



INVESTICE DO ROZVOJE VZDĚLÁVÁNÍ

Realizováno za finanční podpory ESF a státního rozpočtu ČR  
v rámci v projektu *Zkvalitnění a rozšíření možností studia  
na TUL pro studenty se SVP* reg. č. CZ.1.07/2.2.00/29.0011

# XQuery

# XQuery

- dotazovací jazyk pro XML informace (něco jako SQL pro databáze)
- umožňuje jen dotazování, data nelze měnit
- zdrojem nemusí být jen XML, ale i „podobná“ data
- cíl: sjednotit XML a databáze a ke kolekcím XML souborů přistupovat databázovým způsobem
- standardizace dokončena v lednu 2007

# Principy XQuery

- funkcionální jazyk – vše je výraz, jehož vyhodnocením vznikne určitá hodnota
- základní typy – stejné jako v XML Schema:
  - čísla
  - řetězce znaků (nelze do nich zasahovat)
  - pravdivostní hodnoty true/false
  - časové hodnoty
  - několik typů souvisejících s XML, podstatný je typ uzel


# Příklad na úvod – ceník

```
<html>
<head><title>Ceník</title></head>
<body>
<h1>Ceník</h1>
<table>
{ for $zbozi in /cenik/zbozi
  return
    <tr>
      <td>{$zbozi/nazev/text()}</td>
      <td align="right">{$zbozi/cena/text()}</td>
    </tr> }
</table>
</body></html>
```

# Syntax XQuery

- není XML (resp. existuje XML syntax jazyka označovaná jako XQueryX)
- připomíná PHP – míchají se opisované části výstupu s konstrukcemi XQuery
  - uzavřeny do složených závorek { ... }
- nejjednodušším XQuery výrazem je XPath výraz, výsledkem jsou vybrané uzly:  
`doc("cenik.xml")//zbozi[nazev="Rohlík"]`

# Vytvoření prvku (1)

- v XPath nelze vytvořit nový prvek, jen vybírat existující
- pro XQuery není problém, nejjednodušší je opisem:  
`<polozka>Rohlík</položka>`
- lze samozřejmě využívat informace ze vstupu:  
`<polozka>`  
`{$zbozi/nazev/text()}`  
`</polozka>`  
 *proč je zde text()?*

# Vytvoření prvku (2)

- je-li třeba vytvořit název vytvářeného prvku, lze použít **element {jméno} {hodnota}**
- např. do výstupního XML vložíme prvek, jehož jméno odpovídá atributu **druh** prvku **zbozi** ve vstupním XML, obsah zůstane zachován:  
**for \$zbozi in /cenik/zbozi**  
**return**  
**element {\$zbozi/@druh} {\$zbozi/\*}**


# Vytvoření atributu (1)

- opisem nebo konstrukcí z údajů ze vstupu (automaticky se převádí na text)  
**<polozka cena="{ \$zbozi/cena }">  
    { \$zbozi/nazev/text() }  
</polozka>**
- prvky převzaté ze vstupu se přebírají se všemi potomky a atributy



# Vytvoření atributu (2)

- vložením do těla prvku

`<polozka>`  
    `{ $zbozi/@id }`  *přidá prvku <polozka> atribut id*  
    `{ $zbozi/nazev/text() }`  
`</polozka>`

je-li vložen atribut ze vstupních dat, stane se atributem prvku, do nějž byl vložen – výsledek:

`<polozka id="zb001">Rohlík</polozka>`

- chcete-li jen hodnotu, použijte `data($zbozi/@id)` nebo `string($zbozi/@id)`

# Vytvoření atributu (3)

- pro dynamické určení jména atributu:  
**attribute {jméno} {hodnota}**
- existují i další konstruktory, používány méně často:
  - **text {hodnota}** pro vytvoření textového uzlu
  - **document {hodnota}** pro vytvoření dokumentového uzlu

# Výraz vložený v prvku

- jeho výsledek se stává obsahem obalujícího prvku, konkrétní efekt závisí na typu výsledku:
  - **prvek** (nebo sekvence prvků) – potomci
  - **atribut** – stane se atributem obalujícího prvku
  - **atomická hodnota** – převedena na řetězec a vložena jako textový uzel

# Seznamy

- XQuery vedle jednoduchých hodnot podporuje i seznamy (sekvence)
- uzavřené do (...), hodnoty oddělené čárkami
- nelze vnořovat
- k dispozici speciální operace a funkce
  - množinové: union, intersect, except
  - remove, index-of, count, sum, avg, max, min,...

# Vstupní data

- **implicitní**

- XPath výrazy začínající / předpokládají, že zpracovávaná XML data existují
- jak zadat zpracovávaný soubor závisí na implementaci
- **for \$zbozi in /cenik/zbozi**

- **explicitní**

- XPath výrazy začínající **doc()** berou data z konkrétního souboru
- **for \$zbozi in doc("cenik.xml")/cenik/zbozi**

# for

- **for *proměnná* in *hodnoty* return *výsledek***
  - *proměnná* nabude každou *hodnotu* a určí se pro ni *výsledek*
  - vyhodnocení v samostatných vláknech – lze paralelně
- příklad: seznam potravin

<ol>

```
{ for $zbozi in /cenik/zbozi[@druh="potravina"]  
  return  
  <li>{ data($zbozi/nazev) } </li> }
```

</ol>

# FLWOR výrazy

- základní stavební kámen XQuery
- For-Let-Where-Order-Return

```
for $odd in doc("oddeleni.xml")//deptno
let $zam := doc("zamest.xml")//employee[deptno = $odd]
where count($zam) >= 10
order by avg($zam/plat) descending
return
```

```
<megaoddeleni id="{ $odd }">
  <pocetlidi>{ count($zam) }</pocetlidi>
  <prumplat>{ avg($zam/plat) }</prumplat>
</megaoddeleni>
```

# FLWOR pravidla

- (for ... | let ... )+ (where ... )? (order by ... )? return ...
- for a let lze použít v libovolném počtu a pořadí, alespoň jedno být musí
- je povoleno jen jedno where
  - ale může obsahovat složitou podmínku s and a or
- jen jedno order by
  - ale může mít více kritérií pro řazení, oddělovat čárkami
- return je povinné



# let

- **let *\$proměnná* := výraz**
- vyhodnotí *výraz* a výsledek uloží do *proměnné*, ta se dál nemění – nabývá hodnotu jen při vytvoření
- *proměnná* může obsahovat uzel či posloupnost uzlů
- použití: často opakované výrazy, parametry
- *proměnné* mohou zjednodušit složité transformace
  - může být čitelnější než XSLT

# Příklad: Druhy zboží

```
<h1>Druhy zboží</h1>
```

```
<ol>
```

```
{
```

```
  let $druhy := /cenik/zbozi/string(@druh)
```

```
  for $d in distinct-values($druhy)
```

```
    return
```

```
      <li>{$d}</li>
```

```
}
```

```
</ol>
```

# let versus for

- **for \$zb in /cenik/zbozi**
  - vytvoří seznam vazeb \$zb na jednotlivá zboží z ceníku
  - return se vyhodnotí pro každou zvlášť (tolikrát, kolik zboží je v ceníku)
- **let \$zb := /cenik/zbozi**
  - vytvoří jednu vazbu \$zb na sekvenci prvků zboží z ceníku
  - return se vyhodnotí je jednou

# Řazení

- pokud má XQuery procesor k dispozici typy hodnot, bere na ně ohled
- netypované hodnoty řadí lexikograficky
  - lze změnit: **order by number(\$zbozi/cena)**
- nepovinné modifikátory za výrazem:
  - **ascending/descending**
  - **empty greatest/empty least**
  - **collation "URI"** – řadit podle národních specifik
  - **stable** (před order by) – u shodných zachovat pořadí

# Příklad: Druhy zboží

```
<h1>Druhy zboží</h1>
```

```
<ol>
```

```
{
```

```
  let $druhy := /cenik/zbozi/string(@druh)
```

```
  for $d in distinct-values($druhy)
```

```
    order by $d
```

```
    return
```

```
    <li>{$d}</li>
```

```
}
```

```
</ol>
```

# Podmíněný výraz

- *if (podmínka) then výsledek1 else výsledek2*  
v obvyklém významu

```
for $zbozi in doc("cenik.xml")/cenik/zbozi  
  return
```

```
...
```

```
  if ($zbozi/@druh = "potravina")  
    then <kategorie>potravina</kategorie>  
    else <kategorie>ostatni</kategorie>
```

# Struktura dokumentu

- XQuery dokument obsahuje nepovinný prolog a povinné tělo
- **prolog**
  - pouze deklarace
  - proměnných, funkcí, jmenných prostorů apod.
  - oddělovány středníky
- **tělo**
  - akční část dokumentu

# Příklad prologu

```
xquery version="1.0" encoding="UTF-8";  
declare variable $kapacita := 100;  
declare variable $zbozi := /cenik/zbozi;  
declare namespace zb = "http://www.tul.cz/nti/zbozi";
```



# Jmenné prostory (1)

- deklarace v prologu zavádí prefixy pro XQuery
- pokud se vytváří XML dokument, hrají roli i jeho deklarace, pozor na implicitní jmenný prostor – bude uplatněn i na prvky v XPath výrazech
- `<html xmlns="http://www.w3.org/1999/xhtml" ...>`  
...  
`{for $zbozi in /cenik/zbozi ... }`  
způsobí, že **cenik** a **zbozi** budou zařazeny do  
jmenného prostoru XHTML

# Jmenné prostory (2)

- řešení 1: používat pro své jazyky vlastní jmenné prostory

- `<html xmlns="http://www.w3.org/1999/xhtml" xmlns:zb="http://www.tul.cz/nti/zbozi" ...>`

...

`{ for $zbozi in /zb:cenik/zb:zbozi ... }`

- řešení 2: vyhnout se implicitním jm. prostorům

- `<h:html xmlns:h="http://www.w3.org/1999/xhtml" ...>`

...

`{ for $zbozi in /zb:cenik/zb:zbozi ... }`


# Uživatelské funkce (1)

- lze definovat vlastní funkce  
**declare function *jméno (parametry)***  
***{ výraz };***
- jméno musí být kvalifikované (s prefixem)
- výsledek není třeba explicitně definovat – tělem je výraz, ten se vyhodnotí a určí výsledek funkce
- použití stejné jako u vestavěných funkcí:  
***jméno(parametry)***

# Příklad: Připočteme DPH

```
declare namespace zb = "http://www.tul.cz/nti/zbozi";
```

```
declare variable $zb:dphkoef := 1.21;
```

```
declare function zb:dph ( $cena )  deklarace  
{
```

```
    $cena * $zb:dphkoef  
};
```

 *u proměnné není prefix povinný, zařazením do společného prostoru zdůrazňují jejich vztah*

```
...
```

```
for $zbozi in /cenik/zbozi
```

```
    ...  
    <td align="right"> {zb:dph(data($zbozi/cena))}</td>
```

*použití*

# Uživatelské funkce (2)

- tělo může být samozřejmě výrazně složitější, včetně rekurze
- příklad: vydá prvek sám + všechny jeho potomky

```
declare function muj:descendant-or-self ($x)
```

```
{
```

```
    $x,
```

```
    for $y in $x/*
```

```
        return muj:descendant-or-self($y)
```

```
}
```

# Moduly (1)

- složitější XQuery lze rozložit do modulů
- modul obsahuje jen preambuli
- syntaxe lehce pozměněna:
  - začíná  
**module namespace prefix = "URI";**
  - definuje cílový jmenný prostor
  - všechny proměnné a funkce definované modulem musí být v tomto prostoru

# Příklad modulu

```
module namespace zb = "http://www.tul.cz/nti/zbozi";  
declare variable $zb:dphkoef := 1.21;  
declare function zb:dph ( $scena )  
{  
    $scena * $zb:dphkoef  
};
```

# Moduly (2)

- použití modulu:  
**import module namespace *prefix* = "URI"**  
**at "*soubor*"**
- *URI* musí odpovídat deklaraci v modulu, *prefix* se může změnit
- následně lze používat proměnné a funkce definované v *souboru* (s příslušným *prefixem*)
- lze použít libovolný počet modulů



# Typy (1)

- nejsou nezbytné
- lze používat **typy z XML Schema**
  - zpřístupní deklarace jmenného prostoru  
`declare namespace xsd =`  
`"http://www.w3.org/2001/XMLSchema";`
  - přetypování hodnoty: `typ(hodnota)`  
`xsd:decimal($cena) * $zb:dphkoef`

# Typy (2)

- **vlastní typy**

- importovat XML Schema definici:

- `import schema namespace zb =`

- `"http://www.tul.cz/nti/zbozi" at "cenik.xsd";`

- nebo

- `import schema default element namespace "" at "...";`

- následně lze používat prvky, atributy a typy ze schématu

- **testování typů:**

- `instance of`, `castable as`

- `typeswitch`

# Textový výstup (1)

- v principu možný – důsledně do výstupu posílat textové uzly nebo převádět na řetězce znaků
- např. ceník v textové podobě:

```
for $zbozi in /cenik/zbozi  
    return (  
        $zbozi/nazev/text(),  
        "...",  
        $zbozi/cena/text(),  
        "&#xa;"  
    )
```

# Textový výstup (2)

- formátování pomocí konstant
- může vyžadovat potlačení XML prologu a/nebo dalších složek
  - specifické pro konkrétní implementaci
  - např. Saxon:

```
declare namespace saxon = "http://saxon.sf.net/";  
declare option saxon:output "indent=no";  
declare option saxon:output "method=text";
```

# Implementace

- poměrně hojné, i open source, například:
  - **Saxon** (též XSLT procesor) – [saxon.sourceforge.net](http://saxon.sourceforge.net), 100% úspěch v testu compatibility
  - **Galax** – [www.galaxquery.org](http://www.galaxquery.org)
  - **Oracle Berkeley DB XML** – XML databáze
  - **Qizx** – <http://www.axyana.com/qizxopen/>
- test compatibility:  
<http://www.w3.org/XML/Query/test-suite/XQTSReport.html>

# XQuery versus XSLT

- podobné schopnosti, stejný datový model vycházející z XPath (strom dokumentu, uzly,...)
- XSLT implementace bývají optimalizovány na transformaci celých dokumentů, XQuery spíše na vybírání fragmentů
- XQuery více připomíná programovací jazyk
- syntax XQuery bývá kompaktnější