokumentu:
  - Elementy zawierające inne elementy;
  - Elementy zawierające inne elementy i tekst;
  - Elementy zawierające tekst;
  - Elementy puste;

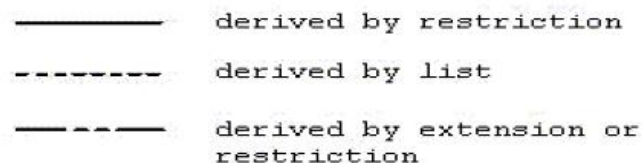
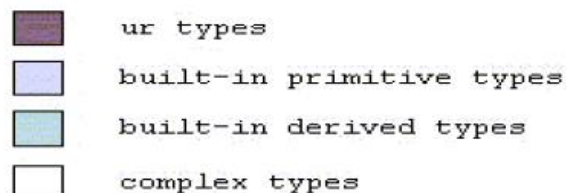
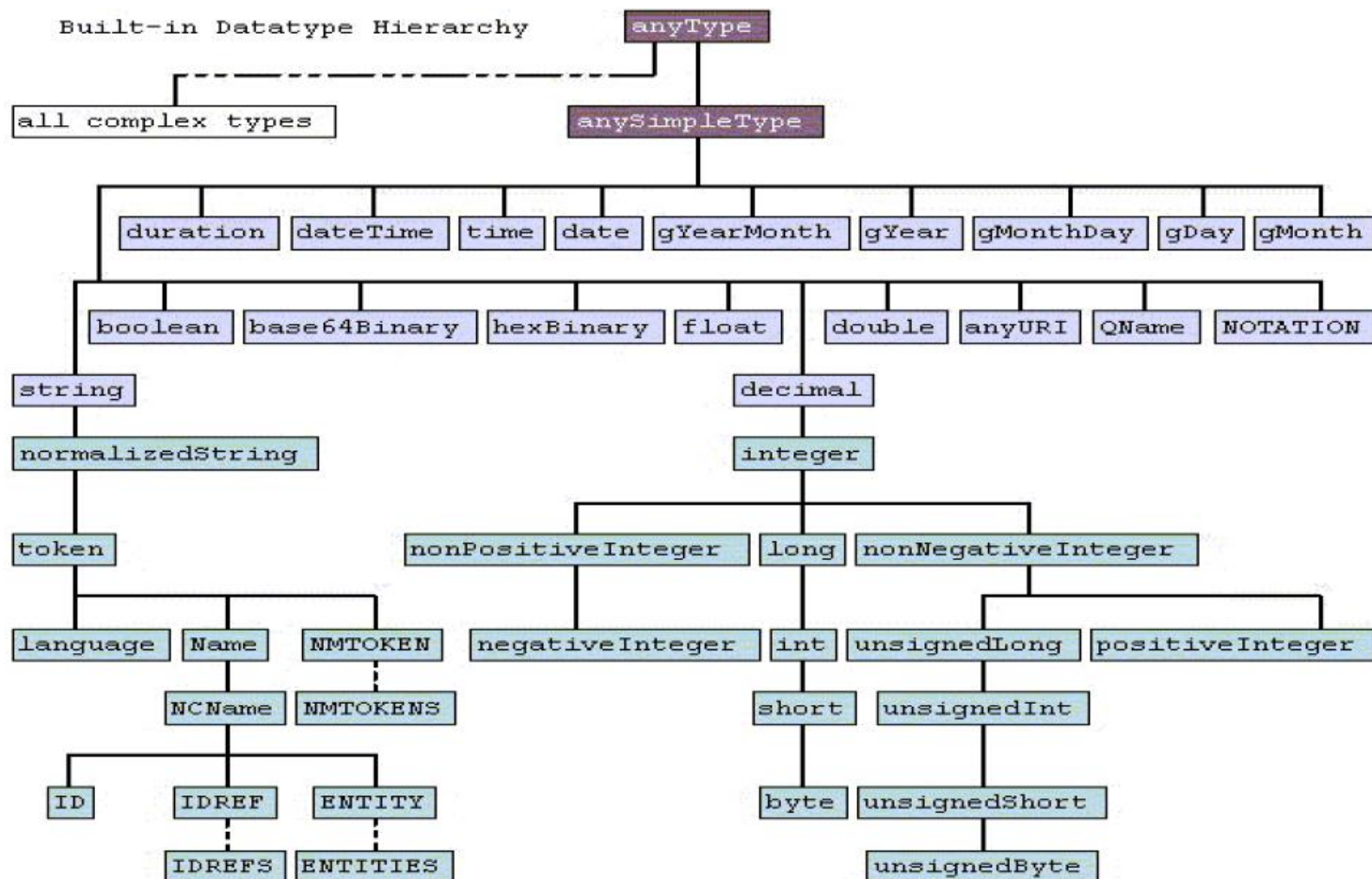
```
<xsd:complexType name="kolor">
```

```
...
```

```
</xsd:complexType>
```



# Typy proste – c.d.



# Wbudowane typy proste

Typy proste – predefiniowane użycie:

```
<xsd:element name="nazwa" type="typ"/>
```

Przykłady typów prostych:

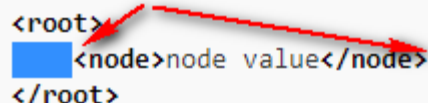
xsd:boolean	– true, false (0 lub 1);
xsd:float	– liczba zmiennoprzecinkowa
xsd:double	– liczba zmiennoprzecinkowa
xsd:decimal	– liczba zapisana dziesiętnie
xsd:integer	– liczba całkowita – większy zakres
xsd:byte	– liczba od -128 do 127;

# Wbudowane typy proste

xsd:date	– data (1999-05-05);
xsd:time	– czas (13:22:12.000);
xsd:century	– wiek
xsd:month	– miesiąc
xsd:year	– rok
xsd:duration	– czas trwania
xsd:uri	– adres URL ( <a href="http://sggw.pl">http://sggw.pl</a> );

# Wbudowane typy proste

- ▶ string – napis
- ▶ normalizedString – napis, w którym każdy biały znak jest podczas przetwarzania zastępowany przez spację
  - Białe znaki – na przykładzie poniżej – tabulator i znak końca wiersza



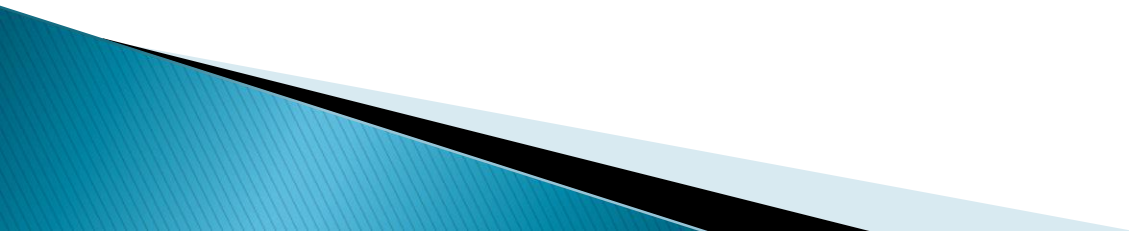
```
<root>  
  <node>node value</node>  
</root>
```

- ▶ token – napis, w którym każdy ciąg białych znaków jest podczas przetwarzania zastępowany przez jedną spację, zaś białe znaki na początku i końcu są usuwane
- ▶ QName – nazwa kwalifikowana
- ▶ NCName – nazwa bez dwukropka
- ▶ base64Binary – dane binarne zapisane w kodowaniu Base64
- ▶ hexBinary – dane binarne zapisane szesnastkowo

# Wbudowane typy proste – przykłady

<i>Typ</i>	<i>Poprawne wartości</i>
decimal	-1.23, 0, 123.4, 1000.00
float, double	-INF, -1E4, -0, 0, 12.78E-2, 12, INF, NaN
QName	os:osoba
date	1968-04-02
time	13:20:00.000, 13:20:00.000-05:00
dateTime	1968-04-02T13:20:00.887
gYearMonth	1968-04
gMonthDay	--04-02
duration	P2Y6M5DT12H35M30S

# Definiowanie typów



# Przestrzenie nazw i schematy

- ▶ Typ atomowy

`<xsd:element name="napis" type="xsd:string"/>`

- ▶ Lista – lista wartości atomowych (w dokumencie rozdzielonych spacjami)
- ▶ Unia typów – połączenie różnych zbiorów poszczególnych typów

# Lista – definiowanie

**list** – tworzenie list wartości typu prostego

```
<xsd:simpleType name="listOfMyInt">  
  <xsd:list itemType="myInteger"/>  
</xsd:simpleType>
```

Wykorzystanie:

```
<listOfMyInt>  
20003 15037 95977 95945  
</listOfMyInt>
```

# Ograniczenia xsd:restriction

- ▶ Można tworzyć własne typy poprzez wprowadzanie ograniczeń typów prostych za pomocą tzw. **facets** (apekty/cechy opisujące typ – relacje równości, porządku itp.)
- ▶ Ograniczenia wskazanego typu prostego przy pomocy tzw. `<xsd:simpleType name=„nazwaTypu”>`
  - `<xsd:restriction base=„jakiśTypProsty”>`
  - `... ograniczenia ...`
  - `</xsd:restriction>`
  - `</xsd:simpleType>`

# Własne typy danych – enumeration – przykład

```
<xsd:element name="kontynent">  
  <xsd:simpleType >  
    <xsd:restriction base="xsd:token">  
      <xsd:enumeration value="Azja"/>  
      <xsd:enumeration value="Afryka"/>  
      <xsd:enumeration value="Europa"/>  
      <xsd:enumeration value="Australia"/>  
    </xsd:restriction>  
  </xsd:simpleType>  
</xsd:element>  
</xsd:schema>
```

Przykładowy plik xml bazujący na powyższym szablonie:

```
<kontynent>Azja</kontynent>
```

# Typy wbudowane

Niektóre ograniczenia (*facets*) mogą być stosowane nie tylko do typów wbudowanych ale także do list:

[length](#),

[minLength](#),

[maxLength](#),

[pattern](#),

[enumeration](#).

# Definiowanie własnego typu prostego

- ▶ Zawężenie zakresu dozwolonych wartości
  - minInclusive, maxInclusive,
  - minExclusive, maxExclusive

```
<xsd:simpleType name="myInteger">  
  <xsd:restriction base="xsd:integer">  
    <xsd:minInclusive value="10000"/>  
    <xsd:maxInclusive value="99999"/>  
  </xsd:restriction>  
</xsd:simpleType>
```

# Tworzenie list – przykład

```
<xs:simpleType name=" NumerLottoTyp">
  <xs:restriction base="xs:integer">
    <xs:minInclusive value="1"/>
    <xs:maxInclusive value="49"/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name=" NumeryLottoTyp">
  <xs:list itemType=" NumerLottoTyp"/>
</xs:simpleType>

<xs:simpleType name="KuponLottoTyp">
  <xs:restriction base=" NumeryLottoTyp">
    <xs:length value="6"/>
  </xs:restriction>
</xs:simpleType>
```

# Tworzenie list – przykład 2

```
<xs:simpleType name="KuponLottoTyp">
  <xs:restriction>
    <xs:simpleType>
      <xs:list>
        <xs:simpleType>
          <xs:restriction base="xs:integer">
            <xs:minInclusive value="1"/>
            <xs:maxInclusive value="49"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:list>
    </xs:simpleType>
    <xs:length value="6"/>
  </xs:restriction>
</xs:simpleType>
```

# Typy wbudowane – przykład

```
<xsd:simpleType name="numbersList">  
  <xsd:list itemType="xsd:int"/>  
</xsd:simpleType>
```

```
<xsd:element name="sixNList" type="SixNumbersList"/>
```

```
<xsd:simpleType name="SixNumbersList">  
  <xsd:restriction base="numbersList">  
    <xsd:length value="6"/>  
  </xsd:restriction>  
</xsd:simpleType>
```

## Przykład

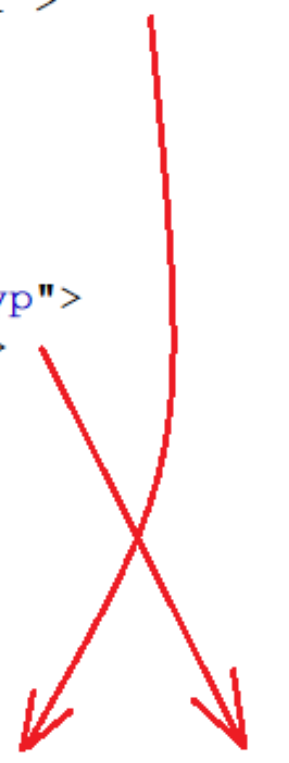
```
<sixNList>  
222  
321  
336  
444  
555  
555  
</sixNList>
```

# Tworzenie unii – przykład

```
<xs:simpleType name=" RozmiarLiczbowyTyp">
  <xs:restriction base="xs:integer">
    <xs:minInclusive value="2"/>
    <xs:maxInclusive value="18"/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name=" RozmiarSMLTyp">
  <xs:restriction base="xs:token">
    <xs:enumeration value="S"/>
    <xs:enumeration value="M"/>
    <xs:enumeration value="L"/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="RozmiarTyp">
  <xs:union
    memberTypes=" RozmiarLiczbowyTyp RozmiarSMLTyp"/>
</xs:simpleType>
```



A diagram consisting of three red arrows. One arrow originates from the closing tag of the first simple type, `</xs:simpleType>`, and points down to the `memberTypes` attribute of the union in the third simple type. A second arrow originates from the closing tag of the second simple type, `</xs:simpleType>`, and also points down to the same `memberTypes` attribute. A third arrow originates from the `memberTypes` attribute and points down to the closing tag of the third simple type, `</xs:simpleType>`. This diagram illustrates how the two individual simple types are combined into a single union type.

# Tworzenie unii – przykład 2

```
<xs:simpleType name="RozmiarTyp">
  <xs:union>
    <xs:simpleType>
      <xs:restriction base="xs:integer">
        <xs:minInclusive value="2"/>
        <xs:maxInclusive value="18"/>
      </xs:restriction>
    </xs:simpleType>
    <xs:simpleType>
      <xs:restriction base="xs:token">
        <xs:enumeration value="S"/>
        <xs:enumeration value="M"/>
        <xs:enumeration value="L"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:union>
</xs:simpleType>
```

# Własne typy danych – enumeration

## prosty typ wyliczeniowy

**enumeration** – ogranicza typ do wyliczonych wartości

```
<xsd:simpleType name=„MateriałyBudowlane">  
  <xsd:restriction base="xsd:string">  
    <xsd:enumeration value=„Deski"/>  
    <xsd:enumeration value=„Cegły"/>  
    <xsd:enumeration value=„Gwoździe"/>  
    <xsd:enumeration value=„Pustaki"/>  
  </xsd:restriction>  
</xsd:simpleType>
```

Przykład zastosowania w pliku XML

```
<MateriałyBudowlane>Cegły</MateriałyBudowlane>
```

# Ograniczenia zakresów typu – string

Typ string i pochodne od niego może mieć następujące ograniczenia:

- ▶ length
- ▶ minLength
- ▶ maxLength
- ▶ pattern
- ▶ enumeration

Ograniczenia długości typu:

```
<xsd: simpleType name=„cena”>  
  <xsd:restriction base=„xsd:string”>  
    <xsd:length value=„6”>  
    lub  
    <xsd:minLength value=„5”/>  
    <xsd:maxLength value=„7”/>  
  </xsd:restriction>  
</xs:simpleType>
```

# Ograniczenia zakresów – przykład

**<xsd:element name=„gotowka”>**

**<xsd:simpleType>**

**<xsd:restriction base=„xsd:integer”>**

**<xsd:maxInclusive value=„5000”/>**

lub

**<xsd:maxExclusive value=„5000”/>**

lub

**<xsd:minInclusive value=„5000”/>**

lub

**<xsd:minExclusive value=„5000”/>**

**</xsd:restriction>**

**</xsd:element>**

**<gotowka>3000</gotowka>**

- mniej niż 5000 lub równo

(nie więcej niż)

- mniej niż 5000

- więcej niż 5000 lub równo

(nie mniej niż)

- więcej niż 5000

**<gotowka>6000</gotowka>**

# Własny typ prosty – union – przykład

- Typ o wartości sumy różnych typów prostych

```
<xsd:simpleType name=„typMoj”>
```

```
  <xsd:union memberTypes=„od1do32 od44do54”/>
```

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

```
<xsd:simpleType name=„od1do32”>
```

```
  <xsd:restriction base="xsd:int">
```

```
    <xsd:minInclusive value="1"/><xsd:maxInclusive value="32"/>
```

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

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

```
<xsd:simpleType name=„od44do54”>
```

```
  <xsd:restriction base="xsd:int">
```

```
    <xsd:minInclusive value="44"/><xsd:maxInclusive value="54"/>
```

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

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

```
<liczba>45</liczba>
```

```
<liczba>12</liczba>
```

```
<liczba>36</liczba>
```

# Własne typy danych – pattern

pattern – służy do definiowania wzorca danych– jako wyrażenia regularnego

```
<xsd:simpleType name="SKU">  
  <xsd:restriction base="xsd:string">  
    <xsd:pattern value="\d{3}-[A-Z]{2}"/>  
  </xsd:restriction>  
</xsd:simpleType>
```

\d{3} – 3 dowolne cyfry

[A-Z]{2} – dwie dowolne małe lub duże litery

# Własne typy danych – pattern – c.d.

## Wzorce (regularne wyrażenia Perla):

- ▶ Litery, symbole i cyfry występują tak jak zostaną podane;
- ▶ `.` – zastępuje dowolny znak;
- ▶ `\d` – dowolną cyfrę
- ▶ `\D` – przeciwnie do `\d`
- ▶ `\s` – dowolny biały znak (spacja, tabulator)
- ▶ `\S` – przeciwnie do `\s`
- ▶ `abc` – ciąg a, b, i c
- ▶ `[abc]` – wystąpienie jednej z wartości z grupy: a, b lub c;
- ▶ `[0-9]` – zakres wartości – od 0 do 9;
- ▶ `x?` – opcjonalne wystąpienie x (może ale nie musi)
- ▶ `Jeden | drugi` – występuje jeden lub drugi;
- ▶ `X{5}` – wystąpi 5 kolejnych X;
- ▶ `X{5,}` – wystąpi co najmniej 5 kolejnych X;
- ▶ `X{5,8}` – wystąpi od 5 do 8 kolejnych X;

# Pattern – przykład

Definicja:

```
<xsd:simpleType name=„kodPocz”>  
  <xsd:restriction base=„xsd:string”>  
    <xsd:pattern value=„\d{2}-\d{3}”/>  
  </xsd:restriction>  
</xsd:simpleType>
```

**<kodPocz>75-632</ kodPocz>**

~~**<kodPocz>735-632</ kodPocz>**~~

# Typy złożone

Typ złożony składa się z deklaracji elementów oraz atrybutów:

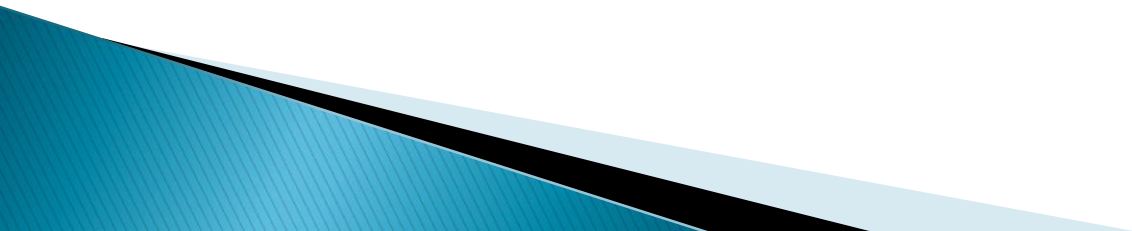
```
<xsd:complexType name=„nazwa”>  
    zawartość  
</xsd:complexType>
```

Definiowanie zawartości:

- |                     |                           |
|---------------------|---------------------------|
| <b>xsd:sequence</b> | – sekwencja elementów     |
| <b>xsd:choice</b>   | – alternatywa elementów   |
| <b>xsd:all</b>      | – dowolny układ elementów |

# Typy złożone

Sekwencja elementów w zawartości może występować w jednej z trzech postaci:

- ▶ **sequence** – elementy muszą wystąpić w określonym porządku (sekwencji)
  - ▶ **choice** – może wystąpić tylko jeden z elementów (wybór)
  - ▶ **all** – wszystkie elementy mogą wystąpić, ale kolejność jest dowolna
- 

# Typ złożony – xsd:sequence

- ▶ Określa kolejność w jakiej mają pojawić się elementy;
- ▶ Muszą wystąpić wszystkie elementy (chyba, że minOccurs= "0")
- ▶ Może zawierać inne sekwencje elementów (np. <xsd:choice>)

```
<xsd:complexType name="typadres">
```

```
  <xsd:sequence>
```

```
    <xsd:element name="ulica" type="xsd:string"/>
```

```
    <xsd:element name="numer.domu" type="xsd:string,, minOccurs="0"/>
```

```
    <xsd:element name="numer.mieszkania" type="xsd:int"/>
```

```
    <xsd:element name="miejscowosc" type="xsd:string"/>
```

```
    <xsd:element name="kod" type="typkod"/>
```

```
  </xsd:sequence>
```

```
</xsd:complexType>
```

# Typ złożony – xsd:choice

- ▶ Daje możliwość wyboru dokładnie 1 elementu z pośród podanych;

```
<xsd:complexType name="typPojazd">  
  <xsd:choice>  
    <xsd:element name="pociag" type="xsd:token"/>  
    <xsd:element name="tramwaj" type="xsd:token"/>  
    <xsd:element name="autobus" type="xsd:string"/>  
  </xsd:choice>  
</xsd:complexType>
```

# Typ złożony – xsd:all

- ▶ Wszystkie elementy mogą wystąpić w dowolnej kolejności
- ▶ Każdy element może wystąpić nie więcej niż raz
- ▶ Jeżeli minOccurs = "0" to może nie wystąpić
- ▶ Nie może zawierać zagnieżdżonych modeli <xsd:sequence> i <xsd:choice>

```
<xsd:complexType name="typAuto">
```

```
  <xsd:all>
```

```
    <xsd:element name="kolor" type="xsd:token"/>
```

```
    <xsd:element name="marka" type="xsd:string"/>
```

```
    <xsd:element name="rokprodukcji" type="xsd:int"/>
```

```
    <xsd:element name="bak" type="xsd:int" minOccurs="0"/>
```

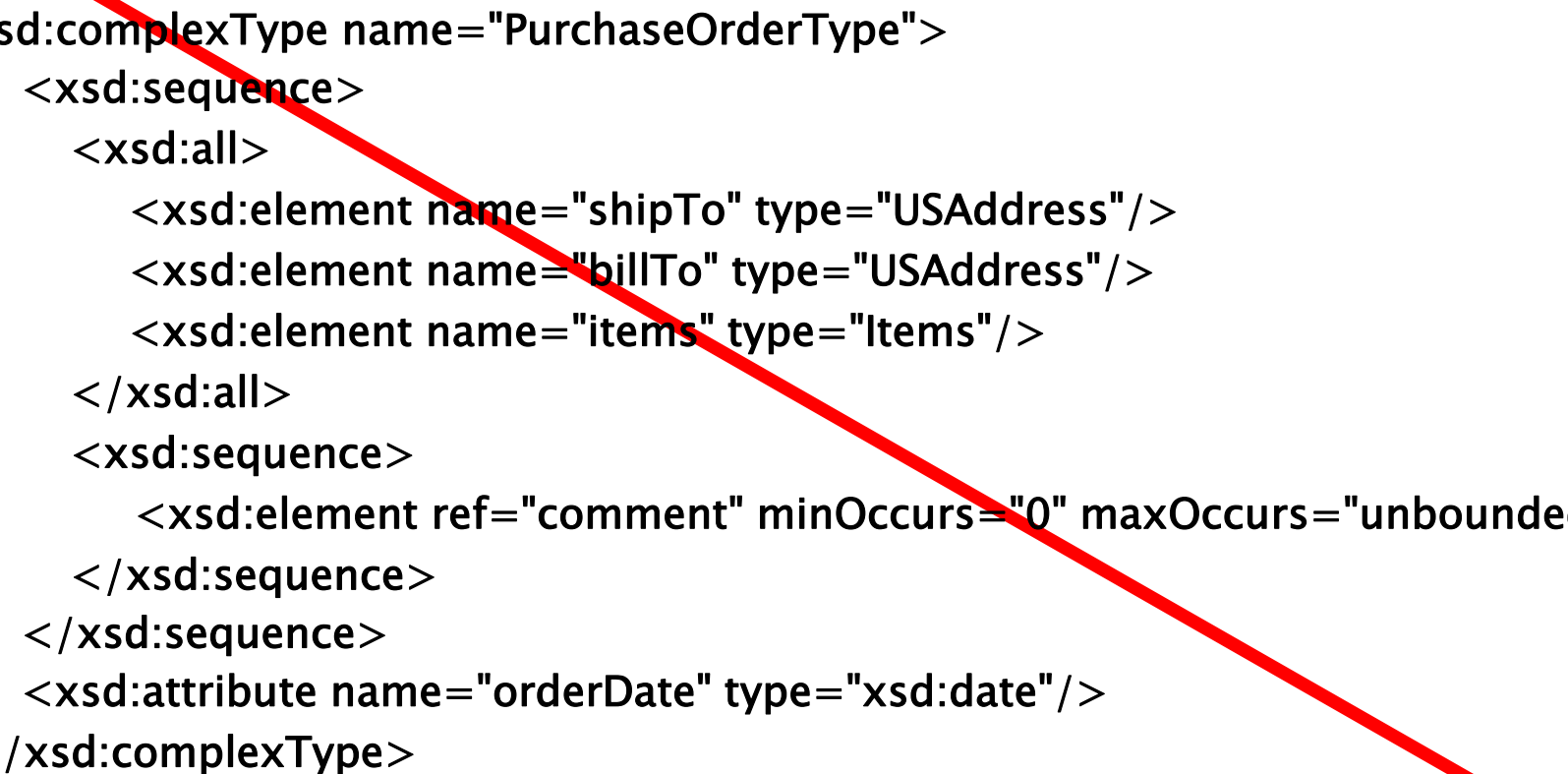
```
  </xsd:all>
```

```
</xsd:complexType>
```

# Zagnieżdżanie elementów typu – xsd:all

Elementy typu xsd:all nie mogą być zagnieżdżane:

```
xsd:complexType name="PurchaseOrderType">
  <xsd:sequence>
    <xsd:all>
      <xsd:element name="shipTo" type="USAddress"/>
      <xsd:element name="billTo" type="USAddress"/>
      <xsd:element name="items" type="Items"/>
    </xsd:all>
  <xsd:sequence>
    <xsd:element ref="comment" minOccurs="0" maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:sequence>
<xsd:attribute name="orderDate" type="xsd:date"/>
</xsd:complexType>
```



# Deklaracja atrybutów

## ► Definicja:

- atrybut o typie określonym – nazwanym
  - atrybuty mogą być tylko typu „prostego”  
`<xsd:attribute name=„nazwa” type=„typ”/>`

- atrybut o typie nieokreślonym – anonimowym  
`<xsd:attribute name=„lotek”>`  
`<xsd:simpleType>`  
`<xsd:list itemType=„xsd:int”/>`  
`</xsd:simpleType>`  
`</xsd:attribute>`

- atrybut lokalny odwołujący się do globalnego  
`<xsd:attribute ref="version”/>`

- **Atrybuty muszą być deklarowane na samym końcu typu złożonego** (tylko i wyłącznie typu złożonego), do którego przynależą, czyli po zadeklarowaniu wszystkich składników typu złożonego – *przykład na kolejnych slajdach*

# Deklaracja atrybutów

- ▶ Użycie atrybutów – za pomocą atrybutu **use**
- ▶ Występowanie atrybutów:
  - Atrybut wymagany – required:  
`<xsd:attribute name=„pozycja” use=„required”... />`
  - Wartość zabroniona – prohibited:  
`<xsd:attribute name=„kolor” use=„prohibited” ... />`
  - Wartość opcjonalna – optional:  
`<xsd:attribute name=„kolor” use=„optional” ... />`

# Występowanie atrybutów – przykład

- ▶ Elementy o poniższym typie mogą zawierać:
  - dowolną niezerową liczbę elementów imie
  - dokładnie jeden element nazwisko.
  - Poza tym element **musi mieć atrybut plec** i **może mieć atrybut email**.

```
<xs:complexType name="OsobaTyp">  
  <xs:sequence>  
    <xs:element ref="imie" minOccurs="1" maxOccurs="unbounded"/>  
    <xs:element ref="nazwisko" minOccurs="1" maxOccurs="1"/>  
  </xs:sequence>  
  <xs:attribute ref="plec" use="required"/>  
  <xs:attribute ref="email" use="optional"/>  
</xs:complexType>
```

# Deklaracja atrybutów – wartości

- ▶ **Wartość domyślna** – default (początkowa atrybutu) – jeżeli atrybut nie zostanie podany to przyjmuje wartość domyślną:

```
<xsd:attribute name=„kolor” default=„blue”... />
```

- ▶ **Wartość ustalona** – fixed– nawet jeżeli atrybut zostanie podany to i tak przyjmuje wartość domyślną:

```
<xsd:attribute name=„kolor” fixed=„blue”... />
```



# Komentarze

W sekcji adnotacji można zawrzeć elementy:

- ▶ **appinfo** – informacje wykorzystywane przez aplikacje
- ▶ **documentation** – komentarze lub tekst – przeznaczone dla użytkowników

<**xsd:annotation**>

<**xsd:appinfo**>

Informacje na temat aplikacji XML do której należy schemat  
Przeznaczony dla aplikacji

</**xsd:appinfo**>

<**xsd:documentation**>

Informacje o schemacie  
Przeznaczony dla ludzi/użytkowników

</**xsd:documentation**>

</**xsd:annotation**>



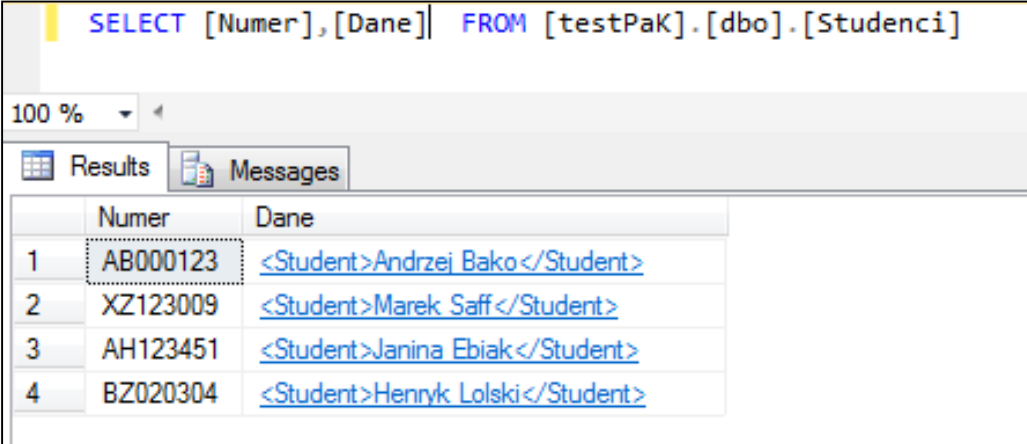
# Typ danych XML w MS SQL Server

# Kolumny XML w tabelach

- 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	<Student>Andrzej Bako</Student>
2	XZ123009	<Student>Marek Saff</Student>
3	AH123451	<Student>Janina Ebiak</Student>
4	BZ020304	<Student>Henryk Loliski</Student>

# Schematy XML na kolumnach XML w tabelach

- ▶ Dane przechowywane w kolumnie typu XML mogą być kontrolowane przez schemat XML podpięty do tej kolumny
- ▶ Do rejestracji typu wykorzystujemy polecenie

## CREATE XML SCHEMA COLLECTION

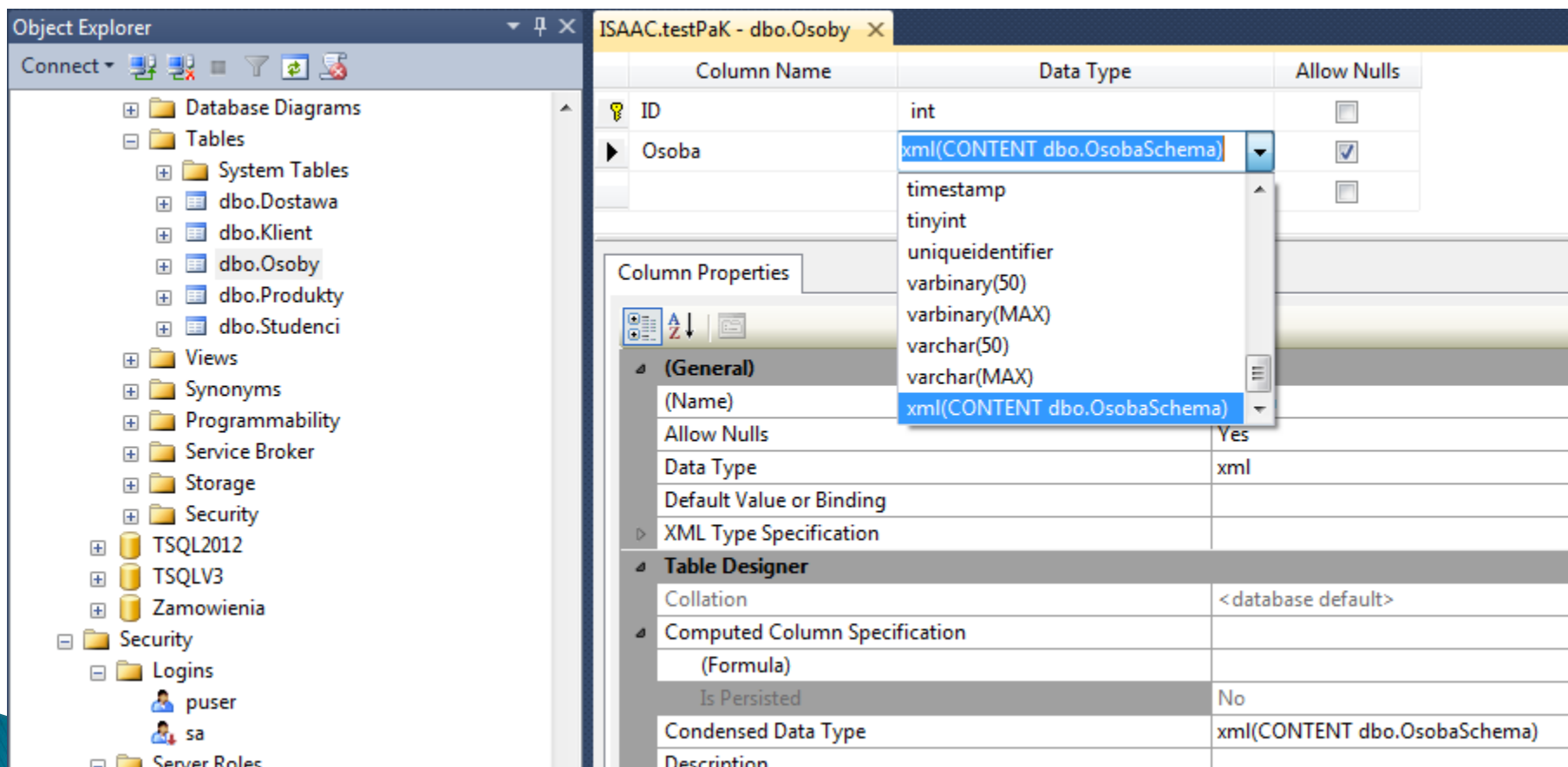
```
CREATE XML SCHEMA COLLECTION OsobaSchema AS
```

```
'<?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))
```

# Schematy XML na kolumnach XML w tabelach

- ▶ Zarejestrowane XML SCHEMA COLLECTION można podpiąć pod kolumnę typu XML



The screenshot displays the SQL Server Enterprise Manager interface. On the left, the 'Object Explorer' shows the database structure, including 'Database Diagrams', 'Tables', 'Views', 'Synonyms', 'Programmability', 'Service Broker', 'Storage', 'Security', 'TSQL2012', 'TSQLV3', 'Zamowienia', 'Security', 'Logins', and 'Server Roles'. The 'dbo.Osoby' table is selected under 'Tables'. The main pane shows the 'Table Designer' for 'ISAAC.testPaK - dbo.Osoby'. The 'Osoba' column is selected, and its data type is being configured as 'xml(CONTENT dbo.OsobaSchema)'. The 'Column Properties' pane is open, showing the 'XML Type Specification' section.

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: Yes

Data Type: xml

Default Value or Binding:

XML Type Specification

Table Designer

Collation: <database default>

Computed Column Specification

(Formula)

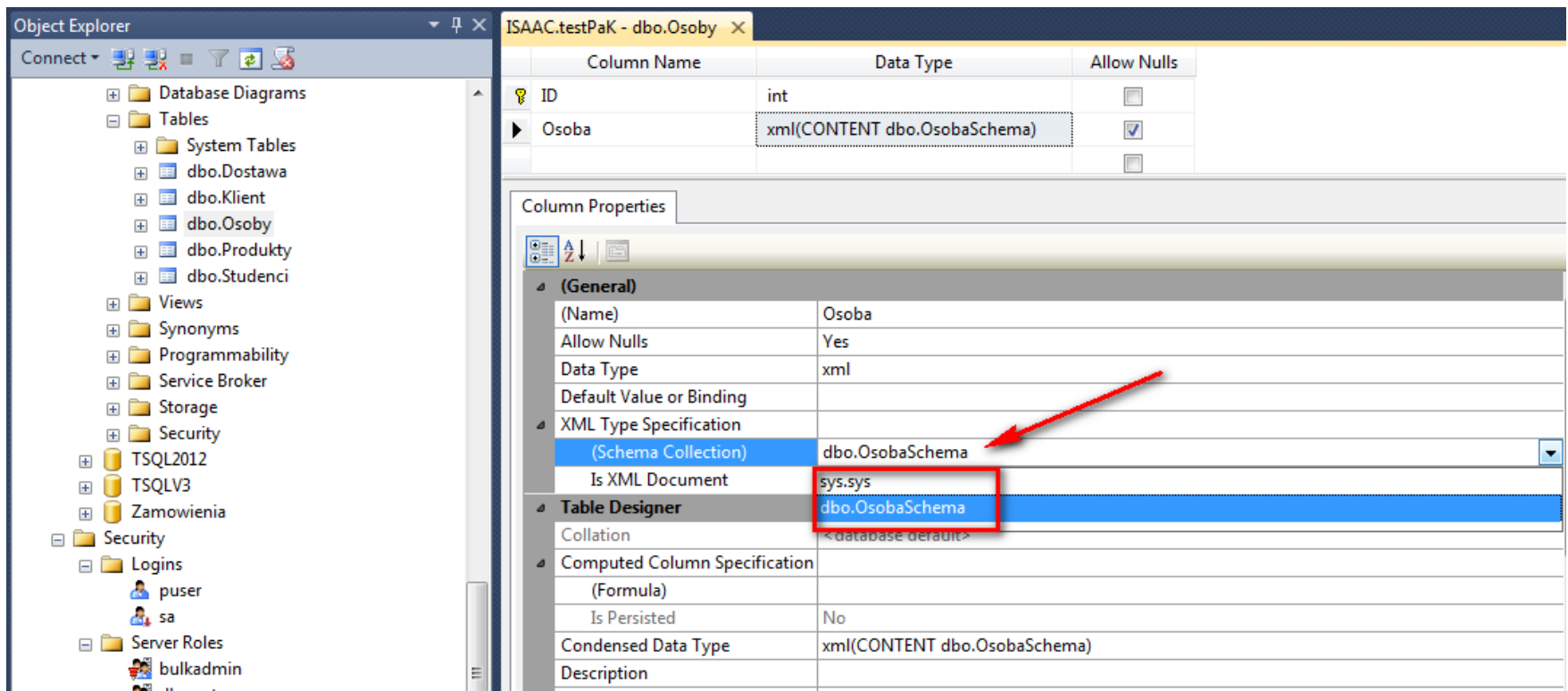
Is Persisted: No

Condensed Data Type: xml(CONTENT dbo.OsobaSchema)

Description:

# Schematy XML na kolumnach XML w tabelach

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



# 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' pane shows the database structure, including 'XML Schema Collections' under the 'dbo' schema. The 'dbo.OsobaSchema' is highlighted. On the right, the 'SQL Query' window shows the execution of the command 'select \* from sys.xml\_schema\_collections'. The 'Results' pane displays a table with the following data:

	xml_collection_id	schema_id	principal_id	name	create_date	modify_date
1	1	4	NULL	sys	2010-04-02 16:59:23.123	2010-04-02 16:59:23.683
2	65543	1	NULL	OsobaSchema	2017-10-10 01:29:41.153	2017-10-10 01:29:41.153

Two red arrows are drawn on the image: one points from the 'dbo.OsobaSchema' in the Object Explorer to the 'OsobaSchema' row in the query results, and the other points from the 'Drop XML SCHEMA COLLECTION OsobaSchema' command text to the same row.

# Schematy XML na kolumnach XML w tabelach

- ▶ Import danych do kolumny XML kontrolowanej przez schemat XML

```
DECLARE @doc xml
```

```
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

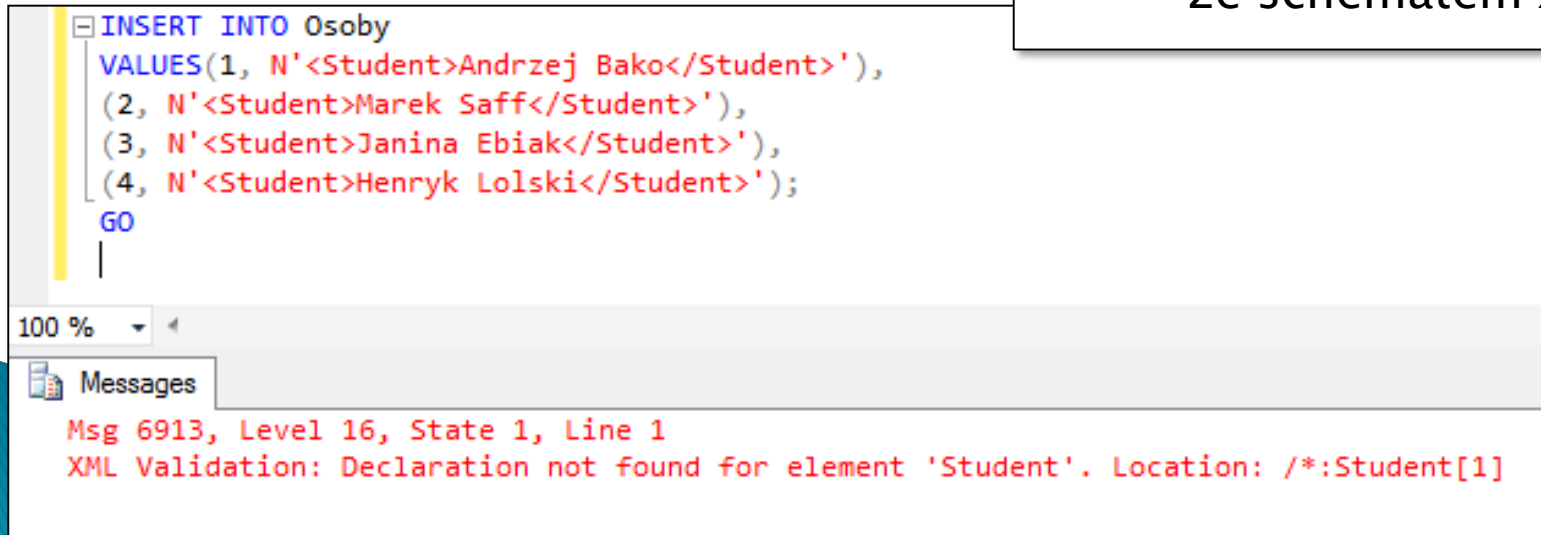
- ▶ Import danych do kolumny XML kontrolowanej przez schemat XML

```
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 Loliski</Student>');
```

```
GO
```

Import danych niezgodnych  
ze schematem XML



The screenshot shows a SQL Server Enterprise Manager interface. The top pane displays a SQL query: 

```
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 Loliski</Student>');  
GO
```

 The bottom pane shows the 'Messages' window with the following error message: 

```
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

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 = 'OsobaSchema';
GO
```

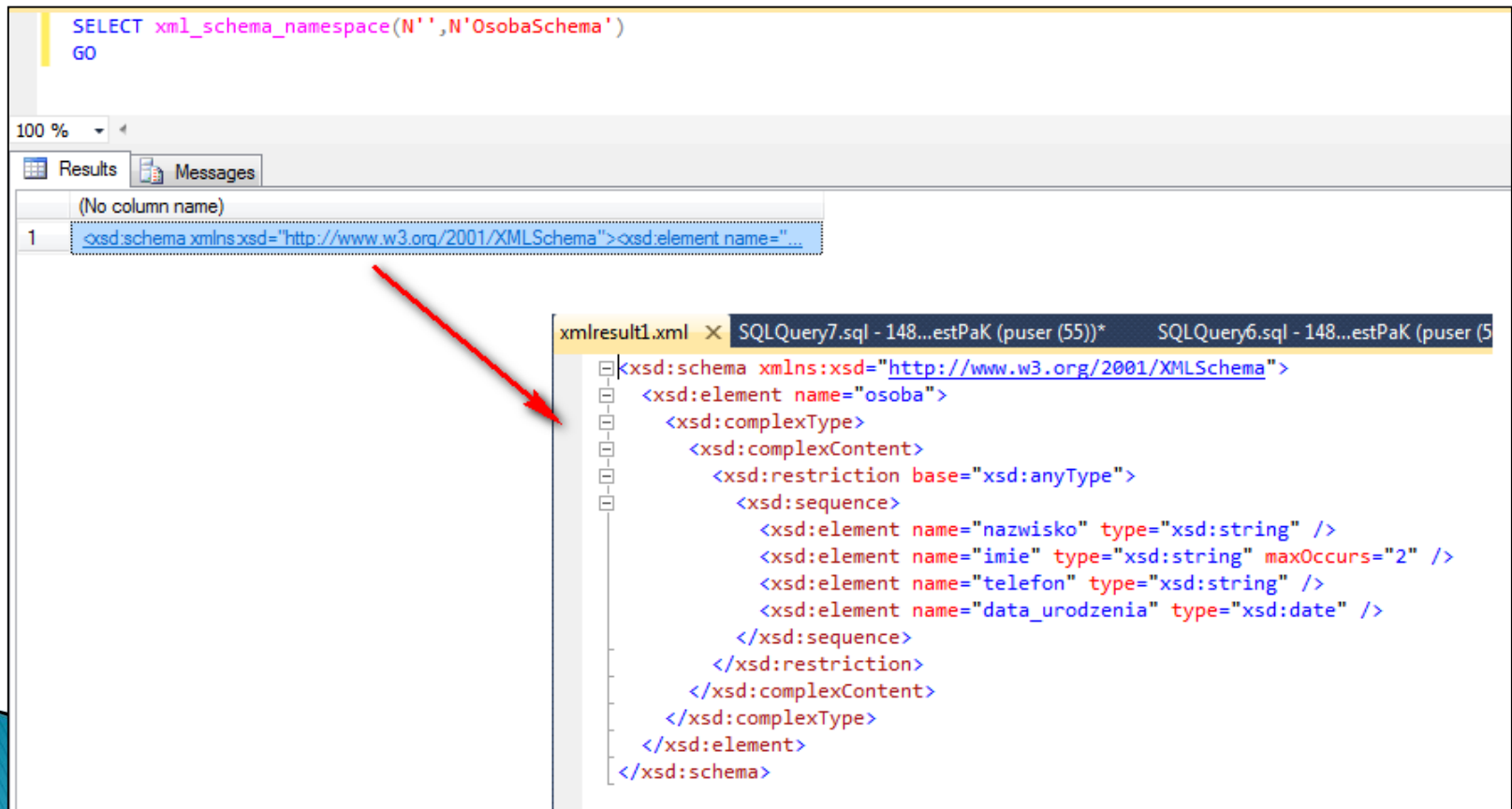
100 %													
Results Messages													
	xml_component_id	xml_collection_id	xml_namespace_id	is_qualified	name	symbol_space	symbol_space_desc	kind	kind_desc	derivation	derivation_desc	base_xml_component_id	scoping_xml_component_id
1	65536	65543	1	1	osoba	E	ELEMENT	E	ELEMENT	N	NONE	NULL	NULL
2	65537	65543	1	0	NULL	T	TYPE	K	COMPLEX_TYPE	R	RESTRICTION	6	65536
3	65538	65543	1	0	NULL	M	MODEL_GROUP	M	MODEL_GROUP	N	NONE	NULL	65537
4	65539	65543	1	0	nazwisko	E	ELEMENT	E	ELEMENT	N	NONE	NULL	65538
5	65540	65543	1	0	imie	E	ELEMENT	E	ELEMENT	N	NONE	NULL	65538
6	65541	65543	1	0	telefon	E	ELEMENT	E	ELEMENT	N	NONE	NULL	65538
7	65542	65543	1	0	data_urodzenia	E	ELEMENT	E	ELEMENT	N	NONE	NULL	65538

# Schematy XML na kolumnach XML w tabelach

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

```
SELECT xml_schema_namespace(N'',N'OsobaSchema')
```

GO



# Pytania?