DOTAZOVACÍ JAZYKY - XML
slajdy k přednášce NDBI006

J. Pokorný
KSI MFF UK

Syllabus pro DJ2

1. XML databáze;
2. Dotazovací jazyky nad XML daty
3. Tři sémantiky DRK. Definitní a bezpečné formule DRK. Důkaz ekvivalence relační algebry a DRK omezeného na definitní formule.
4. Důkaz věty o nemožnosti vyjádřit tranzitivní uzávěr relace v relační algebře.
5. Kompozice výrazů relační algebry, nejmenší pevný bod zobrazení, minimální pevný bod. Datalog - tři možné sémantiky jazyka. Vyhodnocení logického programu bez rekurze.
6. Datalog s rekurzí. Naivní metoda vyhodnocení, metoda diferencí.
7. Datalog s negací, stratifikace. Vyjadřovací síla Datalogu. Vztah k dalším relačním jazykům. Deduktivní databáze. Logické problémy informačních systémů.
8. Rekurzivní SQL
9. Herbrandovské struktury a báze, svazy a Tarského věta o fixpointu, produkční operátor
10. Disjunktivní Datalog
11. Tabulkové dotazy
12. Složitost dotazovacích problemů a statická analýza dotazů
13. Dotazování s užívatelskými preferencemi
14. Web jako databáze. Dotazovací jazyky na Webem. SPARQL

Obsah

- Část 1: Základy - dokumenty (SGML/HTML) a databáze (strukturovaná a semistrukturovaná data)
- Část 2: Datový model XML
- Část 3: XML a dotazovací jazyky
  - příklady: XML-QL, XPath
- Část 4: SQL/XML

Část I: Základy

Dokumenty vs. databáze

Svět dokumentů	Svět databází
> mnoho malých dokumentů	> několik rozsáhlých databází
> obvykle statický	> obvykle dynamický
> implicitní struktura sekce, paragraf, věta,	> explicitní struktura (schéma)
> značkování	> záznamy
> přizpůsobený člověku	> přizpůsobený stroji
> obsah formát/rozvržení na médiu, anotace	> obsah schéma, data, metody
> paradigmata “ulož jako”, wysiwyg	> paradigmata atomicita, souběžnost, izolace, trvanlivost
> metadata autor jméno, datum, předmět	> metadata popis schématu

Co s nimi dělat

Dokumenty	Databáze
• editace	• aktualizace
• tisk	
• lexikální kontrola • počítání slov	• čištění dat
• výběr informací (IR)	• dotazování
• prohledávání	
	• skládání/transformování

Hranice mezi dokumenty a db

- Hranice mezi světem dokumentů a světem databází je nejasná.
- V některých návrzích jsou legitimní oba přístupy.
- Někde uprostřed jsou
  - *formátovací jazyky*
  - *semistrukturovaná data*

výzkum
z 2. pol. 90. let

Semistrukturovaná data jsou definována jako data, která jsou neuspořádaná či neúplná, jejich struktura se může měnit, dokonce nepredikovatelným způsobem.

Př.: data ve webových zdrojích, HTML stránky, Bibtexovské soubory, biologická data.

XML data jsou instancí semistrukturovaných dat.

HTML

- Lingua franca pro publikování hypertextu na WWW
- Navržen pro popis, jak by měl webovský prohlížeč uspořádat text, obrázky a tlačítka na stránce.
- Jednoduchý k učení, ale neudržující strukturu.
- Pevná množina značek.
- HTML $\in$ XML $\subset$ SGML

Text (PCDATA)

otevírající značka

```
<HTML>
<HEAD><TITLE>Sejdeme se Na Vlachovce</TITLE></HEAD>
<BODY>

<H1>Úvod</H1>
<IMG SRC="http://.../logo.jpeg" WIDTH="200" HEIGHT="150" >

</BODY>
</HTML>
```

uzavírající značka

Jméno atributu

Jméno atributu

Struktura XML

- XML sestává ze značek a textu

- značky jsou v párech <datum> ...</datum>

- musí být správně hnízděné

  <datum> <den> ... </den> ... </datum> --- dobře

  <datum> <den> ... </datum>... </den> --- špatně

(nelze dělat ... ...)

XML text

XML má pouze jeden “základní” typ -- text.

Text je ohraničen značkami, např.

<název> Databázová abeceda </název>

<rok> 1999 </ rok> --- 1999 je text

XML text se označuje PCDATA (angl. parsed character data). Používá 16-bitové kódování (Unicode)

Později uvidíme, jak jsou pomocí XML dat specifikovány nové typy

XML struktura

Hnízděné značky mohou být použity k vyjádření různých struktur. Např. n-tice (řádek):

```
<osoba>
  <jméno> Jana Dejmková </jméno>
  <tel> 2191 4264 </tel>
  <email> dejmkova@ksi.ms.mff.cuni.cz </email>
</osoba>
```

XML struktura (pokr.)

- Seznam můžeme reprezentovat opakovaně použitím *stejné* značky:

```
<adresy>
   <osoba> ... </osoba>
   <osoba> ... </osoba>
   <osoba> ... </osoba>
   ...
</adresy>
```

Terminologie

Segment XML dokumentu s počáteční a korespondující koncovou značkou se nazývá *element.* Text mezi značkami je obsah elementu.

```
<osoba>
    <jméno> Jana Dejmková </jméno>
    <tel> 2191 4264 </tel>
    <tel> 2191 4363 </tel>
    <email> dejmkova@ksi.mff.cuni.cz </email>
</osoba>
```

element

element, podelement

není element

Element může být prázdný (nemá obsah - kromě atributů)

<jméno> </jméno> nebo zkráceně *<jméno/>*

Terminologie

Počáteční značka elementu může obsahovat atributy.
Jsou používány typicky k popsání obsahu elementu

```
<položka>
  <slovo jazyk = “A”> cheese </slovo>
  <slovo jazyk = “F”> fromage </slovo>
  <slovo jazyk = “N”> Käse </slovo>
  <význam> potravina vytvořená … </význam>
</položka>
```

Atributy

Další využití - vyjádření dimenze nebo typu

```
<obrázek>
    <výška dim = “cm”> 2400 </výška>
    <šířka dim = “in”> 96 </ šířka>
    <data kódování = “gif” komprese = “zip”>
        M05-.+C$@02!G96YE<FEC ...
    </data>
</obrázek>
```

Smíšený obsah

element může obsahovat směs elementů a dat typu PCDATA

```
<praní>
    <jméno> Persil 1.2 </jméno>
    <motto>
      Světový <pochybný> favorit </pochybný> prášků
    </motto>
</praní>
```

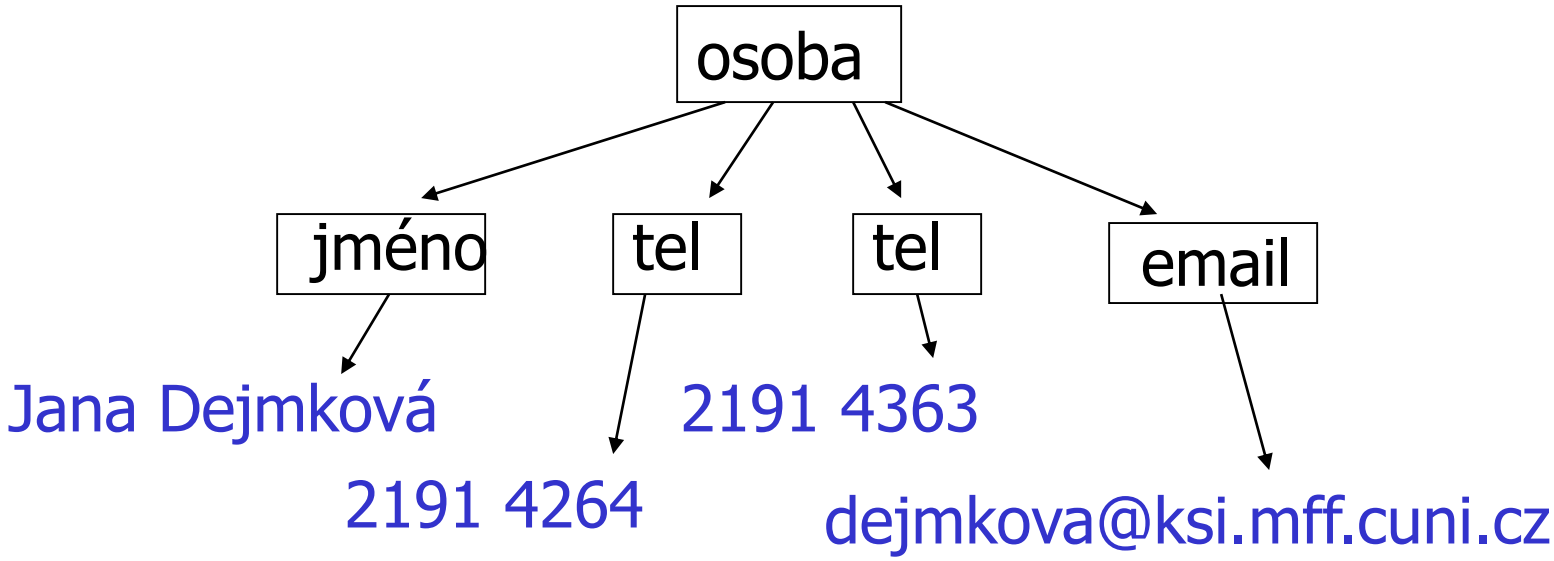
Data tohoto tvaru nejsou typicky generována z (relačních) databází.

Úplný XML Dokument

```
<?xml version="1.0"?>
<osoba>
  <jméno> Jana Dejmková </jméno>
  <tel> 2191 4264 </tel>
  <email> dejmkova@ksi.mff.cuni.cz </email>
</osoba>
```

deklarace XML

XML má stromovou strukturu



Pz.:

- na obrázku je model XML textu
- rozdíly vzhledem k modelům semistrukturovaných dat, které používají typicky ohodnocení hran

Příklad: reprezentace relační DB

projekty:

název	rozpočet	řízen

zaměstnanci:

jméno	rod_č	věk

Relace projekty a zaměstnanci v XML

projekty a zaměstnanci jsou smíchaní

```
<db>
  <projekt>
    <název> Vyhledávání </název>
    <rozpočet> 100000 </rozpočet>
    <řízen> Kopecký, M. </řízen>
  </projekt>
  <zaměstnanec>
    <jméno> Dvorský, J. </jméno>
    <rod_č> 700321/1423 </rod_č>
    <věk> 29 < /věk>
  </zaměstnanec>
  <zaměstnanec>
    <jméno> Mikulová,L. </jméno>
    <rod_č> 715512/0132 </rod_č>
    <věk> 38 </věk>
  </zaměstnanec>
  <projekt>
    <název> Třídění </název>
    <rozpočet> 700000 </rozpočet>
    <řízen> Mikulová,L. </řízen>
  </projekt>
  :
</db>
```

Relace projekty a zaměstnanci v XML(pokr.)

zaměstnanci jsou “za” projekty

```
<db>
   <projekty>
     <projekt>
          <název> Vyhledávání </název>
          <rozpočet> 100000 </rozpočet>
          <řízen> Kopecký, M. </řízen>
     </projekt>
     <projekt>
          <název> Třídění </název>
          <rozpočet> 700000 </rozpočet>
          <řízen> Mikulová,L. </řízen>
     </projekt>
   :
   </projekty>
```

```
<zaměstnanci>
  <zaměstnanec>
     <jméno> Kopecký, M. </jméno>
     <rod_č> 640802/3200</rod_č>
     <věk> 35 </věk>
  </zaměstnanec>
  <zaměstnanec>
     <jméno> Mikulová,L. </jméno>
     <rod_č> 715512/0132 </rod_č>
     <věk>38 </věk>
  </zaměstnanec>
:
<zaměstnanci>
</db>
```

Relace projekty a zaměstnanci v XML(pokr.)

nebo bez značky “separátoru” …

```
<db>
  <projekty>
    <název> Vyhledávání </název>
    <rozpočet> 100000 </rozpočet>
    <řízen> Kopecký, M. </řízen>
    <název> Třídění </název>
    <rozpočet> 700000 </rozpočet>
    <řízen> Mikulová,L </řízen>
    :
  </projekty>
  <zaměstnanci>
    <jméno> Kopecký, M. </jméno>
    <rod_č> 640802/3200 </rod_č>
    <věk> 35 </věk>
    <jméno> Mikulová, L</jméno>
    <rod_č> 715512/0132 </rod_č>
    <věk> 38 </věk>
    :
  </zaměstnanci>
</db>
```

Více o atributech

```
<db>
  <film id="f1">
    <název>Turbína</název>
    <režisér>Novák A.</režisér>
    <obsazení idrefs="h1 h2"></obsazení>
    <rozpočet>100000</rozpočet>
  </film>
  <film id="f2">
    <název>Batalion</název>
    <režisér>Buřita S.</režisér>
    <obsazení idrefs="h2 h9 h21"></obsazení>
    <rozpočet>110000</rozpočet>
  </film>
  <film id="f3">
    <název>Gabriela</název>
    <režisér>Vrchota J.</režisér>
    <obsazení idrefs="h1 h8"></obsazení>
    <rozpočet>90000</rozpočet>
  </film>
  :
```

```
  <herec id="h1">
    <jméno>M. Glázrová</jméno>
    <hrající_v idrefs="f1 f3 f78" >
    </hrající_v>
  </herec>
  <herec id="h2">
    <jméno>K. Höger</jméno>
    <hrající_v idrefs="f1 f2 f11">
    </hrající_v>
    <věk>38</věk>
  </herec>
  <herec id="h3">
    <jméno>H. Vítová</jméno>
    <hrající_v idrefs="f2 f35">
    </hrající_v>
  </herec>
  :
</db>
```

Kdy použít atributy

Není vždy jasné, kdy použít atributy. Atributy nejsou „vidět“.

```
<osoba rod_č= “780730/0013”>
  <jméno> J. Blatný </jméno>
  <email>
     blatný@ksi.mff.cuni.cz
  </email>
  ...
</osoba>
```

```
<osoba>
  <rod_č> 780730/0013 </rod_č>
  <jméno> J. Blatný </jméno>
  <email>
        blatný@ksi.mff.cuni.cz
  </email>
  ...
</osoba>
```

Dokument, který vyhovuje pravidlu “hnízdění značek” a nemá stejné atributy uvnitř značky, se nazývá dobře vytvořený.

Část II: Datový model XML

- Popis typu dokumentu pomocí DTD
- Vztah k objektovému datovému modelu
- Vztah k modelu semistrukturovaných dat
- Rozšíření datového modelu XML

Popis typu dokumentu pomocí DTD

- Document Type Descriptors (DTDs) přiřazují XML dokumentu strukturu.
- existuje *jistý* vztah mezi DTD a schématem databáze,
- DTD je a *syntaktická* specifikace.

Příklad: Osobní adresář

```
<osoba>
    <jméno> Říha Antonín </jméno>
    <s_titulem> Dr. A. Říha </ s_titulem>
    <adresa>Malostranské 25 </adresa>
    <adresa> Praha, 100 00 </adresa>
    <tel> 2191 4268 </tel>
    <fax> 2191 4323 </fax>
    <tel> 2191 4323 </tel>
    <email> riha@ksi.mff.cuni.cz</email>
</osoba>
```

přesně jedno jméno

Max. 1

tolik řádků pro adresy, kolik je třeba

smíchané telefony a faxy

kolik jich je třeba

Specifikace struktury

- jméno — specifikuje element jméno
- s_titulem? — specifikuje nepovinné (0 nebo 1) elementy s_titulem
- jméno, s_titulem? — specifikuje jméno následované nepovinně s_titulem
- adresa* — specifikuje 0 nebo více řádků adresa
- tel | fax — tel *nebo* fax element
- (tel | fax)* — 0 nebo více tel nebo fax
- email* — 0 nebo více elementů email

Specifikace struktury (pokr.)

celá struktura elementu osoba je specifikována

jméno, s_titulem?, adresa*, (tel | fax)*, email*

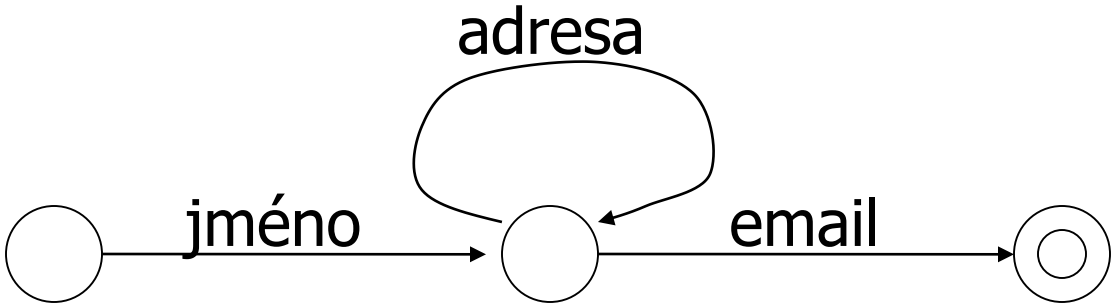
Jde o regulární výraz. Proč je důležitý?

Regulární výrazy

Každý regulární výraz určuje korespondující *konečný automat.*
Příklad:

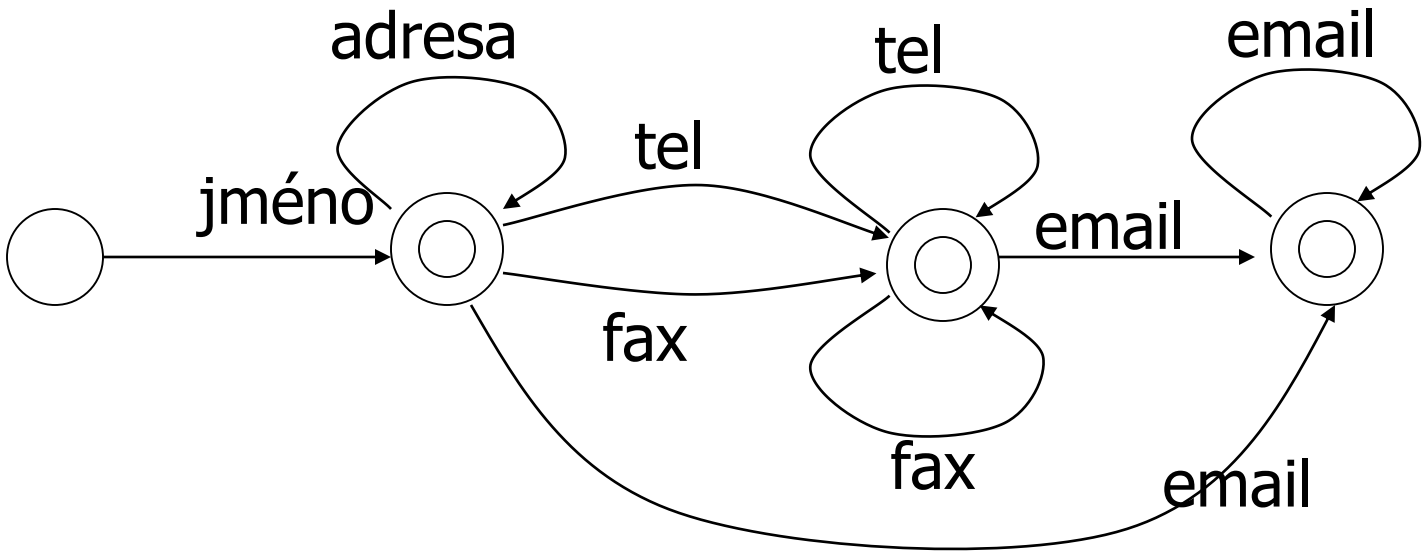
jméno, adresa*, email

Odpovídající jednoduchý program (parser)



Další příklad

jméno, adresa*, (tel | fax)*, email*



Přidání nepovinného s_titulem vede ke komplikacím s velikostí automatu

DTD pro adresář

```
<!DOCTYPE adresář [
  <!ELEMENT adresář (osoba*)>
  <!ELEMENT osoba
      (jméno, s_titulem?, adresa*, (fax | tel)*, email*)>
  <!ELEMENT jméno   (#PCDATA)>
  <!ELEMENT s_titulem    (#PCDATA)>
  <!ELEMENT adresa  (#PCDATA)>
  <!ELEMENT tel     (#PCDATA)>
  <!ELEMENT fax     (#PCDATA)>
  <!ELEMENT email   (#PCDATA)>
]>
```

Revidovaná relační DB

projekty:

název	rozpočet	řízen

zaměstnanci:

jméno	rod_č	věk

Dva DTD relační DB

```
<!DOCTYPE db [
  <!ELEMENT db          (projekty,zaměstnanci)>
  <!ELEMENT projekty    (projekt*)>
  <!ELEMENT zaměstnanci (zaměstnanec*)>
  <!ELEMENT projekt     (název, rozpočet, řízen)>
  <!ELEMENT zaměstnanec (jméno, rod_č, věk)>
  ...
]>
```

```
<!DOCTYPE db [
  <!ELEMENT db          (projekt | zaměstnanec)*>
  <!ELEMENT projekt     (název, rozpočet, řízen)>
  <!ELEMENT zaměstnanec (jméno, rod_č, věk)>
  ...
]>
```

Rekurzivní DTD

```
<!DOCTYPE genealogie [
    <!ELEMENT genealogie (osoba*)>
    <!ELEMENT osoba (
                    jméno,
                    datum_nar,
                    osoba,                    -- matka
                    osoba   )>                -- otec
    ...
]>
```

Kde je problém? Povinní rodiče. Pořadí.

Rekurzivní DTD (pokr.)

```
<DOCTYPE genealogie [
    <!ELEMENT genealogie (osoba*)>
    <!ELEMENT osoba (
                    jméno,
                    datum_nar,
                    osoba?,                    -- matka
                    osoba?  )>                 -- otec
    ...
]>
```

Kde je nyní problém? Pořadí.

Lepší řešení: s ID, IDREF, IDREFS

Některé věci je obtížné specifikovat

Každý zaměstnanecký element obsahuje elementy jméno, věk a rod_č v nějakém pořadí.

```
<!ELEMENT zaměstnanec
   ( (jméno, věk, rod_č) | (věk, rod_č, jméno) |
     (rod_č, jméno, věk) | ...
   )>
```

Předpokládejte situaci, kdy existuje více atributů zaměstnance!

Přehled regularních výrazů v XML

- A — značka A se vyskytuje
- e1,e2 — výraz e1 následovaný e2
- e* — 0 nebo více výskytů e
- e? — nepovinné -- 0 nebo 1 výskytů
- e+ — 1 nebo více výskytů
- e1 | e2 — buď e1 nebo e2
- (e) — seskupování

Specifikace atributů v DTD

```
<!ELEMENT výška (#PCDATA)>
<!ATTLIST výška
    dimenze CDATA #REQUIRED
    přesnost CDATA #IMPLIED >
```

Atribut dimenze je požadován; atribut přesnost je nepovinný.

CDATA je “typ” atributu -- znamená string.

Specifikování atributů ID a IDREF

```
<!DOCTYPE rodina [
  <!ELEMENT rodina   (osoba)*>
  <!ELEMENT osoba  (jméno)>
  <!ELEMENT jméno    (#PCDATA)>
  <!ATTLIST osoba
                    id        ID        #REQUIRED
                    matka     IDREF     #IMPLIED
                    otec      IDREF     #IMPLIED
                    děti      IDREFS    #IMPLIED>
]>
```

Dobře vytvořený dokument mající DTD a vyhovující tomuto DTD se nazývá platný (validní).

Některá validní data

```
<rodina>
    <osoba  id="jana"  matka="marie" otec="josef">
        <jméno> Jana Nováková </jméno>
    </osoba>
    <osoba id="josef" děti="jana vít">
        <jméno> Josef Novák </jméno>
    </osoba>
    <osoba id="marie" děti="jana  vít">
        <jméno> Marie Nováková </jméno>
    </osoba>
        <osoba  id="vít"  matka=”marie" otec="josef">
        <jméno> Vít Novák </jméno>
    </osoba>
</rodina>
```

Konzistence hodnot ID a IDREF(S) atributů

- Je-li atribut deklarován jako ID
  - asociované hodnoty musí být v dokumentu všechny různé
- Je-li atribut deklarován jako IDREF
  - asociovaná hodnota musí existovat jako hodnota nějakého ID atributu (žádné visící “ukazatele”) v daném dokumentu
- podobně pro všechny hodnoty atributu IDREFS
- Atributy typu ID, IDREF a IDREFS *nejsou typovány*

DTD vs. db schémata (nebo typy)

- Z hlediska standardů databází (nebo programovacího jazyka) jsou DTD spíše slabé specifikace.
  - pouze jeden základní typ -- PCDATA
  - žádné užitečné “abstrakce” (např. množiny)
  - IDREF jsou netypované. Ukazuje se na něco, ale neví se na co!
  - žádná IO, např., dítě je inverzní k rodičům
  - žádné metody
- Návrhy na rozšíření XML: schémata, IO
  - DCD (Document Content Desription)
  - XML Schema (návrh W3C)
  - Microsoft v Exploreru 5.

Část III: Dotazovací jazyky

XML-QL

http://www.w3.org/TR/NOTE-xml-ql

http://db.cis.upenn.edu/XML-QL/

XPath

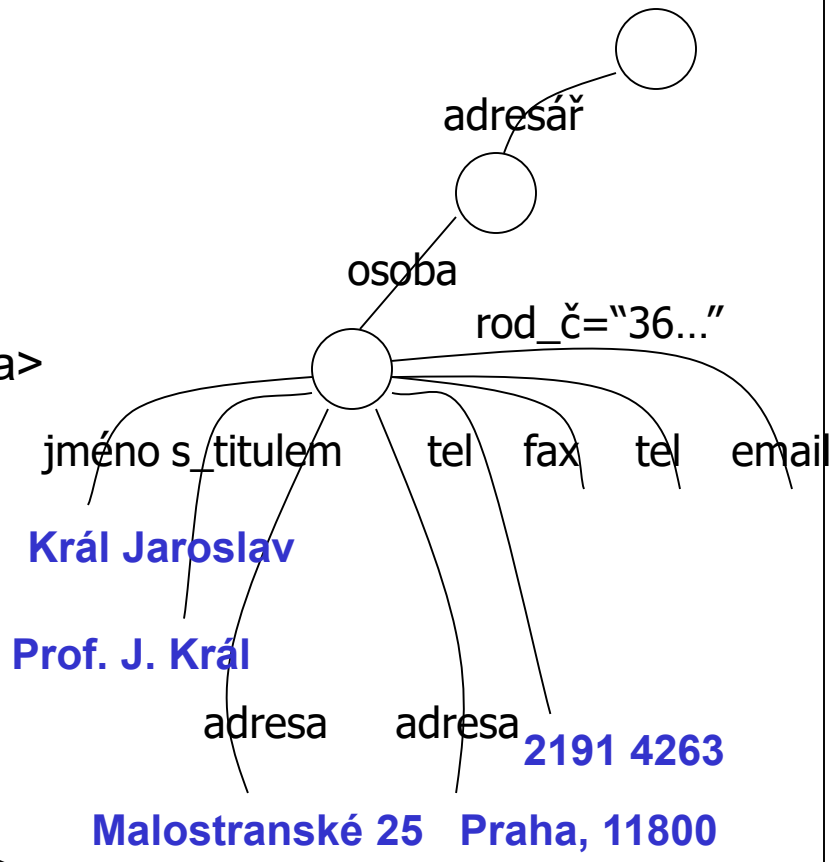
http://www.w3.org/TR/xpath

XML-QL: datový model

- orientovaný ohodnocený graf
- značky jsou reprezentovány jako ohodnocení *hran*
- ohodnocení uzlů jsou množinami dvojic jméno atributu/hodnota
- 2 modely: uspořádaný a neuspořádaný

XML-QL - datový model (pokr.)

```
<osoba rod_č="360528/281"
    <jméno> Král Jaroslav </jméno>
    <s_titulem> Prof. J. Král </s_titulem>
    <adresa> Malostranské 25 </adresa>
    <adresa> Praha, 100 00 </adresa>
    <tel> 2191 4263 </tel>
    <fax> 2191 4323 <fax>
    <tel> 2191 4323 </tel>
    <email> kral@ksi.mff.cuni.cz </email>
</osoba>
```



XML-QL (XML Query Language)

- Proč dotazovací jazyk? Extrakce, restrukturalizace, integrace, listování…
- požadavky:
  - vyhledávání pomocí klíčových slov (podobně jako v systémech úplných textů),
  - navigace podle struktury značek XML,
  - silný přístup ke strukturovaným datům podobný např. možnostem SQL.
  - techniky založené na pojmu podobnost dokumentů či vzdálenosti (proximity) mezi termy.

XML-QL (XML Query Language)

- návrh konsorcia W3, August 1998

- autoři:
  - Mary Fernandez — AT&T
  - Dana Florescu — INRIA
  - Alon Levy — Univ. of Washington
  - Dan Suciu — AT&T
  - Alin Deutsch — Univ. of Pennsylvania

XML-QL: výraz určující cestu

jednoduchá cesta specifikuje jeden krok v navigaci v db.

x/l

cesta je seznam jednoduchých cest.

x/l/k

Klíčový pojem: regulární výraz určující cestu

Př.: biblio/(zpráva | článek) -- větvení

autor/křestní_jméno? -- částečná informace

zpráva/reference*/autor -- Kleeneho uzávěr

biblio/_*/autor

Pz.: R+, kde R je regulární výraz, je ekvivalentní R/R*

zástupné znaky

Revidovaný adresář

```
<adresář>
  <osoba rod_č="420529/102">
      <jméno> Říha Antonín </jméno>
      <s_titulem> Dr. A. Říha </s_titulem>
      <adresa> Malostranské 25 </adresa>
      <adresa> Praha, 100 00 </adresa>
      <tel> 2191 4268 </tel>
      <fax> 2191 4323 <fax>
      <tel> 2191 4323 </tel>
      <email>riha@ksi.mff.cuni.cz </email>
  </osoba>
</adresář>
```

XML-QL: porovnávání vzorků

D1.: Nalezni Říhovu e-mailovou adresu:

```
where  <adresář>
          <osoba>
                <jméno> Říha Antonín </jméno>
                <email>$e</email>
          </osoba>
       </adresář> in "http://kocour.mff.cuni.cz/~honza/adresa.xml"
construct  $e
```

<XML> riha@ksi.mff.cuni.cz </XML>

Selekce (extrakce) dat

XML-QL: konstrukce nových XML dat

D2: Koho můžeme elektronicky kontaktovat?

```
where     <adresář>
            <osoba>
              <s_titulem>$g</s_titulem>
              <email>$e</email>
            </osoba>
          </adresář> in "http://..."
construct <e-kontakt>
            <kdo>$g</kdo>
            <kde>$e</kde>
          </e-kontakt>
```

```
<XML>
  <e-kontakt>
    <kdo> Dr. A. Říha </kdo>
    <kde>riha@ksi.mff.cuni.cz
    </kde>
  </e-kontakt>
  <e-kontakt>
    <kdo>Prof. J. Král</kdo>
    <kde> kral@ksi.mff.cuni.cz
    </kde>
  </e-kontakt>
  ...
</XML>
```

Restrukturalizace dat

XML-QL: spojení

D3: Kdo z našich kontaktů byl zapojen ve filmu?

```
where   <adresář>
          <osoba>
            <s_titulem>$g</s_titulem>
            <email>$e</email>
          </osoba>
        </adresář> in "http://…adresa.xml"
        <film><název>$t</>
                <jméno>$g</>   //jde o jméno herce
        </film>  in "http://www.barrandov.com"
construct  <film-kontakt>
             <kdo>$g</kdo>
             <film>$t</film>
             <kde>$e</kde>
           </film-kontakt>
```

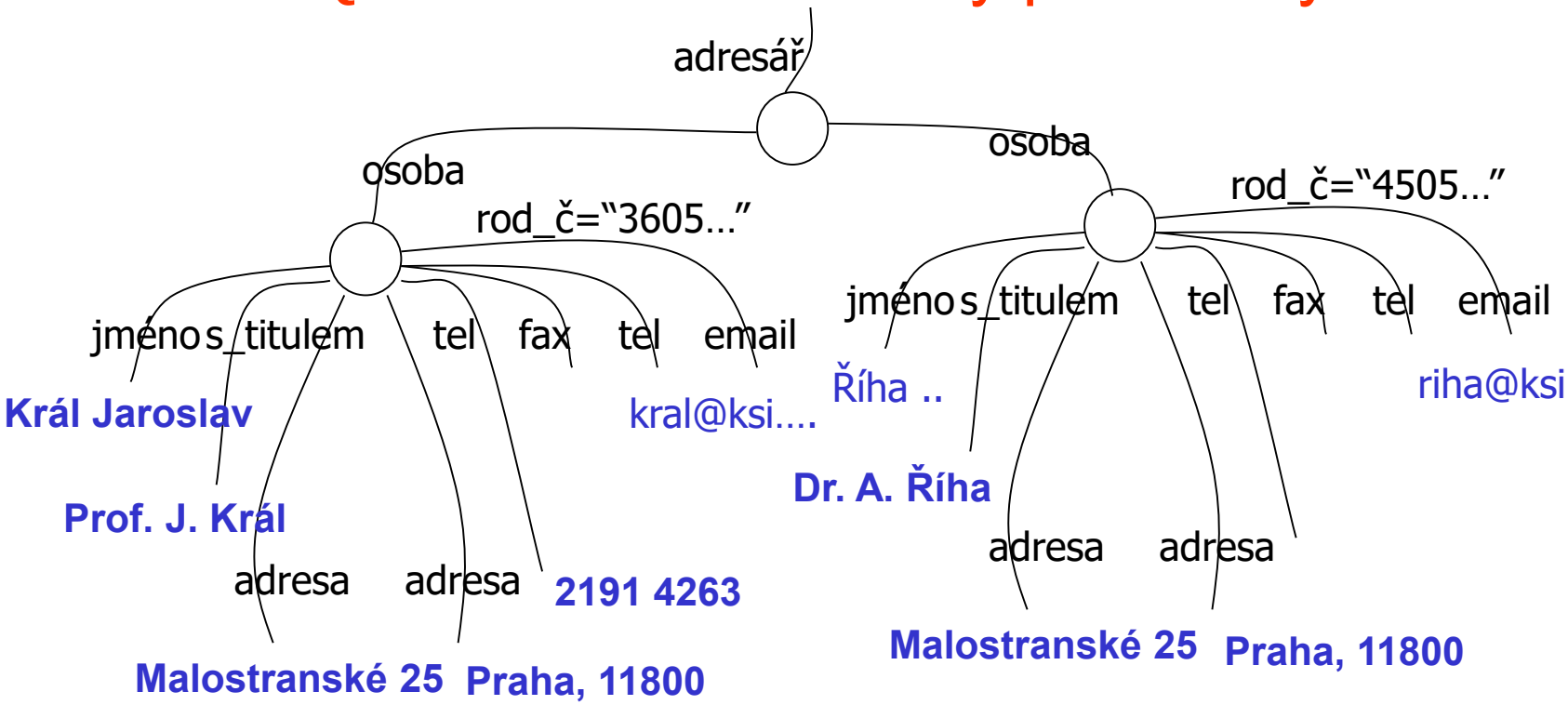
XML-QL: spojení (pokr.)

```
<XML>
  <film-kontakt>
    <kdo> Prof. J. Král </kdo>
    <kde> kral@ksi.mff.cuni.cz </kde>
    <film>Král Šumavy</film>
  </film-kontakt>

  <film-kontakt>
    <kdo>Dr. D. Húsek</kdo>
    <kde>husek@ui.cas.cz</kde>
    <film>Jakpak je dnes u nas doma</film>
  </film-kontakt>
...
</XML>
```

Integrace dat

XML-QL sémantika: vazby proměnných




```
where <adresář>
        <osoba>
          <jméno>$n</>
          <email>$e</>
        </>
      </>
```

$n	$e
Král Jaroslav	kral@ksi.mff…
Říha Antonín	riha@ksi.mff….

XML-QL pro pokročilé

Značkové a atributové proměnné

D4.: Najdi značky podelementů elementů osoba :

```
where      <adresář.osoba.$tag></>
             in "http://kocour.mff.cuni.cz/~honza/adresa.xml"
construct  <elementy_osoby>$tag</>
```

D5.: Najdi atributy elementů osoba:

```
where      <_*.osoba $jm_atr=$h></>
             in "http://kocour.mff.cuni.cz/~honza/adresa.xml"
construct  <atr_osoby>
             <jméno>$jm_atr</>
             <hodnota>$h</>
           </>
```

Listování ve schématu

XML-QL pro pokročilé

Regulární výrazy určující cestu

D6: Najdi všechny emailové adresy a faxová čísla:

```
where      <adresář._*.(email | fax)>$ef</>
             in "http://kocour.mff.cuni.cz/~honza/adresa.xml"
construct  <email_Fax>$ef</>
```

Integrace heterogenních dat

XML-QL pro pokročilé

D3‘: Kdo z našich kontaktů byl zapojen ve filmu?

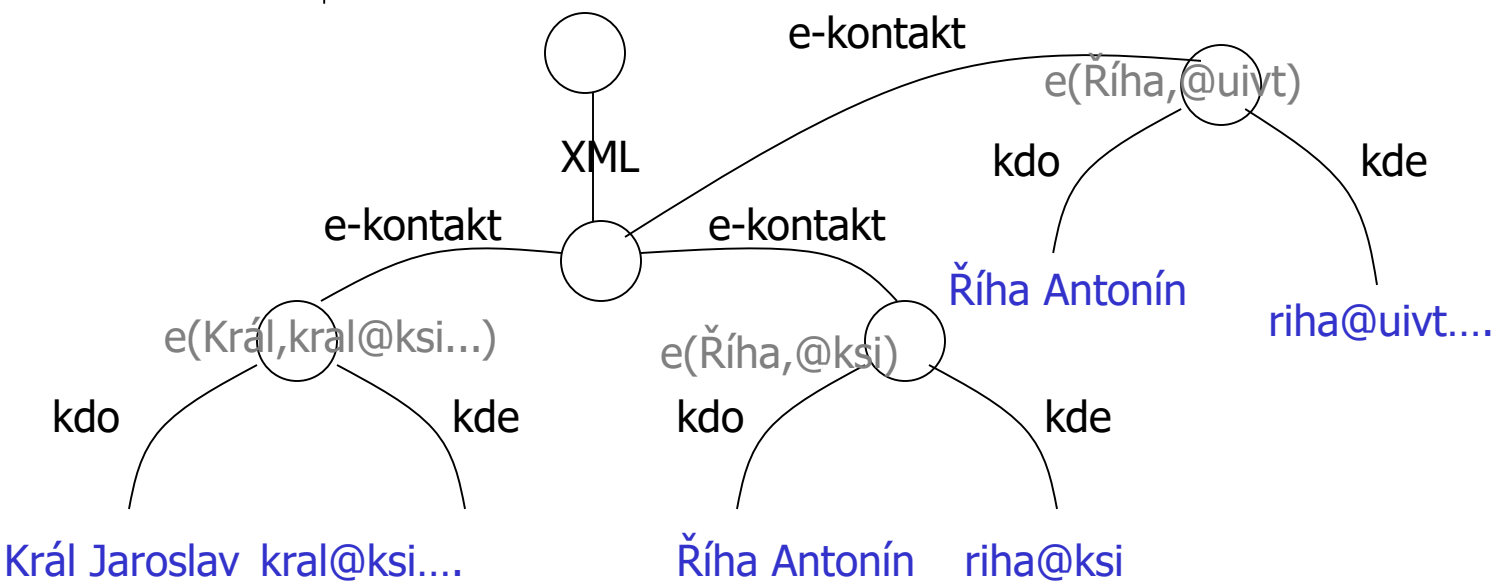
```
where     <adresář>
            <osoba>
              <s_titulem>$g</>
              <email></> ELEMENT_AS >$e
            </osoba>
          </adresář> in “http://…adresa.xml”
          <film><název></> ELEMENT_AS $t
                <jméno>$g</>   //jde o jméno herce
          </film>  in “http://www.barrandov.com”
construct  <film-kontakt>
                <kdo>$g</kdo>
                $t $e
          </film-kontakt>
```

Přenos elementů

XML-QL sémantika: výstup v XML

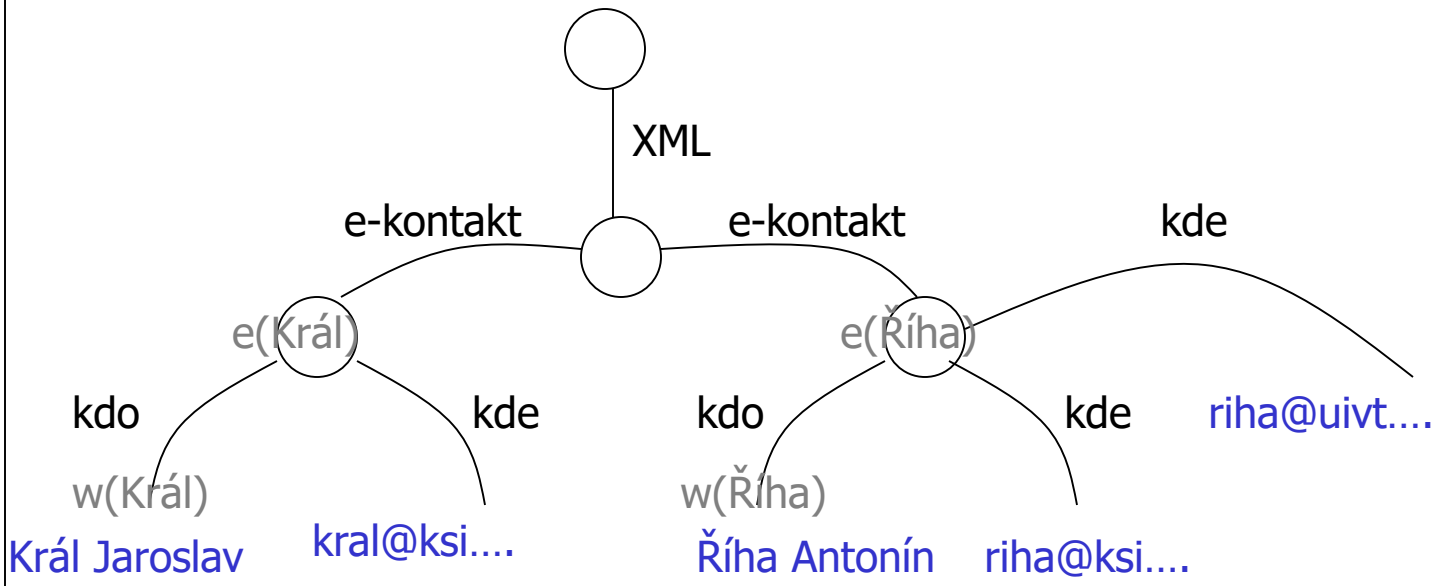
$n	$e
Král Jaroslav	kral@ksi.mff.cuni.cz
Říha Antonín	riha@ ksi.mff.cuni.cz
Říha Antonín	riha@uivt.cas.cz

```
construct <e-kontakt>
            <kdo>$n</kdo>
            <kde>$e</kde>
          </e-kontakt>
```



XML-QL: Skolemovy funkce

```
construct  <e-kontakt ID=e($n) >
              <kdo ID=w($n) >$n</kdo>
              <kde>$e</kde>
           </e-kontakt>
```



XPath

- XPath je syntaxe použitá pro výběr částí XML dokumentu
- Možnosti výsledku vyhodnocení výrazu XPath:
  - množina uzlů,
  - číslo,
  - boolská hodnota,
  - řetězec.

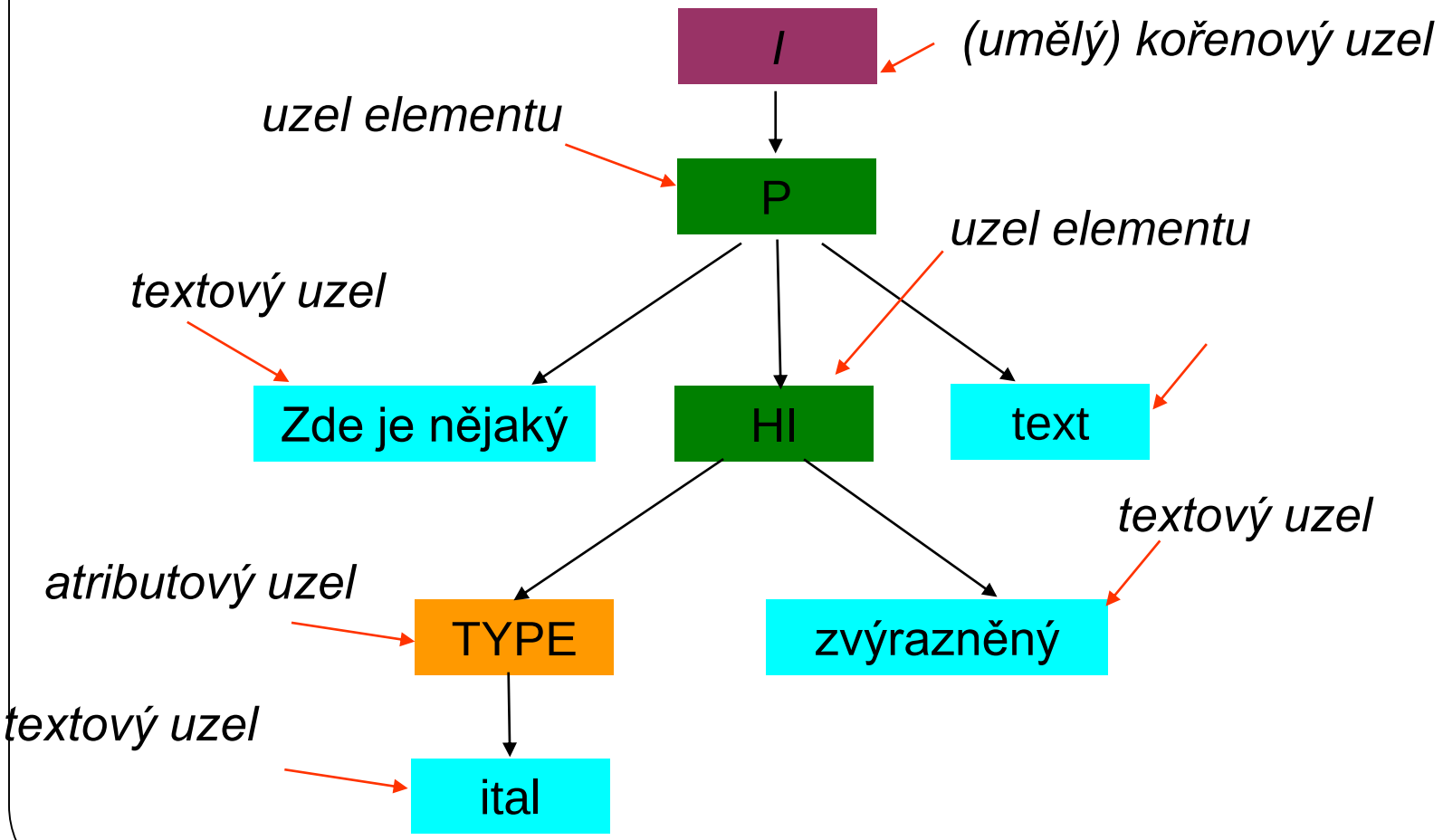
jednoduchá cesta založená na relaci předci-potomci

cesta je seznam jednoduchých cest, např. x/l/k

  - Každý krok je vyhodnocen v rámci nějakého kontextu.
  - Výsledkem každého kroku je množina uzlů. Výsledky nějakého kroku vstupují jako kontext do následujícího kroku.

- Pz: vyhodnocovací algoritmy v PTIME v $|D|$ a $|DB|$

Model používaný v XPath



XPath

Dotazy využívají různé relace mezi uzly (osy v XPath)

ancestor (předci) – uzly ležící na cestě od *u* do kořenu,

ancestor-or-self (předci včetně mne) – *u* a uzly ležící na cestě od *u* do kořenu,

parent (rodič) – první uzel ležící na cestě od *u* do kořenu,

child (děti) – bezprostřední následníci uzlu *u*,

descendant (potomci) - všechny uzly, pro které je uzel *u* předkem,

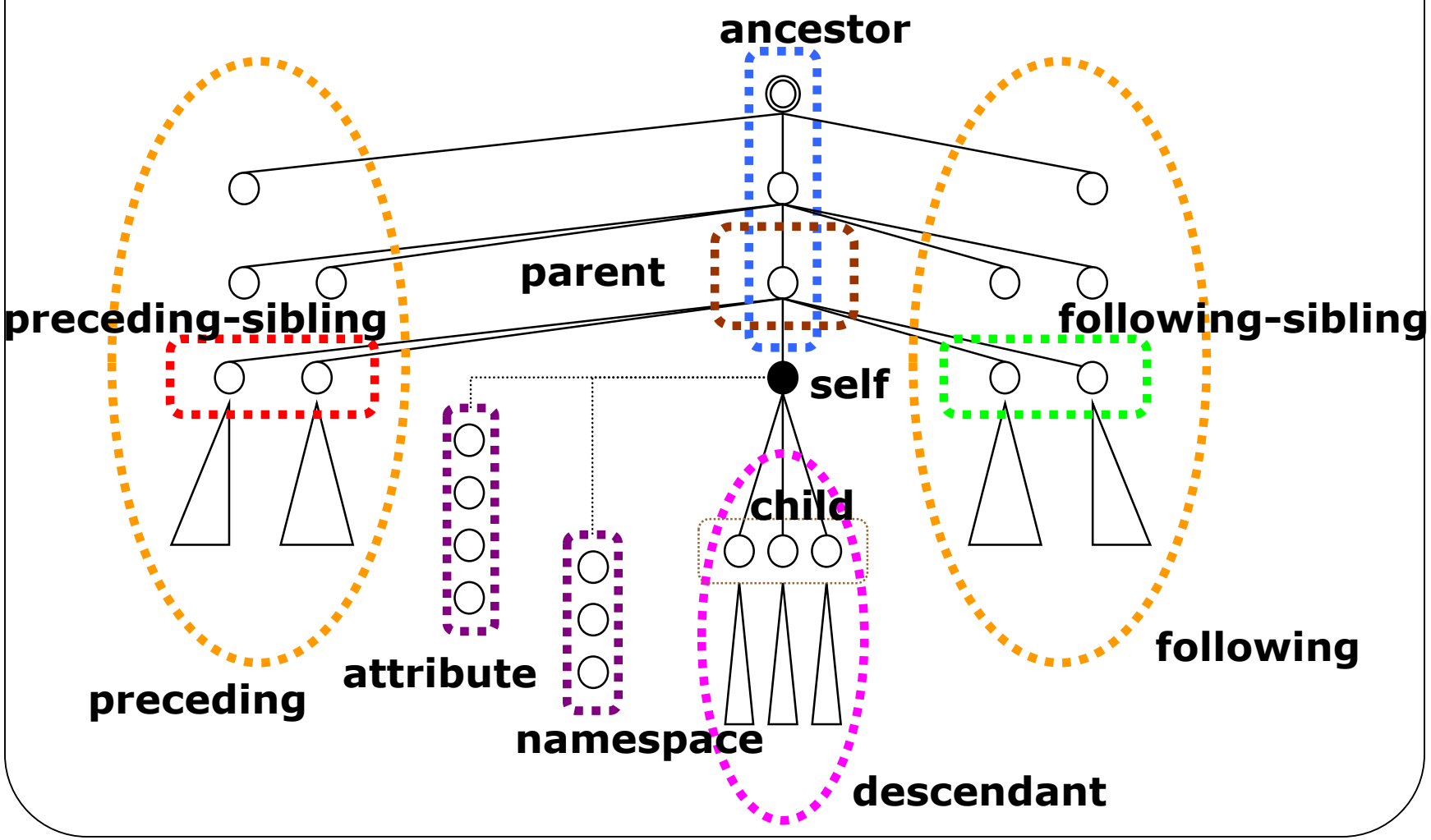
preceding-sibling (předcházející sourozenci) – sourozenci uzlu *u* předcházející *u* při průchodu stromem doleva-do-hloubky,

following-sibling (následující sourozenci) – sourozenci uzlu *u* následující *u* při průchodu stromem doleva-do hloubky,

preceding (předchůdci) – uzly předcházející *u* (kromě jeho předků) při průchodu stromem do doleva-hloubky,

following (následníci) – uzly následující po *u* (kromě jeho potomků) při průchodu stromem doleva-do hloubky.

XPath


ancestor
parent
preceding-sibling
following-sibling
self
child
attribute
namespace
preceding
following
descendant

XPath

Klíčový pojem: regulární výraz určující cestu

podobně jako v XML-QL

- Zde: jde o relativní cestu začínající z běžného (kontextového) elementu.
- Cesta, která začíná / reprezentuje absolutní cestu, začínající z kořene XML dat.

  /kniha
- Cesta začínající // může začínat *kdekoliv* v dokumentu.

  //nadpis vybere každý element nadpis, který se vyskytuje v dokumentu

XPath

Příklady dotazů:

/článek/*/odstavec --- * jako zástupný symbol

článek//obrázek --- // zkratka za relaci potomci

//článek[autor='Jiří Kosek']

Složitější:

//článek[název = 'Lehký úvod do XML']/autor

článek[autor]//název

??

obrázek/ancestor::kapitola/following-sibling::kapitola

XPath

Příklady dotazů:

/článek/*/odstavec --- * jako zástupný symbol

článek//obrázek --- // zkratka za relaci potomci

//článek[autor='Jiří Kosek']

Složitější:

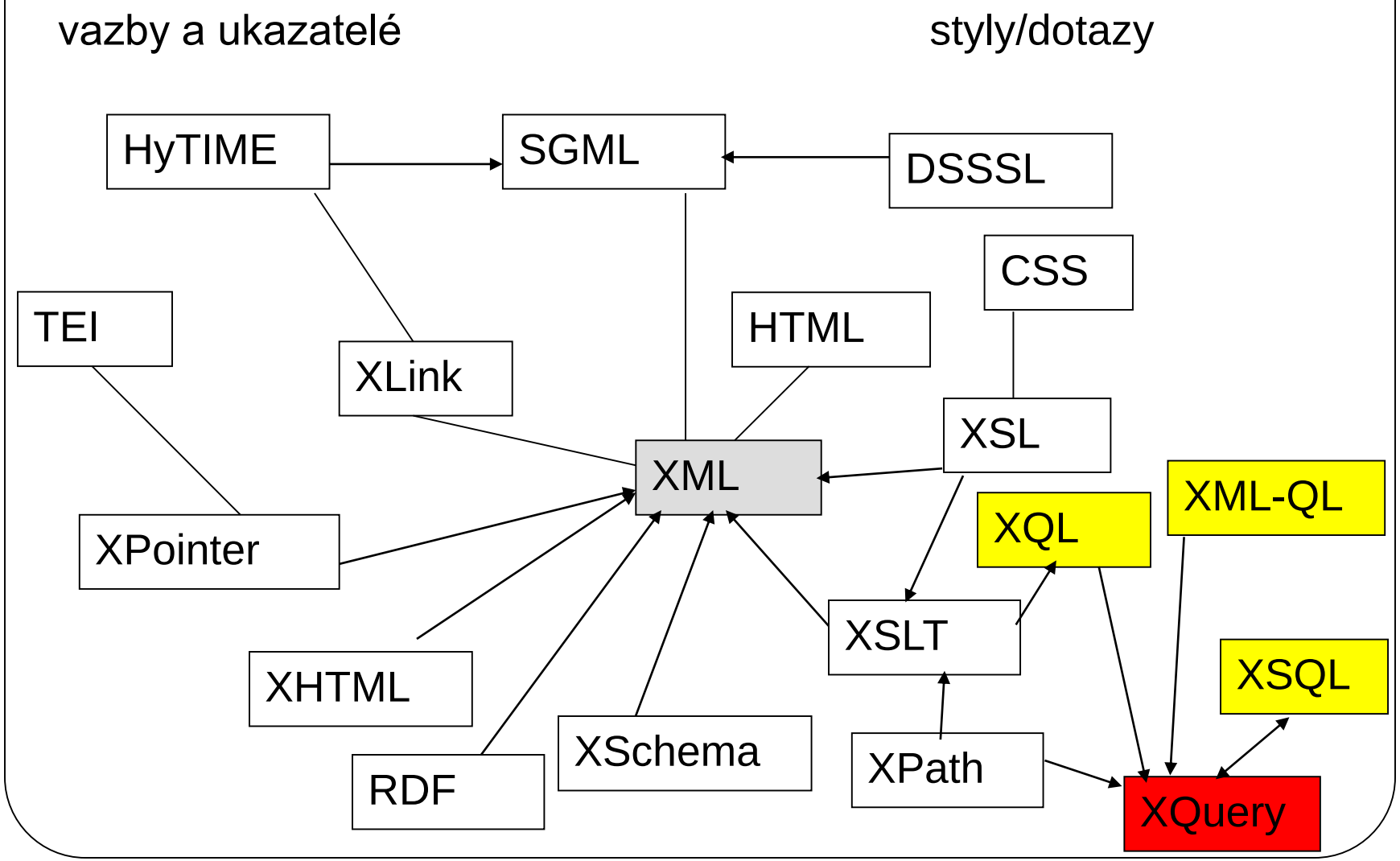
//článek[název = 'Lehký úvod do XML']/autor

článek[autor]//název

obrázek/ancestor::kapitola/following-sibling::kapitola

Vybere kapitolu, která je následujícím sourozencem kapitoly, ve které je obrázek

XML - rodina standardů



Část IV: SQL/XML

http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=53686

SQL/XML

- Rozšíření jazyka SQL umožňující pracovat s XML daty
- Dříve: pomocí CLOB (BLOB) ⇒ nutnost konverzí, neefektivnost
- Část standardu ANSI/ISO SQL2003, dokončeno v SQL:2008
  - nový vestavěny typ dat XML
- Podporován (někdy jen částečně): ORACLE, DB2, DataDirect, PostgreSQL 9.1, MS SQL Server
- ! SQL/XML ≠ SQLXML (proprietární technologie společnosti Microsoft, která se používá v MS SQL Serveru)

Mapovací pravidla

Pro zobrazení

- Nutnost zobrazení SQL „věcí" na XML „věci„ (a naopak).
  - SQL identifikátorů na XML jména
  - Unicode na množiny znaků SQL
  - XML jmen na SQL identifikátory
  - SQL typů na typy jazyka XML Schema
  - SQL hodnot na XML hodnoty
  - SQL tabulek, schémat a katalogů na XML hodnoty
- požadavky na přizpůsobení (vynecháme)

Mapování tabulek SQL, schémat a katalogů do XML dokumentů

- Generují se: XML Schema dokument jako XML dokument
- Schéma tabulky na XML schema dokument
- Tabulka dvojím způsobem:
  - jeden element nebo
  - posloupnost elementů

Mapování tabulek SQL, schémat a katalogů do XML dokumentů

- Element a podlementy

```
<zam>
    <row>
      <id> 1001 </id>
      <příjmení> Novák </příjmení>
    ...
    </row>
    <row>
      <id> 1206 </id>
      <příjmení> Dvořák </příjmení>
    ...
    </row>
</zam>
```

Mapování tabulek SQL, schémat a katalogů do XML dokumentů

- Posloupnost elementů

```
<zam>
    <id> 1001 </id>
    <příjmení> Novák </příjmení>
    ...
</zam>
<zam>
    <id> 1206 </id>
    <příjmení> Dvořák </příjmení>
    ...
</zam>
```

Mapování tabulek SQL, schémat a katalogů do XML dokumentů

- Doporučení pro tabulku: jeden element s podelementy

```
<Zaměstnanci>
   <row>
     <id> 1001 </id>
     <křestní jméno>Josef</křestní jméno>
     <příjmení>Novák</příjmení>
     <datum_nástupu>2000-05-24<datum_nástupu>
     <odd>osobní</odd>
   </row>
   <row>
     <id> 1206 </id>
     <křestní jméno>Josef</křestní jméno>
     <příjmení> Dvořák </příjmení>
                …
   </row>
                …
</Zaměstnanci>
```

Typ XML

- typ XML může být použit, kdykoliv je dovolen SQL typ dat - jako typ sloupce tabulky, parametr programu, atribut UDT, komponenta řádku nebo jako SQL proměnná.
- Hodnoty typu XML jsou různé od své textové reprezentace
- Sémantika operací na hodnotách typu XML je specifikována za předpokladu stromové reprezentace založené na XML Information Set Recommendation (Infoset).
- Model Infoset je modifikován pouze v jedné význačné věci: popis dokumentu (document information item) je nahrazen novou informační položkou – informace o kořenu XML (XML root information).

Typ XML

informační položka v SQL/XML je kterákoliv z následujících:

- informace o kořenu XML (je definována v SQL/XML)
- popis atributu (viz Infoset)
- znaková položka (viz Infoset)
- komentář (viz Infoset)
- popis DTD (viz Infoset)
- popis elementu (viz Infoset)
- popis prostoru jmen (viz Infoset)
- popis notace (viz Infoset)
- popis instrukce (viz Infoset)
- odkaz na neexpandovanou entitu (viz Infoset)
- neanalyzovaná položka (viz Infoset)

Terminologie Infosetu: každá informační položka má vlastnosti

Typ XML

- informace o kořenu XML je položka podobná položce popis dokumentu Infosetu s jedním rozdílem:
  - zatímco XML Infoset kořen povoluje přesně jeden dětský element, *informace o kořenu XML* dovoluje 0 nebo více položek *popis elementu*.
- XML hodnota je buď hodnota NULL, nebo informace o kořenu XML včetně všeho, co (rekurzivně zahrnuje)

  $\Rightarrow$ ne každá XML hodnota je XML dokument (může mít více kořenů)

Operátory pro typ XML

SQL → XML

- XMLCOMMENT vytvoří XML comment uzel
- XMLELEMENT vytvoří z SQL hodnot XML element
- XMLFOREST vytvoří posloupnost XML elementů z názvů sloupců tabulky
- XMLPI vytvoří uzel instrukce
- XMLTEXT vytvoří textový uzel
- XMLPARSE mapuje SQL text na XML hodnotu
- XMLAGG agreguje XML hodnoty do skupin – XML lesu

→ XML

- XMLQUERY vyhodnotí XQUERY výraz

Operátory pro typ XML

XML → XML

- XML document vytvoří uzel dokumentu z XML hodnoty
- XMLCONCAT zřetězí XML hodnoty a vrací výslednou XML hodnotu
- XMLVALIDATE validuje XML hodnotu oproti schématu a dodá kopii této hodnoty

XML→ SQL

- XMLTABLE transformuje XQuery výsledek na SQL tabulku
- XMLITERATE transformuje XQuery posloupnost na tabulku SQL
- XMLSERIALIZE transformuje XML hodnotu na SQL text

Predikáty pro typ XML

- IS DOCUMENT pro testování, zdali XML hodnota má jediný kořenový element.

XMLELEMENT

XMLELEMENT vytváří XML hodnotu k

- identifikátoru SQL, který se stane jménem elementu,
- volitelném seznamu deklarací prostoru jmen,
- volitelnému seznamu pojmenovaných výrazů, které poskytují jména a hodnoty atributů, a
- volitelný seznam výrazů, který poskytuje obsah elementu.

```
SELECT Z.id,
   XMLELEMENT (NAME "zam",
   Z.křestní_jméno || ' ' || Z.příjmení) AS xhodnota
FROM Zaměstnanci Z WHERE ...
```

==>

id	xhodnota
1001	<zam> Josef Novák </zam>
...	

XMLATTRIBUTES

- specifikace atributů je uzávorkována klíčovým slovem XMLATTRIBUTES a musí se vyskytovat jako 3. argument, jestliže je specifikována deklarace prostoru jmen, a jako 2. argument v opačném případě.
- každý atribut může být pojmenován implicitně nebo explicitně.

```
SELECT Z.id,
   XMLELEMENT (NAME "zam",
     XMLATTRIBUTES (Z.id AS "zamid"),
     Z.křestní_jméno || ' ' || Z.příjmení
) AS xhodnota
FROM Zaměstnanci Z WHERE ...
```

==>

id	xhodnota
1001	<zam zamid="1001"> Josef Novák </zam>
...	

XMLELEMENT

- XMLELEMENT může konstruovat hnízděnou strukturu elementu.

```
SELECT Z.id,
   XMLELEMENT (NAME "zam",
     XMLELEMENT (NAME "jméno",
              Z.křestní_jméno || ' ' || Z.příjmení ),
     XMLELEMENT (NAME "datum_nástupu", Z.nástup)
) AS xhodnota
FROM Zaměstnanci Z WHERE ...
```

==>

id	xhodnota
1001	<zam> <jméno>Josef Novák</jméno> <datum_nástupu>2000-05-24</datum_nástupu> </zam>

XMLELEMENT

- XMLELEMENT může konstruovat smíšený obsah.

```
SELECT Z.id,
   XMLELEMENT (NAME "zam", ‚Zaměstnanec ',
     XMLELEMENT (NAME "jméno",
                 Z.křestní_jméno || ' ' || Z.příjmení ), 'nastoupil ',
     XMLELEMENT (NAME "datum_nástupu", Z.nástup)
) AS xhodnota
FROM Zaměstnanci Z WHERE ...
```

==>

```
id                          xhodnota
1001    <zam>
                Zaměstnanec <jméno> Josef Novák </jméno>
                nastoupil
                <datum_nástupu> 2000-05-24 </datum_nástupu>
            </zam>
```

Zaměstnanci(id, křestní_jméno, příjmení,odd, nástup,…)
Děti(…,rodič,…)

XMLELEMENT

- XMLELEMENT může mít jako argumenty skalární poddotazy.

```
SELECT Z.id,
   XMLELEMENT (NAME "zam",
     XMLELEMENT (NAME "jméno", Z.křestní_jméno || ' ' ||
                                Z.příjmení ),
     XMLELEMENT (NAME "děti",
                 (SELECT COUNT (*) FROM děti d
                  WHERE d.rodič = Z.id ) ) ) AS xhodnota
FROM Zaměstnanci Z WHERE ...
```

==>

id	xhodnota
1001	<zam> <jméno> Josef Novák </jméno> <děti> 3 </děti> </zam>

XMLELEMENT

- V XMLELEMENT může být specifikována deklarace prostoru jmen.
- Musí se vyskytovat před specifikací atributů.
- Lexikální pravidla se aplikují dále pro hnízděné elementy.

Syntaxe deklarace prostoru jmen :

```
< deklarace prostoru jmen> ::=
   XMLNAMESPACES ( <XML prostor jmen> [ , .. ] )
<XML prostor jmen> ::=
      <URI> AS <prefix>
      | DEFAULT <URI>
      | NO DEFAULT
<URI> ::= <řetězec znaků>
<prefix> ::= <identifikátor>
```

XMLELEMENT

Příklad s deklarací prostoru jmen :

```
SELECT Z.id,
   XMLELEMENT (NAME "IBM:zam",
     XMLNAMESPACES ('http://a.b.c' AS IBM,
     XMLATTRIBUTES (Z.id AS "zamid"),
     Z.křestní_jméno || ' ' || Z.příjmení)
) AS xhodnota
FROM Zaměstnanci Z WHERE ...
```

==>

id	xhodnota
1001	<IBM:zam xmlns:IBM="http://a.b.c" zamid="1001">Josef Novák </IBM:zam>

Složka OPTION

- Specifikuje přídavné volby pro konstrukci obsahu XML elementu.
- Specifikuje co bude výsledkem, když výraz *E* definující obsah elementu je NULL. Je to buď hodnota NULL nebo prázdný element. Volba má vliv pouze na zacházení s NULL obsahem elementu, netý ká se hodnot atributů. Volba se nedědí pro vnitřní volání funkce XMLELEMENT ve výrazu *E*.
  - EMPTY ON NULL
    - Je-li hodnota každého výrazu *E* NULL, vrací se prázdný element. EMPTY ON NULL je implicitní volba.
  - NULL ON NULL
    - Je-li hodnota každého výrazu *E* NULL, vrací se NULL.

XMLFOREST

- XMLFOREST konstruuje posloupnost XML elementů k volitelné deklaraci prostoru jmen a seznamu pojmenovaných výrazů jako argumentů.
- každý element může být pojmenován implicitně nebo explicitně.
- jestliže se nějaký z výrazů vyhodnotí jako NULL, je ignorován, jestliže se všechny výrazy vyhodnotí jako NULL, výsledkem je XML hodnota NULL.

XMLFOREST

```
SELECT Z.id,
XMLELEMENT (NAME "zam",
XMLFOREST (Z.křestní_jméno || ' ' || Z.příjmení AS
   "jméno", Z.nástup AS "nástup")
) AS xhodnota
FROM Zaměstnanci Z
WHERE ...
```

==>

id	xhodnota
1001	<zam> <jméno> Josef Novák </jméno> <nástup> 200-05-24 </nástup> </zam>

XMLFOREST

Je-li přítomna deklarace prostoru jmen, musí být prvním argumentem.

```
SELECT Z.id,
   XMLELEMENT (NAME "zam",
   XMLFOREST (
   XMLNAMESPACES (DEFAULT 'http://a.b.c', 'http://d.e.f' AS "xx"),
     Z.křestní_jméno || ' ' || Z.příjmení AS "xx:jméno", Z.nástup)
   ) AS xhodnota
FROM Zaměstnanci Z WHERE ...
```

==>

```
id                          xhodnota
1001          <zam>
                <xx:jméno xmlns:xx="http://d.e.f"> Josef
                                                   Novák</xx:jméno>
                <nástup xmlns="http:a.b.c"> 200-05-24 </nástup>
              </zam>
```

XMLCONCAT

- XMLCONCAT vytváří XML hodnotu danou dvěma nebo více výrazy typu XML.
- Jestliže je některý z výrazů vyhodnocen jako NULL, je ignorován. Jestliže jsou všechny výrazy vyhodnoceny jako NULL, je výsledkem XML hodnota NULL.

```
SELECT Z.id,
   XMLCONCAT (XMLELEMENT (NAME "jméno",
                          Z.křestní_jméno || ' ' || Z.příjmení),
     XMLELEMENT (NAME "datum_nástupu", Z.nástup)
   ) AS xhodnota
FROM Zaměstnanci Z WHERE ...
```

==>

id	xhodnota
1001	<jméno> Josef Novák </jméno> <datum_nástupu> 200-05-24 <datum_nástupu>

XMLAGG

- XMLAGG je agregační funkce podobná SUM, AVG, apod.
- Argument pro XMLAGG musí být výraz typu XML.
- Pro každý řádek ve skupině *G*, je výraz vyhodnocen a výsledné XML hodnoty jsou zřetězeny tak, že konstruují pro *G* jedinou XML hodnotu jako výsledek.
- pro uspořádání výsledku může být specifikována složka ORDER BY.
- Všechny NULL hodnoty jsou před zřetězením vynechány.
- jestliže všechny vstupy pro zřetězení jsou NULL nebo jestliže skupina je prázdná, výsledkem je hodnota NULL.

XMLAGG

Příklad:

```
SELECT XMLELEMENT
   (NAME "oddělení",
     XMLATTRIBUTES (Z.odd AS "jméno"),
     XMLAGG (XMLELEMENT (NAME "zam", Z.příjmení) )
   ) AS xhodnota
FROM Zaměstnanci Z
GROUP BY Z.odd;
```

==>

```
        xhodnota
<oddělení jméno=„osobní">
   <zam> Novák </zam>
   <zam> Dvořák </zam>
</oddělení>
....
```

XMLAGG

- pro uspořádání výsledku agregace může být použit ORDER BY.

```
SELECT XMLELEMENT
   (NAME „oddělení",
     XMLATTRIBUTES (Z.odd AS "jméno"),
     XMLAGG (XMLELEMENT (NAME "zam", Z.příjmení)
     ORDER BY Z.příjmení)
   ) AS xhodnota
FROM Zaměstnanci Z
GROUP BY Z.odd;
```

==>

```
                    xhodnota
     <oddělení jméno=„osobní">
       <zam> Dvořák </zam>
       <zam> Novák </zam>
     </oddělení>
```

XMLQUERY

- V pseudofunkci se využívá jazyk XQuery
- Možnosti XQuery
  - FLWOR výrazy: FOR / LET / WHERE / ORDER BY/ RETURN
  - zahrnuje XPath 2.0 (např. //oddělení[@tel = 4241])
  - konstruktory elementů (<zákazník>)
  - operátory zachovávající uspořádání:
    - uspořádání vstupu: FLWR
    - uspořádání dokumentu: výrazy XPath
  - statické typování:
    - detekce chyb se dělá dříve
    - lepší provoz
  - silné typování při existenci schématu, slabé typování bez schématu

Dotazování - příklad

ZaměstnanciXML	id	zaměstnanecXML
	1001	<zaměstnanec> <křestní jméno>Josef</křestní jméno> <příjmení>Novák</příjmení> <nástup>2000-05-24</nástup> <oddělení tel=4241>mzdové</oddělení> </zaměstnanec>
	1006	<zaměstnanec> <příjmení >Dvořák</příjmení> <nástup>2001-04-23</nástup> <oddělení tel=4231>osobní</oddělení> </zaměstnanec>

křestní jméno nemusí být známo

XMLQUERY

- vrací XML hodnotu (pro každý řádek)

```
SELECT
        XMLQUERY(
                'for $p in $sloupec/zaměstnanec return $p/příjmení'
        PASSING zaměstnanecXML AS "sloupec"
        RETURNING CONTENT
        NULL ON EMPTY
                        ) AS výsledek
FROM ZaměstnanciXML WHERE …
```

==>

výsledek
<příjmení>Novák</příjmení> <příjmení>Dvořák</příjmení>

XMLQUERY

Syntaxe: XMLQUERY(

XQuery-expr [PASSING {BY REF | BY VALUE} argument-list] [RETURNING {CONTENT | SEQUENCE} {BY REF | BY VALUE}]] NULL ON EMPTY | EMPTY ON EMPTY

)

- XQuery-expr je literál obsahující výraz v jazyku XQuery
- argument-list je seznam obsahující položky

  value-expr AS indentifier [BY REF | BY VALUE]

  vyjadřující vazbu mezi SQL hodnotou a globální proměnnou v XQuery-expr.
- RETURNING CONTENT BY VALUE vrací SQL serveru serializovanou hodnotu výsledku.
- výsledky ve tvaru posloupností s použitím BY REF zachovávají id uzlů.
- S PASSSING se definuje, jak se předává SQL hodnota přes globální proměnnou uvedenou v XQuery-expr.

XMLQUERY

```
SELECT XMLQUERY (
   'declare namespace c = "urn:příklad/zam";
   for $xc in /c:zaměstnanecXML
   where $xc//c:křestní jméno
   return
      <zaměstnanec telefon="{$xc//oddělení/@tel}">{
         $xc/c:příjmení,
         $xc/c:oddělení
     }</zaměstnanec>'
   PASSING BY REF zaměstnanecXML
   RETURNING CONTENT BY VALUE)
FROM ZaměstnanciXML
```

XMLTABLE

- Vezme XML data a produkuje výsledek v tabulce, která se použije ve složce FROM, jako odvozená relace.

```
SELECT výsledek.*
FROM ZaměstnanciXML, XMLTABLE(
                    'for $p in $sloupec/zaměstnanec return $p/příjmení'
        PASSING ZaměstnanciXML.zaměstnanecXML AS "sloupec"
) AS výsledek
 ==>
```

výsledek	output
	<příjmení>Novák</příjmení> <příjmení>Dvořák</příjmení>

- Každý řádek výsledku vznikne z jednoho členu posloupnosti navrácené vyhodnocením XQuery-express (viz syntaxe).

XMLTABLE

Syntaxe: XMLTABLE(
[namespace-declaration,]
XQuery-expression
[PASSING argument-list]
COLUMNS XMLtbl-column-definitions
)

- argument-list a XQuery-expression znamenají totéž co u XMLQUERY (prvky v argument-list jsou procházeny odkazem).
- XQuery-expression vyjadřuje obsah pro každý generovaný řádek. Výrazu se také říká vzor řádku.
- XMLtbl-column-definitions je seznam definic sloupců. Ty jsou dvojího druhu:

  column-name FOR ORDINALITY /*vrací sekvenční pozici*/

  a column-name data-type
  [BY REF | BY VALUE]
  [default-clause] /*když cesta nevrací hodnotu*/
  [PATH XQuery-Expression]

XMLTABLE

Přepoklad: pro Dvořáka neznáme křestní jméno

```
SELECT výsledek.*
FROM ZaměstnanciXML, XMLTABLE(
  'for $p in $sloupec/zaměstnanec return $p'
   PASSING ZaměstnanciXML.zaměstnanecXML AS "sloupec"
   COLUMNS
   "jméno" VARCHAR(40) PATH 'křestní jméno' DEFAULT 'není
   známo'
   "příjmení" VARCHAR(40) PATH 'příjmení'
   ) AS výsledek
```

==>

výsledek	jméno	příjmení
	Josef ‘není známo’	Novák Dvořák

XMLEXISTS

- je typu Boolean, její argumentem je XQuery výraz. Vrací FALSE, je-li výsledkem vyhodnocení výrazu prázdná posloupnost, UNKNOWN v případě NULL, v ostatních případech TRUE. Použití: za WHERE

```
SELECT id
FROM ZaměstnanciXML
WHERE XMLEXISTS('/zaměstnanec/nástup lt "2001-04-23" '
                PASSING BY VALUE zaměstnanec)
```

Id
1001
1006

- XMLEXISTS nepřidává k jazyku novou funkčnost (lze pomocí XMLQUERY a/nebo XMLTABLE).

XMLQUERY + XMLEXISTS vs. XMLTABLE

- XMLQUERY + XMLEXISTS

```
SELECT id, XMLQUERY('$zam//oddělení[@tel < 4241]'
                    PASSING zaměstnanecXML AS "zam")
  FROM ZaměstnanciXML
  WHERE XMLEXISTS('$zam//oddělení[@tel < 4241]'
                    PASSING zaměstnanecXML AS "zam")
```

- XMLTABLE: efektivnější a méně redundatní

```
SELECT z.id, o.oddělení
  FROM ZaměstnanciXML z,
      XMLTABLE('$zam//oddělení [@tel < 4241]'
              PASSING z.zaměstnanecXML AS "zam"
              COLUMNS
                  "oddělení" XML BY REF PATH '.')
      AS o(oddělení)
```

Operace spojení XML hodnot

```
SELECT o.jméno_o, z.zaměstnanecXML
FROM ZaměstnanciXML z, OdděleníXML o
WHERE XMLEXISTS(‘ $zam//oddělení[@tel = $olinka]’
PASSING zaměstnanecXML AS “zam“, o.linka AS “olinka“)
```

nebo

```
SELECT o.jméno_o, z.zaměstnanecXML
FROM ZaměstnanciXML z, OdděleníXML o
WHERE  o.linka =
    XMLCAST(XMLQUERY(‘$zam//oddělení/@tel’
                    PASSING z.zaměstnanecXML AS “zam“)
          AS CHAR(4))
```

Potřeba unifikovat typy dat

ZaměstnanciXML(id, zaměstnanecXML)
OdděleníXML(jméno_o, linka, vedouci)

XMLROOT

- Pro výraz dávající XML hodnotu vytváří XMLROOT XML hodnotu s vlastnostmi: specifikovanou verzí a příznakem STANDALONE:

```
XMLROOT ( <výraz dávající XML hodnotu> ,
VERSION {<výraz dávající řetězec>| NO VALUE }
[, STANDALONE { YES | NO | NO VALUE } ]
```

Příklad:

```
INSERT INTO Zaměstnanci (id, xhodnota)
VALUES (1001,
XMLROOT (XMLPARSE (DOCUMENT '<zam> Josef
  Novák </zam>'), VERSION '1.0', STANDALONE YES)
```

Vazby s hostitelským jazykem

- XML hodnoty z prostředí SQL do hostitelského jazyka implicitně pomocí operátoru XMLSERIALIZE
- XML hodnoty z hostitelského jazyka do SQL prostředí, implicitně pomocí XMLPARSE

XMLPARSE

- XMLPARSE vytváří XML hodnotu danou řetězcovým výrazem SQL:

  XMLPARSE ({ DOCUMENT | CONTENT }
  <výraz dávající řetězec>
  [{ PRESERVE | STRIP } WHITESPACE])

- Je-li specifikováno DOCUMENT, <výraz dávající řetězec> se vyhodnocuje jako řetězec znaků, který je utvořen podle XML 1.0 tak, jak bylo modifikováno doporučením pro Namespaces; jinak nastane výjimka.

XMLPARSE

- Je-li specifikováno CONTENT, <výraz dávající řetězec> se musí vyhodnotit jako řetězec znaků, který je utvořen podle gramatiky získané z gramatiky XML 1.0 modifikované podle doporučení pro Namespaces s novým „počátečním symbolem", definovaným jako:

  XMLvalue ::= XMLDecl? content

  jinak nastane výjimka. .
- jestliže hodnota argumentu obsahuje připojené DTD, XMLPARSE
  - nahradí všechny odkazy na entity definované v daném DTD jejich expandovanou formou,
  - použije implicitní hodnoty definované v DTD.

XMLPARSE

Příklady:

- INSERT INTO Zaměstnanci (id, xhodnota) VALUES (1001, XMLPARSE (DOCUMENT '<zam> Josef Novák </zam>'))
- INSERT INTO Zaměstnanci (id, xhodnota) VALUES (1001, XMLPARSE (CONTENT 'Josef Novák'))
- INSERT INTO Zaměstnanci (id, xhodnota)

  VALUES (1001, XMLPARSE (CONTENT '<Jméno> Josef </Jméno> <Příjmení> Novák </Příjmení>'))

XMLPARSE

- Zacházení s bílými znaky – je řízeno buď informacemi ze specifikace elementu nebo explicitně pomocí XMLPARSE.
  - PRESERVE WHITESPACE: XMLPARSE zachová všechny bíle znaky v řetězci.
  - STRIP WHITESPACE: XMLPARSE vynechá všechny bílé znaky mezi dvěma sousedními značkami elementů, pokud tam neexistuje alespoň jeden jiný znak.

    STRIP WHITESPACE je implicitní volba

XMLPARSE

Příklad:

INSERT INTO Zaměstnanci (id, xhodnota)

VALUES (1001, XMLPARSE (CONTENT :hv STRIP WHITESPACE))

Nechť :hv obsahuje hodnotu:

```
<?xml    standalone='1.0'    ?>

<fajn/>

Ahoj

<a atribut=' '>
  <přání>     </přání>
  Alenko!
</a>
```

XMLPARSE

XMLPARSE bude zacházet s hodnotou v :hv jako by byla ekvivalentní

```
<?xml standalone='1.0'?>
<fajn/>

Ahoj

<a atribut=' '><přání></přání>
  Alenko!
</a>
```

XMLSERIALIZE

- XMLSERIALIZE vytváří řetězcovou hodnotu SQL na základě výrazu dávajícího XML hodnotu:

  XMLSERIALIZE ({ DOCUMENT | CONTENT }
  <výraz dávající XML hodnotu> AS <typ dat>)

  kde <typ dat> je z {CHAR, VARCHAR, CLOB}.
- je-li je specifikováno DOCUMENT, <výraz dávající XML hodnotu> se musí vyhodnotit jako XML hodnota, která má přesně jeden vrcholový element; jinak nastává výjimka.
- použije-li se na výsledek XMLPARSE se specifikací DOCUMENT nebo CONTENT a PRESERVE WHITESPACE, měla by se získat stejná hodnota jaká je určená <výraz dávající XML hodnotu>.

XMLSERIALIZE

Příklad:

```
XMLSERIALIZE( DOCUMENT
  XMLPARSE (DOCUMENT '<zam>JosefNovák</zam>')
                    AS VARCHAR(100))
```

může vytvořit řetězec

```
<zam>Josef Novák</zam>
```

nebo

```
<?xml encoding="UTF-8" version="1.0"?>
<zam>Josef Novák</zam>
```

nebo

```
<?xml encoding="UTF-8" version="1.0"?>
<zam>Josef Novák</zam>
```

IS DOCUMENT

- predikát IS DOCUMENT kontroluje, zdali specifikovaná XML hodnota je XML dokument, tj. zdali má přesně jeden kořenový element

  <výraz dávající XML hodnotu> IS [NOT] DOCUMENT

Příklad:

```
SELECT XMLSERIALIZE (DOCUMENT xhodnota
   AS CLOB)
FROM Zaměstnanci
WHERE xhodnota IS DOCUMENT;
```

SQL/XML - vyhodnocení

- výhody
  - dědí výhody celé SQL infrastruktury (např. triggery, PL/SQL)
  - podpora transakcí
  - škálovatelnost, klastrování, spolehlivost
  - globální optimalizace (XML a relační)
- nevýhody
  - Požaduje, aby data byla v databázi
    - To není dobré pro temporární XML data
    - Ne příliš cenné pro malé objemy dat
    - složitá databázová komponenta, obtížné v univerzální db architektuře
  - Směs dvou jazyků (SQL, XQuery) není přirozená
  - XQuery není podporován databázovým strojem celý
    - ještě ne aktualizace XML dat