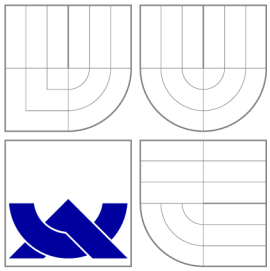
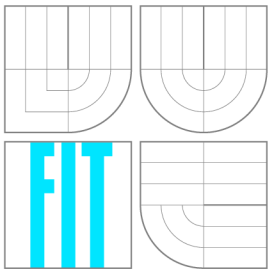


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

TRANSFORMACE A PERZISTENCE XML V RELAČNÍ DATABÁZI

XML TRANSFORMATION AND PERSISTENCE IN RELATIONAL DATABASE

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. RADIM HERNYCH

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. PETR CHMELAŘ

BRNO 2009

Zadání práce

1. Seznamte se s problematikou transformace a perzistence XML, jeho nativními i rozšířenými XML databázemi. Zorientujte se také v problematice perzistence schémat XML. Zaměřte se na standard SQL 2008.
2. Vytvořte přehled současných řešení XML databází, diskutujte jejich výhody a nedostatky.
3. Identifikujte a pokuste se teoreticky vyřešit problémy efektivního uchovávání a dotazování XML dat.
4. Vytvořte systém pro uchování, dotazování a transformaci XML dokumentů a schémat prostřednictvím běžné objektově relační databáze.
5. Experimentálně ověřte funkčnost na reálných datech.
6. Zhodnoťte vlastnosti vytvořeného řešení s existujícími a případné vylepšení systému.

Licenční smlouva

Licenční smlouva je uložena v archívu Fakulty informačních technologií Vysokého učení technického v Brně.

Abstrakt

Tato diplomová práce se zabývá problematikou efektivního ukládání a dotazování XML dokumentů v relační databázi. Nejprve jsou rozebrány technologie spojené s jazykem XML, zejména jazyky pro definici schématu XML dokumentů a jeho dotazování. Dále je nastíněn návrh metody mapování XML dokumentů, založené na XML schématu. V dalších částech je popsán návrh a implementace programové vrstvy nad běžnou objektově relační databází, která v ní umožní ukládat a dotazovat XML dokumenty. Poslední část práce pak obsahuje výsledky a zhodnocení výkonových testů, kterým byl vytvořený systém podroben.

Abstract

This master's thesis deals with problems of effective storage and querying of XML documents in A relational database. In the first part of this thesis XML and related technologies are described, with emphasis on languages for XML Schema definition and it's querying. Then an XML mapping method based on Hybrid method and XML Schema is described here. Hereafter the design and implementation of a native XML database front-end for object-relational database is described. Designed front-end allows storing and querying of XML documents into the underlying database. The last part contains results and evaluation of performance tests of the implemented system.

Klíčová slova

XML, XML dokument, XML Schema, nativní XML databáze, databáze s podporou XML, XQuery, XPath

Keywords

XML, XML dokument, XML Schema, native XML datase, XML-enabled database, XQuery, XPath

Citace

Radim Hernych: Transformace a perzistence XML v relační databázi, diplomová práce, Brno, FIT VUT v Brně, 2009

Transformace a perzistence XML v relační databázi

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Petra Chmelaře. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Radim Hernych
26. května 2009

Poděkování

Chtěl bych poděkovat vedoucímu své diplomové práce Ing. Petru Chmelařovi za jeho cenné rady a odbornou pomoc při řešení práce. Dále také své rodině za podporu při studiu.

Obsah

Kapitola 1

Úvod

Informace a jejich efektivní zpracování mají v dnešní době strategický význam. Pro firmy je velmi důležitá výměna informací s okolím. Dříve se pro výměnu dat používaly proprietární formáty, které však byly uzavřené a výměna dat mezi informačními systémy tak byla vskutku nákladnou záležitostí.

S nástupem technologie XML se však toto změnilo a v dnešní době představuje XML již de facto standard pro elektronickou výměnu informací a dat. Narůstající počet vytvářených, přenesených a prezentovaných XML dokumentů vede k potřebě umět tyto dokumenty efektivně ukládat a spravovat. Trendem se proto stalo ukládání XML dokumentů do databází, což vedlo výrobce databázových systémů k rozšíření stávajících systémů o prostředky pro práci s XML. Vznikl také zcela nový typ databáze – nativní XML databáze.

Poměr využití nativních XML databází ke všem využívaným databázím je však stále velmi malý. Většinu databází tvoří relační nebo objektově relační databáze. Ukládání XML do relačních databází však není jednoduchá záležitost, protože oba datové modely (XML a relační model) jsou postaveny na zcela odlišných principech. Zde přichází na řadu možnost využití mezivrstvy pro přenos dat mezi XML dokumenty a relační databází.

Další výzvou je rychlost databází pracujících s XML. Ze srovnávacích výkonových testů uvedených v knize [15] vyplývá, že nativní XML databáze dosahují v některých případech velmi špatných výsledků v porovnání s relačními nebo objektově orientovanými databázemi. Nativní XML databáze také spotřebovávají nejvíce místa na disku. V některých aplikacích proto může stále být výhodnější použít vhodného mapování XML dokumentů na schéma relační databáze.

Tato diplomová práce navazuje na semestrální projekt, jehož cílem bylo seznámení s problematikou XML a teoretické rozpracování tématu perzistence a transformace XML v objektově relační databázi. Ze semestrálního projektu byly převzaty a dále dopracovány kapitoly 2, 3 a 4.

1.1 Cíl práce

Tato diplomová práce si klade za cíl navrhnout a vytvořit softwarovou vrstvu nad běžnou objektově relační databází, která by umožnila efektivní ukládání a dotazování datově orientovaných XML dokumentů.

1.2 Struktura práce

Kapitola 1 obsahuje stručné uvedení do problematiky, cíle a členění práce.

V kapitole 2 je popsán jazyk XML a technologie okolo něj (možnosti definice XML schématu, transformace XML dokumentů, dotazovací jazyky).

Kapitola 3 se zaměřuje na současné možnosti uložení XML dokumentů do moderních databází s důrazem na představení nativních XML databází.

Kapitola 4 se zabývá uložením XML dokumentů do relační databáze a zkoumá teoretické metody pro mapování XML schématu na databázové schéma. Identifikuje také možné problémy jednotlivých metod a navrhuje řešení.

Následující kapitola 5 popisuje zvolenou, schématem řízenou, metodu mapování XML na schéma relační databáze. Postupně jsou zde popsány metody pro mapování XML schématu na relační schéma, uložení XML dokumentu do vytvořeného databázového schématu, rekonstrukce uloženého XML dokumentu a dotazování pomocí XPath.

Předposlední kapitola 6 se zabývá stručným popisem návrhu a implementace softwarové vrstvy nad objektově relační databází PostgreSQL.

V kapitole 7 jsou popsány a vyhodnoceny výkonové testy vytvořeného systému a také srovnání rychlostí dotazování s již existující XML databází eXist.

Závěrečná kapitola 8 pak shrnuje výsledky práce a nastiňuje další možný vývoj systému.

V přílohách následuje schéma databáze systému pro uložení metainformací o kolekcích B, tutoriál aplikačního rozhraní vytvořeného systému C a ukázky mapování zvoleného vzorového XML dokumentu D.

Kapitola 2

Značkovací jazyk XML

Následující kapitola se zabývá jazykem XML (*eXtensible Markup Language*), strukturou dokumentů psaných v tomto jazyce a možnostmi jejich transformace. V další části kapitoly jsou představeny jazyky pro definici XML schématu a v poslední části jsou rozebrány možnosti dotazování XML dokumentů.

2.1 Charakteristika XML

Jazyk XML [19] je jednoduchý, velmi flexibilní textový formát pro publikování a výměnu dokumentů. Přinesl významný pokrok do oblasti výměny dokumentů a možnosti zpracování informace.

Jazyk XML vznikl (stejně jako např. HTML) jako zjednodušená podmnožina komplexnějšího značkovacího jazyka SGML (*Standard Generalised Markup Language*). V roce 1998 byl standardizován konsorciem W3C (*World Wide Web Consortium*). Původně byl XML navržen pro ukládání strukturovaného a semistrukturovaného textu určeného pro šíření v celé řadě médií, dnes však hraje velmi důležitou úlohu při výměně dat.

XML je značkovací jazyk. Umožňuje popsat strukturu z hlediska významu jednotlivých částí textu. Na rozdíl od jazyka HTML však nedefinuje konečné zobrazení dokumentu. Vzhled dokumentu je pak definován buď připojeným stylem, nebo se provede transformace na jiný formát (např. HTML či jiný dokument XML).

Již při návrhu jazyka XML se počítalo i s použitím jiných jazyků než angličtiny, a proto je výchozí znakovou sadou Unicode. Unicode je tabulka znaků všech existujících abeced, což umožňuje vytvářet dokumenty obsahující texty ve více jazycích najednou. Je samozřejmě možné použít i jiné kódování než Unicode.

Ve své podstatě je XML metajazyk, což znamená, že je určen k popisu dalších jazyků. XML poskytuje naprostou svobodu v pojmenovávání značek, protože nemá (např. narozdíl od HTML) seznam předdefinovaných elementů. Vývojář má tedy možnost v XML dokumentu použít taková jména elementů a atributů, která mají pro danou aplikaci smysl. Tím lze dosáhnout zvýšení srozumitelnosti dokumentu. Přestože se historické kořeny XML nacházejí v oblasti publikování, jazyk je vhodný i pro popis a identifikaci komplexních datových struktur, které nebudou nikdy publikovány [1].

2.2 Struktura XML dokumentu

Dokument XML má fyzickou a logickou strukturu. Zatímco logická rozděluje dokument na pojmenované jednotky a podjednotky nazývané elementy, fyzická struktura umožňuje vkládat do dokumentů samostatné pojmenované části dokumentu zvané entity. Entity mohou být i v samostatných datových souborech, aby mohly být opakovaně použity v různých dokumentech a také aby bylo možné odkazovat data, která nejsou standardu XML (například obrázky či binární soubory).

Elementy se v textu vyznačují pomocí tzv. značek (tagů). Většina elementů má dvě značky – počáteční a ukončovací. Názvy značek se zapisují mezi lomené závorky (tedy znaky < a >), aby se tak odlišily od zbytku textu. Pokud je v textu potřeba tyto znaky použít, je nutné je nahradit zástupnými textovými entitami < a >. Ukončovací značka má před svým názvem ještě znak /, aby se tak odlišila od počáteční. Ve jménech elementů se rozlišují malá a velká písmena, takže například <nazev>, <Nazev> a <NAZEV> jsou tři různé elementy. Elementy mohou obsahovat další vnořené elementy a tím dle potřeby zachycovat strukturu informací uložených v dokumentu. Celý dokument je vložen do jediného kořenového elementu dokumentu [19].

Každý element může ještě kromě svého jména obsahovat další informace, které blíže upřesňují jeho obsah. Tyto informace se nazývají atributy. Každý atribut musí mít jméno, protože element může mít více atributů. Obsah atributu musí být ohraničen v uvozovkách nebo apostrofech. Jednoduchý záznam o osobě v XML dokumentu může například vypadat následovně:

Zdrojový kód 2.1: Ukázka XML dokumentu

```
<osoba>
   <jmeno>Radim</jmeno>
   <prijmeni>Hernych</prijmeni>
   <kontakty>
     <kontakt typ="tel">777123456</kontakt>
     <kontakt typ="email">mr.nobody@gmail.com</kontakt>
   </kontakty>
</osoba>
```

2.3 XML schémata

Schémata jsou dokumenty popisující určitý typ XML dokumentu. Typicky definují povolenou strukturu a obsah XML dokumentů. Existuje více jazyků pro popis XML schémat, nejpoužívanějšími jsou DTD, W3C XML Schema a Relax NG.

Velkou výhodou XML je podpora tzv. jmenných prostorů (namespaces). Díky nim je možné používat v jednom XML dokumentu značkování z více než jednoho schématu.

Schémata lze s výhodou použít k programové kontrole dokumentu – validaci. Validující parser (validátor) ověřuje shodu dokumentu se schématem. To umožňuje odhalit možné nekonzistence v datech ještě před zpracováním dokumentu.

2.3.1 DTD

DTD (*Document Type Definition*) je nejstarším schématem, které XML zdědilo ještě po SGML. DTD je mimo XML a SGML možno použít i k definici dokumentů HTML. Definuje strukturu dokumentu seznamem legálních elementů a atributů a popisuje také, jak mohou

být značky navzájem uspořádány a vnořeny. Dále dovoluje pro každou značku vymezit povolené atributy a jejich typ. Následuje příklad jednoduchého DTD:

Zdrojový kód 2.2: Ukázka jednoduchého DTD

```
<!ELEMENT adresar (osoba*)>
<!ELEMENT osoba (jmeno, adresa?, email+, tel*, icq?)>
<!ELEMENT adresa (mesto, ulice, psc)>
<!ELEMENT jmeno (#PCDATA)>
<!ELEMENT email (#PCDATA)>
<!ELEMENT tel (#PCDATA)>
<!ELEMENT icq (#PCDATA)>
```

Jednotlivé řádky z předchozí ukázky DTD říkají asi následující:

- *adresar* je validním jménem elementu a může obsahovat libovolné množství elementů *osoba*. Libovolný počet je indikován znakem *.
- *osoba* je validním jménem elementu a musí obsahovat element *jmeno*, volitelně element adresa, alespoň jeden element *email*, libovolné množství elementů *tel* a nakonec volitelně element *icq*. Znak ? znamená, že se element může vyskytovat maximálně jedenkrát. Znak + naopak říká, že se element může vyskytovat minimálně jedenkrát.
- *adresa* je validním jménem elementu a musí obsahovat elementy *mesto*, *ulice* a *psc*.
- *jmeno* je validním jménem elementu a může obsahovat text.
- *email* je validním jménem elementu a může obsahovat text.
- *tel* je validním jménem elementu a může obsahovat text.
- *icq* je validním jménem elementu a může obsahovat text.

Příkladem XML dokumentu, který využívá předchozí DTD může být:

Zdrojový kód 2.3: Ukázka XML dokumentu

```
<?xml version="1.0"?>
<adresar>
  <osoba>
    <jmeno>Jan Novák</jmeno>
    <adresa>
      <mesto>Brno</mesto>
      <ulice>Česká</ulice>
      <psc>60200</psc>
    </adresa>
    <email>novak@google.com</email>
    <email>jan.novak@seznam.cz</email>
    <tel>777123456</tel>
  </osoba>
</adresar>
```

Ačkoli se DTD stále používá, není nejvhodnějším jazykem pro popis XML. Asi nejvíce viditelný nedostatek DTD je absence určení typu dat obsahu elementu. Pomocí DTD např. nelze definovat, že element CENA musí obsahovat číslo, které je maximálně čtyřciferné a je větší než nula. Rovněž nelze specifikovat, že element MESIC může obsahovat jen celá čísla od jedné do dvanácti. Nic z toho není příliš důležité u definice jazyka HTML nebo SGML.

Protože se však XML mimo jiné používá i pro počítačové zpracování dat, vyvstala potřeba lepší definice typu dat.

Dalším neméně významným problémem DTD je, že sám o sobě není XML dokumentem. Programy, které čtou a zpracovávají XML, tak nemohou být použity pro zpracování DTD.

Posledním problémem je, že DTD je poměrně špatně rozšiřitelné. Není jednoduché elegantně zkombinovat více nezávislých DTD dohromady. DTD také není příliš kompatibilní se jmennými prostory [22].

Novější schémata XSD a RNG jsou pokusem o vyřešení všech těchto problémů definováním syntaxe založené na XML a detailnější definicí obsahu XML dokumentu včetně rozšířených možností typové kontroly dat.

2.3.2 XML Schema

Jazyk XML Schema (někdy se také uvádí XSD – *XML Schema Definition*) je novější alternativou k DTD a byl (stejně jako XML) standardizován konsorciem W3C. Podobně jako DTD slouží k definování přípustné struktury daného XML dokumentu, v mnohém však možnosti DTD rozšiřuje.

XML Schéma (převzato z [3]):

- definuje elementy, které se mohou v dokumentu vyskytovat
- definuje elementům přípustné atributy
- definuje, které elementy jsou potomky jiných elementů
- definuje pořadí elementů a počty elementů
- definuje, zda element může být prázdný, nebo zda může obsahovat text
- definuje datové typy elementů a jejich atributů
- definuje standardní hodnoty elementů a atributů

Pokud přepíšeme výše uvedený příklad definice DTD do XSD, výsledek by mohl vypadat následovně:

Zdrojový kód 2.4: Ukázka definice XML dokumentu pomocí XML Schema

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <xsd:element name="adresar" type="TAdresar"/>

  <xsd:complexType name="TAdresar">
     <xsd:sequence>
        <xsd:element name="osoba" type="TOsoba" minOccurs="0" maxOccurs="unbounded" />
     </xsd:sequence>
  </xsd:complexType>

  <xsd:simpleType name="TTelefon">
     <xsd:restriction base="xsd:string">
        <xsd:pattern value="[0-9]{9}" />
     </xsd:restriction>
  </xsd:simpleType>
```

```
  <xsd:complexType name="TOsoba">
     <xsd:sequence>
        <xsd:element name="jmeno" type="xsd:string" />
        <xsd:element name="adresa">
           <xsd:complexType>
              <xsd:all>
                 <xsd:element name="mesto" type="xsd:string" />
                 <xsd:element name="ulice" type="xsd:string" />
                 <xsd:element name="psc" type="xsd:string" />
              </xsd:all>
           </xsd:complexType>
        </xsd:element>
        <xsd:element name="email" type="xsd:string" maxOccurs="unbounded" />
        <xsd:element name="tel" type="TTelefon" minOccurs="0" maxOccurs="unbounded" />
        <xsd:element name="icq" type="xsd:positiveInteger" minOccurs="0" />
     </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

2.3.3 Relax NG

Dalším jazykem pro popis struktury XML je Relax NG (čti *relaxing*). Relax NG byl navržen tak, aby bylo pro autory schémat jednoduché se jej naučit a pro vývojáře jednoduché implementovat. Vývoj Relax NG byl veden potřebou vytvořit jednodušší alternativu k poměrně složitému a komplexnímu XML Schema. Návrh jazyka byl inspirován teorií stromových automatů a validátory Relax NG jsou proto typicky implementovány jako stromové automaty. Díky tomuto matematickému modelu, na kterém je Relax NG postaven, jsou hlavními přednostmi jazyka jednoduchost, expresivnost a spolehlivost [32].

Relag NG je založen na konceptu vzorů. Předchozí DTD a XML Schema definují popis a vlastnosti elementu samotného. Naopak definováním elementu v Relax NG je vytvořen vzor, který je pak při validaci srovnáván s elementy ve validovaném dokumentu. Vzor v Relax NG by mohl být přirovnán k regulárnímu výrazu použitému k vyhledávání v textu. Tento přístup poskytuje více flexibility pro psaní, udržování a kombinování schémat.

2.4 XML Transformace

Základní myšlenkou značkovacích jazyků včetně XML je oddělení obsahu dokumentu od jeho vzhledu. Definuje pouze strukturu dokumentu. Umožňuje pojmenování elementů takovými jmény, která popisují podstatu a obsah objektu. Tato informace je samopopisující a lze ji v dokumentu vyhledat, vyjmout a zpracovávat podle potřeby.

Pro definici vzhledu XML dokumentů je nutné použít stylové jazyky, kterých dnes existuje několik. Nejznámějším jazykem je CSS (*Cascading Style Sheets*), který je však vhodný pouze pro jednoduché formátování. Pro náročnější aplikace je určen jazyk XSL.

2.4.1 Co je to XSL?

XSL (*eXtensible Stylesheet Language*) vznikl jako univerzální stylový jazyk. XSL je založeno na XML, takže pro zpracování stylu je možné použít jakékoli nástroje, které umějí pracovat s XML.

Samotné XSL je rozděleno na dvě části. První částí je transformační jazyk XSLT, který lze použít pro transformaci vstupního XML dokumentu do jiného XML dokumentu s odlišnou strukturou, HTML dokumentu, nebo případně do textového formátu. Výsledný transformovaný dokument může využívat stejné schéma jako původní dokument, nebo může jít o dokument zcela jiného schématu. Druhou částí XSL jsou tzv. formátovací objekty, které slouží jako abstraktní popis vzhledu stránky [24].

2.4.2 XSL Transformace

První částí XSL je transformační jazyk XSLT (*XSL Transformations*), který lze použít nezávisle na formátovacím jazyku. Jeho schopnost přenést data z jednoho XML formátu do jiného XML formátu jej činí důležitou částí elektronické výměny dat. Mnoho implementací XSL se tak soustředí výhradně jen na tuto transformační část XSL, neboť ne všechna data je nutné formátovat a vizualizovat [22].

Základem XSL transformací je šablona. XSLT procesor načte vstupní XML dokument a XSLT styl a aplikuje instrukce v šabloně na data ze vstupního dokumentu. Každá šablona definuje, na jakou část vstupního dokumentu se aplikuje a jak bude vybraná část transformována do výstupního dokumentu. Definice stylu XSLT má tvar:

Zdrojový kód 2.5: Ukázka definice stylu v XSLT

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <!-- seznam jednotlivých šablon, na pořadí nezáleží -->
  <xsl:template match="výraz v˜jazyce XPath">
    <!-- výstup šablony -->
  </xsl:template>

</xsl:stylesheet>
```

Z předchozího příkladu je zřejmé, že dokument XSLT stylu používá jmenný prostor *xsl*. Každá šablona musí být obsažena v elementu xsl:template. Atribut *match* elementu xsl:template představuje vzor určující element nebo elementy, na které bude šablona aplikována. Obsah elementu xsl:template je výstupem šablony a může obsahovat další značky, nová data, data extrahovaná ze zpracovávaného dokumentu nebo další příkazy pro XSLT procesor. Všechny příkazy pro XSLT procesor musí mít prefix *xsl*, jinak budou zahrnuty do výstupu [22].

Jako příklad jednoduché šablony můžeme uvést šablonu, která převede ze vstupního dokumentu elementy *odstavec* na odstavce v jazyce HTML. Element *xsl:apply-templates* říká XSLT procesoru, aby na obsah vybraného elementu aplikoval další šablony ze stylu. Protože XSLT procesor obsahuje některé jednoduché šablony pro zpracování samotného textového obsahu elementu, vždy se po provedení *xsl:apply-templates* vypíše alespoň textový obsah elementu [24].

Zdrojový kód 2.6: Jednoduchá XSLT šablona

```
<xsl:template match="odstavec">
  <p><xsl:apply-templates /></p>
</xsl:template>
```

XSLT nabízí při tvorbě šablon velké množství možností. V šablonách lze používat podmínky, cykly a proměnné. Dále je v šablonách možné řadit elementy dle různých kritérií či použít automatické číslování elementů.

2.4.3 XSL Formátovací objekty

Druhou částí XSL je jazyk XSL-FO (*XSL Formatting Objects*), který definuje tzv. formátovací objekty. Formátovací objekty slouží k popisu vzhledu dokumentu a jeho jednotlivých komponent a nabízejí sofistikovanější a bohatší možnosti než např. HTML a CSS.

Pomocí formátovacích objektů lze specifikovat rozměry stránky, způsob zarovnání a použitá písma, barvy atd. Při tvorbě stylu se obvykle použije kombinace XSLT a XSL-FO, kdy XSLT styl transformuje vstupí XML dokument na XSL-FO dokument, který je pak procesorem interpretován (více v [22] a [10]).

2.5 Dotazování

Stále více dokumentů a informací je ukládáno, vyměňováno a prezentováno za použití XML. Proto je stále důležitější umět v XML dokumentech inteligentně a rychle vyhledávat.

2.5.1 XPath

Jazyk XPath je určen pro vyhledání a adresaci částí XML dokumentu. Pro tento účel poskytuje také základní prostředky pro manipulaci s textovými řetězci a čísly. Pro usnadnění použití XPath v URL a atributech XML elementů tento jazyk nepoužívá syntaxi XML. XPath pracuje s abstraktní reprezentací struktury XML dokumentu a používá url–podobnou notaci pro navigování skrze hierarchickou strukturu XML dokumentu [2]. Na jazyku XPath jsou založeny další jazyky pro práci s XML, především XSLT, XPointer a XQuery.

XPath modeluje XML dokument jako strom uzlů, přičemž rozlišuje různé typy uzlů (element, atribut, text).

Primární syntaktickou konstrukcí v XPath je výraz. Vyhodnocení výrazu vrací objekt, který může mít jeden z následujících typů:

- množina uzlů (neuspořádaná kolekce uzlů)
- boolean (true nebo false)
- číslo (float)
- textový řetězec

XPath umožňuje pohodlně pracovat s hodnotami elementů a atributů. Výraz v jazyce XPath sestává ze tří částí:

1. identifikátor osy – určuje uzly, které budou v daném kroku zpracovány. Podle výchozí osy jsou to podřízené uzly aktuálního elementu. Je ale možné adresovat i absolutně od kořene dokumentu.
2. testu uzlu – uzly vybrané identifikátorem osy jsou dále testovány, což dovoluje vybrat jen některé uzly na zvolené ose.
3. podmínky – nad vyhovujícími uzly je vyhodnoceno nula nebo více predikátů, pokud je pro daný uzel predikát pravdivý, uzel zůstává v množině zpracovávaných uzlů.

Výrazy

Jazyk XPath používá pro výběr uzlů v XML dokumentu výrazy určující cestu. V tabulce 2.1 je souhrn nejdůležitějších výrazů pro výběr uzlů [4].

Konstrukce	Popis
/	Představuje kořenový uzel dokumentu. Také je použit jako oddělovač jednotlivých kroků XPath výrazu pro výběr podřízených uzlů aktuálního uzlu.
//	Vybere aktuálnímu uzlu všechny podřízené uzly bez ohledu na to, jak jsou zanořeny v toku dokumentu.
.	Vybere aktuální uzel.
..	Vybere rodičovský uzel aktuálního uzlu.
@	Slouží k výběru atributu.
*	Slouží pro výběr všech přímých potomků aktuálního uzlu.
@*	Slouží pro výběr všech atributů aktuálního uzlu.
[ ]	Hranaté závorky obsahují predikáty. Predikát představuje podmínku, kterou musí element nebo atribut splňovat, aby zůstal v množině zpracovávaných uzlů. Slouží také pro určení indexu do seznamu zpracovávaných uzlů.
funkce	XPath poskytuje velké množství vestavěných funkcí jak pro práci s číselnými hodnotami, tak s řetězci.

Tabulka 2.1: Některé důležité konstrukce jazyka XPath

Predikáty

Predikát představuje podmínku, kterou musí element nebo atribut splňovat, aby zůstal v množině zpracovávaných uzlů. Typicky se jedná o testy na specifický element nebo uzel který obsahuje určitou hodnotu. Predikáty jsou uzavřeny do hranatých závorek. Příklady XPath dotazů lze nalézt v tabulce 7.1.

2.5.2 XQuery

Jazyk XQuery byl vytvořen skupinou XML Query Working Group konsorcia W3C a v roce 2007 se stal W3C doporučením. XQuery umožňuje vyhledávat a extrahovat elementy a atributy v XML dokumentech.

XQuery je funkcionální jazyk – místo provádění příkazů, jako v procedurálních jazycích, je jeho základem *výraz*. XQuery vychází z jazyka XPath (stejný datový model) a k vyhledávání a adresaci částí XML dokumentu používá právě výrazy v XPath 2.0. XQuery však není jen dotazovací jazyk, dokáže provádět i zpracování a transformace XML (zde se jazyk inspiroval XSLT) [23].

FLWOR

Dotazy v XQuery často kombinují informace z jednoho nebo více zdrojů dat do jednoho výsledku. FLWOR (čti jako „flower“) výrazy jsou jedny z nejmocnějších nástrojů XQuery,

podobají se `SELECT FROM WHERE` dotazům v SQL. Zkratka FLWOR vznikla z prvních písmen klauzulí, které se mohou ve výrazu objevit [25]:

- for – výběr posloupnosti uzlů k dalšímu zpracování
- let – přiřazení proměnných pro každý prvek posloupnosti
- where – výběr uzlů posloupnosti, které vyhovují zadaným podmínkám
- order by – seřazení vybraných uzlů
- return – sestavení výsledku pro každý vybraný uzel

Na začátku FLWOR výrazu je jedna nebo více klauzulí `for` nebo `let` (v jakémkoli pořadí) následované nepovinnými klauzulemi `where` a `order by` a nakonec povinnou klauzulí `return`.

Příklad jednoduchého XQuery výrazu, který vrátí tituly všech knih, které mají více než dva autory:

Zdrojový kód 2.7: Příklad jednoduchého XQuery výrazu

```
for $b in doc("books.xml")//book
let $c := $b//author
where count($c) > 2
return $b/title
```

Pokud necháme vyhodnotit předchozí XQuery výraz nad následujícím XML dokumentem, XQuery vrátí jako výsledek `<title>Data on the Web</title>`.

Zdrojový kód 2.8: Výstup předchozího XQuery dotazu (převzato z [23])

```
<bib>
    <book year="1994">
        <title>TCP/IP Illustrated</title>
        <author><last>Stevens</last><first>W.</first></author>
        <publisher>Addison-Wesley</publisher>
        <price>65.95</price>
    </book>
    <book year="2000">
        <title>Data on the Web</title>
        <author><last>Abiteboul</last><first>Serge</first></author>
        <author><last>Buneman</last><first>Peter</first></author>
        <author><last>Suciu</last><first>Dan</first></author>
        <publisher>Morgan Kaufmann Publishers</publisher>
        <price>65.95</price>
    </book>
</bib>
```

2.5.3 SQL/XML

Pro databáze s XML rozšířeními, které jsou založené na relační databázi, je nejdůležitějším XML dotazovacím jazykem SQL/XML. Jazyk SQL/XML je množina rozšíření standardu SQL (je součástí standardu ANSI/ISO SQL 2003) umožnující vytvářet XML dokumenty a fragmenty XML dokumentů z relačních dat. Sestává z datového typu XML, kolekce funkcí pro publikování XML, konverzních funkcí, funkcí pro validaci XML dokumentu vzhledem ke schématu a ještě více.

Při srovnání SQL/XML a XQuery je však XQuery mocnějším nástrojem. XQuery je přímočaré, flexibilní a dokáže vytvořit jakoukoli strukturu XML. Navíc se XQuery nestará o to, co je zdrojem dat (může to být jakýkoli XML dokument, webová služba, relační data atd.). Naproti tomu SQL/XML je limitováno pouze v dotazování relačních dat [9].

Například následující volání funkce XMLELEMENT:

Zdrojový kód 2.9: Ukázka dotazu v SQL/XML

```
SELECT
   XMLELEMENT(NAME Customer,
      XMLELEMENT(NAME Name, customers.name),
      XMLELEMENT(NAME ID, customers.id))
FROM customers
```

vytváří element CUSTOMER pro každý řádek v tabulce zákazníků [31]:

Zdrojový kód 2.10: Výstup dotazu SQL/XML

```
<Customer>
    <Name>customer name</Name>
    <ID>customer id</ID>
</Customer>
```

Novější verze SQL/XML ve standardu SQL:2008 přidává několik významných vylepšení. Asi nejdůležitějšími změnami je změna původního datového modelu na datový model XPath 2.0 a XQuery 1.0 a přidání (mimo jiné) nové publikační funkce XMLQUERY(), která umožňuje vyhodnocení XQuery výrazu a vrácení výsledku SQL aplikaci.

Příklad použití funkce XMLQUERY:

Zdrojový kód 2.11: Dotaz SQL/XML spolu s dotazem XQuery

```
SELECT
  XMLQUERY (
    "for $name in /Customer/Name
     where /Customer/ID = $var1
     return $name"
    PASSING BY VALUE "12" AS var1,
      buying_customers
    RETURNING SEQUENCE BY VALUE )
FROM customers
```

2.5.4 Vizualizace

Další možností dotazování XML dokumentů je jejich vizualizace. XML má tendenci modelovat logickou strukturu dat podobně jako používané modelovací techniky UML. Nicméně struktura není nejdůležitějším nositelem informace v XML. Hlavním nositelem informací jsou data obsažená v elementech a atributech. Pokud pomineme smíšený obsah (který je stejně výsadou spíše dokumentově orientovaných XML dokumentů), je možné v XML identifikovat jednoduché elementy, kolekce elementů, struktury elementů a jejich kombinace.

XML umožňuje popsat strukturu dokumentu, ale nijak nedefinuje konečné zobrazení dokumentu. Nejjednodušším způsobem, jak vizualizovat XML dokument, je zobrazení stromové struktury XML dokumentu. Pro zpřehlednění je možné navíc použít CSS styly, nebo případně XML dokument transformovat pomocí XSL. Tato metoda má jednu zásadní nevýhodu – pro použití stylů je potřeba znát strukturu XML dokumentu. Neexistuje způsob, jak přehledně prezentovat zcela obecný XML dokument.

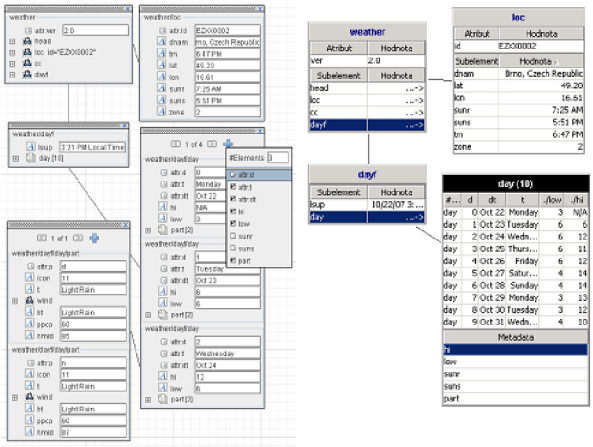
Metody, jak řešit vizualizaci obecného XML dokumentu, nicméně existují [20].

Vizualizace ala UML

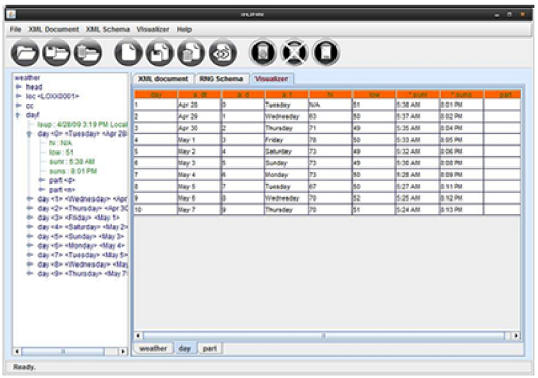
Tato technika hledá inspiraci v diagramech tříd jazyka UML. Modeluje komplexní elementy jako entity (třídy) a přidává jim data jednoduchých elementů a atributů jako vlastnosti. Dokáže také vizualizovat kolekci elementů jako seznam těchto elementů. Hierarchie struktury dokumentu je pak modelována použitím vztahu rodič – potomek. Nástroj používající tuto metodu vizualizace se jmenuje VisualXML [20], ukázka z programu je v levé části obrázku 2.1.

Vizualizace pomocí tabulek

Základní myšlenkou této metody je použití tabulek k zobrazení kolekcí záznamů. Jak bylo řečeno výše, nalezení takových kolekcí v XML dokumentech není příliš velkým problémem. Nejčastěji se v XML objevují právě kolekce elementů, které již samy nejsou kolekcemi, ale strukturami jednoduchých elementů. Příkladem modelovacích možností této metody je aplikace XMLAD a také novější aplikace XMLSpark [28]. Ukázky z programů jsou na obrázcích 2.1 a 2.2.



Obrázek 2.1: Ukázka z programů VisualXML (vlevo) a XMLAD (vpravo) (převzato z [20].



Obrázek 2.2: Ukázka z programu XMLSpark (převzato z [28])

Kapitola 3

XML a databáze

Potřeba uložení a zpracování XML dokumentů vedla k úpravě a rozšíření stávajících relačních databázích a také k vytvoření nového typu – nativní XML databáze.

3.1 XML jako databáze

XML je přirozený formát pro ukládání dat. Je tedy možné jej použít jako databázi?

Ačkoli se samotné XML používá k ukládání informací a dat (např. konfigurační soubory, formáty dokumentů Office Open XML a Open Document Format aj.), postrádá některé klíčové vlastnosti moderních databází.

Ve prospěch použití XML jako databáze hovoří několik faktů [31]:

- hierarchický model XML je ideální pro popis a uložení nejrůznějších typů dokumentů a dat.
- XML je samopopisující (značky popisují význam jednotlivých částí dat).
- používá jako výchozí kódování Unicode, XML dokumenty jsou tedy přenositelné napříč různými systémy.
- textový formát XML je přirozeně čitelný pro lidi.
- navíc zahrnuje celou řadu schémat (XSD, RNG, DTD), dotazovacích jazyků (XPath, XQuery) a dalších aplikací, které jsou v databázovém prostředí užitečné .

Na druhou stranu existuje několik pádných argumentů proti:

- neefektivní formát pro uložení dat (je příliš rozvláčný a přístup k datům je pomalý kvůli nutnému parsování a textovým konverzím).
- nepřítomnost mnoha technologií, které můžeme vidět u moderních databázových systémů (indexování, transakce, víceuživatelský přístup, bezpečnost, referenční integrita atd.).

XML je tedy vhodné spíše jen pro nenáročné jednouživatelské databáze s relativně malým objemem dat, jako jsou například konfigurační soubory, záložky v prohlížečích, kontakt listy instant messengerů apod. V prostředích, která mají mnoho uživatelů, vyžadují striktní integritu dat a rychlost databázového systému, by však nasazení XML selhalo.

3.2 Typy XML dokumentů z hlediska dat

XML dokumenty mohou být rozděleny na textově orientované a datově orientované.

3.2.1 Textově orientované

U textově orientovaných dokumentů převažuje text. Převážná část značek zde slouží pouze k přidání určité informace k textu, ale značky nejsou nezbytně nutné pro pochopení podstaty dokumentu. I kdyby byly všechny značky vynechány, bylo by stále možné pochopit alespoň část textu. Příklad textově orientovaného dokumentu:

Zdrojový kód 3.1: Textově orientovaný XML dokument

```
<paragraph>
  Lorem ipsum <emph>dolor</emph> sit amet consectetuer
  vel <strong>facilisi</strong> Morbi Quisque porta.
</paragraph>
```

3.2.2 Datově orientované

U datově orientovaných dokumentů je tomu jinak. Ty jsou určeny převážně pro počítačové zpracování či transportu dat a nehodí se k přímému čtení uživatelem. Značky zde vytvářejí strukturu dokumentu a přidávají datům určitý interpretační smysl. Kdyby byly z takového dokumentu značky vyjmuty, zbývající data by nedávala žádný smysl.

Datově orientované dokumenty se vyznačují pravidelnou strukturou a malými základními jednotkami dat (na úrovni elementů nebo atributů).

Zdrojový kód 3.2: Datově orientovaný XML dokument

```
<colorimeter>
  <rgb resolution="8">
    <red>0</red>
    <green>255</green>
    <blue>0</blue>
  </rgb>
</colorimeter>
```

Na předchozí ukázce je příklad datově orientovaného XML dokumentu. Pokud bychom z něj vyjmuli všechny značky, zbyl by pouze nicneříkající text „0 255 0“.

V praxi však často není rozdíl mezi textově a datově orientovanými dokumenty tak zřetelný. Například obvykle datově orientovaný dokument obsahující seznam zboží spolu s jejich parametry může obsahovat semi-strukturovaný popis zboží. Navzdory tomu může charakterizace dokumentů pomoci v rozhodování, jaký typ databáze použít. Datově orientované dokumenty se obecně ukládají do relačních, objektově orientovaných či hierarchických databází. Textově orientované XML dokumenty se obvykle ukládají do nativních XML databází.

3.3 Nativní XML databáze

Nativní XML databáze jsou databáze speciálně navržené pro ukládání a práci s XML dokumenty. Termín „nativní XML databáze“ může být zavádějící, protože mnoho dnešních takzvaných nativních databází nejsou vůbec samostatnými databázemi a neukládají XML

v opravdu nativní formě (tedy text). XML:DB initiative definuje nativní XML databáze takto [30]:

- definují logický model XML dokumentu, ukládají a vrací dokumenty podle tohoto modelu. (Model musí minimálně zahrnovat elementy, atributy, PCDATA a pořadí těchto uzlů v dokumentu.)
- mají XML dokument jako základní (logickou) jednotku uložení dat (podobně jako mají relační databáze řádek v tabulce jako základní jednotku uložení dat).
- nemusejí mít žádný konkrétní fyzický model pro uložení dat, mohou být postaveny na relační, hierarchické nebo objektově relační databázi, nebo mohou používat vlastní proprietární formát. Nemusejí vlastně být ani samostatnými databázemi.

Stejně jako jiné databáze i nativní XML databáze podporují transakce, bezpečnost, víceuživatelský přístup, dotazovací jazyky apod. Jediný rozdíl od ostatních databází je, že jejich vnitřní model je založen na XML.

Nativní XML databáze můžeme podle architektury rozdělit do dvou skupin: na textově založené a modelově založené [17].

3.3.1 Textově založené

Textově založené nativní XML databáze ukládají XML dokumenty jako text – např. jako soubor v souborovém systému, BLOB nebo CLOB v relační databázi, nebo jiný proprietární formát. Vytvářejí nad XML dokumenty indexy, které dovolí jednoduše a rychle vyhledat jakékoli místo v dokumentu.

Takovéto databáze jsou velmi rychlé při vracení celých dokumentů nebo souvislých XML fragmentů, protože k tomu typicky stačí jediný pohled do indexu a jediné čtení z paměti. Pokud je však nutné XML dokument nějak zpracovat (např. vybrat část dokumentu, vyhledat nějakou informaci apod.), jsou obvykle pomalejší, než modelově založené.

3.3.2 Modelově založené

Druhým typem nativních XML databází jsou modelově založené databáze, které si vytvářejí vnitřní objektový model dokumentů a tento model ukládají do databáze (do relační nebo objektově relační).

Modelově založené XML databáze, postavené na jiných databázových systémech, mají podobný výkon jako systém, který využívají. Výkon nicméně velmi závisí na návrhu datového modelu a databáze. Například dotazování u nativní XML databáze postavené nad relační databází, která využívá přímé mapování modelu DOM, může sestavení výsledku vést až k provedení klauzule SELECT pro získání každého elementu.

3.3.3 Některé nativní XML databáze

Dnes již existuje poměrně velké množství nativních XML databází. Zde proto budou stručně popsány pouze některé. Více nativních XML databází (otevřených i komerčních) lze najít v [18].

eXist

eXist je nativní XML databáze vyvíjená jako open source projekt. Je kompletně napsaná v Javě a může běžet jako samostatný databázový server, knihovna, nebo jako součást webové aplikace. eXist ukládá dokumenty do hierarchie kolekcí. Kolekce mohou obsahovat další kolekce a dokumenty jakéhokoli schématu [13].

eXist implementuje specifikaci XQuery jako datový model. Podporuje XML:DB API spolu s přidanými službami pro podporu přípravy a spouštění XQuery dotazů a správu uživatelů. Dále podporuje DOM a SAX pro dokumenty vrácené přes XML:DB API funkce. eXist je také možné volat přes XML-RPC, SOAP, WebDAV a další. eXist podporuje i transakce, ale jen pro účely zotavení po havárii. Transakce tak nejsou viditelné pro aplikaci [18].

Databáze podporuje dotazování jazyky XQuery a XPath, pro update uložených dokumentů lze použít XUpdate.

eXist poskytuje prostředí pro vývoj webových aplikací založených na XQuery a s ním spojených standardech. Celé webové aplikace mohou být napsány přímo v XQuery za použití XSLT, XHTML, CSS a Javascriptu.

Xindice

Xindice je nativní XML databáze napsaná v Javě určená speciálně pro ukládání velkého množství malých XML dokumentů a také ne-XML dokumentů. Umí indexovat elementy a atributy a také komprimovat dokumenty pro ušetření místa. Dokumenty ukládá do hierarchie kolekcí. Pro dotazování podporuje Xindice jazyk XPath, přičemž umožňuje použít název kolekce jako součást dotazu, čímž lze vykonat XPath dotaz přes více druhů dokumentů. Pro update dokumentů podporuje Xindice jazyk XUpdate.

Xindice podporuje několik aplikačních rozhraní. Je možné použít XML:DB API, CORBA API a XML-RPC.

Tamino

Tamino je komerční XML databázový server od německé firmy Software AG. Podporuje dotazovací jazyky XPath a XQuery a také technologie DOM, JDOM, SAX. Podporuje také zpracování XML dokumentů na straně serveru s použitím XSL a CSS. Navíc je možné rozšířit jeho funkcionalitu pomocí Tamino X-Tension. Jako aplikační rozhraní podporuje Tamino mimo jiné i XML:DB API.

Databáze umožňuje uložit nejen XML data, ale i ne-XML data (jako např. audio, video).

3.4 Databáze s podporou XML

Databáze s podporou XML jsou takové databáze, které nemají jako vnitřní datový model XML. Jsou to obvykle klasické relační nebo objektově relační databázové systémy obsahující rozšíření pro přenos dat mezi XML dokumenty a jejich vlastními vnitřními strukturami. Toto rozšíření může být integrováno přímo do databáze nebo může být připojeno k databázi externě.

Je obecně platným pravidlem, že databáze s podporou XML nedokážou pracovat se všemi typy XML dokumentů. Například relační databáze nedokáže dost dobře pracovat s XML dokumenty obsahujícími smíšený obsah (*mixed content*), protože jej nelze příliš dobře mapovat na relační model [31].

Narozdíl od nativních XML databází musejí databáze s podporou XML při ukládání XML dokumentu mapovat jednotlivé části dokumentu na schéma specifické struktury. Oproti tomu nativní XML databáze používají generické struktury, do kterých mohou být uloženy jakékoli XML dokumenty. Perzistencí XML v relačních databázích se dále zabývá kapitola 4.

3.5 Shrnutí

Nativní XML databáze se od databází s podporou XML liší především v tom, že nativní databáze dokážou zachovat fyzickou strukturu XML dokumentu (například použití entit, CDATA sekce, komentáře, instrukce pro zpracování atd.), což databáze s podporou XML neumí. Nativní XML databáze dokážou uložit XML dokumenty i bez znalosti jejich schématu (XML Schema, DTD). Jediným rozhraním pro data v nativních databázích je XML a s ním svázané technologie (např. XPath, DOM nebo nějaké XML specifické API, jako například XML:DB API). Databáze s podporou XML na druhou stranu nabízejí přímý přístup k datům skrze ODBC, JDBC apod.

Zatímco nativní XML databáze jsou vhodné především pro ukládání textově orientovaných XML dokumentů, databáze s podporou XML jsou vhodnější spíše pro ukládání XML dokumentů datově orientovaných.

Kapitola 4

Perzistence XML v relační databázi

Pro přenos dat mezi XML dokumenty a databází je nutné nejprve provést mapování XML schématu (DTD, XML, Relax NG atd.) na schéma databáze. Následující kapitola se bude zabývat možnostmi uložení XML dokumentu do relační či objektově relační databáze a metodami pro převod XML schématu na schéma databáze.

Techniky pro mapování XML dokumentů na databázové schéma můžeme v zásadě rozdělit na ([26]):

- generické metody – pro mapování nepoužívají schéma XML dokumentu
- schématem řízené metody – založeny na mapování s použitím existujícího XML schématu
- uživatelem řízené – metody mapování řízené uživatelem

4.1 Generické mapování

Metody generického mapování nepoužívají XML schéma. Tyto metody uvažují XML dokument jako ohodnocený orientovaný graf. Každý element dokumentu je reprezentován uzlem v grafu (každý uzel má v rámci grafu přiřazen unikátní identifikátor). Vztahy mezi elementy (element – podelement) jsou reprezentovány hranami grafu a označeny jménem podelementu. Vlastní data elementů jsou reprezentovány jako listy grafu. Tato grafová reprezentace XML dokumentu je jistým zjednodušením a některé informace mohou být ztraceny. Tomu lze zabránit uložením dodatečných informací do databáze [21].

Tyto metody definují obecné schéma, do kterého je možné uložit libovolný datově orientovaný dokument.

4.1.1 Hranová metoda mapování

Metoda ukládá všechny hrany grafu XML dokumentu do jediné tabulky hran. Tato tabulka hran obsahuje identifikátory zdrojového a cílového objektu grafu a jméno hrany. Dále obsahuje flag, který určuje, zda je hrana mezi dvěma vnitřními uzly grafu, nebo zda vede k listu (tedy hodnotě), a pořadové číslo hrany, protože hrany jsou uspořádané. Hranová tabulka má tedy následující strukturu [21]:

```
Hrana(zdroj, pořadí, jméno, flag, cíl)
```

Primárním klíčem této tabulky jsou sloupce zdroj a pořadí. Autoři metody dále navrhují vytvoření indexu na samotný sloupec zdroj a kombinovaného indexu nad sloupci jméno a cíl. Index nad sloupcem zdroj je užitečný pro rekonstrukci určitého objektu dle zadaného identifikátoru. Druhý uvedený index se hodí v případě zpětných průchodů.

4.1.2 Binární metoda

Tato druhá metoda je velmi podobná předchozí hranové metodě. Narozdíl od ní však sdružuje všechny hrany se stejným jménem a ukládá je do jedné tabulky. Při uložení XML dokumentu je tedy vytvořeno tolik tabulek, kolik se v elementu vyskytuje jmen elementů a atributů. Každá tabulka má následující strukturu:

```
B_jméno(zdroj, pořadí, flag, cíl)
```

Primární klíč zůstává stejný jako u hranové metody. Index nad sloupcem zdroj také zůstává, druhý index je vytvořen jen nad sloupcem target.

4.1.3 Mapování univerzální tabulkou

Poslední navrhovaná metoda používá jednu velkou univerzální tabulku, která odpovídá výsledku vnějšího spojení všech tabulek z binární metody. Pokud jsou n_1,...n_k jména, struktura takové tabulky je následující:

$\texttt{Universal(zdroj,pořadí}_{n1}\texttt{,flag}_{n1}\texttt{,cíl}_{n1}\texttt{,...,pořadí}_{nk}\texttt{,flag}_{nk}\texttt{,cíl}_{nk}\texttt{)}$

Taková tabulka je však denormalizovaná a vede k velkému množství null hodnot ve sloupcích.

4.1.4 Mapování hodnot dokumentu

Následující sekce se stručně věnuje mapování hodnot XML dokumentu (tedy textového obsahu elementů a atributů). V [21] lze najít dva různé přístupy – buďto uložení hodnot spolu s hranami, nebo v samostatných tabulkách. Oba přístupy je možné navíc kombinovat s výše uvedenými metodami mapování xml schématu.

4.1.5 Srovnání metod

Uvedené mapovací metody jsou poměrně jednoduché, avšak i tak dokážou podávat v některých oblastech poměrně dobrý výkon. Nejvýkonnější metodou se zdá být binární metoda v kombinaci s uložením hodnot u hran.

Všechny metody ale bohužel podávají velmi špatné výkony při kompletní rekonstrukci XML dokumentu (viz. [21]).

4.2 Schématem řízené mapování

Metody schématem řízeného mapování jsou založeny na existujícím schématu XML dokumentu. Z existujícího XML schématu je nejprve vygenerováno schéma databáze. Ukládané

XML dokumenty musí být validní vůči tomuto schématu. Při ukládání je dokument rozparsován a dle mapování vložen do tabulek.

Cílem těchto metod je vytvoření optimálního databázového schématu s rozumným počtem tabulek, jehož struktura maximálně odpovídá danému XML schématu.

Metody můžeme rozdělit na *fixní* a *flexibilní*. *Fixní* mapování nepoužívá žádné jiné informace než schéma samotné. Mapování je pak jasně dané jen a pouze schématem dokumentu. *Flexibilní* metody naproti tomu využívají dalších informací (např. statistiky dotazů) a soustředí se na vytvoření optimálního schématu pro danou aplikaci.

Schématem řízené metody mapování mají několik společných vlastností [26]:

- Podelementy s `maxOccurs = 1` jsou namapovány (namísto do nové tabulky) do tabulek s rodičovskými elementy (tzv. *inlining*).
- Elementy, které se mohou vyskytovat vícekrát (`maxOccurs > 1`), jsou namapovány do zvláštních tabulek. Vztah element – podelement je modelován za použití cizích klíčů (*foreign keys*).
- Podelementy s alternativním výskytem jsou mapovány buď do speciálních tabulek, nebo do jedné univerzální tabulky.
- Pokud je nutné zachovat pořadí elementů, informace se mapuje do speciálního sloupce.
- Elementy se smíšeným obsahem (*mixed content*) obvykle nejsou podporovány.
- Rekonstrukce elementu typicky vyžaduje spojení několika tabulek.

4.2.1 Basic, Shared, Hybrid

Basic, Shared a Hybrid jsou asi nejznámějšími zástupci fixních metod. Metody jsou založeny na DTD. Stěžejním tématem těchto metod je rozhodnutí, zda pro element vytvořit novou tabulku, nebo jej inlinovat (zahrnout) do jeho rodičovského elementu. Vztahy elementů jsou pak reprezentovány cizími klíči mezi tabulkami. Jednotlivé metody se liší v množství redundance, kterou mohou generovat [29].

Basic

Metoda Basic se snaží inlinovat co nejvíce potomků elementu. Vytváří však tabulky pro každý element, který se vyskytne ve vstupním DTD. Hlavními nevýhodami metody jsou velký počet nepotřebných tabulek.

Shared

Tato metoda vychází s Basic a snaží se redukovat počet vytvářených tabulek. Její ideou je identifikovat uzly elementů, které jsou reprezentovány ve více tabulkách metody Basic, a sdílet tyto elementy vytvořením speciálních tabulek [29].

Metoda dále také vytváří tabulky pro elementy, jejichž možný počet výskytů v rodičovském elementu je větší než jedna. Všechny uzly s násobností rovnou jedné, které se dále nikde v dokumentu neopakují, jsou inlinovány do tabulky rodiče.

Všechny tři metody používají v tabulkách primární a cizí klíče pro uchování vztahu rodič – potomek.

Hybrid

Metoda Hybrid se od metody Shared liší v podstatě jen tím, že inlinuje navíc některé elementy, které nejsou inlinovány metodou Shared. Hybrid v principu inlinuje i elementy (nerekurzivní), které se mohou v rodičovském elementu vyskytnout více než jednou.

4.2.2 LegoDB

LegoDB je zástupcem flexibilních mapovacích metod. Metoda automaticky prozkoumává prostor všech možných relačních mapování daného xml schématu a vybírá nejlepší mapování pro danou aplikaci.

Vstupem LegoDB jsou parametry, které popisují cílovou aplikaci (XML Schema, statistiky dotazů a příklady XML dokumentů). Výstupem je pak efektivní relační mapování. Více informací lze nalézt v článku [16].

LegoDB vede na nejefektivnější mapování pro danou aplikaci. Nevýhodou však je, že pokud se aplikace změní (uživatel se začne ptát jinak), efektivita rapidně klesá. Modifikace schématu také není triviální záležitostí.

4.3 Možné problémy mapování

4.3.1 Uložení kompletního dokumentu

Při použití některých technik mapování XML na schéma databáze nemusí daná metoda mapovat všechny části XML dokumentu (např. komentáře, instrukce pro zpracování atd.). Při rekonstrukci dokumentu se pak původní a rekonstruovaný dokument mohou lišit.

Toto je možné obejít uložením originálního XML dokumentu buďto do souborového systému, nebo do BLOB/CLOB. Řešení má výhodu i v tom, že při dotazu vracejícím kompletní dokument není nutné rekonstruovat celý dokument (což je časově a výkonnostně náročný proces) a může být vrácen tento zazálohovaný dokument. Bylo by také možné textově uložený dokument proindexovat a některé složitější dotazy zodpovídat přímo zpracováním textu.

Nevýhodou tohoto přístupu je, že uložená kopie XML dokumentu zabírá další místo.

4.3.2 Smíšený obsah

Při uložení XML dokumentu do systému založeném na relační či objektově relační databázi vyvstává problém s uložením smíšeného obsahu elementů, protože jej nelze dost dobře namapovat na relační model dat.

Částečné řešení se nabízí v uložení smíšeného obsahu do sloupce typu BLOB (nebo CLOB, pokud bude zpracováván) a případné zpracování jeho obsahu pomocí SQL funkcí určených pro práci s XML (např. `EXTRACTVALUE()` nebo `EXTRACTXMLCLOBVALUE()`).

4.3.3 Editace uloženého dokumentu

Při použití některých metod mapování vyvstává problém s editací a updatem částí dokumentu. XML dokumenty standardně nedefinují žádné identifikátory elementů, takže upravený obsah není možné adresovat na správné místo v uloženém dokumentu.

Problém se snaží řešit rozšíření jazyka XQuery – XQuery Update Facility, které je momentálně ve fázi W3C Candidate recommendation. XQuery Update Facility poskytuje

výrazy, které mohou být použity k aplikování perzistentních změn v datových modelech XQuery a XPath [12]. Dosud jediným řešením je často přepsání staré verze XML dokumentu změněnou verzí.

4.3.4 Rekurze při průchodu dokumentem

Některé metody mapování vedou k vytvoření velkého množství tabulek, které je při průchodu či vyhledávání v dokumentu potřeba spojovat, což vede k velkému množství dotazů s mnoha spojeními. Navíc pro rekonstrukci stromu XML dokumentu může být nutné provádět dotazy rekurzivně.

Možné řešení spočívá v uložení hierarchie (pomocí tečkové notace) v pomocném identifikátoru uzlu – např. hodnota „1.55.7“ říká, že element s id 7 je potomkem elementu s id 55. Identifikátory v jednotlivých vrstvách navíc udávají pořadí elementu.

Kapitola 5

Použitá metoda mapování

V této práci jsem se pokusil navrhnout a implementovat schématem řízenou metodu mapování XML dokumentů na tabulky relační databáze. Mapování řízené schématem bylo vybráno s ohledem na dobrý poměr počtu tabulek ku složitosti rekonstrukce uloženého dokumentu. Jak již bylo uvedeno v předcházející kapitole, schématem řízené mapování má za cíl mapovat XML schéma na databázové schéma s rozumným počtem tabulek.

Použitá metoda vychází z výše zmíněné mapovací metody Hybrid s několika úpravami. Metoda Hybrid byla navržena pro použití se schématem DTD, jehož možnosti už dnešním požadavkům na definici XML dokumentů nepostačují. Proto bude v této práci použito k definici schématu XML výhradně schematu XML Schema (což vzhledem ke vzájemné převoditelnosti RNG, XSD a ostatních XML schémat nevadí).

Metoda je určena výhradně pro datově orientované dokumenty, nicméně s drobnými úpravami je použitelná i pro dokumenty, které obsahují elementy se smíšeným obsahem. K tomu je ovšem zapotřebí buďto podpora XML v použité relační databázi, nebo dodatečné zpracování.

5.1 Způsob inlinování elementů

Inlinování je výraz pro metodu mapování, kdy je element namísto do své vlastní tabulky namapován jako sloupec v tabulce svého rodičovského elementu.

Nejjednodušší možná metoda mapování inlinování nepoužívá a pro každý element vytváří speciální tabulku. To vede k velkému počtu tabulek v databázi. Dalším nežádoucím efektem je pak velké množství spojení nutné k rekonstrukci dokumentu.

Opakem této jednoduché metody by byla metoda, která inlinuje úplně všechny podřízené elementy a atributy do tabulky rodičovského elementu. Tento extrémní způsob mapování by vedl na jednu jedinou tabulku, která by obsahovala celý dokument. Tato tabulka by byla silně denormalizovaná a umožňovala by uložit pouze jeden konkrétní dokument, pro který byla vytvořena (platí pro složitější XML dokumenty, které obsahují seznamy elementů). Vztahy 1:N by byly totiž modelovány konkrétním počtem inlinovaných podřízených elementů.

Cílem námi popisované metody je nalezení optimálního kompromisu mezi inlinováním a vytvářením speciální tabulky. Metoda se snaží inlinovat, kdykoli je to možné a rozumné. O tom, zda bude element nebo atribut inlinován, rozhoduje četnost jeho výskytu v rodičovském elementu. Tuto informaci lze zjistit ze XML schéma dokumentu. Četnost výskytu obsahují XML Schema atributy *minOccurs* a *maxOccurs*.

Atribut *minOccurs* označuje minimální počet výskytů daného elementu. Defaultní hodnotou, pokud není atribut uveden, je hodnota 1.

Atribut *maxOccurs* obsahuje maximální počet výskytů daného elementu. Defaultní hodnotou je opět 1. Pokud tedy není ani jeden z atributů uveden, je element povinný a může se v rodičovském elementu vyskytnout právě jedenkrát.

Pokud je tedy například u jednoduchého (dále nestrukturovaného) elementu maximální počet výskytů např. 3, je rozumné uvažovat o jeho inlinování do tabulky rodiče. Pokud se však jedná o element, který dále obsahuje další elementy (komplexní typ, viz dále), nebo je jeho možný počet výskytů neomezený (*unbounded*), je rozumnější pro něj vytvořit speciální tabulku.

5.2 Mapování jednoduchých typů

Jednoduchý typ (*simple type*) je XML element, který obsahuje pouze text. Nesmí obsahovat žádný jiný element nebo atribut [3]. Jednoduchým typem jsou v XML automaticky také všechny atributy, protože ty mohou obsahovat pouze atomické hodnoty.

XML Schema má mnoho vestavěných datových typů, ale typicky se pro jednoduché elementy a atributy používají především tyto:

- xs:string – textové řetězce
- xs:decimal – reálná čísla
- xs:integer – celá čísla
- xs:boolean – booleovské hodnoty
- xs:date – uložení data
- xs:time – uložení času

Jednoduché elementy a atributy mohou být v XML Schema definovány více způsoby. Jedním ze způsobů je pro element `<xs:element name="jmenoElementu" type="typ"/>` a atribut `<xs:attribute name="jmenoAtributu" type="typ"/>`, kde atribut *type* obsahuje jméno jednoduchého vestavěného nebo uživatelského datového typu. Ty mohou navíc obsahovat i omezení (*restriction*). Omezením se věnuje samostatná kapitola 5.5.

Atributy jsou vždy mapovány jako sloupce v tabulce svého rodiče, protože v XML nemůže mít element více atributů se stejným jménem (jinak řečeno, atributy mají maximální počet výskytů v rodiči vždy roven 1).

U jednoduchých elementů toto vždy neplatí. Elementy jsou inlinovány v rodiči pouze v případě, že jejich maximální počet výskytů (*maxOccurs*) v rodičovském elementu nepřesáhne rozumnou úroveň. Pokud je jejich možný počet vyšší než tato rozumná mez (např. *unbounded*), je pro ně vytvořena vlastní tabulka.

5.3 Mapování komplexních elementů

Komplexní element je XML element, který může obsahovat jiné elementy a atributy. Tyto podřazené elementy mohou být jednoduché, nebo komplexní. XML Schema rozeznává čtyři druhy komplexních elementů [3]:

- prázdné komplexní elementy - obsahují pouze atributy
- elementy, které obsahují pouze jiné elementy a atributy
- elementy, které obsahují pouze text a atributy
- elementy, které obsahují smíšený obsah (elementy, atributy i text)

První tři typy komplexních elementů jsou navrhovanou metodou podporovány a ukládány strukturovaně do databázového schématu. Čtvrtý typ – element se smíšeným obsahem – je nutno uložit v textové podobě, protože jej nelze dost dobře namapovat na relační model.

Pokud má komplexní element definován maximální počet výskytů v rodiči na 1, je inlinován. Jinak je pro něj vytvořena nová tabulka.

5.4 Mapování vestavěných datových typů XML schématu

Jak již bylo řečeno dříve v této kapitole, jednoduché elementy a atributy jsou mapovány na sloupce v tabulkách svých rodičovských elementů. I komplexní elementy je vždy možné rozložit na strukturu jednoduchých elementů a atributů.

Jelikož jazyk XML Schema (stejně jako ostatní pokročilejší jazyky pro definici XML schématu) umožňuje definovat datové typy jednotlivých elementů a atributů, měla by mapovací metoda tyto datové typy zohlednit. To umožní provádět lepší validaci dokumentu vůči schématu a poskytnout optimální uložení dat.

Vestavěné jednoduché datové typy jsou mapovány na co nejvhodnější datové typy použité databáze [27].

5.4.1 Textové datové typy

Datový typ String a typy z něj odvozené (uvedeny v [3]) jsou mapovány na vhodné textové datové typy databáze. U použité SQL databáze PostgreSQL přicházejí v úvahu typ `CHAR(N)`, `VARCHAR(N)` (kde N je vhodně zvolená konstanta) a typ `TEXT`.

Datový typ `CHAR(N)` má fixně nastavenou délku a je vhodný pouze v případě, kdy je ve schématu uvedena pevná délka obsahu daného elementu nebo atributu.

Datový typ `VARCHAR(N)` umožňuje definovat variabilní délku textu až do maximální délky N, kde N je vhodně zvolená konstanta. Stejně jako u typu `CHAR` je jeho použití vhodné tehdy, kdy z XML schématu vyplývá délkové omezení textu.

Naopak datový typ `TEXT` je v případě databáze PostgreSQL délkově neomezen a může obsahovat řetězec libovolné délky (až do velikosti 2GB). Toho je s výhodou využito při implementaci.

5.4.2 Číselné datové typy

Datové typy představující číselnou hodnotu.

Typ decimal a typy z něj odvozené

Datový typ decimal představuje, stejně jako v databázích, neomezeně velké reálné číslo s možností definice přesnosti. Proto se nabízí mapování na databázový datový typ `DECIMAL(M,N)` nebo `NUMERIC(N,M)` (kde M a N jsou vhodně zvolené konstanty).

Ostatní celočíselné datové typy odvozené od decimal je pak možné vhodně mapovat na typ `SMALLINT`, `INTEGER` nebo `BIGINT` s integritním omezením dle daného typu.

Typ boolean

Typ boolean lze mapovat na SQL typ `BOOLEAN`.

Typy float a double

Typy float a double je možné mapovat na SQL datový typ `FLOAT` nebo `DOUBLE`.

5.4.3 Datum a čas

Následující část popisuje vhodné mapování časových datových typů.

date, dateTime a time

Datový typ Date je možno mapovat na databázový typ `DATE`, dateTime na typ `TIMESTAMP WITH TIME ZONE` a time na `TIME WITH TIME ZONE`.

duration

Datový typ Duration definuje časový interval. Je možné jej namapovat na databázový typ INTERVAL.

ostatní typy data a času

XML Schema dále definuje několik dalších datových typů představujících části data (např. gMonth, gMonthDay, gYear a další). Tyto datové typy není možné mapovat na datový typ date. Jelikož jsou hodnotami těchto typů čísla, lze je namapovat na datový typ `INTEGER`.

5.5 Mapování omezení

XML Schema umožňuje definovat restrikce. Restrikce mohou být použity pro specifikování přípustných hodnot XML elementů nebo atributů. Následující sekce se zabývá možnostmi mapování restrikcí do databázového schématu. Definice SQL omezení jsou pro databázi PostgreSQL, jehož implementace odpovídá standardu SQL:2003.

enumeration

Definuje seznam přípustných hodnot. Je možné mapovat na SQL omezení `CHECK (col IN [množina hodnot])`, kde col je název sloupce.

length, minLength, maxLength

Tyto restrikce mohou být použity na textové datové typy. Specifikují přípustný počet znaků textového řetězce. Tento počet musí být nezáporný. Restrikci *length* lze mapovat na SQL omezení `CHECK (char_length(col)=M)`, minLength na `CHECK (char_length(col)>=M)` a maxLength na `CHECK (char_length(col)<=M)`, kde col je název sloupce a M je hodnota restrikce.

minExclusive, maxExclusive

Tyto restrikce mohou být použity pro číselné datové typy a pro typy data a času. Specifikují horní a dolní hranici pro číselné hodnoty. Přípustná hodnota musí být menší (maxExclusive) nebo větší (minExclusive) než hodnota restrikce. Lze mapovat na SQL omezení `CHECK (col<M)` pro maxExclusive a `CHECK (col>M)` pro minExclusive, kde col je název sloupce a M je hodnota omezení.

minInclusive, maxInclusive

Tyto restrikce mohou být použity pro číselné datové typy a pro typy data a času. Specifikují horní a dolní hranici pro číselné hodnoty. Přípustná hodnota musí být menší nebo rovna (maxInclusive), nebo větší nebo rovna (minInclusive) hodnotě restrikce. Lze mapovat na SQL omezení `CHECK (col<=M)` pro maxInclusive a `CHECK (col>=M)` pro minInclusive, kde col je název sloupce a M je hodnota omezení.

pattern

Regulárním výrazem definuje vzor pro přípustnou hodnotu. Restrikce lze mapovat na `CHECK (col SIMILAR TO regExp)`, kde col je název sloupce a regExp je daný vzor.

whiteSpace

Specifikuje, jak má být zacházeno s bílými znaky. Toto není možné definovat jako omezení databáze, spíše se toto omezení vztahuje na aplikaci, která XML dokument zpracovává.

totalDigits a fractionDigits

Specifikuje přesný počet číslic, který je povolen. Lze mapovat na definici typu sloupce `DECIMAL()` nebo `NUMERIC()`.

další omezení

Dalším omezením, které lze odvodit z XML schématu, je omezení na přítomnost hodnoty. Omezení `NOT NULL` lze použít při `minOccurs=1` a `maxOccurs=1` (u elementu) nebo `use="required"` (u atributu).

Implementovaná databázová vrstva výše popsaná omezení nevyužívá, protože v případě neexistujícího XML schématu je pomocí knihovny Trang z poskytnutého XML dokumentu vygenerováno XML Schéma nové. Protože je však schéma generováno na základě jednoho jediného dokumentu, nemusí přesně odpovídat struktuře všech dokumentů, které by měly být vůči danému schématu validní. Proto jsou některé restrikce záměrně ignorovány, aby se předešlo případným nežádoucím chybám při vkládání dokumentů bez schématu.

5.6 Převod XML schématu na databázové schéma

V této části bude představen postup převodu existujícího XML schématu na schéma relační databáze. Algoritmus byl odzkoušen na schématu XML Schema, ale bude fungovat i s ostatními XML schématy (obdobná metoda založená na RNG je implementována v XMLSpark [28]).

5.6.1 Graf schématu

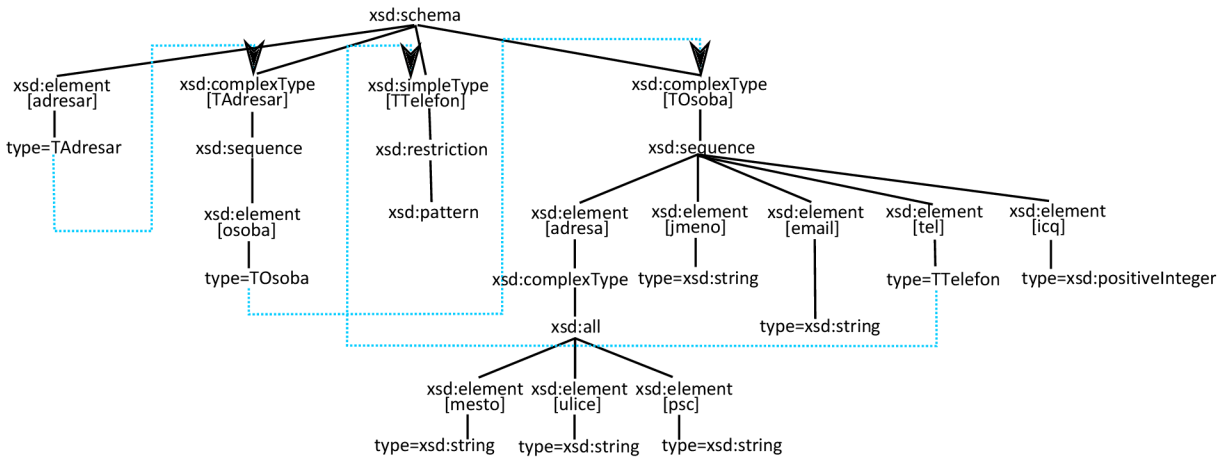
Algoritmus nejprve vytvoří graf schématu, což je DOM strom daného XML schématu s přidanými hranami. Ukázka jednoduchého grafu schématu z 2.4 je na obrázku 5.1 (hrany také mohou být simulovány použitím vhodných funkcí nad DOM stromem schématu).

Zde je možné vidět, že graf schématu má čtyři globální uzly. Globálními uzly jsou všechny přímé poduzly kořenového uzlu xsd:schema. Naproti tomu jako lokální elementy se označují všechny zanořené uzly grafu. Globální uzly mohou definovat strukturu elementů nebo uživatelských datových typů, na které je pak možno se odkazovat z dalších uzlů.

Typové a referenční odkazy jsou znázorněny šipkami (orientovanými hranami) z listových uzlů grafu na jeho globální uzly (viz. obrázek 5.1). Tyto hrany můžeme rozdělit na dva typy:

- hrany spojující element s globálně definovaným jednoduchým nebo komplexním datovým typem (atribut type).
- hrany spojující element, který odkazuje jiný globálně definovaný jednoduchý nebo komplexní element nebo atribut (atribut ref).

Na základě těchto hran algoritmus rozhoduje, zda je aktuálně zpracovávaný uzel jednoduchého nebo komplexního typu. Pokud uzel obsahuje uzel s názvem simpleType, je jednoznačně jednoduchého typu. Pokud obsahuje uzel type, jehož obsahem je název vestavěného jednoduchého datového typu (xs:string, xs:integer apod.), je také jednoduchého typu. Pokud však zpracovávaný uzel obsahuje uzly type nebo ref, které obsahují název globálně definovaného uzlu, je nutné rekurzivně zjistit, jakého typu je tento globální uzel. Komplexní typ je určen tím, že zpracovávaný uzel obsahuje uzel s názvem complexType, nebo se pomocí ref nebo type odkazuje na jiný globální komplexní uzel.



Obrázek 5.1: Ukázka grafu schématu

5.6.2 Průchod grafem

Pro vytvoření databázového schématu je nutné projít a zpracovat vytvořený graf schématu.

Nejprve je nutné nalézt uzel, který bude zpracován jako první. Ten představuje kořenový element všech XML dokumentů validních vůči danému schématu. Tento kořenový element bude algoritmem použit jako výchozí bod při zpracovávání grafu. Obecně může být globálních elementových uzlů ve schématu libovolné množství. Metoda proto používá jako výchozí bod zpracování grafu ten globální uzel, který není nikde v grafu schématu referencován (nevedou do něj v grafu žádné zpětné hrany). V případě, že je takových vstupních uzlů ve schématu více, je nutné vytvořit zvláštní databázové schéma pro všechny tyto kořenové uzly.

Algoritmus tvorby schématu tedy začíná od neodkazovaného globálního elementového uzlu. Postupně jsou zpracovávány všechny podřízené uzly. Pro počáteční kořenový element je vždy vytvořena vlastní tabulka. Ta obsahuje informace o dokumentu, který reprezentuje, a o jeho kódování. Tím je umožněno mít v jedné kolekci dokumentů dokumenty s různým kódováním. Pak jsou postupně rekurzivně zpracovávány všechny poduzly kořenového uzlu.

Pokud je zpracovávaný uzel definicí elementu, je na základě následujících kritérií rozhodnuto co dál:

1. Pokud je uzel jednoduchého typu a zároveň má definovánu maximální četnost menší nebo rovnu M (kde M je vhodně zvolená konstanta), je inlinován do tabulky rodiče. Zpracování uzlu končí.

2. Pokud je uzel jednoduchého typu a zároveň je jeho maximální četnost větší než M, je pro něj vytvořena tabulka. Zpracování uzlu končí.

3. Pokud je uzel komplexního typu a jeho maximální četnost není větší než 1, je inlinován do tabulky rodiče. Zpracování dále pokračuje jeho podřízenými uzly.

4. Pokud je uzel komplexního typu a jeho maximální četnost je větší než 1, je pro něj vytvořena tabulka a zpracování dále pokračuje jeho podřízenými uzly.

Pokud je zpracovávaný uzel definicí atributu, je vždy inlinován do tabulky rodiče.

Pokud je aktuálním zpracovávaným uzlem uzel představující sekvenci uzlů (xsd:sequence), je ke všem dále zpracovávaným uzlům přidáno pořadí (buď jako speciální sloupec v tabulce, nebo v metadatech schématu). Z tohoto pohledu je zpracování uzlu definujícího skupinu uzlů (xsd:all) ignorováno, protože ten pouze říká, že se poduzly mohou vyskytnout v jakémkoli pořadí.

Všechny vytvořené tabulky obsahují jednoznačný identifikátor elementu v dokumentu. Dále všechny (kromě tabulky kořenového elementu) obsahují cizí klíč do tabulky rodičovského elementu (pokud je element sdílen, může být rodičovských elementů obecně více). Příklad vygenerovaného databázového schématu pro graf z obrázku 5.1 je na následujícím diagramu 5.2.

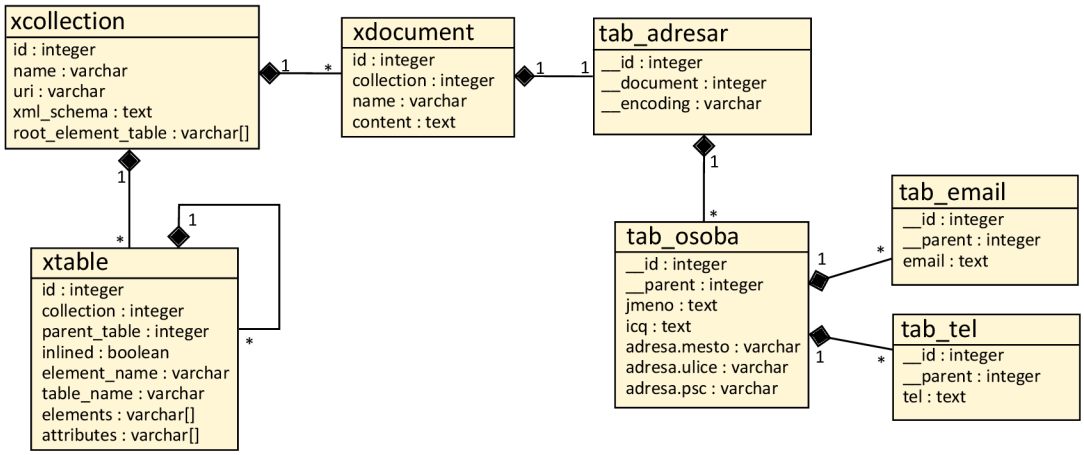
Kromě vytvoření fyzických tabulek schématu je také nutné uložit metadata o těchto tabulkách do připravených tabulek v databázi. K tomu slouží tři tabulky:

- Tabulka `xcollection` slouží k uložení informací o jednotlivých kolekcích dokumentů v databázi.

- V tabulce `xdocument` jsou uloženy informace o dokumentech (id, jméno, obsah atd.).

- Tabulka `xtable` obsahuje metadata o mapování XML schématu na databázové schéma.

Podrobněji jsou tabulky popsány v kapitole 6.3.5.



Obrázek 5.2: Vygenerované databázové schéma

5.7 Ukládání XML dokumentu do databáze

Pro správné uložení XML dokumentu do databáze musí být splněno několik předpokladů:

- dokument musí být správně strukturovaný (well formed)
- dokument musí odpovídat schématu v databázi

Před vlastním uložením dokumentu do databáze je nejprve nutné vytvořit seznam SQL dotazů, které vloží jednotlivé elementy do správných tabulek databáze.

K tomu musí systém vědět, jak přesně jsou mapovány jednotlivé prvky XML dokumentu. Nejprve tedy systém načte z databáze metadata o mapování XML schématu na odpovídající databázové schéma. Tato metadata by měla obsahovat informace o tabulkách a informace, které elementy a atributy jsou do kterých tabulek namapovány. Pomocí těchto dat si vytvoří odpovídající hierarchii tabulek tak, jak se jim odpovídající elementy mohou vyskytovat v toku dokumentu.

V dalším kroku pak systém postupně rekurzivně prochází stromem dokumentu a zároveň stromem tabulek a na jejich základě generuje SQL dotazy pro vložení elementů a atributů do tabulek schématu. Tyto dotazy jsou generovány od listů stromu směrem ke kořenovému uzlu dokumentu, proto jsou nejdříve připraveny dotazy pro vložení nejzanořenějších elementů a atributů. Tyto SQL dotazy však není možné ihned vykonat, protože fyzické tabulky obsahují cizí klíče do tabulek rodičovských elementů. Proto je nutné SQL dotazy spustit, až když jsou vygenerovány úplně všechny. Do každého dotazu je těsně před jeho vykonáním vložena hodnota cizího klíče do nadřazené tabulky.

5.8 Rekonstrukce dokumentu

K rekonstrukci dokumentu, nebo jeho části, musí systém opět načíst všechna metadata schématu dokumentu, který má být zrekonstruován zpět do XML. Rekonstrukce elementu představuje zpracování tabulky, do které je daný element namapován, a také zpracování všech tabulek všech potomků tohoto elementu.

Pro rekonstrukci celého dokumentu tedy systém postupuje od tabulky kořenového elementu a postupně zpracovává všechny tabulky schématu. Jak již bylo řečeno dříve, tabulky jsou provázány cizími klíči, takže lze postupným dotazováním sestavit i jeden konkrétní uložený dokument. To by bylo možné i pomocí jednoho dotazu se spojením všech potřebných tabulek přímo v databázi. Tento způsob by však vyžadoval další parsování výsledků dotazu a také by zřejmě nebyl příliš efektivní.

Po načtení informací z tabulky elementu pak systém podle schématu rekonstruovaného elementu sestavuje DOM strom požadovaného dokumentu. Dokument je pak možné vrátit přímo jako DOM, nebo text (transformací DOM stromu do textové podoby).

5.9 Zpracování XPath dotazu

Následující podkapitola stručně vysvětlí průběh zpracování XPath dotazu a vyhodnocování predikátů nad rekonstruovaným dokumentem. Zabývá se výhradně jednoduchými XPath dotazy.

Při vyhodnocování výrazu v jazyce XPath je nutné postupovat po jednotlivých krocích. Při každém kroku je potřeba udržovat kontext vyhodnocení pro kroky, které mohou následovat. Kontext sestává z aktuálního kroku, aktuální tabulky a aktuálního výsledku databáze.

Zjištění odpovídající tabulky

V každém kroku XPath je nutné nejprve zjistit tabulku, do které je element daného kroku mapován. Proto je v kontextu mezi jednotlivými kroky předávána aktuální tabulka. V prvním kroku je jako aktuální tabulka nastavena tabulka kořenového elementu. Každý krok tedy nejprve zkontroluje, zda je inlinován v aktuální tabulce, nebo zda je mapován do vlastní tabulky. Pokud není inlinován ani mapován do tabulky, jedná se o chybu v XPath dotazu a je vrácen prázdný výsledek.

Pokud je zpracovávaným krokem některý ze speciálních kroků //, * nebo @*, je nutné ze schématu dokumentu odvodit všechny možné cesty k elementu nebo atributu kroku, který bude následovat. Ke každé z nalezených cest je postupně připojen dosud nezpracovaný zbytek XPath výrazu a tento nový XPath výraz je pak v aktuálním kontextu vykonán.

Vyhodnocení predikátů

Poté jsou vyhodnoceny predikáty kroku (pokud nějaké má). Způsob jejich vyhodnocení se liší podle toho, zda je element inlinován, nebo ne. Pokud je prvek inlinován, predikáty jsou vyhodnoceny přímo nad již načtenými daty. Pokud prvek inlinován není, predikáty kroku jsou převedeny na odpovídající SQL a připojeny k dotazu na tabulku (za klauzulí `WHERE` nebo `LIMIT`). Tím je omezen počet možných zpracovávaných výsledků v dalších krocích XPath.

Rekonstrukce výsledků

Pokud krok vyhověl predikátům, postupuje algoritmus k jeho dalšímu zpracování. Pokud krok odpovídá samostatné tabulce, je vykonán SQL dotaz na tuto tabulku (viz. výše) a přes všechny vrácené výsledky je pak iterováno v dalším kroku XPath. Pokud je dosaženo konce XPath výrazu (nebo je dalším krokem výrazu funkce), je aktuální element nebo atribut rekonstruován způsobem popsaným v sekci 5.8.

Kapitola 6

Návrh a implementace API

Kapitola obsahuje popis návrhu a implementace databázové mezivrstvy, která nad použitou relační databází umožní vkládání a dotazování XML dokumentů. Aplikace byla pracovně nazvána *xDB*. Dále v textu bude vyvíjená mezivrstva označována také jako middleware, konektor nebo systém.

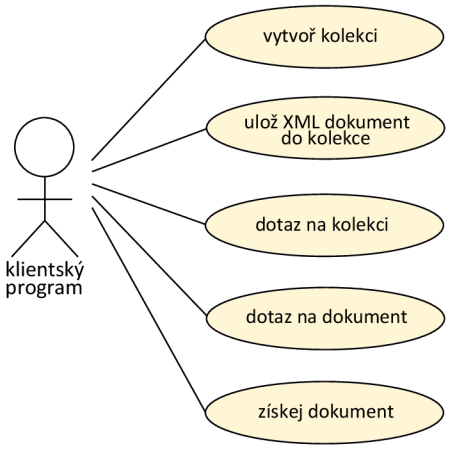
6.1 Specifikace požadavků

Neformální specifikaci je možné shrnout do následujících bodů:

- Systém musí umožňovat pracovat s XML dokumenty (must)
- Systém musí umožňovat nastavení připojení k použité relační databázi (must)
- Systém musí sdružovat dokumenty do kolekcí (must)
- Systém by mohl umožňovat hierarchicky sdružovat kolekce (optional)
- Systém musí umožňovat ukládat a mazat XML dokumenty (must)
- Systém musí umožňovat dotazování pomocí XPath nad zvolenou kolekcí XML dokumentů (must)
- Systém musí umožňovat dotazování pomocí XPath nad zvoleným XML dokumentem (optional)
- API systému musí být kompatibilní s API jiných XML databází (must)

6.2 Případy použití

Na základě specifikace požadavků byl sestaven jednoduchý diagram případů použití, který ukazuje obrázek 6.1. Diagram zobrazuje jednoho aktéra. Je jím klientský program, který používá připojení k relační databázi skrze middleware xDB.



Obrázek 6.1: Diagram případů použití API xDB

6.2.1 Detaily případů použití

UC1. Vytvoření nové kolekce

Případ použití: Vytvoření nové kolekce
Identifikátor: UC1
Popis Klientský program vytvoří v databázi novou kolekci.
Předpoklady Klientský program je připojen k databázi.
Aktéři Klientský program
Hlavní tok 1. Případ použití se spustí, když klientský program zavolá příslušnou metodu pro vytvoření kolekce. 2. Klientský program zadá jméno kolekce. 3. Systém ověří existenci kolekce daného jména. 4. Systém vytvoří novou kolekci. 5. Systém připojí klientský program k nové kolekci.
Následné podmínky Byla vytvořena nová kolekce. Klientský program byl připojen k nové kolekci.
Výjimky Kolekce daného jména již existuje
Výjimka: Kolekce daného jména již existuje
Identifikátor: UC1.E.1
1. Případ použití začíná po kroku 3 hlavního toku. 3. Systém vypíše na chybový výstup informaci o dané chybě. 4. Systém vyvolá příslušnou výjimku.

Tabulka 6.1: Případ použití: Vložení XML dokumentu do kolekce

UC2. Vložení XML dokumentu do kolekce

Případ použití: Vložení XML dokumentu do kolekce
Identifikátor: UC2
Popis Klientský program vloží nový XML dokument do vybrané kolekce v databázi.
Předpoklady Klientský program je připojen k vybrané existující kolekci, do které chce dokument vložit.
Aktéři Klientský program
Hlavní tok 1. Případ použití se spustí, když klientský program zavolá příslušnou metodu pro uložení dokumentu. 2. Pokud vybraná kolekce neobsahuje schéma. 2.1 Systém pro daný dokument vygeneruje XML schéma. 2.2 Systém převede vytvořené XML schéma na databázové schéma kolekce. 3. Systém zkontroluje, zda je vkládaný dokument validní vůči schématu kolekce. 4. Systém uloží dokument do kolekce.
Následné podmínky Dokument byl uložen do databáze.
Výjimky Dokument není validní vůči schématu kolekce.
Výjimka: Dokument není validní vůči schématu kolekce
Identifikátor: UC2.E.1
1. Případ použití začíná po kroku 3 hlavního toku. 2. Systém zruší veškeré změny, které již byly v databázi provedeny. 3. Systém vypíše na chybový výstup informaci o dané chybě. 4. Systém vyvolá příslušnou výjimku.

Tabulka 6.2: Případ použití: Vložení XML dokumentu do kolekce

UC3. XPath dotaz na kolekci

Případ použití: XPath dotaz na kolekci
Identifikátor: UC3
Popis Klientský program spustí nad kolekcí dotaz v jazyce XPath.
Předpoklady Klientský program je připojen k vybrané kolekci, do které chce dokument vložit.
Aktéři Klientský program
Hlavní tok 1. Případ použití se spustí, když klientský program zavolá příslušnou metodu pro dotazování kolekce. 2. Systém zkontroluje, zda je poskytnutý XPath dotaz validní. 3. Systém vykoná dotaz. 3.1 Pokud kolekce neobsahuje žádný dokument, je vrácen prázdný výsledek. 3.2 Systém vrátí všechny dokumenty nebo části dokumentů v kolekci, které odpovídají dotazu.
Následné podmínky Systém vrátil všechny dokumenty nebo části dokumentů, které odpovídají dotazu.
Výjimky XPath dotaz není validní
Výjimka: XPath dotaz není validní
Identifikátor: UC3.E.1
1. Případ použití začíná po kroku 2 hlavního toku. 2. Systém vypíše na chybový výstup informaci o chybě. 3. Systém vyvolá příslušnou výjimku.
Následné podmínky Systém vyvolal příslušnou výjimku spojenou s chybou v XPath dotazu.

Tabulka 6.3: Případ použití: XPath dotaz na kolekci dokumentů

6.3 Implementace API

Tato sekce stručně pojednává o implementaci navrženého systému. Vedle popisovaného databázového middleware byla ještě implementována jednoduchá klientská aplikace pro demonstraci funkčnosti.

Databázová vrstva xDB byla kompletně implementována v Javě. Jako aplikační rozhraní bylo zvoleno XML:DB API, které začíná být standardem na poli XML databází. Pro práci s XML byly využity implementace W3C DOM a dom4j. Pro parsování XPath bylo využito SAXPath parseru, který je součástí javovské knihovny Jaxen. Jelikož systém pracuje s XML schématy, bylo pro jejich generování a vzájemný převod použito knihovny Trang.

Využívá následující knihovny:

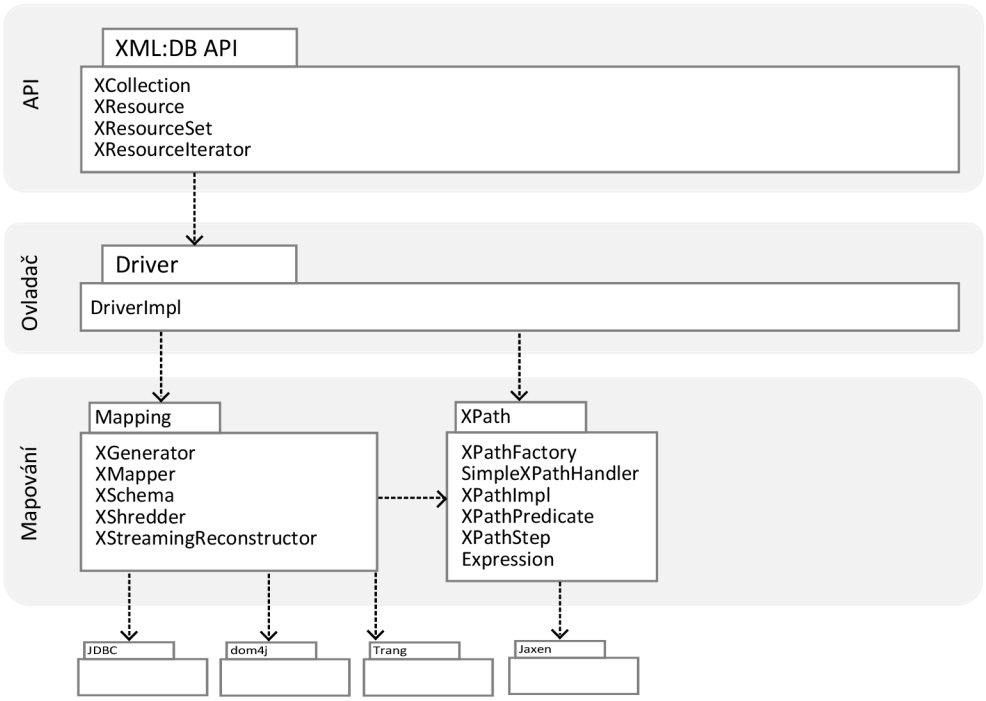
- SDK XML:DB API – SDK usnadňující vývoj aplikací implementujících aplikační rozhraní XML:DB
- dom4j – open source knihovna pro práci s XML, XPath a XSLT. Obsahuje XML parser Aelfred, který byl využit pro parsování XML dokumentů a schémat [6].

- Jaxen – open source knihovna pro práci s XPath. Odtud byl použit parser pro parsování XPath [7].
- Trang – knihovna umožňující převod různých druhů XML schémat mezi sebou. Podporuje XSD, RNG a DTD. Umožňuje také vygenerovat XML schéma z dodaného dokumentu nebo i z více dokumentů [11].

Z pohledu dotazování XPath jsou v aktuální verzi xDB podporovány pouze jednoduché predikátové výrazy (=, ! =, <, >, <=, >=) nad elementy a atributy, jednoduché indexy do seznamu elementů (např. day[1]) a konstrukce //, * a @*. Příklady podporovaných XPath výrazů jsou v tabulce 7.1. Kompletní reference XPath je v [2].

6.3.1 Vrstvy systému

Systém sestává ze tří vrstev, jak je možno vidět na obrázku 6.2. Nejvyšší vrstvou je vrstva aplikačního rozhraní. Ta obsahuje implementaci rozhraní XML:DB API. Aplikační vrstva využívá služeb ovladače, který tvoří další samostatnou vrstvu. Poslední vrstvou je mapovací vrstva, která zajišťuje fyzické mapování XML schémat a XML dokumentů do databáze a také dotazování a rekonstrukci XML dokumentů.



Obrázek 6.2: Zjednodušený diagram balíčků systému s naznačenými vrstvami

6.3.2 Aplikační rozhraní – XML:DB API

Jako aplikační rozhraní systému bylo zvoleno XML:DB API. XML:DB API (nebo také zkráceně XAPI) se snaží být standardem na poli XML databází [5]. API představuje formu implementačně nezávislého přístupu k XML datům. XAPI pro XML databáze se podobá ODBC a JDBC, které se používají u relačních databází.

API je aktuálně implementováno např. databázemi eXist, Apache Xindice, Tamino, BaseX, Gemfire Enterprise, DOMSafeXML, MonetDB/XQuery, myXMLDB, OZONE, Sedna a pravděpodobně mnoha dalšími.

Stručný tutoriál pro použití XML:DB API je možné najít v příloze C.

API vrstva implementuje rozhraní XML:DB s pomocí SDK knihovny. SDK XML:DB API již obsahuje některé základní implementace rozhraní, které bylo možno využít a rozšířit. Fyzicky tvoří tuto vrstvu balík `org.xdb.api`.

6.3.3 Vrstva ovladače

Tuto vrstvu tvoří balík `org.xdb.driver` a obsahuje pouze veřejné rozhraní Driver a veřejnou třídu DriverImpl, krerá toto rozhraní implementuje. Tím je dosaženo modularity systému – v budoucnu bude možné vytvořit různé implementace ovladače i pro jiné relační databáze (jejichž možnosti se mohou lišit), aniž by bylo nutné měnit klientské programy.

Aktuální implementace ovladače zajišťuje připojení k databázi PostgreSQL a poskytuje rozhraní pro správu kolekcí a dokumentů.

6.3.4 Mapovací vrstva

Mapovací vrstva sestává z fyzických balíků `org.xdb.map` a `org.xdb.xpath`. Zajišťuje fyzické mapování XML na databázové schéma. Poskytuje třídy pro vygenerování XML schématu z XML dokumentu (třída XGenerator skrze knihovnu Trang), mapování XML schématu na databázové schéma (XMapper a XSchema) a rekonstrukci uložených XML dokumentů (rozhraní Reconstructor). Třída XStreamingReconstructor implementuje rozhraní Reconstructor a kromě rekonstrukce dokumentů navíc umožňuje i dotazování pomocí XPath. Nalezené výsledky jsou třídou vraceny postupně, aby byla zajištěna rychlá odezva systému i při velkém počtu výsledků.

6.3.5 Uložení metainformací v databázi

V databázi je krom vlastních dokumentů nutno uložit i metainformace o jednotlivých kolekcích dokumentů, schématech a dokumentech. K tomu slouží tři tabulky – `xcollection`, `xdocument` a `xtable`. SQL skripty pro definici tabulek se nachází v příloze B. Tabulky jsou znázorněny i v ukázce schématu na obrázku 5.2.

xcollection

Tabulka `xcollection` obsahuje informace o všech kolekcích dokumentů v databázi. Každá kolekce je v systému identifikována unikátním jménem. Databáze PostgreSQL podporuje vytváření schémat (`CREATE SCHEMA`), která v podstatě vytvářejí jmenné prostory v databázi. Této možnosti bylo v systému využito a pro každou kolekci dokumentů systém v databázi vytváří vlastní SCHEMA. Touto jednoduchou cestou jmenných prostorů je dosaženo jedinečnosti jmen tabulek v rámci celé databáze (jména tabulek v rámci jedné kolekce dokumentů jsou systémem xDB generována jako unikátní).

U jednotlivých kolekcí dokumentů je uchováváno URI definující XML schéma dokumentů kolekce, dále pak v textové podobě dokument XML schématu. Tabulka také obsahuje jméno kořenové tabulky XML schématu.

xdocument

Tabulka `xdocument` obsahuje metainformace o všech dokumentech v databázi. Každý dokument je v rámci databáze identifikován unikátním automaticky generovaným identifikátorem. Dále je u dokumentu uchováváno jeho jméno, cizí klíč do tabulky kolekcí a obsah XML dokumentu v textové podobě (max. do velikosti 2GB). Obsah je vracen při požadavku na konkrétní dokument. Je tím dosaženo rychlejší odezvy, než kdyby musel být dokument rekonstruován. Nehledě na to, že takto mohou být uloženy i dokumenty, jejichž schéma není možné systémem namapovat na databázové schéma. To vše za cenu zvýšených nároků na paměť. V budoucích verzích systému by se měla objevit možnost tuto vlastnost vypnout.

xtable

Tabulka `xtable` uchovává informace o tabulkách mapujících schémata jednotlivých kolekcí. Obsahuje proto cizí klíč do tabulky kolekcí. Každá tabulka mapovacího schématu nese informace o elementu, který mapuje, a také o všech uzlech, které jsou v ní obsaženy. Informace o obsažených uzlech (elementech a atributech) jsou pro jednoduchost uloženy v polích (`TEXT[]`). Tyto informace je však možné získat i ze systémového katalogu databáze.

Tabulky mapovacího schématu vytvářejí hierarchickou strukturu, která v podstatě kopíruje strukturu mapovaného XML schématu. Proto je u každé tabulky cizí klíč na rodičovskou tabulku. Kořenová tabulka schématu má u tohoto klíče hodnotu NULL.

6.4 Implementace konzolové aplikace

Pro účely jednoduché administrace byla vytvořena také jednoduchá konzolová aplikace *xdbadmin*. Tato aplikace umožňuje připojit se ke zvolené databázi a kolekci dokumentů, vložit do kolekce soubor nebo adresář souborů. Xdbadmin umožňuje spustit dotaz v jazyce XPath, přičemž vrácené výsledky vypisuje na obrazovku. Aplikace xdbadmin je přiložena na průvodním CD spolu s README souborem, který obsahuje stručný návod na ovládání programu.

6.5 Shrnutí

Implementovaný databázový konektor xDB umožňuje ukládat a dotazovat XML dokumenty v relační databázi. Do konektoru byla implementována podpora XML:DB API, správa kolekcí, ukládání XML dokumentů do kolekcí a dotazování kolekcí dokumentů pomocí XPath. V rámci optimalizace pak bylo přidáno postupné doručování výsledků dotazu XPath klientské aplikaci.

Možná vylepšení

Databázový konektor by mohl v dalších verzích podporovat hierarchické zanořování kolekcí dokumentů a uložení více typů XML dokumentů do jedné kolekce. Potřebné je také vylepšení podpory složitějších výrazů v XPath a podpora jmenných prostorů. Dalším užitečným vylepšením by byla implementace XUpdate pro aktualizaci obsahu databáze.

Kapitola 7

Výkonové testy a srovnání

Následující kapitola se zabývá měřením a srovnáním výkonu vytvořené databáze xDB s referenční databází eXist.

7.1 Konfigurace databází

7.1.1 Instalace a konfigurace databáze eXist

Instalace eXist proběhla zcela automaticky spuštěním instalačního JAR balíčku, staženého ze [13]. Použita byla aktuální verze eXist 1.2.5. Při instalaci bylo nutné zadat pouze umístění databáze a heslo administrátora. Spuštění i vypnutí databáze je pak zajišťováno předpřipravenými skripty. Standardně běží eXist uvnitř webového serveru Jetty a je dostupný na adrese http://localhost:8080/exist/.

7.1.2 Konfigurace databáze xDB

Pro databázi xDB byla nainstalována poslední verze objektově relační databáze PostgreSQL 8.3.7. Spuštění i vypnutí databáze je taktéž umožněno předpřipravenými skripty. Pro nastavení bylo využito přiloženého grafického nástroje pgAdmin. Především bylo nutné vytvořit a nastavit databázi „xdb“, do které se budou ukládat testovací dokumenty. Také bylo potřeba vytvořit v nově vytvořené databázi tabulky pro uchování metadat systému xDB.

7.2 Uložení dat do databáze

Pro uložení dat do databáze xDB byla využita konzolová aplikace xdbadmin 6.4. Pro testovací účely byla vytvořena nová kolekce a do ní vloženy všechny testovací XML dokumenty.

Pro vložení dokumentů do databáze eXist bylo využito přiloženého administračního nástroje. I zde byla vytvořena nová kolekce pro uchování všech testovacích dokumentů.

Jako testovací data byla využita reálná data ze služby pro předpověď počasí [14]. Do obou databází bylo vloženo celkem 1000 testovacích XML dokumentů. Celková velikost ukládaných dokumentů přesahovala 6MB.

7.3 Testovací XPath dotazy

Tabulka 7.1 obsahuje výpis použitých XPath dotazů spolu s jejich stručným popisem a počtem relevantních výsledků. Dotazy byly voleny tak, aby co nejlépe prověřily schopnosti databáze xDB.

ID	XPath	Popis
Q1	/weather/head/locale	absolutní cesta
Q2	/weather/dayf/day[1]/part/wind	absolutní cesta s indexem elementu
Q3	/weather/dayf/day[1]/part/wind/*	absolutní cesta s indexem elementu a výběrem všech podřízených elementů
Q4	//wind	výběr elementů wind bez ohledu na umístění v dokumentu
Q5	//cc/wind/*	výběr elementů cc kdekoli v dokumentu spolu s výběrem všech podřízených elementů
Q6	//day[2][hi>75]/part/wind/*	současné použití indexu a predikátu
Q7	//cc[obst="Brno, CZECH REPUBLIC"]/wind	predikát s testem na rovnost hodnoty elementu
Q8	//day[@t="Saturday"]/part[1]	predikát s testem atributu a index podřízeného elementu
Q9	//day[@t="Saturday"]/part[@p="n"]	predikáty ve více krocích
Q10	//part/wind	výběr všech elementů wind, které jsou přímými potomky elementu part
Q11	//part/wind/*	výběr všech přímých potomků elementu wind, které jsou přímými potomky elementu part

Tabulka 7.1: Testovací XPath dotazy

7.4 Způsob testování

Pro testovací účely byla vytvořena konzolová aplikace xdbadmin 6.4. Zpracováním výsledků je myšleno iterování přes všechny nalezené výsledky a přistoupení k obsahu každého výsledku. Přístup k obsahu je demonstrován funkcí `getContentAsDOM()`, která vrací obsah výsledku jako DOM strom.

Zde bylo s výhodou využito toho, že obě databáze (xDB i eXist) implementují jednotné aplikační rozhraní XML:DB API. Bylo tedy možné obě testovat stejným programem, čímž by měly být výsledky měření dobře porovnatelné. Ukázka části kódu, která spustí dotaz nad kolekcí a pak iteruje přes všechny výsledky, je na výpisu 7.1.

Pro každý dotaz byly měřeny dva výsledky:

1. čas dotazu - představuje dobu trvání vykonání dotazu.
2. celkový čas dotazu - představuje dobu trvání dotazu plus dobu trvání zpracování všech výsledků.

Všechny testy byly pro větší objektivitu provedeny třikrát za sebou a do výsledných tabulek byl zaznamenán vždy aritmetický průměr naměřených časů. Pro zajištění objektivity byly na testovacím stroji po dobu testů spuštěny jen nejnutnější programy a s testovacím strojem nebylo pracováno. Testovacím strojem byl notebook s procesorem Intel Celeron M 1.6 MHz, 2GB operační paměti a ATA diskem s 5400 otáčkami za minutu.

Zdrojový kód 7.1: Vykonání XPath dotazu a získání výsledků

```
XPathQueryService service = col.getService("XPathQueryService", "1.0");
// zacneme merit casy
totalTime.start();
queryTime.start();
// dotaz
 ResourceSet result = service.query(xpath); // zde se meri cas dotazu
queryTime.stop();
// iterovani pres vsechny vysledky
ResourceIterator i = result.getIterator();
while (i.hasMoreResources()) {
   XMLResource r = (XMLResource) i.nextResource();
   // jednoduchy pokus o~pristup k~obsahu
   r.getContentAsDOM();
}
totalTime.stop();
```

7.5 Naměřené výsledky

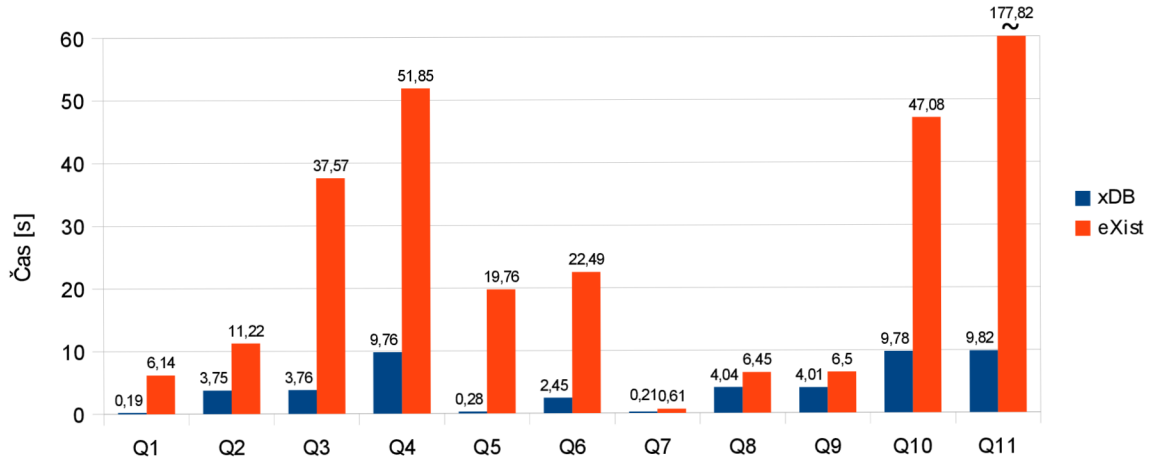
dotaz	poč. výsledků	xDB		eXist	
		čas dotazu	čas celkem	čas dotazu	čas celkem
Q1	1000	120	192	146	6140
Q2	2000	140	3750	203	11219
Q3	8000	140	3755	593	37567
Q4	11000	145	9760	547	51848
Q5	4000	120	275	307	19760
Q6	5144	140	2453	390	22494
Q7	56	125	208	78	609
Q8	1000	140	4041	281	6453
Q9	1000	125	4010	344	6500
Q10	10000	141	9776	532	47078
Q11	40000	141	9823	2156	177817

Tabulka 7.2: Naměřené doby trvání dotazů v milisekundách

Z tabulky naměřených časů 7.2 je zřejmé, že databáze xDB byla u všech testovaných dotazů výrazně rychlejší. Snad nejvýraznější rozdíl je u dotazu Q11, kdy vykonání dotazu trvalo xDB necelých 10 sekund, kdežto eXist stejný dotaz vyhodnocoval bezmála 177 sekund.

Za povšimnutí stojí také doba vykonání dotazu. U xDB je tato doba u všech dotazů téměř identická, rozdíl v desítkách milisekund lze považovat za chybu měření. To ukazuje, že odezva databáze xDB je na jakýkoli testovací dotaz velmi svižná. Naproti tomu u dob vykonání dotazů databáze eXist jsou značné rozdíly. Opět se vymyká především dotaz Q11.

Přehledněji jsou naměřené výsledky zobrazeny v grafu 7.1.



Obrázek 7.1: Graf celkových časů zpracování dotazu a výsledků

Kapitola 8

Závěr

Tato diplomová práce se v první části věnuje jazyku XML a s ním souvisejícím technologiím pro transformaci (XSL), definici struktury (DTD, W3C XML Schema a Relax NG) a dotazování (XPath, XQuery a SQL/XML).

Další kapitola je pak věnována ukládání XML dokumentů do databází. Zamýšlí se také nad tím, jakým způsobem lze ukládání XML dokumentů realizovat. Rozlišuje pak XML dokumenty na datově orientované a textově orientované a popisuje dva typy úložišť s podporou XML: nativní XML databáze a databáze s podporou XML.

V dalších kapitolách jsou diskutovány techniky mapování schématu XML dokumentů na schéma relační nebo objektově relační databáze. Tyto techniky jsou založeny buď na použití existujícího schématu XML dokumentu, nebo používají obecné databázové schéma, do kterého lze uložit jakýkoliv XML dokument. Jsou zde identifikovány možné problémy uložení XML dokumentu do databáze při použití některých těchto metod a navržena možná řešení.

V rámci práce byl navržen a implementován databázový konektor xDB, který nad relační databází (PostgreSQL) umožňuje ukládání a dotazování XML dokumentů. Dokumenty je možné ukládat do kolekcí dokumentů. Konektor také umožňuje dotazování kolekcí jazykem XPath.

Nad hotovým systémem byly provedeny výkonové testy a výsledky byly porovnány s existující XML databází eXist. Databáze xDB ze srovnání nevychází vůbec špatně, naopak je ve všech ohledech rychlejší než eXist.

V rámci FIT je plánováno využití xDB v nástrojích pro interaktivní vizualizaci XML dokumentů. Budoucím klientem bude pravděpodobně nástroj XMLSpark [28], který xDB využije pro ukládání a dotazování vizualizovaných dokumentů.

V dalších verzích vytvořeného systému by bylo možné implementovat podporu pro hierarchické zanořování kolekcí dokumentů a uložení více typů XML dokumentů do jedné kolekce dokumentů. Užitečné by bylo také vylepšení a rozšíření podpory složitějších konstrukcí a výrazů jazyka XPath a přidání podpory jmenných prostorů. Za zmínku také stojí možnost zabudování podpory XUpdate pro update uložených dokumentů. Ačkoli systém prošel několika optimalizacemi, je zde stále prostor pro vylepšování a zrychlování.

Literatura

[1] Extensible Markup Language (XML). 1996-2009, [Online; navštíveno 03.01.2009].
URL http://www.w3.org/XML/

[2] XML Path Language (XPath). 1999, [Online; navštíveno 05.01.2009].
URL http://www.w3.org/TR/xpath

[3] Introduction to XML Schema. 1999-2008, [Online; navštíveno 05.01.2009].
URL http://www.w3schools.com/schema/schema_intro.asp

[4] XPath Tutorial. 1999-2009, [Online; navštíveno 19.05.2009].
URL http://www.w3schools.com/XPath/

[5] XML:DB. 2000-2003, [Online; navštíveno 15.05.2009].
URL http://xmldb-org.sourceforge.net/

[6] dom4j. 2001-2008, [Online; navštíveno 20.05.2009].
URL http://www.dom4j.org/

[7] jaxen: universal Java XPath engine. 2001-2008, [Online; navštíveno 20.05.2009].
URL http://jaxen.codehaus.org/

[8] XML Database API Draft Proposal. 2003, [Online; navštíveno 20.05.2009].
URL http://xmldb-org.sourceforge.net/xapi/UseCases.html

[9] XML Standards XQuery vs SQL/XML. 2005-2009, [Online; navštíveno 6.01.2009].
URL http://www.xquery.com/white_papers/xquery_vs_sql/

[10] Extensible Stylesheet Language (XSL) Version 1.1. 2006, [Online; navštíveno 04.01.2009].
URL http://www.w3.org/TR/xsl/

[11] Trang. 2008, [Online; navštíveno 20.05.2009].
URL http://www.thaiopensource.com/relaxng/trang.html

[12] XQuery Update Facility 1.0. 2008, [Online; navštíveno 6.01.2009].
URL http://www.w3.org/TR/xqupdate/

[13] eXist: Open Source Native XML Database. 2009, [Online; navštíveno 23.05.2009].
URL http://exist.sourceforge.net/

[14] Meteorologists At The Weather Channel. 2009.
URL http://www.weather.com

[15] B.Chaudhri, A.; Rashid, A.; Zicari, R.: *XML Data Management: Native XML and XML-Enabled Database Systems*. Addison-Wesley, 2003, ISBN 0-201-84452-4, 688 s.

[16] Bohannon, P.; Freire, J.; Haritsa, J. R.; aj.: LegoDB: Customizing Relational Storage for XML Documents. 1999.
URL http://www.sigmod.org/vldb/conf/1999/P31.pdf

[17] Bourret, R.: XML and Databases. 1999-2005, [Online; navštíveno 05.01.2009].
URL http://www.rpbourret.com/xml/XMLAndDatabases.htm

[18] Bourret, R.: XML Database Products: Native XML Databases. 2000-2009, [Online; navštíveno 24.05.2009].
URL http://www.rpbourret.com/xml/ProdsNative.htm

[19] Bradley, N.: *XML: kompletní průvodce*. Praha: Grada Publishing s.r.o., 2000, ISBN 80-7169-949-7, 537 s.

[20] Chmelař, P.; Hernych, R.; Kubíček, D.: Interactive Visualization of Data-Oriented XML Documents. 2007.

[21] Florescu, D.; Kossmann, D.: Storing and Querying XML Data using an RDMBS. 1999.
URL http://www.soe.ucsc.edu/classes/cmps290s/Spring03/fk.pdf

[22] Harold, E. R.: *XML 1.1 Bible*. Indianapolis, Indiana: Wiley Publishing, Inc., třetí vydání, 2004, ISBN 0-7645-4986-3.

[23] Katz, H.; Chamberlin, D.; Draper, D.; aj.: *XQuery from the Experts: A Guide to the W3C XML Query Language*. Addison-Wesley, 2004, ISBN 0-321-18060-7.

[24] Kosek, J.: *XML pro každého: podrobný průvodce*. Praha: Grada Publishing s.r.o., první vydání, 2000, ISBN 80-7169-860-1, 164 s.

[25] Kosek, J.: XQuery. 2005, [Online; navštíveno 6.01.2009].
URL http://www.kosek.cz/xml/2005devcon/

[26] Mlynkova, I.; Pokorny, J.: XML in The World Of (Object-)Relational Database Systems. 2005.
URL http://www.ksi.mff.cuni.cz/~mlynkova/doc/tr2003-8.pdf

[27] Mlýnková, I.: *XML Schema a jeho implementace v prostředí relační databáze*. Diplomová práce, Karlova univerzita, 2003.

[28] Seko, M.: *Interaktivní vizualizace XML*. Bakalářská práce, Brno, FIT VUT v Brně, 2009.

[29] Shanmugasundaram, J.; Tufte, K.; He, G.; aj.: Relational Databases for Querying XML Documents: Limitations and Opportunities. 1999.
URL http://www.sigmod.org/vldb/conf/1999/P31.pdf

[30] Staken, K.: Introduction to Native XML Databases. 2001, [Online; navštíveno 6.01.2009].
URL http://www.xml.com/pub/a/2001/10/31/nativexmldb.html

[31] Steegmans, B.; Bourret, R.; Cline, O.; aj.: *XML for DB2 Information Integration.* IBM Redbooks, první vydání, 2004, ISBN 0738490032, 696 s.
URL http://www.redbooks.ibm.com/redbooks/pdfs/sg246994.pdf

[32] van der Vlist, E.: *RELAX NG.* O'Reilly, 2003, ISBN 0-596-00421-4.

Seznam použitých zkratek a symbolů

API – Application Programming Interface

BLOB – Binary Large OBject

CDATA – (Unparsed) Character Data

CLOB – Character Large OBject

CSS – Cascading Style Sheets

DOM – Document Object Model

DTD – Document Type Definition

FLWOR – Klauzule For, Let, Where, Order By, and Return v XQuery

HTML – HyperText Markup Language

JDBC – Java Database Connectivity

ODBC – Open Database Connectivity

ODF – Open Document Format

OOXML – Office Open XML

PCDATA – Parsed Character Data

RNG – Relax NG

SQL – Structured Query Language

SGML – Standard Generalised Markup Language

W3C – World Wide Web Consortium

XAPI – viz XML:DB API

XML – eXtensible Markup Language

XML:DB API – XML Database API

XPath – XML Path Language

XQuery – XML Query Language

XSD – XML Schema Definition

XSL – eXtensible Stylesheet Language

XSLT – eXtensible Stylesheet Language Transformations

XSLFO – eXtensible Stylesheet Language - Formatting Objects

Seznam příloh

Příloha A

CD

Přiložené CD obsahuje elektronickou verzi tohoto textu v `dp.pdf` a také zdrojové kódy práce v Latexu (adresář `/dp`). V adresáři `/db` se nachází SQL skript pro vytvoření a nastavení databáze. V adresáři `/xdb` se nachází veškerá data databázového front-entu xDB. Nachází se zde zdrojové kódy (`/xdb/src`), programová dokumentace (`/xdb/doc`) a také již zkompilovaná verze systému (`/xdb/dist`).

Adresář `/xdbadmin` obsahuje zdrojové kódy (`/xdbadmin/src`) a spustitelnou verzi (v `/xdbadmin/dist`) jednoduché konzolové administrační aplikace. Je zde také soubor `README`, který obsahuje popis ovládání konzole xdbadmin. Adresář `/xdbadmin/input` obsahuje vzorek XML dokumentů.

Příloha B

Struktura databáze xDB

Zdrojový kód B.1: SQL kód pro vytvoření základních tabulek xDB

```
DROP TABLE xcollection CASCADE;
DROP TABLE xtable CASCADE;
DROP TABLE xdocument CASCADE;

CREATE TABLE xcollection (
   id SERIAL,
   name VARCHAR(30),
   uri VARCHAR(30),
   xml_schema TEXT,
   root_element_table VARCHAR(30),
   PRIMARY KEY(id),
   UNIQUE(name)
);

INSERT INTO xcollection VALUES(DEFAULT, 'root', NULL);

CREATE TABLE xtable (
   id SERIAL,
   collection INTEGER NOT NULL REFERENCES xcollection ON UPDATE CASCADE ON DELETE CASCADE,
   parent_table INTEGER REFERENCES xtable ON UPDATE CASCADE ON DELETE CASCADE,
   inlined BOOLEAN,
   element_name VARCHAR(50),
   table_name VARCHAR(50),
   elements TEXT[],
   attributes TEXT[],
   PRIMARY KEY(id),
   UNIQUE (collection, table_name)
);

CREATE TABLE xdocument (
   id SERIAL,
   collection INTEGER NOT NULL REFERENCES xcollection ON UPDATE CASCADE ON DELETE CASCADE,
   name VARCHAR(30),
   content TEXT,
   PRIMARY KEY (id),
   UNIQUE (name, collection)
);

ALTER TABLE xcollection OWNER TO xdb;
ALTER TABLE xtable OWNER TO xdb;
ALTER TABLE xdocument OWNER TO xdb;
```

Příloha C

Tutoriál XML:DB API

Tato příručka se snaží o uvedení příkladů nejběžnějších případů použití databáze xDB, jako například připojení ke kolekci, vložení dokumentu do kolekce a dotazování kolekce. Podrobnější informace lze nalézt na webu [8]. Jednotlivé části zdrojových kódů nejsou úplné, nezabývají se chybami, které se mohou vyskytnout.

Připojení k xdb databázi

Scénář ukazuje jednoduchý přístup k databázi xDB. V databázi xdb existuje kolekci weather, která obsahuje XML dokumenty o předpovedi počasí. Po připojení získá klient referenci na objekt kolekce, jehož prostřednictvím pak dále komunikuje s databází.

Zdrojový kód C.1: Kód pro připojení klienta k xDB databázi

```
// vyber ovladace
String driver = "org.xdb.driver.DriverImpl";
Class c = Class.forName(driver);
// uri databaze
String URI = "xmldb:xdb://localhost/xdb/";

Database database = (Database) c.newInstance();
DatabaseManager.registerDatabase(database);
// ziskame referenci na vybranou kolekci
collection = DatabaseManager.getCollection(URI + "weather");
```

Vytvoření kolekce

Pro vytvoření nové kolekce je nutné se připojit ke kořenové kolekci databáze (v xDB je tohle chování simulováno) a vytvořením podřízené kolekce. Správu kolekcí zajišťuje služba CollectionManagementService.

Zdrojový kód C.2: Kód pro vytvoření nové kolekce

```
Collection root = DatabaseManager.getCollection(URI);
CollectionManagementService mgtService = (CollectionManagementService)
                    root.getService("CollectionManagementService", "1.0");
collection = mgtService.createCollection("novaKolekce"));
```

Vložení XML dokumentu do kolekce

Scénář popisuje uložení XML dokumentu do kolekce. Předpokládá, že klient je již ke kolekci připojen.

Zdrojový kód C.3: Kód pro uložení XML dokumentu do kolekce

```
// predpokladejme, ze document obsahuje validni XML dokument
String document;
// nechame pridelit id
String id = null;
XMLResource resource = (XMLResource) collection.createResource(id, XMLResource.RESOURCE_TYPE);
resource.setContent(document);
collection.storeResource(resource);
```

Získání XML dokumentu

Scénář popisuje získání dokumentu z kolekce pomocí známého id. Obsah dokumentu je získán jako DOM.

Zdrojový kód C.4: Kód pro získání XML dokumentu

```
String id = "10";
XMLResource resource = (XMLResource) collection.getResource(id);
// chceme obsah jako DOM
Document doc = (Document) resource.getContentAsDOM();
```

XPath dotaz na kolekci dokumentů

Dotaz na kolekci XML dokumentů. S výsledky se pak pracuje pomocí DOM. Dotazování je v XML:DB API realizováno pomocí služeb. Tím je zajištěna kompatibilita různých výrobců a také snadná rozšiřitelnost API. Pro dotazování XPath je určena služba XPathQueryService.

Zdrojový kód C.5: Kód pro XPath dotaz na kolekci dokumentů

```
String xpath = "//day[1][@p='May 19']";
// ziskame sluzbu
XPathQueryService service = (XPathQueryService) col.getService("XPathQueryService", "1.0");
ResourceSet resultSet = service.query(xpath);
// získání výsledků
ResourceIterator results = resultSet.getIterator();
while (results.hasMoreResources()) {
   Resource res = results.nextResource();
   System.out.println((String) res.getContent());
}
```

Příloha D

Ukázka mapování XML dokumentu

Ukázkový XML dokument weather.xml

Zdrojový kód D.1: Zkrácený výpis dokumentu weather.xml [14]

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- This document is intended only for use by authorized licensees of The  -->
<!-- Weather Channel. Unauthorized use is prohibited.  Copyright 1995-2009, -->
<!-- The Weather Channel Interactive, Inc.  All Rights Reserved.            -->
<weather ver="2.0">
  <head>
    <locale>en_US</locale>
    <form>MEDIUM</form>
    ...
  </head>
  <loc id="EZXX0002">
    <dnam>Brno, Czech Republic</dnam>
    <sunr>5:11 AM</sunr>
    <suns>8:30 PM</suns>
    ...
  </loc>
  <cc>
    <lsup>5/15/09 2:00 PM Local Time</lsup>
    <obst>Brno, CZECH REPUBLIC</obst>
    <t>Cloudy</t>
    <bar>...</bar>
    <wind>...</wind>
    <moon>...</moon>
    ...
  </cc>
  <dayf>
    <lsup>5/15/09 7:20 AM Local Time</lsup>
    <day d="0" t="Friday" dt="May 15">
      <hi>71</hi>
      <low>55</low>
      <sunr>5:11 AM</sunr>
      <suns>8:30 PM</suns>
      <part p="d">
        <icon>28</icon>
        <t>Mostly Cloudy</t>
        <wind>...</wind>
```

```
        ...
      </part>
      <part p="n">
        <icon>29</icon>
        <t>Partly Cloudy</t>
        <wind> ... </wind>
        ...
      </part>
    </day>
    <day d="1" t="Saturday" dt="May 16">...</day>
    <day d="2" t="Saturday" dt="May 17">...</day>
    ...
  </dayf>
</weather>
```

Vygenerované databázové schéma

Ve výpisu je uvedeno vygenerované databázové schéma pro XML dokument D.1. Byly vytvořeny celkem tři tabulky, a to pro elementy weather, day a part.

Zdrojový kód D.2: Úplný výpis SQL pro vytvoření tabulek databázového schématu

```
CREATE TABLE weather.tab_weather_0
(
  __id serial NOT NULL,
  __document_id integer,
  __encoding character varying(15),
  "@ver" numeric(10,6),
  "head.locale" text,
  "head.form" text,
  "head.ut" text,
  "head.ud" text,
  "loc.dnam" text,
  "loc.tm" text,
  "loc.lat" numeric(10,6),
  "loc.lon" numeric(10,6),
  "loc.sunr" text,
  "loc.suns" text,
  "loc.zone" integer,
  "loc.@id" text,
  "lnks.@type" text,
  "cc.lsup" text,
  "cc.obst" text,
  "cc.tmp" integer,
  "cc.flik" integer,
  "cc.t" text,
  "cc.icon" integer,
  "cc.hmid" text,
  "cc.vis" text,
  "cc.dewp" text,
  "cc.bar.r" text,
  "cc.bar.d" text,
  "cc.wind.s" text,
  "cc.wind.gust" text,
  "cc.wind.d" text,
  "cc.wind.t" text,
  "cc.uv.i" text,
  "cc.uv.t" text,
  "cc.moon.icon" integer,
```

```
  "cc.moon.t" text,
  "dayf.lsup" text,
  CONSTRAINT tab_weather_0_pkey PRIMARY KEY (__id),
  CONSTRAINT tab_weather_0___document_id_fkey FOREIGN KEY (__document_id)
      REFERENCES xdocument (id) MATCH SIMPLE
      ON UPDATE CASCADE ON DELETE CASCADE
)

ALTER TABLE weather.tab_weather_0 OWNER TO xdb;

-- Tabulka tab_day_11
CREATE TABLE weather.tab_day_11
(
  __id serial NOT NULL,
  __parent integer,
  hi text,
  low integer,
  sunr text,
  suns text,
  "@d" integer,
  "@dt" text,
  "@t" text,
  CONSTRAINT tab_day_11_pkey PRIMARY KEY (__id),
  CONSTRAINT tab_day_11___parent_fkey FOREIGN KEY (__parent)
      REFERENCES weather.tab_weather_0 (__id) MATCH SIMPLE
      ON UPDATE CASCADE ON DELETE CASCADE
)

ALTER TABLE weather.tab_day_11 OWNER TO xdb;

-- Index: weather.day_11__parent_fk
CREATE INDEX day_11__parent_fk ON weather.tab_day_11 USING btree (__parent);

-- Tabulka tab_part_12
CREATE TABLE weather.tab_part_12
(
  __id serial NOT NULL,
  __parent integer,
  icon integer,
  t text,
  bt text,
  ppcp integer,
  hmid text,
  "@p" text,
  "wind.s" text,
  "wind.gust" text,
  "wind.d" text,
  "wind.t" text,
  CONSTRAINT tab_part_12_pkey PRIMARY KEY (__id),
  CONSTRAINT tab_part_12___parent_fkey FOREIGN KEY (__parent)
      REFERENCES weather.tab_day_11 (__id) MATCH SIMPLE
      ON UPDATE CASCADE ON DELETE CASCADE
)

ALTER TABLE weather.tab_part_12 OWNER TO xdb;

-- Index: weather.part_12__parent_fk
CREATE INDEX part_12__parent_fk ON weather.tab_part_12 USING btree (__parent);
```