Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

# XML v databázi Oracle

Uživatelská příručka

**Ondřej Chmelař**

# OBSAH

## ÚVOD:

Uživatelská příručka XML v databázi Oracle byla vytvořena za účelem názorné ukázky, jak využívat jazyk XML (eXtensible Markup Language) v databázi Oracle. V první části příručky je nejprve vysvětleno, co je to XML a jsou zde uvedeny základy syntaxe tohoto značkovacího jazyka.

V dalších částech příručky je postupně vysvětlena konstrukce XML dokumentů, XML schémat, dotazovací jazyk nad XML daty XPath a další metody práce s XML. V pokročilejších částech příručky se vychází z pojmů, které byly vysvětleny v úvodu příručky. V příručce je ke většině metod a funkcím přiložena část kódu, která znázorňuje, jak by měla daná část kódu syntakticky vypadat.

V závěrečné a v podstatě hlavní části této příručky jsou uvedeny příklady, jak aplikovat znalost XML v oblasti databáze Oracle. Nejprve jsou zde vysvětleny základní pojmy, se kterými se v databázích v souvislosti s XML setkáváme, a následně je uvedeno množství příkladů, které přímo ukazují co které funkce, operace nebo změny dělají a jak přímo aplikovat XML v databázi. Tyto příklady byly ověřeny a odlaďovány přímo na databázi Oracle.

# 1 ZÁKLADNÍ PRINCIPY XML

Formát XML - eXtensible Markup Language (rozšiřitelný značkový jazyk) byl definován konsorciem W3C jako formát pro přenos obecných dat a dokumentů. XML je podmnožinou standardu SGML – Standard Generalizad Markup Language. Z formátu SGML vychází také formát HTML - HyperText Markup Language, který má ovšem oproti jazyku XML pevně předdefinovanou sadu značek. V jazyce XML si značky musíme nadefinovat sami.

Shrnutí několika základních faktů o formátu XML:

- Jazyk XML byl vytvořen k popisu dat.
- XML je stejně jako HTML značkovací jazyk.
- V XML nejsou tagy předdefinované, ale musíte si definovat své vlastní.
- XML je vytvořeno tak, aby bylo lehce čitelné a přehledné.
- V XML se používají k popisu dat XML schémata nebo DTD (Definice typu dokumentu).
- Formát XML je normou konsorcia W3C od roku 1998

## 1.1 XML dokument

Fyzicky je XML dokument složen z entit. Entita je určitá posloupnost prvků, ale ještě to není logický element XML dokumentu. Každá fyzická entita může obsahovat nerozpoznatelná a rozpoznatelná data. Nerozpoznatelná data jsou taková, která se procesoru XML nepodaří interpretovat jako znaky, nebo jsou tato data určena pro jinou aplikaci. Rozpoznatelná data se skládají ze znaků zvolené abecedy a představují buď značky (*markups*), nebo znaková data. Značky jsou prvkem, který vyznačuje logickou strukturu dokumentu a tím jeho rozložení (*layout*).

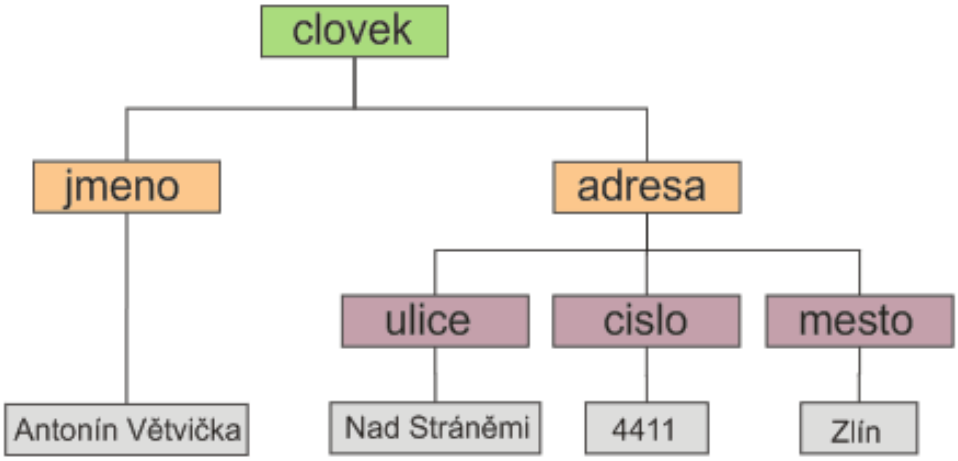
Logicky XML dokument obsahuje:

- prolog
- deklaraci
- elementy
- komentáře
- instrukce pro zpracování jinými aplikacemi

Významným znakem XML dokumentu je jeho stromová struktura s jedním kořenovým elementem (*root element*). Pro značkovací jazyky je charakteristické, že se dokument člení na jednotlivé prvky, které jsou vzájemně provázány (*elementy, atributy, entity, komentáře atd.*), které se zapisují pomocí tzv. *tagů* – významových značek.

Na následujícím příkladu můžeme názorně vidět stromovou strukturu jednoduchého XML dokumentu:

```
<clovek>
  <jmeno>Antonín Větvička</jmeno>
  <adresa>
    <ulice>Nad Stráněmi</ulice>
    <cislo>4511</cislo>
    <mesto>Zlín</mesto>
  </adresa>
</clovek>
```




*Obrázek1 – Ukázka jednoduché stromové struktury XML dokumentu*

Na obrázku 1.1 můžeme vidět znázorněnou stromovou strukturu jednoduchého XML dokumentu z příkladu, který je nad obrázkem.

### 1.1.1 Prolog

Prolog je informace pro software o tom, že se jedná o XML dokument. V prvním řádku prologu se píše deklarace verze daného XML dokumentu, která určuje verzi XML a také označuje typ dokumentu. Deklarace XML není povinná, ale je vhodné ji v dokumentech používat. Píšeme li deklaraci v dokumentu XML, tak je nutné tuto deklaraci uvést hned na začátku tohoto dokumentu. Pro přehlednost dokumentu můžeme případně umístit za prolog nějaký komentář, který zvýší čitelnost dokumentu, ale to není povinné.

Prolog musí obsahovat:

- deklaraci verze XML (máme 1.0 a 1.1)
- může obsahovat informaci o kódování a samostatnosti dokumentu

o Deklarace:

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

o Pokud není v UTF-8:

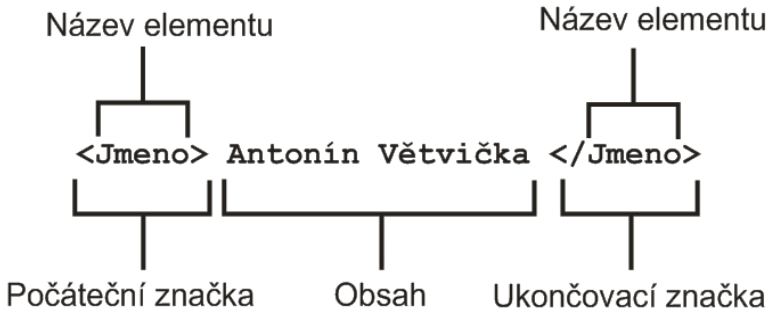
```
<?xml version="1.0" encoding="iso-8859-2"?>
```

o Pokud je bez odkazů mimo dokument:

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

### 1.1.2 Elementy a atributy

Všechny elementy začínají tzv. *počáteční značkou* (start-tag). Start-tag je blok textu, který je uložen v lomených závorkách (**<>**) a obsahuje název elementu případně informace, které ho doplňují. Většina elementů bývá ukončena tzv. *koncovou značkou* (end-tag). End-tag je shodný s počáteční značkou, ale s tím rozdílem, že je před názvem elementu uvedené lomítko (**/**). Data, která jsou mezi počáteční a koncovou značkou elementu jsou *obsahem* elementu. Element může ve svém těle obsahovat i další elementy. Na *obrázku 1* můžete vidět strukturu elementu Jméno.




*Obrázek 2 – Ukázka jednoduchého elementu*

Obsahem elementu může být:

- element
- smíšená hodnota
- jednoduchá hodnota např. text
- žádná hodnota (prázdný element)
- element může mít atributy

Abychom měli správný název elementu, tak musíme dodržovat pravidla:

- Název elementu může obsahovat písmena, čísla i ostatní znaky.
- Název nesmí začínat číslem nebo interpunkčním znaménkem.
- Název nesmí začínat písmeny xml (popř. XML, Xml, atd.).
- Název nesmí obsahovat mezery

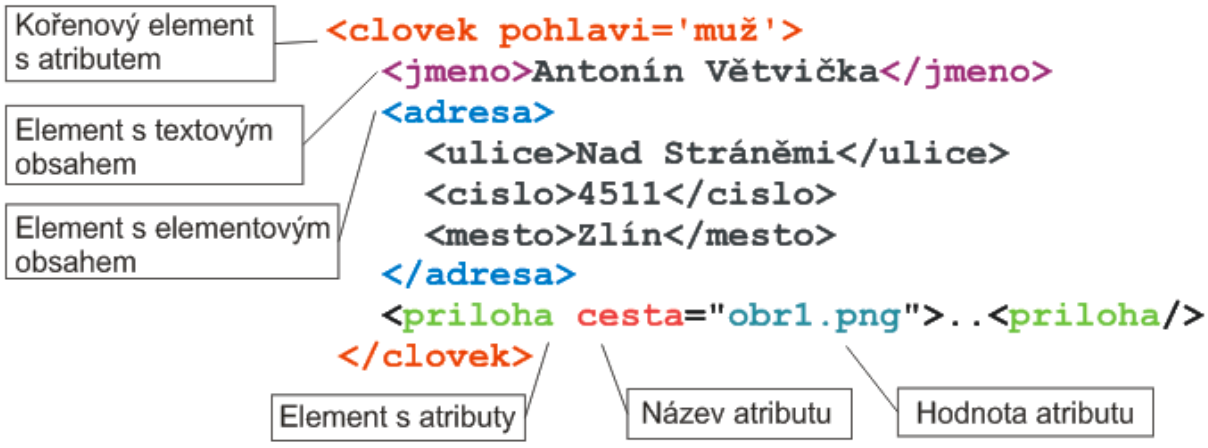
Po dodržení těchto pravidel si v podstatě můžeme zvolit jakýkoliv název elementu, každopádně by bylo velmi rozumné vytvářet názvy, které něco vypovídají o obsahu daného elementu, pro přehlednost dokumentu.

Elementy mohou být blíže charakterizovány tzv. atributy. Atributy se vkládají do otevírací závorky elementu ve formě parametrů. V níže uvedeném příkladě je možné vidět, jak je takovýto atribut zapsán. Atribut je specifikovaný jménem a hodnotou, například tedy **`pohlavi="muž"`**. Hodnota atributu musí být uvedená vždy v apostrofech nebo v uvozovkách.

**Pravidla pro hodnoty atributů:**

Hodnota, kterou přiřazujeme atributu je série znaků, která je ohraničena uvozovkami. Jako hodnotu atributu můžeme stanovit jakoukoli písmennou při dodržení následujících pravidel:

- Řetězec znaků musí být ohraničen buď jednoduchými (‘) nebo dvojitými uvozovkami (“).
- Řetězec musí obsahovat stejné uvozovky, jaké ho ohraničují.
- Řetězec nesmí obsahovat znal levé lomené závorky (<).
- Řetězec může obsahovat odkazy na obecné entity nebo reference.
- Řetězec nesmí obsahovat znak (&) v případě, že jím nezačíná reference na entitu nebo znak.




*Obrázek 3 - jednoduchá syntaxe elementů a atributu elementu*

**Elementy vs. atributy:**

Data můžeme uložit jako element, nebo atribut, jak můžeme vidět na níže uvedených příkladech:

```
1)  <clovek pohlavi=“muž“>
      <jmeno>Antonín</jmeno>
    </clovek>

2)  <clovek>
      <jmeno>Antonín</jmeno>
      <pohlavi>muž</pohlavi>
    </clovek>
```

V prvním příkladě je pohlaví atributem elementu **<clovek>** a v následujícím druhém příkladě je pohlaví potomkem elementu **<clovek>**. V obou těchto případech získáme stejnou informaci. Nejsou stanovena žádná pravidla, která by nám určovala, kdy použít atribut a kdy naopak element

Některé problémy, které mohou nastat při používání atributů:

- Atributy nemohou obsahovat hodnoty, které jsou složené, ale potomci je využívat mohou.
- Atributy nejsou jednoduchým způsobem rozšiřitelné pro budoucí změny.
- Pomocí atributů nemůžeme popisovat struktury, za pomoci potomků ano.
- Atributy jsou horší variantou při editaci nebo při čtení programovým kódem.
- Je obtížné testovat správnost atributu pomocí DTD.

Atributy jsou např. vhodné pro přiřazení ID nějakému elementu, toho můžeme využít především v databázích. ID využijeme jako identifikátor k rozpoznání údaje, který potřebujeme.

### 1.1.3 Prázdný element

Prázdný element v XML dokumentu je element bez jakéhokoliv obsahu. Takovýto element můžeme využít třeba pro vyznačení nějakého místa v našem dokumentu, např. si tak můžeme vyznačit místo, kde později budeme chtít něco vložit. Element, který je prázdný však může obsahovat atributy, které blíže popisují jeho vlastnosti. Můžeme si tak například vytvořit odkaz na objekt, který budeme chtít do daného místa vložit.

```
1) <osoba></osoba>
2) </osoba>
```

První z výše uvedených příkladů ukazuje možný zápis prázdného elementu a druhý příklad je zkrácený zápis takovéhoto prázdného elementu.

### 1.1.4 Kořenový element

Kořenový element musí být obsažen v každém XML dokumentu a je v něm nejvyšším elementem, ve kterém jsou obsaženy všechny další elementy. Červeně vyznačený příklad je nesprávně zapsaný:

```
1) <jmeno>Antonín Větvička</jmeno>
   <ulice>Nad Stráněmi</ulice>
   <cislo>4511</cislo>
   <mesto>Zlín</mesto>

2) <clovek>
       <jmeno>Antonín Větvička</jmeno>
       <ulice>Nad Stráněmi</ulice>
       <cislo>4511</cislo>
       <mesto>Zlín</mesto>
   </clovek>
```

V uvedených příkladech je první příklad nesprávný, protože neobsahuje žádný kořenový element. Druhý příklad již vyhovuje správné syntaxi XML, protože elementy jsou uzavřeny v jednom kořenovém elementu <clovek>. Kořenový element je po prologu druhou hlavní částí dokumentu XML.

### 1.1.5 Komentáře

Komentáře si můžeme do XML dokumentu napsat v podstatě kdekoliv, jen nesmí být umístěny ve značkách. Při využívání komentářů píšeme začátek závorky, ve které bude komentář ve formě "`<!--`" a konec ve formě "`-->`", abychom mohli komentář odlišit od vlastního obsahu dokumentu. V našem komentáři se nesmí vyskytovat řetězec "`--`" z důvodu jeho využití v syntaxi komentářových závorek. Komentáře se nemůžou vnořovat.

```
<!-- Náš komentář k části dokumentu -->
```

Vše co je uvedeno v těle komentáře je bráno jako komentář. Uvedeme-li tedy v komentáři nějaký element, tak tento element je brán jako komentář.

```
<!-- Náš komentář k elementu <Zamestnanec> -->
```

V uvedeném příkladu bude tedy otevírací závorka elementu "Zamestnanec" brána jako komentář.

### 1.1.6 Sekce CDATA

CDATA je sekce typu *znaková data* a využíváme ji v případě, když potřebujeme vložit do dokumentu nějaký blok textu, který by mohl být považován za značku. Přesněji využitím sekce CDATA zamezíme tomu, aby byl tento blok za značku považován.

Máme-li například v dokumentu napsán text:

```
<informace>Doplňující informace</informace>
```

bude chápán jako element <informace> s obsahem "Doplňující informace". Když nebudeme chtít, aby byl text chápán jako značka, ale aby byl jako text součástí obsahu určitého elementu, tak jej musíme zapsat následovně:

```
&lt;informace&gt;Doplňující informace&lt;/informace&gt;
```

Za pomoci sekce CDATA to lze zapsat snadněji:

```
<![CDATA[ <informace>Doplňující informace</informace> ]]>
```

Obecný tvar sekce CDATA je zapsán takto:

```
<![CDATA[text]]>
```

V těle sekce CDATA se nesmí vyskytovat řetězec " ]]>".

### 1.1.7 Instrukce pro zpracování

Instrukce pro zpracování také *procesní instrukce, prováděcí instrukce, processing instructions, PI*. Definice XML dokumentu umožňuje, aby daný dokument obsahoval instrukce, které mají být zpracovány jinou aplikací, než XML procesorem. Tato část dokumentu se nazývá sekce instrukcí (*processing instructions - PI)*. Pro tuto sekci instrukcí se používají speciální závorky se znakem "?". Instrukce musí obsahovat návěští (*PITarget*), kterým se identifikuje aplikace, pro kterou je tato instrukce určena. Návěští nesmí být žádná varianta řetězce písmen "XML". Po návěští následuje text, který může být od návěští oddělen bílým znakem. Text představuje data dané aplikace, pro kterou je určen.

V příkladu je uvedena instrukce, která je pro aplikaci xql:

```
<?xql text?>
```

Obsah instrukcí pro zpracování není brán jako součást dokumentu XML, ale je předán ke zpracování určené aplikaci. Kdybychom chtěli k dokumentu připojit css styl, tak to můžeme provést následující instrukcí:

```
<?xml-stylesheet href="mujstyl.css" type="text/css" ?>
```

## 1.2 Základní pravidla XML

Při vytváření XML dokumentů je třeba si dávat pozor, aby byla syntaxe elementů správně zapsána, proto abychom se vyvarovali zbytečných chyb. Pro správně napsaný XML dokument je třeba dodržovat několik základních pravidel:

1) **Každý element musí mít ukončovací značku (end-tag)**

   Při zápisu elementu nesmíme zapomenout element také ukončit ukončovací značkou. XML nefunguje jako HTML, u kterého prohlížeč dokáže sám vyhodnotit, kde element nejspíše končí.

2) **Názvy elementů rozlišují malá a velká písmena**

```
1) <jmeno>Antonín</JMENO>
2) <jmeno>Antonín</Jmeno>
3) <jmeno>Antonín</jmeno>
```

   První dva červeně znázorněné příklady jsou nesprávné, protože ukončovací značka elementu není totožná s počáteční značkou, správně je tedy pouze třetí příklad.

3) **Elementy musí být správně zanořeny do sebe**

   Je třeba si dát pozor na to, aby elementy byly vhodně zanořeny a nekřížily se, to znamená, že pokud má nějaký element počáteční značku uvnitř jiného elementu, musí v tomtéž elementu také ukončen. Na následujících příkladech si ukážeme, jak vypadá správně a nesprávně zanořený element:

```
1) <clovek><jmeno>Antonín</clovek></jmeno>
2) <clovek><jmeno>Antonín</jmeno></clovek>
```

   V prvním příkladě se jedná o nesprávný zápis, protože dochází ke křížení elementů. V druhém případě jsou již elementy zanořeny správně.

4) **Název značek elementu musí být totožný**

   Název počáteční i koncové značky vytvořeného elementu musí být totožný, kdyby se názvy lišily, tak by byl zápis chybný.

5) **XML dokument musí obsahovat jeden hlavní kořenový element**

   Kořenový element je element nejvyšší úrovně a všechny ostatní elementy musí být vloženy v kořenovém elementu. V uvedeném příkladě je hlavním kořenovým elementem element <clovek>. Žádná část kořenového elementu nesmí být obsažená v jiném elementu.

```
<clovek>
  <jmeno>Antonín Větvička</jmeno>
  <adresa>
    <ulice>Nad Stráněmi</ulice>
    <cislo>4511</cislo>
    <mesto>Zlín</mesto>
  </adresa>
</clovek>
```

# 2 DEFINICE SCHÉMATU XML

XML schéma je samostatný dokument, který definuje obsah a strukturu třídy XML dokumentů. Pomocí XML schéma můžeme popsat elementy a atributy, které mohou být obsaženy v konkrétním XML dokumentu a způsob, jakým mají být tyto elementy rozvrženy v hierarchické struktuře tohoto dokumentu.

- XML schéma:
  - Slouží pro popis přípustné struktury XML dokumentů a je popsané v některém z jazyků určených pro jeho popis.
  - Popisuje atributy a elementy, které se smí v daném XML dokumentu vyskytovat a popisuje vzájemné vztahy mezi nimi.
- Nástroje pro definici struktury:
  - Jazyk DTD (*Document Type Definition*)
  - Jazyk XML Schema
  - Schematron, Relax, atd.

V této příručce se budeme zabývat hlavně tvorbou XML schémat za pomoci jazyka XML Schema, ale určitě je důležité se zde alespoň zmínit o definici XML dokumentu za pomoci jazyka DTD (Document Type Definition).

## 2.1 Definice typu dokumentu DTD

Deklarace typu dokumentu může být součástí XML dokumentu a je to blok XML značek, který se zapisuje do prologu platného XML dokumentu. Tuto deklaraci můžeme umístit kamkoliv do prologu za deklaraci XML, mimo ostatní značky. Když vkládáme deklaraci XML, tak musí být umístěna na začátku dokumentu. Když máme XML dokument definován pomocí jazyka DTD, připojí se k dokumentu tato definice pomocí konstrukce **`DOCTYPE`**. XML dokument je validní pouze v případě pokud odpovídá svému schématu.

Pomocí DTD můžeme definovat:

- Elementy a atributy
- Vztahy mezi elementy a podelementy, vztahy mezi elementy a atributy
- Obsah elementů (textový, elementový, prázdný, smíšený atd.)
- Pořadí elementů a počet výskytů elementů v rámci nadelementu (,+*?|)
- V malé míře také datové typy, povinnost výskytu a implicitní hodnoty atributů

V následujícím příkladě jsou uvedeny ukázky jednoduché definice typu dokumentu pomocí DTD:

```
1)  <?xml version="1.0"?>
    <!DOCTYPE pozdrav [<!ELEMENT pozdrav (#PCDATA)>]>
    <pozdrav>Dobrý den</pozdrav>
2)  <?xml version="1.0"?>
    <!DOCTYPE pozdrav SYSTEM "Pozdrav.dtd">
    <pozdrav>Dobrý den</pozdrav>
```

Deklaraci typu XML dokumentu můžeme uvést přímo v našem dokumentu a je to tzv. *lokální deklarace*, kterou můžete vidět výše na příkladu č. 1. Druhou možností je, že se můžeme odkázat na externí entitu, která tuto deklaraci obsahuje tzv. *externí deklarace*, kterou můžete vidět v druhém uvedeném příkladě, kdy se odkazujeme na deklaraci, která je obsažena v souboru "Pozdrav.dtd". Je také možné využít oba způsoby najednou. Gramatika celého dokumentu se tvoří spojením všech deklarací a lokální deklarace má přednost před externí deklarací.

Jednou z možností je také dokument označit jako samostatný (standalone) a v případě, že tomu tak bude, tak dokument nesmí obsahovat odkazy na externí entity nebo parametry tzn. musí být zpracovatelný sám o sobě. Definice typu dokumentu pomocí jazyka DTD může obsahovat:

Deklaraci typu dokumentu:

```
<!DOCTYPE ... >
```

Deklaraci typu elementu:

```
<!ELEMENT ... >
```

Deklaraci seznamu atributů:

```
<!ATTLIST ... >
```

Deklarace entity:

```
<!ENTITY ... >
```

Deklarace notace:

```
<!NOTATION ... >
```

Příklad XML dokumentu s definicí dokumentu DTD:

```
<?xml version="1.0"?>
<!DOCTYPE uzivatel [
<!ELEMENT uzivatel (jmeno,pohlavi,zamestnani,adresa+)>
<!ELEMENT jmeno (#PCDATA)>
<!ELEMENT pohlavi (#PCDATA)>
<!ELEMENT zamestnani (#PCDATA)>
<!ELEMENT adresa (ulice,cislo,mesto)>
<!ELEMENT ulice (#PCDATA)>
<!ELEMENT cislo (#PCDATA)>
<!ELEMENT mesto (#PCDATA)>
]>

<uzivatel>
  <jmeno>Antonín Větvička</jmeno>
..<pohlavi>muž</pohlavi>
  <zamestnani>IT technik</zamestnani>
  <adresa>
    <ulice>Nad Stráněmi</ulice>
    <cislo>4511</cislo>
    <mesto>Zlín</mesto>
  </adresa>
</uzivatel>
```

## 2.2 XML Schema

Výhody jazyka XML Schema pro tvorbu schémat:

- XML Schema podporuje velké množství datových typů. Jeho obsahem je velká sada vestavěných typů jako jsou např. **`string`**, **`boolean`**, **`date`** a součastně je možnost si v něm definovat vlastní uživatelsky definované datové typy. Je možné také využít vícehodnotové datové typy.
- U jazyka XML Schema oproti DTD není vyžadována žádná speciální syntaxe. Z toho vyplývá, že XML Schémata napsaná v jazyce XML Schema jsou XML dokumenty s pevně danou strukturou. Pro zpracování XML Schémat není nutné umět dva jazyky určené pro jejich zpracování.
- XML Schema umožňuje přesně vyjádřit počty výskytu elementů v rámci jejich rodičovského elementu.
- Při modelování je možné opětovně využívat prvky, které již byly nadefinovány (datové typy, množiny atributů, množiny elementů atd.) v důsledku toho se značně urychluje a zjednodušuje specifikace schématu.
- XML Schema důsledně rozlišuje mezi množinou elementů a posloupností, tzn. že umožňuje v rámci rodičovského elementu zobrazit snadno libovolné pořadí podelementů.
- Je možné využívat mnoho objektově orientovaných prvků při modelování reality, které jsou pro tyto účely využití již ověřené a jsou praktické.
- XML Schema dále umožňuje definovat větší množství elementů se stejným názvem, ale s odlišným obsahem.
- Pomocí XML Schema je možné definovat, jak prázdné elementy, tak i elementy bez předem definovaného obsahu.
- XML Schema zachovává většinu prvků z jazyka DTD.
- Pomocí XML Schema můžeme provést definování několika různými způsoby.

### 2.2.1 Základy XML Schema

XML Schéma napsané v jazyce XML Schema je současně samostatným XML dokumentem, který obsahuje jen speciálně nadefinované elementy. Vzhledem k tomu, že se jedná o samostatný dokument, tak se na začátku každého XML Schématu se musí nacházet prolog a celé XML Schéma musí být uzavřeno v kořenovém elementu **<schema>.**

```
<?xml version="1.0" encoding="utf-8"?>
<schema>
  <!-- Definice XML Schématu -->
</schema>
```

Požadované XML schéma je specifikováno prostřednictvím podelementů a atributů elementů, které jsou jednotlivými prvky jazyka XML Schema a tyto elementy jsou definovány ve jmenném prostoru XML Schema, který musí být připojen k definovanému XML schématu. Po připojení k předchozímu příkladu by to vypadalo následovně:

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           targetNamespace="http://www.fai.utb.cz/MojeSchema"
           xmlns="http://www.fai.utb.cz/MojeSchema"
  <!-- Definice XML Schématu -->
</xs:schema>
```

**xmlns:xs** – Atribut, který specifikuje, že element schema i podelementy v něm, které mají prefix "xs" patří do jmenného prostoru XML Schema. Identifikátorem jazyka XML Schema je řetězec “ http://www.w3.org/2001/XMLSchema“. Jedná se tedy o specifikaci XML schématu v jazyce XML Schema.

**targetNamespace** – je jedním z možných atributů elementu <schema> a určuje URI jmenného prostoru, který definujeme vytvářeným schématem. Tento definovaný jmenný prostor je současně atributem xlmns specifikován jako implicitní jmenný prostor a to proto, abychom se na prvky tohoto prostoru mohli odkazovat bez prefixu.

Následný příklad ukazuje možné připojení XML schématu k XML dokumentu:

```
<?xml version="1.0" encoding="utf-8"?>
<KorenovyElementDokumentu
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.fai.utb.cz/MojeSchema schema.xsd"
    xmlns="http://www.fai.utb.cz/MojeSchema">

  <!-- Náš XML Dokument -->

</KorenovyElementDokumentu>
```

**xmlns:xsi** – atribut, který určuje jmenný prostor instancí XML Schémat, jeho identifikátorem je “ http://www.w3.org/2001/XMLSchema-instance“.

**xsi:schemaLocation** – je globální atribut, který určuje, o jaké XML schéma se jedná a pochází z určeného jmenného prostoru. Hodnotou tohoto atributu je URI jmenného prostoru příslušného XML schématu a URL na xsd soubor, ve kterém je uložené toto schéma.

**xmlns** – určuje, že implicitním jmenným prostorem bude jmenný prostor XML schématu našeho vytvořeného XML dokumentu. To zajišťuje, že názvy elementů můžeme psát bez prefixu.

**xsi:noNamespaceSchemaLocation** – tento atribut využíváme pro určení XML schématu v případě, že nemáme specifikovaný jmenný prostor (tedy atribut **targetNamespace** v elementu **<schema>**). Názvy elementů zde píšeme bez prefixu.

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

  <!-- Náš XML Dokument -->

</KorenovyElementDokumentu>
```

K elementu <KorenovyElementDokumentu> z předchozího příkladu je nutné zmínit, že kořenovým elementem XML dokumentu, který odpovídá určitému XML schématu může být v podstatě jakýkoliv *globálně definovaný element*.

Pojmem *globálně definovaný element* rozumíme všechny elementy, které jsou přímými podelementy elementu <schema>.

V následně uvedeném příkladu můžeme vidět ukázku XML schéma, kde jdou dva globálně nadefinované elementy a to elementy <brigadnici> a <zamestnanci>

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="brigadnici">
      <!-- Definice obsahu elementu -->
    </xs:element>

    <xs:element name="zamestnanci">
      <!-- Definice obsahu elementu -->
    </xs:element>

    <!-- Případná definice následujících elementů -->
</xs:schema>
```

Instalace tohoto schématu do XML dokumentu by mohla vypadat následovně:

```
1)  <?xml version="1.0" encoding="utf-8"?>
    <brigadnici
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:noNamespaceSchemaLocation="brigadnici.xsd"
      <!-- Obsah elementu brigadnici -->
    </brigadnici>

2)  <?xml version="1.0" encoding="utf-8"?>
    <zamestnanci
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:noNamespaceSchemaLocation=" zamestnanci.xsd"
      <!-- Obsah elementu zamestnanci -->
    </ zamestnanci>
```

Oba výše uvedené příklady jsou správné, tedy kořenovým elementem dokumentu může být element <brigadnici> nebo element <zamestnanci>. V tomto příkladě schéma nedefinuje cílový jmenný prostor.

### 2.2.2 Datové typy

Specifikace XML schématu spočívá v tom, že musíme definovat datové typy a následně je přiřadit k elementům nebo atributům.

Rozdělení datových typů:

- *Jednoduché* – tyto datové typy mohou být přiřazeny elementům i atributům. Jednoduché datové typy popisují nestrukturované textové hodnoty.
- *Složené* – mohou být přiřazeny jen elementům. Složené datové typy popisují sekvence elementů a množiny atributů.

### 2.2.3 Jednoduché datové typy

Jednoduché datové typy jsou nejjednodušími typy jazyka XML Schema. Jak již bylo v kapitole 2.2.2 zmíněno, může být tento typ přiřazen jak elementům, tak i atributům. Obsahem elementů a atributů s jednoduchým datovým typem je vždy text, nicméně prostřednictvím tohoto typu můžeme nastavit omezení, která na daný text klademe.

Druhy jednoduchých datových typů:

- *Vestavěné* neboli předdefinované. Tyto datové typy jsou předdefinované v jazyce XML Schema a pokrývají běžně používané datové typy (např. řetězec, celé číslo atd.).
- *Uživatelsky definované* – specifikované uživatelem prostřednictvím odvozování (odvozenými z uživatelsky definovaných typů i z vestavěných).

### 2.2.4 Vestavěné datové typy

Tyto datové typy jsou součástí jazyka XML schema.

Vestavěné datové typy je možné rozdělit na:

- *Základní* – Základní datové typy můžeme vidět v Tabulce 2.1. Těchto typů je 19 a patří mezi ně např. boolean, string, decimal, float atd..
- *Odvozené* – Tyto datové typy jsou odvozené z vestavěných typů. Např. od typu "string" (datový typ, který definuje řetězec znaků) jsou odvozeny typy Name, Language, Token atd. Máme 12 datových typů odvozených od typu "string". Od typu "decimal" je odvozeno 13 číselných datových typů např. integer, int, byte atd.

### 2.2.5 Uživatelsky definované datové typy

Uživatelsky definovaný datový typ je možnost si definovat vlastní jednoduchý datový typ. Definici takto vytvořeného typu je možné provést odvozením nového typu z již existujícího vestavěného, nebo uživatelsky definovaného datového typu.

Definici můžeme provést třemi způsoby:

- Restrikcí (*restriction*)
- Seznamem (*union*)
- Sjednocením (*list*)

Při vytváření nového uživatelsky definovaného typu tento typ definujeme za pomoci elementu **`simpleType.`** Jako podelement tohoto elementu zapíšeme zvolený typ odvození tedy **`restriction, union`** nebo **`type.`**

Když je tento jednoduchý datový typ definován jako globální, tak musí být pojmenován prostřednictvím atributu **`name`**. Pokud budeme chtít zakázat další odvozování od námi vytvořeného typu, tak jej zakážeme pomocí atributu **`final`**. Hodnotou tohoto atribudu bude buď **`restriction, union, type nebo #all`** podle toho co všechno chceme zakázat.

| **Název typu** | **Popis významu** |
|---|---|
| string | Řetězec znaků |
| boolean | Logické hodnoty 0 a 1, případně *true* a *false* |
| decimal | Reálné číslo (kladné nebo záporné) s libovolnou přesností.<br>(*př.* 1.256 , -4.4788) |
| float | 32-bitové číslo s plovoucí desetinou čárkou a jednoduchou přesností vyjádřené pomocí mantisy a exponentu. Případně můžeme využít symboly pro nekonečno (*INF*,*-INF*,*+INF*) nebo výraz "Not a Nubmer" (*NaN*).<br>(*př.* -2E6, 1.24896) |
| double | 64-bitové číslo s plovoucí desetinou čárkou a dvojitou přesností se stejnými vlastnostmi jako *float.* |
| duration | Délka časového úseku. Zapisuje se ve tvaru PnYnMnDTnHnMnS, kde P a T jsou oddělovače, nY znamená n let, nM znamená n měsíců atd.<br>(*př.* -P7Y4M, P1Y2MT3H) |
| dateTime | Datum a čas ve tvaru: YYY-MM-DDThh:mm:ss.ss, kde T je oddělovač. |
| time | Čas ve tvaru hh:mm:ss.ss |
| date | Datum ve tvaru: YYYY-MM-DD |
| gYearMonth | Měsíc v roce ve tvaru: YYYY-MM |
| gYear | Rok ve tvaru: YYYY |
| gMonthDay | Den v měsíci ve tvaru: -MM-DD |
| gMonth | Mesíc ve tvaru: -MM |
| gDay | Den ve tvaru: --DD |
| hexBinary | Kódovaná binární data kódováním *HexBin.* |
| base64Binary | Kódovaná binární data kódováním *Base64.* |
| anyURI | Absolutní nebo relativní URI. |
| QName | Kvalifikované jméno specifikace XML (*XML Qualified Name*).<br>Řetězec ve tvaru *prefix: lokální část,* kde prefix znamená označení jmenného prostoru a lokální část je název prvku, který do tohoto jmenného prostoru patří. |
| NOTATION | Odkaz na notaci. |

*Tabulka 1 – Vestavěné datové typy*

| **Název typu** | **Popis významu** |
|---|---|
| normalizedString | Textový řetězec s normalizovanými bílými znaky, neobsahuje znaky CR, LF a tabulátor |
| token | normalizedString, který nemá mezery na začátku ani na konci a neobsahuje více než jednu po sobě jdoucí mezeru. |
| language | Identifikátor jazyka (př. “en“,“en-GB“ atd.) |
| Name | Řetězec, který obsahuje písmena, číslice, nebo znaky ‘-‘,‘_‘,‘:‘ a ‘.‘ |
| NCName | Name, který neobsahuje znak ‘:‘ |
| NMTOKEN | Hodnota o velikosti jednoho slova. |
| NMTOKENS | Seznam více jednoslovných hodnot. |
| ID | Jednoslovná hodnota v rámci XML dokumentu. |
| IDREF | Odkaz na hodnotu, která je typu ID. |
| IDREFS | Seznam odkazů, které vedou na hodnoty typu ID. |
| ENTITY | Odkaz na entitu. |
| ENTITIES | Seznam odkazů, které vedou na entity. |

*Tabulka2 – Vestavěné datové typy odvozené od typu string*

| **Název typu** | **Popis významu** |
|---|---|
| integer | Celé číslo libovolné délky. |
| positiveInteger | Kladné celé číslo libovolné délky |
| negativeInteger | Záporné celé číslo libovolné délky |
| nonPositiveInteger | Nekladné celé číslo (0 a méně) libovolné délky. |
| nonNegativeInteger | Nezáporné celé číslo (0 a více) libovolné délky. |
| long | 64-bitové celé číslo se znaménkem. $<-2^{63},2^{63}-1>$ |
| int | 32-bitové celé číslo se znaménkem. $<-2^{31},2^{31}-1>$ |
| short | 16-bitové celé číslo se znaménkem. $<-2^{15},2^{15}-1>$ |
| byte | 8-bitové celé číslo se znaménkem. $<-2^{7},2^{7}-1>$ |
| unsignedLong | 64-bitové celé číslo bez znaménka. Celé číslo menší než $2^{64}$ |
| unsignedInt | 32-bitové celé číslo bez znaménka. Celé číslo menší než $2^{32}$ |
| unsignedShort | 16-bitové celé číslo bez znaménka. Celé číslo menší než $2^{16}$ |
| unsignedByte | 8-bitové celé číslo bez znaménka. Celé číslo menší než $2^{8}$ |

*Tabulka3- Vestavěné datové typy odvozené od typu decimal*

### 2.2.6 Atributy

Atributy se skládají z názvu + jednoduchý datový typ a dělíme je:

- Vestavěný (*hodnotou atributu type*)
- Globálně definovaný (*hodnotou atributu type*)
- Lokálně definovaný (*jako podelement*)

Pravidla pro tvorbu atributů v XML schématech jazyka XML Schema:

- Atributy v jazyce XML Schema definujeme pomocí elementu **`attribute`**.
- Každý z jednotlivých elementů musí mít svůj název, který se definuje pomocí atributu **`name`** a také musí mít datový typ.
- Datový typ atributu může být pouze jednoduchý, tedy vestavěný nebo uživatelsky definovaný a můžeme jej určit dvěma způsoby a to buď pomoci atributu **`type`** nebo vytvořením podelementu <**`simpleType>`** k elementu **`attribute`**.
- V případě přiřezování datového typu pomocí atributu **`type`** můžeme přiřadit globálně definované nebo vestavěné typy.
- V případě využití podelementu **`simpleType`** přiřezujeme lokálně definovaný typ určený pouze pro daný atribut.

```
1)  <xs:attribute name="Jmeno" type="xs:string"/>
2)  <xs:attribute name="Vek" type=" xs:positiveInteger"/>
```

U atributů můžeme určit nejen jejich jméno a typ, ale také zda tyto atributy mají být povinné, jaké je jejich defaultní hodnota apod. Např.:

```
3)  <xs:attribute name="Jmeno" type="xs:string" use="required" />
4)  <xs:attribute name="Vek" type=" xs:positiveInteger"
    default="USD"/>
```

**`default –`** touto hodnotou specifikujeme implicitní hodnotu atributu

**`fixed`** – hodnota, kterou specifikujeme konstantní hodnotu atributu

**`use`** – specifikuje nepovinnost, nebo povinnost výskytu atributu v dokumentu, nabývá hodnot **`optional`** a **`required`**: **`optional`** – nepovinný výskyt, **`required`** – povinný výskyt.

Přiřaďme atributy z předchozího příkladu například elementu zaměstnanec v XML dokumentu by výsledná forma vypadala např. následovně:

```
<zamestnanec Jmeno="Vladimír Větvička" Věk="38"/>
```

Jdou nadefinovat i složitější formy atributů např:

```
<xs:attribute name="RodneCislo">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="\d{6}/\d{4}"/>
    </xs:restriction>
  </xs:simpleType>
</xs:attribute>
```

Odpovídající element **RČ** , kterému byl přiřazen atribut **RodneCislo** by vypadal následovně:

```
<RČ RodneCislo="781123/1234"
```

### 2.2.7 Elementy

Elementy nadefinujeme za pomoci elementu <**element>.** Každý z jednotlivých elementů, které si definujeme, musí mít specifikovaný název pomocí atributu **name** a datový typ.

Datový typ elementu může být:

- Jednoduchý určený podelementem **<xs:simpleType>**
- Složený určený podelementem **<xs:complexType>**.
- Datový typ můžeme určit pouze atributem **type**.

V případě, že datový typ elementu specifikujeme pomocí atributu **type**, tak můžeme elementu přiřazovat globálně definované nebo vestavěné typy. V případě, že datový typ elementu specifikujeme pomocí podelementů, tak elementu přiřazujeme pouze lokálně definovaný typ, který je určený pouze pro daný element.

```
<xs:element name="Jmeno" type="xs:string/>
<xs:element name="Vek" type="xs:positiveInteger"/>
```

### 2.2.8 Komplexní datové typy

Komplexní neboli také složené datové typy nám slouží k navrhování struktury dokumentu, protože tyto datové typy se mohou skládat z elementů a atributů. U elementů si můžeme nadefinovat, v jakém pořadí se elementy mají vyskytovat, kolikrát se mohou opakovat a zda je jejich definice povinná, nebo volitelná.

Komplexní datový typ nadefinujeme pomocí elementu **`complexType`**. Tento datový typ můžeme nadefinovat také samostatně, aby šel používat opakovaně pro různé elementy.

Na následujícím příkladu si můžeme ukázat jak nedefinovat dva elementy (Zamestnanec a Brigadnik) tak, že mají stejný obsah pomocí společně uživatelsky definovaného komplexního typu **`MujTyp`**.

```
<xs:element name="Pracovnici">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Zamestnanec" type="subjektType"/>
      <xs:element name="Brigadnik" type="subjektType"/>
      ...
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:complexType name="MujTyp">
  <xs:sequence>
    <xs:element name="Jmeno" type="xs:string" />
    <xs:element name="Adresa" type="xs:string" />
    <xs:element name="Pohlavi" type="xs:string" />
    <xs:element name="Vek" type=" xs:positiveInteger" />
  </xs:sequence>
</xs:complexType>
```

Uvnitř elementu **`complexType`** pak můžeme použít další elementy **`sequence`**, **`choice`** a **`all`**. **`sequence`** - Elementy nadefinované uvnitř elementu **`sequence`** se v dokumentu musí v dokumentu vyskytovat v takovém pořadí v jakém byly nadefinovány.

**`choice`** – Tento element znamená, že se v dokumentu může vyskytovat pouze jeden z obsažených elementů.

**`all`** – Definuje, že se elementy mohou vyskytovat v libovolném pořadí.

### 2.2.9 Sekvence elementů – sequence

Při využití komplexního datového typu se nejčastěji využívá konstrukce sekvence elementů. Sekvenčně definovaný element znamená, že se v něm obsažené definice elementů musí vyskytovat přesně v uvedeném pořadí. Počet výskytů jednotlivých elementů je možné ovlivnit pomocí atributů minOccurs a maxOccurs. Implicitně je na nich nastavena hodnota jedna, což znamená, že element se musí v dokumentu vyskytovat (povinně).

```
<Zamestnanec>
  <jmeno>Jaroslav Lipa</jmeno>
  <pozice>IT technik</pozice>
  <osobniUdaje>...</osobniUdaje>
  <osobniUdaje>...</osobniUdaje>
</Zamestnanec>

<xs:element name="Zamestnanec">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="jmeno" type="xs:string"/>
      <xs:element name="pozice" type="xs:string" minOccurs="0"/>
      <xs:element name="osobniUdaje" type="xs:string"
maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

V předchozím příkladu je nadefinován element pro uložení zaměstnance, který má povinné jméno, nepovinnou pracovní pozici a dále máme libovolný počet osobních údajů přičemž musí být uveden alespoň jeden element.

### 2.2.10 Výběr jednoho z elementů – choice

Choice se využívá v případě, že chceme říci, že se na jednom místě v daném dokumentu využívá jeden z několika elementů. Na jejím místě se pak může vyskytovat jakýkoliv z elementů. které byly nadefinovány v konstrukci

```
<Firma>
  <zamestnanec>
    <jmeno>Pavel Novák</jmeno>
    <rc>781123/1234</rc>
  </zamestnanec>
  <zamestnanec>
    <jmeno>Jaroslav Lípa</jmeno>
    <pas>123127831</pas>
  </zamestnanec>
  <zamestnanec>
    <jmeno>Josef Dobrý</jmeno>
    <SSN>7896544268</SSN>
  </zamestnanec>
</Firma>
```

```
<xs:element name="Firma">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="zamestnanec" maxOccurs="unbounded">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="jmeno" type="xs:string"/>
            <xs:choice>
              <xs:element name="rc" type="xs:string"/>
              <xs:element name="pas" type="xs:string"/>
              <xs:element name="SSN" type="xs:string"/>
            </xs:choice>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Uvedený příklad je schéma zaměstnanců firmy a u každého zaměstnance musí být vždy uveden alespoň jeden z identifikátorů rc (rodné číslo), číslo pasu, a nebo č. soc. pojištění.

### 2.2.11 Elementy v libovolném pořadí – all

Obsah komplexního datového typu definujeme pomocí elementu all když nám nezáleží na tom v jakém pořadí jsou elementy uvedeny. Element je omezen tím, že počet jednotlivých elementů ve skupině all je omezen od nuly do jedné. Větší počet výskytů není povolen z důvodů velké složitosti obsahu dokumentu.

```
<zamestnanec>
  <jmeno>Josef Dobrý</jmeno>
  <pozice>Manager</pozice>
</zamestnanec>
<zamestnanec>
  <pozice>Petr Novák</pozice>
  <jmeno>Operátor</jmeno>
</zamestnanec>
<zamestnanec>
  <titul>Ph.D.</titul>
  <jmeno>Pavel Novák </jmeno>
  <pozice>Josef Dobrý</pozice>
</zamestnanec>
<zamestnanec>
  <jmeno>Jaroslav Lípa</jmeno>
  <pozice>IT technik</pozice>
  <titul>Ing.</titul>
</zamestnanec>

<xs:element name="zamestnanec">
  <xs:complexType>
    <xs:all>
      <xs:element name="jmeno" type="xs:string"/>
      <xs:element name="pozice" type="xs:string"/>
      <xs:element name="titul" type="xs:string" minOccurs="0"/>
    </xs:all>
  </xs:complexType>
</xs:element>
```

Příklad ukazuje definici elementu zamestnanec, který může obsahovat jméno a pozici zaměstnance v libovolném pořadí a také titul zaměstnance.

V tomto schématu by byl problém zaměstnanec se dvěma tituly (před i za jménem), protože uvnitř skupiny all nejde nastavit maxOccurs větší než 1.

### 2.2.12 Prázdné elementy

Prázdné elementy v dokumentu mají většinou své atributy a ty se v komplexním typu definovat jako jeho součást. Např. můžeme mít element **osoba** s atributem **pohlaví**, ale lze ovšem vymyslet mnoho dalších příkladů. Plná syntaxe vychází z představy, že elementu odebereme jeho obsah a přidáme k němu atribut, taková definice elementu by vypadala následovně:

```
<osoba pohlavi="muž"/>

<xs:element name="osoba">
  <xs:complexType>
    <xs:complexContent>
      <xs:restriction base="xs:anyType">
        <xs:attribute name="pohlavi" type="xs:string"/>
      </xs:restriction>
    <xs:complexContent>
  </xs:complexType>
</xs:element>
```

### 2.2.13 Prázdné hodnoty

V XML je z pohledu databází menším problémem, že zde neexistuje standardní forma pro to, když by nějaký element obsahoval nedefinovanou hodnotu **NULL**.

```
1)  <osoba></osoba>
2)  </osoba>
```

Pro zachycení nedefinované hodnoty můžeme využít globálního atributu:

```
1)  <osoba xsi:nil="true"></osoba>
2)  <osoba xsi:nil="true"/>
```

Ve schématu nadefinujeme element, který může obsahovat prázdnou hodnotu takto:

```
<xs:element name="osoba" nillable="true" type="xs:string"
```

### 2.2.14 Přístup k návrhu schématu

Máme více možností k tomu jak správě sestavit XML schéma. V dokumentu je třeba správně rozhodnout kdy využívat globálně a kdy lokálně nadefinované elementy, to má vliv na to, jak moc bude schéma použitelné v dalších schématech.

Jako první možnost můžeme využít to, že jeden element máme nadefinovaný jako globální a všechny ostatní elementy, které jsou uvnitř tohoto elementu jsou brány jako lokální:

```
1)  <xs:element name="zamestnanec">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="jmeno" type="xs:string"/>
          <xs:element name="adresa" type="xs:string"/>
          <xs:element name="pozice" type="xs:string"/>
          <xs:element name="plat" type="xs:decimal"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
```

Výhodou tohoto schématu je, že je krátké a kompaktní z toho vyplývá, že i rychle napsané, ovšem instance tohoto schématu může začínat pouze elementem zaměstnanec.

Druhou možností je, že jsou všechny elementy nadefinovány jako globální a poté jsou složeny dohromady za pomoci odkazů

```
2)  <xs:element name="jmeno" type="xs:string"/>
    <xs:element name="adresa" type="xs:string"/>
    <xs:element name="pozice" type="xs:string"/>
    <xs:element name="plat" type="xs:decimal"/>

    <xs:element name="zamestnanec">
      <xs:complexType>
        <xs:sequence>
          <xs:element ref="jmeno"/>
          <xs:element ref="adresa"/>
          <xs:element ref="pozice"/>
          <xs:element ref="plat"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
```

V takto navrženém schématu může dokument začínat libovolným elementem a kterýkoliv z elementů můžeme znovu použít. Nevýhodou tohoto přístupu je, že pro element stejného názvu nemůžeme definovat dva různé modely obsahu. V prvním přístupu tato možnost nabídnuta je.

Ve třetím a posledním přístupu nejprve nadefinujeme typy, které následně můžeme znovu používat. Elementy jsou pak pomocí těchto typů definovány lokálně (při shodě jmen mohou mít různé modely obsahu). Nevýhodou tohoto přístupu je velká pracnost a složitost.

```
3)  <xs:simpleType name="jmenoType">
      <xs:restriction base="xs:string">
        <xs:minLength value="1"/>
        <xs:maxLength value="20"/>
      </xs:restriction>
    </xs:simpleType>
    <xs:simpleType name="adresaType">
      <xs:restriction base="xs:string">
        <xs:minLength value="1"/>
        <xs:maxLength value="20"/>
      </xs:restriction>
    </xs:simpleType>
    <xs:simpleType name="poziceType">
      <xs:restriction base="xs:string ">
        <xs:minLength value="1"/>
        <xs:maxLength value="10"/>
      </xs:restriction>
    </xs:simpleType>
    <xs:simpleType name="platType">
      <xs:restriction base="xs:decimal">
        <xs:minInclusive value="0"/>
      </xs:restriction>
    </xs:simpleType>
    <xs:complexType name="zamestnanecType">
      <xs:sequence>
        <xs:element name="jmeno" type="jmenoType"/>
        <xs:element name="adresa" type="adresaType"/>
        <xs:element name="pozice" type="poziceType"/>
        <xs:element name="plat" type="platType"/>
      </xs:sequence>
    </xs:complexType>
    <xs:element name="zamestnanec" type="zamestnanecType"/>
```

# 3 XPATH

Jazyk XPath je základní dotazovací jazyk nad XML dokumenty. Jak již bylo dříve zmíněno, tak XML data mají stromovou strukturu a jazyk XPath umožňuje vybírat uzly z této stromové reprezentace XML dat.

## 3.1 Úvod do jazyka XPath

Prvním účelem tohoto jazyka je adresování vybraných částí XML dokumentů, ale tento jazyk obsahuje také základní operace, které nám umožňují manipulovat s XML daty (řetězci, čísly a log. hodnotami). Pro zapisování dotazů v jazyce XPath je využívána kompaktní syntaxe, ve které není značkování

XPath je charakterizován následujícími rysy:

- XPath je základní jazyk, který slouží k dotazování nad XML dokumenty. Resp. k výběru částí XML dokumentu.
- Základní myšlenka jazyka XPath se podobá navigaci v systému souborů.
- Notace jazyka XPath není vyjádřena v jazyce XML.
- Na XPath jsou založeny jazyky XSLT, XPointer, XQuery

### 3.1.1 Datový model XPath

Každý dotazovací jazyk nad daty pracuje s určitým modelem dat, což je pro daný jazyk charakteristické. Datový model v XPath je orientovaný stromově a rozlišuje se v něm sedm typů uzlů. Typy uzlů v modelu:

- kořenový uzel
- uzly elementů
- textové uzly
- uzly atributů
- uzly komentářů
- uzly instrukcí pro zpracování
- uzly jmenných prostorů

Pár dalších faktů o datovém modelu:

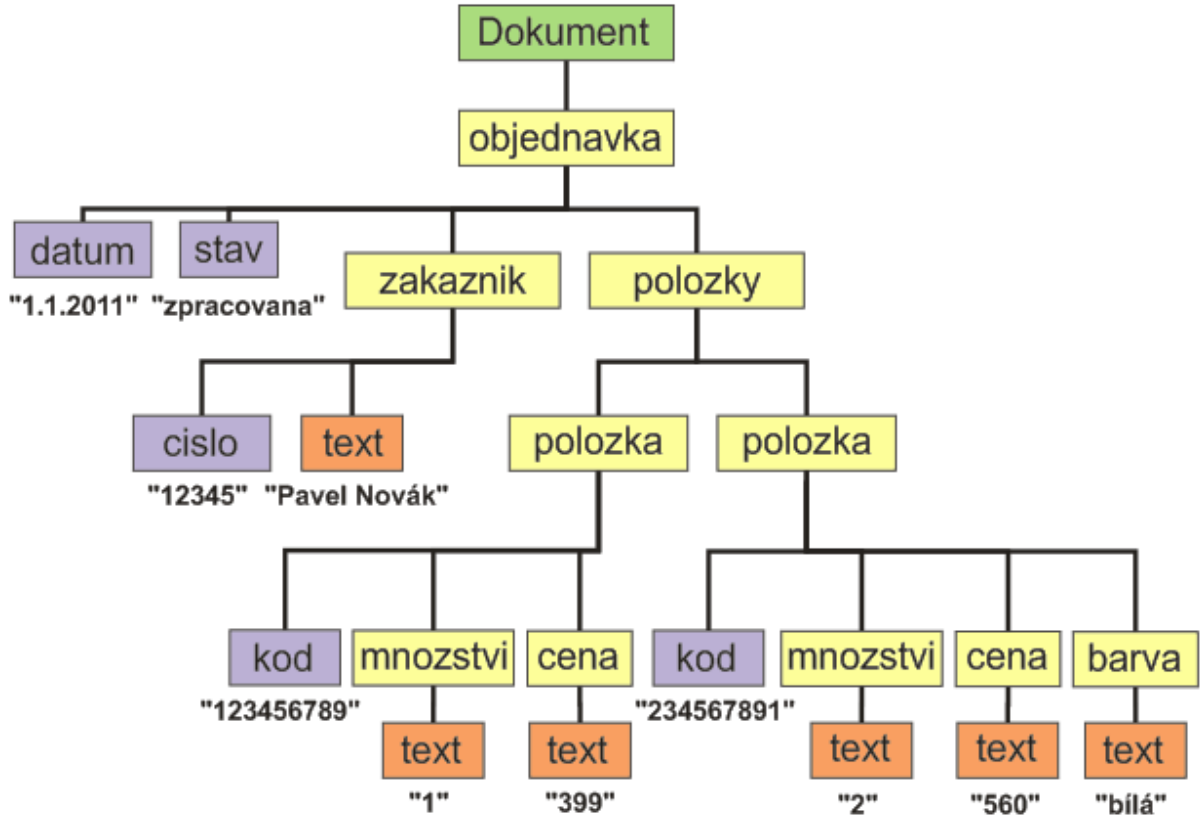
- V typech uzlů modelu chybí sekce CDATA a DTD.
- Kořen datového stromu není stejný jako kořenový element resp. uzel kořenového elementu.
- **`xmlns`** mají ve stromové struktuře zvláštní uzly.

Specifika:

- Atributy nejsou zařazeny do seznamu potomků a dětí daného uzlu.
- Jako speciální uzly jsou chápány atributy ze jmenného prostoru **`xmlns`**.
- Kořenový uzel reprezentuje celý XML dokument. Kořenový element je brán jako dítě kořenového uzlu.

Ukázka modelu XML dat, budeme při ní vycházet z této ukázky XML dokumentu:

```
<objednavka datum="1.1.2011" stav="zpracovana">
  <zakaznik cislo="12345">Pavel Novák</zakaznik>
  <polozky>
    <polozka kod="123456789">
      <mnozstvi>1</mnozstvi>
      <cena>399</cena>
    </polozka>
    <polozka kod=“234567891">
      <mnozstvi>2</mnozstvi>
      <cena>560</cena>
      <barva>bílá</barva>
    </polozka>
  </polozky>
</objednavka>
```




*Obrázek 4 – Model XML dat v XPath*

Z obrázku č.4 můžeme vidět, že uzly ve stromu mají klasické uspořádání, které je dané standardním uspořádáním jednotlivých položek v dokumentu.

Proto, abychom mohli pracovat s XML modelem dat, je důležité znát pojem *řetězcová hodnota uzlu*.

| **Typ uzlu** | **Řetězcová hodnota uzlu** |
|---|---|
| kořen | zřetězení všech potomků kořenu a textových uzlů podle uspořádání |
| element | zřetězení všech potomků elementu a textových uzlů podle uspořádání |
| atribut | hodnota atributu |
| text | data, která jsou obsažena v textovém uzlu |
| komentář | obsah komentáře |
| instrukce pro zpracování | část instrukce pro zpracování |
| jmenný prostor | URI daného jmenného prostoru |

*Tabulka 4 – Tabulka řetězcových hodnot*

### 3.1.2 Výrazy

Hlavním syntaktickým konstruktorem v jazyce XPath je výraz (*expression*). Výsledkem vyhodnocení takového výrazu mohou být objekty. Objekty se dělí do čtyř základních typů:

- Množina uzlů (*node set*)
- Logická hodnota (*boolean*)
- Číslo (*number*)
- Řetězec (*string*)

Vyhodnocení všech výrazů probíhá vzhledem k nějakému kontextu. Kontext si můžeme představit jako určitou množinu proměnných, které jsou k dispozici po dobu vyhodnocování daného výrazu. Specifikace XPathu má v sobě základní definici kontextu.

Obsah kontextu definovaného ve specifikaci: kontextový uzel, knihovna funkcí, množina proměnných, množina deklarací jmenných prostorů, pozice a velikost kontextu.

### 3.1.3 Dotazy

V jazyce XPath mají výhradní postavení cesty, které tvoří klíčový konstrukt tohoto jazyka.

- *Krok* – jeden krok v modelu XML dat ve tvaru x/l.
- *Cesta* – cesta se dá definovat jako seznam kroků. Cesta je v podstatě nejdůležitější částí syntaxe jazyka XPath. Takovýto výraz vyjadřuje cestu ve stromu XML dokumentu a jeho výsledkem je množina uzlů, které splňují podmínku danou výrazem. Každá cesta se skládá z posloupnosti výrazů typu *krok* (Step), které jsou odděleny lomítkem "/". Kroky se vyhodnocují postupně zleva doprava a výsledkem dané cesty je, jak již bylo zmíněno množina uzlů. Největší krok se vyhodnocuje v kontextu, který je na začátku výrazu. Výsledek každého jednotlivého kroku určuje kontext pro následující krok. Po vyhodnocení aktuálního kroku se vždy mění velikost kontextu (context size) a množina uzlů. Výsledkem posledního kroku je vyhodnocení celého výrazu a nazveme jej cílovou množinou.

Cesty dělíme na dva typy:

- *absolutní cesta* – začíná lomítkem "/" a počátečním kontextovým uzlem absolutní cesty je vždy kořenový uzel dokumentu.
- *relativní cesta* - počátečním uzlem takové cesty je uzel kontextu, ve kterém je výraz vyhodnocován

Složení jednotlivých kroků (3části):

- *osa (axis)* – určuje množinu všech uzlů, které jsou v nějakém vztahu s uzlem kontextovým.
- *test uzlu (node test)* – určuje typ uzlů, nebo jméno uzlů na dané ose.
- *predikát (predicate)* – podmínka, kterou musí splňovat uzel, který leží na dané ose.

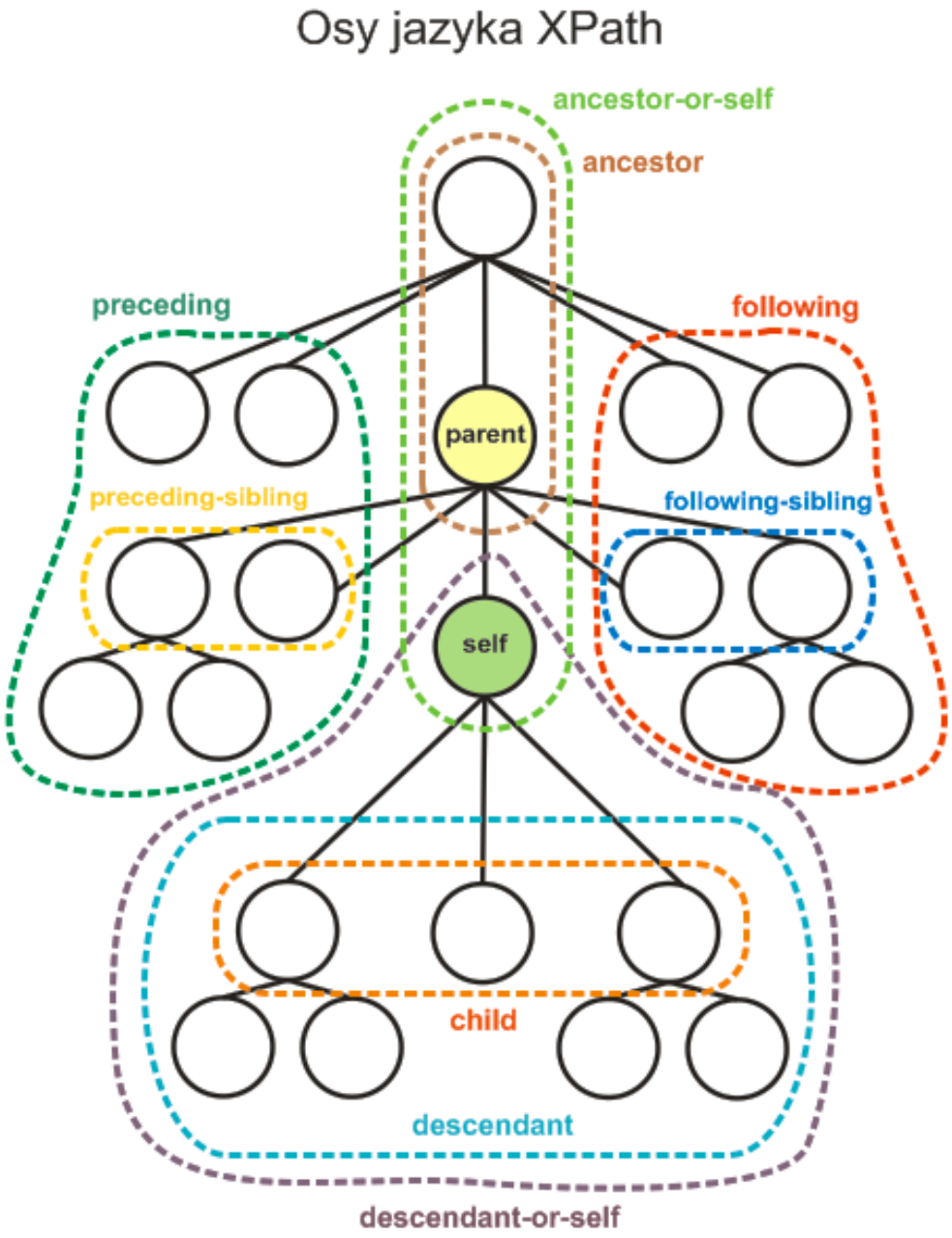
Kompletní zápis cesty musí obsahovat všechny tyto tři části

Ukázka obecného zápisu cesty:

```
/axis::node-test[predicate]..[predicate]/../axis::node-test[predicate]
```

### 3.1.4 Osy

Osy jsou důležitou součástí jazyka XPath. Osy jsou relace mezi uzly, určují množinu uzlů a také typ vztahu ke kontextovému uzlu, který je platný pro všechny uzly v této množině.




*Obrázek 5 – Osy jazyka XPath*

V XPath máme definováno celkem 13 typů os:

- *child* – přímí potomci kontextového uzlu
- *descendant* – přímí i nepřímí potomci kontextového uzlu
- *parent* – rodiče kontextového uzlu
- *ancestor* – předek kontextového uzlu
- *following-sibling* – následující sourozenci kontextového uzlu
- *preceding-sibling* – předcházející sourozenci kontextového uzlu
- *following* – všechny uzly, které se v dokumentu nacházejí za kontextovým uzlem (s výjimkou potomků)
- *preceding* – všechny uzly, které se v dokumentu nacházejí před kontextovým uzlem (s výjimkou předků)
- *attribute* – atributy kontextového uzlu
- *namespace* – všechny uzly reprezentující jmenné prostory dostupné kontextového uzlu
- *self* – kontextový uzel
- *descendant-or-self* – sjednocení os descendant a self
- *ancestor-or-self* – sjednocení os ancestor a self

### 3.1.5 Predikáty

Predikáty jsou určité logické podmínky, kterými můžeme omezit specifikovaný výběr uzlů. Predikát je booleovský výraz a zapisujeme jej do závorek '**[**' **a** '**]**', které se umisťují na konec kroku. Predikát musí být typu boolean nebo do něj být převeden

- **=, !=, >, <, >=, <=** - porovnávací predikáty
- **+** - sčítání
- **-** - odečítání
- ***** - násobení
- **mod** - zbytek po dělení
- **div** – dělení (nezaměňovat s /)

Uvedené příklady zápisu predikátů budou vztaženy k struktuře dokumentu z části 3.1.1 obrázek 4.

Výraz pro výběr posledního "*child*" elementu **polozka** aktuálního uzlu:

```
child::polozka[position()=last()]
```

Výraz pro výběr prvního "*child*" elementu **polozka** aktuálního uzlu:

```
child::polozka[position()=1]
```

V následujícím příkladě výraz vybere všechny elementy **polozka**, z elementu **polozky**, kde má element **cena** hodnotu 560:

```
objednavky/polozky/polozka[cena=560]
```

Následující výraz vybere druhý element **polozka**, který je potomkem aktuálního uzlu:

```
/descendant::polozka[position()=2]
```

Tento výraz vybere první dva "*child*" elementy **polozka** aktuálního uzlu:

```
child::polozka[position()<3]
```

Výraz pro výběr předposledního "*child*" elementu **polozka** aktuálního uzlu:

```
child::polozka[position()=last()-1]
```

### 3.1.6 Příklady použití os:

Zápis v příkladu znamená to samé jako zápis /X:

```
/child::X
```

Kontextový uzel:

```
self::node()
```

Všichni potomci každého elementu **polozky**:

```
//polozky/descendant::*
```

Všichni potomci **zakaznik** z každého elementu **objednavka**:

```
//objednavka/descendant::zakaznik
```

Všechny elementy, které jsou rodičem některého uzlu:

```
//perent::*
```

Rodiče všech elementů **polozka**:

```
//polozka/parent::*
```

Všechny elementy **polozky**, které jsou rodičem (mají děti):

```
//parent::polozky
```

Vše co následuje za první položkou objednávky:

```
/objednavka/polozky/polozka[1]/following::*
```

### 3.1.7 Funkce

V jazyce XPath máme na výběr ze speciálních funkcí, které nám umožňují splnit různou úroveň náročnosti uživatele jako např. práci s řetězci, výběr mezi sourozenci elementu atd. V následující tabulce naleznete funkce, které jsou nadefinovány na množinách uzlů, řetězcích, Booleovských hodnotách a číslech.

| **Jméno(*argument*)** | **Hodnota** |
|---|---|
| `count(množ. uzlů)` | velikost množiny uzlů |
| `sum(množ. uzlů)` | u množiny uzlů je řetězová hodnota každého uzlu převedena na číslo a poté se provede součet získaných čísel |
| `id(objekt)` | v případě, že je argumentem množina uzlů, je hodnota id použita na jednotlivé uzly v množině a ve výsledku dostaneme sjednocení všech jednotlivých výsledků, v opačném případě je argument převeden na řetězec vyjádřený jako hodnota atributu id elementů, které získáme ve výsledku |
| `name(množ. uzlů)` | jméno prvního uzlu dle uspořádání daného dokumentu |
| `position()` | pozice kontextového uzlu v množině uzlů (podle uspořádání dokumentu) |
| `ceiling(číslo)` | nejbližší vyšší celé číslo |
| `floor(číslo)` | nejbližší nižší celé číslo |
| `number(číslo)` | převádí objekt na číslo |
| `round(číslo)` | zaokrouhlení k nejbližšímu celému číslu |

*Tabulka 5 – Tabulka funkcí v jazyce XPath*

### 3.1.8 Zkrácený zápis výrazů

Pro některé osy můžeme využít zkráceného zápisu, který je přehlednější a zjednodušuje některé typy dotazů. Zkrácené notace jsou uvedeny níže v tabulce č.5:

| Úplný zápis | Zkrácený zápis |
|---|---|
| `child::` | |
| `attribute::` | `@` |
| `self::node()` | `.` |
| `self::node()/descendant-or-self::node()/child::X` | `.//X` |
| `parent::node()` | `..` |
| `parent::node()/child::X` | `../X` |
| `/descendant-or-self::node()/` | `//` |
| `/descendant-or-self::node()/child::X` | `//X` |
| `[position()=znak]` | `[znak]` |

*Tabulka 6 – Zkrácené notace os*

V praxi je naprostá většina dotazů založena na ose child, když žádaná osa není specifikována, tak se implicitně využije právě osa child.

Příklady určitých vybraných zkrácených zápisů, budeme vycházet ze struktury dokumentu na *Obrázku 4.*, který je uveden na začátku části 3.1.1 Datový model XPath:

Tyto dva výrazy vyberou všechny elementy **`polozka`**, které jsou potomky kontextového uzlu a mají atribut **`kod`** roven **`"123456789"`**:

```
1)  child:polozka[attribute::kod="123456789"]
2)  polozka[@kod="123456789"]
```

Následující dva výrazy použijeme v případě, že v cestě není uvedena osa a vyberou se jimi všechny elementy **`polozky`**, které jsou přímými potomky kontextového uzlu:

```
3)  child::polozky
4)  polozky
```

Těmito výrazy jsou vybrány elementy, jejichž jméno je **`polozka`**, které jsou přímým potomkem nějakého potomka kontextového uzlu (klidně i nepřímého):

```
5)  descendant-or-self::node()/child::polozka
6)  //polozka
```

Výrazy vyberou všechny elementy **`polozka`**, které jsou přímými potomky kontextového uzlu:

```
7)  self::node()/child::polozka
8)  ./polozka
```

Výrazy vyberou všechny elementy s daným názvem **`polozka`**, které jsou přímými potomky rodiče kontextového uzlu:

```
9)  parent::node()/child::polozka
10) ..//polozka
```

### 3.1.9 Testy uzlů

V množině, kterou získáme vyhodnocením osy, se následně vyhledávají uzly, u kterých se kladně vyhodnotí tzv. test uzlu. Typ daného uzlu můžeme určit následovně (*za pomlčkou je vždy napsané co vše je vybráno*):

- * – všechny elementy, atributy, nebo jmenné prostory (ovšem v závislosti na ose);
- *prefix:** – všechny atributy nebo elementy ležící ve jmenném prostoru, který je určený daným prefixem
- *jméno uzlu* – všechny atributy nebo elementy s daným jménem a určeným jmenným prostorem
- *prefix:jméno uzlu* – všechny atributy nebo elementy s daným jménem ležící ve jmenném prostoru, který je určený daným prefixem
- *node()* – všechny uzly
- *text()* – všechny textové uzly
- *comment()* – všechny komentáře
- *processing-instruction()* – všechny instrukce pro zpracování
- *processing-instruction(jméno instrukce)* – instrukce pro zpracování s určeným jménem instrukce.

### 3.1.10 Výběr elementů

Uvedené příklady budou vztaženy k struktuře dokumentu z části 3.1.1 obrázek 4.

Následující příklad vybere všechny elementy **cena** ze všech elementů **polozka** z kořenového elementu **objednavka**:

```
/objednavka/polozky/polozka/cena
```

Když cesta začíná jedním lomítkem "/", tak reprezentuje absolutní cestu k danému elementu. V případě, že cesta začíná dvěma lomítky "//", tak platí pro celý dokument.

Následující příklad vybere všechny elementy **polozka** v dokumentu:

```
//polozka
```

V případě, že vybíráme neznámý element, tak využíváme znaménka "*". V následujícím příkladě za pomoci výrazu vybereme všechny přímé potomky kontextového uzlu z elementu **polozky**:

```
/objednavka/polozky/*
```

V následujícím příkladě výraz vybere všechny elementy cena, které jsou "*vnoučata*" elementu **objednavka**:

```
/objednavka/*/cena
```

Tento výraz vybere všechny elementy cena, které mají dva předky:

```
/*/*/cena
```

Následující výraz vybere všechny elementy v celém dokumentu:

```
//*
```

Další element můžeme specifikovat pomocí hranatých závorek "[ ]". Následující příklad XPath výrazu vybere druhý "*child*" element **polozka** z elementu **polozky**:

```
objednavky/polozky/polozka[2]
```

V následujícím příkladě výraz vybere poslední "*child*" element **`polozka`** z elementu **`polozky`**:

```
objednavky/polozky/polozka[last()]
```

Zde je nutné upozornit na skutečnost, že k funkci **`last`** neexistuje opačná funkce **`first`**.

Další příklad výrazu vybere všechny elementy **`polozka`**, které obsahují element **`mnozstvi`**:

```
objednavka/polozky/polozka[mnozstvi]
```

V následujícím příkladě výraz vybere všechny elementy **`polozka`**, z elementu **`polozky`**, kde má element **`cena`** hodnotu 560:

```
objednavky/polozky/polozka[cena=560]
```

### 3.1.11 Výběr atributů

V XPath specifikujeme všechny atributy prefixem "@". Uvedené příklady budou opět vztaženy k struktuře dokumentu z části 3.1.1 obrázek 4.

Výraz v následujícím příkladě vybere všechny atributy s názvem **`cislo`**:

```
//@cislo
```

Výraz vybere všechny elementy **`polozka`**, ve kterých je obsažen atribut **`barva`**:

```
//polozka[@barva]
```

Výraz vybere všechny elementy **`mnozstvi`** a **`cena`**, které jsou obsaženy v dokumentu:

```
//mnozstvi | //cena
```

Následující výraz vybere všechny elementy **`polozka`**, které obsahují nějaký atribut:

```
//polozka[@*]
```

Výraz vybere všechny elementy **`polozka`**, které obsahují atribut **`kod`** s hodnotou "**`123456789`**":

```
//polozka[@kod="123456789"]
```

# 4 XML V ORACLE DATABÁZI

V Oracle XML DB je XML dokument považován za přirozený datový typ této databáze. XML datový model obsahuje zároveň strukturovaná data a nestrukturovaný obsah. V aplikacích se využívá standardní XML a SQL pro vytváření složitých XML dokumentů z SQL dotazů a XML dokumentů.

- Oracle XML DB se dá definovat jako množina zabudovaných, vysoce výkonných vyhledávácích a úložných technologií, které jsou zaměřeny na XML.
- Oracle XML DB plně podporuje W3C XML datový model.
- Spojuje všechny výhody technologie XML a zároveň relačních databází.
- Dualita XML/SQL. Lze provádět XML operace nad SQL daty a také SQL operace nad XML daty.
- Oracle XML DB umožňuje ukládat a spravovat strukturovaná i nestrukturovaná data.
- Podporuje XPath a také SQL/XML

## 4.1 Úvod XML v Oracle

V Oracle máme možnosti dokumentového nebo datově orientovaného přístupu. V případě, že se chceme na XML dívat jako na dokumentově zaměřené dokumenty tedy např. nějaké články, nebo knihy atd. pak můžeme v Oracle XML DB využít *Repositář* (úložiště). Toto úložiště je přístupné pomocí standardních protokolů a pomocí SQL.

V případě, že máme XML dokumenty, které jsou datově zaměřené (strukturované), tedy např. adresy, objednávky, faktury, data o zaměstnancích atd. pak pro uložení v XML DB využijeme *XML Type*. XML Type je pro Oracle XML DB přirozeným datovým typem a poskytuje podporu pro jazyk XML Schema, XPath atd.

### 4.1.1 Vývojový nástroj Oracle Developer

Tato aplikace sloužící pro vývoj obsahuje základní nástroje pro práci s XML daty jako je např. čtená, manipulace, transformace a zobrazení jednotlivých XML dokumentů. Tyto nástroje jsou využitelné pro programovací jazyky C, C++ a Java. Díky tomuto nástroji aplikaci se dají efektivním způsobem vytvářet aplikace, které využívají pro svou funkčnost databází Oracle.

### 4.1.2 Hlavní rysy Oracle XML DB

V této části příručky jsou v jednotlivých bodech vypsány některé z hlavních rysů Oracle XML databáze:

- XML Type
  - Vyhledávání pomocí XPath
  - Rychlý přístup k obsahu XML dat za pomocí XML indexů
  - Transformace XSL pro XML Type
  - XML pohledy
  - SQLX operátory
  - Možnost aktualizace úseků XML dokumentu pomocí XPath
- XML Schema
  - Věrná reprodukce DOM a dokumentu včetně bílých znaků
  - Plně podporuje standard W3C (XML Schema)
- XML DB Repositář
  - ACL – zabezpečení přístupu k datům
  - Správa jednotlivých verzí
  - Vyhledávání v XML Repositáři
- Přístupy k datům za pomoci FTP, WebDAV, HTTP
- Přístupové XML API:
  - C, PL/SQL, Java
  - Serverlets/JSP (JDBC)
  - XSL/XSPs (dokumenty XML + DOM)
- XML/SQL dualita

### 4.1.3 XMLType

Způsobů jak uložit XML obsah do Oracle Databáze existuje celá řada. V případě, že bychom nevyužili Oracle XML DB, tak bychom uložení XML dokumentu mohli realizovat použitím:

- Vývojářská XML výbava Oracle XDK,proto, abychom mohli rozebrat XML dokument mimo databázi Oracle a poté mohli využít extrahovaná data popořadě do databázových tabulek.
- BLOB – Binary Large Object, CLOB – Character Large Object, BFILE – Binary File nebo VARCHAR.

Oba výše uvedené příklady jsou pro práci s XML obsahem nedostačující. Pří zavedení *XMLType* se vytvořili nové techniky zacházení s XML daty, které usnadňují práci s daty a zachovávají stálost dat v databázi.

XML dokumenty je možné ukládat jako XMLType slupce v tabulce nebo přímo vytvářet tabulky, jejichž obsahem je XML.

Dokument, který je přístupný pomocí jazyka SQL může být reprezentován XMLType nebo můžeme použít metody, které se využívají při vytváření, získávání a nebo indexaci dat, která jsou uložena v databázi.

SQL funkce, které pracují nad XML Typem a v podstatě nad celým dokumentem:

Funkce, která zjišťuje přítomnost uzlu v XML dokumentu:

```
existNode
```

Funkce, která extrahuje XML dokument:

```
extract
```

**Způsoby ukládání:**

- LOB – Large Object zachovává přesnou reprezentaci daného XML dokumentu a je u něj velká přizpůsobivost změně schématu.

- Objektově-relační reprezentace – Při tomto způsobu ukládání je omezená přizpůsobivost změně schématu, vypouštějí se bílé znaky a komentáře, je u něj vysoký výkon DML a dobrá přístupnost existujících SQL rysů jako jsou indexy, integritní omezení atd.

### 4.1.4 XML Repositář

XML DC Repository se využívá při obsahově orientovaném přístupu k databázi a je to v podstatě jakýsi sklad pro správu a přístup k XML dokumentům. XML DB Repository se skládá z částí:

- ACL - seznam a správa přístupových práv včetně vlastních práv
- Správa složek
- WebDAV a FTP protokol využívaný pro přístup ke složkám
- Využití jazyka SQL při vyhledávání v XML Repositáři
- API pro práci s XML Repositářem
- Přístup za pomoci Java servletu využívaný k manipulaci s objekty

### 4.1.5 Využívání databázových možností

Dříve se bez podpory XML v databázích XML data ukládala jako nestrukturovaná data CLOB nebo v souborech XML, kde nebylo možné využít některé z klíčových vlastností databází jako jsou:

- *indexace a vyhledávání* – V Oracle XML DB je možné efektivně a strukturovaně vyhledávat nad XML daty.
- *aktualizace a transakční zpracování*
- *možnost více pohledů na data* – Kombinování dat a relační pohledy.
- *výkonnost a škálovatelnost* – V oracle XML DB je možné rychlé ukládání, dotazování a aktualizace XML dat.

### 4.1.6 Možnosti XML v Oracle DB

- *nezávislost na uložení* – **`XMLType`** nám umožňuje psát aplikace bez toho, abychom přesně věděli, kde jsou data uložena, v jakém jsou data formátu, v jaké jsou tabulce, jaké jsou vztahy mezi těmito tabulkami. Dovoluje administrátorovi, aby mapoval strukturovaná data na fyzické tabulky a sloupce.
- *strukturální nezávislost* – Model obsahu nelze jednoduše zachytit pomocí tabulek a sloupců.
- *jednoduchost prezentace* – XML je podporováno v prohlížečích, dále v desktopových aplikacích. U relačních dat je potřeba naprogramovat vrstvy pro zpřístupnění dat uživateli. V Oracle XML DB je možné ukládat i vyvolávat data jako XML.
- *jednoduchost výměny* – XML umožňuje velmi jednoduchou výměnu dat bez toho, aby se data překládala, protože při překládání dat může dojít ke ztrátě informace.

### 4.1.7 SQL Operátory pro práci s XML

Následující operátor vyhodnocuje, zda dokument obsahuje uzel, který je definován XPath výrazem, jako výsledek vrací 1 a 0:

```
existsNode()
```

Operátor vrátí hodnotu textového uzlu nebo atributu, který je asociován s XPath výrazem. V tomto případě je datový typ převeden na odpovídající datový typ v SQL:

```
extractValue()
```

Tento operátor využijeme v případě, když nám XPath vrátí množinu uzlů typu XMLType, jak již jsme se dříve zmínili v tomto případě může být výsledkem dokumentu část XML dokumentu nebo celý XML dokument:

```
extract()
```

Následující operátor umožňuje návrat celého XML dokumentu:

```
value()
```

### 4.1.8 SQL/XML

- SQL/XML je jakési rozšíření stávajícího jazyka SQL tak, že je mu umožněno pracovat s XML daty.
    - Je to nový vestavěný datový typ
    - Dotazování XML dat
- SQL/XML není SQLXML
- XML hodnota je v SQL/XML klíčové slovo
    - Je buď množinou XML elementů, nebo XML elementem
- Sémantika XML hodnoty
    - Vychází z SQL/XML:2003 modelu Infoset
    - V následném vývoji je SQL/XML založeno na datovém modelu jazyka XQuery

**Funkce pro publikaci XML dat:**

Za pomoci těchto funkcí se z SQL výrazů generují XML hodnoty.

Vytvoření XML elementu:

```
XMLELEMENT
```

Vytvoření XML atributů:

```
XMLATTRIBUTES
```

Vytvoření XML elementů z názvu sloupců:

```
XMLFOREST
```

Ze seznamu vytvořených výrazů vytvoří jednu XML hodnotu ve formě posloupnosti elementů:

```
XMLCONCAT
```

Ze seznamu řádků vytvoří posloupnost elementů, každý řádek obsahuje jednu XML hodnotu:

```
XMLAGG
```

## 4.2 Příklady v Oracle Databázi

V této kapitole příručky jsou uvedeny praktické příklady, které byly přímo testovány v databázi Oracle a stručně je popsáno, co které části kódů dělají. Je zde uvedeno jak vytvořit a zaregistrovat schéma, jak vytvořit XML tabulky a nahrát data do tabulky, jak se dotazovat nad vloženými daty atd. Toto byl jen stručný výčet následujících příkladů. Tato kapitola je ukázkou toho, jak zužitkovat znalosti práce s XML přímo v databázi Oracle.

### 4.2.1 Načítání XML dokumentů do Oracle XML DB

Proto, abychom mohli v Oracle XML DB načítat různé obsahy XML dokumentů, je nutné mít nadefinovaný adresář, ze kterého se soubory budou načítat.

Nadefinování adresáře obecně:

```
CREATE DIRECTORY xmldir AS path_to_folder_containing_XML_file';
```

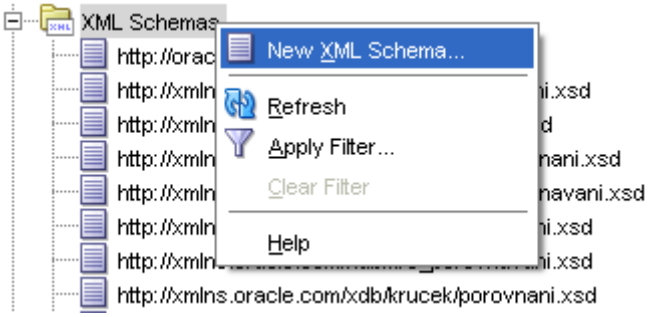
Příklad nadefinování adresáře, který dále budeme v příručce v dalších příkladech využívat:

```
CREATE DIRECTORY Soubory_XML AS cesta_do_adresáře_s_daty';
```

Tento adresář musíme nadefinovat předtím, než budeme do tabulky s datovým sloupcem XML Type vkládat data z nějakého XML souboru. Adresář musí být na serveru.

### 4.2.2 Vytvoření XML Schématu

Vytvoření XML Schématu v je databázi Oracle je poměrně jednoduché. V pravém sloupci Connections klikneme po připojení k databázi pravým tlačítkem myši na XML Schemas a dáme vytvořit nové schéma, jak je znázorněno na následujícím obrázku:



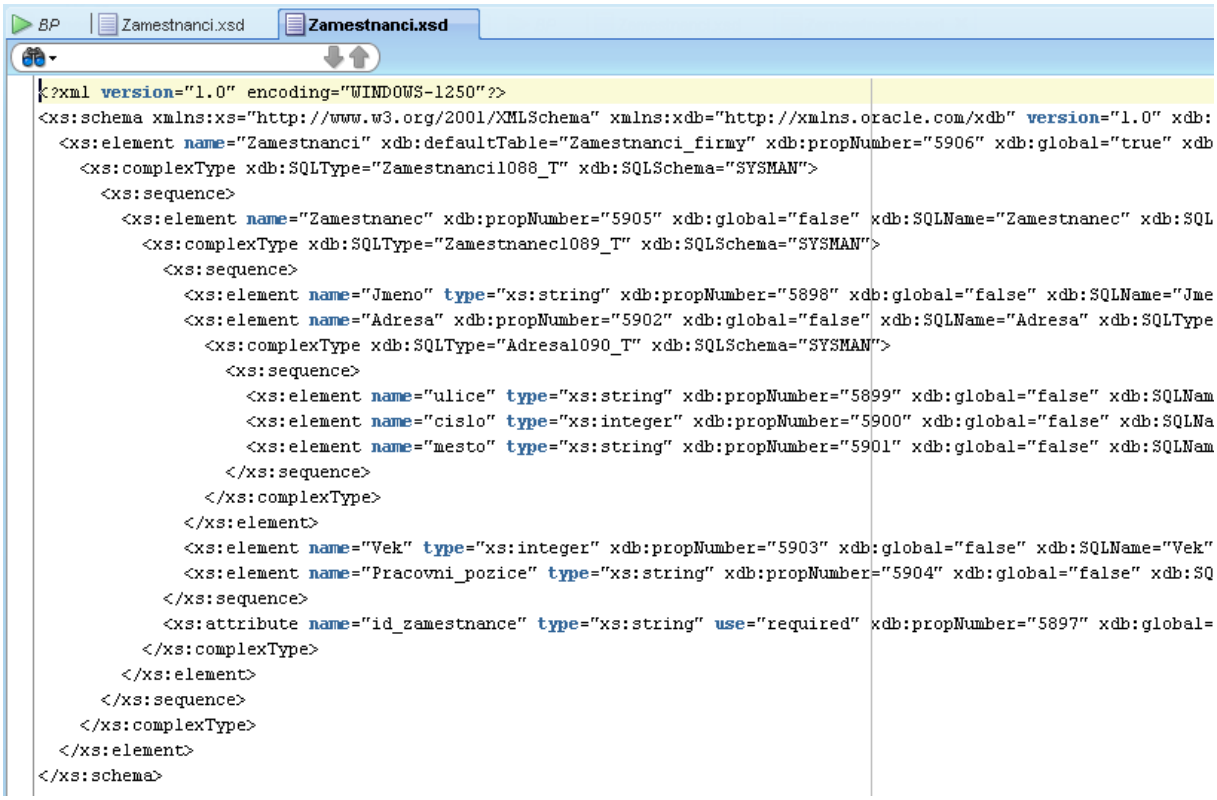

*Obrázek 6 – Ukázka vytvoření nového schéma*

Vyskočí nám prázdný list, do kterého napíšeme své schéma a následně uložíme. V následujícím příkladu je vytvořené schéma “Zamestnanci.xsd“ s nadefinovanou výchozí tabulkou "Zamestnanci_firmy".

Zamestnanci.xsd:

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns:xdb="http://xmlns.oracle.com/xdb"
           version="1.0">
<xs:element name="Zamestnanci" xdb:defaultTable="Zamestnanci_firmy">
   <xs:complexType>
      <xs:sequence>
         <xs:element name="Zamestnanec">
            <xs:complexType>
               <xs:sequence>
                  <xs:element name="Jmeno" type="xs:string"/>
                  <xs:element name="Adresa">
                     <xs:complexType>
                        <xs:sequence>
                           <xs:element name="ulice" type="xs:string"/>
                           <xs:element name="cislo" type="xs:integer"/>
                           <xs:element name="mesto" type="xs:string"/>
                        </xs:sequence>
                     </xs:complexType>
                  </xs:element>
                  <xs:element name="Vek" type="xs:integer"/>
                  <xs:element name="Pracovni_pozice" type="xs:string"/>
               </xs:sequence>
              <xs:attribute name="id_zamestnance" type="xs:string"
use="required"/>
          </xs:complexType>
         </xs:element>
      </xs:sequence>
   </xs:complexType>
</xs:element>
</xs:schema>
```

Při následném otevření uloženého schématu v databázi si můžeme povšimnout, že databáze si po uložení schéma přizpůsobí a přidá např. vlastníka schématu, tedy toho uživatele, který schéma vytvořil, v našem případě je to "**SYSMAN**", dále např. přidá informaci, zda jsou elementy nadefinované globálně nebo ne, také k datovým typům XML si databáze přiřadí datové typy, které jsou využívány v databázích tedy pro XML datový typ **"xs:string"** přidá analogický databázový datový typ **"VARCHAR2"**. Na následujícím obrázku můžete vidět krátkou ukázku přizpůsobeného schématu.




*Obrázek 7 – Ukázka části schématu uloženého v DB Oracle*

To bylo jen pro ukázku, jak si databáze Oracle přizpůsobí schéma.

### 4.2.4 Vytváření tabulek s připojeným XML schématem

#### Vytvoření tabulky s jedním sloupcem typu XML Type

V následujícím příkladě se nám vytvoří tabulka **`Zamestnanci_firmy3`** s jedním sloupcem XMLType, který se bude jmenovat **object_value**. Když chceme takto utvořit tabulku, tak nesmíme zapomenout také napsat název kořenového elementu v daném schématu (**`Zamestnanci`**) a název schéma, které k sloupci XMLType připojujeme v tomto případě "Zamestnanci.xsd", záleží na tom, jak máme pojmenované schéma v databázi, takže to může být i například **`xmlschema "http://xmlns.oracle.com/xdb/Zamestnanci.xsd"`** kde v uvozovkách je název schématu.

```
create table Zamestnanci_firmy3 of XMLType
xmlschema "Zamestnanci.xsd"
ELEMENT "Zamestnanci"
```

#### Nahrání dat do vytvořené tabulky

Do vytvořené tabulky z předchozího příkladu nahrajeme data ze souboru **`Zamestnanci.xml`**:

```
Insert into Zamestnanci_firmy3 values
(XMLType(bfilename('Soubory_XML', 'Zamestnanci.xml'),
nls_charset_id('AL32UTF8')));
```

#### Výpis dat vložených do tabulky

Za pomoci XPath dotazu si můžeme vypsat všechny XML vložená data, která jsme do sloupce object_value nahráli v předchozím příkladě:

```
select extract(object_value,'/*') from Zamestnanci_firmy3;
```

Dotazování se nad XML daty se věnuje celá kapitola s příklady 4.3.

#### Vytvoření tabulky s více sloupci a jedním XML Type sloupcem

```
CREATE TABLE Zamestnanci_firmy4(
XMLID NUMBER,
Zamestnanci XMLType,
CONSTRAINT Zamestnanci_firmy4_PK PRIMARY KEY (XMLID)
)
XMLType COLUMN Zamestnanci
STORE AS OBJECT RELATIONAL
XMLSCHEMA "Zamestnanci.xsd" ELEMENT "Zamestnanci";
COMMIT;
```

Předchozí příklad vytvoří tabulku, ve které budou dva sloupce, první sloupec bude sloupec datového typu NUMBER s názvem **`XMLID`** a druhý sloupec bude datového tymu XMLType s názvem **`Zamestnanci`**. K sloupci **`Zamestnanci`** připojí schéma "Zamestnanci.xsd".

### 4.2.5 Registrace a mazání XML Schéma obecně

Hlavní argumenty pro registraci XML schéma **`DBMS_XMLSCHEMA.registerSchema`** jsou:

**`schemaURL`** – je identifikátor, který funguje při registraci schématu pouze v rámci databází Oracle. Zapisuje se ve formě URL, ale není to podmínkou. XML schéma URL se využívá Oraclem například pro identifikování dokumentů.

**`schemaDoc`** – Umístění našeho XML schéma.

**`CSID`** – slouží pro specifikaci kódování, ve kterém má být registrace XML schéma uložena (ve většině případů využíváme kódování UTF8).

Když ve schématu máme nadefinovanou výchozí tabulku, tak ji nemusíme vytvářet, protože se vytvoří sama (tato tabulka bude mít jeden sloupec datového typu XMLType, který bude pojmenovaný **object_value**).

```
BEGIN
DBMS_XMLSCHEMA.registerSchema(
SCHEMAURL =>
'http://xmlns.oracle.com/xdb/documentation/MojeSchema.xsd',
SCHEMADOC => bfilename('XMLDIR','MojeSchema.xsd'),
CSID => nls_charset_id('AL32UTF8'));
END;
/
```

Registrace XML schématu jako lokálního XML schéma provedeme přidáním **`LOCAL => TRUE:`**

```
BEGIN
DBMS_XMLSCHEMA.registerSchema(
SCHEMAURL =>
'http://xmlns.oracle.com/xdb/documentation/MojeSchema.xsd',
SCHEMADOC => bfilename('XMLDIR','MojeSchema.xsd'),
LOCAL => TRUE,
CSID => nls_charset_id('AL32UTF8'));
END;
/
```

Registrace jako globální XML schéma:

```
LOCAL => FALSE
```

**Odstranění registrace XML schématu obecně**

Stejně jako máme možnost zaregistrovat naše XML schéma pomocí **`DBMS_XMLSCHEMA.registerSchema`**, tak máme také možnost jej smazat pomocí **`XMLSCHEMA.DELETESCHEMA`**. Jak smažeme schéma vetvořené v předchozím příkladě se můžete podívat na následující ukázce:

```
BEGIN
DBMS_XMLSCHEMA.deleteSchema(
SCHEMAURL => 'http://xmlns.oracle.com/xdb/documentation/MojeSchema.xsd',
DELETE_OPTION => dbms_xmlschema.DELETE_CASCADE_FORCE);
END;
/
```

### 4.2.6 Registrace a mazání Schématu z příkladu 3.2.2

Pokud máme definovanou ve schématu výchozí tabulku, tak už nemusíme následně tabulku vytvářet, protože se nám při registraci schématu tato tabulka vytvoří sama. V případě schématu, které jsme v tomto příkladě vytvořili se nám vytvoří tabulka “Zamestnanci_firmy“ (**`xdb:defaultTable="Zamestnanci_firmy"`**), která bude mít jeden sloupec datového typu XMLType. V případě, že výchozí tabulku nemáme nadefinovanou, tak se při registraci nevytvoří.

```
BEGIN
  DBMS_XMLSCHEMA.registerSchema(
    SCHEMAURL => 'Zamestnanci.xsd',
    SCHEMADOC => bfilename('Soubory_XML','Zamestnanci.xsd'),
    CSID => nls_charset_id('AL32UTF8'));
END;
/
```

Při registraci schématu si též musíme dávat pozor, jestli máme správně nadefinovanou cestu do adresáře, kde máme schéma uloženo, to uděláme v záložce “Directories“. V tomto případě máme při registraci schématu cestu nadefinovanou pod **`'Soubory_XML'`**.

Stejně jednoduše, jako jsme schéma zaregistrovali, tak jej můžeme také smazat:

```
BEGIN
  DBMS_XMLSCHEMA.deleteSchema(
  SCHEMAURL => 'Zamestnanci.xsd',
  DELETE_OPTION => dbms_xmlschema.DELETE_CASCADE_FORCE);
END;
/
```

### 4.2.7 Vložení dat do XML Tabulky ze souboru

Následující příklad vytvoří tabulku "**Zamestnanci_firmy**" a v ní sloupec "**Zamestnanci_1**", který je definovaný jako XMLType a naplní jej daty z dokumentu "Zamestnanci.xml":

```
CREATE TABLE Zamestnanci_firmy (id NUMBER,
                                Zamestnanci_1 XMLType);
INSERT INTO Zamestnanci_firmy (id, Zamestnanci_1)
VALUES (1,XMLType(bfilename('Soubory_XML ',
'Zamestnanci.xml'),nls_charset_id('AL32UTF8')));
```

Nahrávaný XML dokument "Zamestnanci.xml" jehož struktura je popsána podle již vytvořeného schématu "Zamestnaci.xsd":

```
<?xml version="1.0"?>
<Zamestnanci xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:noNamespaceSchemaLocation="Zamestnanci.xsd">
  <Zamestnanec id_zamestnance="z1">
    <Jmeno>Josef Novák</Jmeno>
    <Adresa>
      <ulice>Dlouhá</ulice>
      <cislo>13</cislo>
      <mesto>Zlín</mesto>
    </Adresa>
    <Vek>45</Vek>
    <Pracovni_pozice>Technik</Pracovni_pozice>
  </Zamestnanec>
  <Zamestnanec id_zamestnance="z2">
    <Jmeno>Pavel Kobliha</Jmeno>
    <Adresa>
      <ulice>Krátká</ulice>
      <cislo>47</cislo>
      <mesto>Olomouc</mesto>
    </Adresa>
    <Vek>38</Vek>
    <Pracovni_pozice>Vývojář</Pracovni_pozice>
  </Zamestnanec>
  <Zamestnanec id_zamestnance="z3">
    <Jmeno>Jana Černá</Jmeno>
    <Adresa>
      <ulice>Kratochvílova</ulice>
```

```
      <cislo>26</cislo>
      <mesto>Brno</mesto>
    </Adresa>
    <Vek>49</Vek>
    <Pracovni_pozice>Personalistka</Pracovni_pozice>
  </Zamestnanec>
</Zamestnanci>
```

### 4.2.8 Vložení dat do XML Tabulky přímo

Následující kód vytvoří tabulku, do které se přímo nahraje XML dokument. Takovouto tvorbu tabulky můžeme využít v případě, že nemáme mnoho dat a hlavně neměli jsme při registraci schématu nadefinovanou výchozí tabulku, která se vytváří sama. V následujícím příkladě také využíváme již vytvořené schéma schéma “Zamestnanci.xsd“, které popisuje XML dokument, který do tabulky nahráváme. Ve vytvořené tabulce budou dva sloupce a to sloupec ID datového typu NUMBER a sloupec datového typu XMLType s názvem “Zamestnanci_1“.

```
CREATE TABLE Zamestnanci_firmy (id NUMBER,
                                Zamestnanci_1 XMLType);
INSERT INTO Zamestnanci_firmy (id, Zamestnanci_1)

VALUES (1,XMLTYPE ('
<Zamestnanci>
  <Zamestnanec id_zamestnance="z1">
    <Jmeno>Josef Novák</Jmeno>
    <Adresa>
      <ulice>Dlouhá</ulice>
      <cislo>13</cislo>
      <mesto>Zlín</mesto>
    </Adresa>
    <Vek>45</Vek>
    <Pracovni_pozice>Technik</Pracovni_pozice>
  </Zamestnanec>
  <Zamestnanec id_zamestnance="z2">
    <Jmeno>Pavel Kobliha</Jmeno>
    <Adresa>
      <ulice>Krátká</ulice>
      <cislo>47</cislo>
      <mesto>Olomouc</mesto>
    </Adresa>
    <Vek>38</Vek>
```

```
    <Pracovni_pozice>Vývojář</Pracovni_pozice>
  </Zamestnanec>
  <Zamestnanec id_zamestnance="z3">
    <Jmeno>Jana Černá</Jmeno>
    <Adresa>
      <ulice>Kratochvílova</ulice>
      <cislo>26</cislo>
      <mesto>Brno</mesto>
    </Adresa>
    <Vek>49</Vek>
    <Pracovni_pozice>Personalistka</Pracovni_pozice>
  </Zamestnanec>
</Zamestnanci>    '));
COMMIT;
```

## 4.3 Příklady dotazů nad XML daty

Pojmenování XML Type sloupce “zamestnanci_1“ nám v příkladech pomůže k rozlišení toho, že se jedná skutečně o tento sloupec, abychom neměli napsané všude jen “Zamestnanci“.

***1)*** Následující SQL dotaz vypíše vše, co jsme uložili do prvního řádku sloupce typu XMLType s názvem “zamestnanci_1“ tabulky “**zamestnanci_firmy**“, takže výsledek bude celý obsah dokumentu “Zamestnanci.xml“ z předchozího příkladu, který jsme do tabulky vložili.

```
DOTAZ:

SELECT zamestnanci_1 AS RESULT FROM zamestnanci_firmy WHERE ROWNUM < 2;
```

***2)*** Následující dotaz vrátí všechny elementy **<jmeno>** v dokumentu a jejich obsah. Takže jména všech zaměstnanců firmy:

```
DOTAZ:
SELECT extract(zamestnanci_1,'/Zamestnanci/Zamestnanec/Jmeno')
FROM zamestnanci_firmy

VÝSLEDEK:
<Jmeno>Josef Novák</Jmeno>
<Jmeno>Pavel Kobliha</Jmeno>
<Jmeno>Jana černá</Jmeno>
```

***3)*** Tento dotaz je v podstatě stejný jako v předchozím příkladě pouze v zjednodušeném zápisu. Dotaz vrátí všechny elementy **<jmeno>** v dokumentu a jejich obsah. Takže jména všech zaměstnanců firmy:

```
DOTAZ:
SELECT extract(zamestnanci_1,'//Jmeno')
FROM zamestnanci_firmy

VÝSLEDEK:
<Jmeno>Josef Novák</Jmeno>
<Jmeno>Pavel Kobliha</Jmeno>
<Jmeno>Jana černá</Jmeno>
```

***4)*** Dotaz vrátí jména a pracovní pozici všech zaměstnanců ve firmě:

```
DOTAZ:
SELECT extract(zamestnanci_1, '//Jmeno | //Pracovni_pozice')
FROM zamestnanci_firmy;

VÝSLEDEK:
<Jmeno>Josef Novák</Jmeno>
<Pracovni_pozice>Technik</Pracovni_pozice>
<Jmeno>Pavel Kobliha</Jmeno>
<Pracovni_pozice>Vývojář</Pracovni_pozice>
<Jmeno>Jana černá</Jmeno>
<Pracovni_pozice>Personalistka</Pracovni_pozice>
```

***5)*** Dotaz vrátí adresu v pořadí druhého zaměstnance firmy. Vyberou se všichni přímí potomci uzlu Adresa, což je Ulice, Číslo a Město.

```
DOTAZ:
SELECT extract(zamestnanci_1,'//Zamestnanec[2]/Adresa/*')
FROM zamestnanci_firmy

VÝSLEDEK:
<ulice>Krátká</ulice>
<cislo>47</cislo>
<mesto>Olomouc</mesto>
```

***6)*** Kdybychom chtěli z adresu z předchozího příkladu vypsat i s elementem Adresa stačilo by dotaz upravit následovně. Dotaz vrátí adresu v pořadí druhého zaměstnance firmy. Vyberou se všechny elementy v elementu Adresa, což je Ulice, Číslo a Město včetně jej samotného.

```
DOTAZ:
SELECT extract(zamestnanci_1,'//Zamestnanec[2]/Adresa')
FROM zamestnanci_firmy

VÝSLEDEK:
<Adresa>
  <ulice>Krátká</ulice>
  <cislo>47</cislo>
  <mesto>Olomouc</mesto>
</Adresa>
```

***7)*** Následující dotaz vypíše poslední“*child*“ element **`zamestnanec`** z elementu **`zamestnanci`**:

```
DOTAZ:
SELECT extract(zamestnanci_1,'/*/Zamestnanec[last()]')
FROM zamestnanci_firmy

VÝSLEDEK:
<Zamestnanec id_zamestnance="z3">
  <Jmeno>Jana černá</Jmeno>
  <Adresa>
    <ulice>Kratochvílova</ulice>
    <cislo>26</cislo>
    <mesto>Brno</mesto>
  </Adresa>
  <Vek>49</Vek>
  <Pracovni_pozice>Personalistka</Pracovni_pozice>
</Zamestnanec>
```

***8)*** Následující dotaz vypíše předposlední“*child*“ element **`zamestnanec`** z elementu **`zamestnanci`**:

```
DOTAZ:
SELECT extract(zamestnanci_1,'/*/Zamestnanec[last()-1]')
FROM zamestnanci_firmy

VÝSLEDEK:
<Zamestnanec id_zamestnance="z2">
  <Jmeno>Pavel Kobliha</Jmeno>
  <Adresa>
    <ulice>Krátká</ulice>
    <cislo>47</cislo>
    <mesto>Olomouc</mesto>
  </Adresa>
  <Vek>38</Vek>
  <Pracovni_pozice>Vývojář</Pracovni_pozice>
</Zamestnanec>
```

***9)*** Následující příklad vypíše v pořadí druhý element **`zamestnanec`** v elementu **`zamestnanci`**, takže vrátí všechny údaje o druhém zaměstnanci:

```
DOTAZ:
SELECT extract(zamestnanci_1,'/*/*[2]')
FROM zamestnanci_firmy

VÝSLEDEK:
<Zamestnanec id_zamestnance="z2">
  <Jmeno>Pavel Kobliha</Jmeno>
  <Adresa>
    <ulice>Krátká</ulice>
    <cislo>47</cislo>
    <mesto>Olomouc</mesto>
  </Adresa>
  <Vek>38</Vek>
  <Pracovni_pozice>Vývojář</Pracovni_pozice>
</Zamestnanec>
```

***10)*** Dotaz vybere všechny elementy a vypíše je, takže výsledkem tohoto dotazu bude vložený XML dokument:

```
DOTAZ:
SELECT extract(zamestnanci_1,'//*')
FROM zamestnanci_firmy
```

***11)*** Dotaz vypíše obsahy všech elementů 4. úrovně, na které se nachází adresa zaměstnanců u zaměstnance s atributem **`id_zamestnance="z3"`**:

```
DOTAZ:
SELECT extract(zamestnanci_1, '/*/Zamestnanec[@*="z3"]/*/*/text()')
FROM zamestnanci_firmy;

VÝSLEDEK:
Kratochvílova
26
Brno
```

**`text()-`** můžeme využít také jako omezující podmínku, kdybychom třeba v případě jména napsali v nějakém příkladě **`Jmeno[text()="Pavel Kobliha"]`**

***12)*** Dotaz vrátí pracovní pozice všech zaměstnanců ve firmě:

```
DOTAZ:
SELECT extract(zamestnanci_1, '/*/*/Pracovni_pozice')
FROM zamestnanci_firmy;

VÝSLEDEK:
<Pracovni_pozice>Technik</Pracovni_pozice>
<Pracovni_pozice>Vývojář</Pracovni_pozice>
<Pracovni_pozice>Personalistka</Pracovni_pozice>
```

***13)*** Následující dotaz vrátí jméno zaměstnance firmy, jehož **id_zamestnance** je “z1“:

```
DOTAZ:
SELECT extractValue(zamestnanci_1,
'/Zamestnanci/Zamestnanec[@id_zamestnance="z1"]/Jmeno')
FROM zamestnanci_firmy;

VÝSLEDEK:
Josef Novák
```

***14)*** Následující dotaz vrátí pracovní pozici zaměstnance firmy, jehož atribut **id_zamestnance** je “z2“:

```
DOTAZ:
SELECT extractValue(zamestnanci_1,
'/Zamestnanci/Zamestnanec[@id_zamestnance="z2"]/Pracovni_pozice')
FROM zamestnanci_firmy;

VÝSLEDEK:
Vývojář
```

***15)*** Pro ukázku zjednodušený dotaz z předchozího příkladu, který vrátí pracovní pozici zaměstnance s **id_zamestnance =“z2“**:

```
DOTAZ:
SELECT extractValue(zamestnanci_1,
'/*/*[@id_zamestnance="z2"]/Pracovni_pozice')
FROM zamestnanci_firmy;

VÝSLEDEK:
Vývojář
```

***16)*** Předchozí příklad lze ještě více zjednodušit tím, že si vypíšeme pracovní pozici zaměstnance s **`id_zamestnance="z2"`** tak, že hledáme atribut elementu s hodnotou "z2", který má 1 předka (element **`zamestnanci`**). To můžeme využít v případě, když víme, že se jedná o opravdu jedinečný atribut:

```
DOTAZ:
SELECT extractValue(zamestnanci_1, '/*/*[@*="z2"]/Pracovni_pozice')
FROM zamestnanci_firmy;

VÝSLEDEK:
Vývojář
```

***17)*** Tento dotaz vrátí města, ve kterých žijí všichni zaměstnanci, pokud pracuje ve firmě zaměstnanec na pozici "Vývojář". Jestli na dané pozici ve firmě nikdo nepracuje, tak se města nevypíšou:

```
DOTAZ:
SELECT extract(z.zamestnanci_1, '/Zamestnanci/Zamestnanec/Adresa/mesto')
FROM zamestnanci_firmy z
WHERE existsNode(z.zamestnanci_1,
'/Zamestnanci/Zamestnanec[Pracovni_pozice="Vývojář"]')= 1;

VÝSLEDEK:
<mesto>Zlín</mesto>
<mesto>Olomouc</mesto>
<mesto>Brno</mesto>
```

***18)*** Dotaz z předchozího příkladu bychom mohli utvořit ve zkráceném zápisu. Dotaz vybere všechny elementy "**`mesto`**", které mají tři předky (/Zamestnanci/Zamestnanec/Adresa/) ovšem opět pouze v případě, že pracuje ve firmě nějaký "Vývojář".

```
DOTAZ:
SELECT extract(z.zamestnanci_1, '/*/*/*/mesto')
FROM zamestnanci_firmy z
WHERE existsNode(z.zamestnanci_1,
'/Zamestnanci/Zamestnanec[Pracovni_pozice="Vývojář"]')= 1;

VÝSLEDEK:
<mesto>Zlín</mesto>
<mesto>Olomouc</mesto>
<mesto>Brno</mesto>
```

***19)*** Následující dotaz testuje existenci uzlu 2. úrovně, jestliže uzel existuje vrací 1, jestliže ne, tak vrátí 0:

```
DOTAZ:
SELECT existsNode(zamestnanci_1, '/*/*')
FROM zamestnanci_firmy;

VÝSLEDEK:
1
```

***20)*** Dotaz testující existenci uzlu 5. úrovně, což by byla úroveň, kdyby ještě některý z elementů **<ulice>,<cislo>,<mesto>** v elementu **<Adresa>** měli své podelementy:

```
DOTAZ:
SELECT existsNode(zamestnanci_1, '/*/*/*/*/*')
FROM zamestnanci_firmy;

VÝSLEDEK:
0
```

V předchozích příkladech naleznete nejzákladnější dotazy nad XML daty, které se dají různými změnami modifikovat do takové formy, aby byla uživateli vrácena taková hodnota, jakou požaduje. Příkladů na dotazování se nad XML daty jazykem XPath by šlo nejspíš vymyslet nesčetné množství, ale bylo by to pravděpodobně zbytečné, protože by byly velmi analogické k již zde vytvořeným příkladům.

## 4.4 Příklady vytváření XML z dat tabulky

Proto, abychom měli příklady vytváření XML z dat tabulky na čem předvádět, tak si nejprve vytvoříme tabulku s ukázkovými daty. Vytvořená tabulka bude mít 7 sloupců (**id_zamestnance, jmeno, prijmeni, vek,ulice,mesto,pozice**) a bude naplněna údaji o čtyřech zaměstnancích firmy.

```
CREATE TABLE Zamestnanci_firmy2 (id_zamestnance NUMBER(4) PRIMARY KEY,
                                 jmeno VARCHAR2(10),
                                 prijmeni VARCHAR2(10),
                                 vek NUMBER(2),
                                 ulice VARCHAR2(20),
                                 mesto VARCHAR2(10),
                                 pozice VARCHAR2(15));

INSERT INTO Zamestnanci_firmy2 VALUES (1, 'Karel', 'Kobliha',45,'Zelená
14','Ostrava','Správce sítě');
INSERT INTO Zamestnanci_firmy2 VALUES (2, 'Josef', 'Novák',49,'Dlouhá
13','Zlín','Technik');
INSERT INTO Zamestnanci_firmy2 VALUES (3, 'Pavel', 'Malina',36,'Krátká
47','Olomouc','Vývojář');
INSERT INTO Zamestnanci_firmy2 VALUES (4, 'Jana',
'Černá',29,'Kratochvílova 46','Brno','Personalistka');
COMMIT;
```

Pro některé ukázkové příklady si vytvoříme ještě další tabulku **Zamestnanci_firmy2_platy** s platy zaměstnanců, kterou budeme propojovat se zaměstnanci:

```
CREATE TABLE Zamestnanci_firmy2_platy (id_zamestnance NUMBER(4) PRIMARY
KEY,
                                       plat NUMBER(6));

INSERT INTO Zamestnanci_firmy2_platy VALUES (1, 18000);
INSERT INTO Zamestnanci_firmy2_platy VALUES (2, 16000);
INSERT INTO Zamestnanci_firmy2_platy VALUES (3, 22000);
INSERT INTO Zamestnanci_firmy2_platy VALUES (4, 17000);
COMMIT;
```

### 4.4.1 XMLElement

***1)*** Následující příklad vybere ze všech sloupců **`jmeno`** a **`prijmeni`** a následně z nich vytvoří element **`<Jmeno>`**. Vrátí tak Jména všech zaměstnanců:

```
DOTAZ:
SELECT XMLElement("Jmeno",z.jmeno||' '||z.prijmeni) as "XMLvysledek" from
zamestnanci_firmy2 z

VÝSLEDEK:
<Jmeno>Karel Kobliha</Jmeno>
<Jmeno>Josef Novák</Jmeno>
<Jmeno>Pavel Malina</Jmeno>
<Jmeno>Jana černá</Jmeno>
```

***2)*** Následující příklad vybere ze sloupců **`jmeno`** a **`prijmeni`** zaměstnance, který má příjmení “Malina“ a následně z něj vytvoří element **`<Jmeno>`**:

```
DOTAZ:
SELECT XMLElement("Jmeno",z.jmeno||' '||z.prijmeni) as "XMLvysledek"
FROM zamestnanci_firmy2 z
WHERE prijmeni like 'Malina'

VÝSLEDEK:
<Jmeno>Pavel Malina</Jmeno>
```

***3)*** Následující příklad vytvoří element **`<Zamestnanec>`**, který bude obsahovat podelementy **`<Jmeno>`** a **`<Vek`**. To vše u zaměstnance s hodnotou id “2“:

```
DOTAZ:
SELECT XMLElement("Zemestnanec",XMLElement("Jmeno",z.jmeno||'
'||z.prijmeni),XMLElement("Vek",z.vek)) FROM zamestnanci_firmy2 z
WHERE id_zamestnance=2

VÝSLEDEK:
<Zemestnanec>
  <Jmeno>Josef Novák</Jmeno>
  <Vek>49</Vek>
</Zemestnanec>
```

***4)*** Následující příklad vybere ze sloupců **`jmeno`** a **`prijmeni`** , dále pracovní pozici zaměstnance, který má příjmení "Kobliha" a následně vytvoří element **`<Zamestnanec>`**, který bude mít dva podelementy **`<Jmeno>`** a **`<Pozice>`**:

```
DOTAZ:
SELECT XMLElement("Zemestnanec",XMLElement("Jmeno",z.jmeno||'
'||z.prijmeni),XMLElement("Pozice",z.pozice))
AS "XMLvysledek"
FROM zamestnanci_firmy2 z
WHERE prijmeni LIKE 'Kobliha'

VÝSLEDEK:
<Zemestnanec>
  <Jmeno>Karel Kobliha</Jmeno>
  <Pozice>Správce sítě</Pozice>
</Zemestnanec>
```

***5)*** Následující příklad vybere ze sloupců **`jmeno`** a **`prijmeni`** , dále pracovní pozici zaměstnance, který má příjmení "Černá" a následně vytvoří element **`<Zamestnanec>`**, který bude mít tři podelementy **`<Jmeno>,<Pozice> a <Adresa>.`** V elementu adresa budou dva podelementy **`<Ulice>`** a **`<Mesto>`**:

```
DOTAZ:
SELECT XMLElement("Zemestnanec",XMLElement("Jmeno",z.jmeno||'
'||z.prijmeni),XMLElement("Pozice",z.pozice),
XMLElement("Adresa",XMLElement("Ulice",z.ulice),XMLElement("Mesto",z.mesto)))
AS "XMLvysledek"
FROM zamestnanci_firmy2 z
WHERE prijmeni LIKE 'Černá'

VÝSLEDEK:
<Zemestnanec>
  <Jmeno>Jana Černá</Jmeno>
  <Pozice>Personalistka</Pozice>
  <Adresa>
    <Ulice>Kratochvílova 46</Ulice>
    <Mesto>Brno</Mesto>
  </Adresa>
</Zemestnanec>
```

***6)*** Příklad na smíšený obsah elementů vypíše, v jakém městě bydlí zaměstnanec s id zaměstnance "3":

```
DOTAZ:
SELECT
XMLElement("Zemestnanec",'Zaměstnanec',XMLElement("Jmeno",z.jmeno||'
'||z.prijmeni),'bydlí ve městě',XMLElement("Mesto",z.mesto))
FROM zamestnanci_firmy2 z
WHERE id_zamestnance=3

VÝSLEDEK:
<Zemestnanec>
  Zaměstnanec<Jmeno>Pavel Malina</Jmeno>bydlí ve městě
<Mesto>Olomouc</Mesto>
</Zemestnanec>
```

***7)*** Příklad vytvoří XML se zaměstnanci, jejichž věk je větší než 45 let. Vytvoří se element **<Zamestnanec>** s podelementy **<Jmeno>** a **<Vek>:**

```
DOTAZ:
SELECT XMLElement("Zemestnanec",XMLElement("Jmeno",z.jmeno||'
'||z.prijmeni),XMLElement("Vek",z.vek))
AS "XMLvysledek"
FROM zamestnanci_firmy2 z
WHERE vek>40

VÝSLEDEK:
<Zemestnanec>
  <Jmeno>Karel Kobliha</Jmeno>
  <Vek>45</Vek></Zemestnanec>
<Zemestnanec>
  <Jmeno>Josef Novák</Jmeno>
  <Vek>49</Vek>
</Zemestnanec>
```

### 4.4.2 XMLAttributes

Specifikace jednotlivých atributů se musí vyskytovat jako:

- 3. argument v případě, že je specifikována deklarace jmenných prostorů
- 2. argument v případě, že deklarace specifikována není

***1)*** Příklad vytvoří element **<Zamestnanec>**, ve kterém vypíše do elementů **<Jmeno>** a **<Pozice>** hodnoty zaměstnance s **id_zamestnance** "3". Id zaměstnance vloží jako atribut ID do elementu **<Zamestnanec>:**

```
DOTAZ:
SELECT XMLElement("Zemestnanec",XMLAttributes (z.id_zamestnance AS
"ID"),XMLElement("Jmeno",z.jmeno||'
'||z.prijmeni),XMLElement("Pozice",z.pozice))
AS "XMLvysledek"
FROM zamestnanci_firmy2 z
WHERE id_zamestnance=3

VÝSLEDEK:
<Zemestnanec ID="3">
  <Jmeno>Pavel Malina</Jmeno>
  <Pozice>Vývojář</Pozice>
</Zemestnanec>
```

***2)*** Příklad vytvoří element **<Zamestnanec>** s atributy **Vek_zamestnance** a **Pracovni_pozice** u zaměstnance jehož jméno je "Josef":

```
DOTAZ:
SELECT XMLElement("Zemestnanec",XMLAttributes(z.vek AS "Vek_zamestnance",
z.pozice AS "Pracovni_pozice"),XMLElement("Jmeno",z.jmeno||'
'||z.prijmeni))
AS "XMLvysledek"
FROM zamestnanci_firmy2 z
WHERE jmeno LIKE 'Josef'

VÝSLEDEK:
<Zemestnanec Vek_zamestnance="49" Pracovni_pozice="Technik">
  <Jmeno>Josef Novák</Jmeno>
<Zemestnanec>
```

***3)*** Následující příklad vytvoří elementy **<Zamestnanec>** jejichž obsahem bude **<Jmeno>** zaměstnance a **<Pozice>** zaměstnance u všech zaměstnanců ve firmě dá do elementu **<Zamestnanec>** ID zaměstnance:

```
DOTAZ:
SELECT XMLElement("Zemestnanec",XMLAttributes(z.id_zamestnance AS
"ID"),XMLElement("Jmeno",z.jmeno||'
'||z.prijmeni),XMLElement("Pozice",z.pozice))
AS "XMLvysledek"
FROM zamestnanci_firmy2 z

VÝSLEDEK:
<Zemestnanec ID="1">
  <Jmeno>Karel Kobliha</Jmeno>
  <Pozice>Správce sítě</Pozice>
</Zemestnanec>
<Zemestnanec ID="2">
  <Jmeno>Josef Novák</Jmeno>
  <Pozice>Technik</Pozice>
</Zemestnanec>
<Zemestnanec ID="3">
  <Jmeno>Pavel Malina</Jmeno>
  <Pozice>Vývojář</Pozice>
</Zemestnanec>
<Zemestnanec ID="4">
  <Jmeno>Jana černá</Jmeno>
  <Pozice>Personalistka</Pozice>
</Zemestnanec>
```

***4)*** Příklad vytvoří element **`<Zamestnanec>`** v němž bude jméno, pozice zaměstnance a také adresa, v níž budou jako atributy uvedeny hodnoty **`Ulice`** a **`Mesto`**. To vše u zaměstnance s příjmením “Černá“:

```
DOTAZ:
SELECT XMLElement("Zemestnanec",XMLElement("Jmeno",z.jmeno||'
'||z.prijmeni),XMLElement("Pozice",z.pozice),
XMLElement("Adresa",XMLAttributes(z.ulice AS "Ulice",z.mesto AS
"Mesto")))
AS "XMLvysledek"
FROM zamestnanci_firmy2 z
WHERE prijmeni LIKE 'černá'

VÝSLEDEK:
<Zemestnanec>
  <Jmeno>Jana černá</Jmeno>
  <Pozice>Personalistka</Pozice>
  <Adresa Ulice="Kratochvílova 46" Mesto="Brno"></Adresa>
</Zemestnanec>
```

### 4.4.3 XMLForest

***1)*** Příklad vypíše element **<Zamestnanec>** s podelementy **<Jmeno>** a **<Prijmeni>** zaměstnance s **id_zamestnance** "2":

```
DOTAZ:
SELECT  XMLElement("Zemestnanec",XMLForest(z.jmeno  AS  "Jmeno",z.prijmeni
AS "Prijmeni")) FROM zamestnanci_firmy2 z
WHERE id_zamestnance=2

VÝSLEDEK:
<Zemestnanec>
  <Jmeno>Josef</Jmeno>
  <Prijmeni>Novák</Prijmeni>
</Zemestnanec>
```

***2)*** Následující příklad vypíše v XML všechny informace o zaměstnanci, jehož jméno je "Pavel":

```
DOTAZ:
SELECT XMLElement("Zemestnanec",XMLForest(z.id_zamestnance AS "ID",
z.jmeno || ' ' || z.prijmeni AS "Jmeno",z.pozice AS
"Pracovni_pozice",z.vek AS "Vek",z.ulice || ', ' || z.mesto AS "Adresa"
)) FROM zamestnanci_firmy2 z
WHERE z.jmeno LIKE 'Pavel'

VÝSLEDEK:
<Zemestnanec>
  <ID>3</ID>
  <Jmeno>Pavel Malina</Jmeno>
  <Pracovni_pozice>Vývojář</Pracovni_pozice>
  <Vek>36</Vek>
  <Adresa>Krátká 47, Olomouc</Adresa>
</Zemestnanec>
```

***3)*** Příklad vypíše všechny zaměstnance v tabulce, jejichž **id_zamestnance** je větší než "2". Vypíše u nich elementy **<Jmeno>** a **<Pozice>:**

```
DOTAZ:
SELECT XMLElement("Zemestnanec",XMLForest(z.jmeno || ' ' || z.prijmeni AS
"Jmeno" ,z.pozice AS "Pozice"))
FROM zamestnanci_firmy2 z
WHERE id_zamestnance>2

VÝSLEDEK:
<Zemestnanec>
  <Jmeno>Pavel Malina</Jmeno>
  <Pozice>Vývojář</Pozice>
</Zemestnanec>
<Zemestnanec>
  <Jmeno>Jana černá</Jmeno>
  <Pozice>Personalistka</Pozice>
</Zemestnanec>
```

***4)*** Příklad vypíše v XML všechny zaměstnance v tabulce, jejichž **plat** je menší než "17000". Vypíše u nich elementy **<Jmeno>**, **<Plat>** a **<Pozice>:**

```
DOTAZ:
select XMLElement("Zamestnanec",XMLForest(
  z.jmeno||' '||z.prijmeni as "Jmeno",
  p.plat as "Plat",z.pozice AS "Pozice")) as "XMLvysledek"
from zamestnanci_firmy2 z
join zamestnanci_firmy2_platy p
on z.id_zamestnance= p.id_zamestnance
WHERE p.plat<17000

VÝSLEDEK:
<Zamestnanec>
  <Jmeno>Josef Novák</Jmeno>
  <Plat>16000</Plat>
  <Pozice>Technik</Pozice>
</Zamestnanec>
```

***5)*** Příklad v XML všechny zaměstnance firmy s elementy **<Jmeno>** a **<Plat>**, aby bylo vidět, kolik, který zaměstnanec bere. V příkladu jsou využity data z připojené tabulky: :

```
DOTAZ:
select XMLElement("Zamestnanec",XMLForest(
  z.jmeno||' '||z.prijmeni as "Jmeno",
  p.plat as "Plat")) as "XMLvysledek"
from zamestnanci_firmy2 z
join zamestnanci_firmy2_platy p
  on z.id_zamestnance= p.id_zamestnance

VÝSLEDEK:
<Zamestnanec>
  <Jmeno>Karel Kobliha</Jmeno>
  <Plat>18000</Plat>
</Zamestnanec>
<Zamestnanec>
  <Jmeno>Josef Novák</Jmeno>
  <Plat>16000</Plat>
</Zamestnanec>
<Zamestnanec>
  <Jmeno>Pavel Malina</Jmeno>
  <Plat>22000</Plat>
</Zamestnanec>
<Zamestnanec>
  <Jmeno>Jana černá</Jmeno>
  <Plat>17000</Plat>
</Zamestnanec>
```

### 4.4.4 XMLConcat

XMLConcat vytváří XML hodnotu, která je daná dvěma nebo více výrazy typu XML.

***1)*** Příklad spojí dvě XML hodnoty Jméno zaměstnance a jeho pracovní pozici:

```
DOTAZ:
SELECT XMLConcat(XMLElement("Jmeno",z.jmeno||'
'||z.prijmeni),XMLElement("Pozice",z.pozice)) as "XMLvysledek"
FROM zamestnanci_firmy2 z
WHERE id_zamestnance=4

VÝSLEDEK:
<Jmeno>Jana černá</Jmeno>
<Pozice>Personalistka</Pozice>
```

***2)*** Následující příklad spojí dva XML zaměstnance a to konkrétně zaměstnance s **id_zamestnance** “1“ a “3“:

```
DOTAZ:
SELECT  XMLElement("Zamestnanci",XMLCONCAT (
(select XMLElement("Zamestnanec",XMLForest(z.jmeno||' '||z.prijmeni AS
"Jmeno")) AS "Zamestnanec1"
FROM zamestnanci_firmy2 z
WHERE id_zamestnance=1) , (select XMLElement("Zamestnanec",
XMLForest(z.jmeno||' '||z.prijmeni AS "Jmeno")) AS "Zamestnanec2"
FROM zamestnanci_firmy2 z
WHERE id_zamestnance=3))) AS "XMLvysledek"
FROM zamestnanci_firmy2 z

VÝSLEDEK:
<Zamestnanci>
  <Zamestnanec>
    <Jmeno>Karel Kobliha</Jmeno>
  </Zamestnanec>
  <Zamestnanec>
    <Jmeno>Pavel Malina</Jmeno>
  </Zamestnanec>
</Zamestnanci>
```

### 4.4.5 Další funkce pro práci s XML

- **`updateXML`** – nahradí XML uzel libovolného typu
- **`insertXMLbefore`** a **`insertXMLafter`** – vloží XML uzly libovolného typu před/za daný (neatributový) uzel
- **`insertChildXML`** – vloží dané elementy nebo atributy jako "*child*" uzly daného elementu
- **`insertChildXMLbefore`** a **`insertChildXMLafter`** – vloží novou kolekci XML elementů před/za danou kolekci elementů stejného typu
- **`appendChildXML`** – vloží XML uzly libovolného typu za potomky daného elementu
- **`deleteXML`** – smaže XML uzel libovolného typu

Funkcí pro práci s XML je mnohem více, než jsme si v této příručce uvedli, a kdybychom je chtěli vypsat i s příklady všechny, tak by to nejspíše vydalo na docela obsáhlou knihu. Proto jsou v příručce uvedeny ty nejpraktičtější příklady, jaké se dají pro začátky práce s XML v databázi Oracle využít.