Bakalářská práce

Vladimír Mareš

Pedagogická fakulta Jihočeské univerzity
Katedra informatiky

Dotazovací jazyky pro XML a nativní XML databáze

bakalářská práce

Autor: Vladimír Mareš

Vedoucí bakalářské práce: PaedDr. Petr Pexa

České Budějovice 2005

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně s použitím literatury a zdrojů uvedených v části Použité zdroje.

……

Vladimír Mareš

Obsah:

1. Úvod

Tato bakalářská práce se zabývá dotazovacími jazyky pro XML a nativními XML databázemi. Budu se zde okrajově zabývat i možností uložení XML dokumentů do „klasických“ relačních databází. Předpokládám, že čtenář má alespoň obecné znalosti jazyka XML a databázových systémů, a proto se zde nebudu zabývat jejich detailním popisem. Na začátku ale přesto uvedu několik pojmů používaných v této bakalářské práci, aby byla ulehčena orientace v popisované problematice.

1.1. Používané zkratky

Zde jsou uvedeny některé zkratky používané v této bakalářské práci.

API – aplikační programové rozhraní

BLOB – Binary Large Object

CLOB – Character Large Object

DB – databáze

DOM – objektový model dokumentu

DTD – definice typu dokumentu

RDBS – relační databázový systém

SQL – Structured Query Language

SOAP – standard umožňující komunikaci mezi objekty běžícími v jiné aplikační doméně, tento standard je použit v pozadí webových služeb .NET platformy

SŘBD – systém řízení báze dat

WebDAV – standard umožňující číst a měnit datové elementy databáze podobně jako složky a soubory souborového systému

XML – Extensible markup language

XML RPC – standard umožňující komunikaci mezi objekty běžící v jiné aplikační doméně

XSL– eXtensible Stylesheet Language

XSLT – XSL Transformations

1.2. XML

Ve stále více globalizujícím se světě nabývají informace na důležitosti a vzrůstá potřeba jejich efektivního zpracovávání. Formátů pro výměnu dat existují stovky, ale většina z nich má jen úzké zaměření a mnohá omezení. XML je značkovací jazyk, který přináší lepší možnosti vyhledávání a zpracovávání informací. V dnešní době je většina dostupných informací uložena v nestrukturované podobě (např. textové dokumenty) a efektivní vyhledávání a zpracovávání dat je v tomto případě velmi náročné a mnohdy vyžaduje každý typ dokumentu individuální řešení. Naproti tomu se snadno vyhledávají informace v databázích, ve kterých jsou všechny údaje přehledně strukturovány. Problém je ale v tom, že v databázích je uloženo jen nepatrné procento informací, které jsou k dispozici. XML nabízí možnost vytváření strukturovaných dokumentů. XML je značkovací jazyk a jednotlivé části dokumentu označujeme značkami, které přesně specifikují jejich význam. Část dokumentu s informacemi například o CD, které máme doma by mohla vypadat jako v příkladu s číslem 1.

Příklad 1. – ukázka struktury XML dokumentu

```
<katalog>
   <cd>
      <umelec>Rolling stones</umelec>
      <titul>Bridges to Babylon</titul>
      <zanr>rock</zanr>
      <cena>500</cena>
      <stat>UK</stat>
   </cd>
     <cd>
        <umelec>Pink floyd</umelec>
        <titul>The Wall</titul>
        <zanr>rock</zanr>
        <cena>600</cena>
        <stat>UK</stat>
   </cd>
</katalog>
```

Takto strukturované informace lze snadno prohledávat a následně zpracovávát. Je samozřejmě zapotřebí, aby všichni autoři dokumentů označili informace odpovídajícími značkami, a aby pro stejný typ informace používali značky se stejnými názvy. Tento problém se snaží řešit například DocBook.

1.3. Typy XML dokumentů

XML dokumenty lze rozdělit na dvě skupiny a to buď na datově orientované dokumenty nebo na dokumentově orientované dokumenty.

1.3.1. Datově orientované XML dokumenty

Datově orientované XML dokumenty se nazývají XML dokumenty, které plní funkci samopopisné obálky pro přenos dat nebo manipulaci s nimi. Datově orientované XML dokumenty se vyznačují tím, že se nejmenší jednotky dat nacházejí na úrovni hodnot typu PCDATA a atributů. Obvykle nezáleží na pořadí sourozeneckých elementů. Datově orientované XML dokumenty jsou většinou ukládány do (objektově) relačních databází nebo jsou z těchto databází generovány. Jedním z typických použití je přenos tabulek z relačních databází bez ztráty informace o vzájemných relacích. Jako další příklady použití datově orientovaných XML dokumentů lze uvést např. objednávky, faktury, různé seznamy, vědecká a technická data atd.

Příklad 2. – ukázka struktury datově orientovaného XML dokumentu

```
<adresar>
   <kontakt>
      <id>4</id>
      <jmeno>Karel</jmeno>
      <prijmeni>Novák</prijmeni>
      <email>karel.novak@seznam.cz</email>
      <telefon>777478985</telefon>
   </kontakt>
```

```
<kontakt >
      <id>5</id>
      <jmeno>Martina</jmeno>
      <prijmeni>Zimová</prijmeni>
      <email>zima@centrum.cz</email>
      <telefon>608245623</telefon>
   </kontakt>
</adresar>
```

1.3.2. Dokumentově orientované XML dokumenty

Dokumentově orientované XML dokumenty se nazývají XML dokumenty, které mají méně pravidelnou nebo nepravidelnou strukturu a data v těchto dokumentech nejsou rozdělena na tak malé části jako v případě datově orientovaných XML dokumentů. Dokumentově orientované XML dokumenty mají velmi často smíšený obsah, ve kterém obvykle záleží na pořadí, v jakém jsou jednotlivé elementy uvedeny. Tyto dokumenty bývají často psány ručně přímo ve formátu XML nebo jsou do něj exportovány z různých textových formátů. Jako příklady dokumentově orientovaných XML dokumentů lze uvést např. knihy, WWW stránky ve formátu XHTML, emaily atd.

Příklad 3. – ukázka struktury dokumentově orientovaného XML dokumentu

```
<novinky>
   <clanek id = „2154“>
      <nadpis>Dokumentově orientované XML</nadpis>
      <odstavec> Dokumentově orientované XML dokumenty se
                 nazývají XML dokumenty, které mají méně
                 pravidelnou nebo nepravidelnou
                 strukturu.
      </odstavec>
      <odstavec> Tyto dokumenty bývají často psány ručně
           přímo ve formátu XML nebo jsou do něj
           exportovány z různých textových formátů.
      </odstavec>
   </clanek>
```

```
   <clanek id=„4788“>
     …
     …
   </clanek>
</novinky>
```

1.4. XML databázové produkty

V dnešní době již existuje celá řada produktů, které se dají rozdělit do několika skupin. Já zde využiji jejich rozdělení podle Ronalda Bourreta, který je v oblasti XML databází uznávaným odborníkem, jehož rozdělení produktů je velmi zdařilé a často používané i jinými experty v této oblasti. V praxi se ale můžeme setkat s produkty, které nejdou jednoduše zařadit do nějaké skupiny nebo je jeden produkt zařazen do více skupin (např. nativní databázový XML systém Tamino je označován i jako XML server).

1.4.1. Databázové systémy s podporou XML

Databázové systémy s podporou XML jsou databázové systémy, které jsou schopny pracovat s XML dokumenty. Tyto databáze jsou zatím asi nejlepší variantou pro práci s datově orientovanými XML dokumenty. Databáze s podporou XML formátu se označuje jako „XML enabled“ a v dnešní době je „XML enabled“ většina komerčních relačních databází. Tyto databáze nám mohou data poskytnout zpět v jinak strukturovaném XML dokumentu, než v námi ukládaném. Příčinou tohoto jevu jsou převodní algoritmy (table based, object based a atd.), které nejsou schopny předat informace o specifických, pro data nepodstatných elementů. Jako příklady systémů s podporou XML můžeme uvést např. Microsoft SQL server 2000, Oracle 9i, Oracle 10g, IBM DB2, Sysbase 12.5.1. V budoucnu lze v tomto oboru předpokládat stále narůstající podporu jazyka XML. Databázové systémy s podporou XML jsou zaměřeny především a datově orientované XML dokumenty.

1.4.2. Nativní XML databázové systémy

Nativní XML databázové systémy jsou databázové systémy, které ukládají XML dokument v jeho nativní (přirozené) podobě, tedy včetně jeho logické struktury, poznámek atd. Proto je vhodné používat nativní XML databázové systémy při práci s XML dokumenty, u kterých záleží na pořadí jednotlivých elementů. Do těchto databázových systémů se nejčastěji ukládají dokumentově orientované XML dokumenty a právě tyto dokumenty byly hlavní příčinou pro jejich vznik. Jako příklady nativních XML databázových systémů můžeme uvést např. Tamino (fa Software AG), exit (fa Wolfgang Meier) atd. Nativní databázové systémy jsou podrobněji rozebrány v kapitole 3. Nativní XML databázové systémy. Nativní databázové systémy jsou určeny pro datově i dokumentově orientované XML dokumenty.

1.4.3. Hybridní XML databázové systémy

Hybridní XML databázové systémy je možno používat buďto jako nativní XML databázové systémy nebo jako Databázové systémy s podporou XML v závislosti na požadavcích aplikace. Jako příklad můžeme uvést databázový systém Ozone.

1.4.4. Middleware

Pojmem middleware se označují programy, které slouží jako prostředníci mezi různými aplikacemi, které by nebyly schopné spolupracovat a právě middleware jim spolupráci umožňuje. V případě databázových systémů a formátu XML můžeme jako middleware označit nástroje, které slouží k transferu XML dat do databáze nebo dat z databáze do XML formátu. Jako příklad můžeme uvést Microsoft ADO. Middleware programy jsou určeny především pro datově orientované XML dokumenty.

1.4.5. XML servery

XML jsou aplikační servery, které poskytují data ve formátu XML. Jako příklad můžeme uvést Codlfusion (Macromedia) nebo Cocoon (Apache Software Fundation). XML servery jsou určeny pro práci s datově i dokumentově orientovanými XML dokumenty.

1.4.6. Wrappery

Wrapper je program, který „obalí“ (wrapper je angl. názvem pro obálku) a zpřístupní data uložená v relační formě a potom lze s dokumentem pracovat pomocí jazyka SQL. Wrappery jsou určeny především pro datově orientované XML dokumenty.

1.4.7. Systémy pro správu obsahu

Systémy pro správu obsahu jsou systémy vytvořené pro vytváření, správu, distribuci a zpřístupňování informací (např. firemních). Systém pro správu obsahu by v našem případě tvořil jakousi nadstavbu nad nativním XML databázovým systémem. Systémy pro správu obsahu jsou určeny především pro dokumentově orientované XML dokumenty.

1.4.8. Nástroje pro dotazování nad XML

Nástroje pro dotazování nad XML jsou programy, které umožňují provádět operace s daty uloženými ve formátu XML. V současné době se asi nejvíce používají pro dotazování dotazovací jazyky XPath a Xquery (oba od konsorcia W3C). Dotazovacím jazykům je věnována kapitola 2. Dotazovací jazyky pro XML. XML dotazovací jazyky jsou určeny pro práci s datově i dokumentově orientovanými XML dokumenty.

1.4.9. XML data binding nástroje

XML data binding nástroje jsou programy, které z XML dokumentu vytvářejí objekty zapouzdřující data v něm obsažená. Tyto programy vytvoří třídy podle XML schématu a tyto třídy jsou používány pro vytváření jim odpovídajícím objektům z dat z XML dokumentu. Jako příklad XML data binding nástrojů si můžeme uvést .Net Framework (Microsoft) nebo JAXP (Java architekture for XML binding od firmy Sun Microsystems). XML data binding nástroje jsou určeny především pro datově orientované XML dokumenty.

1.5. Datový model XML

Datový model XML označuje logickou a fyzickou strukturu obecného XML dokumentu.

1.5.1. XML strom

Strom vyjadřující XML dokument se skládá z jednotlivých uzlů.

1.5.2. Uzel

Uzly si můžeme rozdělit na několik typů zde uvedených:

- element
- dokument
- atribut
- text
- instrukce
- komentář
- deklarace jmenného prostoru (namespace)

Každý uzel má svou identitu a je tedy možné od sebe rozlišit dva uzly stejného jména a hodnoty. U každého uzlu známe pořadí v dokumentu v dané úrovni zanoření uzlu. Uzel také může mít jiné uzly jako potomky (rekurzivní definice stromu), jméno (název elementu, atributu atd.) a hodnotu nějakého typu definovaného v XSchema.

1.5.2.1. Uzel typu dokument

Uzel typu dokument je kořenovým uzlem stromu. V dokumentu se může teno uzel vyskytovat pouze jednou a všechny následující uzly jsou jeho potomky a tento uzel nemůže mít žádné atributy, ale může obsahovat URL, které odkazuje na specializovaný datový model určený pro tento uzel a jeho potomky. Zápis uzlu typu dokument: `<!doctype „název uzlu“ …`
Pokud není uvedeno `<!doctype…>`, má dokument tzv. anonymní kořen.

1.5.2.2. Uzel typu element

Elementy se v XML dokumentu vyznačují pomocí tzv. tagů. Ve většině případů je element vyznačen počátečním a ukončovacím tagem.
Např. `<para>Obsah elementu</para>`

Název elementu je uveden mezi znaky „<” a „>”. V ukončovacím tagu je před názvem ještě znak „/”. Některé elementy nemusí mít žádný obsah. V tomto případě máme dvě varianty, jak element zapsat.

1. `<para></para>`
2. `<para/>`

Ve druhém případě je „ukončovací“ znak „/“ uveden za názvem elementu v počátečním tagu.

Každý element tvoří samostatný uzel stromu a hierarchie elementů v XML dokumentu odpovídá hierarchii uzlů ve stromové reprezentaci.

1.5.2.3. Uzel typu atribut

Uzel typu atribut je ve stromu vždy připojen k odpovídajícímu elementovému uzlu. Atribut není chápán jako dítě tohoto uzlu, ale tento uzel je chápán jako rodič atributu. Jedná se vždy o listový uzel. Název uzlu je názvem atributu a hodnota uzlu je hodnota atributu.

1.5.2.4. Uzel typu text

Textové uzly odpovídají textovému obsahu elementů. Uzel je vždy listovým uzlem (nemá žádné děti) a nemá název.

1.5.2.5. Uzel typu instrukce pro zpracování

Uzel typu instrukce pro zpracování je vždy listovým uzlem. Obsah tohoto uzlu je tvořen cílem instrukce, obsahem instrukce a bílými znaky. Uzel typu instrukce je vyznačen znaky „<?“a „?>”.

1.5.2.6. Uzel typu komentář

Jedná se vždy o listový uzel. Má podobný význam jako instrukce pro zpracování, s tím rozdílem, že komentáře jsou určeny spíše pro potřeby lidí (programátorů, uživatelů apod.) a jsou v nich uváděna např. vysvětlení konkrétní části dokumentu nebo různé poznámky.

1.5.2.7. Uzel typu deklarace jmenného prostoru

Vždy se jedná o listový uzel. Uzel není chápán jako dítě elementu ke kterému je připojený, ale tento element je chápán jako jeho rodič. Název uzlu odpovídá prefixu jmenného prostoru a hodnotou uzlu je URI adresa jmenného prostoru.

2. Dotazovací jazyky nad XML

2.1. Jazyk XPath

XPath je jazyk od konsorcia W3C. Jazyk XPath pracuje s abstraktním datovým modelem XML dokumentu. V paměti počítače je pro XML dokument vytvořen strom, nad kterým může XPath vyhodnocovat své výrazy. Pro procházení stromu jsou využívány tzv. osy, pomocí kterých lze definovat cestu v XML dokumentu. V současné době je jazyk XPath ve verzi 2.0 a oproti verzi 1.0 se liší hlavně přidáním podpory více datových typů. Může využít i informace o daném typu s příslušného XML schématu, patřícímu k XML dokumentu. Jazyk XPath je velice často využíván v ostatních dotazovacích jazycích pro XML, které jsou mnohdy jen jakousi nadstavbou jazyka XPath.

2.1.1. Jazyk XPath 1.0 a Jazyk XPath 2.0

V současné době se používají standarty XPath 1.0 i XPath 2.0, které jsou si velmi podobné. XPath 2.0 navazuje na silné stránky XPath 1.0, ke kterým je navíc přidáno několik nových možností.

2.1.1.1. Omezení Jazyka XPath 1.0

Ačkoli specifikace XPath 1.0 přinesla řadu zjednodušení při práci s XML daty, existuje v ní několik nejasností, které bylo třeba vylepšit. Konsorcium W3C se snaží XPath zdokonalit tak, aby lépe podporoval ostatní standarty jako XQuery, XML Schema, XSLT 2.0. Ve specifikaci XPath 1.0 chybí hlavně podpora různých typů, a proto vznikla specifikace XPath 2.0, která lépe vyhoví požadavkům ostatních standardů.

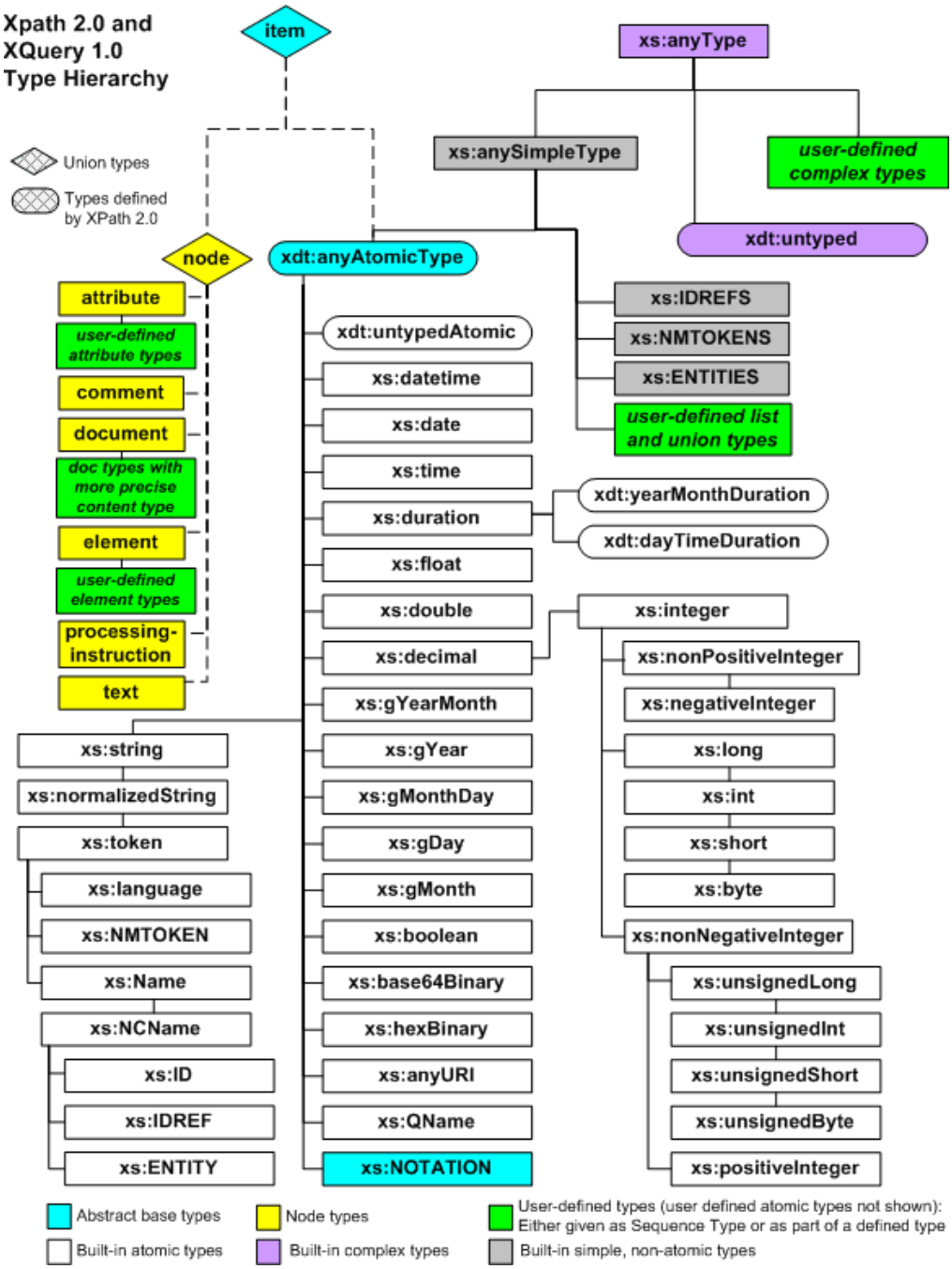
2.1.1.2. Jazyk XPath 2.0

Základní seznam požadavků na specifikaci XPath 2.0 je následující:

- měla by udržovat zpětnou kompatibilitu
- měla by zjednodušit svoje používání
- měla by lépe podporovat manipulaci s řetězci
- podpora ostatních standardů (XSLT 2.0, XQuery 1.0)
- lepší podpora XML schéma (jednoduché a komplexní typy)

Tyto požadavky specifikace XPath 2.0 celkem dobře splňuje. Zpětná kompatibilita není sice stoprocentní, protože se při vývoji muselo vyhovět ostatním požadavkům, které jsou zřejmě důležitější pro současné použití a budoucí vývoj XPath a ostatních standardů, které XPath využívají. Hlavní změna v XPath 2.0 je ve specifikaci datového modelu, který je založen na infosetu (stromová reprezentace XML dokumentu) s nezbytnou podporou XML schéma. Datový model definuje 7 typů uzlů v XML dokumentu (dokument, element, text, atribut, jmenný prostor, instrukce pro zpracování a komentáře). Uzly v datovém modelu XPath 2.0 jsou velmi podobné jako XPath 1.0, ale v případě elementů a atributů je rozšířena možnost typové informace v XML schématu pro správnou validaci XML dokumentu. Výsledný „typový“ datový model je označován jako „Post schema validation infoset“ (PSVI).

Obrázek 1. – typy v jazyce Xpath 2.0 a XQuery



2.1.2. Syntaxe XPath

XML dokument může být reprezentován jako strom uzlů (je to podobné adresářové struktuře na vašem PC).

XPath využívá cesty vyjádření k identifikace uzlů v XML dokumentu. Cesta se skládá z lomítka oddělujícího seznam *child* elemntů vyjadřujících cestu XML dokumentem. Příkaz vybere elementy shodné se zadanou cestou.

2.1.2.1. Výběr elementů

Uvedené XPath dotazy se budou provádět nad XML dokumentem z příkladu č. 1.

Následující XPath vyjádření vybere všechny elementy *cena* ze všech elementů *CD* z elementu *katalog*: „`/katalog/cd/cena`“.

Jestliže cesta začíná lomítkem „/“, reprezentuje absolutní cestu k elementu a jestliže cesta začíná dvěma lomítky „//“, tak platí pro celý dokument.

Následující XPath výraz vybere všechny elementy *cd* v dokumentu: „`//cd`“.

K vybrání neznámých elementů se používá znaménko „*“.
Následující XPath výraz vybere všechny *child* (dětské) elementy z elementu *cd* z elementu *katalog*: „`/katalog/cd/*`“.

Následující XPath výraz vybere všechny elementy *cena*, které jsou „grandchild“ elementu *katalog*: „`/katalog/*/cena`“.

Následující XPaht výraz vybere všechny elementy *cena*, které mají 2 předky:
„*`/*/*/cena`*“.

Následující XPaht výraz vybere všechny elemnty z dokumentu: „/*“.

Použitím hranatých závorek můžeme specifikovat další element.
Následující XPath výraz vybere první „child“ element *CD* z elementu *katalog*: „`/katalog/cd[1]`“.

Následující XPath výraz vybere poslední „child“ element *cd* z elementu *katalog*: „`/katalog/cd[last()]`“, (funkce first() neexistuje).
Následující XPaht výraz vybere všechny elementy *cd*, které obsahují element cena: „`/katlog/cd[cena]`“.

Následující XPath výraz vybere všechny elementy *cd* z elementu *katalog*, kde má element *cena* hodnotu 500: „`/katalog/cd[cena=500]`“ .

Následující XPath výraz vybere všechny elementy *cd* z elementu *katalog*, kde má element *cena* hodnotu 600: „`/katalog/cd[cena=600]/cena`“.

2.1.2.2. Výběr atributů

V XPath jsou všechny atributy specifikovány prefixem @.
„`//@stat`“ – vybere všechny atributy s názvem *stat.*
„`//cd[@stat]`“ – všechny elementy *cd*, které mají atribut s názvem *stat.*
„`//titul | //umelec`“ – vybere všechny elementy *titul* a *umelec* z dokumentu.
„`//cd[@*]`“ – vybere všechny elementy *cd* s nějakým atributem.
„`//cd[@stat='UK']`“ – vybere všechny elementy *cd* s atributem *stat* s hodnotou *UK*.

2.1.2.3. Cesta k umístění

Cesta k umístění může být absolutní nebo relativní. Absolutní cesta k umístění začíná lomítekem (/) a relativní cesta lomítkem nezačíná. Oba případy obsahují jeden nebo více částí umístění, všechny odděleny lomítkem.
Absolutní cesta: `/step/step/...`
Relativní cesta: `step/step/...`

Cesta k umístění je ohodnocena pouze jednou, z leva do prava. Každý krok vyjadřuje jeden uzel z aktuální množiny uzlů. Jestliže je cesta absolutní, tak aktuální množina uzlů obsahuje hlavní uzel root. Jestliže je cesta relativní, obsahuje aktuální množina uzlů uzly které výraz obsahuje. Části cesty jsou složeny z osy, která specifikuje strom vztahů mezi uzly vybranými aktuálním krokem cesty a aktuálním uzlem.

Výběr několika cest

S použitím operátoru | v XPath výrazu můžeme vybrat několik cest.

Výběr elementů *titul* a *umelec* z elementu *cd* z elementu *katalog:*

„`/katalog/cd/titul | /katalog/cd/umelec`“.

Výběr všech elementů *titul* z elementu *cd* z elementu *katalog* a všechny elementy *umelec* z dokumentu: *„`/katalog/cd/titul | //umelec`“*.

2.1.2.4. XPath osy

Osa definuje množinu uzlů relativně k aktuálnímu uzlu. Test uzlu se používá k identifikaci uzlu uvnitř osy. Můžeme vykonávat test uzlu podle jména nebo typu.

Jména os

„`ancestor`“(předek) – obsahuje všechny předky (*parent, grandparent* atd.) aktuálního uzlu a vždy zahrne *root* uzel, jestliže aktuální uzel není *root.*

„`ancestor-or-self` “ – obsahuje aktuální uzel + všechny předky.

„`attribute`“ – obsahuje všechny atributy aktuálního uzlu.

„`child`“ – obsahuje všechny děti aktuálního uzlu.

„`descendant`“ – obsahuje všechny potomky aktuálního uzlu neobsahuje atributy a jmenné prostory uzlů.

„`descendant-or-self` “– aktuální uzel + všichni potomci.

„`following`“ – obsahuje všechno v dokumentu po uzavíracím tagu aktuálního dokumentu.

„`following-sibling`“ – obsahuje všechny sourozence po aktuálním uzlu jestliže je aktuální uzel atribut uzel nebo *namespace* uzel, bude tato osa prázdná.

„`namespace`“ – obsahuje všechny *namespace* uzly aktuálního uzlu.

„`parent`“ – obsahuje rodiče aktuálního uzlu.

„`preceding`“ – obsahuje vše v dokumentu, co senachází před otevíracím tagem aktuálního uzlu.

„`preceding-sibling`“ – všechny sourozence před aktuálním uzlem. Jestliže je aktuální uzel atribut nebo jmenný prostor, tak je výsledná množina prázdná.

„`self` “ – obsahuje aktuální uzel.

„`child::cd`“ – vybere všechny elementy *cd*, které jsou dětmi aktuálního uzlu. Jestliže nemá žádné děti je výsledná množina prázdná.

„`attribute::src`“ – vybere *src* atribut aktuálního uzlu. Pokud uzel nemá tento atribut, vrací prázdnou množinu.

„`child::*`“ – vybere všechny „děti“ elementy aktuálního uzlu.

„`attribute::*`“ – vybere všechny atributy aktuálního uzlu.

„`child::text()`“ – vybere text dětského uzlu aktuálního uzlu.

„`child::node()`“ – vybere všechny „děti“ aktuálního uzlu.

„`descendant::cd`“ – vybere všechny elementy *cd*, které jsou potomky aktuálního uzlu.

„`ancestor-or-self::cd`“ – vybere všechny elementy *cd*, které jsou potomky aktuálního uzlu a jestliže je aktuální uzel element *cd*, tak také ten.

„`child::*/child::cena`“ – všechny ceny „vnoučat“ aktuálního uzlu.

„/“ – vybere kořen dokumentu.

2.1.2.5. Predikáty

Predikát filtruje množinu uzlů na novou množinu uzlů. Predikát je uvnitř hranatých závorek.

„`child::cena[cena=400]`“ – vybere všechny price elementy, které jsou dětmi aktuálního uzlu kde se cenový prvek rovná 400.

„`child::cd[position()=1]`“ – vybere první „child“ element *cd* aktuálního uzlu.

„`child::cd[position()=last()]`“ – vybere poslední „child“ element *cd* aktuálního uzlu.

„/descendant::cd[position()=7]“ –vybere sedmý element *cd,* který je potomkem aktuálního uzlu.
„child::cd[attribute::type="rock"]“ – vybere všechny „child“ elementy cd s atributem classic aktuálního uzlu.
„child::cd[position()=last()-1]“ – vybere předposlední „child“ element cd aktuálního uzlu.
„child::cd[position()<6]“ – vybere prvních 5 „child“ elementů *cd* aktuálního uzlu.

2.1.2.6. Zkrácený zápis

Zkratka „“ (nic neuvedeme) znamená „child::“.
Např.: „cd“ je zkratka pro „child::cd“.

Zkratka „@“ znamená „attribute::“.
Např. „cd[@type="rock"]“ je zkratka pro
„child::cd[attribute::type="rock"]“

Zkratka „.“ znamená „self::node()“.
Např.: „.//cd“ je zkratka pro
„self::node()/descendant-or self::node()/child::cd“.

Zkratka „..“ znamená „parent::node()“.
Např.: „../cd“ je zkratka pro „parent::node()/child::cd“.

Zkratka „//“ znamená „/descendant-or-self::node()/“ .
Např.: „//cd“ je zkratka pro
„/descendant-or-self::node()/child::cd“.

Pokud uvedeme název uzlu, získáme všechny elementy s tímto názvem, které jsou dětmi aktuálního uzlu.
Např.: „kniha“ – vybere všechny dětské uzly *kniha* aktuálního uzlu

„ `*` “ – vybere všechny dětské uzly aktuálního uzlu.

„`text()`“ – vybere všechny textové dětské uzly aktuálního uzlu.

„`@src`“ – vybere všechny atributy *src* aktuálního uzlu.

„`@*`“ – vybere všechny atributy aktuálního uzlu.

„`kniha[1]`“ – vybere první dětský uzel *kniha* aktuálního uzlu.

„`kniha[last()]`“ – vybere poslední dětský uzel *kniha* aktuálního uzlu.

„`*/kniha`“ – vybere všechny „vnukovské“ (*grandchildren*) uzly *kniha* aktuálního uzlu.

„`/kniha/kapitola[3]/odstavec[1]`“ – vybere první odstavec ze třetí kapitoly z uzlu kniha.

„`//kniha`“ – vybere všechny uzly *kniha,* které jsou potomky kořenového uzlu a potom všechny uzly *kniha,* které jsou potomky aktuálního uzlu.

„ `.` “ – vybere aktuální uzel.

„`.//kniha`“ – vybere všechny uzly *kniha*, které jsou potomky aktuálního uzlu.

„`..`“ – vybere rodičovský element aktuálního uzlu.

„`../@src`“ – vybere atributy *src* rodičovského uzlu.

„`kniha[@type='klasika']`“ – vybere všechny dětské uzly *kniha* aktuálního uzlu, které mají atribut s hodnotu klasika.

„`kniha[@type='klasika'][5]`“ – vybere pátý dětský uzel *kniha* aktuálního uzlu, který má atribut s hodnotou klasika.

„`kniha[5][@type='klasika']`“ vybere pátý dětský uzel *kniha* aktuálního uzlu, který má atribut s hodnotou klasika.

„`kniha[@type and @isbn]`“ – vybere všechny dětské uzly aktuálního uzlu, které mají oba uvedené parametry.

2.1.3. Výrazy

Výrazy v jazyce XPath 2.0 jsou stejné jako v jazyce XQuery, a proto už v části o jazyku XQuery nebudou příliš rozebírány.

Matematické výrazy

Matematické výrazy s příkladem použití jsou uvedeny v tabulce č. 1.

Tabulka 1. – matematické výrazy

Operátor	Význam	Příklad	Výsledek
+	sčítání	6 + 4	10
–	odečítání	6 – 4	2
*	násobení	6 * 4	24
div	dělení	8 div 4	2
mod	zbytek po dělení	5 mod 2	1

Porovnávací výrazy

Porovnávací výrazy s příkladem použití jsou uvedeny v tabulce č. 2.

Tabulka 2. – porovnovácí výrazy

Operátor	Význam	Příklad	Výsledek
=	rovno	cena = 10	true (jestliže je cena rovna 10)
!=	nerovno	cena != 10	true (jestliže je cena není rovna 10)
>	větší	cena > 10	true (jestliže je cena větší než 10)
>=	větší nebo rovno	cena >= 10	true (jestliže je cena větší nebo rovna 10)
<	menší	cena < 10	true (jestliže je cena menší než 10)
<=	menší nebo rovno	cena <=10	true (jestliže je cena menší nebo rovna 10)

2.1.4. Funkce

Funkce v jazyce XPath 2.0 jsou stejné jako v jazyce XQuery.

Číselné funkce

Číselné funkce s příkladem použití jsou uvedeny v tabulce č. 3.

Tabulka 3. – číselné funkce

Funkce	Příklad	Výsledek
abs()	abs(55.5)	55.5
	abs(–55.5)	55.5
ceiling()	ceiling (55.5)	56
	ceiling (–55.5)	–55
floor()	floor (55.5)	55
	floor (–55.5)	–56
round()	round(55.5)	56
	round(–55.5)	55

Funkce na řetězích

Funkce na řetězích příkladem použití jsou uvedeny v tabulce č. 4.

***Tabulka 4.* – funkce na řetězích**

Funkce	Příklad	Výsledek
concat()	concat(‘ne’,’bezbečný’)	nebezpečný
string–join()	string–join(‘Ahoj,’, ‘jak’, ‘to’, ‘…’)	Ahoj, jak to …
substring()	substring(‘123456’,2,3)	234
string–length()	string–length(‘12345‘)	5
normalize–space()	normalize–space(‘Ahoj, jak to jde?)	Ahoj, jak to jde?
upper–case()	upper–case(‘aBcdE’)	ABCDE
lower–case()	lower–case(‘aBcdE’)	abcde
translate()	translate(‘Ahoj‘, ‘A’,’a’)	ahoj

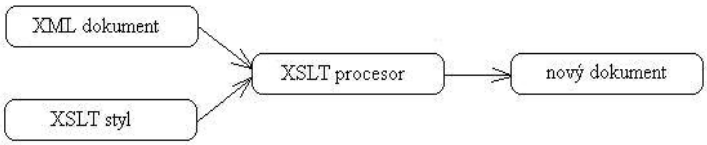
Pro jazyk XPath 2.0 a jazyk XQuery existuje ještě celá řada funkcí, které zde nebudu všechny popisovat. Funkce, které jsem uvedl, slouží pro lepší představu o tom, co vše jazyky XPath 2.0 a XQuery umožňují. Nezmínil jsem zde ani funkce pro práci s datumem a časem, kterých je také celá řada a pokud se s nimi chcete seznámit podívejte se na http://www.w3.org/TR/xpath–functions/.

2.2. XSLT

Jednou ze základních myšlenek značkovacích jazyků je oddělení obsahu dokumentu od jeho vzhledu. Značky, které používá jazyk XML označují význam jednotlivých částí (např. id zákazníka, adresa atd.), ale neříkají nic o tom, jak se konkrétní údaj zobrazí na obrazovce nebo vytiskne na tiskárně. Pro vyjádření vzhledu jednotlivých částí dokumentů se používají tzv. stylové jazyky (např. XSL, CSS). Jazyk XSL se používá ke transformaci dokumentů a definici vzhledu jejich formátování. Během příprav standardu XSL od něj byla oddělena část určená pro transformaci dokumentů, která nese název XSLT (XSL Transformations). Druhá část XSL, která je určena k přesnému popisu vzhledu dokumentu (řádkování, okraje atd.) se nazývá XSL FO (XSL Formatting Objects).

2.2.1. Princip XSLT transofrmace

Obrázek 2. – princip XSLT transformace



XSLT procesor je program, který provádí XSLT transformaci. Asi nejznámějšími zástupci XSLT procesorů jsou programy SAXON a XALAN. Pomocí XSLT transformace také můžeme do výsledného dokumentu přidat nové elementy nebo naopak některé elementy z původním dokumentu netransformovat do nového dokumentu. K přístupu k jednotlivým částem dokumentu využívá XSLT jazyka XPath.

2.2.2. Jmenné prostory

Dokument XSLT je ve formátu XML. Aby šlo v jednom dokumentu používat více sad značek, používají se jmenné prostory. Například před jména všech elementů, které má XSLT procesor zpracovat se píše prefix *xsl:*. Celý styl musí být uvnitř elementu *stylesheet* nebo *transform.*

Př.:
```
<xsl:stylesheet version="1.0"
   xmlns:xsl=http://www.w3.org/1999/XSL/Transform>
   …
   definice stylu
</xsl:stylesheet>
```

nebo
```
<xsl:transform version="1.0"
   xmlns:xsl=http://www.w3.org/1999/XSL/Transform>
   …
   definice stylu
</xsl:transform>
```

2.2.3. Funkce v XSLT

XSLT obsahuje řadu funkcí pro práci XML daty, a pro Vaši představu jich zde několik uvedu.

„`current()`" – vrací aktuální uzel.

„`document()`" – zpřístupňuje uzly z extérního XML dokumentu.

„`element-available()`" – testuje, zda je element podporován XSLT procesorem.

„`format-number()`" – konvertuje číslo na řetězec.

„`function-available()`" – testuje, zda je funkce podporována XSLT procesorem.

„`generate-id()`“ – vrací řetězcovou hodnotu, která jednoznačně identifikuje uzel.

„`key()`“ – vrací uzly určené klíčem.

„`system-property()`“ – vrací hodnoty systémových proměnných.

„`unparsed-entity-uri()`“ – vrací URI zadané entity.

Uvedé funkce nejsou zdaleka všechny a pokud Vás zajímá více, můžete se podívat na obsáhlý manuál na http://www.w3.org/TR/2005/WD–xslt20–20050404/.

2.2.4. XSLT elementy

V XSLT se používají různé elementy ke konstrukci požadovaného výstupu.

Příklad 4. – ukázka XML souboru, který budeme transformovat

```
<?xml version="1.0" encoding="ISO-8859-2"?>
<?xml-stylesheet type="text/xsl" href="knihovna.xsl"?>
<knihovna>
   <kniha>
      <autor>Karel Čapek</autor>
      <nazev>R.U.R.</nazev>
      <rok_vydani>1935</rok_vydani>
      <isbn>7895548</isbn>
   </kniha>
.
.
.
</knihovna>
```

Element *<xsl:template>*

Element `<xsl:template>` se používá pro určení uzlu, na který se má aplikovat šablona. Pokud chceme šablonu aplikovat na celý dokument používá se lomítko „/“ (`<xsl:template match=„/“>`).

Element *<xsl:value–of>*

Element <xsl:value-of> se používá pro získání hodnoty elementu z XML dokumentu. Následující příkladu získáme hodnotu elementu *autor,* která bude vložena do řádku tabulky v HTML dokumentu.

Příklad 5. – vložení hodnoty elementu do html tabulky

```
<tr>
   <td>
      <xsl:value-of select=„knihovna/kniha/autor"/>
   </td>
</tr>
```

Element *<xsl:for–each>*

Element <xsl:for-each> se používá, pokud chceme vybrat každý XML element z definované uzlu. Pokud bychom například chtěli vybrat všechny knihy z knihovny a předat jména autorů do tabulky v HTML dokumentu, můžeme použít následující příklad.

Příklad 6. – do řádek tabulky jsou vloženi všichni autoři z knihovny

```
<xsl:for-each select="knihovna/kniha">
   <tr>
      <td><xsl:value-of select="autor"></td>
   </tr>
<xsl:for-each>
```

Element *<xsl:sort>*

Element <xsl:sort> se používá ke třídění výsledků v elementu <xsl:for-each>. Atribut elementu <xsl:sort> určuje, podle čeho se bude třídit. Pokud bychom například chtěli vybrat všechny knihy z knihovny a předat jména autorů do tabulky v HTML dokumentu jako v předchozím případě a navíc je setřídit podle jména autora, můžeme použít následující příklad.

Příklad 7. – do řádek tabulky jsou vloženi podle abecedy všichni autoři z knihovny

```
<xsl:for-each select="knihovna/kniha">
   <xsl:sort select="autor"/>
   <tr>
      <td><xsl:value-of select="autor"></td>
   </tr>
<xsl:for-each>
```

Element *<xsl:if>*

Element `<xsl:if>` obsahuje šablonu, která se provede pouze v případě, že daná podmínka vrátí *true*. V následujícím příkladě si ukážeme, jak vypsat knihy z knihovny, které mají rok vydaní větší než 1981.

Příklad 8. – výběr knih podle roku vydání

```
<xsl:for-each select="knihovna/kniha">
   <xsl:if test="rok_vydani &gt; 1981">
      <tr>
         <td><xsl:value-of select="autor"></td>
      </tr>
   </xsl:if>
<xsl:for-each>
```

Element *<xsl:choose>*

Element `<xsl:choose>` se spolu s elementy `<xsl:when>` a `<xsl:otherwise>` používá pro vícenásobné podmíněné testy. V následujícím příkladě bude ukázáno, jak knihy vydané po roce 1981 vypsat tučným písmem a ostatní kurzivou.

Příklad 9. – různé formátování písma podle roku vydání knihy

```
<xsl:for-each select="knihovna/kniha">
    <tr>
      <xsl:choose>
         <xsl:when test ="rok_vydani &gt; 1981">
            <td><b><xsl:value-of select="autor"></b></td>
         </xsl:when>
        <xsl:otherwise>
           <td><i><xsl:value-of select="autor"></i></td>
        </xsl:otherwise>
   </tr>
<xsl:for-each>
```

Element *<xsl:apply–templates>*

Element `<xsl:apply-templates>` se používá k přiřazení pravidla ze šablony pro aktuální uzel nebo „dětské“ elementy aktuálního uzlu. Tento element se může použít s atributem, který určí, na jaké dětské elementy se bude šablona aplikovat. V následujícím příkladě si ukážeme, jak aplikovat šablonu na určité elementy, konkrétně elementy *autor* a *nazev.*

Příklad 10. – použití šablon

```
<xsl:template match=„kniha“>
   <p>
      <xsl:apply-templates select=„autor“/>
      <xsl:apply-templates select=„nazev“/>
   </p>
</xsl:template>

<xsl:template match=„autor“>
    Title: <span style=„color:#FF0000“>
   <xsl:value-of select=„.“/></span>
   <br />
</xsl:template>
```

```
<xsl:template match=„nazev“>
    Artist: <span style=„color:#008000“>
   <xsl:value-of select=„.“/></span>
   <br />
</xsl:template>
```

Element *<xsl:apply–imports>*

Element `<xsl:apply-imports>` slouží k použití importované šablony.

Element *<xsl:attribute>*

Element `<xsl:attribute>` se používá k přidání atributu.

Element *<xsl:attribute–set>*

Element `<xsl:attribute-set>` slouží k pojmenování množiny atributů.

Element *<xsl:call–template>*

Element `<xsl:call-template>` slouží k zavolání pojmenované šablony.

Element *<xsl:comment>*

Element `<xsl:comment>` slouží k vytvoření komentářového uzlu ve výsledném stromu.

Element *<xsl:copy>*

Element `<xsl:copy>` slouží k vytvoření kopie aktuálního uzlu, bez dětských uzlů a atributů.

Element *<xsl:copy–of>*

Element `<xsl:copy-of>` slouží k vytvoření kopie aktuálního uzlu s dětskými uzly a atributy.

Element <*xsl:decimal–format*>

Element `<xsl:decimal-format>` slouží k určení znaků, které mají být konvertovány z čísel na řetězce pomocí funcke `format-number()`.

Element <*xsl:element*>

Element `<xsl:element>` slouží k vytvoření elementu ve výsledném dokumentu.

Element <*xsl:fallback*>

Element `<xsl:fallback>` specifikuje náhradní kód, který se provede pokud XSLT processor nepodporuje určitý element.

Element <*xsl:import*>

Element `<xsl:import>` slouží k inportování obsahu jedné formátovací šablony do jiné.

Element <*xsl:include*>

Element `<xsl:include>` slouží ke vkládání obsahu jedné formátovací šablony do jiné.

Element <*xsl:key*>

Element `<xsl:key>` slouží k definici jména klíče používaného v šabloně funkcí `key()`.

Element <*xsl:message*>

Element `<xsl:message>` slouží k výpisu zprávy na výstup (výpis errorů).

Element <*xsl:namespace–alias*>

Element `<xsl:namespace-alias>` slouží k přemístění jmenného prostoru ze šablony do jiného jmenného prostoru na výstupu.

Element <*xsl:number*>

Element <xsl:number> určí číslo pozice aktuálního uzlu.

Element <*xsl:output*>

Element <xsl:output> určuje formát výstupního dokumentu.

Element <*xsl:param*>

Element <xsl:param> slouží k deklaraci lokálních nebo globálních parametrů.

Element <*xsl:preserve–space*>

Element <xsl:preserve-space> slouží k určení elementů, ve kterých mají být zachovány „bílé“ znaky.

Element <*xsl:processing–instruction*>

Element <xsl:processing-instruction> slouží k vypsání instrukce ke zpracování do výstupu.

Element <*xsl:strip–space*>

Element <xsl:strip-space> slouží k určení elementů, ze kterých mají být odstraněny „bílé“ znaky.

Element <*xsl:text*>

Element <xsl:text> slouží k vypsání textu do výstupu.

Element <*xsl:variable*>

Element <xsl:variable> slouží k deklaraci lokálních nebo globálních proměnných.

Element <*xsl:with–param*>

Element `<xsl:with-param>` definuje hodnotu parametru k procházení šablony.

2.2.5. XSLT 2.0

Standart XSLT 1.0 se stal velmi používaným pro svou kvalitu a možnosti, ale nebyly v něm vyřešeny problémy s konstrukcí složitějších struktur, které jsou ve standartu 2.0 velmi dobře řešeny. Pokud chceme pracovat se standartem XSTL 2.0 musíme zároveň používat i XPath verze 2.0.

2.2.6. Přínos XSLT 2.0

V této části jsou uvedeny některé výhody standartu XSLT 2.0 oproti XSLT 1.0.

2.2.6.1. Vytváření skupin

XSLT 2.0 umožňuje sdružit vybrané elementy do skupiny pomocí instrukce `<xsl:for-each-group>`.

2.2.6.2. Vícenásobný výstup

XSLT 2.0 umožňuje generovat z jedné transformace více výstupních dokumentů pomocí instrukce `<xsl:result-document>`. Díky tomu můžeme generovat do různých formátů (html, xhtml, doc, pdf atd.) nebo různé jazykové mutace.

2.2.6.3. Datové typy

XSLT 2.0 podporuje vestavěné XML Schéma, které definuje různé datové typy. XSLT 2.0 umožňuje definovat datový typ i pro proměnné *xsl:variable* a parametry *xsl:param,* se kterými můžeme pracovat v typově závislých operacích.

2.2.6.4. Uživatelské funkce

XSLT 2.0 umožňuje snadno deklarovat uživatelské funkce.

Příklad 11. – ukázka funkce v XSLT 2.0

```
<xsl>function name="obsah_obdelniku">
   <xsl:param name="x />
   <xsl:param name="y"/>
   <xsl:result select="x*y"/>
</xsl:function>
```

2.2.6.5. Dočasné stromy

V XSLT 1.0 jsou proměnné a parametry reprezentovány jako řetězcové hodnoty. V XSLT 2.0 lze vytvářet tzv. dočasné stromy, které lze procházet a vyhodnocovat. S dočasným stromem lze pracovat lépe než s řetězcem a můžeme ho dále zpracovávat pomocí XPath.

2.3. XML–QL

XML–QL je jeden z prvních jazyků pro dotazování nad dokumenty ve formátu XML. Z počátku to vypadalo, že je to celkem podařený jazyk a bude se hojně využívat. Z jeho nastupující pozice ho ale vytlačil standart XPath 2.0 od konsorcia W3C, na jehož základě vznikl zatím nejkomplexnější dotazovací jazyk pro XML s názvem XQuery.

Silnou stránkou jazyka XML–QL je jednoduchá tvorba dotazů, která je trochu podobná SQL. XML–QL dotaz se skládá ze dvou základních částí. První část můžeme označit jako *dotazovací*, protože specifikuje vlastní dotaz. Druhou část můžeme označit jako *konstrukční,* protože definuje strukturu vrácených dat, která splnila podmínky dotazu.

Příklad 12. – ukázka xml dokumentu nad kterým budeme vykonávat dotazy

```
<lednicka>
   <polozka id=”1”>
      <nazev>Párky</nazev>
      <mnozstvi jednotka=”kg”>0,5</mnozstvi>
      <typ>maso</typ>
</polozka>
.
.
.
</lednicka>
```

Příklad 13. – XML–QL dotaz pro získání názvů položek

```
where <lednicka>
              <polozka id=”1”>
                 <nazev>$nazev</nazev>
              </polozka>
        </lednicka> in „lednicka.xml“
construct $nazev
```

Výsledek tohoto dotazu:

```
<XML>párky</XML>
```

Do dotazovací části začínající klíčovým slovem `where` se zapíší XML značky tak, aby odpovídaly XML značkám v XML dokumentu nad kterým provádíme dotazy. Pokud známe nějaké hodnoty z dokumentu, tak je můžeme uvést. Tímto způsobem se provádí selekce, protože pokud uvedeme `<typ>maso</typ>`, vyberou se pouze položky, které mají jako hodnotu elementu typ maso. Pokud hodnotu elementu neznáme nebo ji v dotazu nepotřebujeme uvést, zapíšeme pouze tagy elementu a mezi ně napíšeme tzv. vazební proměnou, která musí začínat znakem „$”. Můžeme tedy napsat např. `<typ>$maso</typ>` a na vazební proměnnou se budeme odkazovat v části dotazu `construct`. Za posledním tagem v části `where` následuje klíčové slovo `in`, za nímž je uveden název souboru na

jehož data se dotazujeme. Potom následuje část `construct`, ve které určíme, jak bude vypadat výsledek našeho dotazu.

Příklad 14. – ukázka dokumentu zakaznici.xml

```
where <lednicka>
              <polozka id=$id>
                 <nazev>$nazev</nazev>
                 <typ>maso</nazev>
              </polozka>
      </lednicka> in „lednicka.xml“
construct <mastne_vyrobky>
                    <polozka id=„$id“>
                       <nazev>$nazev</nazev>
                    </polozka>
          </mastne_vyrobky>
```

Výsledek tohoto dotazu:

```
<mastne_vyrobky>
    <polozka id=„1“>
       <nazev>parky</nazev>
    </polozka>
</mastne_vyrobky>
```

2.4. XQuery

XQuery je dotazovací jazyk od konsorcia W3C, který vychází z dotazovacího jazyka Quilt a XML–QL. XQuery je pro dotazovaní nad XML dokumenty stejně dobrý jako je SQL dobré pro dotazování v relačních databázích. XQuery je momentálně ve verzi 1.0 a ve velké míře využívá standartu XPath 2.0 pro adresování částí dokumentu. Jazyk XQuery se těší stále větší oblibě a je hojně využíván v nativních XML databázích, ale jeho podporu a možnost použití můžeme nalézt i v nových verzích relačních databází, kde ho můžeme používat pro dotazování na XML daty uloženými v takové databázi.

2.4.1. Lokalizace uzlů

XQuery používá výrazy pro určení cesty pro lokalizování uzlů v XML datech. Výrazy pro určení cesty v XML jsou odvozeny z jazyka XPath 1.0 a jsou identické s výrazy pro určení cesty v jazyku XPath 2.0. Funkčnost výrazů pro určení cesty je úzce spjata s datovým modelem.

2.4.2. Vytváření uzlů

Jazyk XQuery dokáže vytvářet uzly dokumentu (dokumenty, elementy, atributy, textové uzly, instrukce ke zpracování, komentáře a jmenné prostory) se správnou syntaxí jazyka XML.

Dokumentový uzel vytvoříme pomocí konstruktoru. Konstruktor `dokument{}` má za následek vytvoření prázdného dokumentového uzlu. Pomocí konstruktorů lze vytvořit celý XML dokument včetně připojení stylesheetu a XML komenářů.

Příklad 15. – vytvoření XML dokumentu

```
Dokument {
<?xml- stylesheet type="text/xsl"
 href=http://www.nejakaadresa/nazev.xslt?>
<!-Tohle výborný zákazník.->
<zakaznik id='12'>
   <jmeno>Karel</jmeno>
   <prijmeni>Omacka</prijmeni>
   <telefon>987789256</telefon>
</zakaznik>
}
```

Konstruktory je samozřejmě možné kombinovat s ostatními XQuery dotazy a tvořit tak obsah XML dokumentu dynamicky.

Dotazy v XQuery jsou často používány pro zpracování informací z více zdrojů a z těchto zdrojů je vytvořen nový dokument.

2.4.3.Vstupní funkce

XQuery používá pro identifikaci dat dvě vstupní funkce.

1. `doc()`

Tato funkce vrací celý dokument určený pomocí URI. Přesněji řečeno vrací dokumentový uzel.

Příklad 16. – použití funkce doc()

```
doc(„zakaznici.xml”)
```

Tento dotaz nám zpřístupní data uložená v souboru *zakaznici.xml.*

2. `collection()`

Tato funkce vrací sekvenci uzlů určených pomocí URI. Tato funkce slouží k určení databáze k provedení dotazu.

2.4.5. FLWOR výrazy

FOR–LET–WHERE–ORDER BY–RETURN jsou výrazy, které se při používání jazyka XQuery velmi často používají ke konstrukci dotazů nad XML daty.

F – for

Příkaz `for` slouží pro procházení cyklem se zadanými podmínkami.

L – let

Příkaz `let` slouží pro přiřazování hodnot proměnným.

W – where

Klauzule `where` slouží pro omezení množiny výsledků.

O – order by

Klauzule `order by` setřídí výsledek vrácený příkazem `return` podle určitého kritéria (např. podle abecedy).

R – return

Příkaz `return` může jako návratovou hodnotu obsahovat jakýkoliv výraz.

2.4.6. Proměnná *at* pro určování pozice

Příkaz `for` podporuje proměnou `at` pro určování pozice, která identifikuje pozici dané položky ve vygenerovaném výrazu.

Příklad 17. – určení pozice ve vygenerovaném výrazu

```
for $zakaznik at $pozice in
 doc(„zakaznici.xml”)//prijmeni
return
<prijmeni pozice=”{$i}”>{string($prijmeni)}</prijmeni>
```

Výsledek tohoto dotazu může vypadat například takto:

```
<prijmeni pozice=”1”>Nováková</prijmeni>
<prijmeni pozice=”2”>Malá</prijmeni>
<prijmeni pozice=”3”>Selnerová</prijmeni>
```

V některých případech případech se můžeme setkat i s tím, že pozice uzlu určuje jeho význam. Nyní si na příkladu ukážeme, jak získat data z XHTML dokumentu, který reprezentuje následující tabulku.

Příklad 18. – ukázka XHTML tabulky se zákazníky

Jméno	**Příjmení**	**Telefon**
Hana	Černá	987132747
Irena	Sazimová	354789369
Lenka	Skořepová	648898898

Příklad 19. – XHTML zdrojový kód pro tabulku

```
<table>
  <thead>
    <tr>
      <td>jmeno</td>
      <td>prijmeni</td>
      <td>telefon</td>
    </tr>
  </thead>
  <tbody>
```

```
    <tr>
      <td>Hana</td>
      <td>Černá</td>
      <td>987132747</td>
    </tr>
    <tr>
      <td>Irena </td>
      <td>Sazimová </td>
      <td>354789369</td>
    </tr>
    <tr>
      <td>Lenka </td>
      <td>Skořepová</td>
      <td>648898898</td>
    </tr>
  </tbody>
</table>
```

V hlavičce této tabulky je určeno na jakém místě (ve kterém sloupci) se bude nacházet jaká informace, např. jméno se nachází vždy v prvním sloupci na každé řádce, příjmení ve druhém sloupci a telefon ve třetím sloupci.

V následujícím příkladě si ukážeme, jak pomocí proměnné pro určování pozice, můžeme vygenerovat XML dokument ve kterém si vytvoříme element zákazník a vložíme do něj odpovídající data z XHTML dokumentu.

Příklad 20. – práce s XHTML dokumentem

```
let $dokument:=doc(„zakaznici.xhtml”)//table[1]
for $radek in $dokument/tbody/tr
return
   <zakaznik>
   {
     for $zaznam at $pozice in $radek/td
      return
       element
       {
         lower-case(data($dokument/thead/tr/td[$pozice]))
       }
       {
         string($zaznam)
       }
    }
   </zakaznik>
```

Ukázka jednoho elementu vygenerovaného tímto dotazem

```
<zakaznik>
   <jmeno>Hana</jmeno>
   <prijmeni>Černá</prijmeni>
   <telefon>987132747</telefon>
</zakaznik>
```

2.4.7. Eliminace duplicitních hodnot pomocí *distinct–values()*

Data často obsahují duplicitní hodnoty, a proto se ve FLWOR výrazech používá funkce `distinct-values()`, která je určena k jejich odstranění.

2.4.8. Kombinace datových zdrojů pomocí *for* a *where*

Pro názorné znázornění příkladů použití jazyka XQuery si vytvoříme dva XML dokumenty, které budou uchovávat záznamy o zákaznících a jim provedeným službách v kadeřnictví. První XML dokument bude uchovávat záznamy o zákaznících navštěvujících kadeřnictví. Každého zákazníka bude reprezentovat jeden záznam, ve kterém bude uvedeno jméno, příjmení, pohlaví, narozeniny, telefon a jednoznačné identifikační číslo.

Příklad 21. – ukázka dokumentu zakaznici.xml

```
<zakaznci>
   <zakaznik>
      <id_zakaznika>1</id_zakaznika>
      <jmeno>Pavla</jmeno>
      <prijmeni>Nováková</prijmeni>
      <pohlavi>zena</pohlavi>
      <narozeniny>1981-03-03</narozeniny>
      <telefon>777532698</telefon>
   </zakaznik>
<zakaznici>
```

Ve druhém XML dokumentu budou ukládána data o jednotlivých návštěvách zákazníků v kadeřnictví. Každý záznam bude obsahovat id zákazníka, datum návštěvy, střih, barvu, poznámku, cenu, spropitné, identifikační číslo.

Příklad 22. – ukázka dokumentu zaznamy.xml

```
<zaznamy>
   <zaznam>
      <id_zaznamu>1</id_zaznamu>
      <id_zkaznika>12</id_zakaznika>
      <datum_navstevy>2005-02-05</datum_navstevy>
      <barva>červenozelený melír</barva>
      <strih>vzadu sestříháno, pěšinka uprostřed</strih>
      <poznamka>Zákaznici doporučen a prodán šampon
            Matrix. Příště se zeptat na spokojenost.
      </poznamka>
      <cena>650</cena>
      <spropitne>150</spropitne>
  </zaznam>
<zaznamy>
```

V praxi musíme často zkombinovat data z více datových zdrojů. Nyní si ukážeme použití jazyka XQuery pro vytvoření elementu *staly_zakaznik,* ve kterém bude uvedeno jméno a příjmení zákazníka, který navšívil kadeřnictví alespoň pětkrát, plus skutečný počet navštěv kadeřnictví. V tomto příkladě budeme používat data ze souborů *zakaznici.xml* a *zaznamy.xml*.

Příklad 23. – vypsání stálých zákazníků

```
for $zakaznik in document(„zakaznici.xml”)//zakaznik
let $zaznam := document(„zaznamy.xml”)
   //zaznam[id_zakaznika=$zakaznik/id_zakaznika]
where count($zaznam>4)
return
   <staly_zakaznik>
   {
       $zakaznik/jmeno
       $zakaznik/prijmeni
       <pocet_navstev>{count ($zaznam)}</pocet_navstev>
   }
   </staly_zakaznik>
```

Ukázka možného výsledku

```
<staly_zakaznik>
   <jmeno>Karel</jmeno>
   <prijmeni>Mařík</prijmeni>
    <pocet_navstev>7</pocet_navstev>
</staly_zakaznik>
```

2.4.9.Operátory

Operátory v XQuery jsou obdobné jako u jazyka XPath 2.0 a se používají standartním způsobem jako v ostatních jazycích.

2.4.9.1. Aritmetické operátory

„+“ – sčítání

„−“ – odečítání

„*“ – násobení

„`div`“ – dělení

„`idiv`“ – lze použít pouze pro hodnoty typu integer a jako výsledek vrací také integer

„`mod`“ – zbytek pod dělení

2.4.9.2. Operátory pro porovnávání

Operátory pro porovnávání si můžeme rozdělit do několika skupin, podle toho co porovnáváme.

Operátory pro porovnání hodnoty: `eq, ne, lt, le, gt, ge`.
Používá se pro skalární hodnoty nebo sekvence délky 1, jinak nastane vyjímka.

Operátory pro obecné porovnání: `=, !=, <, <=, >, >=`.

Rozdíl v zápisu porovnávacího operátoru je v kardinalitě.

př.

`kniha[cena gt 500]` – existuje právě jedna kniha s cenou větší než 500.

`kniha[cena>500]` – existuje alespoň jedna kniha s cenou větší než 500.

Porovnání uzlů

`is` a `isnot`

`$uzel1 is $uzel2` – vrátí true pokud obě proměnné odkazují na stejný uzel.

Porovnání pořadí

`$uzel1<<$uzel2` – $uzel1 je před $uzel2

`$uzel2>>$uzel2` – $uzel1 je za $uzel2

2.4.9.3. Logické operátory

Logické operátory `and`, `or` a `not` se používají například pro kombinaci jednotlivých podmínek.

2.4.10. Sekvence

Sekvence je ohraničená kulatými závorkami. ().

př. `(78, 99, 98)`

V číselných sekvencích je možnost použití mezí.

př. `(10, 5 to 8)==(10, 5, 6, 7, 8)`

Prvkem sekvence nemůže mít jiná sekvence.

př. `(1,10,(),(5, 4))==(1, 10, 5, 4)`

Sequence mohou obsahovat duplicitní hodnoty.

2.4.10.1. Operace na sekvencích

`union` – sjednocení

`intersect` – průnik

`except` – rozdíl

Tyto operátory také odstraní ve výsledku duplikace.

2.4.11. Kvantifikované výrazy

Výraz `some` vrací `true`, pokud podmínku splňuje alespoň jeden prvek z dané sekvence.

`some $i in (5,7,15) satisfies $i>10` – vrátí true

Výraz `every` vrací *true*, pokud podmínku splňují všechny prvky z dané sekvence.

`every $i in (5,7,15) satisfies $i>10` – vrátí false

2.4.12. Funkce v XQuery

V jazyce XQuery je možné používat buď knihovní funkce nebo si můžeme naprogramovat svoje vlastní funkce.

2.4.12.1. Knihovní funkce

Jazyk XQuery obsahuje celou řadu knihovních funckí. Např. dokument(), `collection(), avg(), sum(), count(), max(), min()` atd.

2.4.12.2. Uživatelsky definované funkce

V jazyku XQuery je samozřejmě možné definovat vlastní funkce.

Styl zápisu funkce:

```
define function nazev(typ parametr, typ parametr, …)
      returns typ
```

U definice vlastní funkce můžeme vynechat typy parametrů i návratový typ.

Příklad 24. – rekurzivní volání funkce

```
define function hloubka(element $e)
     returns xs:integer
{
   if(empty($e/*))
      then 1
   else max(for $h in $e/*
      return hloubka($h))+1
}
```

Příklad 25. – ukázka volání funkce

```
hloubka(dokument(„zakaznici.xml”))
```

Tato funkce nám vrátí maximální hloubku zanoření uzlu v dokumentu *zakaznici.xml*.

2.4.13. Typy v XQuery

XQuery je typový jazyk, který používá stejné typy jako XPath 2.0.

Základní typy – základni typy v Xquery jsou např. `integer, decima, float, double, string,...`

Uživatelské typy – uživatelské typy jsou často definovány ve schématech (např. typ kontakt)

Elementy a jejich názvy – použití elementů a jejich názvů jako typů (např. element jmeno).

Pokud chceme zjistit typ proměnné, můžeme použít funkci `typeswitch`.

Příklad 26. – použití typeswitch

```
typeswitch($x)
   case $i as xs:integer
      return <cislo>integer</cislo>
   case $d as xs:double
      return <cislo>double</cislo>
   default return <cislo>neznamy typ</cislo>
```

2.4.14. Ukázky použití XQuery

V dalším přikladu si ukážeme, jak vytvořit zprávu o jednotlivých návštěvách zákazníků v kadeřnictví. Jednotlivým záznamům bude přiřazen status, jak byl zákazník spokojen. Spokojenost určíme podle velikosti spropitného. Spropitné větší než 99 Kč bude znamenat, že zákazník byl velice spokojen. Spropitné 10–99 bude znamenat, že zákazník byl spokojen. Spropitné menší než 10 Kč znamená, že zákazník nebyl spokojen.

Příklad 27. – ukázka dotazu na rozdělení zákazníků podle spokojenosti

```
<zprava_o_navstevach>
   for $zakaznik in document („zakaznici.xml”)/zakaznik
   return
   {
   <zakaznik>
       $zakaznik/jmeno
       $zakaznik/prijmeni
       for $zaznam in document („zaznamy.xml”)
         /*/zaznam[id_zakaznika=$zazaznik/id_zakaznika]
       return
         <navsteva>
            {
               $zaznam/datum_navstevy
               <spokojenost>
               {
                    if($zaznam/spropitne>99)
                      then „velmi spokojen”
                    else if($zaznam/spropitne>9)
                      then „spokojen”
                    else „nespokojen”
               }
              </spokojenost>
            }
       </navsteva>
    }
  </zakaznik>
 </zprava_o_navstevach>
```

Ukázka výsledku:

```
<zprava_o_navstevach>
    <zakaznik>
       <jmeno>Karel</jmeno>
       <prijmeni>Mařík</prijmeni>
        <navsteva>
            <datum_navstevy>2005-02-02</datum_navstevy>
            <spokojenost>spokojen</spokojenost>
         </navsteva>
         <navsteva>
             <datum_navstevy>2005-03-03</datum_navstevy>
             <spokojenost>velmi spokojen</spokojenost>
         </navsteva>
     </zakaznik>
</zprava_o_navstevach>
```

V tomto příkladě máme velikost spropitného pro určení spokojenosti zákazníka určenou přímo v dotazu, ale je samozřejmě možné vytvořit další XML dokument, kde by byly parametry pro rozdělení zákazníků uloženy a prováděný dotaz by si je z tohoto souboru načetl.

2.5. XqueryX

XQueryX je XML reprezentací XQuery. XQueryX není vhodný formát pro zpracovávání lidmi (vytváření nebo modifikace dotazů), ale o to vhodnější je pro zpracovávání různými programy, které jsou schopny analyzovat XML data. Velikou výhodou XQueryX je právě to, že je ve formátu XML, a proto mohou vytvářet, interpretovat a modifikovat dotazy v XQueryX standardní nástroje pro práci s XML daty.

Příklad 28. – ukázka dotazu zapsaného v XQuery

```
<bib>
 {
  for $b in
    doc(„http://bstore1.example.com/bib.xml“)/bib/book
  where $b/publisher = „Addison-Wesley“
        and $b/@year > 1991
  return
    <book year=„{$b/@year}“>
     {$b/title}
    </book>
 }
</bib>
```

Příklad 29. Stejný dotaz jako v příkladě 21.zapsaný pomocí XQueryX

```
<?xml version=„1.0“ encoding=„UTF-8“?>
<?xml-stylesheet type=„text/xsl“ href=„xqueryx.xsl“?>
<xqx:module xmlns:xsi=„http://www.w3.org/2001/XMLSchema-instance“
      xmlns:xqx=„http://www.w3.org/2005/02/XQueryX“
      xsi:schemaLocation=„http://www.w3.org/2005/02/XQueryX
xqueryx.xsd“>
  <xqx:mainModule>
    <xqx:queryBody>
      <xqx:expr xsi:type=„xqx:elementConstructor“>
        <xqx:tagName>bib</xqx:tagName>
        <xqx:elementContent>
          <xqx:exprWrapper>
          <xqx:expr xsi:type=„xqx:flworExpr“>
            <xqx:forClause>
              <xqx:forClauseItem>
                <xqx:typedVariableBinding>
                  <xqx:varName>b</xqx:varName>
                </xqx:typedVariableBinding>
                <xqx:forExpr>
                  <xqx:expr xsi:type=„xqx:pathExpr“>
                    <xqx:argExpr>
                    <xqx:expr xsi:type=„xqx:functionCallExpr“>
                      <xqx:functionName>doc</xqx:functionName>
                      <xqx:parameters>
                        <xqx:exprWrapper>
                          <xqx:expr
                            xsi:type=„xqx:stringConstantExpr“>
                          </xqx:expr>
                        </xqx:exprWrapper>
                      </xqx:parameters>
                    </xqx:expr>
                    </xqx:argExpr>
                    <xqx:stepExpr>
                      <xqx:xpathAxis>child</xqx:xpathAxis>
                      <xqx:nameTest>bib</xqx:nameTest>
                    </xqx:stepExpr>
                    <xqx:stepExpr>
                      <xqx:xpathAxis>child</xqx:xpathAxis>
                      <xqx:nameTest>book</xqx:nameTest>
                    </xqx:stepExpr>
                  </xqx:expr>
```

```
    </xqx:forExpr>
  </xqx:forClauseItem>
</xqx:forClause>
<xqx:whereClause>
  <xqx:expr xsi:type=„xqx:operatorExpr“>
    <xqx:infixOp/>
    <xqx:opType>and</xqx:opType>
    <xqx:parameters>
      <xqx:exprWrapper>
        <xqx:expr xsi:type=„xqx:operatorExpr“>
          <xqx:infixOp/>
          <xqx:opType>=</xqx:opType>
          <xqx:parameters>
            <xqx:exprWrapper>
              <xqx:expr xsi:type=„xqx:pathExpr“>
                <xqx:argExpr>
                  <xqx:expr xsi:type=„xqx:varRef“>
                    <xqx:name>b</xqx:name>
                  </xqx:expr>
                </xqx:argExpr>
                <xqx:stepExpr>
                  <xqx:xpathAxis>child</xqx:xpathAxis>
                  <xqx:nameTest>publisher</xqx:nameTest>
                </xqx:stepExpr>
              </xqx:expr>
            </xqx:exprWrapper>
            <xqx:exprWrapper>
            </xqx:exprWrapper>
          </xqx:parameters>
        </xqx:expr>
      </xqx:exprWrapper>
      <xqx:exprWrapper>
        <xqx:expr xsi:type=„xqx:operatorExpr“>
          <xqx:infixOp/>
          <xqx:opType>&gt;</xqx:opType>
          <xqx:parameters>
            <xqx:exprWrapper>
              <xqx:expr xsi:type=„xqx:pathExpr“>
                <xqx:argExpr>
                  <xqx:expr xsi:type=„xqx:varRef“>
                    <xqx:name>b</xqx:name>
                  </xqx:expr>
                </xqx:argExpr>
                </xqx:stepExpr>
              </xqx:expr>
            </xqx:exprWrapper>
            <xqx:exprWrapper>
              <xqx:expr
              </xqx:expr>
            </xqx:exprWrapper>
          </xqx:parameters>
        </xqx:expr>
      </xqx:exprWrapper>
    </xqx:parameters>
  </xqx:expr>
</xqx:whereClause>
<xqx:returnClause>
  <xqx:expr xsi:type=„xqx:elementConstructor“>
```

```
                <xqx:tagName>book</xqx:tagName>
                <xqx:attributeList>
                  <xqx:attributeConstructor>
                    <xqx:attributeName>year</xqx:attributeName>
                    <xqx:attributeValueExpr>
                      <xqx:expr xsi:type=„xqx:pathExpr“>
                        <xqx:argExpr>
                          <xqx:expr xsi:type=„xqx:varRef“>
                            <xqx:name>b</xqx:name>
                          </xqx:expr>
                        </xqx:argExpr>
                        <xqx:stepExpr>
                          <xqx:xpathAxis>attribute</xqx:xpathAxis>
                          <xqx:nameTest>year</xqx:nameTest>
                        </xqx:stepExpr>
                      </xqx:expr>
                    </xqx:attributeValueExpr>
                  </xqx:attributeConstructor>
                </xqx:attributeList>
                <xqx:elementContent>
                  <xqx:exprWrapper>
                    <xqx:expr xsi:type=„xqx:pathExpr“>
                      <xqx:argExpr>
                        <xqx:expr xsi:type=„xqx:varRef“>
                          <xqx:name>b</xqx:name>
                        </xqx:expr>
                      </xqx:argExpr>
                      <xqx:stepExpr>
                        <xqx:xpathAxis>child</xqx:xpathAxis>
                        <xqx:nameTest>title</xqx:nameTest>
                      </xqx:stepExpr>
                    </xqx:expr>
                  </xqx:exprWrapper>
                </xqx:elementContent>
              </xqx:expr>
            </xqx:returnClause>
          </xqx:expr>
          </xqx:exprWrapper>
        </xqx:elementContent>
      </xqx:expr>
    </xqx:queryBody>
  </xqx:mainModule>
</xqx:module>
```

2.6. XUpdate

Xupdate je specifikace vyvíjená iniciativou XML:DB, navržená pro snadnou aktualizaci XML dokumentů. Specifikace XUpdate není určena jen pro práci s jednotlivými XML dokumenty, ale i pro použití v XML databázích. Pro vyjádření cesty v dokumentu XUpdate využívá jazyka XPath. XUpdate nejprve najde v dokumentu požadované uzly a tyto uzly jsou poté aktualizovány na nové hodnoty. Modifikace, které XUpdate umožňuje budou předvedeny v následujících příkladech.

Příklad 30. – ukázka struktury XML dokumentu se kterým budeme pracovat

```
<adresar>
   <kontakt id =”1”>
    <!-- Tohle je uzivatelske jmeno-->
      <jmeno>
         <krestni>Michal</krestni>
         <prijmeni>Hora</prijmeni>
      </jmeno>
      <mesto>Strakonice</mesto>
      <stat>Česká republika</stat>
      <telefon type=”domaci”>789878987</telefon>
      <telefon type=”pracovni”> 88977887</telefon>
      <poznamka>
        <![CDATA[Tohle je nový uživatel]]></poznamka>
   </kontakt>
</adresar>
```

Vložení elementu před (Insert element before)

Příklad 31. – přidání elementu prostredni jméno před element prijmeni

```
<xupdate:modifications version=”1.0.”
  xmlns:xupdate=”http://www.xmldb.org/xupdate”>
  <xupdate:insert-before
     select=”/adresar/kontakt[@id=1]/jmeno/prijmeni”>
       <xupdate:element name
          „prostredni”>Václav</xupdate:element>
  </xupdate:insert-before>
</xupdate:modifications>
```

Vložení elementu za (Insert element after)

Příklad 32. – přidání elementu telefon s atributem mobilni za element telefon s atributem domaci

```
<xupdate:modifications version="1.0."
  xmlns:xupdate="http://www.xmldb.org/xupdate">
   <xupdate:insert-after
      select="/adresar/kontakt[@id=1]/
              telefon[@type='domaci']">
      <xupdate:element name="telefon">
         <xupdate:attribute name="type">
           mobilni</xupdate:attribute>789789787
   </xupdate:element>
  </xupdate:insert-after>
</xupdate:modifications>
```

Přidání elementu (Append element)

Příklad 33. – přidání elementu psc do záznamu kontakt

```
<xupdate:modifications version="1.0."
  xmlns:xupdate="http://www.xmldb.org/xupdate">
    <xupdate:append select="/adresar/kontakt[@id=1]">
       <xupdate:element name="psc">38719
       </xupdate:element>
   </xupdate:append>
</xupdate:modifications>
```

Vložení atributu (Insert attribute)

Příklad 34. – Přidání atributu linka k elementu telefon s atributem pracovni.

```
<xupdate:modifications version="1.0."
  xmlns:xupdate="http://www.xmldb.org/xupdate">
   <xupdate:append
      select="/adresar/kontakt[@id=1]/
              telefon[@type='pracovni']">
     <xupdate:attribute name="linka">110
     </xupdate:attribute>
   </xupdate:append>
</xupdate:modifications>
```

Vložení textového obsahu (Insert text content)

Příklad 35. – přidáme text „stredni Evropa" k elementu stat.

```
<xupdate:modifications version="1.0."
  xmlns:xupdate="http://www.xmldb.org/xupdate">
   <xupdate:append
      select="/adresar/kontakt[@id=1]/stat">
      <xupdate:text>stredni Evropa</xupdate:text>
   </xupdate:append>
</xupdate:modifications>
```

Vložení XML bloku (Inser XML block)

Příklad 36. – přidání nového kontaktu do adresáře

```
<xupdate:modifications version="1.0."
  xmlns:xupdate="http://www.xmldb.org/xupdate">
   <xupdate:append select="/adresar">
      <xupdate:element name="kontakt">
         <xupdate:attribute name="id">2
         </xupdate:attribute>
         <jmeno>
            <krestni>Alfons</krestni>
            <prijmeni>Mucha</krestni>
         </jmeno>
         <mesto>Praha</mesto>
         <stat> Česká republika</stat>
         <telefon type="domaci">789788548</telefon>
      </xupdate:element>
   </xupdate:append>
</xupdate:modifications>
```

Vložení instrukce pro zpracování (Insert a processing instruction)

Příklad 37. – vložení XML instrukce pro zpracování do kořenové části XML dokumentu

```
<xupdate:modifications version="1.0."
  xmlns:xupdate="http://www.xmldb.org/xupdate">
   <xupdate:insert-before select="/adresar">
      <xupdate:processing-instruction
         name="cocoon-process">type="xsp"
      </xupdate:processing-instruction>
   </xupdate:insert-before>
</xupdate:modifications>
```

Vložení komentáře (Insert a comment)

Příklad 38. – vložení komentáře „tohle je telefon do práce“ před element telefon s atributem pracovni

```
<xupdate:modifications version="1.0."
  xmlns:xupdate="http://www.xmldb.org/xupdate">
   <xupdate:insert-before
      select="/adresar/kontakt[@id=1]
              /telefon[@type='pracovni']">
      <xupdate:comment> tohle je telefon do práce
      </update:comment>
   </update:insert-before>
</update:modifications>
```

Vložení obsahu CDATA (Insert CDATA content)

Příklad 39. – přidání elementu poznamka, jehož obsahem je sekce CDATA s obsahem ohraničeným HTML tagy

```
<xupdate:modifications version="1.0."
  xmlns:xupdate="http://www.xmldb.org/xupdate">
   <xupdate:append select="/adresar/kontakt[@id=1]">
      <xupdate:element name="poznamka">
         <xupdate:cdata><![CDATA[<b>Obchodní
                        kontakt</b>]]>
         <xupdate:cdata>
      </xupdate:element>
   </xupdate:append>
</xupdate:modifications>
```

Aktualizace elementu (Update element)

Příklad 40. – Změna křestního jméno u kontaktu s id=1 na Jan

```
<xupdate:modifications version="1.0."
  xmlns:xupdate="http://www.xmldb.org/xupdate">
   <xupdate:update select="/adresar/kontakt[@id=1]
                           /jmeno/krestni">Jan
   </xupdate:update>
<xupdate:modifications>
```

Aktualizace atributu (Update attribute)

Příklad 41. – změna typu telefonního čísla 88977887 u kontaktu s id=1 z pracovni na mobilni

```
 <xupdate:modifications version="1.0."
  xmlns:xupdate="http://www.xmldb.org/xupdate">
   <xupdate:update select="/adresar/kontakt[@id=1]
                           /telefon[.=' 88977887']
                           /@type">mobilni
   </xupdate:update>
 </xupdate:modifications>
```

Aktualizace komentáře (Update comment)

Příklad 42. – změna komentáře před elementem jmeno, který začíná „Tohle"

```
 <xupdate:modifications version="1.0."
   xmlns:xupdate="http://www.xmldb.org/xupdate">
   <xupdate:update
           select="/adresar/kontakt[@id=1]
                   /comment()[starts-with(.,'Tohle')]">
                   Tohle je uzivatelske jmeno obchodniho
                   partnera.
   </xupdate:update>
</xupdate:modifications>
```

Aktualizace obsahu sekce CDATA (Update CDATA content)

Příklad 43. – změna sekce CDATA u elementu poznamka

```
<xupdate:modifications version="1.0."
  xmlns:xupdate="http://www.xmldb.org/xupdate">
   <xupdate:update select="/adresar/kontakt[@id=1]
                          /poznamka/text()">
                          <![CDATA[<i>Změněno</i>]]>
   </xupdate:update>
 </xupdate:modifications>
```

Přejmenování elementu (Rename element)

Příklad 44. – přejmenování elementu poznamka na komentar

```
<xupdate:modifications version="1.0."
  xmlns:xupdate="http://www.xmldb.org/xupdate">
   <xupdate:rename select="/adresar/kontakt[@id=1]
                          /poznamka">komentar
   </xupdate:rename>
</xupdate:modifications>
```

Přejmenování atributu (Rename attribute)

Příklad 45. – přejmenování atributu type u elementu telefon na umisteni

```
<xupdate:modifications version="1.0."
  xmlns:xupdate="http://www.xmldb.org/xupdate">
   <xupdate:rename select="/adresar/kontakt[@id=1]
                          /telefon/@type">umisteni
   </xupdate:rename>
</xupdate:modifications>
```

Smazání elementu (Delete element)

Příklad 46. – smazání všech elementů telefon

```
<xupdate:modifications version="1.0."
  xmlns:xupdate="http://www.xmldb.org/xupdate">
   <xupdate:remove
      select="/adresar/kontakt[@id=1]/telefon"/>
</xupdate:modifications>
```

Smazání atributu (Delete atribute)

Příklad 47. – smazání všech atributů type z elementu telefon

```
<xupdate:modifications version="1.0."
  xmlns:xupdate="http://www.xmldb.org/xupdate">
   <xupdate:remove
      select="/adresar/kontakt[@id=1]/telefon/@type"/>
</xupdate:modifications>
```

Smazání textového obsahu elementu (Delete text content of an element)

Příklad 48. – smazání obsahu elementu stat

```
<xupdate:modifications version="1.0."
  xmlns:xupdate="http://www.xmldb.org/xupdate">
   <xupdate:remove
      select="/adresar/kontakt[@id=1]/stat/text()"/>
</xupdate:modifications>
```

Smazávání komentáře (Delete comment)

Příklad 49. – smazání prvního komentáře před elementem jmeno

```
<xupdate:modifications version="1.0."
  xmlns:xupdate="http://www.xmldb.org/xupdate">
   <xupdate:remove
      select="/adresar/kontakt[@id=1]/comment()[1]"/>
</xupdate:modifications>
```

Smazávání obsahu sekce CDATA (Delete CDATA content)

Příklad 50. – smazání obsahu sekce CDATA v elementu poznamka

```
<xupdate:modifications version="1.0."
  xmlns:xupdate="http://www.xmldb.org/xupdate">
   <xupdate:remove
      select="/adresar/kontakt[@id=1]/note/text()"/>
</xupdate:modifications>
```

Kopírování uzlu (Copying a node)

Příklad 51. – zkopírování uzlu stat a umístění jeho kopie za uzel poznamka

```
<xupdate:modifications version="1.0."
  xmlns:xupdate="http://www.xmldb.org/xupdate">
   <xupdate:variable name="stat"
      select="/adresar/kontakt[@id=1]/stat"/>
   <xupdate:insert-after
      select="/adresar/kontakt[@id=1]/poznamka">
      <xupdate:value-of select="$stat"/>
   </xupdate:insert-after>
</xupdate:modifications>
```

Přesunutí uzlu (Moving a node)

Příklad 52. – přesunutí uzlu poznamka před uzel telefon s atributem domaci

```
<xupdate:modifications version="1.0."
  xmlns:xupdate="http://www.xmldb.org/xupdate">
   <xupdate:variable  name="poznamka"
      select="/adresar/kontakt[@id=1]/poznamka"/>
   <xupdate:remove
      select="/adresar/kontakt[@id=1]/poznamka"/>
   <xupdate:insert-before
      select=„/adresar/kontakt[@id=1]/telefon[@domaci]">
        <xupdate:value-of select=„$poznamka"/>
   </xupdate:insert-before>
</xupdate:modifications>
```

2.7. SQL/XML

Dotazovací jazyk SQL/XML je vyvíjen skupinou H2.3 Task Group, ve které participují firmy Oracle, IBM, Microsoft, DataDirect Technologies a Sysbase. Z vyjmenovaných firem je zřejmé, že SQL/XML je vyvíjen pro klasické „databázové systémy". Tento dotazovací jazyk je rozšířením objektově relačního jazyka SQL.

2.7.1. Datový typ XML

V SQL/XML můžeme XML dokument uložit jako hodnotu typu VARCHAR, CLOB nebo jako XML. XML je nově zavedený typ, jehož hodnotou může být dokument, element, textový uzel nebo smíšený obsah.

Příklad 53. – v tabulce zakaznici bude vytvořen sloupec adresa typu XML

```
CREATE TABLE zakaznici
   (id integer,
     adresa XML)
```

2.7.2. Vytváření hodnot typu XML z relačních dat

Pro vytváření hodnot typu XML jsou k dispozici tyto operátory:

`XMLELEMENT` – vytvoření nového XML elementu určeného jména

`XMLATTRIBUTES` – definuje atributy vytvářeného elementu

`XMLFOREST` – vytvoření posloupnosti XML dokumentů

`XMLCONCAT` – vytvoření jedné hodnoty s více elementů

`XMLROOT` – vytvoření kořene XML dokumentu

`XMLCOMMENT` – vytvoření XML komentáře

`XMLPI` – vytvoření XML instrukce ke zpracování

`XMLPARSE` – rozloží řetězec jako XML a vrátí výslednou XML strukturu

3. Nativní XML databázové systémy

Termín nativní XML databáze má původ v komerční sféře. Termín „*nativní XML databázové systémy*“ se poprvé objevil v marketingové kampani firmy AG Software (http://softwareag.com) při propagaci nativního XML databázového systému Tamino. Tento termín se rychle rozšířil a převzaly jej i další firmy vyvíjející obdobný software.

Nativní XML databázové systémy jsou databáze specializované na ukládání XML dokumentů. XML dokumenty jsou v nich ukládány v nativní (přirozené) podobě. V těchto databázích se místo s tabulkami pracuje s kolekcemi XML dokumentů. Dokumenty v kolekci mohou, ale nemusí odpovídat určitému schématu. S libovolnou částí XML dokumentu lze manipulovat (výběr, změna, přidání, smazání). Nativní XML databázi lze snadno indexovat a prohledávat.

3.1. Definice nativního XML databázové systému

Tady bych rád uvedl definici nativního XML databázového systému od XML:DB Initiative.

Nativní XML databáze definuje logický model pro XML dokument a umožňuje ukládat a znovu získávat dokumenty odpovídající danému modelu. Model musí obsahovat minimálně elementy, atributy, PCDATA a identifikaci dokumentu. Příkladem takových dokumentů je model XPath, XML infoset a modely vycházející z DOM či SAX.

Nativní XML databáze má jako základní logickou jednotku XML dokument stejně jako má relační databáze základní logickou jednotku řádek v tabulce.

Není potřeba žádné speciální fyzické úložiště dat. Například je možné použít relační nebo objektově orientované databáze či indexovaný soubor.

Z této definice je patrné, že nativní XML databáze nepředstavuje nový databázový model, ale je to spíše „doplněk“, který poskytuje vývojářům robustní úložiště pro manipulaci s XML daty.

3.2. Architektura nativní XML databázových systémů

Architekturu nativních XML databázových systémů dělíme na dvě kategorie. Na textovou a modelovou architekturu.

3.2.1. Nativní XML databázové systémy s textovou architekturou

Nativní XML databázové systémy s textovou architekturou ukládají XML dokumenty jako prostý text, který může být uložen např. v souboru v souborovém systému počítače, v poli typu BLOB (Binary Large OBject) či CLOB (Character Large OBject) v relační databázi nebo v jiném textovém formátu. Jedním s rysů nativních XML databázových systémů s textovou architekturou jsou indexy umožňující přechod do jakéhokoliv místa v jakémkoliv XML dokumentu. Tyto indexy zaručují databázi rychlost pří vykonávání dotazů a načítání XML dokumentů nebo jejich částí. Nativní XML databázové systémy s textovou architekturou jsou podobné hierarchickým databázím, které při načítání dat hierarchické struktury mohou předčit relační databáze. Ale v případě, že nebudou načítána data v odpovídající hierarchické struktuře, dostaví se problémy s nízkým výkonem databáze.

3.2.2. Nativní XML databázové systémy s modelovou architekturou

Modelové nativní XML databázové systémy si vytvoří z XML dokumentu objektový model, který uloží do databáze. Některé nativní XML databázové systémy ukládají tento model do relační nebo objektově orientované databáze a některé si ho ukládají do vlastního formátu. Pokud ukládáme data jako DOM v relační databázi nejspíše vzniknou tabulky jako *elements, attributes, PCDATA* atd.

Výkon nativních XML databázových systémů s modelovou strukturou hodně závisí na výkonu databáze, nad kterou jsou vybudovány.

Pokud budeme srovnávat výkon nativních XML databázových systémů s modelovou strukturou a nativní XML databázové systémy s textovou architekturou, tak zjistíme, že je výkon téměř stejný. Nativní XML databázové systémy s modelovou strukturou jsou výkonnější při načítání XML dokumentů do DOM a

nativní XML databázové systémy s textovou architekturou jsou výkonnější při načítání XML dokumentů v textové podobě.

3.3. Kolekce dokumentů

Kolekci dokumentů v nativní XML databázi si můžeme představit jako tabulku v relační databázi. Některé nativní XML databázové systémy nevyžadují schéma, které by určovalo typ dat v dané kolekci a to umožňuje do kolekce uložit různorodé XML dokumenty bez ohledu na schéma (tato vlastnost se nazývá *schema independent – nezávislost na schématu)*. *Schema independent* umožňuje větší flexibilitu při návrhu databáze a jednoduší návrh aplikace používající databázi, ale stinnou stránkou této možnosti je nízká integrita dat.

3.4. Dotazování nad nativními XML databázemi

Pro dotazování nad nativními XML databázemi musely vzniknou nové dotazovací jazyky určené pro práci s daty ve formátu XML. Nejvíce se osvědčil jazyk XPath, ale stále více se prosazuje jazyk XQuery, který má s jazykem XPath mnoho společného. Dotazovacím jazykům je věnována kapitola 2. Dotazovací jazyky.

3.5. Editace dat v nativních XML databázových systémech

Editace dat v nativních XML databázových systémech je slabší stránkou těchto systémů. V mnoha XML databázových systémech nejde provádět editace „přímo“ a musí se nejdříve načíst XML dokument z databáze, provést změnu v načteném dokumentu (pomocí nástrojů pro práci s XML daty) a poté uložit editovaný dokument zpět do databáze. Některé nativní databázové systémy již implementovaly jazyk určený pro „přímou“ editaci dat XUpdate od XML:DB Initiative. V současné době se pracuje na přidání možnosti editace do jazyka XQuery.

3.6. Indexování v nativních XML databázových systémech

Indexování ukládaných dat slouží ke zrychlení provádění operací nad databází. Nativní XML databázové systémy jsou velmi rychlé pokud zpracovávají XML dokument v takové formě v jaké je uložen. Pokud je potřeba zpracovat data jinak než jsou uloženy, nastává problém jak je rychle a efektivně zpracovat. Nativní XML databázové systémy proto často indexují všechny elementy a atributy. Tato indexace má za následek zrychlení výběrových operací, ale stinnou stránkou je zpomalení aktualizace dat.

3.7. Transakce, zamykání a souběžný přístup

Transakce podporují asi všechny nativní databázové systémy. Zamykání je většinou pouze na úrovní celého XML dokumentu. Je tedy problematické, pokud k jednomu XML dokumentu chce přistupovat více uživatelů najednou. Zamykání pouze částí XML dokumentů není v současnosti možné, ale v budoucnu lze tuto možnost očekávat.

3.8. Round–Tripping

Round–Tripping je velmi důležitá vlastnost nativních XML databázových systémů. Round–Tripping znamená, že pokud do databáze uložíme XML dokument a následně ho budeme chtít z databáze načíst, bude načtený dokument identický s dokumentem uloženým. Round–Tripping je důležitý pro aplikace, pro které jsou důležité CDATA, komentáře, instrukce ke zpracování a kde závisí na konkrétním pořadí elementů v dokumentu.

3.9. Referenční integrita

Referenční integrita v relační databázi znamená, že cizí klíč ukazuje na platný primární klíč (existuje odpovídající záznam s primárním klíčem). V nativní XML databázi referenční integrita znamená použití odkazu na platný dokument nebo jeho část. Odkazy v XML dokumentech jsou vytvářeny například pomocí atributů ID/IDREF, key/keyref nebo pomocí XLink. Odkazy vyjadřující referenční integritu lze rozdělit na dvě skupiny a to na interní ukazatele (ukazatele uvnitř jednoho

dokumentu) a externí ukazatele (ukazatele mezi dokumenty). Referenční integrita interních odkazů je celkem bezproblémová a v současné době ji podporuje většina nativních XML databázových systémů. Referenční integritu externích odkazů nelze zatím plně zajistit a nativních XML databázové systémy ji podporují velice zřídka, ale v do budoucna se počítá s tím, že i referenční integritu externích odkazů by měla mít v těchto systémech dobrou podporu.

3.10. API – Application Programming Interfaces

Aby bylo možné s nativními XML databázemi dobře pracovat, je nutné, aby měly nějaké rozhraní (obdoba ODBC), které poskytuje připojení k databázi, procházení dat, vykonávání dotazů atd. Asi nejznámějším API pro nativní XML databáze je XML:DB API, které nabízí přímý přístup k datům v databázi.

Pro pohodlnější přístup k databázím, které podporují XML:DB API slouží aplikace XMLdbGUI, která umožňuje jednoduchou správu nativních XML databází. Možnosti XMLdbGUI:

– přidávání, editace, mazání dokumentů

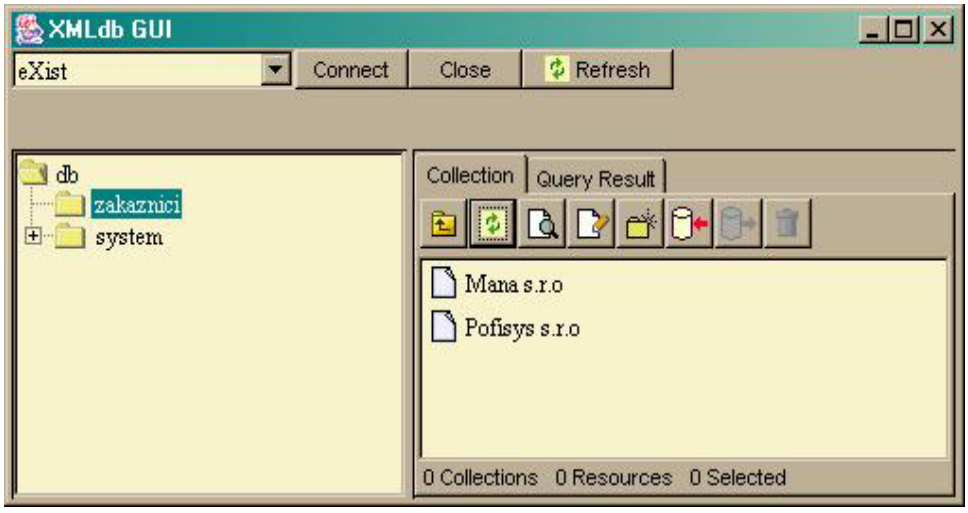
– přidávání a mazání kolekcí

– import a export dokumentů z/do systému souborů

– vyhledávání dat pomocí XPath, export výsledku do souboru

– modifikace dokumentů jazykem XUpdate

Obrázek 3. – ukázka grafického rozhraní XMLdbGUI



3.11. Příklady použití XML databází

- katalogy zboží
- portály
- sklady dat
- řízení obsahu v prostředí internetu
- integrace zasílání dat a aplikací
- vyhledávání v textech
- dolování dat
- depozitáře pro management konfigurací

3.12. Nativní XML databáze eXist

Developer: Wolfgang Meier

URL: http://exist.sourceforge.net

Licence: Open Source

Typ databáze: Vlastní model

eXist je nativní XML databáze, která využívá vlastní úložiště dat, konkrétně používá B+ stromy a stránkovací soubory. Dokumenty jsou hierarchicky uspořádané v kolekcích (uspořádání je podobné souborovému systému). Kolekce mohou obsahovat další kolekce, přičemž nezáleží na nějakém zvláštním schématu nebo dokumentovém typu.

eXist může fungovat ve 3 základních režimech:

„Standalone“ režim – databáze je spuštěna ve vlastním procesu

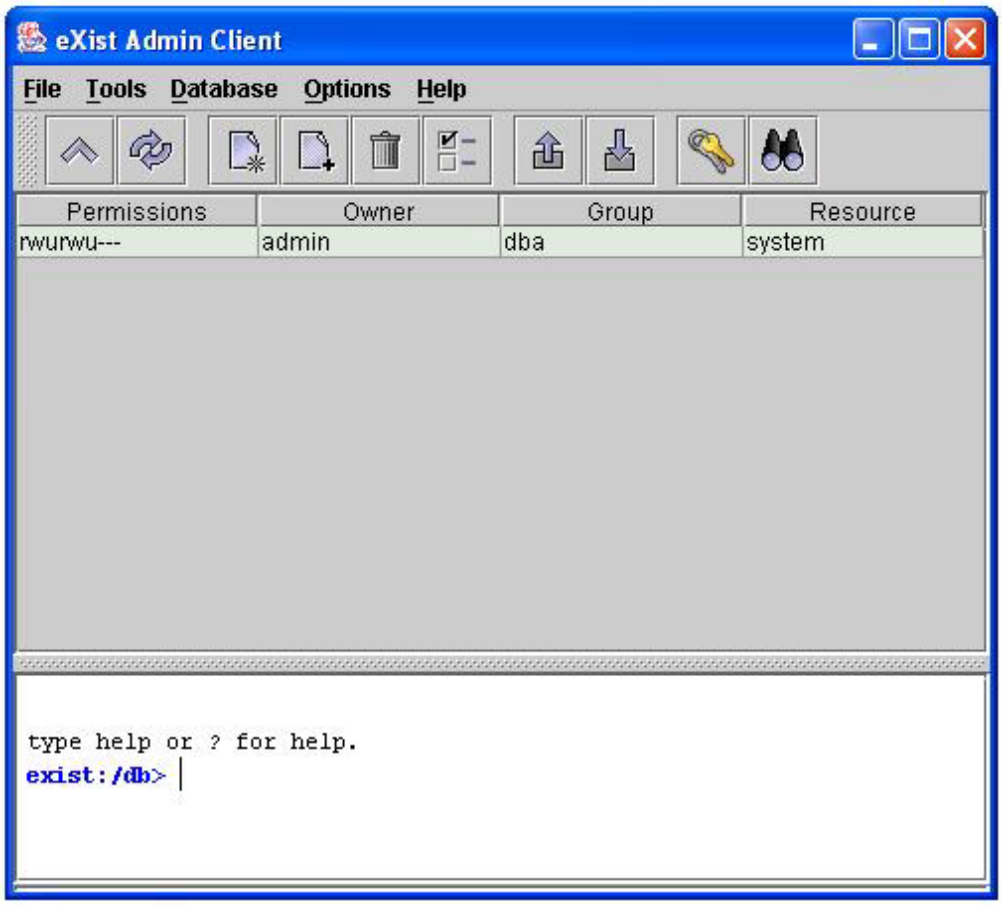
„Embedded“ režim – databáze je přilinkována ke klientské aplikaci a je spuštěna ve společném procesu

„Servlet“ režim – databáze je spuštěna formou servetů a je kontrolována servlet strojem (Servlet engine)

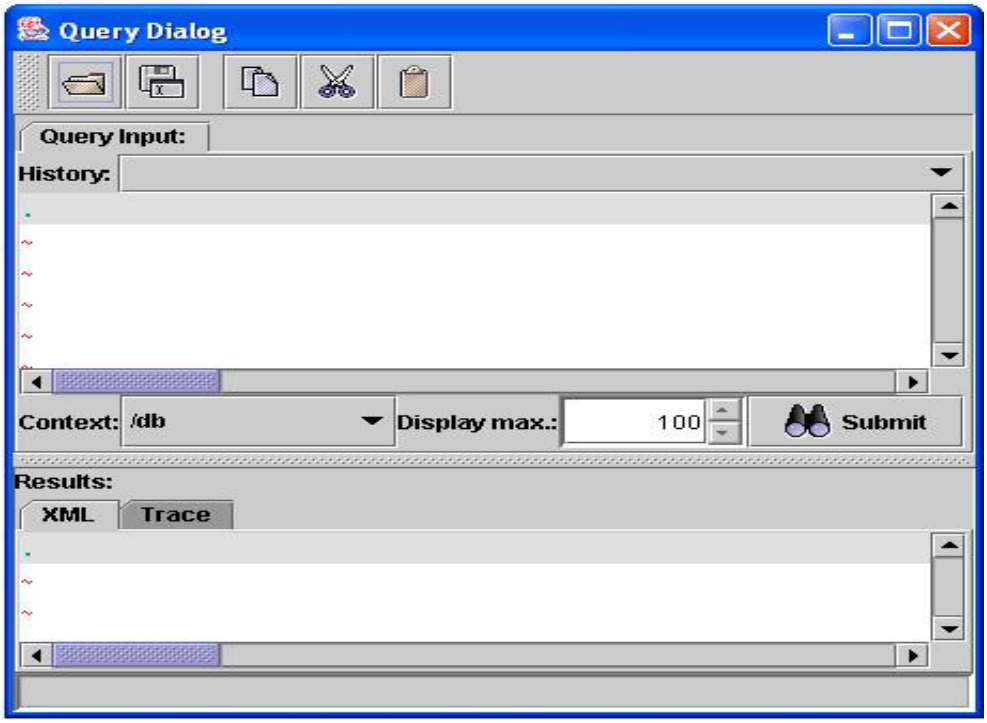
Jednotlivé režimy ovlivňují množinu aktivních komunikačních standartů, například SOAP a Webdav protokoly jsou aktivní pouze v servlet režimu.

Obrázek 4. – přihlašovací okno eXist

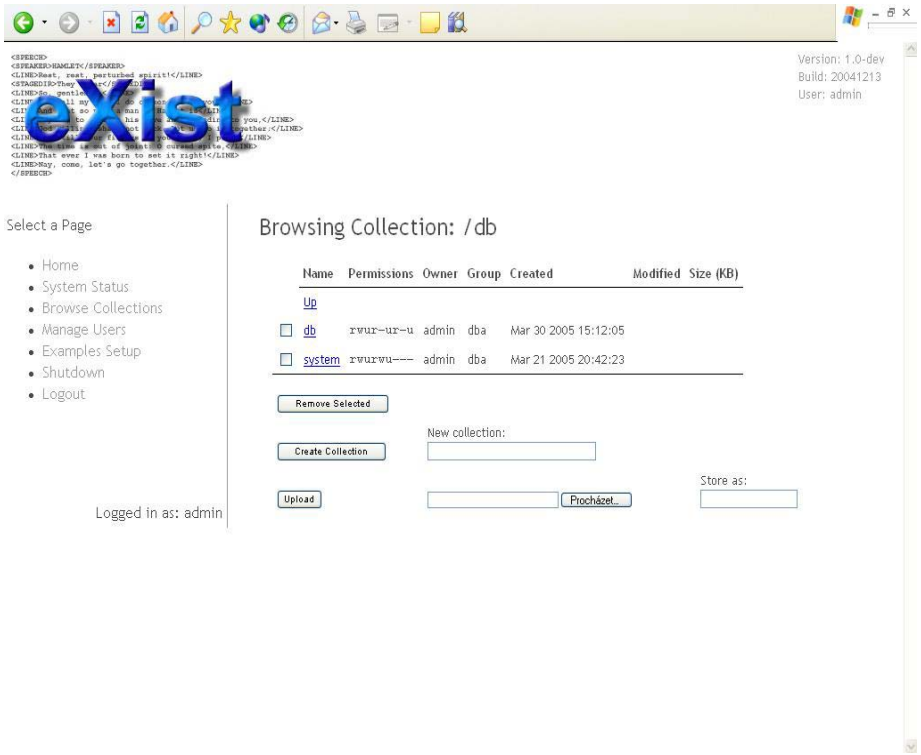
Obrázek 5. – Základní administrátorské rozhraní eXist



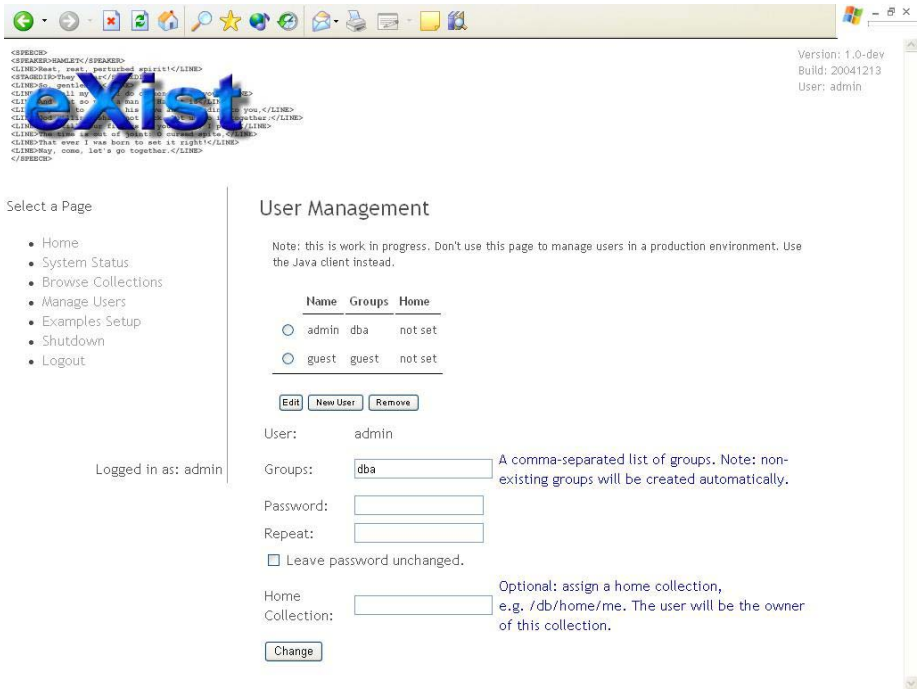
Obrázek 6. – dotazovací dialog eXist



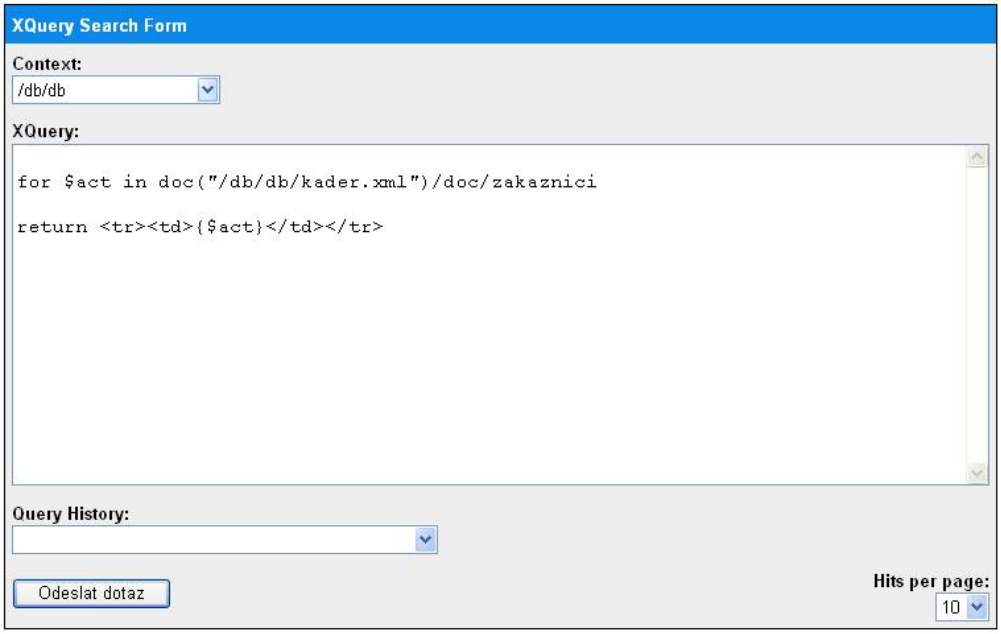
Obrázek 7. – správa kolekcí eXist přes webové rozhraní



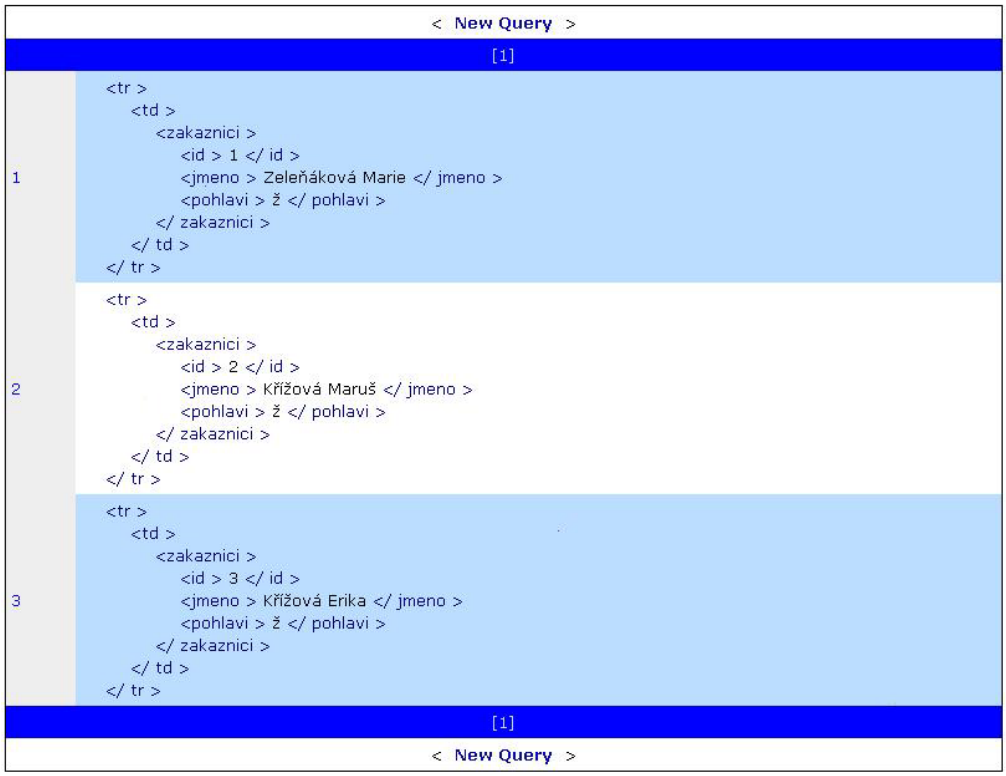
Obrázek 8. – správa uživatelů eXist přes webové rozhraní



Obrázek 9. – dotazovací formulář XQuery přes webové rozhraní



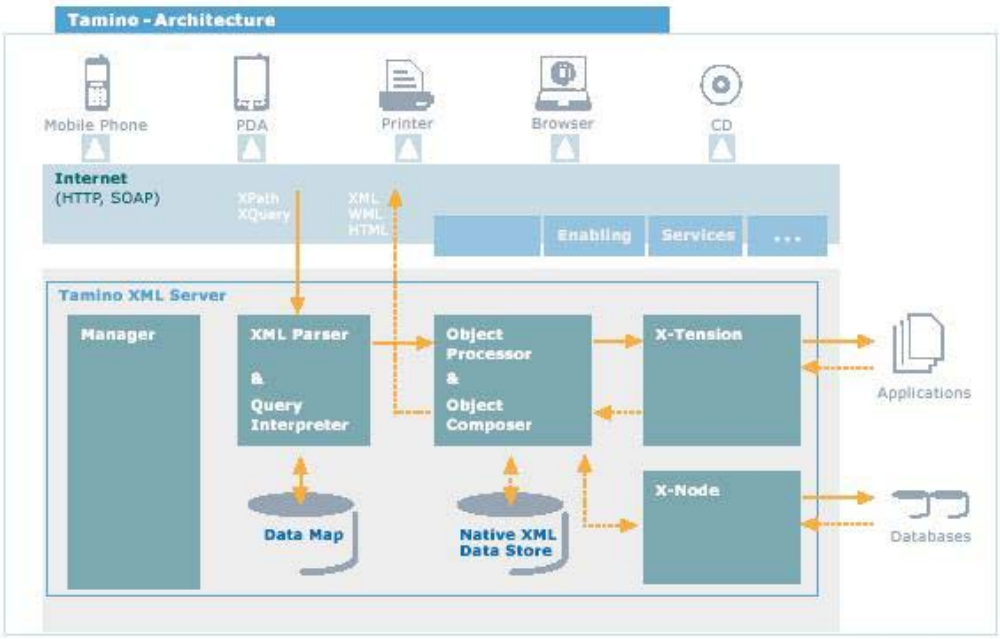
Obrázek 10. – výsledek dotazu XQuery přes webové rozhraní



3.14. Nativní databázový systém Tamino

Developer: Software AG
URL: http://www.softwareag.com/tamino/architecture.htm
Licence: Komerční
Typ databáze: Vlastní model. Relační přes ODBC

Obrázek 11. – architektura Tamina



Základem Tamina je databázové jádro umožňující správu dat v XML formátu a na toto jádro je navázáno aplikační okolí, které se skládá s různých modulů. Mezi nejdůležitější moduly patří například webový server, administrátorská aplikace, moduly zajišťující bezpečnost, nebo rozhraní pro přístup k externím zdrojům (X–Node). Tamino dále obsahuje službu X–Tension, která uživatelům umožňuje programovat rozšířenou funkcionalitu serveru. Pomocí aplikačního serveru můžeme přistupovat jak k lokální XML databázi, tak k Tamino XML databázi na vzdáleném počítači. Administrátorská aplikace pro webové prostředí se jmenuje Tamino Manager a jejím prostřednictvím je možné spravovat databázi včetně jejího zálohování a obnovy, spravovat uživatelské účty, sledovat probíhající procesy s možností jejich řízení atd. Celkem zajímavou vlastností je možnost vytvářet a využívat databázi na CD. Jako primární datové úložiště slouží XML databáze. Tamino navíc obsahuje SQL engine a přes rozhraní ODBC, JDBC a OLE/DB je možné s použitím integračního modulu X–Node přistupovat k datům uloženým na

jiných databázových serverech. K práci s dokumenty využívá XML engine tzv. Data Map, která popisuje, kde jsou data v daném XML dokumentu uložena. Tato vlastnost umožňuje XML dokumentům skládat se z více různorodých zdrojů, jako je například nativní XML databáze, relační databáze a souborový systém. Tamino také podporuje další nástroje pro práci s XML jako DOM, JDOM, SAX, XML:DB API.

3.15. Nativní databáze Xindice

Developer: Apache Software Foundation

URL: http://xml.apache.org/xindice/

Licence: Open Source

Typ databáze: Vlastní model. (Model–based)

Databázový server Xindice [zýndýče] byl od základu vytvořen speciálně pro ukládání XML dat. Xindice se vyvíjí jako Open Source pod Apache Software Foundation a všechny zdrojové kódy jsou dostupné na www stránkách projektu http://xml.apache.org/xindice. Nativní XML databáze Xindice je napsána v jazyce Java a je určena pro ukládání velkého počtu malých XML dokumentů. Xindice nejdříve indexuje elementy a atributy a poté uloží zkomprimovaný dokument. Dokumenty jsou hierarchicky uspořádané v kolekcích (uspořádání je podobné souborovému systému). Jako dotazovací jazyk využívá Xindice jazyk XPath. Pro aktualizaci Xindice podporuje jazyk XUpdate od XML:DB Initiative. Xindice přichází s experimentálním jazykem pro odkazování částí dokumentů, který je podobném principu jako XLink a umožňuje rychlé přesouvání a vkládání obsahu XML dokumentu. Xindice poskytuje implementaci XML:DB API (pro aplikace napsané v jazyce Java), CORBA API (pro aplikace v jiných jazycích) a XML–RPC plugin (pro přístup k databázi jazyky PHP, Perl, Appletscript). Dále Xindice poskytuje XML objekty, které uživatelům umožňují rozšířit funkcionalitu serveru. Přístup k serveru Xindice lze provádět buď programově přes API nebo z příkazové řádky použitím přiložených nástrojů.

Správa dokumentů v Xindice

Příklad 54. Přidání s dokumentu faktury_45.xml do kolekce /podnik/zakazky s určeným klíčem fa_45

```
xindice add_document -c /podnik/zakazky -f faktury_45.xml -n fa_45
```

Příklad 55. – tentýž příklad jako 54. bez zadání hodnoty klíče (pokud není zadána hodnota klíče, server automaticky klíč vygeneruje)

```
xindice add_document -c /podnik/zakazky -f faktury_45.xml
```

Příklad 56. získání dokumentu faktury_45.xml z databáze

```
xindice retrieve_document -c /podnika/zakazky -f faktury_45.xml -n
fa_45
```

Příklad 57. – odstranění dokumentu faktury_45 z databáze

```
xindice delete_document -c /podnika/zakazky -f
  faktury_45.xml -n fa_45
```

3.16. Ostatní nativní XML databáze

4Suite, 4Suite Server

Developer: FourThought

URL: http://4suite.org/index.xhtml

Licence: Open Source

Typ databáze: Objektově orientovaná

Berkeley DB XML

Developer: Sleepycat Software

URL: http://www.sleepycat.com/products/xml.shtml

Licence: Open Source

Typ databáze: Klíč–hodnota

Birdstep RDM XML

Developer: Birdstep

URL: http://www.birdstep.com/database_technology/rdm_xml.php3

Licence: Komerční

Typ databáze: Objektově orientovaná

Centor Interaction Server

Developer: Centor Software Corp.

URL: http://www.centor.com/solutions/technology.shtml

Licence: Komerční

Typ databáze: Vlastní model

DBDOM

Developer: K. Ari Krupnikov

URL: http://dbdom.sourceforge.net/

Licence: Open Source

Typ databáze: Relační

dbXML

Developer: dbXML Group

URL: http://www.dbxml.com/product.html

Licence: Open Source

Typ databáze: Vlastní model

DOMSafeXML

Developer: Ellipsis

URL: http://www.ellipsis.nl/content/products.htm

Licence: Commercial

Typ databáze: Souborový systém

eXtc

Developer: M/Gateway Developments Ltd.

URL: http://www.mgateway.tzo.com/eXtc.htm

Licence: Komerční

Typ databáze: Post–relační (Cache)

GoXML DB

Developer: XML Global

URL: http://www.xmlglobal.com/prod/db/index.jsp

Licence: Komerční

Typ databáze: Vlastní model

Ipedo

Developer: Ipedo

URL: http://www.ipedo.com/html/products.html

Licence: Komerční

Typ databáze: Vlastní model

Lore

Developer: Stanford University

URL: http://www–db.stanford.edu/lore/home/index.html

Licence: Výzkum

Typ databáze: Semi–strukturovaná

Mark Logic Content Interaction

Developer: Mark Logic Corp.

URL: http://www.marklogic.com/prod.html

Licence: Komerční

Typ databáze: Vlastní model

myXMLDB

Developer: Mladen Adamovic

URL: http://sourceforge.net/projects/myxmldb/

Licence: Open Source

Typ databáze: MySQL

Natix

Developer: data ex machina

URL: http://www.dataexmachina.de/natix.html

Licence: Komerční

Typ databáze: Vlastní model

NaX Base

Developer: Naxoft

URL: http://www.naxoft.com/produit–presentation.html

Licence: Komerční

Typ databáze: Vlastní model

Neocore XMS

Developer: Xpriori

URL: http://www.xpriori.com/index.html

Licence: Komerční

Typ databáze: Vlastní model

Ozone

Developer: ozone–db.org

URL: http://ozone–db.org/frames/home/what.html

Licence: Open Source

Typ databáze: Objektově orientovaná

Sedna XML DBMS

Developer: Management Of Data & Information Systems, Institute for System Programming of the Russian Academy of Sciences

URL: http://www.modis.ispras.ru/Development/sedna.htm

Licence: Volná

Typ databáze: Vlastní model

Sekaiju

Developer: Media Fusion

URL: http://www.mediafusion.co.jp/usa/seihin/sekaiju/index.html

Licence: Komerční

Typ databáze: Vlastní model

SQL/XML–IMDB

Developer: QuiLogic

URL: http://www.quilogic.cc/

Licence: Komerční

Typ databáze: Vlastní XML úložiště plus relační úložiště

Sonic XML Server (formerly eXcelon)

Developer: Sonic Software (who bought eXcelon Corp.)

URL:

http://www.sonicsoftware.com/products/sonic_xml_server/index.ssp

Licence: Komerční

Typ databáze: Objektově orientovaná

TeraText DBS

Developer: TeraText Solutions

URL:

http://www.teratext.com/get/page/browser/browser?category=Products/TeraText%20DBS

Licence: Komerční

Typ databáze: Vlastní model

TEXTML Server

Developer: IXIA, Inc.

URL: http://www.ixiasoft.com/default.asp?xml=/xmldocs/webpages/textml–server.xml

Licence: Komerční

Typ databáze: Vlastní model

TigerLogic XML Data Management Server (XDMS)

Developer: Raining Data

URL: http://www.rainingdata.com/products/tl/abouttl.html

Licence: Komerční

Typ databáze: Vlastní model

TOTAL XML

Developer: Cincom

URL: http://tiger.cincom.com/pages/aboutTotalXML.html

Licence: Komerční

Typ databáze: Objektově–relační, relační přes ODBC

Virtuoso

Developer: OpenLink Software

URL: http://www.openlinksw.com/virtuoso/

Licence: Komerční

Typ databáze: Vlastní model. Relační přes ODBC

XDBM

Developer: Matthew Parry, Paul Sokolovsky

URL: http://sourceforge.net/projects/xdbm/

Licence: Open Source

Typ databáze: Vlastní model

XDB

Developer: ZVON.org

URL: http://zvon.org/index.php?nav_id=61

Licence: Open Source

Typ databáze: Relační

X–Hive/DB

Developer: X–Hive Corporation

URL: http://www.x–hive.com/products/

Licence: Komerční

Typ databáze: Vlastní model. Relační přes JDBC

XML Transactional DOM

Developer: Ontonet

URL: http://ontonet.com/XML_Product.html

Licence: Komerční

Typ databáze: Objektově orientovaná

XpSQL

Developer: Makoto Yui

URL: http://gborg.postgresql.org/project/xpsql/projdisplay.php

Licence: Open Source

Typ databáze: Relační

XStreamDB Native XML Database

Developer: Bluestream Database Software Corp.

URL: http://www.bluestream.com/xstreamdb

Licence: Komerční

Typ databáze: Vlastní model

Xyleme Zone Server

Developer: Xyleme SA

URL: http://www.xyleme.com/en/rubriques/Products/Architecture

Licence: Komerční

Typ databáze: Vlastní model

4. Databázové systémy s podporou XML

Databázové systémy s podporu XML jsou „klasické“ relačně–objektové databáze, ve kterým je použita technologie pro podporu formátu XML nebo takové databáze, které jsou s pomocí middleware nástrojů schopny s XML formátem pracovat.

4.1. Principy ukládání dat

V databázových systémech s podporou XML se pro ukládání XML dat využívají hlavně tři následující přístupy.

4.1.1. Nestrukturované ukládání

Celý XML dokument se ukládá v jedné komponentě řádku tabulky. Nestrukturované ukládání se využívá hlavně tam, kde potřebujeme z databáze rekonstruovat originál XML dokumentu (musí být zachováno pořadí elementů, zachovány mezery atd.). Tato vlastnost je důležitá například u lékařských zpráv nebo právnických dokumentů. Velkou nevýhodou tohoto přístupu je aktualizace uložených dat. Často se musí zrekonstruovat celý dokument, provést na něm změny a znovu ho uložit.

4.1.2. Rozkládání XML dokumentu

XML dokument je dekomponován do řádků a sloupců více tabulek. Tato dekompozice je buď definována uživatelem nebo je prováděna implicitně. Rozkládání XML dokumentu se využívá hlavně pro data, která jsou často aktualizována. Aktualizace dat v tomto případě je velmi rychlá. Zpětná rekonstrukce uložených dat tu není základním požadavkem.

4.1.3. strukturované ukládání

V tomto případě je zachována hierarchie XML dat. Tento přístup je pro práci s XML daty nejpokročilejší a dodavatelé relačních databází na vývoji tohoto přístupu usilovně pracují. Způsob zajištění strukturovaného ukládání XML dat je různý. V některých případech se ukládá XML dokument spolu s tabulkami, které popisují jeho strukturu (nad těmito tabulkami může být vytvořen ještě index, např. B–strom).

4.1.4. Převod XML dat do tabulky

Příklad 58. – ukládání XML dokumentu do tabulky

XML data

```
<zakaznici>
   <zakaznik>
      <id>1</id>
      <jmeno>Alena</jmeno>
      <prijmeni>Krátká</prijmeni>
   </zakaznik>
   <zakaznik>
      <id>2</id>
      <jmeno>Irena</jmeno>
      <prijmeni>Sobotková</prijmeni>
   </zakaznik>
</zakaznici>
```

Tabulka 5. – v tabulce s názvem zakaznici by dokument vypadal následovně

id	jmeno	prijmeni
1	Alena	Irena
2	Krátká	Sobotková

4.1.5. Převod XML dat do objektu

Příklad 59. – mapování XML do objektu

XML data

```
<zakaznik>
   <id>1</id>
   <jmeno>Alena</jmeno>
   <prijmeni>Krátká</prijmeni>
</zakaznik>
```

Objekt

```
object zakaznik {
   id=„1“
   jmeno=„Alena“
   prijmeni=„Krátká“
}
```

4.2. Oracle Databaze 10g

Oracle Databáze 10g využívá pro práci s XML daty vlastní technologii s názvem Oracle XML DB. I předchozí verzi s názvem Oracle Databaze 9i má zabudovanou podporu XML a mezi verzemi 9i a 10g není v tomto směru téměř žádný rozdíl.

4.2.1. Hlavní vlastnosti Oracle XML DB:

- nativní XML typ *xmltype*
- použití XML schémat
- provádění XML operací nad relačními daty
- provádění SQL operací nad XML daty
- podpora dotazovacích jazyků XPath, XQuery a SQL/XML
- přístup k databázi přes HTTP, FTP a WebDav

4.2.2. Ukládání xmltype obsahu

Při ukládání dat typu xmltype můžeme určit jestli chceme, aby se data uložila strukturovaně nebo nestrukturovaně. Pokud budeme ukládat data nestrukturovaně, tak se xmltype vnitřně uloží do datového typu CLOB. Při práci s takto uloženým dokumentem se vždy musí načíst/uložit celý jeho obsah. Zvolíme–li pro ukládání strukturovaný xmltype, tak se tento xmltype vnitřně mapuje do tabulek. Pro strukturované ukládání je třeba XML schéma ukládaného dokumentu. Při práci s takto uloženým dokumentem se nenačítá celý jeho obsah, ale operace jsou prováděny pouze nad určenou množinou dat. Při strukturovaném ukládání nemusíme z databáze dostat zpět stejný dokument, obsah vráceného dokumentu se může lišit v bílých znacích, ale struktura a pořadí elementů se nezmění.

4.3. Srovnání vybraných databází

Tabulka 6. – Porovnání vybraných databázových systémů

	MS SQL 2000	**IBM DB2**	**Oracle 10g**	**Sysbase ASE 12.5**	**IBM Informix 9**
typ DBS	relační	relační	objektově–relační	relační	relační
název pro XML rozšíření	SQLXML	XML Extender	XML DB		Web DataBlade
XML pohled na relační data	ano	ano	ano	ano	ano
Relační pohled na XML data	ano	ano	ano	ano	ne
Možnost uložení XML do speciálního XML datového typu	ne	ano	ano	ano	ne

4.4. Ostatní databázové systémy s podporou XML

Access 2002

Developer: Microsoft

URL: http://msdn.microsoft.com/library/en–us/dnacc2k2/html/odc_acxmllnk.asp

Licence: Komerční

Typ databáze: Relační

Cache

Developer: InterSystems Corp.

URL: http://www.intersystems.com/cache/technology/components/xml/index.html

Licence: Komerční

Typ databáze: Post–relační

DB2

Developer: IBM

URL: http://www–3.ibm.com/software/data/db2/extenders/xmlext/

Licence: Komerční

Typ databáze: Relační

eXtremeDB

Developer: McObject LLC

URL: http://www.mcobject.com/extremedb.shtml

http://www.mcobject.com/soap/

Licence: Komerční

Typ databáze: Navigační

FileMaker

Developer: FileMaker

URL: http://www.filemaker.com/xml/overview.html

Licence: Komerční

Typ databáze: FileMaker

FoxPro

Developer: Microsoft

URL: http://msdn.microsoft.com/library/default.asp?url=/library/en–us/fox7help/html/lngrfxmlfunctions.asp

Licence: Komerční

Typ databáze: Relační

Informix

Developer: Informix (nyní vlastněný IBM)

URL: http://www.informix.com/idn–secure/webtools/ot/, http://www–4.ibm.com/software/data/informix/blades/web/

Licence: Komerční

Typ databáze: Relační

Matisse

Developer: Matisse Software

URL: http://www.matisse.com/product_information/

Licence: Komerční

Typ databáze: Objektově–orientovaná

Oracle 8i, Oracle 9i XDB

Developer: Oracle

URL: http://otn.oracle.com/tech/xml/xmldb/content.html

http://otn.oracle.com/tech/xml/xmldb/pdf/xmldb_92twp.pdf

Licence: Komerční

Typ databáze: Relační

SQL Server 2000

Developer: Microsoft

URL: http://msdn.microsoft.com/library/periodic/period00/sql2000.htm (see „XML Support“ section), http://msdn.microsoft.com/library/periodic/period00/thexmlfiles.htm

Licence: Komerční

Typ databáze: Relační

Sybase ASE 12.5

Developer: Sybase

URL: http://my.sybase.com/content/1013051/3929XMLwpaperv9.pdf http://manuals.sybase.com:80/onlinebooks/group–as/asg1250e/jcs_kona/@Generic__BookTextView/12930

Licence: Komerční

Typ databáze: Relační

UniData

Developer: IBM

URL: http://www–306.ibm.com/software/data/u2/unidata/

Licence: Komerční

Typ databáze: Nested relational

UniVerse

Developer: IBM

URL: http://www–306.ibm.com/software/data/u2/universe/

Licence: Komerční

Typ databáze: Nested relational

Versant enJin

Developer: Versant Corp.

URL: http://www.versant.com/products/enjin/datasheet.html

Licence: Komerční

Typ databáze: Objektově orientovaná

5. Závěr

Ve své bakalářské práci jsem se zabýval hlavně jazykem XQuery a nativními XML databázemi, ale pro lepší orientaci v dané problematatice bylo nutné uvést i související technologie. V úvodu jsem tyto technologie ve zkratce popsal a vysvětlil jejich použití. V kapitole 2. Dotazovací jazyky jsem se nezabýval pouze jazykem XQuery, ale i jazyky XPath, XSLT, XUpdate, aby si čtenář mohl udělat vlastní obrázek o vývoji v této oblasti. Tato kapitola není učebnicí dotazovacíh jazyků a ani to tak nebylo zamýšleno, spíše jsem se snažil ukázat možnosti jednotlivých jazyků a uvést příklady, ve kterých je možné tytot jazyky použít. V kapitole 3. Nativní XML databázové systémy jsem se zabýval problematikou zpracování XML dat v těchto poměrně nových systémech pro jejich správu a myslím, že jsem jejich základní možnosti, klady i zápory dobře popsal. V této kapitole jsem stručně popsal i tři konkrétní nativní XML databázové systémy. Pokud by se jimi chtěl někdo více zabývat může využít dokumentaci, která je k těmto systémům dostupná na webu. Ve čtvté kapitole jsou stručně popsány možnosti zpracování XML dat v klasických databázových systémech, které stále více zdokonalují podporu nativního formátu XML dat. Jak už jsem napsal, tato bakalářská práce není a ani neměla být učebnicí probíraných technologií, ale je to spíše jejich zkrácený popis, který slouží k snadnému a rychlému zorientování v této problematice a to bylo mým cílem.

Součástí této bakalářské práce je CD s několika vzorovými příklady použití dotazovacích jazyků pro XML

Použité zdroje

Literatura v elektronické podobě dostupná na internetu:

[1] Pitner Tomáš – XML a databáze,
http://www.fi.muni.cz/~tomp/slides/pb138/6databases/slides/

[2] Pokorný Jaroslav – XML databáze,
http://kocour.ms.mff.cuni.cz/~pokorny/vyuka/xml-dj/ppframe.htm

[3] Pokorný Jaroslav – XML databáze: Současný stav a perspektivy
http://kocour.ms.mff.cuni.cz/~pokorny/papers/DATAKON04-ready.pdf

[4] Robie Jonathan – XQuery: A Guided Tour
http://www.datadirect.com/developer/xquery/xquerybook/index.ssp

[5] Vanický Miroslav – XML a databáze,
http://ploppy.vande.cz/dp/

[6] Vaškovič Pavol – Natívné XML databázy
http://pali.sk/NXD.doc

[7] Vávra Jan, Přemysl Volf – XQuery
http://xien.jikos.cz/document/xquery.ppt

Internetové zdroje:

[8] http://badame.vse.cz

[9] http://exist.sourceforge.net

[10] http://interval.cz/

[11] http://www.kosek.cz

[12] http://www.microsoft.com

[13] http://www.oracle.com

[14] http://www.root.cz

[15] http://www.rpbourret.com

[16] http://www1.softwareag.com/corporate/products/tamino

[17] http://www.w3c.org

[18] http://www.w3schools.com

[19] http://www.xml.com

[20] http://xml.apache.org/xind

Dotazovací jazyky pro formát XML a nativní XML databáze